conference
proceedings

4th USENIX Symposium on Vehicle Security and Privacy (VehicleSec '26)

Baltimore, MD, USA
August 10–11, 2026

Sponsored by
USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

ISBN 978-1-939133-56-4

USENIX
Proceedings of the 4th USENIX Symposium on Vehicle Security and Privacy (VehicleSec '26) Baltimore, MD, USA August 10–11, 2026

VehicleSec '26 Sponsors

Diamond Sponsor

UC Irvine
Donald Bren
School of Information
& Computer Sciences

Bronze Sponsors

DENSO
Crafting the Core

INDIANA UNIVERSITY BLOOMINGTON
DEPARTMENT OF COMPUTER SCIENCE

ZOOX

Philanthropic Supporter

craig newmark philanthropies

USENIX Supporters

USENIX Patrons

Bloomberg • Google

Open Access Supporter

Google

Open Access Publishing Partner

PeerJ

USENIX Association

Proceedings of the 4th USENIX Symposium on Vehicle Security and Privacy (VehicleSec '26)

August 10–11, 2026
Baltimore, MD, USA

© 2026 by The USENIX Association

All Rights Reserved

This volume is published as a collective work. Rights to individual papers remain with the author or the author's employer. Permission is granted for the noncommercial reproduction of the complete work for educational or research purposes. Permission is granted to print, primarily for one person's exclusive use, a single copy of these Proceedings. USENIX acknowledges all trademarks herein.

ISBN 978-1-939133-56-4

Conference Organizers

General Chairs
Hyungsub Kim, *Indiana University*
Aiping Xiong, *The Pennsylvania State University*

Program Co-Chairs
Sara Rampazzi, *University of Florida*
Ning Zhang, *Washington University in St. Louis*

Program Committee
Daniele Antonioli, *EURECOM*
MD Armanuzzaman, *The University of Texas at El Paso*
Luca Arnaboldi, *University of Birmingham*
Antonio Bianchi, *Purdue University*
Gedare Bloom, *University of Colorado Colorado Springs*
Divanilson R. Campelo, *Universidade Federal de Pernambuco*
Yulong Cao, *NVIDIA*
Stephen Checkoway, *Oberlin College*
Hanlin Chen, *Oak Ridge National Laboratory*
Yimin Chen, *University of Massachusetts Lowell*
Wonsuk Choi, *Korea University*
Andrew Clark, *Washington University in St. Louis*
Mauro Conti, *University of Padova*
Jiarun Dai, *Fudan University*
Jeremy Daily, *Colorado State University*
Pritam Dash, *The University of British Columbia*
Marco De Vincenzi, *Institute of Informatics and Telematics (IIT), CNR*
Bruce DeBruhl, *California Polytechnic State University*
Denis Donadel, *University of Verona*
Alessandro Erba, *Karlsruhe Institute of Technology*
Yiheng Feng, *Purdue University*
Thomas Forest, *General Motors*
Clement Fung, *Carnegie Mellon University*
Xing Gao, *University of Delaware*
Ryan Gerdes, *Virginia Tech*
Mohammad Hamad, *Technical University of Munich*
Hans-Joachim Hof, *Technische Hochschule Ingolstadt*
Bardh Hoxha, *Toyota Research Institute of North America (TRINA)*
Hongxin Hu, *University at Buffalo*
Shengtuo Hu, *Palo Alto Networks*
Murtuza Jadliwala, *The University of Texas at San Antonio*
Shalabh Jain, *Center of Data for Public Good and University of Bangalore*
Xiaoyu Ji, *Zhejiang University*
Justin Kappos, *New York University*
Ameer Kashani, *DENSO International America, Inc.*
Chung Hwan Kim, *The University of Texas at Dallas*
Huy Kang Kim, *Korea University*
Seulbae Kim, *Pohang University of Science and Technology (POSTECH)*
Taegyu Kim, *The Pennsylvania State University*
Yongdae Kim, *Korea Advanced Institute of Science and Technology (KAIST)*
Sebastian Köhler, *The University of Manchester*
Vireshwar Kumar, *Indian Institute of Technology Delhi*
Xiaojing Liao, *University of Illinois Urbana–Champaign*
Zhiqiang Lin, *The Ohio State University*
Peng Liu, *The Pennsylvania State University*
Stefano Longari, *Politecnico di Milano*
Li Lu, *Zhejiang University*
Tatsuya Mori, *Waseda University*
Ivan De Oliveira Nunes, *University of Zurich*
Habeeb Olufowobi, *The University of Texas at Arlington*
Muslum Ozgur Ozmen, *Arizona State University*
Christos Papadopoulos, *The University of Memphis*
Jonathan Petit, *Qualcomm*
Hanif Rahbari, *Rochester Institute of Technology*
Indrakshi Ray, *Colorado State University*
Amir Salarpour, *Clemson University*
Takami Sato, *University of California, Irvine*
Neetesh Saxena, *Cardiff University*
Shanghao Shi, *Washington University in St. Louis*
Yasser Shoukry, *University of California, Irvine*
David Starobinski, *Boston University*
Takeshi Sugawara, *The University of Electro-Communications*
Xi Tan, *University of Colorado Colorado Springs*
Nils Ole Tippenhauer, *CISPA Helmholtz Center for Information Security*
Yazhou Tu, *Auburn University*
Hrushikesh Varma, *University of Florida*
Jinwen Wang, *The University of Texas at Dallas*
Lan Wang, *The University of Memphis*
Ningfei Wang, *Meta*
André Weimerskirch, *Block Harbor Cybersecurity*
Yang Xiao, *University of Kentucky*
Luyi Xing, *Indiana University*
Kun Yang, *Zhejiang University*
Min Yang, *Fudan University*
Kentaro Yoshioka, *Keio University*
Ce Zhou, *Missouri University of Science and Technology*
Xugui Zhou, *Louisiana State University*
Yi Zhu, *Wayne State University*
Shuguo Zhuo, *Zhejiang University*
Saman Zonouz, *Georgia Institute of Technology*

Tutorial Co-Chairs
Ming Li, *University of Arizona*
Karthik Pattabiraman, *University of British Columbia*

Poster/Demo Program Co-Chairs
Habiba Farrukh, *University of California, Irvine*
Jinwen Wang, *University of Texas Dallas*

Poster/Demo Program Committee
Jiahao Cao, *Tsinghua University*
Noah T. Curran, *University of Michigan*
Mengyu Liu, *Washington State University*
Zeyan Liu, *University of Louisville*
Pedram MohajerAnsari, *Clemson University*
Marion Sudvarg, *Washington University in St. Louis*
Chenyi Wang, *University of Arizona*
Guangjing Wang, *University of South Florida*
Xinda Wang, *The University of Texas at Dallas*
Xiaoli Zhang, *University of Science and Technology Beijing*
Zhuangzhai Zhu, *George Mason University*

Publications Chair
Xiali Hei, *University of Louisiana at Lafayette*

Lightning Talks Co-Chairs
Marcio Juliato, *Waymo*
Mulong Luo, *Florida State University*

Publicity Co-Chairs
Nidhi Rastogi, *Rochester Institute of Technology*
Zhiyuan Yu, *Texas A&M University*

Sponsorship Co-Chairs
Mert Pesé, *Clemson University*
Qiben Yan, *Michigan State University*

Artifact Evaluation Committee Co-Chairs
Taegyu Kim, *The Pennsylvania State University*
Dario Stabili, *University of Modena and Reggio Emilia*

Artifact Evaluation Committee
Zain Ul Abideen, *University of Idaho*
Hassan El Alami, *Howard University*
Rohan Dilip Bhosle, *Microsoft*
Zheyun Feng, *University of New Hampshire*
Dimitri Galli, *University of Modena and Reggio Emilia*
Kamel Ghali, *KAGE Corporation, Car Hacking Village*
Avirek Ghatia, *Salesforce*
Ishaan Gupta, *Google*
Fan Huang, *Indiana University Bloomington*
Joseph Khoury, *Louisiana State University*
Nahyun Lee, *The Pennsylvania State University*
Rao Li, *The Pennsylvania State University*
Gianmarco Lusvardi, *University of Modena and Reggio Emilia*
Xingyu Lyu, *University of Massachusetts Lowell*
Naga Nithin Manne, *Argonne National Laboratory*
Jainta Paul, *University of Utah*
Ravi Chandra Sadineni, *Google*
Partho Sarthi, *NVIDIA*
Filip Valgimigli, *University of Modena and Reggio Emilia*
Connor Weyers, *Indiana University Bloomington*
Doguhan Yeke, *Purdue University*
Syed Mohammad Abbas Zaidi, *NED University of Engineering and Technology*
Ruixuan Zhang, *New York University*
Giovanni Gambigliani Zoccoli, *University of Modena and Reggio Emilia*

Steering Committee
Gail-Joon Ahn, *Arizona State University*
David Balenson, *USC Information Sciences Institute*
Chunming Qiao, *University at Buffalo*
Kang Shin, *University of Michigan*
Mani Srivastava, *University of California, Los Angeles*
Gene Tsudik, *University of California, Irvine*
Dongyan Xu, *Purdue University*

Message from the VehicleSec ’26 Program Co-Chairs

On behalf of the VehicleSec ’26 Organizing and Steering Committees, we welcome you to the 4th USENIX Symposium on Vehicle Security and Privacy.

This year, continuing the success of VehicleSec ’25, VehicleSec ’26 is co-located with the USENIX Security Symposium. The event will span two days. It has been a distinct honor to serve as general and program co-chairs for the symposium, and a privilege to continue collaborating with the exceptional organizing committee and USENIX team.

VehicleSec aims to bring together an audience of university researchers, scientists, industry professionals, and government representatives to contribute new theories, technologies, and systems on security/privacy issues related to vehicles (e.g., ground, aerial, in/on water, space), their sub-systems (e.g., in-vehicle networks, autonomy, connectivity, human-machine interfaces), supporting infrastructures (e.g., transportation infrastructure, charging station, ground control station), and related fundamental technologies (e.g., sensing, control, AI/ML/DNN/LLM, wireless communication, real-time computing, edge computing, location service, simulation, digital twin, multi-agent protocol/system design, and human-machine interaction).

For the current iteration of the symposium, a total of 67 submissions were received, marking a new record for full research papers submitted. These submissions were categorized as 55 full papers, 6 Work-in-Progress (WIP), and 6 short papers. In addition, the symposium received 6 poster submissions, 12 demo submissions, and 1 tutorial submission.

There are 80 Program Committee members representing 67 distinct institutions, and the Poster/Demo Program Committee consists of 11 members from 11 different institutions. All submitted papers for the symposium underwent a rigorous review process, with each paper being evaluated by a minimum of two reviewers. In parallel, every poster and demo submission received two reviews.

Continuing with the success of the past year, we provide the authors of particularly relevant papers that necessitate minor revision to enter the shepherding process. A shepherd was assigned to ensure the changes were adequately addressed before the camera-ready deadline.

From 67 paper submissions, 18 full papers, 3 WIP papers, and 5 short papers were accepted. From 6 posters and 12 demo submissions, 4 posters and 12 demos were accepted. From these, we are pleased to announce that the symposium will feature Best Paper and Best Demo awards.

This year, the symposium continues its lightning talk session, providing a platform for interested attendees to present concise, 5-minute in-person talks on various topics relevant to the VehicleSec community. These topics include, but are not limited to, emerging hot topics, preliminary research findings, practical challenges, lessons learned, introductions to tutorials, and educational materials related to vehicles. This session serves as an opportunity for participants to broadcast their ongoing efforts or new ideas and seek feedback at a community level. After careful review, three lightning talks were accepted for their interesting and impactful content.

A new addition to the 2026 symposium is the introduction of the artifact evaluation process to emphasize the importance of research artifacts, the benefits to the authors and the community as a whole, and to promote the availability and reproducibility of experimental results. Authors of accepted papers were encouraged to submit their artifacts for availability, functionality, and reproducibility assessments. We established a new Artifact Evaluation Committee (AEC) that reviewed each submitted artifact against the following badges: “Available,” “Functional,” and “Reproduced.” The committee is composed of 25 members from 19 institutions and is supervised by two Artifact Evaluation Chairs.

Seventeen of the accepted papers submitted their artifacts. Among those, five received the “Available” badge, two received the “Available” and “Functional” badges, one received the “Available” and “Reproduced” badges, and nine papers received all three badges. From these, we announce that the symposium will grant the Distinguished Artifact Award.

Beyond the technical program, demo and poster session, and lightning talks, VehicleSec ’26 will feature two keynote presentations. The keynote speakers include André Weimerskirch, COO of Block Harbor Cybersecurity, who will present “The Trajectory of Automotive Cybersecurity and the Role of AI.” The second keynote is by Professor Stefan Savage from University of California, San Diego, titled “Research only? Understanding the Scope of Automotive Cyberattacks in the Wild.”

In addition, we introduced two new special sessions. The first is the Government and Industry Panel, where representatives from federal agencies and industry discuss the current advancement and funding landscape. The panel aims to bring researchers together to discuss upcoming funding priorities, industry perspectives, and proposal strategies. The second session consists of three invited talks selected from the ESCAR USA Conference. This session is part of VehicleSec’s effort to bring private industry professionals, researchers, and engineers from the automotive community to discuss emerging challenges in vehicle security and privacy.

This year, we are pleased to acknowledge the generous support of our sponsors: The School of Information and Computer Sciences at UC Irvine, DENSO, Indiana University, Zoox, and philanthropic supporter Craig Newmark Philanthropies.

The successful organization of VehicleSec '26 is the result of the collaborative efforts of numerous individuals and entities.

First, we extend our sincere gratitude to all authors for their valuable submissions to the symposium. Their research forms the core of this event and drives the advancement of the field.

Second, we express our appreciation to all the program committee members for their diligent efforts in reviewing submissions, providing constructive feedback to authors, and actively participating in online discussions. Their expertise and commitment are vital to maintaining the high quality of the symposium's technical program. Here, we also extend a special thanks to the VehicleSec Steering Committee, whose guidance was instrumental in the program improvement. Their dedication ensures the symposium's seamless execution.

Furthermore, our sincere thanks go to all the sponsors for their generous financial support, which has been crucial in enabling travel grants, the community reception, and the poster and demo organization.

Lastly, we are deeply grateful to the USENIX team, with whom VehicleSec is co-located this year alongside USENIX Security. Their guidance, invaluable help, and endless patience throughout all phases of the organization have been truly exceptional.

Hungsub Kim, *Indiana University Bloomington*
Aiping Xiong, *The Pennsylvania State University*
VehicleSec '26 General Co-Chairs

Sara Rampazzi, *University of Florida*
Ning Zhang, *Washington University in St. Louis*
VehicleSec '26 Program Co-Chairs

4th USENIX Symposium on Vehicle Security and Privacy (VehicleSec '26)
August 10–11, 2026
Baltimore, MD, USA

Monday, August 10

Drone Security

Blind Spot in Sensor Fusion: Exploiting an Architectural Vulnerability to Hijack and Precisely Control UAVs 1
Jongsoo Han and Seulbae Kim, *Pohang University of Science and Technology*

SHIELD: Secure Human-Machine Interaction with Evidential Learning and Dynamic Trust for Drone Swarm Control 19
Dawood Wasif, *Virginia Tech;* Terrence J. Moore, *US DEVCOM Army Research Laboratory;* Seunghyun Yoon and Hyuk Lim, *Korea Institute of Energy Technology (KENTECH);* Dan Dongseong Kim, *The University of Queensland;* Frederica F. Nelson, *US DEVCOM Army Research Laboratory;* Jin-Hee Cho, *Virginia Tech*

SLEIGHT: Real-time Closed-Loop Attack Analytics Framework for Holistic Intrusion Detection Systems in Unmanned Aerial Vehicles 39
Muneeba Asif, *Florida International University;* Maryna Veksler, *Virginia Commonwealth University;* Devrim Aras, *Florida International University;* Kemal Akkaya, *Virginia Commonwealth University;* Mohammad Ashiqur Rahman, *Florida International University*

Vehicle Attacks and Defenses 1

E-Trojans: Ransomware, Tracking, DoS, and Data Leaks on the Xiaomi E-Scooter Ecosystem 57
Marco Casagrande, *EURECOM;* Riccardo Cestaro, Mauro Conti, and Eleonora Losiouk, *University of Padua;* Daniele Antonioli, *EURECOM*

RVDebloater: Mode-based Adaptive Firmware Debloating for Robotic Vehicles 77
Mohsen Salehi and Karthik Pattabiraman, *The University of British Columbia*

Vehicle Network Security

CanBert+: An Optimized Multi-Layer BERT-Based Intrusion Detection for CAN Bus 97
Jun Yeon Won, Qadeer Ahmed, and Zhiqiang Lin, *The Ohio State University*

MIRAGE: Detecting Fake Emergency Electronic Brake Light Attacks in V2X Networks via Event-Gated Behavioral Analysis 117
Eunhan Ka, Doguhan Yeke, Z. Berkay Celik, and Satish Ukkusuri, *Purdue University*

WIP: On the Real-World Applicability of Automotive CAN Intrusion Detection Systems. 135
Kristijan Lazeski, *Darmstadt University of Applied Sciences and European University of Technology;* Michael Zohner, *University of Applied Sciences Fulda;* Jannis Hamborg, *Darmstadt University of Applied Sciences and European University of Technology;* Christoph Krauß and Alexander Wiesmaier, *Darmstadt University of Applied Sciences, European University of Technology, and National Research Center for Applied Cybersecurity ATHENE*

Short: CARdea: Two-Phase Plausibility-Based Vehicle-to-Vehicle Anomaly Detection System. 147
Mert D. Pesé, *Clemson University;* Arman Tabaddor, *The University of Michigan;* Teja Guruvelli, *Clemson University;* Roland Varriale, *Argonne National Laboratory;* Marco De Vincenzi, *IIT-CNR;* Kang G. Shin, *The University of Michigan*

WIP: SNFuzz: Gateway-Centric State-Oriented Fuzzing for MQTT-SN 157
Christian de la Pena, Samin Y. Chowdhury, Mohammad Rahman, Weidong Zhu, and Ruimin Sun, *Florida International University*

Infrastructure Security

Digital Eyes on the Road: Using Street Cameras to Verify Traffic Integrity and Detect Sybil Attacks at City Scale 167
Jhonatan Tavori, Mehmet Kerem Turkcan, Zoran Kostic, Salvatore Stolfo, and Gil Zussman, *Columbia University*

Charge It to My Neighbor: A Relay Attack on ISO 15118 Plug and Charge Payment 181
Jakob Löw, Vishwa Vasu, Thomas Hutzelmann, and Hans-Joachim Hof, *Technische Hochschule Ingolstadt*

CIAplates: Covert, Intelligent, Adversarial Attack on Automated License Plate Recognition Systems using Digital License Plates 191
Namo Asavisanu, Tina Khezresmaeilzadeh, Rohan Sequeira, and Konstantinos Psounis, *University of Southern California*

Short: Race to Charge: Concurrency Bugs and State Integrity Violations in EV Charging Infrastructure 209
Jaeyeong Lee, *Independent Researcher;* Seonhyeong Lee, *SK Shieldus*

Tuesday, August 11

Threat Analysis and Risk Assessment

A Study on Extending the TARA Methodology for Physical AI: A Case Study of ADS 219
Jihun Woo and Junhyeong Lee, *Korea Testing & Research Institute*

AutoHack: A Physically Verified Multi-Bus CAN Dataset for Intrusion Detection System Evaluation 237
Yuchan Song, *Korea University;* Sejun Ahn, *Kookmin University;* Seungjin Baek, *Korea University;* Hyeonseong Kim, *Kookmin University;* Huy Kang Kim, *Korea University;* Sanghoon Jeon, *Kookmin University*

Bridging Two TARAs: Integrating MITRE CTSA/CRRA Prioritization into ISO/SAE 21434 for Automotive Security 253
Minhyuk Park, Yejun Kim, and Seungjoo Kim, *School of Cybersecurity, Korea University*

WIP: A Benchmark for Security Evaluation of Adversarial Manoeuvre Attacks in Autonomous Driving 271
Adibeh Emamy, Marcello Maugeri, and PingChen Lin, *CybPass Limited;* Aryan Pasikhani, *The University of Sheffield*

Sensor Security

Perspective-Shift Attacks Against Optical Perception Sensors: A Novel Attack Vector on LiDAR and Camera . . . 283
Marco Calipari, Michael Kühr, Dominik Kulmer, Maximilian Luedecke, Mohammad Hamad, and Sebastian Steinhorst, *Technical University of Munich*

Seeing is Deceiving: Systematic Vulnerability Analysis of LiDAR-Based Autonomous Driving to Mirror-Induced Perception Failures 301
Selma Yahia and Ildi Alla, *Interdisciplinary Centre for Security, Reliability and Trust* (*SnT*)*, University of Luxembourg;* Girija B. Mohan and Daniel Rau, *University of Applied Science Saarland (htw saar);* Mridula Singh, *CISPA Helmholtz Center for Information Security;* Valeria Loscri, *Inria Lille - Nord Europe*

Short: Investigating Stealthy Latency Attacks Against Camera-Based Object Detection Systems 319
Lichen Xia, Xing Gao, and Weisong Shi, *University of Delaware*

Vehicle Privacy

Beyond Classical PKI: A Quantum-Safe Authentication Framework for Automotive OTA Updates 331
Wei-Lun Lin, Ting-Yu Chen, and Arijit Karati, *National Sun Yat-sen University*

Short: VIN2VICTIM: From Vehicle Identification to Targeted Tracking via Data Broker Exploitation 349
Hannaneh B. Pasandi, *UC Berkeley;* Mohammad Sepahi, *Rivian Inc. and Old Dominion University*

Short: Is Your ECU Really Yours? 359
Abbas Ali and Hafiz Malik, *University of Michigan-Dearborn*

Vehicle Attacks and Defenses 2

RAIL: Risk-Aware Human-in-the-Loop Framework with Adaptive Intrusion Response for Autonomous Vehicles . . . 371
Dawood Wasif, *Virginia Tech;* Terrence J. Moore, *US DEVCOM Army Research Laboratory;* Seunghyun Yoon and Hyuk Lim, *Korea Institute of Energy Technology* (*KENTECH*)*;* Dan Dongseong Kim, *The University of Queensland;* Frederica F. Nelson, *US DEVCOM Army Research Laboratory;* Jin-Hee Cho, *Virginia Tech*

Reversing a Brake ECU Firmware Update 389
Ben Gardiner, *National Motor Freight Traffic Association Inc.*



Blind Spot in Sensor Fusion: Exploiting an Architectural Vulnerability to Hijack and Precisely Control UAVs

Jongsoo Han
Pohang University of Science and Technology
hanjs@postech.ac.kr

Seulbae Kim*
Pohang University of Science and Technology
seulbae@postech.ac.kr

Abstract

To achieve nimble real-time control while remaining robust to error accumulation, PX4 and ArduPilot UAV control pipelines typically combine high-rate IMU-based attitude controls with low-rate GPS-aided state estimation. We show that this foundational design choice introduces a persistent blind spot: injected gyroscope bias immediately alters control outputs yet remains invisible to the state estimator until slow GPS position updates reveal state inconsistencies. Leveraging this architectural vulnerability, we design a black-box reinforcement learning framework that learns to induce trajectory manipulation. Through asymmetric actor-critic training, we derive a highly effective attack policy that, given bias-injection and external observation capabilities, achieves over 85 % success in steering UAVs into a 10 m-radius cylindrical target region, transfers across different autopilot and airframe types without retraining, bypasses a state-of-the-art anomaly detector and built-in failsafes, and transfers to real-world UAVs.

1 Introduction

Unmanned Aerial Vehicles (UAVs) are now an essential component of modern military and civilian operations, supporting a wide range of tasks such as reconnaissance, logistics, and emergency response. This proliferation has elevated UAV security from a theoretical challenge to an urgent real-world necessity, because UAVs perform tasks where failures can lead to life-threatening consequences. As a result, researchers have increasingly focused on understanding how UAVs can be attacked and how such threats can be effectively detected, mitigated, or prevented.

Existing Attacks. Prior work has exploited UAVs through various attack vectors, among which sensor attacks were dominant. For example, GPS spoofing attempts to redirect a UAV by injecting false satellite signals stronger than authentic ones. However, such attacks are trivially detectable by modern sensor fusion systems and built-in failsafe mechanisms [7, 31].

*Corresponding author.

To avoid GPS-based detection and mitigation, attackers have targeted inertial sensors, *e.g.*, a gyroscope, directly. Acoustic injection attacks [12, 44, 45] and laser-based injection attacks [25] can induce controlled signal injections into MEMS inertial sensors, exploiting a physical channel that leaves no digital trace. Building on these injection capabilities, false data injection (FDI) attacks on magnetometers demonstrated that a UAV's heading can be manipulated by carefully crafting magnetometer measurements that evade EKF-based anomaly detection [5]. Each of these approaches, however, suffers from practical limitations. Inertial sensor injection techniques lack the feedback mechanism required for precise trajectory tracking, resulting in either complete destabilization [32] or only coarse directional drifts rather than targeted navigation [45]. Meanwhile, the magnetometer FDI attack [5] strictly requires the UAV to operate in a GPS-denied environment, where it must rely solely on the magnetometer-based heading estimation.

These limitations expose a *fundamental gap between injection capability and strategic exploitation*. While channels for controlled sensor injection exist, no prior work demonstrates targeted steering of UAVs. Existing attacks either produce uncontrolled effects or are functional only under restrictive conditions, such as GPS-denied environments, limiting their real-world applicability. However, practical adversarial scenarios in GPS-enabled environments necessitate steering a UAV to an attacker-defined region rather than merely causing disruption. Targeted steering of a UAV's final position unlocks advanced objectives that cannot be achieved through uncontrolled drift or forced crashes, such as seizing high-value cargo from logistics UAVs, extracting proprietary technology through hardware reverse engineering, or reprogramming the vehicle for adversarial missions. Achieving this level of targeted control in operational environments where GPS and sensor fusion remain active remains an unsolved and critical challenge, and we aim to address it in this work.

An Architectural Vulnerability. We discovered an architectural vulnerability in widely-used UAV control pipelines that can be exploited to enable targeted steering in GPS-

enabled environments. Both PX4 [39] and ArduPilot [36], the two prominent open-source autopilot systems that power diverse platforms (*e.g.*, agricultural [35, 40] and military UAVs [11, 21]), route gyroscope measurements through two pathways that operate at significantly different effective rates: fast updates feed the rate controller almost immediately, whereas bias estimation by the state estimator, the sensor fusion module, depends on much slower GPS updates. This asymmetry creates a latency window in which injected gyroscope bias influences the controller *before* the state estimator.

Moreover, this latency is amplified by the indirect nature of sensor fusion. Since the gyroscope is the only sensor measuring the angular velocity, the fusion cannot directly validate these readings but must infer errors from accumulated position discrepancies. This slows down the estimator's error correction process, creating an extended window where the injected bias influences the controller before being corrected. As a result, an attacker can introduce attitude deviations that accumulate into position drift.

Two Challenges of Exploitation. We assume that attackers are capable of injecting bias and observing UAV states via existing attack techniques [3, 12, 15, 17, 25, 32, 44, 45]. With these capabilities, exploiting the architectural vulnerability to achieve targeted steering presents two technical challenges. First, the attack policy must remain effective against an adaptive estimator. Since the sensor fusion algorithm continuously compensates for sensor errors, a static bias injection would eventually be neutralized, halting the vehicle. Thus, the policy must *dynamically generate biases* that sustain a sufficient difference from the estimator's internal state to induce drift.

Second, the policy faces a *black-box state inference* challenge: to maintain the bias-to-estimation difference, it must predict the estimator's compensation behavior and inject bias that differs from the estimator's current belief. This requires a predictive model capable of inferring the estimator's hidden internal state (*i.e.*, how much bias it has learned), based solely on the externally observable states. However, analytically constructing such a model is intractable, since the UAV's control pipeline is a meticulous integration of cascaded control loops, nonlinear sensor fusion algorithms, and trajectory planning modules whose interactions create a complex, high-dimensional state space that complicates analytical modeling.

Our Approach. We address these challenges by designing a reinforcement learning framework that learns to exploit the architectural vulnerability via adaptive, state-aware gyroscope bias injection. Our approach uses an asymmetric actor-critic architecture [1, 27]; during training, the critic has access to privileged internal states so it can provide rich feedback to the actor, while the actor is trained to operate using only externally observable information. This solves the black-box state inference challenge, making it directly deployable in real-world black-box scenarios. Since the attack must reason about temporal effects, we equip both the actor and critic with Long Short-Term Memory (LSTM) networks, enabling the agent to learn temporal patterns and implicitly model the estimator's learning dynamics. The trained attack policy, called `GAP` (**G**yroscope **A**ttack **P**olicy), continuously yields gyroscope bias sequences that maintain sufficient magnitude to alter UAV trajectories, solving the dynamic bias generation challenge. Moreover, these sequences remain below the detection thresholds of sensor fusion-based anomaly detectors and physics-based invariant checks [6], concealing the root cause of position drift while steering the UAV toward the target.

Results. Under the assumed injection and observation capabilities, GAP achieves high-accuracy steering into attacker-selected target regions, characterizing an upper bound on steering capability rather than a complete end-to-end physical demonstration. Our evaluation demonstrates an 85.1 % success rate in steering UAVs into a 10 m-radius cylindrical target region, outperforming baselines. In addition, our policy is robust to noise, generalizes across autopilots and airframes without retraining, transfers to real-world flights, and evades a state-of-the-art detector in 76 % of successful attacks.

Contribution. Our key contributions are:

- **Architectural vulnerability analysis:** We identify the gyroscope-EKF correction lag and indirect fusion as a fundamental architectural vulnerability in PX4 and ArduPilot, the two prominent open-source UAV autopilots [2].
- **RL-based black-box steering policy:** We design and present a reinforcement learning-based attack policy that relies solely on external observations to achieve targeted position steering while evading anomaly detectors. We validate the attack in real-world flight experiments and successfully redirect a physical UAV into an attacker-defined target region. Demonstration videos are available at `https://tinyurl.com/UAV-Hijack`.
- **Cross-platform generalizability:** We demonstrate that a single policy trained on PX4 seamlessly transfers to ArduPilot across multiple airframe types without retraining, confirming that the attack exploits shared architectural patterns rather than platform-specific bugs.
- **Open Science:** We have released the trained models, code for evaluation, and all datasets used for evaluation at `https://github.com/postech-compsec/GAP`.

2 Background: UAV Control Pipeline

Overview. Both PX4 and ArduPilot, the two prominent open-source UAV autopilot systems [2], rely on a sophisticated, yet common architectural pattern despite implementation differences [34, 38]. The autopilot system integrates data from multiple sensors: an Inertial Measurement Unit (IMU) containing gyroscopes and accelerometers, sampled at 200 Hz, a GPS for absolute positioning updated at 5 Hz to 10 Hz, and a magnetometer for heading reference updated up to 100 Hz.

An autopilot system processes these sensor inputs through four components, as illustrated in Figure 1. The *Sensors* (❶)

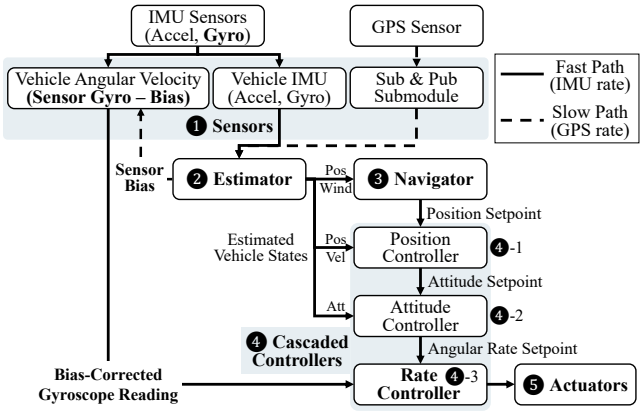


Figure 1: Control pipeline shared by PX4 and ArduPilot.

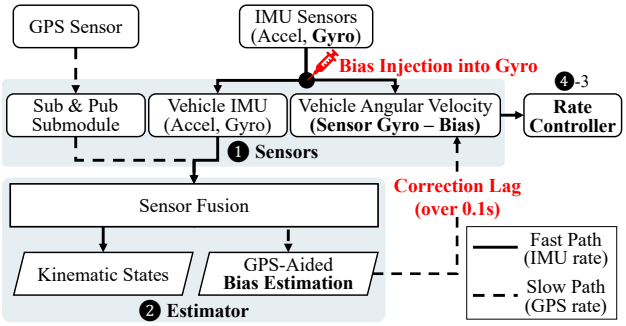


Figure 2: Gyroscope correction lag.

read raw sensor data and publish them to other modules. The *Estimator* (❷) implements sensor fusion, combining sensor data to compute vehicle states. The *Navigator* (❸) generates desired trajectories based on mission objectives. The *Controllers* (❹), via cascaded control loops, emit motor commands to follow the trajectory. The *Actuator* (❺) translates control commands to motor thrusts.

Estimator and Extended Kalman Filter. The Estimator (❷) is implemented using an Extended Kalman Filter (EKF) [30]. It runs at 100 Hz to 200 Hz, fusing multi-sensor measurements to estimate a UAV's states (*i.e.*, the position, orientation, velocities, and sensor biases). To handle varying sensor arrival timings, it synchronizes kinematic states (position, velocity, and attitude) with the IMU's high sampling rate and other states, such as sensor biases (gyroscope and accelerometer) with the GPS's low sampling rate. Most importantly, the estimated sensor biases are published every second. The Sensors module (❶) subscribes to these bias estimates and subtracts them from the raw sensor readings before publishing the corrected angular velocity to the Rate Controller (❹-3).

Indirect Bias Observability. The estimator (EKF) cannot directly observe gyroscope or accelerometer biases because no sensor explicitly measures these biases. It must infer them by detecting inconsistencies between inertial dead reckoning and absolute measurements from external sensors. However, the observability of these biases differs fundamentally. Accelerometer measures linear acceleration, which is what GPS also measures. This enables *head-to-head comparison*; the EKF can immediately detect and learn accelerometer bias by comparing IMU-integrated velocity against GPS velocity measurements, allowing rapid correction when GPS updates arrive. On the other hand, gyroscope measures angular velocity, for which *no other sensor provides a direct measurement*. The EKF cannot directly compare gyroscope readings against any external reference. Instead, gyroscope bias must be *indirectly inferred* by observing discrepancies in downstream states. For example, biased gyroscope produces attitude errors, causing velocity errors that accumulate into position drift. Only by detecting this position drift relative to GPS measurements can the EKF infer gyroscope bias.

Cascaded Control Loops. The Controller module (❹) implements a cascaded control structure with three sequential loops that process control commands from high-level position goals to low-level motor actuation. The *Position Controller* (❹-1), upon receiving the position setpoint from the Navigator (❸), first compares it with the current position from the Estimator (❷) to derive a desired velocity. This is subsequently compared with the current velocity from the Estimator (❷) to generate a desired acceleration. Finally, this acceleration is converted to a desired attitude. The *Attitude Controller* (❹-2) compares the current and desired attitude, producing a desired angular rate. Finally and critically, the *Rate Controller* (❹-3) compares the desired angular rate with the current gyroscope angular rate reading from the sensors module. The rate controller then computes motor thrust commands operating synchronously with gyroscope samples.

3 Preliminary Study

3.1 A Vulnerability Hypothesis

The UAV control pipeline described in §2 is shared across PX4 and ArduPilot. Both route gyroscope-derived body rates into the inner rate-control loop while EKF-level corrections fuse slower external measurements [37, 41]. From this architecture, we hypothesize that the system contains three *architectural vulnerabilities* exploitable as critical attack vectors:

V1. **Fast Path**: The Rate Controller (❹-3) consumes signal-level filtered gyroscope readings, before EKF-level bias correction takes effect. As a result, injected angular velocity bias immediately alters control outputs to Actuators (❺), producing instant physical reactions.

V2. **Slow Path**: The Estimator (❷) attempts to learn and correct sensor biases. However, this process is limited by the arrival rate of GPS updates, which is far slower than that of IMU updates. This creates an inherent delay for the estimator to begin responding to injected errors.

V3. Indirect Observability: As shown in Figure 2, even after GPS readings arrive, the estimator still cannot directly measure gyroscope bias. It must indirectly infer it from physical state discrepancies, making bias estimation intrinsically slow and imprecise.

These observations motivate the following hypothesis: gyroscope bias injection is uniquely capable of exploiting all three architectural vulnerabilities, whereas other sensor injection attacks are not. To validate this hypothesis, we perform the following preliminary experiments.

3.2 Experimental Validation

To validate our discovery of the architectural vulnerability (§3.1), we conducted controlled experiments comparing *gyroscope* and *accelerometer* bias injection attacks' impacts. Our primary objective is to confirm that gyroscope bias injection, unlike other sensor attacks, can effectively exploit the fast path (**V1**) to induce immediate physical deviations, the slow path (**V2**) and the indirect observability (**V3**) to delay correction of these deviations.

Experimental Setup. We implemented a simulated sensor attack environment using PX4 v1.15.4 and the jMAVSim simulator. For the experiment, we utilized custom MAVLink[1] messages to enable real-time bias injection with precise timestamp logging. The simulated UAV was an Iris quadcopter with GPS and IMU sensors. In each trial, the attack was initiated while the UAV hovered at 60 m altitude. We repeated this procedure 50 times per sensor. The GPS update rate in the Software in the Loop (SITL) simulator ran at 20 Hz, higher than the typical 5 Hz to 10 Hz in real deployments.

Since angular velocity and linear acceleration represent fundamentally different physical quantities, direct magnitude comparison of gyroscope and accelerometer attack is infeasible. For fair comparison, we calibrated the two attacks' magnitudes so that their resulting physical impact over the first second of injection is identical. For gyroscope attacks, we injected a bias of 0.06 rad/s on the x-axis and the y-axis, which accumulates to an attitude error of 0.06 rad after one second. This attitude error rotates the gravity vector g where $|g| = 9.8\,\mathrm{m/s^2}$ to produce approximately $0.6\,\mathrm{m/s^2}$ horizontal acceleration (as $g \cdot \sin(0.06) \approx 0.59\,\mathrm{m/s^2}$). For accelerometer attacks, we directly injected a bias of $0.6\,\mathrm{m/s^2}$ on the y-axis, matching the approximate horizontal acceleration produced by the gyroscope attack.

Result 1: Immediate Control Reaction. We first examine the UAV's immediate physical response under each attack type to confirm that gyroscope attack can exploit **V1**.

Figure 3 shows the impact of gyroscope bias injection, averaged over 50 flights. $\dot{\phi}_{est}$ (roll rate estimate, orange line) represents the gyroscope reading used by the controller after

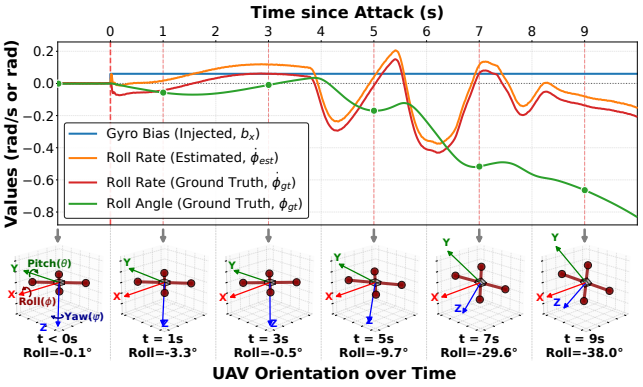


Figure 3: Rate controller's immediate reaction to gyroscope bias, leading to instant tilting of the UAV.

subtracting the currently estimated bias. When a 0.06 rad/s b_x (injected gyroscope bias, blue line) is injected at $t = 0$, the controller interprets this step change as unintended rotation. To maintain the original hover setpoint (*i.e.*, $\phi = 0$), the controller commands an immediate counter-movement to nullify this perceived error. Consequently, $\dot{\phi}_{gt}$ (roll rate ground truth, red line) shifts to -0.06 rad/s, as the Rate Controller (❹-3) applies counter-control to cancel the error. From the controller's perspective, stability appears restored as $\dot{\phi}_{est}$ returns to zero. However, this stability is illusory. To keep the biased sensor reading at zero, the UAV must physically rotate at -0.06 rad/s along the x-axis. This persistent and uncorrected b_x eventually causes the attitude (*i.e.*, ϕ_{gt}: roll ground truth) to diverge continuously, tilting the UAV from -0.1° at $t = 0$ to -38.0° at $t = 9$ as shown in Figure 3 (bottom).

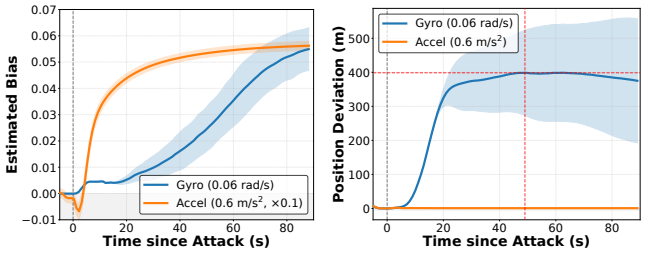
This pipeline-level behavior complements prior Attitude and Heading Reference System (AHRS)-level analysis. Nashimoto *et al.* [23] showed that under low-noise conditions the AHRS Kalman filter heavily weights accelerometer and magnetometer corrections, suppressing gyroscope-only spoofing at the estimator. However, their analysis examines the AHRS in isolation. In the integrated pipeline, the spoofed gyroscope reaches the rate controller via the fast path (**V1**) before estimator-level correction, producing observed tilting.

Result 2: Delayed Estimation Response. To evaluate the exploitability of the rate-limited correction (**V2**) and indirect observability (**V3**), we compare how the EKF estimates bias under gyroscope and accelerometer attacks.

Table 1: Impact comparison of gyroscope (Gyro.) vs. accelerometer (Accel.) attacks. Relative differences highlight gyroscope bias's strength over accelerometer bias.

Metric	Gyro.	Accel.	Relative Diff.
Detection Delay (s)	2.464 ± 0.045	0.138 ± 0.1	17.9× longer
Peak Deviation (m)	404.5 ± 125.4	2.2 ± 0.0	184× larger
Time to Peak Dev. (s)	49.0	3.0	16.3× longer

[1] A lightweight, open communication protocol used by UAVs and ground stations to exchange telemetry, command, and sensor data in real time.



(a) Bias estimation process **(b)** Position deviation

Figure 4: Comparison of the bias estimation and drift behaviors upon gyroscope and accelerometer attacks.

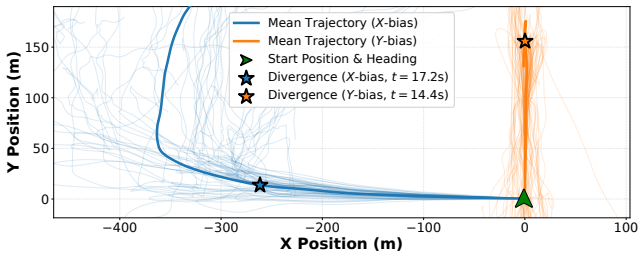


Figure 5: UAV trajectories observed across 50 bias injection trials. Trajectory consistency severely drops beyond the divergence point, marked with stars.

The results are summarized in Table 1. Although both attacks remain unnoticed until the next GPS update (**V2**), the detection delay, defined as the moment when the EKF's bias estimate first changes by more than the publication threshold, differs dramatically; accelerometer bias is detected almost immediately at 0.138 s because the EKF can directly compare IMU-integrated velocity with GPS velocity. In contrast, gyroscope bias detection requires 2.464 s (17.9× longer), since the EKF must wait for downstream position discrepancies to accumulate before inferring angular-rate errors through the indirect observability chain (**V3**). Figure 4(a) visualizes these dynamics. The accelerometer estimate exhibits an initial dip followed by a rapid rise along a steep curve (orange line), reflecting fast and precise estimation. Gyroscope bias estimation, however, begins only after a substantial delay and increases gradually along a gentle slope (blue line), demonstrating the slow and indirect nature of gyroscope bias inference. This confirms the existence of an exploitable window.

Result 3: Divergent Position Dynamics. The estimator's slow response to gyroscope bias attacks also leads to clearly different physical outcomes in the two attacks. As shown in Figure 4(b), the accelerometer attack produces only minor displacement from the initial position. Since the estimator quickly compensates for the injected bias, the peak deviation is only 2.2 m (shown in Table 1). In contrast, the gyroscope attack leverages the indirect fusion pathway, causing the UAV to drift much farther. The displacement continues to increase until it reaches 404.5 m at 49 s (denoted by red dotted lines in Figure 4(b)) at which point the estimator has gathered enough information to start countering the motion.

Result 4: Predictable Trajectory. Finally, we analyze the trajectory patterns across the 50 independent attacks to identify the time interval during which UAV motion remains consistent. The results are summarized in Table 2.

Table 2: Predictable attack trajectory characteristics.

Characteristic	X-axis Attack	Y-axis Attack
Size of Attack Window (s)	17.2	14.4
Mean Position at Divergence (m)	(-260.1, 13.5)	(0.2, 155.8)
Drift Direction	Leftward	Forward

Under identical conditions and identical attacks, the UAV exhibits highly reproducible movement patterns. These repeatable drift patterns arise from how the injected bias affects the controller in the UAV's body frame. The bias is applied on the body-frame axes: the x-axis points forward, corresponding to ϕ (roll), and the y-axis points right, corresponding to θ (pitch). When a bias is injected on the x-axis, the controller interprets it as an unintended right-roll motion and issues a counter-movement by commanding left tilt. This tilt causes the UAV to drift leftward in the global North-East-Down (NED) frame. Similarly, a y-axis bias is interpreted as an unintended pitch change, and the controller compensates by tilting forward, producing forward drift in global coordinates.

As illustrated in Figure 5, across 50 repeated flights with randomized internal states (*e.g.*, EKF covariance and controller setpoints), these counter-movements produce nearly identical early-stage trajectories. This consistency indicates that the UAV's response to bias injection is highly predictable during this initial interval, creating a reliable attack window.

In light of this, we define "attack window" as the period where trajectory standard deviation remains below 10 m, which represents the time during which UAV movements can be reliably predicted. Specifically, for x-axis attacks, this window lasts until 17.2 s, at which point the mean position reaches (-260.1, 13.5) m relative to the hovering position with a standard deviation of 10.2 m. For y-axis attacks, the window closes at 14.4 s with the mean position of (0.2, 155.8) m and a standard deviation of 10.0 m. Beyond these points, trajectory variance increases rapidly as EKF learning progresses at different rates across trials with diverging GPS measurement timing variations and accumulated errors.

Summary. These preliminary experiments validate the architectural vulnerabilities hypothesized in §3.1 and establish the foundation for our attack. **Result 1** confirms that the rate controller reacts instantly to gyroscope bias. **Result 2** shows that slow GPS updates and indirect fusion create a sizable window where injected bias remains uncorrected. **Results 3 and 4** show that UAV motion within this window is highly predictable. Together, these findings expose an exploitable and learnable attack surface.

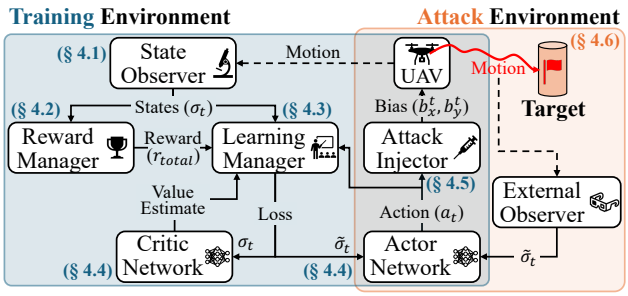


Figure 6: Design and workflow of our reinforcement learning-based gyroscope bias injection attack framework. The trained Actor Network (*i.e.*, attack policy) from the training environment is deployed to attack environment.

3.3 Threat Model

Attacker Capabilities. Building upon the preliminary study, we assume that attackers possess the capability to inject controlled bias into the X and Y axes of the UAV's gyroscope, maintaining a chosen value for one second and updating it at a 1 Hz decision rate. Touch-based acoustic attacks [12] supply this capability via a paster-like unit attached through one-shot physical contact, which drives the gyroscope through solid-medium acoustic coupling during flight, bypassing the attenuation and resonance issues of remote acoustic injection. Gao *et al.* demonstrated this on a Pixhawk4 drone running PX4, sustaining controllable in-flight gyroscope bias of 5 deg/s. Laser-based injection [25], supply chain compromise or sensor driver exploits are additional plausible vectors. We also assume that attackers can observe the UAV's external state (*e.g.*, position and attitude) through visual tracking [3, 15, 17] or publicly broadcast telemetry.

We adopt a realistic *black-box* threat model where attackers cannot read raw sensor values, EKF state, or controller states and setpoints. This reflects typical scenarios where attackers must operate solely on observable UAV behavior.

Attacker Goals. The attacker's primary objective is redirecting a UAV from its current position to an attacker-selected target region (*e.g.*, hundreds of meters away). Success is defined by target-region entry. Arbitrary waypoint-level control and stable terminal stopping are out of scope.

4 Design

The preliminary study in §3 reveals an exploitable window with predictable trajectory patterns. However, deriving deterministic dynamics models to exploit this is infeasible due to nonlinear dynamics of the autopilot stack. We thus propose a closed-loop reinforcement learning (RL) framework where the RL agent autonomously learns modulating gyroscope bias injection to steer the UAV toward a target region.

Overview. Figure 6 illustrates our framework, which first trains an RL model in a training environment and then deploys it to an attack environment. During training, the **State Observer** (§4.1) extracts a 78-dimensional state vector σ_t from the simulator. The **Reward Manager** (§4.2) evaluates σ_t and computes a scalar reward r_{total} that reflects progress toward target-region steering. The **Learning Manager** (§4.3) receives σ_t and constructs two representations: full σ_t for the Critic Network and a partial state vector $\tilde{\sigma}_t$ for the Actor Network. It also computes learning signals such as advantages and loss terms. The **Asymmetric Actor-Critic Networks** (§4.4), both equipped with LSTM layers, process these inputs: the Critic Network consumes σ_t to produce value estimates $v(\sigma_t)$, while the actor consumes only $\tilde{\sigma}_t$ to produce a continuous gyroscope-bias action $a_t = (b_x^t, b_y^t)$. The **Attack Injector** (§4.5) applies this action to the UAV, influencing its motion.

After training, in the **Attack Environment** (§4.6), the trained Actor Network is paired with an External Observer that reconstructs $\tilde{\sigma}_t$ from only the UAV's visible motion, adhering to our threat model (§3.3). This allows the Actor Network to generate bias values and steer the UAV toward the target region in a fully black-box setting.

4.1 State Observer

The State Observer serves as the interface between a UAV's physical dynamics and the learning pipeline. During training, it accesses all internal simulator variables, including:

- **s1**. Normalized time since attack start ($\mathbb{R}^1$)
- **s2**. Relative position from UAV to target region ($\mathbb{R}^3$)
- **s3**. Three-axis linear velocities ($\mathbb{R}^3$)
- **s4**. Three-axis attitude angles ($\mathbb{R}^3$)
- **s5**. Three-axis angular velocities ($\mathbb{R}^3$)
- **s6**. Geometric features ($\mathbb{R}^8$): distance and direction to the target region, approaching velocity, heading error, distance to altitude limits and geofence boundaries, success indicator, and total velocity magnitude
- **s7**. Previous action (a_{t-1}: last injected gyroscope bias, $\mathbb{R}^2$)
- **s8**. EKF bias estimates and variances for gyroscope and accelerometer ($\mathbb{R}^{12}$)
- **s9**. GPS and heading innovation terms and variances ($\mathbb{R}^{14}$)
- **s10**. Controller setpoints ($\mathbb{R}^{12}$): position setpoints, velocity setpoints, attitude setpoints, and angular rate setpoints
- **s11**. Test ratios and position accuracy metrics ($\mathbb{R}^5$)
- **s12**. Navigation state, control and failsafe mode flags ($\mathbb{R}^9$)
- **s13**. Difference between EKF-estimated gyroscope bias and the injected bias ($\mathbb{R}^3$)

The observer converts these into a 78-dimensional state vector σ_t and feeds the Reward Manager (§4.2) and Learning Manager (§4.3), allowing the learning pipeline to track immediate effects of injected gyroscope bias on flight dynamics.

4.2 Reward Manager

The Reward Manager computes the scalar reward signal that guides RL policy learning. Given the full state vector σ_t, it evaluates the UAV's progress toward the target region and computes the total reward

$$r_{\text{total}} = w_d \cdot r_{\text{distance}} + w_v \cdot r_{\text{velocity}} + r_{\text{time}} + r_{\text{termination}} \quad (1)$$

where weights w_d and w_v are configurable[2].

- The **distance reward** $r_{\text{distance}} = (d^{t-1}_{\text{target}} - d^{t}_{\text{target}})$ incentivizes reducing the 3D Euclidean distance d_{target} from the UAV to the target region.
- The **velocity reward** $r_{\text{velocity}} = v_{\text{closing}}$ encourages approaching the target with positive closing velocity, where v_{closing} is the velocity component toward the target. These distance and velocity rewards provide dense feedback that helps the actor discover effective policies even when success is sparse, acting as a hint to guide exploration.
- The **time penalty** $r_{\text{time}} = -0.01$ encourages faster convergence, preventing the actor from learning inefficient attack strategies that require excessive mission duration.
- The **terminal reward** $r_{\text{termination}}$ encodes success and failure conditions. When the UAV enters the target region, the agent receives a reward of $+20$, while timeout, altitude violation, or excessive deviation from the target each incur a penalty of -20. These terminal conditions are continuously evaluated to ensure that brief or transient entries into the target region are correctly detected.

This reward structure balances dense feedback that accelerates learning with sparse terminal rewards for mission success.

4.3 Learning Manager

The Learning Manager coordinates the entire training loop. Upon receiving the full 78-dimensional σ_t from the State Observer, it constructs the partial observation $\tilde{\sigma}_t$ used by the actor. This 23-dimensional vector consists of 21 externally observable quantities (**s1-s6** in §4.1) along with the previous action **s7**: $a_{t-1} = (b_x^{t-1}, b_y^{t-1})$.

In addition to producing state vectors, the Learning Manager computes learning signals, including advantages and loss terms for both Actor and Critic Networks. It then updates both networks through backpropagation of the combined policy and value losses.

4.4 Asymmetric Actor-Critic Networks

Asymmetric Architecture. To satisfy a realistic threat model (§3.3) while maintaining efficient training performance, we adopt an asymmetric actor-critic architecture [1, 27] that separates privileged and non-privileged information pathways. During training, the **Critic Network** receives the full 78-dimensional state vector σ_t, enabling it to compute accurate value estimates and provide informative gradients that accelerate learning. However, the **Actor Network** receives only the 23-dimensional partial state vector $\tilde{\sigma}_t$ that an attacker can realistically construct from external sensing.

LSTM Layers. To further enhance the agent's capability to exploit the temporal attack window (14 s to 17 s depending on axis identified in §3.2), both networks incorporate Long Short-Term Memory (LSTM) layers [14]. This recurrent architecture allows the policy to capture the temporal dynamics of the estimator's gradual bias compensation and the delayed physical response, enabling the agent to reason about *when* to modulate bias injection intensity.

Bounded Actions. The actor ultimately outputs a continuous two-dimensional gyroscope bias **action** $a_t = (b_x^t, b_y^t)$, where each dimension is constrained within $\pm 0.06\,\text{rad/s}$. These bounds were selected to preserve stealthiness while providing sufficient controllability, based on PX4's bias detection thresholds. Its estimator triggers a failsafe when the gyroscope bias exceeds $0.15\,\text{rad/s}$ ($\approx 8.59\,^\circ/\text{s}$), identifying it as a sensor malfunction. Thus, we restrict the attack bias to $0.06\,\text{rad/s}$, providing a $2.5\times$ safety margin below the detection threshold.

4.5 Attack Injector

The Attack Injector applies the actor-generated bias values to the UAV's gyroscope measurements. For each decision step, it modifies the IMU data stream by adding the continuous bias vector to the UAV's angular-rate readings. This perturbation propagates through the UAV's sensor-fusion stack and cascaded control loops, leading to trajectory deviation. This injection capability is grounded in known methodologies. Physical channels such as acoustic [12, 45] and laser-based [25] injection can impose controlled bias on MEMS gyroscopes without requiring software or network access to the UAV; supply chain compromise or sensor driver exploits represent additional system-level vectors. Critically, the attacker needs only to add a bounded bias to the gyroscope stream, with no access to flight commands or internal states. Such channels can maintain the bias for several seconds, meeting the 1 s hold duration required by our 1 Hz decision rate.

4.6 Attack Environment

In the deployment environment, only the trained Actor Network, *i.e.*, the actor policy, is used. Because the attacker has no access to internal UAV states, an External Observer reconstructs the $\tilde{\sigma}_t$ used by the actor. This observer relies solely on the physically observable quantities, such as motion and attitude, inferable from external sensing. Like in the training environment, the actor consumes $\tilde{\sigma}_t$ and outputs a_t, which the Attack Injector (§4.5) applies to the target UAV. The injection loop repeats continuously, enabling the attacker to steer the UAV toward the target region in real time despite having no visibility into the internal estimator states or controller logic.

[2] We empirically tuned the weights to $w_d = 0.1$ and $w_v = 0.025$.

5 Training Strategy

We adopt a stage-wise training approach to achieve the final objective: guiding the UAV into a 10 m-radius target region from an initial distance of 220 m with over 80 % success rate.

Initial Conditions. In all training episodes, the target region is fixed at the origin, and a UAV is initialized in position-hold mode at a random angular position on a 220 m-radius circle around the origin at 60 m altitude (see Figure 7). We implement domain randomization [19, 26, 43] by varying this initial spawn position while fixing the UAV's initial heading to North. This setup is geometrically equivalent to placing the UAV at the origin and commanding it to move toward the target from all possible relative directions in its body frame, ensuring the policy learns omnidirectional control rather than overfitting to specific maneuvers (*e.g.*, only flying forward).

The 220 m starting distance serves as a representative standard for validating long-range robustness; success at this range implies that shorter-range attacks are readily achievable, while longer-range attacks can be constructed by chaining multiple segments as sequential waypoints.

Intermediate Policy: Learning Coarse 3D Control. Training begins with a 20 m-radius *spherical* target region. This geometry offers uniform feedback across all three axes, allowing the agent to first acquire coarse 3D stabilization skills. Especially, the spherical region provides dense, well-shaped reward signals during early exploration, enabling the Actor-Critic Networks to learn how to regulate the UAV's trajectory without diverging. We ran this stage until the success rate exceeded 80 %, which required approximately 10K iterations.

Final Policy: Refined Horizontal Control. After establishing robust coarse control, we transition into the final objective: a *cylindrical* target region with a 10 m-radius and a relaxed 0 m to 120 m altitude tolerance. This transition is motivated by the physical characteristics of the gyroscope bias attacks; injected bias provides strong horizontal steering controls through accumulated attitude deviation, but its influence over the vertical motion is limited, which is primarily governed by throttle control. The cylindrical geometry is well-aligned with the true controllable dimensions of the attack: it demands high horizontal precision, while avoiding unrealistic vertical constraints that the attack cannot satisfy.

We transferred the learned weights from the intermediate policy and continued training with the new cylindrical objective. This curriculum learning strategy [4, 22] allows the actor to reuse the stabilization and trajectory-shaping capabilities acquired during the initial training, while refining horizontal accuracy for the final objective. The resulting policy exceeded 80 % success rate within 6K iterations.

We refer to the final attack policy as `GAP` (**G**yroscope **A**ttack **P**olicy).

6 Implementation

Configuration. We implement the training environment of our framework by integrating the RL architecture from §4 with a high-fidelity UAV simulation environment. The UAV runs an unmodified PX4 autopilot with default estimator and controller configurations. We focus on off-the-shelf multicopter UAVs that rely on IMU, magnetometer, and GPS-based state estimation, with vision-based localization (e.g., VIO) and camera-assisted navigation out of scope. All sensors (GPS, accelerometers, and magnetometers) operate normally, except the gyroscope stream, where the Attack Injector (§4.5) injects bias values produced by the Actor Network (§4.4).

Attack Injector. As defined in §3.3, our threat model assumes controlled gyroscope bias injection via any physical channel. In our implementation, we emulate this capability via a custom MAVLink message, used solely for deterministic and reproducible simulation where physical channels (e.g., acoustic, optical) cannot be faithfully reproduced. The injected values are applied at the sensor driver prior to EKF fusion, emulating the effect of a physical signal injection. This design choice follows standard evaluation practice in cyber-physical security, where hardware-triggered effects are emulated through software injection to ensure repeatability.

Training Infrastructure. Our implementation depends on Python 3.10.18, PyTorch 2.8.0, and Ray RLlib 2.48.0 [18, 46], a scalable distributed RL framework, with the APPO (Asynchronous Proximal Policy Optimization) algorithm [20]. Our custom `AsymmetricLSTMModule` extends its `TorchRLModule` base class to support separate Actor and Critic Networks with distinct observation spaces.

Training employs 30 distributed workers (each running a PX4 instance in jMAVSim simulator with 1 CPU core and 0.1 GPU, distributed across 1 RTX 4060 Ti and 2 Tesla V100) and a learner with 4 CPU cores and 1 RTX 4090 GPU.

Training Environment Details. We bridge our RL framework to PX4 v1.15.4's Software-in-the-Loop firmware via MAVLink protocol. By default, the state estimator and the Rate Controller operate at 250 Hz in the simulator.

The environment launches PX4 and jMAVSim instances per parallel worker, injects gyroscope bias via a custom MAVLink message, receives ground truth kinematic states and privileged information. All simulations run at 4× the wall clock speed. During training, we use ground truth states for the actor's observation (assuming ideal external tracking) and disable external aerodynamic disturbances for stable learning.

Network Architecture Details. The **Actor Network** comprises a 3-layer fully-connected encoder with 512 hidden units per layer, followed by a 2-layer LSTM with 512 hidden units per layer. All fully connected layers use ReLU activation and layer normalization. The **Critic Network** uses a 4-layer, fully-connected encoder with 1024 hidden units per layer followed by a 2-layer LSTM with 1024 hidden units, reflecting the

critic's need to model complex EKF convergence dynamics from privileged information while the actor operates under partial observability with a simpler decision space.

Hyperparameters. Key RL hyperparameters: learning rate $\alpha = 5 \times 10^{-5}$, gradient clipping $\tau_{\text{grad}} = 0.5$, discount factor $\gamma = 0.995$, PPO clip ratio $\varepsilon = 0.2$, entropy $\beta = 0.02$, and value loss $c_v = 1.0$. We use V-trace clipping thresholds $\rho_{\text{clip}} = \bar{\rho}_{\text{clip}} = 1.5$ for off-policy stabilization, with batch size 4096 timesteps, 64-timestep rollout fragments per worker, 1024-timestep minibatches, and 2 epochs per iteration.

7 Evaluation

In this section, we evaluate the effectiveness of our attack framework by answering the following research questions:

RQ1. Can our attack policy (GAP) successfully steer UAVs to a desired position, compared to baselines? (§7.1)
RQ2. Is GAP robust to tracking and injection noises? (§7.2)
RQ3. Is GAP transferable to heterogeneous autopilot platforms and airframes without retraining? (§7.3)
RQ4. Can GAP bypass existing anomaly detectors? (§7.4)
RQ5. Is GAP resilient to built-in failsafe mechanisms? (§7.5)
RQ6. Is the proposed attack feasible in real-world flight missions under environmental disturbances? (§7.6)

Experimental Setup. Unless stated otherwise, all experiments use PX4's default configurations under calm conditions (no wind) with inherent sensor noise enabled for realistic EKF behavior. During evaluation, state observations are derived from the simulator's ground truth kinematic outputs, assuming that an attacker is equipped with external tracking capabilities.

Episode Initialization. To rigorously evaluate the policy's omnidirectional adaptability, UAVs spawn on the circumference of a 220 m-radius circle centered on the target, at a fixed altitude of 60 m. We generate 12 equally spaced spawn locations (every 30°) to verify, with deterministic and uniform coverage of all approach directions, whether the domain-randomization-trained policy generalizes without overfitting to specific trajectories. Episodes timeout upon safety violations or upon reaching a maximum duration of 300 s.

Attack Target Region. To comprehensively assess control precision, we report the attack success rate for the following four target regions: 20 m-radius cylinder and sphere, and 10 m-radius cylinder and sphere. Unless otherwise stated, the "success rate" is measured on the 10 m cylinder target.

Demo Video. The video recordings of our experiments, including the attack on real-world UAVs (§7.6), are available at https://tinyurl.com/UAV-Hijack.

7.1 RQ1: Effectiveness of Trained Policy

We evaluated the effectiveness of GAP (the final policy acquired in §5) by comparing it against three baseline attacks: random bias injection, directional bias injection, and adaptive bias injection. Random bias injection represents a non-strategic adversary who injects gyroscope bias values uniformly sampled from the range $[-0.06, 0.06]$ rad/s into the roll (x) and pitch (y) axes, resampled and updated at a frequency of 1 Hz. Directional bias injection applies a static, feedback-free strategy: based on the UAV's initial spawn angle θ relative to target ($\theta = 0°$ for North, increasing clockwise), it injects a fixed bias vector $b_x = 0.06 \cdot \sin(\theta)$, $b_y = -0.06 \cdot \cos(\theta)$ to induce a persistent drift toward the center. Adaptive bias injection extends these baselines by recalculating the bias vector at 1 Hz, based on the UAV's current position and heading, representing the strongest baseline.

Results. We conducted a comparative evaluation consisting of 24 episodes for each baseline (2 trials at each of the 12 spawn positions) and a large-scale test of 1,200 episodes for our attack policy to ensure statistical robustness. As summarized in Table 3, our policy achieved an 85.1 % success rate on the 10 m cylinder target, outperforming all baselines.

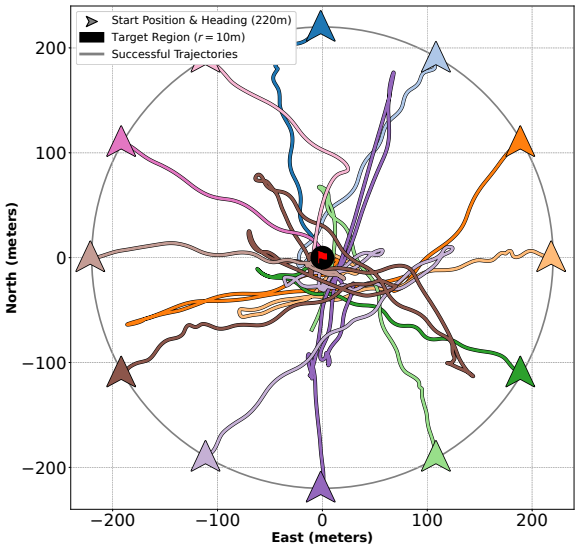
Random bias failed to induce consistent drift, while directional bias allowed the estimator to accumulate error corrections, causing the trajectory to diverge before reaching the target. Adaptive bias reached only 20.8 % on the 10 m cylinder target, as it could not model the EKF's internal compensation dynamics. Among the cases where adaptive bias succeeded, GAP reached the target 64 % faster on average (54.5 s compared to 151.3 s on the 10 m cylinder), further demonstrating the efficiency of GAP.

Figure 7(a) visualizes representative successful trajectories from all 12 spawn positions, showing consistent steering across varying approach angles.

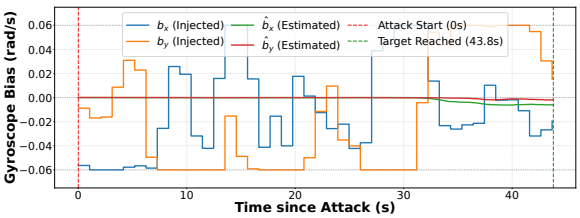
Anatomy of a Successful Attack. Figure 7(b) illustrates the policy's exploitation of the UAV's dynamics in a West-spawned episode. To reach the target at the center, the UAV must drift rightward. Accordingly, the agent initially sustains a maximum x-axis bias of -0.06 rad/s for 7 s. Physically, injecting this negative bias deceives the controller to trigger a counter-acting rightward roll command that drives the vehicle toward the target. This sustained injection is critical: it forces the UAV to tilt significantly, overcoming the vehicle's initial inertia and position-hold stabilization. Once the UAV gains sufficient momentum, the agent switches to a dynamic modulation pattern, updating the bias every second. This behavior demonstrates that the RL policy has learned to leverage the predictable attack window (14 s to 17 s) identified in §3.2.

Table 3: Baseline vs. Our RL-based attack comparison.

Attack Method	Success Rate (%)			
	Cyl. 20 m	Sph. 20 m	Cyl. 10 m	Sph. 10 m
Random Bias	0.0	0.0	0.0	0.0
Directional Bias	4.2	4.2	0.0	0.0
Adaptive Bias	41.7	29.2	20.8	8.3
GAP	93.2	85.0	**85.1**	59.1



(a) Successful trajectories into a 10 m-radius cylinder



(b) An example of state-aware gyroscope bias injection

Figure 7: Analysis of GAP's successful attack cases.

Underlying these decisions, a key requirement is maintaining a discrepancy between the injected bias (b_x and b_y) and the EKF's estimated gyroscope bias($\hat{b}_x$ and $\hat{b}_y$). Once the EKF fully learns the injected bias, the bias-corrected angular rate input to the controller is neutralized, eliminating the induced drift. As shown in Figure 7(b), both $\hat{b}_x$ and $\hat{b}_y$ remain close to zero throughout the episode, suggesting that GAP keeps the estimator from converging to the injected bias.

7.2 RQ2: Robustness to Realistic Noise

Our threat model assumes that the attacker observes the target UAV state via an external vision-based tracker, which tends to be noisy. To further test the robustness of our policy against a realistic attacker, we corrupted the simulator UAV state before feeding it to GAP by adding two types of noises:

Noise source 1: Tracking. We derived noise parameters from Li *et al.* [17], which evaluates 6-DoF pose estimation of a quadrotor at 20 m to 52 m using a telephoto lens. Under their metric, they report that 87.49 % of position estimates have errors below 0.707 m, and 7.42 % of attitude estimates have errors below 0.707°. We modeled per-axis Gaussian noise for position and attitude as $\Delta p \sim \mathcal{N}(0, \sigma_p^2)$ and $\Delta\theta \sim \mathcal{N}(0, \sigma_\theta^2)$. Matching the reported acceptance probabilities yielded $\sigma_p = 0.295$ m and $\sigma_\theta = 1.033°$. Since velocity is estimated as the difference of two consecutive position observations, its noise variance doubles; at the 1 Hz policy sampling interval this gives $\sigma_v = \sqrt{2}\sigma_p/\Delta t = 0.417$ m/s and $\sigma_\omega = \sqrt{2}\sigma_\theta/\Delta t = 0.026$ rad/s. These noises were applied to the states fed to GAP.

Noise source 2: Injection delays and losses. We modeled injection-side imperfections based on our real-world flight logs obtained in §7.6: bias injection command loss with 12.5 % probability, and bias injection period jitter sampled from $\max(1, \mathcal{N}(1.087, 0.076^2))$ s, both derived from measurements across seven flight logs.

Results. Table 4 summarizes results over 1,200 evaluation episodes for each noise source. GAP remained effective under all criteria, demonstrating robustness against tracker-induced uncertainty as well as injection-side delays and losses, without any noise-aware retraining. These results stemmed from three factors. First, injection delays were inherent during training, as the RL framework and the PX4 ran as separate processes communicating over MAVLink, naturally exposing the policy to timing imperfections. Second, because the policy recalculated a new bias based on the newly observed state at each step, missed injections were compensated for at the subsequent decision step. Lastly, robustness to tracking noise was consistent with the known properties of recurrent policies under noisy observations [14]. These noise sources approximate key real-world imperfections but do not cover all aspects of a physical deployment, such as residual bias from injection hardware or distance-dependent tracking degradation. We further discuss this gap in §8.

7.3 RQ3: Cross-platform Generalizability

To show the generalizability of our attacks, we extended our evaluation to a different autopilot platform (ArduPilot) with diverse airframe types, and to a different simulator (*i.e.*, PX4 with Gazebo simulator).

We deployed GAP, which was originally trained on PX4 and jMAVSim simulator, directly to these two configurations *without* any retraining. We conducted 24 evaluation episodes per configuration, consisting of two independent trials at each

Table 4: GAP success rates under tracking noise and injection imperfections. 'None' (no noise) results are from RQ1, and 'Both' condition is when both noise sources were present.

Noise Condition	Success Rate (%)			
	Cyl. 20 m	Sph. 20 m	Cyl. 10 m	Sph. 10 m
None	93.2	85.0	85.1	59.1
Tracking Noise	95.3	89.9	89.2	67.3
Delay & Loss	94.9	87.4	87.3	64.2
Both	95.4	88.1	87.2	63.9

of the 12 equally-spaced spawn positions.

Results. Table 5 shows the success rates across 9 ArduPilot airframes and the PX4 Gazebo environment. The policy achieved 70.8 % to 87.5 % success rates on all multirotor configurations except coaxcopter. This successful transfer confirmed that our attack exploits a fundamental architectural vulnerability (§3.2) rather than platform-specific implementation bugs. In the case of coaxcopter, the success rate dropped to 54.2 % because its control pipeline significantly differed from the common pipeline that we identify in §2.

7.4 RQ4: Detection Evasion

We evaluated the stealthiness of GAP against CI [6], a powerful anomaly detector that identifies attacks by checking physics-based control invariants, *i.e.*, the correlation between sensor measurements and control commands. We utilized their VMware image, which runs ArduPilot v3.4.

Results. We conducted 24 attack episodes, consisting of two independent trials at each of the 12 equally-spaced spawn positions. The attack achieved an 87.5 % success rate on the 10 m cylinder target, with CI detecting only 23.8 % of these successful episodes. Notably, detections occurred exclusively at specific approach angles (3, 4, and 8 o'clock directions), with zero detections in all other orientations, showing that our attack effectively exploited blind spots in the invariant relationships modeled by the detector.

7.5 RQ5: Resilience to Failsafe Mechanisms

The attack experiments in RQ1 (§7.1), also revealed that our attack is robust against built-in failsafe mechanisms. Our observations can be summarized as follows:

Observation 1: Gyroscope Failsafes are Bypassed. Across all PX4 experiments, the gyroscope bias failsafe was *never* triggered. This indicates that the injected bias consistently remained below the platform's detection thresholds, preventing the autopilot from attributing the resulting anomalies to gyroscope sensor corruption.

Table 5: Success Rate Across Platforms and Airframes.

Platform	Airframe	Success Rate (%)			
		Cyl. 20 m	Sph. 20 m	Cyl. 10 m	Sph. 10 m
PX4	x500 (Gazebo)	91.7	70.8	83.3	33.3
ArduPilot	coaxcopter	75.0	62.5	54.2	37.5
	dodeca-hexa	87.5	83.3	70.8	62.5
	hexa	87.5	87.5	79.2	79.2
	octa	91.7	91.7	75.0	66.7
	octaquad	91.7	91.7	87.5	83.3
	quad	87.5	83.3	79.2	75.0
	singlecopter	95.8	95.8	83.3	79.2
	tri	83.3	79.2	79.2	79.2
	y6	95.8	91.7	79.2	70.8

Observation 2: Our attack ignores other failsafes. In 20.3 % of attack episodes, attack-induced drift triggered generic failsafes (*e.g.*, "invalid local position" and "invalid local velocity"), causing the autopilots to initiate safety behaviors such as Landing or Return-to-Launch (RTL). However, entering these modes did not neutralize our attack; in every case, the triggered failsafe was immediately deactivated within an average of 2.1 s and our attack continued to steer the UAV to the target region. This demonstrates that even when failsafe logic engages, it is fundamentally ineffective against our gyroscope-based attacks.

7.6 RQ6: Sim-to-Real Validation

Finally, we evaluate whether GAP transfers to physical flight through real-world flight experiments.

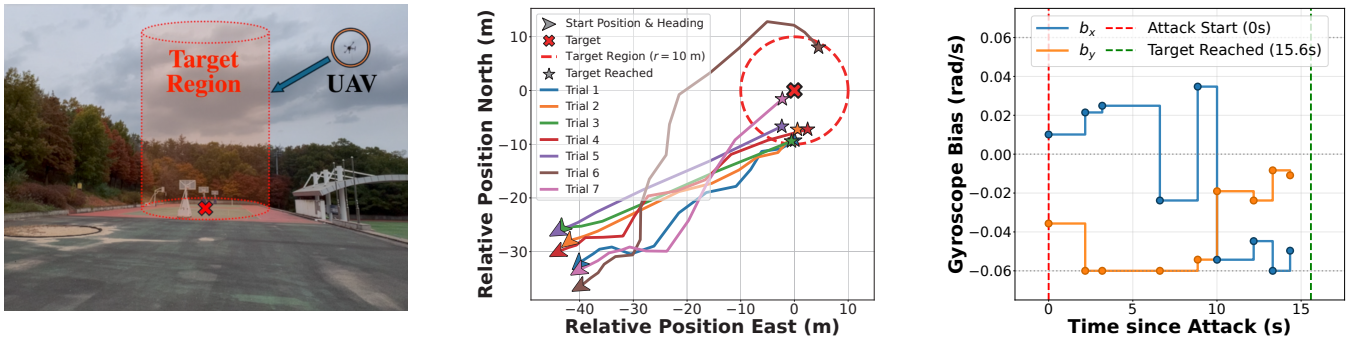
Policy Adaptation. Our threat model (§3.3) assumes that the attacker has access to external tracking equipment that provides ground-truth kinematic states. Although such equipment and supporting technology are available [3, 15, 17], we did not have access to them during hardware testing. Consequently, we trained and adapted a `sim-to-real` policy that relies solely on onboard sensor data.

A key challenge in using onboard sensor data in place of observation is that the UAV's fused state estimates are corrupted by the injected gyroscope bias. To address this, we *reconstruct* observations from raw IMU and GPS readings to approximate the underlying physical states before the fusion algorithm uses them. Specifically:

- **Position and Velocity:** We replace ground-truth values with raw GPS data, deriving horizontal velocity from ground speed and course-over-ground, and vertical velocity from altitude differences across GPS updates.
- **Angular Velocity:** Since the IMU readings include the injected bias, our controller module subtracts the injected bias to reconstruct the true angular rates before feeding them to the policy.
- **Attitude:** Although attitude estimates are affected by the attack-induced estimator drift, no direct raw sensor measurement is available. Thus, we use the fused attitude as an observation despite its gradual corruption. To compensate for the corruption, we augment the actor's partial state vector $\tilde{o}_t$ with linear acceleration and horizontal distance to target. The critic's privileged state vector o_t is similarly extended with 24-dimensional ground truth kinematic states.

The adapted `sim-to-real` policy was trained with over 36K iterations on a 10 m cylindrical objective.

Real-World Experiment Setup. We used Holybro quadcopters with Pixhawk 6C flight controller running PX4: an X500 airframe and an S500 airframe. We conducted a total of seven real-world flight trials across four days to evaluate the attack under varying environmental conditions. In each trial, the UAV hovered at an initial position at 10 m above the



(a) Field experiment photo

(b) Flight trajectories

(c) Injected bias sequence

Figure 8: Real-world flight attack data. A physical UAV running PX4 was attacked in seven independent trials.

ground, and the cylindrical target region was approximately 52 m away, as shown in Figure 8(a) and Figure 8(b). During the attack, the UAV was tethered to a ground anchor with a 50 m safety rope, reaching a maximum displacement of 37 m so the rope remained slack and did not influence the trajectory.

As in simulation, gyroscope bias was injected into the target UAV through a MAVLink message over a telemetry radio link between the UAV and the attacker's ground control station. As detailed in §4.5, in real deployments, this role would be fulfilled by physical injection channels such as acoustic [12, 45] or laser-based [25] methods; MAVLink serves here solely as an interface for controlled experiments.

The field experiment was conducted under dynamic weather conditions (Figure 8(a)), with average wind speeds of 1.7 m/s to 4.4 m/s and gusts up to 7.9 m/s. The onboard GPS provided updates at an average rate of 8 Hz, receiving 26 to 29 satellites during the experiment.

Results. We deployed the `sim-to-real` policy to evaluate our attack on a physical UAV and repeated the experiment seven times, as summarized in Table 6. In *every* trial, the attack successfully redirected the UAV into the 10 m cylindrical objective, following the respective trajectories shown in Figure 8(b). On average, the UAV reached the target in 13.7 s with a trajectory length of 53.6 m.

Figure 8(c) shows bias injection logs from the first trial. The policy initiated with a sustained injection phase lasting 4 s to 10 s, injecting bias of roughly -0.01 rad/s to 0.04 rad/s on the x-axis and -0.06 rad/s on the y-axis. Then, it transitioned to dynamic modulation for trajectory refinement. Although magnitudes vary across trials due to real-world factors, the overall injection patterns were consistent.

Table 6: Summary of seven real-world flight trials.

Trial	Airframe	Init. Dist. (m)	Time (s)	Traj. Len. (m)	Wind (m/s)	Gusts (m/s)
1	X500	51.4	16.0	49.9	1.7 to 2.1	2.8 to 3.1
2	S500	50.5	12.0	48.3	1.9 to 2.8	2.8 to 3.2
3	S500	50.4	8.0	46.0	3.4 to 3.9	4.5 to 6.4
4	S500	53.3	15.0	54.7	3.4 to 4.4	4.7 to 7.9
5	S500	51.3	9.0	46.2	3.4 to 4.4	4.7 to 7.9
6	S500	54.1	22.0	76.5	3.4 to 4.4	4.7 to 7.9
7	S500	52.3	13.9	53.6	3.4 to 4.4	4.7 to 7.9

Robustness Analysis. Our real-world attacks demonstrated robustness against four key real-world factors:

- **Aerodynamic Disturbances:** The experiments were performed over four days with wind speeds ranging 1.7 m/s to 4.4 m/s and gusts up to 7.9 m/s. Our attack policy successfully steered the UAV across all trials, even while the flight controller actively compensated for wind gusts.
- **Airframe Variation:** The Holybro X500 airframe was used in the first trial, whereas the S500, which has a different frame geometry and weight distribution, was used in the other six. The policy succeeded on both airframes without retraining, demonstrating the simulation-based cross-platform results in RQ3 (§7.3).
- **Network Jitter:** Telemetry latency caused irregular injection intervals (Figure 8(c)): across seven flight logs, 87.5 % averaged 1.087 s ($\sigma = 0.076$ s) while 12.5 % were longer due to command losses. The policy nonetheless maintained coherent control. This aligns with the finding in RQ2 (§7.2).
- **Failsafe Resilience:** The GPS enabled the EKF to detect position anomalies, yet no position or velocity failsafe was triggered in any of the flights. The dedicated gyroscope bias failsafe was likewise never activated, again demonstrating resilience to failsafe mechanisms (RQ5, §7.5).

8 Discussion

Our work has a few limitations. First, although our seven real-world flights across four days in §7.6 successfully steered the UAV into the target region, they were performed at distances of approximately 52 m. Validating the policy at longer ranges remains important future work. Second, our attack assumes access to controlled gyroscope-bias injection and external state observation. Although these capabilities lie beyond casual misuse, they are realistic for well-resourced adversaries tar-

geting high-value autonomous UAVs. End-to-end deployment combining injection hardware (*e.g.*, acoustic or laser emitters) with a tracking device would introduce imperfections beyond our simulation, such as residual bias and distance-dependent tracking errors. Examples include residual bias from hardware imprecision and tracking errors that scale with distance and motion. RQ2 (§7.2) provides a first-order assessment by emulating injection-side delays and losses from our real-world flight logs and tracker noise from a published vision tracker [17], under which GAP retained over 87 % success on the 10 m cylinder. A full hardware-in-the-loop characterization remains future work. Third, our policy is optimized for reliably reaching the target region rather than fine-grained terminal behavior, occasionally causing overshoot after arrival. Refining terminal control, such as adjusting bias modulation near the target or incorporating explicit stopping policies, remains future work. Finally, hyperparameter tuning of our RL framework remains future work, as we prioritized demonstrating this previously unexplored attack vector.

Ethics and Disclosure. We disclosed our findings to the PX4 and ArduPilot maintainers and are awaiting their feedback regarding its architectural nature and software-level mitigations.

9 Related Work

Attacks on Reference Sensors. Attacks targeting external reference sensors, such as GPS and magnetometers, have been extensively studied. GPS spoofing [24,31,47] attempts to redirect UAVs by injecting false satellite signals. However, these attacks are difficult to execute; modern sensor fusion algorithms can detect inconsistencies between abrupt GPS shifts and inertial measurements [7,33], and simply injecting spoofing signals triggers receiver-level alarms [16,28,42], which activate failsafes and cause immediate landing or return-to-launch responses, causing hijacking attempts to fail. Similarly, magnetometer false data injection attacks [5] can manipulate heading estimates, but they typically require GPS-denied environments where failsafes are disabled, limiting their operational scope. In contrast, our approach targets the inertial sensors directly, exploiting the fusion architecture to bypass cross-validation mechanisms even when GPS is available.

Attacks on Inertial Sensors. To bypass GPS-layer defenses, recent works have focused on manipulating inertial sensors. Acoustic injection attacks [32,45] exploit MEMS gyroscope resonances to induce drift. Laser-based injection attacks [25] modulate light to induce mechanical vibration that excites MEMS gyroscope resonances, enabling remote signal injection. While stealthy, these physical-layer attacks lack feedback for precise trajectory tracking, resulting in uncontrolled, coarse drifts rather than targeted steering. ConfuSense [10] pioneers RL-driven UAV attacks by using IEMI-based sensor reconfiguration to suspend or slow IMU sampling. Its policy learns when to activate interference (binary timing), leading to coarse directional deviation (*e.g.*, left/right drift). In contrast, GAP preserves the gyroscope stream and injects continuous two-axis bias, learning a closed-loop control policy over bias values. This shifts the attack from discrete disruption to continuous control of the estimation dynamics. As a result, GAP enables precise target-region steering at non-trivial distances by exploiting estimator–controller lag, going beyond coarse directional deviation (§7.1).

Defenses and Evasion. Various physics-based detectors [6, 13, 29] identify sensor attacks by monitoring violations in control-theoretic relationships between control commands and sensor responses. However, recent studies have shown that these defenses rely on static thresholds or fixed invariant models [9]. Our evaluation shows the attack achieves a 76 % evasion rate against threshold-based defenses (*e.g.*, CI [6]), not by learning evasion strategies, but because the bias magnitude, constrained to bypass the EKF's internal failsafe, inherently falls within the detection blind spots of such detectors. Recovery-oriented defenses such as SpecGuard [8] achieve strong recovery success across a broad range of physical attacks by using RL to keep UAVs compliant with mission specifications, running Proactive Control (PC) throughout the flight and engaging Reactive Control (RC) after attack detection. However, two factors suggest GAP would likely remain effective against both. First, GAP keeps bias below PX4's gyroscope failsafe and triggers other failsafes or the CI detector in only about 20 % of runs (§7.4, §7.5), so RC is unlikely to engage in most executions. Second, the publicly available SpecGuard implementation models gyroscope attacks by modifying EKF attitude estimates instead of sensor-driver-level readings. As a result, it evaluates only the Slow Path effect (§3.1) of gyroscope spoofing and not the Fast Path that GAP exploits. We leave empirical validation of GAP against SpecGuard-PC/RC in this setting as future work.

10 Conclusion

We identify an architectural vulnerability in the UAV control pipeline: high-rate control loops operating atop low-rate, indirectly fused state estimates. This temporal mismatch opens an exploitable window in which carefully crafted gyroscope bias injections can steer a UAV to an arbitrary target region, while remaining undetected by a state-of-the-art detector and being robust to built-in failsafes. Leveraging this insight, we developed a reinforcement learning-based black-box attack policy that exploits this window under assumed bias-injection and external observation capabilities. Our trained policy, GAP, achieved 85 % success in target-region steering and bypassed a detector in 76 % of successful attack cases. We validated its real-world feasibility, successfully steering a physical UAV into the target region under varying winds and airframes.

11 Acknowledgment

We would like to thank our shepherd and the anonymous reviewers for their insightful comments and constructive feedback that helped improve this paper. We are also grateful to Jangseop Choi, Junwoong Doh, Chiheon Kim, and Minki Lee from the POSTECH CompSec Lab for their field setup assistance with the physical drone experiments. This research was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2025-23523174), and the Institute of Information & Communications Technology Planning & Evaluation (IITP) - ITRC (Information Technology Research Center) grant funded by the Korea government (MSIT) (IITP-2026-RS-2024-00437866 and IITP-2026-RS-2026-25529760).

References

[1] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[2] Nourdine Aliane. A survey of open-source uav autopilots. *Electronics*, 13(23):4785, 2024.

[3] Nancy Alshaer, Reham Abdelfatah, Tawfik Ismail, and Haitham Mahmoud. Vision-Based UAV Detection and Tracking Using Deep Learning and Kalman Filter. *Computational Intelligence*, 41(1):e70026, 2025.

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[5] Wenxin Chen, Yingfei Dong, and Zhenhai Duan. Manipulating drone dynamic state estimation to compromise navigation. In *2018 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9. IEEE, 2018.

[6] Hongjun Choi, Wen-Chuan Lee, Yousra Aafer, Fan Fei, Zhan Tu, Xiangyu Zhang, Dongyan Xu, and Xinyan Deng. Detecting attacks against robotic vehicles: A control invariant approach. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 801–816, 2018.

[7] Sagar Dasgupta, Mizanur Rahman, Mhafuzul Islam, and Mashrur Chowdhury. A sensor fusion-based GNSS spoofing attack detection framework for autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):23559–23572, 2022.

[8] Pritam Dash, Ethan Chan, and Karthik Pattabiraman. Specguard: Specification aware recovery for robotic autonomous vehicles from physical attacks. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 1849–1863, 2024.

[9] Pritam Dash, Mehdi Karimibiuki, and Karthik Pattabiraman. Stealthy attacks against robotic vehicles protected by control-based intrusion detection techniques. *Digital Threats: Research and Practice*, 2(1):1–25, 2021.

[10] Alessandro Erba, John H Castellanos, Sahil Sihag, Saman Zonouz, and Nils Ole Tippenhauer. ConfuSense: Sensor Reconfiguration Attacks for Stealthy UAV Manipulation. In *3rd USENIX Symposium on Vehicle Security and Privacy (VehicleSec 25)*, pages 45–62, 2025.

[11] Center for Strategic and International Studies. How Ukraine's Operation Spider's Web Redefines Asymmetric Warfare, June 2025. https://www.csis.org/analysis/how-ukraines-spider-web-operation-redefines-asymmetric-warfare. Accessed: 2025-11-11.

[12] Ming Gao, Lingfeng Zhang, Leming Shen, Xiang Zou, Jinsong Han, Feng Lin, and Kui Ren. Exploring practical acoustic transduction attacks on inertial sensors in MDOF systems. *IEEE Transactions on Mobile Computing*, 23(5):3539–3557, 2023.

[13] Jairo Giraldo, David Urbina, Alvaro Cardenas, Junia Valente, Mustafa Faisal, Justin Ruths, Nils Ole Tippenhauer, Henrik Sandberg, and Richard Candell. A survey of physics-based attack detection in cyber-physical systems. *ACM Computing Surveys (CSUR)*, 51(4):1–36, 2018.

[14] Matthew J Hausknecht and Peter Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. In *AAAI fall symposia*, volume 45, page 141, 2015.

[15] Brian KS Isaac-Medina, Matt Poyser, Daniel Organisciak, Chris G Willcocks, Toby P Breckon, and Hubert PH Shum. Unmanned aerial vehicle visual detection and tracking using deep neural networks: A performance benchmark. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1223–1232, 2021.

[16] Andrew J Kerns, Daniel P Shepard, Jahshan A Bhatti, and Todd E Humphreys. Unmanned aircraft capture and control via GPS spoofing. *Journal of field robotics*, 31(4):617–636, 2014.

[17] Chujun Li, Yiyang Wu, Sheng Zhuge, Xia Yang, Bin Lin, Xiangpeng Xu, and Xiaohu Zhang. Robust 6D

Pose Estimation of the UAV Based on Hybrid Features. *IEEE Transactions on Instrumentation and Measurement*, 2024.

[18] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In *International conference on machine learning*, pages 3053–3062. PMLR, 2018.

[19] Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics*, 36(1):1–14, 2019.

[20] Michael Luo, Jiahao Yao, Richard Liaw, Eric Liang, and Ion Stoica. Impact: Importance weighted asynchronous architectures with clipped target networks. *arXiv preprint arXiv:1912.00167*, 2019.

[21] ModalAI, Inc. ModalAI Launches VOXL 2, the Smallest, Smartest Blue UAS Framework Autopilot, 2022. https://www.businesswire.com/news/home/20220425005133/en/. Accessed: 2025-12-05.

[22] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50, 2020.

[23] Shoei Nashimoto, Daisuke Suzuki, Takeshi Sugawara, and Kazuo Sakiyama. Sensor con-fusion: Defeating kalman filter in signal injection attack. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 511–524, 2018.

[24] Juhwan Noh, Yujin Kwon, Yunmok Son, Hocheol Shin, Dohyun Kim, Jaeyeong Choi, and Yongdae Kim. Tractor beam: Safe-hijacking of consumer drones with adaptive GPS spoofing. *ACM Transactions on Privacy and Security (TOPS)*, 22(2):1–26, 2019.

[25] Kunihiko OI and Takeshi SUGAWARA. Signal Injection Attack on Inertial Measurement Unit using Continuous-Wave Laser. *IEICE Transactions on Information and Systems*, 2025.

[26] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[27] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017.

[28] Mark L Psiaki and Todd E Humphreys. GNSS spoofing and detection. *Proceedings of the IEEE*, 104(6):1258–1270, 2016.

[29] Raul Quinonez, Jairo Giraldo, Luis Salazar, Erick Bauman, Alvaro Cardenas, and Zhiqiang Lin. SAVIOR: Securing autonomous vehicles with robust physical invariants. In *29th USENIX security symposium (USENIX Security 20)*, pages 895–912, 2020.

[30] Maria Isabel Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties. *Institute for Systems and Robotics*, 43(46):3736–3741, 2004.

[31] Harshad Sathaye, Martin Strohmeier, Vincent Lenders, and Aanjhan Ranganathan. An experimental study of GPS spoofing and takeover attacks on UAVs. In *31st USENIX security symposium (USENIX security 22)*, pages 3503–3520, 2022.

[32] Yunmok Son, Hocheol Shin, Dongkwan Kim, Youngseok Park, Juhwan Noh, Kibum Choi, Jungwoo Choi, and Yongdae Kim. Rocking drones with intentional sound noise on gyroscopic sensors. In *24th USENIX security symposium (USENIX Security 15)*, pages 881–896, 2015.

[33] Çağatay Tanıl, Samer Khanafseh, Mathieu Joerger, and Boris Pervan. Kalman filter-based INS monitor to detect GNSS spoofers capable of tracking aircraft position. In *2016 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pages 1027–1034. IEEE, 2016.

[34] ArduPilot Development Team. Code Overview (Copter), 2025. https://ardupilot.org/dev/docs/apmcopter-code-overview.html. Accessed: 2025-12-05.

[35] ArduPilot Development Team. Crop Sprayer, 2025. https://ardupilot.org/copter/docs/common-sprayer.html. Accessed: 2026-05-12.

[36] ArduPilot Development Team. ArduPilot Documentation, 2026. https://ardupilot.org. Accessed: 2026-04-27.

[37] ArduPilot Development Team. Identification of a Multicopter, 2026. https://ardupilot.org/copter/docs/systemid-model-development.html. Accessed: 2026-04-27.

[38] PX4 Development Team. PX4 Architectural Overview, 2025. https://docs.px4.io/main/en/concept/architecture. Accessed: 2025-12-05.

[39] PX4 Development Team. PX4 Autopilot Documentation, 2025. https://docs.px4.io/main/en/. Accessed: 2025-11-29.

[40] PX4 Development Team. PX4 Commercial Systems, 2025. https://px4.io/ecosystem/commercial-systems/. Accessed: 2025-12-05.

[41] PX4 Development Team. PX4 Controller Diagrams, 2026. https://docs.px4.io/main/en/flight_stack/controller_diagrams. Accessed: 2026-04-27.

[42] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. On the requirements for successful GPS spoofing attacks. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 75–86, 2011.

[43] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[44] Timothy Trippel, Ofir Weisse, Wenyuan Xu, Peter Honeyman, and Kevin Fu. WALNUT: Waging doubt on the integrity of MEMS accelerometers with acoustic injection attacks. In *2017 IEEE European symposium on security and privacy (EuroS&P)*, pages 3–18. IEEE, 2017.

[45] Yazhou Tu, Zhiqiang Lin, Insup Lee, and Xiali Hei. Injected and delivered: Fabricating implicit control over actuation systems by spoofing inertial sensors. In *27th USENIX security symposium (USENIX Security 18)*, pages 1545–1562, 2018.

[46] Zhanghao Wu, Eric Liang, Michael Luo, Sven Mika, Joseph E Gonzalez, and Ion Stoica. RLlib flow: Distributed reinforcement learning is a dataflow problem. In *Conference on Neural Information Processing Systems (NeurIPS)*, page 54, 2021.

[47] Kexiong Curtis Zeng, Shinan Liu, Yuanchao Shu, Dong Wang, Haoyu Li, Yanzhi Dou, Gang Wang, and Yaling Yang. All your GPS are belong to us: Towards stealthy manipulation of road navigation systems. In *27th USENIX security symposium (USENIX security 18)*, pages 1527–1544, 2018.

USENIX VehicleSec '26 Artifact Appendix: Blind Spot in Sensor Fusion: Exploiting an Architectural Vulnerability to Hijack and Precisely Control UAVs

Jongsoo Han
Pohang University of Science and Technology
hanjs@postech.ac.kr

Seulbae Kim
Pohang University of Science and Technology
seulbae@postech.ac.kr

A Artifact Appendix

A.1 Abstract

This artifact accompanies the paper *Blind Spot in Sensor Fusion: Exploiting an Architectural Vulnerability to Hijack and Precisely Control UAVs*. It contains patched PX4/ArduPilot firmware, trained GAP policies, experiment wrappers, analysis scripts, and pre-baked outputs. It supports six claims on attack effectiveness, robustness to tracking noise and delay/loss, cross-platform transfer, CI-detector evasion, failsafe analysis, and sim-to-real feasibility. The recommended evaluator path uses the prepared VM or archived source snapshot to run the smoke test and verify the shipped pre-baked claims. Optional fresh reruns are provided as additional experiments.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The normal evaluation path is software-only. It uses either shipped pre-baked logs or software-in-the-loop evaluation, and does not require disabling host security features. The optional CI-detector rerun uses a legacy VMware image that should be treated as an isolated guest environment. The optional real-flight rerun uses a physical UAV and should only be attempted with appropriate permissions, a safe outdoor flight site, and independent flight-safety procedures.

A.2.2 How to access

The stable archival copy of the artifact is available through Zenodo: https://doi.org/10.5281/zenodo.19652756. The archive provides two entry points: (i) a prepared Ubuntu 22.04 VirtualBox image with the repository, Python environment, built firmware, QGroundControl, policies, and pre-baked outputs already installed, and (ii) a versioned source snapshot that can be built from scratch with `./setup.sh`. The Zenodo bundle also includes the trained model archive, the optional legacy VMware image for CI-detector reruns, and the QGroundControl AppImage for source-based users.

A.2.3 Badges

This artifact was awarded the following badges:

- **Artifacts Available**
- **Artifacts Functional**
- **Results Reproduced**

A.2.4 Hardware dependencies

The main path requires a recent `x86_64` host with 12 CPU cores, 16GB RAM, and no GPU. A 100GB disk is recommended, as `./setup.sh` increased disk usage by about 30GB on our Ubuntu 22.04 guest. Optional physical reruns require a multicopter, Pixhawk, telemetry radios, and a safe flight site.

A.2.5 Software dependencies

The prepared VM already includes all dependencies. Source-based users need Ubuntu 22.04, Python 3.10, Java for jMAVSim, Ray/RLlib, NumPy, pandas, matplotlib, scipy, pyulog, geographiclib, the patched PX4 and ArduPilot source code, and the custom MAVLink support installed by `./setup.sh`. VirtualBox is needed for the provided VM, while VMware and QGroundControl are needed only for the optional CI-detector and real-flight workflows, respectively.

A.2.6 Benchmarks

The artifact evaluates the shipped trained GAP and sim-to-real policies and the following result sets under `results/pre-baked/`:

- shipped pre-baked results
- PX4 attack-flight CSVs under `flight-logs/`
- real-flight ULog files under `rq6/`

A.3 Set-up

A.3.1 Installation

Path A is recommended: download the prepared GAP VM from Zenodo, import it into VirtualBox, log in with user/password `gap`/`gap`, and open a terminal at `$HOME/GAP`. All

dependencies, built firmware, trained policies, and QGroundControl are already installed.

Path B uses the archived source snapshot or GitHub repository. From `$HOME/GAP`, run:

```
./setup.sh
source .venv/bin/activate
```

The setup script installs dependencies, compiles patched PX4 and ArduPilot, creates a local Python environment, and took about 30 minutes on our 12-core Ubuntu 22.04 host. If starting from GitHub rather than the Zenodo snapshot, download `gap-models.zip` from Zenodo and unpack it into `src/gap/models/` as documented in the repository README.

A.3.2 Basic Test

Run:

```
./experiments/run_smoke_test.sh
```

Expected outcome: `[PASS] All 11 checks passed`. This test checks the environment and verifies the presence and consistency of the shipped data without running fresh experiments.

A.4 Evaluation Workflow

A.4.1 Major claims

(C1) GAP outperforms the three baselines in PX4-jMAVSim and reaches the 10 m cylindrical target region with an 85.1 % success rate.

(C2) GAP remains effective under tracking noise, injection delay/loss, and their combination.

(C3) The trained policy transfers without retraining to PX4 Gazebo and nine ArduPilot airframes.

(C4) GAP bypasses the CI detector in most successful attacks. In the shipped RQ4 set, 21 of 24 attacks succeed and only 5 of the successful attacks trigger the detector.

(C5) Built-in failsafes do not reliably stop the attack. In the shipped PX4 attack-flight corpus, the gyroscope-bias failsafe never triggers, while generic failsafes appear in 213 of 1,048 successful attacks.

(C6) The sim-to-real policy succeeds in all seven shipped real-world flight trials, reaching the 10 m cylindrical target in 13.794 s on average with a mean trajectory length of 53.576 m.

A.4.2 Experiments

(E0) Shipped reference path [5 human-minutes + 10 compute-minutes]. Run:

```
bash analysis/verify_claims.sh pre-baked
bash analysis/generate_all.sh pre-baked
```

This verifies the claims from the pre-baked data, regenerates all shipped tables and figures, and supports C1-C6.

(E1) RQ4 CI-detector analysis [5 human-minutes + 5 compute-minutes]. Run:

```
python3 -m analysis.generate_rq4_analysis \
--source pre-baked
```

This regenerates the RQ4 CI-detector evasion CSV and figure from shipped JSON logs and supports C4.

(E2) RQ5 failsafe analysis [5 human-minutes + 5 compute-minutes]. Run:

```
python3 -m analysis.generate_rq5_analysis \
--source pre-baked
```

This post-processes the shipped PX4 attack-flight corpus to compute failsafe activation statistics and supports C5.

(E3) RQ6 real-flight analysis [5 human-minutes + 5 compute-minutes]. Run:

```
python3 -m analysis.generate_rq6_table6 \
--source pre-baked
python3 -m analysis.generate_rq6_figure8 \
--source pre-baked
```

This regenerates Table 6 and Figure 8 from the shipped real-flight ULog files and supports C6.

For all experiments, expected outputs are CSV files under `analysis/csv/` and figures under `analysis/figures/`.

A.5 Additional Experiments - Fresh Runs

Fresh reruns of RQ1-RQ3 are provided as additional experiments for exercising the simulator execution path. They are stochastic and are intended as directional validation, while the shipped pre-baked data remains the exact paper result set. The one-command fresh path is:

```
./experiments/run_all.sh --mode approx
```

E4 reruns RQ1 and regenerates Table 3/Figure 7. E5 reruns RQ2 and regenerates Table 4. E6 reruns RQ3 and regenerates Table 5. Detailed commands and optional CI-detector and real-flight reruns are documented in the artifact repository.

A.6 Notes on Reusability

The repository separates reusable evaluation code under `src/evaluation/` from thin experiment wrappers under `experiments/`. The same analysis scripts can be run on shipped pre-baked data or user-generated fresh data, and outputs are source-tagged to avoid overwriting one another. The artifact covers evaluation of the shipped trained policies. The training pipeline is out of scope.

A.7 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.

ARTIFACT EVALUATED USENIX AVAILABLE
ARTIFACT EVALUATED USENIX FUNCTIONAL

SHIELD: Secure Human-Machine Interaction with Evidential Learning and Dynamic Trust for Drone Swarm Control

Dawood Wasif[1] *Terrence J. Moore*[2] *Seunghyun Yoon*[3] *Hyuk Lim*[3]
Dan Dongseong Kim[4] *Frederica F. Nelson*[2] *Jin-Hee Cho*[1]

[1]*Virginia Tech*
[2]*US DEVCOM Army Research Laboratory*
[3]*Korea Institute of Energy Technology (KENTECH)*
[4]*The University of Queensland*

Abstract

Drone swarms face a dual challenge: human operators cannot effectively supervise more than a dozen Unmanned Aerial Vehicles (UAVs) before situational awareness degrades, yet fully autonomous coordination remains vulnerable to cyber-physical attacks such as GPS spoofing, communication jamming, Byzantine manipulation, replay, malware injection, and network intrusion. We present SHIELD, a framework for secure human–swarm teaming built on a single unifying insight: *evidential vacuity*, derived from Dirichlet-based perception, jointly drives inter-agent trust modulation, intrusion detection, and human escalation within a decentralized control loop. A dynamic trust graph filters unreliable agents through behavioral consistency and quarantine, while an embedded intrusion response system autonomously handles detection, containment, and recovery. Operator involvement is reserved for genuinely ambiguous cases via an event-triggered binary interface that minimizes workload. In the considered waypoint-surveillance simulation scenarios with six attack classes and swarms of 8–32 UAVs, SHIELD achieves up to 60% relative improvement in mission success, detects over 80% of compromised agents with under 15% false positives, and reduces human interventions by 53% compared to baselines. Ablation and scalability studies confirm complementary module contributions and graceful performance degradation.

1 Introduction

Human Scalability and Adversarial Vulnerability. Autonomous drone swarms enable large-scale coverage in defense, surveillance, and search-and-rescue by coordinating dozens or hundreds of Unmanned Aerial Vehicles (UAVs) [8]. However, human operators cannot effectively supervise more than 10–12 UAVs before situational awareness and decision quality degrade [9]. At the same time, fully autonomous swarms remain vulnerable to cyber-physical attacks such as GPS spoofing, communication jamming, Byzantine manipulation, malware injection, and network intrusion [3]. These twin constraints of *cognitive scalability* and *adversarial resilience* reveal a fundamental gap: swarm autonomy must reduce operator workload while remaining robust to compromised agents and uncertain environments.

Limits of Existing Human–Swarm Frameworks. Human-in-the-loop and shared-autonomy methods improve learning efficiency and workload distribution [27, 36, 43], yet they largely assume benign settings and treat agent outputs as equally reliable. Most lack calibrated uncertainty estimation, decentralized trust modeling, and mechanisms to isolate compromised UAVs. Decision authority is often centralized, creating single points of failure and requiring frequent low-level interventions that undermine scalability [46]. Consequently, these systems neither detect adversarial manipulation nor degrade gracefully under uncertainty.

Fragmented Multi-Agent Resilience Approaches. Recent resilience methods address isolated aspects of the problem. Trust-adaptive control isolates non-cooperative agents but assumes centralized observers and omits uncertainty quantification and human oversight [22, 48]. Authority-allocation frameworks formalize intervention policies but target single-vehicle settings and do not scale to decentralized swarm settings [42]. Existing communication-robustness defenses neglect the human–machine teaming dimension. Critically, no prior work provides a unified uncertainty signal that governs trust modulation, intrusion detection, and operator escalation within a decentralized control loop.

Our Approach. We address these limitations with SHIELD (Secure Human-machine Interaction with Evidential Learning and Dynamic trust), a unified framework for scalable and adversary-resilient human–swarm teaming. SHIELD integrates evidential deep learning for calibrated uncertainty estimation [40] with dynamic trust modeling for decentralized, peer-weighted coordination [38]. Its central contribution is the use of a single uncertainty quantity, *evidential vacuity*, derived from Dirichlet-based perception, to jointly govern three critical functions: inter-agent trust adaptation, intrusion detection and quarantine, and event-triggered human escalation. By embedding uncertainty-aware perception, trust-weighted

consensus, and divergence-based safety overrides within one decentralized control loop, SHIELD eliminates single points of failure, isolates compromised UAVs in real time, and reduces unnecessary operator interventions. Unlike prior approaches that treat trust, anomaly detection, and human oversight as separate modules, SHIELD unifies them through a principled uncertainty signal, enabling robust performance under both operational uncertainty and cyber-physical attack.

Contributions. This work makes three primary contributions:

First, we introduce a **unified uncertainty-driven architecture** for secure human–swarm teaming. SHIELD leverages a single evidential vacuity signal, derived from Dirichlet-based perception, to jointly govern inter-agent trust adaptation, intrusion detection, and event-triggered human escalation within a decentralized control loop. By consolidating trust modeling, anomaly detection, and authority allocation through one calibrated uncertainty signal [22, 42, 48], the framework reduces system complexity while strengthening adversarial resilience.

Second, we develop a **dynamic trust graph with real-time quarantine** that enables decentralized, trust-weighted consensus and isolates compromised UAVs without centralized supervision. Trust updates are driven by calibrated uncertainty and behavioral consistency, enabling context-aware adaptation and graceful degradation under adversarial disruption.

Third, we provide **comprehensive adversarial evaluation** across six cyber-physical attack classes. In waypoint-surveillance simulations, SHIELD achieves up to 60% relative improvement in mission success, higher survival, and over 50% reduction in human interventions compared to baselines. The embedded intrusion response system detects over 80% of compromised agents with under 15% false positives and restores safe operation within 10 seconds.

2 Related Work

This section discusses prior work on reinforcement learning with human feedback, human–swarm interaction, multi-agent trust modeling, and cyber-physical UAV security, highlighting their limitations in achieving unified, uncertainty-aware, and adversary-resilient human–swarm coordination.

2.1 RL with Human Feedback (RLHF)

Human feedback (HF) has been widely incorporated into reinforcement learning (RL) to improve sample efficiency and safety. Deep TAMER [43], DQfD [20], and preference-based methods [4, 5, 6, 26, 31] leverage demonstrations, scalar rewards, or comparisons to guide policy learning. More recent work integrates meta-learning and uncertainty-aware reward modeling [19, 34]. However, these approaches largely assume single-agent settings and rely on frequent human input. They do not address decentralized coordination, adversarial compromise, or selective escalation in multi-UAV systems.

2.2 Human–Drone Teaming

Research on human–swarm interaction focuses on interface design and mixed-initiative control, including gesture-based control [1, 29], AR and haptic feedback [2, 49], and high-level strategic guidance [25, 30]. These systems improve situational awareness and workload distribution [12, 44], yet typically assume benign environments. They lack principled uncertainty quantification and do not provide mechanisms to determine when autonomous decisions should be trusted, overridden, or escalated under adversarial conditions.

2.3 Trust and Resilient Multi-Agent Systems

Trust-based coordination improves robustness in multi-agent systems. Behavioral-consistency updates [17, 38] and trust-weighted consensus enhance fault tolerance, while blockchain and reflective-control approaches strengthen reputation tracking and adaptation [18, 45]. Evidential reasoning has been explored for secure state inference [16], but prior work does not integrate calibrated uncertainty, decentralized trust adaptation, and human oversight within a unified control framework. As a result, trust modeling remains disconnected from intrusion detection and operator escalation.

2.4 Cyber-Physical Security for Swarms

Defenses against cyber-physical attacks on UAVs include sensor reconfiguration detection [15], protection against false localization attacks [13], and runtime behavior monitoring [10]. In multi-agent communication, active defense [47], Bayesian-game formulations [33], graph bottleneck filtering [11], and trust-aware communication [41] strengthen message reliability. However, these approaches operate as standalone detection or filtering modules: they identify anomalies or harden message channels but do not close the loop by feeding detection signals into trust adaptation, autonomous quarantine, or calibrated human escalation in a unified control architecture.

In contrast, SHIELD unifies calibrated uncertainty estimation, decentralized trust adaptation, intrusion response, and event-triggered human oversight within a single control architecture, directly addressing the scalability and adversarial gaps identified above.

3 System Model, Threats, and Operational Hazards

This section formalizes the system architecture, security assumptions, adversary capabilities, attack surface, and non-adversarial hazards considered in our evaluation.

3.1 System and Security Assumptions

We consider a cooperative drone swarm of N UAVs coordinated by a single human operator over a shared wireless communication network. Each UAV executes a local control policy based on onboard sensors (depth cameras, GPS, IMU) and peer-to-peer state messages exchanged with neighboring agents. The operator issues high-level mission objectives and may intervene when prompted.

The swarm operates in a bounded three-dimensional arena. Each UAV i maintains a local state estimate $\mathbf{s}_i(t)$ derived from its own sensors and messages received from neighbors $\mathcal{N}_i(t)$. Communication links are bidirectional and constrained by range and line-of-sight. The operator communicates with the swarm through a trusted and authenticated command channel.

We adopt the following security assumptions (S#), which define the boundaries of our evaluation and the guarantees under which the framework operates:

S1. The operator's command link is secure, authenticated, and cannot be spoofed or intercepted by the adversary.

S2. At most $f < \lfloor N/3 \rfloor$ UAVs are compromised at any time, ensuring that an honest majority remains for meaningful consensus and trust aggregation. This is a standard Byzantine fault tolerance prerequisite [32], not an assumption unique to this work; for our default $N = 8$, this permits $f = 2$ compromised agents.

S3. Sensor noise under benign conditions is bounded and follows known distributions (Section 5), enabling statistical deviation detection.

S4. UAVs do not share cryptographic secrets for inter-agent authentication; message integrity relies on behavioral consistency rather than full PKI (Public Key Infrastructure) guarantees.

S1 reflects standard military command-and-control practice (dedicated encrypted links), while **S4** reflects the computational constraints of lightweight UAV platforms that motivate behavior-based trust as an alternative to full PKI.

3.2 Adversary Model

Capabilities. The adversary may compromise UAVs or their communication channels. Specifically, the attacker can: inject false sensor readings (e.g., spoofed GPS); drop, delay, or jam inter-agent messages; cause compromised UAVs to broadcast arbitrary or inconsistent state information (Byzantine behavior); replay stale sensor data; deploy malicious firmware through supply-chain or update exploitation; or gain unauthorized access to the swarm's communication infrastructure to inject or intercept packets. The adversary may operate externally (RF injection, network manipulation) or from within the swarm's trust boundary.

Goals. The adversary seeks to degrade swarm integrity or performance along measurable dimensions evaluated in Section 5.3: (i) *mission disruption*, reflected by reduced Mission Success Rate (MSR) or Survival Rate (SR); (ii) *collision induction*, captured by worsened Crash Penalty (CP); (iii) *operator overload*, indicated by increased Human Intervention Count (HIC); (iv) *trust erosion*, observable through elevated state disagreement across agents and degraded collective task performance; and (v) *persistent stealth*, characterized by sustained performance degradation without corresponding detection signals, i.e., low Detection Rate (DR) and elevated False Positive Rate (FPR) that mask true compromise..

Knowledge Model. We consider two levels of adversary knowledge. Under a *black-box* model, the attacker knows the general architecture but not internal thresholds, trust weights, or monitoring variables. Under a *grey-box* model, the attacker knows the existence of uncertainty-based monitoring and trust-weighted consensus but cannot observe real-time internal states of uncompromised agents. Critically, internal defense variables—trust weights τ_{ij}, vacuity scores u_i, and override decisions—are computed and stored locally on each UAV and are *not* included in inter-agent broadcast messages $(\mathbf{s}_i, b_i, p_i)$. A compromised agent can therefore observe its own defense variables and the broadcast content of neighbors, but cannot directly read the trust scores or escalation states maintained by honest peers. This limits the adversary's ability to craft inputs that precisely manipulate another agent's trust or escalation logic. Experiments primarily evaluate the black-box setting; grey-box implications, including adaptive evasion strategies, are discussed in Section 7.

3.3 Attack Surface and Classes

The attack surface spans perception, communication, coordination, software integrity, and network layers. We instantiate six attack classes grounded in the UAV security literature [3, 8], covering both external and internal compromise.

A1: GPS Spoofing. False GPS signals corrupt the position estimate of targeted UAVs:

$$\hat{\mathbf{s}}_i(t) = \mathbf{s}_i(t) + \boldsymbol{\Delta}, \quad \boldsymbol{\Delta} \sim \mathcal{U}(-\delta_{\max}, \delta_{\max})^3. \tag{1}$$

Spoofing occurs periodically (approximately every 20 s) for 5 s, targeting the perception layer and potentially inducing collisions or formation breakdown.

A2: Communication Jamming. During 3–7 s windows, messages between affected agents are independently dropped with probability 0.5:

$$\mathcal{M}_{ij}(t) = \begin{cases} \emptyset & \text{with probability } 0.5, \\ \mathcal{M}_{ij}(t) & \text{otherwise.} \end{cases} \tag{2}$$

This degrades coordination and consensus at the communication layer, leading to incomplete neighborhood information.

A3: Byzantine Faults. Compromised UAVs broadcast fabri-

cated state information with probability $p = 0.10$:

$$\mathcal{M}_{ij}^{\text{Byz}}(t) = \begin{cases} \text{sample}(\mathcal{D}_{\text{states}}) & \text{with probability } p, \\ \mathcal{M}_{ij}(t) & \text{otherwise.} \end{cases} \quad (3)$$

This targets the coordination layer by introducing inconsistent state estimates that can fragment collective decision-making.

A4: Replay Attacks. Outdated sensor readings are re-injected into the perception pipeline:

$$x_i^{\text{replay}}(t) = x_i(t - \Delta t_r), \quad \Delta t_r = 10\,\text{s}. \quad (4)$$

Replay undermines temporal consistency in dynamic environments, causing decisions based on stale state information.

A5: Malware Injection. Compromised UAVs execute altered policies:

$$\pi_i^{\text{mal}}(a \mid s) \neq \pi_i(a \mid s), \quad (5)$$

with intermittent or state-dependent divergence. This targets software integrity and is hard to detect because compromised agents retain valid credentials and communication access.

A6: Network Intrusion. An adversary gains unauthorized access to the swarm's communication infrastructure, enabling packet injection or eavesdropping from within the network. Unlike jamming, this attack operates inside the communication channel and targets network integrity, potentially allowing coordinated manipulation across multiple agents.

Parameter Justification. Attack parameters are calibrated relative to the simulator's sensor noise floor and grounded in UAV security literature. GPS spoofing magnitude $\delta_{\max} = 3.0\,\text{m}$ corresponds to $\sim 10\sigma$ of benign GPS noise ($\sigma_{\text{GPS}} = 0.3\,\text{m}$), within the 1–5 m civilian spoofing range reported in field studies [24, 28]. Byzantine probability $p = 0.10$ models intermittent deception below trivially detectable thresholds [32]. Replay lag $\Delta t_r = 10\,\text{s}$ (500 control steps at 50 Hz) produces meaningful state divergence within commodity buffer limits [21]. Malware drift $\|\Delta\pi\| \leq 0.15$ balances operational impact against immediate detectability. Jamming drop probability (0.5) over 3–7 s windows reflects partial denial under power-limited RF contestation [35].

3.4 Non-Adversarial Operational Hazards

In addition to adversarial threats, the swarm must handle environmental uncertainties that may resemble attacks.

H1: Obstacle Encounters. Dynamic or previously unmapped obstacles may require rapid replanning. Legitimate evasive maneuvers must not be misclassified as compromise.

H2: Airspace Congestion. High agent density can saturate communication bandwidth and degrade coordination, resembling jamming or Byzantine faults at the behavioral level.

H3: Emergency Conditions. Environmental disturbances (e.g., wind gusts), sensor failures, or low battery events may cause deviations that trigger false alarms if not distinguished from adversarial manipulation.

These hazards require a defense capable of joint attribution across security and environmental states, minimizing both missed detections and excessive false positives.

3.5 Scope and Exclusions

The threat model focuses on cyber-physical attacks and operational hazards within the swarm's communication and control stack. We explicitly exclude: (i) physical capture, hardware tampering, or side-channel extraction of UAV components; (ii) compromise of the operator's authenticated command channel; and (iii) coordinated multi-agent deception campaigns that violate the fault bound $f < \lfloor N/3 \rfloor$.

The current evaluation considers parameterized, non-adaptive attacks. A more capable adversary could adopt an *intermittent Byzantine* strategy, reverting to honest behavior upon sensing impending quarantine and resuming after reintegration. SHIELD's reintegration protocol partially mitigates this by requiring m consecutive consistent steps before restoring trust from τ_{init}, imposing a minimum honest-operation cost per evasion cycle. Each cycle forces the adversary to behave honestly for at least $m \cdot \Delta t$ seconds during which it cannot advance its attack objective, and trust is restored only to τ_{init} rather than its pre-quarantine level, bounding the number of feasible evasion cycles per episode. The gap between detection without IRS and with full IRS (Table 2) confirms that layered defense substantially narrows the stealth window, though systematic evaluation of adaptive and RL-based adversaries is deferred to future work.

The instantiated attacks (A1–A6) and operational hazards (H1–H3) collectively span perception, communication, coordination, software integrity, and network layers. Within this scope, the defense must quantify perceptual uncertainty, evaluate peer reliability, attribute anomalies across security and environmental causes, and enforce safe fallback behavior. The next section introduces a framework designed to meet these requirements.

4 Proposed Approach: SHIELD

SHIELD enables scalable and trustworthy human–swarm teaming under uncertainty and cyber-physical threats. As illustrated in Figure 1, each UAV performs uncertainty-aware perception, coordinates with neighbors through a trust-weighted shared mental model, arbitrates decisions via consensus and safety overrides, and manages compromised agents through quarantine and reintegration mechanisms.

We model the human–swarm team as a time-varying, trust-weighted graph:

$$\mathcal{G}(t) = (V, E(t), \tau(t)), \quad (6)$$
$$V = \{0, 1, 2, \ldots, N\}, \quad E(t) \subseteq V \times V, \quad \tau_{ij}(t) \in [0, 1].$$

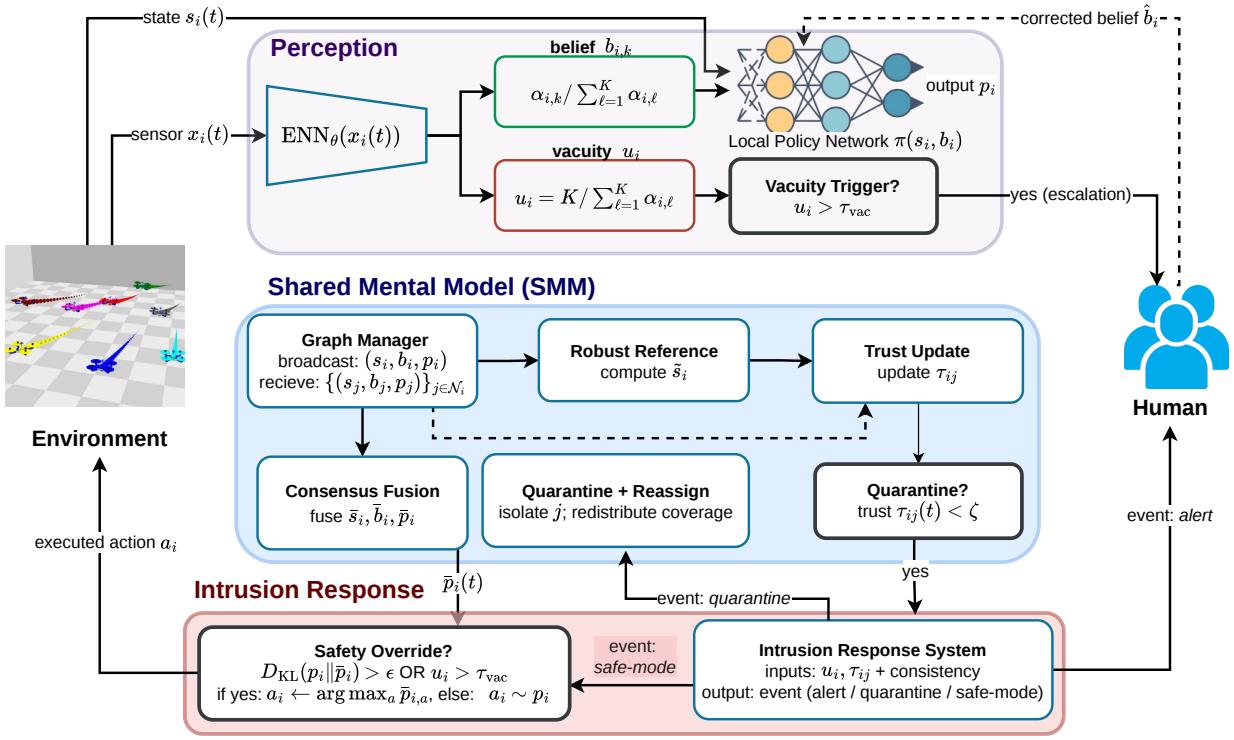


Figure 1: Proposed SHIELD overview for secure human–swarm teaming.

Node 0 represents the human operator and nodes $i = 1, \ldots, N$ represent UAVs. Each edge $(i, j) \in E(t)$ carries trust weight $\tau_{ij}(t)$, governing reliance on neighbor information. Edges are removed upon quarantine and restored during reintegration, enabling decentralized, trust-aware coordination.

4.1 Evidential Perception and Uncertainty-Driven Escalation

This component equips each UAV with calibrated belief estimation and epistemic uncertainty quantification, enabling principled human escalation only when observations are ambiguous or out-of-distribution.

Evidential Perception. Each UAV i processes sensor input $x_i^K(t) \in \mathcal{X}$ using an evidential neural network ENN_θ, producing Dirichlet concentration parameters:

$$\boldsymbol{\alpha}_i^K(t) = \mathrm{ENN}_\theta\big(x_i^K(t)\big) \in \mathbb{R}_{>0}^K. \tag{7}$$

Belief and vacuity are computed as:

$$b_{i,k}(t) = \frac{\alpha_{i,k}(t)}{\sum_{\ell=1}^{K}\alpha_{i,\ell}(t)}, \tag{8}$$

$$u_i(t) = \frac{K}{\sum_{\ell=1}^{K}\alpha_{i,\ell}(t)}. \tag{9}$$

Here, $b_i(t) \in \Delta^{K-1}$ represents the belief distribution and $u_i(t) \in (0, 1]$ captures epistemic uncertainty; higher vacuity indicates unfamiliar or out-of-distribution observations.

Uncertainty-Driven Escalation. If $u_i(t) > \tau_{\mathrm{vac}}$, UAV i escalates the observation to the operator. Upon receiving a corrected belief $\mathbf{b}_i'(t)$, vacuity is reset $u_i(t) \leftarrow 0$ and operation resumes. The ENN uses a softplus output layer to ensure $\alpha_{i,k} > 0$ and is trained offline with evidential calibration losses [40]; no retraining is required during deployment.

4.2 Adaptive Trust Modeling and Structural Resilience

This component maintains a dynamic reliability estimate over inter-agent communication, enabling the swarm to attenuate inconsistent peers and preserve coordination under faults.

For each edge $(i, j) \in E(t)$, we maintain consistency counters $C_{ij}(t)$ and $T_{ij}(t)$. To prevent circular feedback from trust-weighted consensus, a robust reference state is computed:

$$\tilde{\mathbf{s}}_i(t) = \mathrm{RobustAgg}\big(\{\mathbf{s}_j(t)\}_{j\in\mathcal{N}_i(t)}\big). \tag{10}$$

Here, RobustAgg is the coordinate-wise median across all neighbor states, chosen to avoid circular dependency: since trust scores depend on consistency with the reference, the reference must be computed without trust weights. The median

tolerates up to f arbitrary values per coordinate, matching the $f < \lfloor N/3 \rfloor$ fault bound; e.g., a spoofed position 3 m from the true cluster is excluded before trust scores are updated, preventing corrupted states from bootstrapping high trust.

Vacuity and trust serve complementary roles: vacuity (u_i) captures self-uncertainty in local perception, while trust (τ_{ij}) captures the reliability of a neighbor's communications. The two interact at the safety override (Eq. 16), where either signal independently triggers the consensus fallback.

Counters update as:

$$C_{ij}(t+\Delta t) = C_{ij}(t) + \mathbf{1}\{\|\mathbf{s}_j(t) - \tilde{\mathbf{s}}_i(t)\|_2 < \delta\}, \quad (11)$$

$$T_{ij}(t+\Delta t) = T_{ij}(t) + 1. \quad (12)$$

Trust evolves according to:

$$\tau_{ij}(t+\Delta t) = \frac{C_{ij}(t+\Delta t)+\kappa}{T_{ij}(t+\Delta t)+\kappa} \cdot \exp\left[-\beta(t - t_{ij}^{\text{last}})\right]. \quad (13)$$

Decay is applied only if no message is received beyond $\Delta t_{\text{timeout}}$, preventing unintended trust collapse during short outages. If $\tau_{ij} < \zeta$, edge (i, j) is removed, isolating unreliable agents and reinforcing structural resilience.

4.3 Consensus-Based Safety Arbitration

This component translates distributed beliefs and trust scores into safe collective actions, ensuring that local proposals remain aligned with group consensus and uncertainty-aware safeguards.

Trust-Weighted Fusion. Each UAV i aggregates neighbor information using trust-weighted averaging:

$$\bar{\mathbf{s}}_i = \frac{\sum_{j\in\mathcal{N}_i}\tau_{ij}\mathbf{s}_j}{\sum_{j\in\mathcal{N}_i}\tau_{ij}},\ \bar{b}_i = \frac{\sum_{j\in\mathcal{N}_i}\tau_{ij}b_j}{\sum_{j\in\mathcal{N}_i}\tau_{ij}},\ \bar{p}_i = \frac{\sum_{j\in\mathcal{N}_i}\tau_{ij}p_j}{\sum_{j\in\mathcal{N}_i}\tau_{ij}}. \quad (14)$$

If $\sum_{j\in\mathcal{N}_i}\tau_{ij} < \eta$, UAV i falls back to local estimates, preventing unstable decisions when trusted neighbors are unavailable.

Divergence-Based Safety Override. Let $p_i(t)$ denote the local action distribution and $\bar{p}_i(t)$ the fused distribution. We define the one-way KL divergence:

$$D_{\text{KL}}(p_i\|\bar{p}_i) = \sum_a p_{i,a} \ln \frac{p_{i,a}}{\bar{p}_{i,a}}. \quad (15)$$

A safety override is enforced if

$$D_{\text{KL}}(p_i\|\bar{p}_i) > \varepsilon \quad \text{or} \quad u_i > \tau_{\text{vac}}. \quad (16)$$

In this case,

$$a_i(t) = \arg\max_a \bar{p}_{i,a}(t), \quad (17)$$

otherwise $a_i(t) \sim p_i(t)$. This rule replaces extreme or uncertainty-driven proposals with a consensus action, preserving safety without eliminating decentralized autonomy.

4.4 Resilient Agent Lifecycle Management

This component ensures continuity of swarm operation by isolating unreliable agents, redistributing responsibilities, and safely reintegrating recovered nodes.

Quarantine and Coverage Reassignment. When agent j is quarantined, its patrol region is redistributed using a max-heap $\mathcal{H}$ keyed by capacity score

$$w_k(t) = \lambda_1 E_k(t) + \lambda_2 C_k^{\text{avail}}(t) - \lambda_3 L_k(t), \quad (18)$$

where $E_k(t) \in [0,1]$ denotes the remaining battery charge of UAV k normalized by its full capacity, $C_k^{\text{avail}}(t) \in [0,1]$ represents its available operational capacity computed as the fraction of onboard sensing and computation resources not currently committed to active tasks, and $L_k(t) \in [0,1]$ encodes the ratio of waypoints currently assigned to UAV k relative to the maximum single-agent coverage limit. The nonnegative weights $\lambda_1, \lambda_2, \lambda_3$ control the tradeoff among endurance, spare capacity, and task load. Intuitively, agents with more energy and idle resources but fewer existing assignments are preferred as reassignment targets. The reassignment process executes in $O(\log N)$ time via a single heap extraction and update per reassigned waypoint, ensuring scalable redistribution as swarm size increases.

Reintegration. A quarantined UAV may rejoin after a cooldown T_q if it satisfies both low vacuity ($u_j \le \tau_{\text{vac}}$) and behavioral consistency with neighbors for m consecutive steps. Upon reintegration, trust is restored gradually from τ_{init} to prevent abrupt structural shifts.

Algorithm 1 outlines the per-timestep execution of SHIELD. At each control interval Δt, the procedure updates uncertainty, adapts trust, performs trust-weighted fusion, and enforces safety arbitration before execution. This yields a decentralized, trust-aware control loop in which routine decisions remain within the swarm, while sustained uncertainty or divergence triggers operator escalation.

5 Experimental Setup

This section outlines the task, simulation environment, adversarial setup, evaluation metrics, human-feedback protocol, baselines, and implementation details for SHIELD.

5.1 Operational Task and Environment

Operational Task. We evaluate SHIELD on a waypoint-surveillance mission requiring safe navigation. Each UAV must sequentially visit assigned waypoints while avoiding collisions with obstacles and other agents, then return to a designated home position. Because waypoints are spatially distributed across the arena, agents must coordinate and share local observations to prevent redundant coverage. When a UAV is quarantined (Section 4), its remaining waypoints are

Algorithm 1: SHIELD Per-Timestep Update

Input: Agent i, graph $\mathcal{G}(t)$, local state $\mathbf{s}_i(t)$, sensor $x_i(t)$, policy π_i

1. Evidential Perception

$\boldsymbol{\alpha}_i \leftarrow \mathrm{ENN}_\theta(x_i(t))$

$(b_i, u_i) \leftarrow \mathrm{DirichletBeliefVacuity}(\boldsymbol{\alpha}_i)$

if $u_i > \tau_{\mathrm{vac}}$ **then**

  $b_i \leftarrow \mathrm{QueryHuman}(x_i(t))$

  $u_i \leftarrow 0$

end if

2. Local Action Proposal

$p_i \leftarrow \pi_i(\mathbf{s}_i(t), b_i)$

3. Communication

Broadcast $(\mathbf{s}_i, b_i, p_i)$ to $\mathcal{N}_i$; receive $\{(\mathbf{s}_j, b_j, p_j)\}$

4. Trust Update

$\tilde{\mathbf{s}}_i \leftarrow \mathrm{RobustAgg}(\{\mathbf{s}_j\}_{j\in\mathcal{N}_i})$

for all $j \in \mathcal{N}_i$ **do**

  $C_{ij} \leftarrow C_{ij} + \mathbf{1}_{\|\mathbf{s}_j - \tilde{\mathbf{s}}_i\| < \delta}$

  $T_{ij} \leftarrow T_{ij} + 1$

  $\tau_{ij} \leftarrow \dfrac{C_{ij} + \kappa}{T_{ij} + \kappa}$

  if $t - t_{ij}^{\mathrm{last}} > \Delta t_{\mathrm{timeout}}$ **then**

    $\tau_{ij} \leftarrow \tau_{ij} \cdot e^{-\beta(t - t_{ij}^{\mathrm{last}})}$

  end if

  if $\tau_{ij} < \zeta$ **then**

    Remove edge (i, j) from $\mathcal{G}(t)$

    Quarantine(j); ReassignCoverage(j)

  end if

end for

5. Trust-Weighted Fusion

if $\sum_{j\in\mathcal{N}_i} \tau_{ij} \geq \eta$ **then**

  $\bar{p}_i \leftarrow \mathrm{TrustWeightedAvg}(\{p_j, \tau_{ij}\})$

else

  $\bar{p}_i \leftarrow p_i$ ▷ fallback when insufficient trusted neighbors

end if

6. Safety Arbitration

if $D_{\mathrm{KL}}(p_i \| \bar{p}_i) > \varepsilon$ **or** $u_i > \tau_{\mathrm{vac}}$ **then**

  $a_i \leftarrow \arg\max \bar{p}_i$

else

  $a_i \sim p_i$

end if

7. Execution

Execute a_i and update $\mathbf{s}_i(t + \Delta t)$

Output: Updated state $\mathbf{s}_i(t + \Delta t)$

reassigned to healthy neighbors, testing the swarm's ability to maintain coverage under agent attrition. This task reflects practical drone applications such as perimeter monitoring, infrastructure inspection, and search-and-rescue coverage.

Simulation Environment. All experiments are conducted in QuadSwarm, a multi-quadrotor simulator integrated with Sample Factory via a Gym-style interface [23]. The simulator runs at 50 Hz and models quadrotor dynamics including motor lag, aerodynamic effects, and bounded Gaussian sensor noise ($\sigma_{\mathrm{GPS}} = 0.3$ m, $\sigma_{\mathrm{IMU}} = 0.01$ rad/s). The arena is a $20\,\mathrm{m} \times 20\,\mathrm{m} \times 10\,\mathrm{m}$ volume. In nominal settings the arena is obstacle-free; in obstacle tests, 20% of the volume contains static OctoMap occluders. Each UAV observes its two nearest neighbors through a 90° depth camera. Episodes start from randomized positions within a 2 m radius and terminate upon goal completion, crash, or after 2.5×10^7 environment steps.

5.2 Adversarial Parameterization

Adversarial experiments instantiate the six attack classes defined in Section 3. Table 3 summarizes their parameterization. Attacks are injected on a staggered schedule such that no two attack types activate simultaneously on the same agent.

GPS spoofing (A1) injects a uniform displacement $\boldsymbol{\Delta} \sim \mathcal{U}(-3.0, 3.0)^3$ m for 5 s approximately every 20 s. Communication jamming (A2) independently drops inter-agent messages with probability 0.5 during 3–7 s intervals. Byzantine faults (A3) replace 10% of outgoing state messages with samples drawn from the global state distribution. Replay attacks (A4) provide observations delayed by 10 s for 5 s episodes occurring roughly every 25 s. Malware injection (A5) induces intermittent policy drift bounded by $\|\Delta\pi\| \leq 0.15$, causing selective waypoint omission or heading deviation. Network intrusion (A6) enables passive monitoring of one mesh segment and selective injection of crafted state messages that mimic legitimate agents.

In all trials, at most $f = 2$ of $N = 8$ agents are simultaneously compromised, consistent with the $f < \lfloor N/3 \rfloor$ constraint in Section 3.

5.3 Evaluation Metrics

We report three core mission metrics, each computed per episode and averaged over 30 independent evaluation trials. Let T denote the episode length and N the number of UAVs in the swarm.

(1) Crash Penalty (CP).

$$\mathrm{CP} = \sum_{t=1}^{T} r_t^{\mathrm{crash}}, \tag{19}$$

where $r_t^{\mathrm{crash}} < 0$ is applied whenever a UAV collides with an obstacle or another agent. CP captures cumulative safety degradation over the entire episode; values closer to zero indicate fewer or less severe collisions.

(2) Mission Success Rate (MSR).

$$\mathrm{MSR} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\{\mathrm{goal}_i \wedge \neg\mathrm{crash}_i\}, \tag{20}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function and $goal_i$ indicates that UAV i has visited all assigned waypoints and returned to its home position. MSR measures the fraction of agents that complete all assigned waypoints without crashing in a given episode. It jointly reflects task completion and safety compliance and is reported as a proportion in $[0, 1]$.

(3) Survival Rate (SR).

$$\mathrm{SR} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\{\neg\mathrm{crash}_i\}, \tag{21}$$

which measures the fraction of agents that remain crash-free at episode termination. SR isolates safety performance from mission success and is reported in $[0,1]$.

For intrusion-robustness experiments (Exp. 2), we additionally evaluate detection and recovery performance.

(4) Detection Rate (DR).

$$\mathrm{DR} = \frac{\text{Number of correctly flagged attacks within 5 s}}{\text{Total number of injected attacks}}, \quad (22)$$

measuring the ability of the Intrusion Response System to identify attacks shortly after onset.

(5) False Positive Rate (FPR).

$$\mathrm{FPR} = \frac{\text{Number of benign timesteps flagged as attacks}}{\text{Total benign timesteps}}, \quad (23)$$

quantifying how often normal behavior is incorrectly classified as adversarial.

(6) Recovery Time (RT).

$$\mathrm{RT} = \min\{\Delta t : \|\mathbf{s}(t+\Delta t) - \mathbf{s}_{\mathrm{nom}}\| \leq 0.1\|\mathbf{s}_{\mathrm{nom}}\|\}, \quad (24)$$

where $\mathbf{s}_{\mathrm{nom}}$ denotes nominal formation state. RT captures how quickly the swarm returns within 10% of nominal formation after an attack ends.

(7) Human Intervention Count (HIC).

$$\mathrm{HIC} = \sum_{t=1}^{T} \mathbf{1}\{\mathrm{escalation}_t = 1\}, \quad (25)$$

representing the number of binary operator confirmations per episode. Lower HIC indicates reduced operator workload and stronger autonomous filtering.

5.4 Human Feedback Protocol

In all experiments, the human operator is modeled as a scripted oracle with access to ground-truth agent states and attack labels, consistent with prior human-in-the-loop RL evaluations [36, 43]. When SHIELD triggers escalation (vacuity $u_i > \tau_{\mathrm{vac}}$ or divergence $D_{\mathrm{KL}} > \varepsilon$), the oracle issues a binary accept/reject response within a fixed 0.5 s latency (25 simulator steps). Baselines receive comparable feedback under identical timing constraints: COACH receives corrective targets, TAMER receives scalar reward feedback, and SAHO blends human input at the same trigger frequency. Interventions are event-driven rather than continuous, and no method is granted unlimited human input. We adopt a binary interface to reduce cognitive load under supervisory time pressure [7].

5.5 Baselines

We compare five methods under identical observation and action spaces, training budget (2.5×10^7 steps), and simulation backend to ensure fair evaluation.

(1) Advantage Actor-Critic with Policy Optimization (APPO) [39]. APPO (also referred to as *Baseline DRL*) uses pure deep reinforcement learning without human input. The policy is trained end-to-end from environmental rewards and serves as a fully autonomous reference model without uncertainty modeling, trust adaptation, or intervention mechanisms.

(2) Corrective Advice Communicated by Humans (COACH) [36]. COACH incorporates human corrective feedback by treating intervention signals as supervised action targets. When triggered, human input directly adjusts the policy toward corrected behavior, enabling interactive refinement without structured uncertainty reasoning or trust adaptation.

(3) Shared Autonomy via Human Override (SAHO) [27]. SAHO blends human input with autonomous control using a cost-to-go formulation under a latent-goal partially observable Markov decision process (POMDP). Human guidance is integrated into action selection, supporting shared autonomy without explicit intrusion detection or trust-weighted fusion.

(4) Training an Agent Manually via Evaluative Reinforcement (TAMER) [43]. TAMER learns a predictive reward model from scalar human feedback and optimizes behavior using the learned reward. The approach improves learning efficiency but relies on reactive shaping rather than proactive uncertainty-driven escalation or structural resilience.

(5) SHIELD. Our proposed framework augments deep reinforcement learning with evidential perception, dynamic trust-weighted consensus, divergence-based safety arbitration, and an embedded intrusion-response mechanism to enable scalable and trustworthy human-swarm teaming.

These baselines represent four paradigms: autonomous DRL, corrective demonstrations, shared autonomy blending, and human reward shaping. Table 1 summarizes which capabilities each method provides; notably, only SHIELD integrates all four components. The HMT baselines were designed for benign settings and do not include attack detection, which is precisely the gap SHIELD addresses. While recent robust multi-agent communication methods exist (e.g., active defense [47], Bayesian-game MARL [33], targeted trust-based communication [41]), they focus on inter-agent message robustness rather than the human-swarm teaming loop central to this work. Their relationship to SHIELD is discussed in Section 2.

5.6 Implementation Details

Each UAV employs an evidential neural network (ENN) for perception, consisting of a four-layer convolutional backbone (32–64–128–256 filters with batch normalization and ReLU activations) followed by two fully connected layers. The network outputs $K = 10$ Dirichlet concentration parameters via a softplus activation, corresponding to the semantic classes defined in Table 17 in Appendix A: five security states (NORMAL, SUSPICIOUS, ATTACK, MALWARE, INTRU-

Table 1: Capability comparison across methods.

Method	Uncertainty	Trust	Intrusion Det.	Human Esc.
APPO [39]	✘	✘	✘	✘
COACH [36]	✘	✘	✘	▲
SAHO [27]	✘	✘	✘	▲
TAMER [43]	✘	✘	✘	▲
SHIELD	✔	✔	✔	✔

Note: ✔indicates full support of the capability, ▲indicates partial support, and ✘indicates no support.

SION) and five environmental states (CLEAR_PATH, OBSTACLE, CONGESTED, EMERGENCY, UNKNOWN). These parameters are normalized into a belief vector with an associated vacuity score.

The control policy is a multilayer perceptron that takes the 10-dimensional belief vector concatenated with a 24-dimensional local state vector (position, velocity, orientation, angular velocity, and neighbor encodings). It passes through three hidden layers (256–256–128 units with tanh activations) and outputs $A = 7$ discrete thrust commands via softmax. All methods are trained using APPO for 2.5×10^7 steps at 50 Hz with minibatch size 64, 8 epochs per update, learning rate 3×10^{-4}, discount factor $\gamma = 0.99$, Generalized Advantage Estimation (GAE) parameter $\lambda = 0.95$, value loss coefficient 0.5, and entropy coefficient 0.01.

The six SHIELD thresholds ($\tau_{\text{vac}} = 0.4$, $\kappa = 0.1$, $\beta = 0.05$, $\zeta = 0.2$, $\varepsilon = 0.3$, $\delta = 0.15$) are selected via grid search on a held-out validation split and remain fixed across experiments. Sensitivity analysis in Section 6 shows robustness to $\pm 20\%$ perturbations. All UAVs share ENN and policy weights via parameter sharing. Policies are checkpointed every 5×10^5 steps and evaluated over 30 independent trials per configuration. Per-agent inference latency is 4.2 ms on a single GPU with 8 UAVs, well within the 20 ms control budget at 50 Hz.

Experimental Scenarios. We evaluate SHIELD under three complementary protocols designed to assess nominal performance, robustness under attack, and scalability.

(1) Nominal Benchmarking. All five methods are evaluated in open-field and dense-field (20% obstacle) settings with $N = 8$ UAVs. We measure convergence and steady-state performance using CP, MSR, and SR over 2.5×10^7 steps.

(2) Intrusion Robustness. We activate six attack classes (A1–A6, Table 3) in dense-field settings and compare three variants: full SHIELD with IRS, SHIELD without IRS, and a quarantine-only configuration. We report Detection Rate (DR), False Positive Rate (FPR), Recovery Time (RT), and Human Intervention Count (HIC) aggregated across attacks.

(3) Ablation, Scalability, and Sensitivity. We disable individual SHIELD modules (vacuity escalation, trust updates, or safety override) and vary swarm size ($N \in \{8, 16, 32\}$) and obstacle density ($10\%, 20\%, 30\%$). Performance is evaluated via CP, MSR, and SR, and per-step computational overhead (inference time, message size, memory) is reported to assess deployment scalability.

6 Experimental Results & Analyses

We organize results around the three experimental scenarios defined in Section 5, analyzing nominal performance, robustness under adversarial attacks, and scalability through ablation and sensitivity studies.

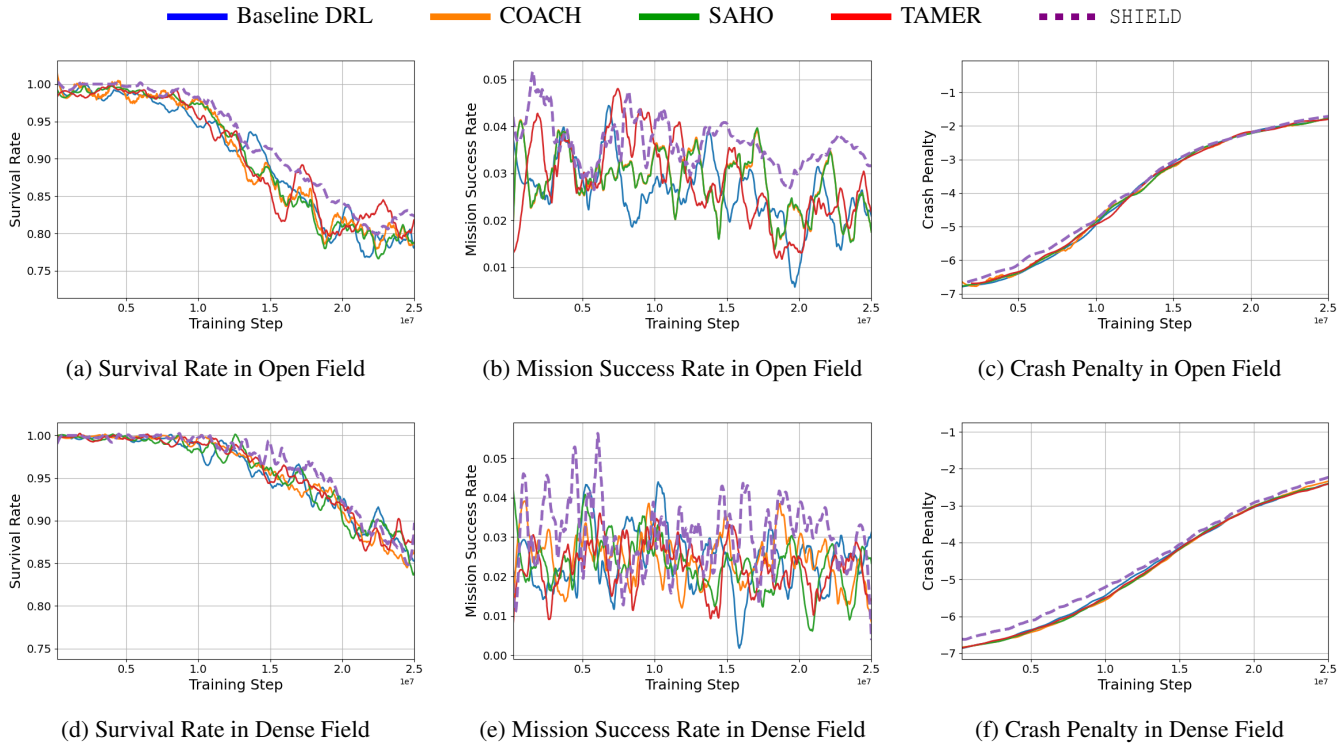
6.1 Nominal Performance and Convergence Analysis

Figure 2 reports *training-time* trajectories for Survival Rate (SR), Mission Success Rate (MSR), and Crash Penalty (CP) across five methods in open-field and dense-field settings. Unless otherwise noted, tables in this section present *test-time* metrics averaged over 30 independent evaluation trials; absolute SR values in Tables 4–6 therefore differ in scale from the training curves. Training-time SR (Figure 2) includes exploratory episodes with high-risk actions, inflating crash rates, while test-time SR (tables) uses deterministic evaluation from the converged checkpoint over 30 trials. The two regimes are standard in deep RL and should not be compared directly.

Open Field. In the obstacle-free setting (Figure 2(a)–(c)), all methods begin with near-perfect survival and progressively adopt more exploratory behaviors. By the end of training, SHIELD stabilizes near 82% SR, exceeding the 78–79% achieved by Baseline DRL, COACH, SAHO, and TAMER. Although MSR remains low due to task difficulty (all 8 UAVs must complete full waypoint sequences without collision), SHIELD consistently leads: it peaks above 5% early in training and converges to approximately 3.5–4.0%, while competing methods settle around 2.0–2.5%. CP improves from roughly -7.0 to -1.8 across methods, with SHIELD achieving comparable or slightly better final values. These results confirm that trust adaptation and uncertainty-driven escalation do not impede learning under benign conditions.

Dense Field. With 20% obstacle occupancy (Figure 2(d)–(f)), collision frequency increases and MSR declines for all methods. Nevertheless, SHIELD maintains a consistent advantage. Final SR reaches approximately 86%, compared to 84–85% for baselines. MSR peaks between 4–6% and converges around 2.5–3.0%, outperforming alternatives below 2.0%. CP decreases steadily across methods, with SHIELD showing sustained improvement after 1×10^7 steps.

Convergence Characteristics. The performance gap is most pronounced during early and mid-training (before 1.5×10^7



(a) Survival Rate in Open Field (b) Mission Success Rate in Open Field (c) Crash Penalty in Open Field

(d) Survival Rate in Dense Field (e) Mission Success Rate in Dense Field (f) Crash Penalty in Dense Field

Figure 2: **Training Dynamics Across Environments.** Each panel plots the smoothed evolution (2.5×10^7 training steps) of one performance metric in two settings. **(a)–(c)** show an obstacle-free "Open Field," where UAVs only contend with downwash and dynamics. **(d)–(f)** repeat the same metrics in a "Dense Field" containing 20 % static OctoMap obstacles and limited neighbor visibility. **Survival Rate** measures the fraction of crash-free agents per episode, **Mission Success Rate** counts the fraction of agents completing all waypoints without collision, and **Crash Penalty** is the cumulative negative reward from impacts (less negative is better).All values are proportions in $[0, 1]$ except CP which is unbounded negative. These are training-time metrics; test-time evaluation metrics (Tables 4–6) are computed separately from the final checkpoint and differ in scale.

steps), when uncertainty guides exploration. As policies converge and obstacle avoidance stabilizes, SR and CP differences narrow; however, SHIELD retains a persistent MSR advantage. We note that the narrowing of SR and CP gaps at convergence is expected: all methods eventually learn collision avoidance in this arena. The practical significance of SHIELD's advantage lies in two aspects. First, the *early-phase* safety gap matters for deployment, where policies operate under partial adaptation or distribution shift rather than full convergence. Second, SHIELD's persistent MSR advantage (about 50–75% relative improvement over baselines at convergence) indicates that trust-weighted coordination improves task completion even after basic collision avoidance is learned.

6.2 Intrusion Detection Performance

Table 2 reports Detection Rate (DR), False Positive Rate (FPR), Recovery Time (RT), and Human Intervention Count (HIC) for four defense variants under the six attack classes (A1–A6) defined in Section 3 and parameterized in Table 3. Results are aggregated across attack types. For variants without an explicit IRS alarm stream, Detection Rate is computed consistently: an attack is detected if a defensive action (quarantine or override) occurs within 5 s of onset. For SHIELD+IRS, detection corresponds to an IRS alarm optionally confirmed by the operator.

Overall Performance. Without IRS, SHIELD relies solely on trust decay, detecting only 12.0% (±6.16) of intrusions and requiring 28.7 human confirmations per episode. The vacuity-only variant increases detection to 70.4% but incurs 18.3% false positives, 20.5 s recovery time, and 20.8 interventions; the high detection variance (±17.59) indicates unstable alerting. Quarantine-only achieves 58.2% detection with faster recovery (15.4 s) but suffers 22.4% false positives and 19.8 confirmations. In contrast, full SHIELD+IRS achieves 82.5% detection (±3.45), 14.6% false positives (±1.56), 9.3 s recovery (±2.02), and 13.5 confirmations (±1.03), while exhibiting the lowest variance across all metrics. These results demonstrate both higher robustness and more stable behavior under diverse attack conditions.

False Positive Characteristics. As shown in Table 2, the 14.6% false positive rate of SHIELD+IRS is lower than vacuity-

Table 2: Intrusion robustness in the dense-field scenario (20% static obstacles) with 8 UAVs. Values are mean$_{\pm\text{std}}$ over 30 randomized attack trials. Dash (–) indicates metric not applicable for SHIELD w/o IRS, since without an active IRS there are no automated alarms (hence no false positives) nor recovery actions to time.

Variant	Detection Rate (%)	False Positive Rate (%)	Recovery Time (s)	Human Interventions
SHIELD w/o IRS	$12.04_{\pm 6.16}$	–	–	$28.67_{\pm 4.12}$
Vacuity Only IDS	$70.36_{\pm 17.59}$	$18.34_{\pm 5.72}$	$20.45_{\pm 4.56}$	$20.78_{\pm 2.30}$
Quarantine Only	$58.21_{\pm 8.45}$	$22.37_{\pm 6.31}$	$15.35_{\pm 9.11}$	$19.82_{\pm 3.19}$
SHIELD + IRS	$\mathbf{82.52}_{\pm 3.45}$	$\mathbf{14.55}_{\pm 1.56}$	$\mathbf{9.25}_{\pm 2.02}$	$\mathbf{13.47}_{\pm 1.03}$

Table 3: Attack parameterization for intrusion-robustness experiments. All attacks follow the threat model in Section 3.

Attack	**Duration / Interval**	**Severity**	**Target**
A1: GPS Spoofing	5 s on / 20 s off	$\delta_{\max} = 3.0$ m	1 random UAV
A2: Comm. Jamming	3–7 s window	Drop prob. $p = 0.5$	2 random links
A3: Byzantine Faults	Continuous	Inject prob. $p = 0.10$	1 compromised UAV
A4: Replay Attack	5 s on / 25 s off	Lag $\Delta t_r = 10$ s	1 random UAV
A5: Malware Injection	Continuous	Policy drift $\|\Delta\pi\| \leq 0.15$	1 compromised UAV
A6: Network Intrusion	Continuous	Eavesdrop + selective inject	1 network segment

only (18.3%) and quarantine-only (22.4%). Most false alarms stem from legitimate operational hazards such as obstacle encounters or congestion that temporarily elevate vacuity. The joint security–environmental ENN classification (Section 4), separating five security from five environmental states, mitigates misattribution by distinguishing environmental anomalies from adversarial events. The low standard deviation ($\pm 1.56\%$) further indicates consistent alert behavior.

Operator Workload Reduction. SHIELD+IRS reduces human confirmations by 53% compared to the no-IRS variant (13.5 vs. 28.7 per episode). IRS resolves events when vacuity, trust decay, and attack signatures are consistent, escalating only ambiguous cases. The event-triggered binary interface keeps intervention frequency low and predictable, with variance limited to ± 1.03. Under nominal (attack-free) conditions, Table 2 shows fewer than five queries per 500 s episode. Given sub-second binary decision times and error rates below 5% under time pressure [14], total operator engagement remains under 5 s per mission, lower than continuous shared-autonomy paradigms.

Table 4: Component ablation in dense field (20% obstacles) using 8 UAVs. Results are mean$_{\pm\text{std}}$ over 30 trials.

Variant	CP	MSR	SR
Full SHIELD	$\mathbf{-3.85}_{\pm 0.45}$	$\mathbf{0.040}_{\pm 0.006}$	$\mathbf{0.120}_{\pm 0.012}$
w/o Vacuity	$-4.35_{\pm 0.50}$	$0.032_{\pm 0.005}$	$0.095_{\pm 0.010}$
w/o Trust	$-4.62_{\pm 0.55}$	$0.025_{\pm 0.005}$	$0.081_{\pm 0.015}$
w/o Safety	$-5.19_{\pm 0.60}$	$0.018_{\pm 0.004}$	$0.077_{\pm 0.026}$

Table 5: Effect of obstacle density with 8 UAVs on SHIELD performance. Values are mean$_{\pm\text{std}}$ over 30 trials.

Density (%)	CP	MSR	SR
10	$\mathbf{-3.67}_{\pm 0.42}$	$\mathbf{0.045}_{\pm 0.004}$	$\mathbf{0.138}_{\pm 0.015}$
20	$-3.85_{\pm 0.45}$	$0.040_{\pm 0.006}$	$0.120_{\pm 0.012}$
30	$-4.17_{\pm 0.50}$	$0.035_{\pm 0.006}$	$0.103_{\pm 0.019}$

Table 6: Scalability of SHIELD in dense field (20% obstacles). Values are mean$_{\pm\text{std}}$ over 20 trials.

Swarm Size	CP	MSR	SR
8	$\mathbf{-3.85}_{\pm 0.45}$	$\mathbf{0.040}_{\pm 0.006}$	$\mathbf{0.120}_{\pm 0.012}$
16	$-3.99_{\pm 0.48}$	$0.026_{\pm 0.006}$	$0.115_{\pm 0.018}$
32	$-4.02_{\pm 0.58}$	$0.021_{\pm 0.002}$	$0.114_{\pm 0.016}$

6.3 Ablation, Scalability, and Sensitivity

Component Ablation. Table 4 isolates the contribution of each SHIELD module in the dense-field setting (20% obstacles, $N = 8$). Removing vacuity escalation degrades CP from -3.85 to -4.35 and reduces MSR by 20% relative (0.040 to 0.032), confirming that evidential uncertainty is critical for early risk detection. Disabling trust updates causes further degradation (CP -4.62, MSR 0.025, SR 0.081), indicating that peer filtering prevents cascading coordination failures. Removing safety overrides produces the largest drop (CP -5.19, MSR 0.018, SR 0.077) and increases variance (± 0.026 SR versus ± 0.012 for full SHIELD), demonstrating that KL-based arbitration is the final safeguard against catastrophic actions. The monotonic performance decline across ablations confirms that modules contribute complementary, non-redundant

Table 7: Vacuity threshold τ_{vac} sensitivity.

τ_{vac}	10% density			20% density			30% density		
	MSR	CP	SR	MSR	CP	SR	MSR	CP	SR
0.32	0.040	−3.30	0.150	0.035	−3.60	0.130	0.030	−4.00	0.110
0.40	0.045	−3.67	0.138	0.040	−3.85	0.120	0.035	−4.17	0.103
0.48	0.042	−4.10	0.125	0.037	−4.00	0.110	0.033	−4.30	0.095

Table 8: Trust-decay rate β sensitivity.

β	10%			20%			30%		
	MSR	CP	SR	MSR	CP	SR	MSR	CP	SR
0.04	0.046	−3.60	0.140	0.042	−3.78	0.122	0.036	−4.10	0.106
0.05	0.045	−3.67	0.138	0.040	−3.85	0.120	0.035	−4.17	0.103
0.06	0.043	−3.75	0.135	0.038	−3.92	0.117	0.033	−4.24	0.100

Table 9: KL-divergence threshold ε sensitivity.

ε	10%			20%			30%		
	MSR	CP	SR	MSR	CP	SR	MSR	CP	SR
0.24	0.042	−3.40	0.145	0.038	−3.65	0.128	0.032	−4.05	0.108
0.30	0.045	−3.67	0.138	0.040	−3.85	0.120	0.035	−4.17	0.103
0.36	0.044	−3.90	0.132	0.039	−3.98	0.115	0.034	−4.30	0.098

benefits.

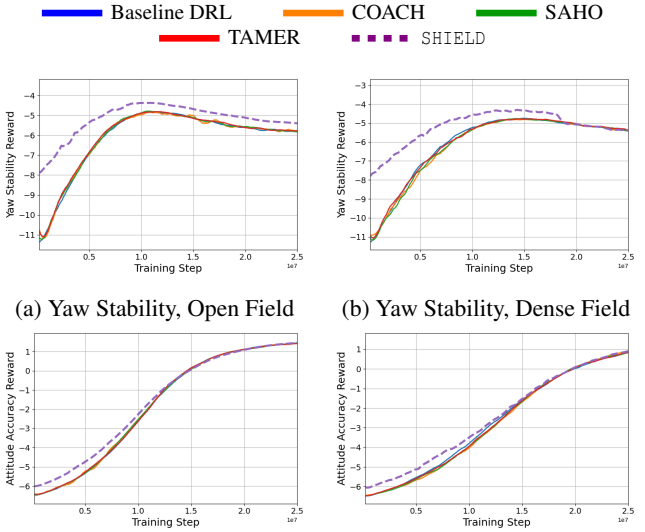
Obstacle Density Robustness. Table 5 shows graceful degradation as obstacle density increases from 10% to 30%. CP shifts from -3.67 to -4.17, MSR declines from 0.045 to 0.035, and SR decreases from 0.138 to 0.103. The contained performance loss across a threefold increase in obstacle density indicates that trust-weighted consensus adapts effectively under constrained visibility and higher collision risk.

Scalability with Swarm Size. Table 6 evaluates swarm sizes $N = \{8, 16, 32\}$. SR remains stable (0.120 to 0.114), suggesting decentralized trust scaling without coordination collapse. MSR decreases from 0.040 to 0.021, reflecting higher task competition as agents increase within a fixed arena. CP degrades marginally (-3.85 to -4.02), and variance remains bounded, indicating stable behavior at larger scales.

Threshold Sensitivity. Threshold robustness is evaluated by sweeping each parameter $\pm 20\%$ from its default. Detailed results are reported in Tables 7–10. Performance remains stable across all sweeps, confirming that SHIELD is not brittle to moderate parameter perturbations. The vacuity threshold τ_{vac} exhibits the greatest sensitivity: at 30% density, increasing from 0.40 to 0.48 reduces SR by 8% and worsens CP by 12% (Table 7). Trust-decay rate β induces less than 4% variation across metrics (Table 8). The KL threshold ε and smoothing factor κ similarly show bounded impact (Tables 9 and 10). In practice, $\tau_{vac} \in [0.35, 0.45]$ and $\varepsilon \in [0.25, 0.35]$ maintain near-optimal performance without extensive retuning.

Attack Parameter Sensitivity. To verify that detection performance is not sensitive to specific attack parameterizations, we additionally swept each attack parameter $\pm 50\%$ from its Table 3 default: $\delta_{max} \in \{1.5, 3.0, 4.5\}$ m, jamming drop probability $\in \{0.25, 0.5, 0.75\}$, Byzantine injection probability $\in \{0.05, 0.10, 0.15\}$, and replay lag $\in \{5, 10, 15\}$ s. Across all configurations, SHIELD+IRS maintains DR in $[78.3, 86.1]\%$ and FPR in $[12.8, 16.4]\%$. Stronger attacks (higher δ_{max}, higher jamming probability) paradoxically increase DR because larger deviations are easier to detect, at the cost of modestly higher FPR. Weaker attacks reduce both DR and FPR. Recovery time varies between 7.8–11.2 s across settings. These results confirm that the reported conclusions are not artifacts of specific parameter choices.

Computational Overhead. Runtime and communication statistics are summarized in Tables 11–13. The full SHIELD pipeline operates within the 20 ms control budget at 50 Hz. Per-agent inference time increases sublinearly with swarm size due to batched GPU execution, remaining 7.8 ms at $N = 64$ (Table 11). Message rates remain below 300 pps (Table 12), well within typical mesh network capacity. Memory overhead scales linearly with neighborhood size but remains modest (Table 13). These results confirm deployment feasibility without computational bottlenecks.



(a) Yaw Stability, Open Field (b) Yaw Stability, Dense Field

(c) Attitude Accuracy, Open Field (d) Attitude Accuracy, Dense Field

Figure 3: Yaw Stability and Attitude Accuracy rewards over training. SHIELD achieves consistently higher yaw stability than baselines, particularly under obstacle clutter, while converging to comparable attitude accuracy ($\sim$1.0), confirming that its safety mechanisms do not degrade orientation control.

Additional Stability Metrics. We report yaw stability and attitude accuracy to verify that SHIELD's safety and trust mechanisms do not degrade low-level flight dynamics (Figure 3).

Yaw stability rewards (Figures 3(a)–(b)) show that SHIELD achieves higher stability than baselines, particularly in dense environments where collision avoidance and consensus arbitration are most active. In obstacle settings, SHIELD maintains

Table 10: Trust-smoothing κ sensitivity.

κ	10%			20%			30%		
	MSR	CP	SR	MSR	CP	SR	MSR	CP	SR
0.08	0.043	−3.78	0.133	0.038	−3.92	0.115	0.033	−4.25	0.098
0.10	0.045	−3.67	0.138	0.040	−3.85	0.120	0.035	−4.17	0.103
0.12	0.047	−3.55	0.142	0.042	−3.75	0.125	0.037	−4.05	0.108

approximately 0.5–1.0 reward points higher yaw stability, indicating smoother rotational control when uncertainty-driven overrides suppress erratic proposals.

Attitude accuracy rewards (Figures 3(c)–(d)) converge to nearly identical final values (approximately 1.0) across all methods in both open and dense fields. This confirms that the additional trust updates, escalation triggers, and KL-based safety overrides introduce no measurable degradation in orientation control or steady-state tracking performance.

Table 11: Per-step runtime (ms) per UAV.

Swarm Size	8	16	32	64
Runtime (ms)	4.2	5.1	6.3	7.8
Std. Dev.	0.3	0.4	0.5	0.7

Table 12: Packets per second (pps) per UAV under consensus.

Swarm Size	8	16	32	64
Sent/Recv pps	200	230	260	290
Std. Dev.	15	18	22	26

7 Discussion and Limitations

Nominal and Adversarial Performance Insights. Across benign and adversarial settings, SHIELD improves learning stability and operational robustness through evidential vacuity, trust adaptation, decentralized fusion, and an IRS (Figure 2). The advantage is most pronounced during early training when uncertainty guides exploration, i.e., the regime most relevant to deployment under distribution shift, while SHIELD retains a persistent MSR advantage at convergence.

Layered Defense and Robustness–Sensitivity Trade-offs. Across the six attack classes, full SHIELD+IRS achieves the most balanced performance (Table 2), combining strong detection, fewer false positives, faster recovery, and reduced operator confirmations relative to partial variants. Vacuity-only and quarantine-only results reveal a trade-off: greater anomaly sensitivity improves detection but increases false alarms, while aggressive isolation accelerates recovery at the risk of unnecessary quarantines.

Ablation results (Table 4) confirm a layered defense structure: vacuity detects anomalies, trust filters unreliable peers, and KL-based overrides provide containment. Removing modules causes monotonic degradation, indicating complementarity. Reduced variance in detection and recovery further demonstrates stable and predictable defensive behavior essential for safety-critical cyber-physical systems.

Table 13: Per-agent memory footprint.

Component	Memory	Growth
ENN Weights (GPU)	5.0 MB	constant
Trust Counters (CPU)	0.1 MB/neighbor	linear
Total (2 neighbors)	5.2 MB	–

Scalability and Structural Stability. Scalability experiments (Table 6) and obstacle-density sweeps (Table 5) show gradual performance degradation as swarm size and environmental complexity increase. Survival Rate remains stable across larger swarms, while Mission Success Rate declines predictably due to increased coordination load within a fixed arena. Crash Penalty degradation remains bounded.

The decentralized trust graph provides implicit fault containment: compromised agents influence only local neighborhoods until trust decay isolates them. Decisions are formed locally and propagated through neighborhood fusion rather than a centralized controller, limiting bottlenecks and reducing global destabilization as complexity grows.

Threshold Robustness and Parameter Interpretability. Sensitivity sweeps (Tables 7–10) demonstrate bounded performance variation under $\pm 20\%$ perturbations of all thresholds. While the vacuity threshold τ_{vac} exhibits the greatest influence, particularly at higher obstacle densities, trust-decay and smoothing parameters induce only minor variation. These results suggest that thresholds operate within semantically meaningful ranges rather than narrow brittle optima, supporting deployment without exhaustive retuning.

Operator Load Calibration. Full SHIELD+IRS reduces human confirmations by 53% relative to the no-IRS variant while maintaining higher detection performance. This reflects calibrated escalation rather than suppressed alerts: the IRS resolves aligned vacuity, trust, and attack signals, escalating only ambiguous cases. The event-triggered binary interface keeps operator workload predictable, as indicated by low variance in intervention counts.

Practical Acceptability of Detection Performance. The 82.5% DR and 14.6% FPR must be interpreted within SHIELD's layered defense architecture. Undetected attacks (the remaining $\sim$17.5%) are not unmitigated: trust decay and KL-based safety overrides still attenuate impact, as shown by the monotonic degradation when these modules are removed (Table 4). The 14.6% FPR predominantly comprises environmental hazards (obstacles, congestion) that warrant heightened caution; the joint security–environmental ENN classification reduces misattribution relative to single-category

detectors (vacuity-only: 18.3%, quarantine-only: 22.4%). For comparison, reported false positive rates for UAV anomaly detectors range from 10–25% [10, 15], placing SHIELD+IRS at the lower end while also providing trust adaptation and human escalation that standalone detectors lack.

Limitations and Deployment Constraints. Several limitations arise from the current scope. First, evaluation is simulation-based; real deployments may introduce unmodeled aerodynamics, sensing artifacts, and latency affecting vacuity calibration and trust dynamics. Specific concerns include sensor drift affecting ENN calibration, communication latency altering trust decay dynamics, and actuator saturation limiting the effectiveness of consensus overrides. Physical-platform validation is planned as future work.

8 Conclusion and Future Work

This paper presents SHIELD, a unified framework for scalable human–swarm teaming that addresses safety and mission continuity under limited operator attention, uncertain observations, and cyber-physical compromise. SHIELD integrates evidential perception, dynamic trust adaptation, decentralized consensus, and divergence-based safety overrides within a single control loop, supported by an embedded Intrusion Response System that enables quarantine, safe-mode execution, and structured escalation only when necessary. Extensive simulations show improved convergence under nominal conditions and stronger resilience under attacks, including higher mission success, faster recovery, reduced collisions, and lower, more predictable operator workload. Overall, SHIELD demonstrates that tightly coupling uncertainty estimation, decentralized trust, and structured containment improves both robustness and practical deployability in multi-UAV systems.

Future work will extend SHIELD to physical UAV platforms to evaluate robustness under realistic sensing noise, actuator latency, and intermittent communication. We will investigate adaptive thresholding and context-conditioned policies to address nonstationary operating conditions, as well as richer, low-cognitive-load human feedback mechanisms beyond binary confirmation. Additional efforts will focus on communication-efficient trust update protocols for large-scale swarms and systematic evaluation against coordinated multi-agent deception and long-horizon intrusion strategies that explicitly manipulate consensus formation dynamics.

Ethical Considerations. This work is defensive in nature, aiming to improve the security and human oversight of cooperative drone swarms. All experiments are conducted entirely in simulation using the open-source QuadSwarm environment [23], with no physical deployments, real communication disruptions, or human subjects involved. The six attack classes evaluated are well-documented in UAV security literature [3, 8, 28, 32, 35], with parameterizations derived from published field studies. While adversarial strategies are described in sufficient detail for reproducibility, we introduce no novel offensive techniques nor provide artifacts directly applicable to deployed systems. The defensive contributions of this work, including uncertainty-aware intrusion detection, decentralized quarantine, and structured human escalation, substantially outweigh the marginal risk of informing adversarial strategy. We encourage future physical-platform extensions to follow responsible disclosure practices.

Acknowledgments

This work was supported by the Army Research Office under Grant No. W911NF-24-2-0241.

Use of Generative AI

Generative AI tools were used for editorial purposes in this manuscript, and all outputs were inspected by the authors to ensure accuracy and originality.

References

[1] ABDI, S. S., AND PALEY, D. A. Safe operations of an aerial swarm via a cobot human swarm interface. In *2023 IEEE International Conference on Robotics and Automation (ICRA)* (2023), IEEE, pp. 1701–1707.

[2] AGRAWAL, A., AND CLELAND-HUANG, J. Explaining autonomous decisions in swarms of human-on-the-loop small unmanned aerial systems. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing* (2021), vol. 9, pp. 15–26.

[3] BABU, G. A., KRISHNA, G. V., TIWARI, V., ANTONY, R., SURAJ, C., VIKAS, K., AND MURTHY, P. Drone-borne magnetic measurements in india. *Current Science 119*, 4 (2020), 598–600.

[4] BIYIK, E., AND SADIGH, D. Batch active preference-based learning of reward functions. In *Conference on robot learning* (2018), PMLR, pp. 519–528.

[5] BROWN, D., GOO, W., NAGARAJAN, P., AND NIEKUM, S. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International conference on machine learning* (2019), PMLR, pp. 783–792.

[6] BROWN, D. S., GOO, W., AND NIEKUM, S. Better-than-demonstrator imitation learning via automatically-ranked demonstrations. In *Conference on robot learning* (2020), PMLR, pp. 330–359.

[7] CHEN, J. Y., AND BARNES, M. J. Human–agent teaming for multirobot control: A review of human factors

issues. *IEEE Transactions on Human-Machine Systems 44*, 1 (2014), 13–29.

[8] CHUNG, S.-J., PARANJAPE, A. A., DAMES, P., SHEN, S., AND KUMAR, V. A survey on aerial swarm robotics. *IEEE Transactions on Robotics 34*, 4 (2018), 837–855.

[9] CUMMINGS, M. L., AND GUERLAIN, S. Developing operator capacity estimates for supervisory control of autonomous vehicles. *Human factors 49*, 1 (2007), 1–15.

[10] DESAI, A., DAI, R., CHEN, Y., HO, K., KEE, A., BULATOVIC, S., SHAFIUZZAMAN, M., BAI, K. Y., JEONG, I. U., SIU, D., ET AL. Stateful behavior inference and runtime enforcement for vehicle network security. In *3rd USENIX Symposium on Vehicle Security and Privacy (VehicleSec 25)* (2025), pp. 1–17.

[11] DING, S., DU, W., DING, L., GUO, L., AND ZHANG, J. Learning efficient and robust multi-agent communication via graph information bottleneck. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2024), vol. 38, pp. 17346–17353.

[12] DIVBAND SOORATI, M., CLARK, J., GHOFRANI, J., TARAPORE, D., AND RAMCHURN, S. D. Designing a user-centered interaction interface for human–swarm teaming. *Drones 5*, 4 (2021), 131.

[13] EBINE, Y., NOMOTO, K., TANAKA, Y., KOBAYASHI, R., TSURUOKA, G., AND MORI, T. {WIP}: Evaluating the {End-to-End} impact of false localization attacks on {vSLAM-Based} autonomous drones. In *3rd USENIX Symposium on Vehicle Security and Privacy (VehicleSec 25)* (2025), pp. 73–80.

[14] ENDSLEY, M. R. Toward a theory of situation awareness in dynamic systems. In *Situational awareness*. Routledge, 2017, pp. 9–42.

[15] ERBA, A., CASTELLANOS, J. H., SIHAG, S., ZONOUZ, S., AND TIPPENHAUER, N. O. {ConfuSense}: Sensor reconfiguration attacks for stealthy {UAV} manipulation. In *3rd USENIX Symposium on Vehicle Security and Privacy (VehicleSec 25)* (2025), pp. 45–62.

[16] FRIEDBERG, I., HONG, X., MCLAUGHLIN, K., SMITH, P., AND MILLER, P. C. Evidential network modeling for cyber-physical system state inference. *IEEE Access 5* (2017), 17149–17164.

[17] FUNG, H. L., DARVARIU, V.-A., HAILES, S., AND MUSOLESI, M. Trust-based consensus in multi-agent reinforcement learning systems. *arXiv preprint arXiv:2205.12880* (2022).

[18] GUO, Y., PANG, Y., LYONS, J., LEWIS, M., SYCARA, K., AND LIU, R. Trust-aware reflective control for fault-resilient dynamic task response in human–swarm cooperation. *AI 5*, 1 (2024), 446–464.

[19] HEJNA III, D. J., AND SADIGH, D. Few-shot preference learning for human-in-the-loop rl. In *Conference on Robot Learning* (2023), PMLR, pp. 2014–2025.

[20] HESTER, T., VECERIK, M., PIETQUIN, O., LANCTOT, M., SCHAUL, T., PIOT, B., HORGAN, D., QUAN, J., SENDONARIS, A., OSBAND, I., ET AL. Deep q-learning from demonstrations. In *Proceedings of the AAAI conference on artificial intelligence* (2018), vol. 32.

[21] HIGHNAM, K., ANGSTADT, K., LEACH, K., WEIMER, W., PAULOS, A., AND HURLEY, P. An uncrewed aerial vehicle attack scenario and trustworthy repair architecture. In *2016 46th annual IEEE/IFIP international conference on dependable systems and networks workshop (DSN-W)* (2016), IEEE, pp. 222–225.

[22] HOU, J., WANG, F., WANG, L., AND CHEN, Z. Reinforcement learning based multi-agent resilient control: From deep neural networks to an adaptive law. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2021), vol. 35, pp. 7737–7745.

[23] HUANG, Z., BATRA, S., CHEN, T., KRUPANI, R., KUMAR, T., MOLCHANOV, A., PETRENKO, A., PREISS, J. A., YANG, Z., AND SUKHATME, G. S. Quadswarm: A modular multi-quadrotor simulator for deep reinforcement learning with direct thrust control. *arXiv preprint arXiv:2306.09537* (2023).

[24] HUMPHREYS, T. E., LEDVINA, B. M., PSIAKI, M. L., O'HANLON, B. W., KINTNER, P. M., ET AL. Assessing the spoofing threat: Development of a portable gps civilian spoofer. In *Proceedings of the 21st International technical meeting of the satellite division of the institute of navigation (ION GNSS 2008)* (2008), pp. 2314–2325.

[25] HUSSEIN, A., GHIGNONE, L., NGUYEN, T., SALIMI, N., NGUYEN, H., WANG, M., AND ABBASS, H. A. Characterization of indicators for adaptive human-swarm teaming. *Frontiers in Robotics and AI 9* (2022), 745958.

[26] IBARZ, B., LEIKE, J., POHLEN, T., IRVING, G., LEGG, S., AND AMODEI, D. Reward learning from human preferences and demonstrations in atari. *Advances in neural information processing systems 31* (2018).

[27] JAVDANI, S., SRINIVASA, S. S., AND BAGNELL, J. A. Shared autonomy via hindsight optimization. *Robotics science and systems: online proceedings 2015* (2015), 10–15607.

[28] KERNS, A. J., SHEPARD, D. P., BHATTI, J. A., AND HUMPHREYS, T. E. Unmanned aircraft capture and control via gps spoofing. *Journal of field robotics 31*, 4 (2014), 617–636.

[29] KHEN, G., ZHAO, D., AND BACA, J. Intuitive human-swarm interaction with gesture recognition and machine learning. In *Proceedings of the Twenty-fourth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing* (2023), pp. 453–456.

[30] KHOPKAR, P. Mixed-initiative flexible autonomy in drone swarms for covid-19 applications. In *2020 IEEE International Symposium on Technology and Society (ISTAS)* (2020), IEEE, pp. 457–461.

[31] LEE, K., SMITH, L., AND ABBEEL, P. Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training. *arXiv preprint arXiv:2106.05091* (2021).

[32] LEWIS-PYE, A., AND ROUGHGARDEN, T. Byzantine generals in the permissionless setting. In *International Conference on Financial Cryptography and Data Security* (2023), Springer, pp. 21–37.

[33] LI, S., GUO, J., XIU, J., XU, R., YU, X., WANG, J., LIU, A., YANG, Y., AND LIU, X. Byzantine robust cooperative multi-agent reinforcement learning as a bayesian game. *arXiv preprint arXiv:2305.12872* (2023).

[34] LIANG, X., SHU, K., LEE, K., AND ABBEEL, P. Reward uncertainty for exploration in preference-based reinforcement learning. *arXiv preprint arXiv:2205.12401* (2022).

[35] LICHTMAN, M., JOVER, R. P., LABIB, M., RAO, R., MAROJEVIC, V., AND REED, J. H. Lte/lte-a jamming, spoofing, and sniffing: threat assessment and mitigation. *IEEE Communications Magazine 54*, 4 (2016), 54–61.

[36] MACGLASHAN, J., HO, M. K., LOFTIN, R., PENG, B., WANG, G., ROBERTS, D. L., TAYLOR, M. E., AND LITTMAN, M. L. Interactive learning from policy-dependent human feedback. In *International conference on machine learning* (2017), PMLR, pp. 2285–2294.

[37] OLFATI-SABER, R., AND MURRAY, R. M. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on automatic control 49*, 9 (2004), 1520–1533.

[38] PANG, Y., AND LIU, R. Trust-aware emergency response for a resilient human-swarm cooperative system. In *2021 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)* (2021), IEEE, pp. 15–20.

[39] PETRENKO, A., HUANG, Z., KUMAR, T., SUKHATME, G., AND KOLTUN, V. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *International Conference on Machine Learning* (2020), PMLR, pp. 7652–7662.

[40] SENSOY, M., KAPLAN, L., AND KANDEMIR, M. Evidential deep learning to quantify classification uncertainty. *Advances in neural information processing systems 31* (2018).

[41] SUN, C., ZANG, Z., LI, J., LI, J., XU, X., WANG, R., AND ZHENG, C. T2mac: Targeted and trusted multi-agent communication through selective engagement and evidence-driven integration. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2024), vol. 38, pp. 15154–15163.

[42] TANG, X., ZHANG, M., LIU, W., DU, B., AND LIU, Z. Towards a model of human-cyber–physical automata and a synthesis framework for control policies. *Journal of Systems Architecture 144* (2023), 102989.

[43] WARNELL, G., WAYTOWICH, N., LAWHERN, V., AND STONE, P. Deep tamer: Interactive agent shaping in high-dimensional state spaces. In *Proceedings of the AAAI conference on artificial intelligence* (2018), vol. 32.

[44] WATTEARACHCHI, W. D., LAKSHIKA, E., KASMARIK, K., AND BARLOW, M. A study on human-swarm interaction: A framework for assessing situation awareness and task performance. *arXiv preprint arXiv:2503.14810* (2025).

[45] WEI, L., WU, J., AND LONG, C. Blockchain-enabled trust management in service-oriented internet of things: Opportunities and challenges. In *Proceedings of the 2021 3rd International Conference on Blockchain Technology* (2021), pp. 90–95.

[46] WU, X., XIAO, L., SUN, Y., ZHANG, J., MA, T., AND HE, L. A survey of human-in-the-loop for machine learning. *Future Generation Computer Systems 135* (2022), 364–381.

[47] YU, L., QIU, Y., YAO, Q., SHEN, Y., ZHANG, X., AND WANG, J. Robust communicative multi-agent reinforcement learning with active defense. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2024), vol. 38, pp. 17575–17582.

[48] ZHANG, M., LI, N., LI, J., LIAO, J., AND LIU, J. Memory-based resilient control against non-cooperation in multi-agent flocking. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems* (2024), pp. 2075–2084.

[49] ZHAO, C., ZHENG, C., ROLDAN, L., SHKURTI, T., NAHARI, A., NEWMAN, W., TYLER, D., LEE, K., AND FU, M. J. Adaptable mixed-reality sensorimotor interface for human-swarm teaming: Person with limb loss case study and field experiments. *Field Robotics 3* (2023), 243–265.

A Implementation Details

A.1 Model Architecture and Design Rationale

We employ a two-stage architecture that decouples *uncertainty-aware perception* from *risk-sensitive decision making*. The perception module is implemented as an Evidential Neural Network (ENN) backbone that produces Dirichlet-distributed class evidence, while the control module is a policy MLP head that integrates ENN beliefs with the UAV state for discrete action selection.

Table 14 details the ENN backbone architecture. The network consists of four convolutional blocks with progressive spatial downsampling, followed by a fully connected embedding layer and a Softplus output layer that produces non-negative Dirichlet concentration parameters. The use of Softplus ensures strictly positive evidence values, enabling principled uncertainty quantification through subjective logic. The final 10-dimensional output corresponds to class-wise evidence, which is later transformed into belief masses and uncertainty measures used by the policy.

Table 14: ENN Backbone Architecture

Layer	Output Shape	Kernel/Stride/Pad	Notes
Conv1+BN+ReLU	$32 \times 64 \times 64$	$3 \times 3/1/1$	BatchNorm, ReLU
Conv2+BN+ReLU	$64 \times 32 \times 32$	$3 \times 3/2/1$	Spatial downsample
Conv3+BN+ReLU	$128 \times 16 \times 16$	$3 \times 3/2/1$	Further downsample
Conv4+BN+ReLU	$256 \times 8 \times 8$	$3 \times 3/2/1$	Final feature map
Flatten	16384	–	–
FC+ReLU	512	–	Hidden embedding
FC+Softplus	10	–	Dirichlet parameters

Table 15 presents the policy head architecture. The policy receives a concatenated input comprising the 10-dimensional ENN belief vector and a 24-dimensional UAV state representation, resulting in a 34-dimensional input. The multilayer perceptron uses three hidden layers with tanh activations to maintain bounded representations and improve training stability. The final Softmax layer outputs a categorical distribution over $A = 7$ discrete thrust commands. This design enables the controller to adapt its action selection based not only on state dynamics but also on epistemic uncertainty encoded by the ENN.

Table 15: Policy MLP Head Architecture

Layer	Output Dim	Notes
Input concat	34	10 ENN belief + 24 state dims
FC1+tanh	256	First hidden layer
FC2+tanh	256	Second hidden layer
FC3+tanh	128	Third hidden layer
FC4+softmax	$A = 7$	Discrete thrust commands

A.2 Training Configuration and SHIELD Control Parameters

We train the policy using PPO with generalized advantage estimation for stable updates and variance reduction. Table 16 reports both the standard RL hyperparameters, which govern optimization, exploration, and critic scaling, and the SHIELD-specific parameters, which control uncertainty escalation, trust adaptation, quarantine, and consensus consistency checks. All results are averaged across independent random seeds, each trained for 2.5×10^7 environment steps.

Table 16: Training and SHIELD-Specific Hyperparameters

Parameter	Value	Description
Learning rate	3×10^{-4}	Adam optimizer
Discount γ	0.99	Return estimation
GAE λ	0.95	Advantage estimation
Minibatch size	64	Samples per update
Epochs/update	8	Passes per minibatch
Entropy coeff.	0.01	Exploration regularization
Value-loss coeff.	0.5	Critic loss weighting
Total steps	2.5×10^7	Per seed
τ_{vac}	0.4	Vacuity escalation threshold
κ	0.1	Trust smoothing constant
β	0.05	Trust decay rate
ζ	0.2	Quarantine activation threshold
ε	0.3	KL-divergence override threshold
δ	0.15	Consensus consistency radius

Table 17: ENN Semantic Classes ($K = 10$)

ID	Class	Category
0	NORMAL	Security
1	SUSPICIOUS	Security
2	ATTACK	Security
3	MALWARE	Security
4	INTRUSION	Security
5	CLEAR_PATH	Environmental
6	OBSTACLE	Environmental
7	CONGESTED	Environmental
8	EMERGENCY	Environmental
9	UNKNOWN	Environmental

A.3 Environment and Attack Configuration

The simulator uses a $20 \times 20 \times 10$ m arena at 50 Hz with downwash ($C_d = 0.1$), motor lag ($\tau = 0.15$ s), and Gaussian sensor noise ($\sigma_{\text{GPS}} = 0.3$ m, $\sigma_{\text{IMU}} = 0.01$ rad/s). Dense-field tests occupy 20% of the volume with 0.5 m cubic OctoMap occluders. Episodes last up to 25,000 steps (500 s). Attack parameterization for the six classes appears in Table 3. Attacks use non-overlapping windows with at most $f = 2$ compromised agents, and per-timestep attack labels are logged for post hoc analysis.

A.4 ENN Semantic Classes

Table 17 lists the $K = 10$ semantic classes. Classes 0–4 capture security states; classes 5–9 capture environmental conditions. The joint taxonomy enables the ENN to distinguish adversarial anomalies from benign operational hazards, reducing false positives in the IRS.

B Theoretical Guarantees

We formalize four properties of SHIELD that underpin its operational reliability: trust estimates converge to true peer quality, safety overrides preserve invariant sets, intrusion detection provides quantifiable coverage, and consensus tolerates quarantine-induced topology changes.

Assumptions. The communication graph $\mathcal{G}(t)$ is time-varying but connected over every interval of length L; all signals lie in compact sets; trust parameters satisfy $\kappa > 0$, $\beta > 0$, $\zeta \in (0,1)$; inter-agent message delays are bounded by D; and the consensus action distribution $\bar{p}_i$ is derived from at least $(2f+1)$ honest neighbors.

Theorem 1 (Trust Convergence)**.** *(Ensures that trust scores faithfully track long-run peer reliability.)* If agent j's messages are consistent with the robust aggregate at fixed probability $p_{ij} > 0.5$ and delays are bounded by D, then $\tau_{ij}(t) \to p_{ij}$ almost surely.

Proof sketch. Write $\tau_{ij}(t) = r_{ij}(t) \cdot d_{ij}(t)$ where $r_{ij}(t) = (C_{ij} + \kappa)/(T_{ij} + \kappa)$ and $d_{ij}(t) = \exp[-\beta(t - t_{ij}^{\text{last}})]$. Each consistency check $\mathbf{1}\{\|\mathbf{s}_j - \tilde{\mathbf{s}}_i\| < \delta\}$ is a Bernoulli trial with success probability p_{ij}. Since κ is a fixed constant, the ratio $r_{ij}(t) \to p_{ij}$ almost surely by the Strong Law of Large Numbers as $T_{ij} \to \infty$. Under bounded delays ($t - t_{ij}^{\text{last}} \le D$), the decay factor satisfies $d_{ij}(t) \ge e^{-\beta D} > 0$ and $d_{ij}(t) \to 1$ whenever communication persists. The product therefore converges: $\tau_{ij}(t) \to p_{ij} \cdot 1 = p_{ij}$. □

Theorem 2 (Safety Invariance)**.** *(Ensures that the override mechanism cannot drive the system out of a safe operating region.)* Let $V : \mathcal{S} \to \mathbb{R}_{\ge 0}$ be a control Lyapunov function such that for every state s there exists an action a^* with $\dot{V}(s, a^*) \le 0$, and suppose $\arg\max_a \bar{p}_{i,a}(t)$ selects such an action. Then the override rule

$$a_i(t) = \begin{cases} \arg\max_a \bar{p}_{i,a}(t), & \text{if } D_{\text{KL}}(p_i \| \bar{p}_i) > \varepsilon \text{ or } u_i > \tau_{\text{vac}}, \\ \text{sample from } p_i(t), & \text{otherwise,} \end{cases} \tag{26}$$

ensures the state remains in the safe set $\mathcal{S} = \{s \mid V(s) \le c\}$ for all $t \ge 0$.

Proof sketch. Case 1 (override active): The consensus action $\arg\max_a \bar{p}_{i,a}$ satisfies $\dot{V} \le 0$ by assumption, so V is non-increasing. *Case 2 (override inactive):* $D_{\text{KL}}(p_i \| \bar{p}_i) \le \varepsilon$ implies $\|p_i - \bar{p}_i\|_1 \le \sqrt{2\varepsilon}$ by Pinsker's inequality. Since $\bar{p}_i$ concentrates mass on Lyapunov-decreasing actions and $\dot{V}$ is continuous in the action distribution, the sampled action satisfies $\dot{V}(s, a_i) \le 0$ provided ε is chosen such that $\sqrt{2\varepsilon}$ lies within the continuity radius of the Lyapunov decrease condition. □

Theorem 3 (Detection Lower Bound)**.** *(Quantifies the minimum fraction of attacks the IRS is guaranteed to detect.)* Let u_{nom} denote the nominal vacuity under benign operation and suppose intrusions shift vacuity upward by $\Delta_u > 0$, so that the post-attack vacuity distribution has mean $u_{\text{nom}} + \Delta_u$. If τ_{vac} satisfies $u_{\text{nom}} < \tau_{\text{vac}} < u_{\text{nom}} + \Delta_u$, then the IRS detects at least a fraction $1 - e^{-\lambda T \beta_d}$ of attacks over a mission of duration T, where $\beta_d = 1 - F(\tau_{\text{vac}}; u_{\text{nom}} + \Delta_u)$ is the per-step detection probability and λ is the attack arrival rate.

Proof sketch. At each attacked timestep, the vacuity $u_i(t)$ is drawn from a distribution with mean $u_{\text{nom}} + \Delta_u$. The probability that $u_i(t) > \tau_{\text{vac}}$ is $\beta_d = 1 - F(\tau_{\text{vac}}; u_{\text{nom}} + \Delta_u) > 0$ since $\tau_{\text{vac}} < u_{\text{nom}} + \Delta_u$. For a Poisson attack process with rate λ, the expected number of attacked timesteps in $[0, T]$ is λT, and each is independently detected with probability β_d. The probability of missing *all* detection opportunities is at most $e^{-\lambda T \beta_d}$, yielding the stated bound. □

Theorem 4 (Consensus Resilience)**.** *(Ensures that quarantining compromised agents does not prevent the honest subswarm from reaching agreement.)* If at most f agents are quarantined and the remaining subgraph $\mathcal{G}'(t)$ is $(2f+1)$-connected, then the trust-weighted consensus protocol converges to the honest-agent average with error $O(\zeta)$.

Proof sketch. Quarantined agents have $\tau_{ij} < \zeta$ for all honest neighbors i, so their contribution to any trust-weighted average is bounded by $\zeta / (\sum_j \tau_{ij})$, which vanishes as $\zeta \to 0$. The remaining $(N - f)$ honest agents form a $(2f+1)$-connected subgraph. By classical results on consensus under symmetric, connected, doubly-stochastic weight matrices [37], iterative trust-weighted averaging over this subgraph converges to the uniform average of honest states. The residual influence of quarantined agents introduces a bias of $O(\zeta)$ per iteration, yielding the stated bound. □





USENIX VehicleSec '26 Artifact Appendix: SHIELD: Secure Human-Machine Interaction with Evidential Learning and Dynamic Trust for Drone Swarm Control

Dawood Wasif[1], Terrence J. Moore[2], Seunghyun Yoon[3], Hyuk Lim[3], Dan Dongseong Kim[4], Frederica F. Nelson[2], Jin-Hee Cho[1]

[1]Virginia Tech [2]U.S. Army Research Laboratory [3]Korea Institute of Energy Technology
[4]University of Queensland

A Artifact Appendix

A.1 Abstract

This artifact provides the complete implementation of SHIELD, a framework for secure human–swarm teaming in which evidential vacuity (from Dirichlet-based perception) jointly drives inter-agent trust modulation, intrusion detection, and human escalation within a decentralized control loop. It contains: (1) the SHIELD framework implementing Algorithm 1 (evidential perception, adaptive trust modeling, trust-weighted consensus, KL-divergence safety arbitration, and resilient agent lifecycle management); (2) an Intrusion Response System (IRS) with all six cyber-physical attack classes (A1–A6, Table 3); (3) four baselines (Baseline DRL/APPO, COACH, TAMER, SAHO); and (4) training, evaluation, and plotting scripts that reproduce all figures and tables. The artifact supports the paper's claims C1–C4 (see Major Claims below). This artifact was awarded the *Artifacts Available* and *Artifacts Functional* badges.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact is defensive and poses no security or privacy risk to evaluators. All experiments run entirely in simulation using the open-source QuadSwarm environment [1]; no physical UAV deployments, real communication disruptions, or human subjects are involved. The six attack classes are implemented as parameterised perturbations inside the simulator and cannot affect real systems. No personal data is collected or processed.

A.2.2 How to access

The artifact is permanently archived on Zenodo with a stable DOI: **10.5281/zenodo.20208768** (https://doi.org/10.5281/zenodo.20208768).

A.2.3 Badges

This artifact was awarded *Artifacts Available* (publicly archived on Zenodo with a stable DOI) and *Artifacts Functional* (the artifact includes all components described in the paper—Algorithm 1, all six attack classes, all four baselines, all three experimental protocols—with documentation and end-to-end scripts that produce the reported results).

A.2.4 Hardware dependencies

An NVIDIA GPU with ≥8 GB VRAM (e.g., RTX 3080 or A100, CUDA 11.7+); ≥8 CPU cores (Sample Factory uses multiple workers); ≥32 GB RAM; and ≥20 GB disk for checkpoints and logs. Per-agent inference latency is 4.2 ms with 8 UAVs (Table 11), within the 20 ms control budget at 50 Hz.

A.2.5 Software dependencies

Ubuntu 22.04 and Python 3.11 (system packages `build-essential`, `python3.11`, `python3.11-venv`, `python3.11-dev`). All Python dependencies install via `pip install -e .`, the authoritative dependency path for the artifact.

A.2.6 Benchmarks

All scenarios are generated procedurally by the QuadSwarm simulator [1] (50 Hz; σ_{GPS}=0.3 m, σ_{IMU}=0.01 rad/s; 20×20×10 m arena); no external datasets are required. The six attack classes (A1–A6) are parameterised per Table 3 and injected by the `AttackInjector` module.

A.3 Set-up

A.3.1 Installation

```
# fetch the code from the Zenodo DOI, then:
cd SHIELD
```

```
python3.11 -m venv shield_env
source shield_env/bin/activate
pip install -U pip setuptools wheel
pip install -e .
# verify component registration:
python -c "from swarm_rl.train import \
 register_swarm_components as reg; \
 reg(); print('OK')"
```

A.3.2 Basic Test

A short training run confirms the environment and SHIELD wrapper function:

```
python -m swarm_rl.train \
 --env=quadrotor_multi \
 --hmt_approach=shield \
 --quads_num_agents=8 \
 --quads_neighbor_visible_num=0 \
 --train_for_env_steps=100000 \
 --experiment=sanity_check
```

Expected: training starts, prints the SHIELD parameter summary (Table 16: τ_{vac}=0.4, κ=0.1, β=0.05, ζ=0.2, ε=0.3, δ=0.15), and writes TensorBoard logs to `./train_dir/`.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Nominal performance. SHIELD attains higher Survival Rate (∼82–86%) and Mission Success Rate (∼3.5–4.0%), with comparable or better Crash Penalty, versus Baseline DRL, COACH, SAHO, and TAMER in open- and dense-field settings. Proven by (E1); see Section 6.1 (Figure 2) and Tables 4–6.

(C2): Intrusion detection. SHIELD +IRS achieves 82.5% Detection Rate, 14.6% False Positive Rate, 9.3 s Recovery Time, and 13.5 Human Interventions/episode, outperforming all partial variants. Proven by (E2); see Section 6.2, Table 2.

(C3): Operator workload. SHIELD +IRS reduces human confirmations by 53% versus the no-IRS variant (13.5 vs. 28.7/episode). Proven by (E2); see Section 6.2, Table 2.

(C4): Complementary modules. Removing any single module (vacuity escalation, trust updates, or safety override) causes monotonic degradation. Proven by (E1) and (E3); see Section 6.3, Table 4.

A.4.2 Experiments

(E1): *Nominal benchmarking* [∼20–40 compute-hours]: trains all five methods for 2.5×10^7 steps (Table 16 hyperparameters), open field via `bash train_local_hmt_comparison.sh` and dense field via `train_local_hmt_comparison_obstacles.sh`. Monitor with `tensorboard --logdir=./train_dir/`; test-time metrics (30 trials, final checkpoint) reproduce Figure 2 and Tables 4–6 within the reported standard deviations (validates C1, C4).

(E2): *Intrusion robustness* [∼20–40 compute-hours]: `bash train_irs_security_evaluation.sh` activates A1–A6 (Table 3) in the dense field with the IRS enabled, then `python -m swarm_rl.plot_irs_results` summarises DR/FPR/RT/HIC, reproducing Table 2 (e.g., DR 82.5±3.45%) (validates C2, C3).

(E3): *Ablation & scalability* [∼20–40 compute-hours]: disable modules with `--shield_tau_vac=100` (w/o vacuity), `--shield_zeta=0` (w/o trust), and `--shield_epsilon=100` (w/o safety override); vary `--quads_num_agents`$\in\{8,16,32\}$ with `--quads_use_obstacles=True`. Expect monotonic degradation (Table 4) and stable SR across N (Table 6) (validates C4).

A.5 Notes on Reusability

SHIELD is modular and extensible: new attack classes extend `AttackInjector.step()` (`irs_security_evaluation.py`); custom trust rules replace `ShieldNode.update_trust()` (`node.py`); alternative human interfaces implement the `HumanOracle` API (`human_interface.py`); other Gymnasium environments exposing per-agent positions can replace QuadSwarm; and all SHIELD parameters (Table 16) are command-line arguments sweepable via the shell scripts.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.

References

[1] Z. Huang, S. Batra, T. Chen, R. Krupani, T. Kumar, A. Molchanov, A. Petrenko, J. A. Preiss, Z. Yang, and G. S. Sukhatme, "QuadSwarm: A modular multi-quadrotor simulator for deep reinforcement learning with direct thrust control," *arXiv preprint arXiv:2306.09537*, 2023.

SLEIGHT: Real-Time Closed-Loop Attack Analytics Framework for Holistic Intrusion Detection Systems in Unmanned Aerial Vehicles

Muneeba Asif
Florida International University

Maryna Veksler
Virginia Commonwealth University

Devrim Aras
Florida International University

Kemal Akkaya
Virginia Commonwealth University

Mohammad Ashiqur Rahman
Florida International University

Abstract

Unmanned aerial vehicles (UAVs) are increasingly secured by advanced intrusion detection systems (IDSs) that monitor the holistic sensor–control–actuator pipeline. However, the robustness of these IDSs has not been systematically characterized against adaptive, real-time, physically plausible, and stealthy telemetry perturbations during closed-loop flight. Prior adversarial studies often rely on open-loop, offline testing, failing to capture the temporal, dynamic, and closed-loop constraints inherent to UAV operation. Consequently, it remains unclear what mission-level impact an in-flight data-stream attacker can induce while maintaining benign-looking behavior. We propose SLEIGHT, a novel, real-time, attack analytics framework that maps the stealth–impact frontier of closed-loop UAV IDSs under physically plausible perturbations. SLEIGHT trains a phase-conditioned reinforcement-learning policy with attacker-side plausibility enforcement to synthesize bounded perturbations maximizing mission impact under stealth and feasibility constraints. Using Pixhawk hardware- and software-in-the-loop experiments with MAVSDK, SLEIGHT benchmarks four IDSs, including state-of-the-art controller- and actuator-aware holistic IDSs such as PID-Piper and ConFIDe. SLEIGHT consistently achieves high no-alarm rates across diverse and multi-UAV missions (>96% in single-channel GPS, sustaining >95% and >90% against ConFIDe and PID-Piper, respectively) while inducing trajectory drift and landing displacement up to ≈11 m in single UAV HITL and > 60 m in swarm scenarios.

1 Introduction

Unmanned aerial vehicles (UAVs) are increasingly deployed in disaster response, defense, agriculture, infrastructure inspection, and environmental monitoring [30, 43]. Modern UAVs achieve autonomy through a closed-loop pipeline that couples sensing, state estimation, control, and actuation. At each control cycle, navigation-related measurements such as position, velocity, attitude, and altitude are fused by the estimator and used by the controller to compute actuator commands [12] (Fig. 1). This measurement-driven loop creates a practical cyber-physical attack surface, i.e., if an attacker rewrites MAVLink state messages consumed by an offboard controller (e.g., via a compromised companion relay or MAVLink router on the autopilot–offboard link), small navigation biases can propagate through feedback to cause mission-level effects such as trajectory drift, unsafe separation, or off-target landing [5, 21, 22].

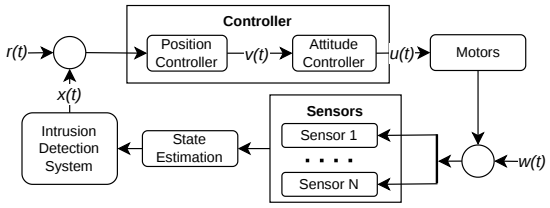


Figure 1: UAV closed-loop and measurement-driven operation.

As UAVs increasingly operate in beyond-visual-line-of-sight (BVLOS) settings with minimal human oversight, intrusion detection systems (IDSs) are widely used to detect anomalous behavior and trigger failsafe responses [19, 37]. Recent UAV IDSs are holistic, combining multivariate signals spanning sensors, estimator outputs, controller variables, and actuator behavior [6, 8, 10, 41]. Such fusion can improve robustness against single-channel spoofing and simple cross-signal inconsistencies [10, 17]. However, robustness against adaptive, real-time telemetry manipulation in closed-loop flight remains insufficiently characterized; for instance, perturbations that stay below alarm thresholds at each step can still accumulate over long horizons into meaningful operational as well as safety risk.

A central reason is that many adversarial evaluations rely on offline datasets or open-loop perturbations [11, 14]. These settings miss temporal dependencies, estimator dynamics, and feedback effects that determine feasibility and detectability during flight. Thus, it is difficult to quantify the boundary between *stealth* (remaining within detector tolerance) and *impact* (inducing mission-level deviation) under realistic access and feasibility constraints. Moreover, studies often report either detection outcomes or disruption metrics in isolation,

rather than their joint tradeoff under a thresholded decision rule [42, 44]. Concretely, we lack a principled way to answer that under physically plausible telemetry manipulation, how much mission deviation can an attacker induce while remaining nominal-looking to a deployed IDS over an entire flight?

Hence, we present **SLEIGHT** (Synthesizing Legitimate-looking Evasive Inputs to Generate Hidden Threats), a closed-loop attack analytics framework that maps the stealth–impact frontier of UAV IDSs under physically plausible telemetry perturbations. As shown in Fig. 2, bounded telemetry rewrites at the autopilot–offboard interface lead to a gradual navigation shift while remaining below the IDS detection score. SLEIGHT uses stealthy perturbation generation as the mechanism for robustness evaluation; its primary objective is to characterize where UAV IDSs remain robust, and where bounded closed-loop telemetry manipulation can still induce mission-level deviation. SLEIGHT enables systematic evaluation across single- and coordinated multi-sensor manipulations, varying flight and environmental complexities, and multi-UAV scenarios. SLEIGHT operates under a partial-information adversary operating along realistic data paths. The attacker can bias a limited subset of navigation-related channels via an onboard relay (e.g., a companion computer) that forwards flight telemetry (e.g., MAVLink), consistent with prior UAV communication attack and detection settings [5, 21, 22]. The attacker obtains benign-like telemetry through reconnaissance, as in prior UAV threat models [18] and CPS deception settings [33]. The threat model is formalized in Section 4.

To synthesize perturbations online, SLEIGHT combines phase-conditioned sequential learning with runtime plausibility enforcement. Offline, it learns (i) an attacker-side plausibility scorer from reconnaissance telemetry and (ii) a phase-conditioned sequential policy regularized by a realism prior to maximize an impact proxy under bounded feasibility constraints. Online, the policy is executed with lightweight per-step scaling and projection (via bisection on perturbation magnitude) to enforce plausibility and physical feasibility at each control cycle, enabling closed-loop, state-dependent adaptation without using deployed IDS scores or parameters.

We evaluate SLEIGHT in a closed-loop PX4 environment using MAVSDK, and validate key findings with hardware-/software-in-the-loop experiments on Pixhawk flight-control firmware. Across diverse single- and multi-UAV missions and four IDS architectures (including controller- and actuator-aware holistic IDSs), SLEIGHT maintains high no-alarm rates while inducing measurable trajectory drift and landing displacement (up to $\approx$11 m in our testbed). We also analyze robustness under practical uncertainty sources such as flight complexity, environmental conditions, threshold estimation error, and decision-boundary drift. Our results quantify the stealth-impact frontier of thresholded UAV IDSs under limited adversarial access and limited prior knowledge. We summarize our contributions as follows:

- We formalize closed-loop UAV IDS robustness analysis as stealth-constrained mission impact maximization, enabling direct quantification of the stealth-impact frontier for thresholded detectors under feasibility constraints.
- We present SLEIGHT, an attack analytics framework that synthesizes physically plausible, fusion-consistent perturbations for closed-loop IDS robustness evaluation across missions and multi-UAV scenarios.
- We conduct a comprehensive closed-loop evaluation showing consistent stealth–impact tradeoffs across several representative IDS designs, including mission-level effects under bounded, plausible perturbations.
- We provide hardware-/software-in-the-loop validation on flight-controller firmware and release a reproducible PX4 SITL + MAVSDK pipeline with scenario configurations to support future IDS benchmarking [39].

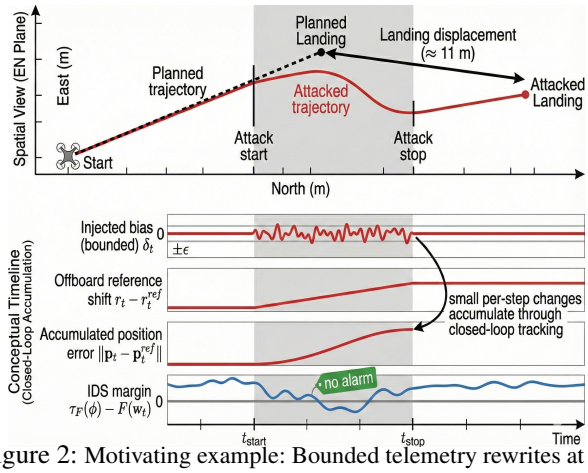


Figure 2: Motivating example: Bounded telemetry rewrites at the MAVLink interface bias offboard references, accumulating into landing displacement while the IDS stays below its alarm threshold.

2 Background

Here, we overview the closed-loop pipeline and its constraints, the IDS families evaluated in this work, and MAVLink telemetry mediation as a practical injection surface.

UAV Telemetry and Real-Time Pipelines: At control step t with period Δt, the stack produces a time-aligned telemetry vector $\mathbf{x}_t$ and updates estimation/control:

$$\mathbf{s}_t = \mathcal{E}(\mathbf{x}_{1:t}), \qquad \mathbf{u}_t = \mathcal{C}(\mathbf{s}_t, \mathbf{r}_t), \qquad \mathbf{z}_{t+1} = \mathcal{P}(\mathbf{z}_t, \mathbf{u}_t, \mathbf{w}_t)$$

Autopilots run nested loops at heterogeneous rates, with IMU-driven estimation and attitude/rate control faster than GPS navigation updates [32]. Thus, telemetry perturbations must be feasible per update: values must satisfy message type, unit, range, timing, and per-step dynamics expected by the estimator/controller. Closed-loop attackers must therefore synthesize within the targeted channel's per-step budget and maintain temporal coherence at the runtime update rate. Otherwise, the injected stream can lag or create rate/update inconsistencies, even if it appears benign offline. PX4 scheduling and configured publication rates make this a runtime constraint [3].

UAV IDS Families and Limitations: IDSs typically compute an anomaly score from telemetry data and issue an alarm when the score exceeds a threshold. A common class is reconstruction-based detection:

$$\ell_{\text{recon}}(\mathbf{x}_t) = \|\mathbf{x}_t - f_\theta(\mathbf{x}_t)\|_2^2, \qquad \text{alarm if } \ell_{\text{recon}}(\mathbf{x}_t) > \tau \quad (1)$$

Boundary-based one-class methods similarly define an acceptance region. Holistic designs augment features with controller and actuator signals, detecting via cross-signal consistency rather than sensor statistics alone. Our evaluation covers these families, including controller/actuator-aware PID-Piper [17] and ConFIDe [10]. Section 3 details this limitation in prior work.

Telemetry Injection Surface (MAVLink and Companion Mediation): Telemetry can be manipulated along MAVLink forwarding paths between the flight controller, GCS, and onboard components. MAVLink2 supports optional signing but no encryption; without signing, an on-path adversary can modify or replay protocol-valid telemetry [2]. We assume signing, or equivalent integrity protection, is absent on the targeted link; otherwise, endpoint/key compromise would be required and is out of scope. Signing hardens protected links against on-path modification, but does not prevent attacks from trusted endpoints, such as compromised companion processes. Thus, we treat integrity protection as hardening while still evaluating IDS robustness at realistic telemetry interfaces.

Companion computers use MAVLink to relay traffic among the autopilot, GCS, and peripherals [1, 3], making companion-mediated rewriting a realistic injection point. Prior work demonstrates MAVLink injection in SITL/Pixhawk HITL, and MUVIDS evaluates IDS detection under false MAVLink messages [22]. Although we implement attacks through MAVLink mediation, common in PX4 offboard/companion setups, SLEIGHT is not MAVLink-specific. The same enforcement and frontier analysis apply to other telemetry-consumer interfaces, such as ROS/middleware, by redefining x_t, M, and Π_Ω.

3 Related Works

We group prior work into four themes: (i) UAV sensor/protocol attack surfaces, (ii) adversarial ML for IDS evasion, (iii) CPS stealth–impact analysis/falsification, and (iv) holistic UAV IDSs that fuse sensing, control, and actuation.

Sensor-level attacks on UAVs: Dash et al. [16] learn invariant thresholds but do not model environmental variation. Khazraei et al. [25, 26] coordinate camera and sensor attacks, reporting reduced impact under sensor fusion. Tractor Beam [34] and Son et al. [45] demonstrate single-sensor spoofing (e.g., GPS, acoustic) with mission diversion, without end-to-end evaluation under multi-sensor fusion and closed-loop IDS monitoring. RVFuzzer [28] and MAYDAY [27] focus on input flaws or offline analysis and do not address adaptive, closed-loop synthesis. ConfuSense [18] shows persistent effects from a single malicious bus message. These works motivate realistic attack surfaces, but they do not quantify long-horizon mission deviation under bounded, temporally coherent perturbations while staying within the operating region of holistic UAV IDSs.

Adversarial ML for IDS evasion: Adversarial ML shows that learned models can be evaded under bounded perturbations. Kurakin et al. [29] study adversarial examples for time-series inputs in mostly static settings. In UAV security, Asif et al. [11] use adversarial learning for IDS hardening through retraining rather than attacks. Alshahrani et al. [7] and Sarikaya and Bahtiyar [38] consider GAN-generated IDS attacks but omit closed-loop constraints and multi-sensor coordination. Strickland et al. [40] combine GANs with RL for IDS improvement, but evaluate offline. Wang et al. [44] discuss telemetry manipulation, focusing on deviation metrics without closed-loop evasion analysis. Xue et al. [46] show GPS spoofing as a single-sensor weakness. Uludag et al. [42] exploit cross-sensor relations for deception, but not real-time closed-loop synthesis with per-step feasibility. Transferability studies [35] motivate attacker-side reconnaissance learning without deployed-model access.

IDS evasion and stealth–impact analysis in CPS: Prior work links stealth to operational impact via invariants (SAVIOR [36]), offline residual-driven objectives [24], or temporal-logic falsification [15]; related works in safe RL focus on policy safety rather than IDS evasion [23]. SLEIGHT provides a real-time, multi-sensor perturbation generator integrated with PX4 closed-loop control, with per-step feasibility enforcement and mission-level impact measurement.

Real-time and online evasion with reinforcement learning: Recent work demonstrates that RL agents can perturb streaming inputs to evade sequence-based anomaly detectors under bounded modification constraints (e.g., online evasion pipelines [20]). This motivates real-time evasion evaluation beyond offline replay. SLEIGHT specializes this direction to closed-loop UAV telemetry, where feasibility is defined by runtime message semantics and update rates, and where mission impact must be measured through closed-loop execution rather than offline score manipulation.

Holistic and fusion-aware UAV IDSs: ConFIDe [10] and PID-Piper [17] are holistic UAV IDSs that fuse sensing, control, and actuation signals to resist single-channel spoofing and simple inconsistencies, relying on offline-trained detection boundaries. Following community practice in evasion-oriented IDS evaluations that benchmark multiple detector families and operating points [13], we evaluate multiple IDSs, including holistic designs under closed-loop operation.

Differences: To address these limitations, SLEIGHT instantiates a closed-loop attack analytics pipeline in PX4 that (i) learns attacker-side models from reconnaissance telemetry, (ii) proposes phase-conditioned perturbation directions, and (iii) enforces attacker-side plausibility and feasibility constraints at runtime via maximal-feasible scaling, without querying deployed IDS scores during synthesis. We then

quantify the resulting stealth–impact frontier across missions, multi-sensor and multi-UAV settings, and multiple IDS families, including holistic controller- and actuator-aware designs.

4 Threat Model and Problem Definition

We study stealth-constrained telemetry-integrity attacks in closed-loop flight, where an adversary perturbs limited navigation fields while an IDS monitors the sensor-control-actuation pipeline. We distinguish the deployed IDS score $F(\cdot)$, used only for evaluation, from the attacker-side plausibility score $S(\cdot)$, used for synthesis and runtime enforcement; the attacker never observes, queries, or optimizes $F(\cdot)$ online.

4.1 System Model

At control step $t \in \{1,\dots,T\}$ with period Δt, the system produces telemetry $\mathbf{x}_t \in \mathbb{R}^d$ and updates estimation/control in closed loop as follows:

$$\mathbf{s}_{t+1} = \mathcal{F}(\mathbf{s}_t, \mathbf{u}_t, \text{env}), \qquad \mathbf{u}_t = \pi(\mathbf{s}_t, \mathbf{r}_t) \tag{2}$$

In our PX4+MAVSDK setting, mission logic runs as an offboard control process that consumes a recent telemetry window and outputs references $\mathbf{r}_t$ (e.g., position/velocity setpoints) sent back to the autopilot. Thus, manipulating the telemetry stream that this offboard process consumes can alter $\mathbf{r}_t$ and induce mission-level deviation under closed-loop control. A deployed IDS computes an anomaly score over a recent window $\mathbf{w}_t = [\mathbf{x}_{t-W+1},\dots,\mathbf{x}_t] \in \mathbb{R}^{W\times d}$:

$$F(\mathbf{w}_t) \in \mathbb{R}_{\geq 0}, \qquad \text{alarm if } F(\mathbf{w}_t) > \tau_F(\phi(t)) \tag{3}$$

where $\phi(t) \in \{\text{takeoff, transit, hover, landing}\}$ is the flight phase. The thresholding rule in (3) models the deployed detector; it is *not* available to the attacker during synthesis. As one example, reconstruction-based IDSs instantiate $F(\cdot)$ as a windowed reconstruction error; our attacker does not observe F and does not tune attacks to any specific F form.

4.2 Adversary Capabilities and Restrictions

Placement: We model an in-path adversary that can read and rewrite MAVLink messages between the autopilot and the offboard controller (and co-located IDS), e.g., via a compromised relay or MITM. This aligns with prior work demonstrating false MAVLink injection in SITL/HITL settings [22].

Manipulation: Let $\mathbf{M} \in \{0,1\}^d$ denote a mask over attackable channels, where $\mathbf{M}$ is a strict subset of the IDS feature set. For example, it covers navigation-related channels, while actuators/PWM/control variables used by holistic IDSs are never modified. At each step, the attacker injects

$$\mathbf{x}'_t = \mathbf{x}_t + \mathbf{M} \odot \delta_t \tag{4}$$

where δ_t is bounded. Attacks may be single-channel (e.g., GPS position) or coordinated across multiple channels (e.g., GPS altitude and barometric altitude) to preserve estimator-level consistency. The attacker preserves protocol-valid formatting and field semantics for modified messages.

Restrictions: The attacker does not modify flight firmware, estimator/controller code, safety logic, mission plan/waypoints, cryptographic keys, or the IDS implementation, or assume RF denial (e.g., jamming). Attacker does not access IDS's internals or query the deployed IDS score $F(\cdot)$ online. We assume that MAVLink2 optional message signing is not enabled on the targeted communication link; when signing is configured, an attack would require endpoint/key compromise, which is outside the scope of this work. As such, attackers target systems where signing is disabled, inconsistently deployed, or not enforced end-to-end. Although signing is recommended for secure deployment, it requires explicit key provisioning and coordination across components, introducing operational overhead [4]. This can result in inconsistent implementations, particularly in SITL/HITL and companion-mediated pipelines.

4.3 Reconnaissance and Attacker Plausibility

Because the attacker does not observe $F(\cdot)$, we parameterize *benign-looking* behavior using an attacker-side plausibility model trained from benign-like data on the attackable channels. The attacker is assumed to obtain benign telemetry $\mathcal{D}_{\text{benign}}$ for the attackable channels through reconnaissance (e.g., controlled collection on a comparable platform or observation of operationally similar flights). Using this benign dataset, the attacker trains a plausibility scorer:

$$S(\text{vec}(\mathbf{w}_t)) \in \mathbb{R}_{\geq 0} \tag{5}$$

where $\text{vec}(\cdot)$ flattens a window. SLEIGHT is agnostic to the specific plausibility scorer ($S(\cdot)$); we instantiate it with an AE residual, but any one-class score (e.g., OCSVM/SVDD or density/predictor residuals) can be substituted without changing Alg. 1 or (11). The attacker calibrates ($\tau_A(\phi)$) from benign reconnaissance via (6), while robustness is evaluated solely against the deployed IDS ($F(\cdot)$).Attacker sets operational stealth threshold from benign quantiles (phase-conditioned):

$$\tau_A(\phi) = \text{Quantile}_q\left(\{S(\text{vec}(\mathbf{w})) : \mathbf{w} \in \mathcal{D}_{\text{benign}}, \phi(\mathbf{w}) = \phi\}\right) \tag{6}$$

The attacker estimates $\phi(t)$ from the observable window; it is not assumed as a hidden label. To model realistic uncertainty, we evaluate calibration mismatch by varying (i) the amount of reconnaissance data used to fit $S(\cdot)$ and $\tau_A(\cdot)$, (ii) biased or noisy $\tau_A(\cdot)$ estimates, and (iii) phase-estimation errors.

Reconnaissance mismatch can reduce reliability. If target flights differ from reconnaissance data due to wind, sensor noise, payload, or dynamics, the attacker-side score distribution may shift. This can increase the time required to find feasible perturbations and may increase exposure risk when $\tau_A(\phi)$ is overestimated. SLEIGHT mitigates this by tightening thresholds via κ and by optionally rolling and recalibrating from recent benign-looking windows.

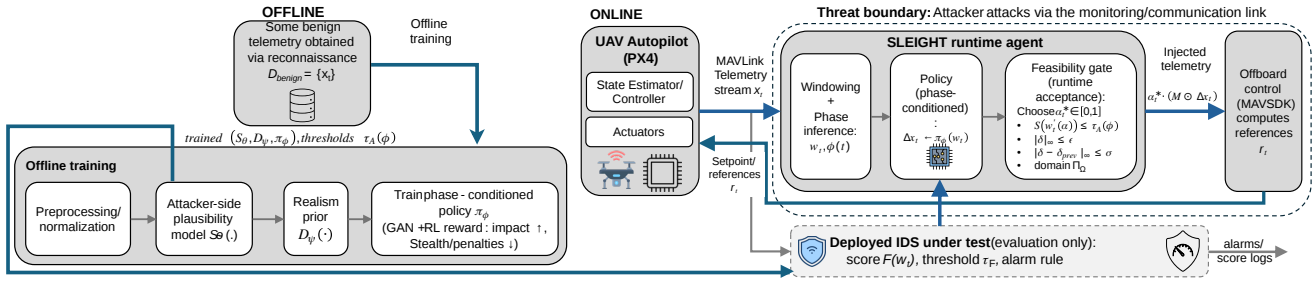


Figure 3: SLEIGHT overview: Offline, benign telemetry trains preprocessing, the plausibility model ($S(\cdot)$), realism prior ($D(\cdot)$), and a phase-conditioned perturbation policy. Online, the policy proposes a direction and selects the largest feasible scale ($\alpha_t^\star$) before injection.

4.4 Stealth and Feasibility Constraints

SLEIGHT enforces stealth w.r.t the attacker-side score:

$$S(\mathrm{vec}(\mathbf{w}_t')) \le \tau_A(\phi(t)) \quad \forall t \tag{7}$$

where $\mathbf{w}_t' = [\mathbf{x}_{t-W+1}', \ldots, \mathbf{x}_t']$. To reduce stealth failures under reconnaissance mismatch, the attacker can enforce a tightened threshold ($\tau_A'(\phi) = (1-\kappa)\tau_A(\phi)$) ($(\kappa \in [0,1))$), trading impact for robustness. If benign-like segments are observable online, ($\tau_A(\phi)$) can also be updated via rolling quantiles of recent ($S(\cdot)$) scores We enforce bounded and smooth perturbations in (8) and project injected values onto a valid domain Ω before encoding in (9):

$$\|\mathbf{M} \odot \delta_t\|_\infty \le \varepsilon, \qquad \|\mathbf{M} \odot (\delta_t - \delta_{t-1})\|_\infty \le \sigma \tag{8}$$

$$\mathbf{x}_t' \leftarrow \Pi_\Omega(\mathbf{x}_t') \tag{9}$$

At runtime, SLEIGHT proposes a perturbation direction $\Delta\mathbf{x}_t$ and applies a scale $\alpha \in [0,1]$:

$$\mathbf{x}_t'(\alpha) = \mathbf{x}_t + \alpha(\mathbf{M} \odot \Delta\mathbf{x}_t) \tag{10}$$

SLEIGHT chooses the maximal feasible scale

$$\begin{aligned} \alpha_t^\star = \max_{\alpha \in [0,1]} \alpha \quad & \text{s.t.} S\big(\mathrm{vec}(\mathbf{w}_t'(\alpha))\big) \le \tau_A(\phi(t)), \\ & \big\|\mathbf{M} \odot \delta_t(\alpha)\big\|_\infty \le \varepsilon, \big\|\mathbf{M} \odot \big(\delta_t(\alpha) - \delta_{t-1}\big)\big\|_\infty \le \sigma, \\ & \mathbf{x}_t'(\alpha) \in \Omega \end{aligned} \tag{11}$$

where $\delta_t(\alpha)$ is the induced perturbation in (4) under scaling (10). If no $\alpha > 0$ satisfies the constraints for a given step, SLEIGHT injects $\mathbf{x}_t$ (i.e., $\alpha_t^\star = 0$) unmodified, accepting zero impact at that step, instead of risking detection.

Objective and Metrics: The attacker maximizes mission-level deviation over the horizon subject to (7)-(11). Impact is measured via closed-loop outcomes (trajectory/landing deviation; for multi-UAVs, minimum pairwise separation). We use the term anomaly score only for IDS outputs such as reconstruction error or CUSUM residuals, whereas trajectory deviation, landing displacement, and trajectory-deviation MSE denote physical mission-impact metrics rather than detector scores. IDS robustness uses the deployed alarm rule (3), reporting per-step and per-flight no-alarm rates. Attacker models (S, τ_A), budgets (ε, σ), and the SLEIGHT pipeline are held fixed across IDSs; only evaluation changes via the deployed $F(\cdot)$ and $\tau_F(\cdot)$.

5 Technical Details of SLEIGHT Framework

SLEIGHT synthesizes and injects bounded telemetry perturbations under the threat model in Section 4. At step t, $\mathbf{x}_t \in \mathbb{R}^d$ denotes the feature vector constructed from MAVLink messages consumed by the offboard control process to compute references $\mathbf{r}_t$ in the PX4+MAVSDK loop (e.g., position, attitude, etc.) We write the recent window as $\mathbf{w}_t = [\mathbf{x}_{t-W+1}, \ldots, \mathbf{x}_t] \in \mathbb{R}^{W \times d}$ and inferred flight phase as $\phi(t) \in \{\text{takeoff}, \text{transit}, \text{hover}, \text{landing}\}$. The attack mask is $\mathbf{M} \in \{0,1\}^d$. SLEIGHT uses attacker-side plausibility score $S(\cdot)$ and a realism prior $D(\cdot)$ during training and online enforcement. Deployed IDS scores $F(\cdot)$ are used only for evaluation.

5.1 Overall Architecture and Workflow

SLEIGHT operates in two stages. **Offline training:** Given benign telemetry traces $\mathcal{D}_{\text{benign}} = \{\mathbf{x}_t\}$, SLEIGHT: (i) fits preprocessing (normalization and per-channel scaling) used consistently by all attacker-side models, (ii) trains an attacker-side plausibility scorer $S_\theta(\cdot)$ over windows $\mathbf{w}_t$, (iii) trains a realism prior $D_\psi(\cdot)$ over windows to score benign-likeness, (iv) trains a phase-conditioned sequential policy π_φ that proposes a per-step perturbation direction $\Delta\mathbf{x}_t$ from the current window representation and phase.

Online synthesis and injection: At each step, as shown in Fig. 3, SLEIGHT receives the current window $\mathbf{w}_t$ and phase estimate $\phi(t)$, samples a perturbation direction $\Delta\mathbf{x}_t \leftarrow \pi_\varphi(\cdot)$, and applies runtime acceptance to select the maximal scale $\alpha_t^\star \in [0,1]$ satisfying the attacker-side stealth constraint, feasibility bounds, and domain constraints. The injected telemetry is $\mathbf{x}_t' = \mathbf{x}_t + \alpha_t^\star(\mathbf{M} \odot \Delta\mathbf{x}_t)$ after domain projection, and it is written into the MAVLink stream consumed by the offboard process so that $\mathbf{r}_t$ is computed from the perturbed window. At runtime, the feasibility gate acts as a post-processor over the policy's proposed perturbation direction Δx_t, i.e., rather than injecting Δx_t directly, SLEIGHT scales and (if needed) projects the proposal to ensure all feasibility/stealth constraints in (7)- (11) are satisfied at each time step.

5.2 Telemetry Representation and Plausibility Enforcement

Telemetry Representation and Domain Projection: Let $\Omega \subset \mathbb{R}^d$ denote the set of valid values induced by message semantics and physical/interface limits (units, quantization, and fixed enumerations). SLEIGHT enforces validity via

$$\mathbf{x} \leftarrow \Pi_\Omega(\mathbf{x}) \tag{12}$$

implemented as per-channel clamping in the model-scaled space, followed by inverse scaling prior to message encoding. The same preprocessing and inverse mapping are used during offline rollouts and online injection.

Attacker-Side Plausibility Model $S(\cdot)$: Plausibility scorer $S_\theta(\cdot)$ assigns a nonnegative score to each window, with lower values corresponding to benign-like windows. We implement the plausibility scorer $S_\theta(\cdot)$ as a window reconstruction residual. We train an autoencoder $A_\theta(\cdot)$ on benign windows:

$$\min_\theta \ \mathbb{E}_{w\sim\mathcal{D}_{\text{benign}}}\left[\|\text{vec}(w) - A_\theta(\text{vec}(w))\|_2^2\right] \tag{13}$$

Plausibility score is: $S_\theta(\text{vec}(w)) = \|\text{vec}(w) - A_\theta(\text{vec}(w))\|_2^2$, where lower values correspond to more benign-like windows. The operational threshold $\tau_A(\phi)$ is selected from phase-conditioned benign quantiles as in (6). Online enforcement evaluates S_θ on the modified window w'_t.

Realism Prior $D(\cdot)$: The realism prior $D_\psi(\cdot)$ provides a continuous shaping signal within the benign region enforced by $S_\theta(\cdot)$. We train D_ψ as a discriminator over windows using benign windows as positive samples and bounded synthetic perturbations as negative samples. Negative samples are constructed by sampling δ under the same mask and feasibility bounds used at runtime and applying the projection $\Pi_\Omega(\cdot)$. The realism shaping term is:

$$R_{\text{real}}(\mathbf{w}'_t) = \log\big(\max(D_\psi(\text{vec}(\mathbf{w}'_t)), \varepsilon)\big) \tag{14}$$

with $\varepsilon > 0$. R_{real} is used during offline policy training.

5.3 Online Attack Synthesis and Injection

Policy Output, Attack Formation, and Online Injection: The policy π_φ outputs a raw perturbation direction $\Delta\mathbf{x}_t \in \mathbb{R}^d$ in the model-scaled space. Each component of $\Delta\mathbf{x}_t$ is clamped to a fixed range before masking. For a candidate scale $\alpha \in [0,1]$, SLEIGHT forms the following:

$$\mathbf{x}'_t(\alpha) = \mathbf{x}_t + \alpha\big(\mathbf{M} \odot \Delta\mathbf{x}_t\big) \tag{15}$$

then applies $\mathbf{x}'_t(\alpha) \leftarrow \Pi_\Omega(\mathbf{x}'_t(\alpha))$.

Runtime Acceptance and Maximal Feasible Scaling: Runtime acceptance selects the maximal feasible scale $\alpha_t^\star$:

$$\alpha_t^\star = \max_{\alpha\in[0,1]} \ \alpha \quad \text{s.t.} \quad \begin{cases} S_\theta\big(\text{vec}(\mathbf{w}'_t(\alpha))\big) \le \tau_A(\phi(t)), \\ \|\mathbf{M} \odot \delta_t(\alpha)\|_\infty \le \varepsilon, \\ \|\mathbf{M} \odot (\delta_t(\alpha) - \delta_{t-1})\|_\infty \le \sigma, \\ \mathbf{x}'_t(\alpha) \in \Omega \end{cases} \tag{16}$$

where $\delta_t(\alpha) = \alpha(\mathbf{M} \odot \Delta\mathbf{x}_t)$ and $\mathbf{w}'_t(\alpha) = [\mathbf{x}'_{t-W+1}, \ldots, \mathbf{x}'_t(\alpha)]$. If no $\alpha > 0$ satisfies the constraints, it injects $\mathbf{x}_t$ for that step. In implementation, (16) is solved by bisection with a fixed iteration budget B. Each iteration evaluates $S_\theta(\cdot)$ on the candidate window, applies $\Pi_\Omega(\cdot)$, and checks the ℓ_∞ bounds.

5.4 Policy Training and Optimization

Training Objective: SLEIGHT trains π_φ offline using episodic rollouts in closed-loop execution. Training optimizes an impact proxy in feature space, while mission impact is measured during evaluation using closed-loop outcomes (trajectory deviation and landing displacement). We define the per-step feature deviation proxy:

$$\Delta_{\text{feat}}(\mathbf{x}'_t, \mathbf{x}_t) = \|\mathbf{M} \odot (\mathbf{x}'_t - \mathbf{x}_t)\|_2 \tag{17}$$

An optional cap term penalizes excessive deviations:

$$P_{\text{cap}}(\mathbf{x}'_t, \mathbf{x}_t) = \max\big(0,\ \Delta_{\text{feat}}(\mathbf{x}'_t, \mathbf{x}_t) - \Delta_{\max}\big) \tag{18}$$

Let $\text{Viol}(\mathbf{x}'_t(\alpha))$ denote pre-acceptance feasibility violations for candidate α during training rollouts (zero when all checks pass). The per-step reward used in training is:

$$\begin{aligned} r_t = {} & \lambda_I \Delta_{\text{feat}}(\mathbf{x}'_t, \mathbf{x}_t) + \lambda_R R_{\text{real}}(\mathbf{w}'_t) \\ & - \lambda_S \left[S_\theta(\text{vec}(\mathbf{w}'_t)) - \tau_A(\phi(t))\right]_+ \\ & - \lambda_C P_{\text{cap}}(\mathbf{x}'_t, \mathbf{x}_t) - \lambda_F \,\text{Viol}(\mathbf{x}'_t) \end{aligned} \tag{19}$$

where $[u]_+ = \max(u, 0)$ and $\lambda_I, \lambda_R, \lambda_S, \lambda_C, \lambda_F \ge 0$ are fixed weights. The stealth term depends only on $S_\theta(\cdot)$ and $\tau_A(\cdot)$.

Policy Optimization: We optimize π_φ using a policy gradient estimator on rollout returns. Attacker state includes flattened window, phase, and previous action:

$$\mathbf{o}_t = \big[\text{vec}(\mathbf{w}_t),\ \text{onehot}(\phi(t)),\ \Delta\mathbf{x}_{t-1}\big] \tag{20}$$

The action is the proposed direction $\Delta\mathbf{x}_t \sim \pi_\varphi(\cdot \mid \mathbf{o}_t)$. With return $\hat{G}_t = \sum_{k=t}^T \gamma^{k-t} r_k$, the gradient estimate is:

$$\nabla_\varphi J(\varphi) \approx \sum_{t=1}^T \nabla_\varphi \log \pi_\varphi(\Delta\mathbf{x}_t \mid \mathbf{o}_t)\ \hat{G}_t \tag{21}$$

6 Implementation

We implement SLEIGHT on a PX4-based stack in software- and hardware-in-the-loop (SITL/HITL). The attacker runs as a user-space MAVLink mediation proxy on the autopilot to the offboard-control path in Section 4.1. The proxy forwards MAVLink traffic while rewriting a restricted subset of navigation-related fields. All rewrites preserve MAVLink message IDs, field encodings, and original stream rates. Offboard control process computes references $\mathbf{r}_t$ from rewritten stream, so perturbations propagate through closed loop without modifying PX4 firmware.

SITL/HITL Stack (1) SITL: We run PX4 SITL with JMAVSIM and connect over UDP. Missions are executed using MAVSDK offboard control. The mediation proxy runs on the same host and forwards MAVLink traffic between PX4 and the MAVSDK process. **(2) HITL:** Pixhawk hardware is connected to a companion host over a telemetry link. Mediation proxy runs on the companion and forwards MAVLink traffic between Pixhawk and offboard control process. Feature construction/runtime enforcement match SITL configuration. Simulator roles. We use JMavSim for the primary closed-loop SITL evaluations because it is the lightweight PX4 backend used with our MAVSDK offboard loop. Gazebo is used only for environmental-stress experiments with wind and sensor noise. We do not pool numerical results across simulators; claims from nominal closed-loop missions are scoped to JMavSim, while environmental-robustness results are scoped to Gazebo.

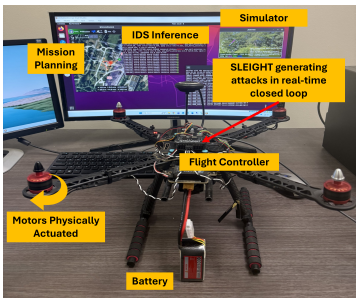


Figure 4: HITL setup: Pixhawk runs PX4, streams MAVLink over telemetry link. SLEIGHT proxy mediates the MAVLink autopilot-to-offboard control stream on companion host.

Telemetry Features and Injection Interface: At each step t, SLEIGHT builds $\mathbf{x}_t \in \mathbb{R}^d$ from the MAVLink state stream consumed by the offboard controller, using navigation messages such as `LOCAL_POSITION_NED` and `ATTITUDE`, with GPS fields from `GPS_RAW_INT` when enabled. For holistic baselines, we also ingest the required actuator-output and estimator-status fields. Feature order follows the attacker-fitted scaler, and each step forms one synchronized row from the latest required messages before scaling. PX4 publication rates are set through `MAV_CMD_SET_MESSAGE_INTERVAL`: navigation messages are requested at 10 Hz, and GPS/estimator-status messages at 5 Hz, with rates unchanged during injection. The mask $\mathbf{M} \in \{0,1\}^d$ selects attackable channels. Injection is performed by the mediation proxy, which rewrites only the corresponding outbound MAVLink fields in $\mathbf{x}'_t$ and forwards all non-masked fields unchanged.

Runtime Enforcement and Online Loop: Online injection uses only attacker-side models. At each step, the proxy maintains a sliding window $\mathbf{w}_t = [\mathbf{x}_{t-W+1}, \ldots, \mathbf{x}_t]$, applies attacker-fitted scaling, and calls the policy to propose a direction $\Delta\mathbf{x}_t$. Runtime acceptance then selects the maximal feasible scale $\alpha_t^\star$ under attacker-side stealth constraint and feasibility bounds, using bisection with iteration budget B.

Runtime Cost and Logging: The proxy logs per-step runtime `step_time` from wall-clock timestamps, as shown in Alg. 1, together with the fixed bisection budget B. Logs include attacker-side plausibility scores for benign and injected windows, $\alpha_t^\star$, ℓ_∞ perturbation and smoothness norms, phase estimates, and PX4/MAVSDK mission outcomes such as trajectory deviation, landing displacement, and minimum pairwise separation in multi-UAV runs. All norms use the same scaled feature space as $S_\theta(\cdot)$ and $\Pi_\Omega(\cdot)$, and exported logs support replay and recomputation of stealth and impact metrics. Table 1 summarizes runtime hyperparameters; Appendix A provides full list/design-choice justifications. λ_C and λ_F are retained for ablation consistency; in the reported runs, smoothness/feasibility are enforced as runtime constraints through projection and bisection rather than as soft reward penalties.

Algorithm 1: SLEIGHT Runtime Operation

Input: current feature vector $\mathbf{x}_t$, window buffer $\mathbf{w}_t$, phase estimate $\phi(t)$, policy π_φ, plausibility model S_θ, threshold $\tau_A(\cdot)$, mask $\mathbf{M}$, bounds (ε, σ), projection Π_Ω, bisection budget B
Output: injected $\mathbf{x}'_t$ and per-step logs
while *mission is active* **do**
  receive MAVLink messages and assemble $\mathbf{x}_t$; update $\mathbf{w}_t$
  $\Delta\mathbf{x}_t \leftarrow \pi_\varphi(\mathrm{vec}(\mathbf{w}_t), \phi(t), \Delta\mathbf{x}_{t-1})$; clamp $\Delta\mathbf{x}_t$ in scaled space
  $\alpha_L \leftarrow 0$; $\alpha_U \leftarrow 1$
  for $i \leftarrow 1$ **to** B **do**
    $\alpha \leftarrow (\alpha_L + \alpha_U)/2$
    $\mathbf{x}'_t(\alpha) \leftarrow \Pi_\Omega(\mathbf{x}_t + \alpha(\mathbf{M} \odot \Delta\mathbf{x}_t))$
    form $\mathbf{w}'_t(\alpha) = [\mathbf{x}'_{t-W+1}, \ldots, \mathbf{x}'_t(\alpha)]$
    $s \leftarrow S_\theta(\mathrm{vec}(\mathbf{w}'_t(\alpha)))$
    $\delta_t(\alpha) \leftarrow \alpha(\mathbf{M} \odot \Delta\mathbf{x}_t)$
    if $s \le \tau_A(\phi(t))$ ***and*** $\|\delta_t(\alpha)\|_\infty \le \varepsilon$ ***and*** $\|\delta_t(\alpha) - \delta_{t-1}\|_\infty \le \sigma$ **then**
      $\alpha_L \leftarrow \alpha$
    else
      $\alpha_U \leftarrow \alpha$
    end
  end
  $\alpha_t^\star \leftarrow \alpha_L$
  if $\alpha_t^\star = 0$ **then**
    $\mathbf{x}'_t \leftarrow \mathbf{x}_t$
  else
    $\mathbf{x}'_t \leftarrow \Pi_\Omega(\mathbf{x}_t + \alpha_t^\star(\mathbf{M} \odot \Delta\mathbf{x}_t))$
  end
  rewrite the MAVLink fields corresponding to the attackable channels in $\mathbf{x}'_t$ and forward messages to the offboard control process
  define $\mathbf{w}'_t = [\mathbf{x}'_{t-W+1}, \ldots, \mathbf{x}'_t]$
  log $(t, \phi(t), S_\theta(\mathrm{vec}(\mathbf{w}_t)), S_\theta(\mathrm{vec}(\mathbf{w}'_t)), \tau_A(\phi(t)), \alpha_t^\star, \|\delta_t\|_\infty, \|\delta_t - \delta_{t-1}\|_\infty, \mathtt{step_time})$
end

7 Evaluation

We evaluate SLEIGHT in closed-loop flight using four metrics: (i) **Evasion Success (%)**, the per-window/timestep no-

Table 1: Runtime and training hyperparameters used in SLEIGHT.

Parameter	Value	Description
W	50	Telemetry window length used by the detector and attacker-side scorer
d	11	Feature dimensionality of each telemetry vector
B	8	Bisection budget for runtime perturbation scaling
γ	0.99	Discount factor used during policy training
$\alpha_{\max}$	0.25	Maximum runtime perturbation scale
κ	0.01	Safety margin below the attacker-side plausibility threshold
λ_I	1.0	Mission-impact reward weight
λ_R	1.0	Realism-prior reward weight
λ_S	50.0	Stealth/plausibility penalty weight
λ_C	0	Control-smoothness penalty weight
λ_F	0	Feasibility/projection penalty weight
f_{ctrl}	10 Hz	Runtime telemetry/control update rate

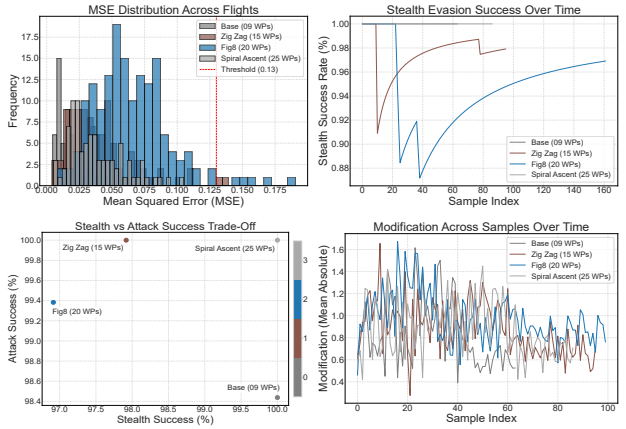


Figure 5: Performance of SLEIGHT with varying flight durations.

alarm rate under the deployed IDS; where applicable, we also report per-flight no-alarm rate to assess mission-level robustness. (ii) **Attack Success (%)** as the fraction of manipulations achieving the attack goal (e.g., sustained GPS offset or route deviation), (iii) **Perturbation Magnitude** as the mean/max $|\delta_t|$ per timestep and capped violations, and (iv) **Mission Impact** measured through trajectory deviation (RMSE between intended and actual paths), waypoint miss distance, and landing displacement. We also report runtime overhead and enforcement behavior (α_t^* distribution and rejection rate). Trajectory-deviation MSE/RMSE denotes physical mission deviation and is distinct from IDS anomaly score or autoencoder reconstruction error.

7.1 Generalization: Missions and Environment

We first evaluate whether SLEIGHT's stealth and attack effectiveness generalize beyond a single mission profile, progressing from individual flight plans to dynamic maneuver complexity, to realistic environmental perturbations, and finally to coordinated swarm operations.

Performance Across Flight Paths and Durations: To evaluate duration sensitivity, we created four flight plans with increasing numbers of waypoints (WPs): *Base* (9 WPs), *Zig Zag* (15 WPs), *Fig8* (20 WPs), and *Spiral Ascent* (25 WPs). In each mission, a single-sensor attack (SSA) targeted GPS coordinates. Our results in Fig. 5 and Table 2 indicate consistently high evasion across all flight durations, with rates above 96%. Attack success remains consistent, between 85% and

Table 2: Aggregated Success Metrics Across Flights

Flight	Evasion Success (%)	Attack Success (%)	Avg Modification (Sum)
Base	100	85.94	0.83
Zig Zag	97.92	86.46	0.88
Fig 8	96.91	85.19	0.93
Spiral Ascent	100	85.06	0.88

Table 3: SLEIGHT's performance with varying flight complexity.

Flight Complexity	Mean Stealth Rate	Mean MSE
No Jumps	1.000	0.041
Every 2nd WP	0.985	0.045
Every 3rd WP	0.993	0.045
Every 5th WP	0.985	0.046

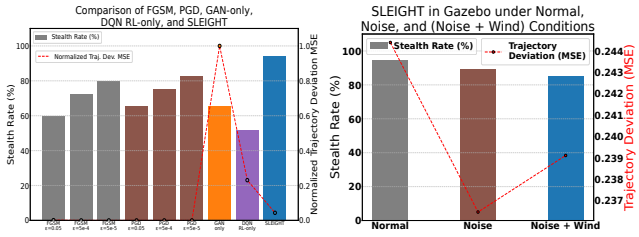


Figure 6: SLEIGHT gradient- and non-gradient attack baselines, and under varied environmental conditions (wind, noise) Gazebo.

86.5%, suggesting that small incremental GPS manipulations accumulate into measurable deviations regardless of mission length. Similarly, we observe consistent modification magnitude averaging between 0.82 and 0.93, while reconstruction error remains below the bound, demonstrating that SLEIGHT maintains subtle perturbations over extended trajectories. For cross-method comparability, we use a single phase-specific reference bound, while runtime enforcement continues to rely on the phase-effective thresholds $\tau_{\text{eff}}(\phi)$.

Effects of Flight Complexity on Stealth and Attack Success: We increased flight complexity by introducing sudden altitude jumps (10–35 m shifts), applied under four configurations: (1) *no jumps*, (2) *jumps every 5th WP*, (3) *jumps every 3rd WP*, and (4) *every other WP* (representing the most complex case). Results in Table 3 show a gradual decrease in average stealth as altitude jumps become more frequent. However, even in the most complex scenario, SLEIGHT maintained a stealth rate above 98.5%. Reconstruction error increases slightly under frequent jumps, but remains close to the detection threshold, indicating that adversarial samples remain stable despite added trajectory variability. Our analysis demonstrates that SLEIGHT tolerates maneuver-induced instability while preserving stealth.

Impact of Environmental Conditions: We evaluate SLEIGHT in Gazebo simulator under realistic disturbances, including moderate GPS noise and wind conditions with 8 m/s primary component along the x-axis and additional cross-winds. We then performed adversarial injections online. Fig. 6(b) shows that stealth decreases from near 100% under normal conditions to approximately 80–85% when noise or combined noise and wind are present, while the MSE remains relatively stable around 0.238–0.244. This

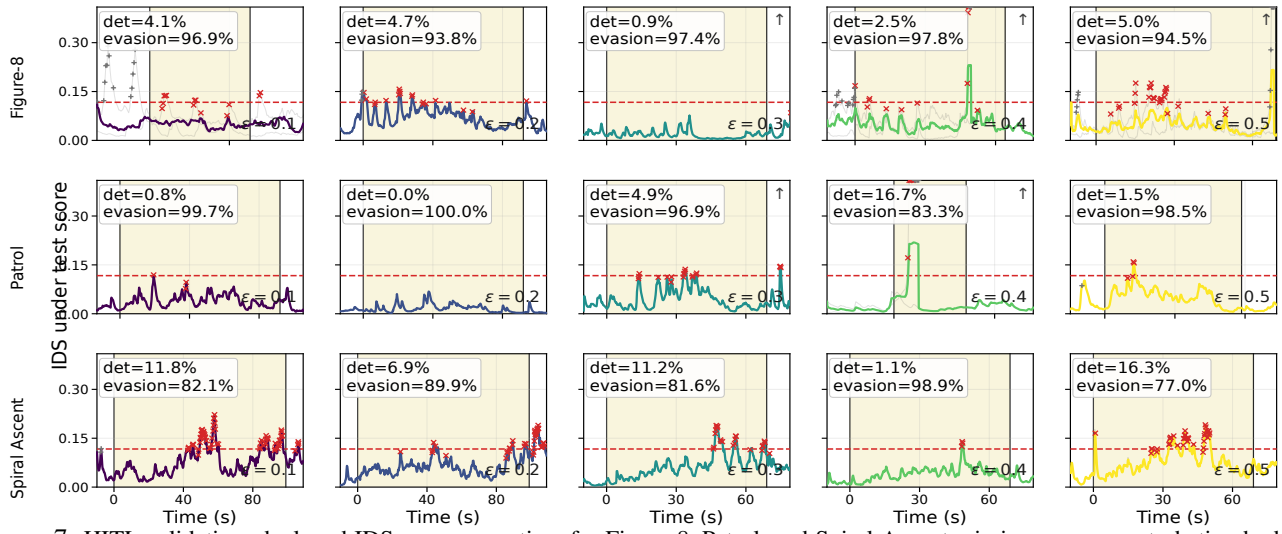


Figure 7: HITL validation: deployed IDS scores over time for Figure-8, Patrol, and Spiral Ascent missions across perturbation budgets $\varepsilon \in \{0.1, \ldots, 0.5\}$. Each panel reports detection and evasion rates. Dashed lines indicate IDS thresholds; arrows (↑) mark transient threshold exceedances during phase transitions. Detection is the fraction of timesteps (or windows) where the deployed IDS score exceeds its threshold.

modest reduction in stealth suggests that wind gusts occasionally push sensor readings closer to the IDS threshold. However, SLEIGHT's reward-based adversarial generator adapts quickly and largely maintains evasion. Hence, even under realistic environmental conditions, SLEIGHT achieves high evasion while inducing trajectory deviation, showing robustness at the single-vehicle level under environmental uncertainty. We use the two simulators for distinct roles. **JMavSim** supports the main closed-loop SITL evaluations, while **Gazebo** is used only for environmental-stress tests with wind and sensor noise. Results are not pooled across simulators: JMavSim supports nominal-flight claims, Gazebo supports environmental-robustness claims, and HITL Pixhawk results provide independent firmware-level validation.

Hardware-/Software-in-the-Loop Validation: To test transfer beyond simulation, we ran the same attack configurations on Pixhawk firmware using HITL. Fig. 7 reports deployed IDS scores for Figure-8, Patrol, and Spiral Ascent missions across $\varepsilon \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. Evasion ranges from 77.0% to 100.0%, with most configurations above 90%. Moderate budgets ($\varepsilon \in \{0.2, 0.3\}$) sustain high evasion ($\geq$93.8%) across all missions, while aggressive budgets (ε=0.5) sometimes increase detection on longer missions, e.g., 16.3% on Spiral Ascent, due to tighter firmware-level dynamics. Thus, the SITL stealth-impact trade-off persists across hardware timing, sensor noise, and estimator dynamics. Fig. 8 shows one HITL run with trajectory deviation, attacker-side plausibility score S, and deployed IDS score. Drift accumulates to about 6 m, S remains bounded, and the IDS score stays below threshold for most of the flight. Isolated in-flight and post-flight alarms occur near phase transitions, where the effective threshold tightens; runtime scaling reduces α_t^* and restores stealth within a few steps.

Takeaway: Across four flight plans, four complexity levels, two environmental conditions, and HITL validation, SLEIGHT sustains >96% nominal evasion and >80% under combined noise and wind, with 85–86.5% attack success and bounded modification. HITL confirms that SITL-derived stealth–impact trends transfer to real firmware.

7.2 Impact on Swarm Missions

We extended SLEIGHT to a UAV swarm and varied which subset of vehicles received adversarial inputs, for example, perturbing Veh-2 and Veh-3 (Fig. 9). Stealth was evaluated against the *phase-aware* threshold $\tau_{\text{eff}}(\phi)$ during transit/hover, while impact was measured at both vehicle and formation levels using per-vehicle *AE score vs.* τ_{eff}.

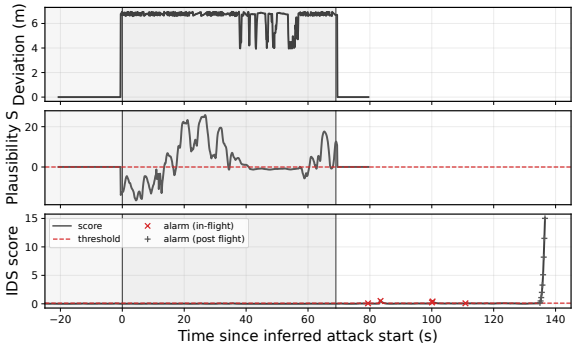


Figure 8: HITL time-series: (top) trajectory deviation in meters, (middle) attacker-side plausibility score S, (bottom) deployed IDS score vs. threshold with in-flight and post-flight alarm markers.

Table 4 (A) summarizes per-vehicle smoothed AE scores reported relative to threshold. We computed *instantaneous drift* for vehicle i as $\text{drift}_i(t) = \|p_i^{\text{EN}}(t) - p_i^{\text{EN,ref}}(t)\|_2$ using time-aligned benign references in the local East–North

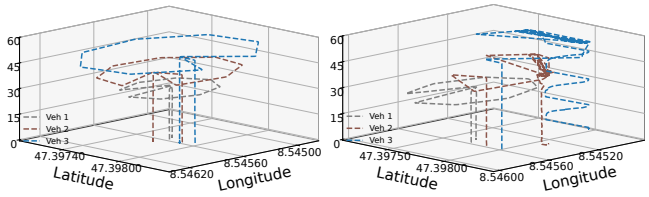


Figure 9: Swarm UAV missions (left) all benign and (right) with vehicles 2 and 3 under SLEIGHT attacks.

Table 4: SLEIGHT on Swarm (A) AE vs. effective threshold; (B) distance from target landing point (meters).

(A) IDS AE (smoothed) normalized by $\tau_{\text{eff}}(\phi)$

Vehicle	Peak (AE/τ_{eff})	Time of peak [s]	Steady (AE/τ_{eff})	τ_{eff}
veh1	0.33	8-9	0.12	1
veh2	0.27		0.07	
veh3	0.32		0.1	

(B) Distance from target landing point (m)

Case	veh1	veh2	veh3	Note
Target	0.00	0.00	0.00	reference
Benign	0.04	0.09	0.08	small bias
veh2 attacked	0.04	52.45	21.66	veh2 largest error
veh2,3 attacked	0.05	58.32	57.73	multi-veh drift
veh1-3 attacked	60.14	88.32	61.04	largest error

(EN) plane. To assess *formation metrics* we defined $p_i^{\text{EN}}(t) = [E_i(t), N_i(t)]^\top$ as the local EN position of vehicle i. The *minimum pairwise separation* is:

$$d_{\min}(t) = \min_{i<j} \left\| p_i^{\text{EN}}(t) - p_j^{\text{EN}}(t) \right\|_2$$

computed over all vehicle pairs at each t; we monitor $d_{\min}(t)$ against a safety bound *sep_min*. Formation dispersion is RMS distance to instantaneous centroid $\bar{p}(t) = \frac{1}{3}\sum_i p_i^{\text{EN}}(t)$:

$$\text{RMS}(t) = \sqrt{\frac{1}{3}\sum_{i=1}^{3} \left\| p_i^{\text{EN}}(t) - \bar{p}(t) \right\|_2^2}$$

Table 4 (B) reports per-vehicle distance from the target landing point across cases. The results show: (a) **Stealth is preserved per vehicle:** Table 4 (A) shows AE peaks and steady-state values $\ll \tau_{\text{eff}}(\phi)$, matching the single-UAV trend. (b) **Coordinated drift without instability:** In the Veh-2&3 setting, Fig. 9(b) exhibits stable lateral offsets in the attacked UAVs while the non-attacked UAV tracks nominally-selective, closed-loop impact under per-step stealth. (c) **Formation safety maintained:** The separation trace in Fig. 10 shows $d_{\min}(t) \geq \text{sep_min}$ throughout, indicating no unsafe pairwise encounters as offsets accumulate. (d) **Mission-level effect scales with the targeted subset:** Table 4 (B) shows larger landing-point displacement when multiple UAVs are perturbed; the Veh-2&3 case produces landing displacements of $\approx$58 m per attacked vehicle while remaining under $\tau_{\text{eff}}(\phi)$, and attacking all three vehicles yields displacements >60 m.

7.3 Impact of Attack Surface and Adversarial Capability

SLEIGHT's effectiveness depends on the attacker's control of sensor streams and knowledge of the operational stealth

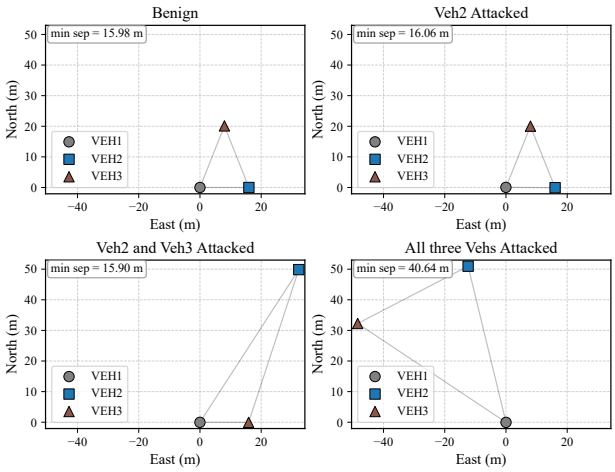


Figure 10: Swarm positions (local EN) and min-pairwise separation $d_{\min}(t)$ across pairs (1,2),(1,3),(2,3); dashed line marks sep_min = 10 m. Veh-2 and 3 are perturbed while $d_{\min}(t) \geq$ sep_min.

threshold. We move from multi-sensor manipulation to isolating individual modalities and finally to assessing how the threshold estimate τ_A governs achievable stealth and impact. SLEIGHT does not require a separately designed policy for each sensor combination. We train a single generator/policy over the telemetry feature vector and restrict the active attack surface via a sensor/feature mask M and per-feature feasibility bounds. At runtime, the generated perturbation is multiplied by M, projected into the valid domain Π_Ω, and then scaled by bisection to satisfy the attacker-side stealth constraint. Thus, GPS-only, single-sensor, and multi-sensor settings use the same attack pipeline; only the enabled feature dimensions change according to the attacker-controlled streams.

Multi-Sensor Attacks: To evaluate whether SLEIGHT can coordinate attacks across multiple sensing modalities, we extended our framework to manipulate several sensor streams concurrently, combining GPS, accelerometer (Accel), and magnetometer (Mag) readings in 4 combinations: (1) GPS + Accel + Mag, (2) GPS + Accel, (3) Accel + Mag, and (4) GPS + Mag. As shown in Table 5, all multi-sensor attacks achieve evasion rates >80%, with the GPS+Accel+Mag configuration exhibiting the highest rate. We use the same phase-conditioned policy procedure across these settings; the attack mask M selects the active channels for each sensor combination. Joint manipulation produces more noticeable changes in flight metrics, particularly in speed and heading, compared to single-sensor scenarios. We observe that while combining the three sensors (GPS+Accel+Mag) increases potential for detection (due to cross-sensor checks), SLEIGHT's reward-driven adversarial generator keeps all modifications relatively small yet consistent. Our results show that SLEIGHT scales with the attack surface and can coordinate perturbations across heterogeneous sensors.

Single-Sensor Attacks Across Modalities: We attack GPS, accelerometer, and magnetometer streams individually. Ta-

Table 5: SLEIGHT's performance when a combination of multiple sensors is attacked (GPS + accelerometer + magnetometer).

Sensor Combination	Traj. Deviation (m)	Speed Change	Heading Change	Evasion (%)
GPS+Accel+Mag	140	0.010	0.001	99
GPS+Accel	130	0.008	0.001	100
Accel+Mag	20	0.012	-0.003	98
GPS+Mag	95	0.006	0.001	98

Table 6: SLEIGHT when individual sensors are attacked.

Sensor	Traj. Deviation (m)	Speed Change	Heading Change	Evasion (%)
GPS	67	2.8	2.6	100
Accel	48	2.5	2.6	96
Mag	44	2.5	2.6	100

ble 6 reports trajectory deviation, speed/heading changes, and stealth success. GPS causes the largest offset, up to 70 m, reflecting the UAV's reliance on position; operationally, this can exceed typical low-altitude corridor or geofence half-widths (10–25 m), pushing the UAV outside its intended safety envelope while staying below the IDS threshold. Accelerometer attacks create moderate speed oscillations with high stealth because small acceleration changes resemble benign turbulence. Magnetometer attacks produce smaller immediate impact than GPS, but heading instability accumulates into gradual drift. Overall, SLEIGHT is sensor-agnostic, while impact magnitude and type depend on the targeted modality.

Effect of Attacker Threshold Estimation Capability: SLEIGHT chooses the per-step scale α_t using the attacker-side threshold τ_A, computed as a phase-conditioned reconnaissance quantile (see (6)), rather than the defender threshold τ_F. We evaluate this dependence under three cases: (i) calibrated τ_A, (ii) $\pm 20\%$ estimation error, and (iii) a time-varying effective boundary. Fig. 11 shows CUSUM stealth violations, drift rate, projection scale α versus margin $(\tau_D - \mathrm{AE})$, and attacker advantage $\Delta\tau(t) = \tau_A - \tau_D$. With calibrated estimates, violations remain near zero, α quickly reaches its feasible maximum when the margin is positive, and drift grows steadily. Under bounded uncertainty, stealth is mostly preserved, while impact decreases slightly because scaling reduces perturbation size as the margin shrinks. With a drifting boundary, outdated estimates reduce stealth: threshold tightening can make the margin negative, increase violations, and create late drift spikes when brief feasible windows appear. Thus, SLEIGHT tolerates moderate threshold error by trading impact for stealth, but drifting boundaries require periodic recalibration of τ_A or a larger safety margin κ.

7.4 Robustness Against IDS Architectures

We now evaluate SLEIGHT against a diverse set of IDS architectures to determine whether its stealth and impact generalize beyond a single detector design. Our analyses begin by positioning SLEIGHT relative to traditional gradient-based

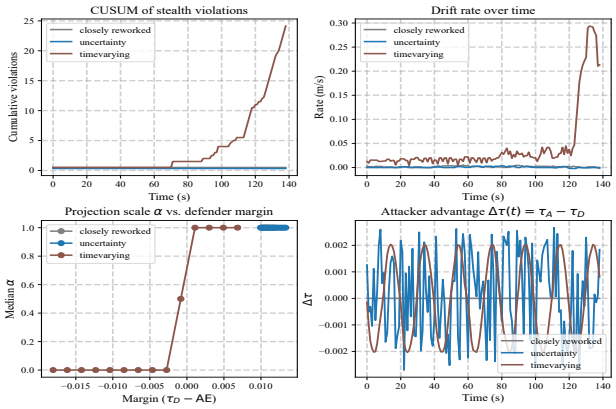


Figure 11: SLEIGHT's sensitivity to the attacker's threshold estimate: calibrated, $\pm 20\%$ error, and a time-varying threshold. Panels: CUSUM of violations, drift rate, α vs. margin, and gap $\Delta\tau(t)$.

adversarial attacks, then compare performance across standard ML-based IDS models, and finally assess evasion against representative holistic and control-aware IDS frameworks, namely ConFIDe [10] and PID-Piper [17]. We validated our ConFIDe and PID-Piper benchmark implementations (Appendix E and D) against the available public artifacts and published detector descriptions. Since the released artifacts are not directly reproducible in our PX4/MAVSDK closed-loop setting without environment, preprocessing, and log-format adaptation, we re-instantiated the detector pipelines using the same feature definitions and alarm logic. ConFIDe was validated by reproducing its 37-feature autoencoder-style reconstruction detector and confirming low alarm rates on benign UAV logs. PID-Piper was validated by reproducing its LSTM-based attitude prediction and residual/CUSUM alarm logic. We therefore use these as faithful reimplementations as detector-side benchmarks for evasion evaluation.

Comparison with Prior Attack Baselines: We compare SLEIGHT with FGSM, PGD, GAN-only, and RL-only baselines. GAN-only removes closed-loop scaling/enforcement, while RL-only uses a bounded DQN policy without the GAN realism prior/continuous feasibility projection. Fig. 6(a) reports stealth rate and normalized trajectory-deviation MSE. FGSM/PGD achieves moderate stealth at a small ε but near-zero mission impact. GAN-only increases deviation but drops to 65.3% no-alarm rate; RL-only reaches 51.6%. SLEIGHT achieves 94.3% no-alarm over active attack windows (96.4%), maintaining bounded, physically plausible deviation, showing a stronger stealth-impact tradeoff than baseline attacks.

Robustness Across ML-Based IDSs: To evaluate SLEIGHT across detector families, we trained AE, K-means, and OCSVM models on the same benign data. Fig. 12(a) shows 100% evasion against AE, indicating that SLEIGHT keeps samples near the learned reconstruction manifold. K-means and OCSVM are more resilient, with evasion rates of 90.2% and 87.3%. Fig. 12(b) shows that SLEIGHT remains below the AE threshold, with occasional spikes for the other detec-

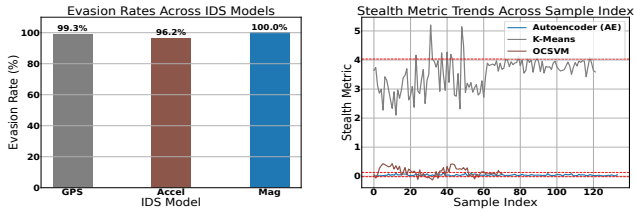


Figure 12: SLEIGHT's performance with various IDS models.

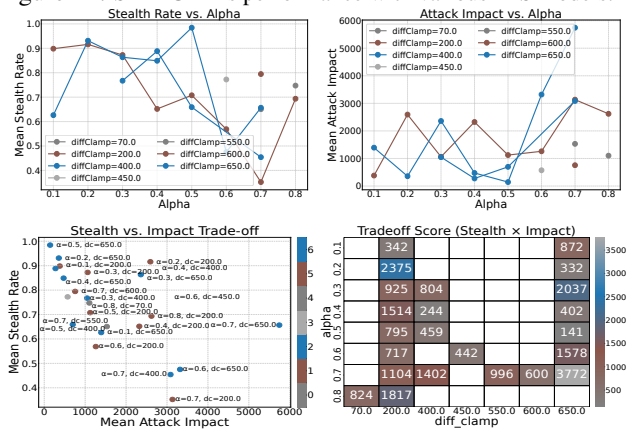


Figure 13: Evasion against ConFIDe: (a) stealth rates, (b) attack impacts, (c) stealth-impact trade-off, and (d) trade-off heatmap.

tors. Thus, reconstruction-based IDSs are most susceptible, while boundary- and clustering-based methods add partial robustness but do not prevent evasion.

Evasion of Holistic IDS (ConFIDe): We evaluated SLEIGHT against ConFIDe, a holistic autoencoder-based IDS that applies reconstruction thresholds over a 37-dimensional feature vector. We varied (1) α (blending for actuator/telemetry perturbations), (2) `diffClamp` (max scaled-space actuator change), and (3) a physical penalty discouraging implausible control (used as an ablation knob). At runtime, SLEIGHT samples a masked increment, projects to the valid domain Π_Ω, and performs a line-search/bisection on α so that the candidate satisfies the effective *phase* threshold $T \equiv \tau_{\text{phase}}(\phi(t)) - \kappa$ and the feasibility limits; only then is it injected. Fig. 13 summarizes: (i) stealth trends across α and `diffClamp`, (ii) induced impacts, (iii) the stealth–impact trade-off, and (iv) a heatmap of optimal regimes. SLEIGHT reliably evades ConFIDe, sustaining >95% stealth in moderate settings (Fig. 13(a)) while delivering meaningful actuator-level deviations (Fig. 13(b)); the heatmap (Fig. 13(d)) highlights balanced configurations where both stealth and impact are jointly high. We have elucidated the implementation details of ConFIDe in Appendix B.

Evasion of Control-Aware IDS (PID-Piper): We evaluate SLEIGHT against PID-Piper, which predicts roll, pitch, and yaw with LSTMs and applies one-sided CUSUM to cumulative residuals. SLEIGHT perturbs IMU, GPS, and EKF inputs, using Alg. 1 with $S_\theta(\cdot)$ and $\tau_A(\phi)$ to accept rewrites online; PID-Piper residuals and alarms are evaluated offline on the resulting telemetry. We vary max perturbation Δ, clamps

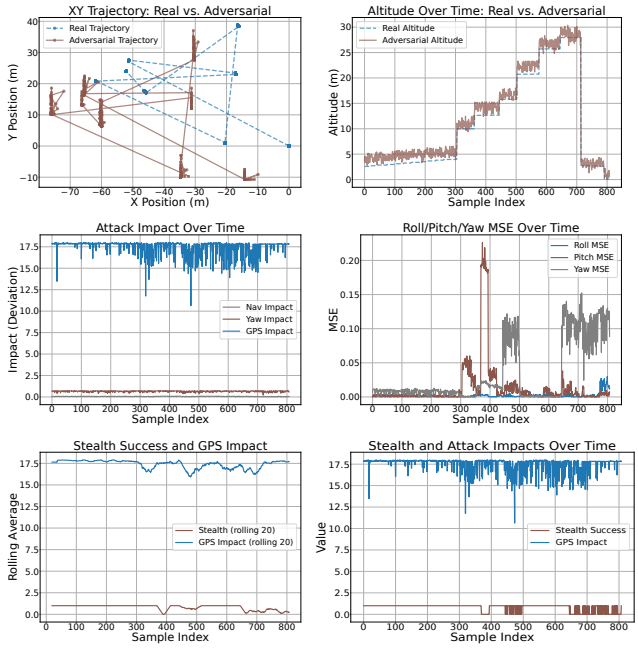


Figure 14: SLEIGHT evasion performance against PID-Piper IDS. (Top row) XY and altitude trajectories (real vs. adversarial); (Middle row) attack impact and per-axis MSE over time; (Bottom row) rolling avg. of stealth and GPS impact, and overall stealth-impact.

on `navRoll`, `navPitch`, and `yaw`, and the physical penalty. Fig. 14 shows controlled XY/altitude deviations, persistent impact, tight per-axis MSE residual control, rolling stealth above 90%, and the time-varying stealth–impact balance. Thus, SLEIGHT evades PID-Piper while producing physically plausible multi-sensor impact. Appendix C gives implementation details. We summarize cross-IDS stealth-impact using a unified frontier plot. Each point is one mission-ε configuration, with stealth as deployed no-alarm rate and impact as mean trajectory drift $\bar{d}$ over the attack window. Fig. 15 shows the Pareto frontier, where high stealth can still permit substantial mission deviation for holistic IDSs.

Takeaway: SLEIGHT evades all tested IDS families: AE (≈99%), K-means (90.2%), OCSVM (87.3%), ConFIDe (>95%), and PID-Piper (>90%) [Fig. 15]. Reconstruction-based detectors are most susceptible, while boundary- and residual-based detectors reduce but do not remove the stealth-impact tradeoff. FGSM/PGD fail to maintain closed-loop stability at budgets where SLEIGHT remains reliable.

7.5 Runtime Overhead and Practicality

We measure SLEIGHT's runtime overhead as the time from telemetry read to adversarial write per flight step. Across 50 experimental runs, the logged overhead averages 1.917×10^{-4} s, with observed values ranging between 1.2×10^{-4} s and 3.2×10^{-4} s, which is SLEIGHT's worst-case measured proxy overhead per step, and this is well within the 0.1 s period at 10 Hz. This overhead is negligible compared to the

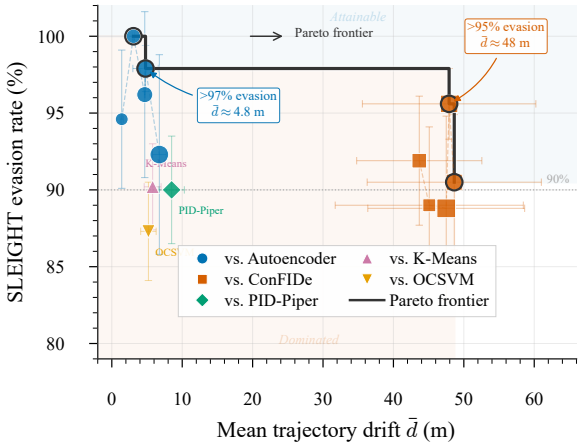


Figure 15: Stealth-impact frontier across IDSs. Stealth is the deployed per-window (or per-step) no-alarm rate under $F(\cdot)$ with thresholds $\tau_F(\phi)$. Impact is mean trajectory drift $\bar{d} = \frac{1}{|\mathcal{T}_{atk}|}\sum_{t\in\mathcal{T}_{atk}} \|p^{EN}(t) - p^{EN,ref}(t)\|_2$. Markers denote mean across runs (error bars: ± 1 std); marker size scales with ε. The Pareto frontier is computed over the plotted mean operating points.

flight controller's typical control cycle of 10–20 ms, indicating that SLEIGHT operates within real-time constraints and does not introduce measurable latency in the control loop. The per-step cost comprises policy inference, attacker-side plausibility evaluation $S_\theta(\cdot)$, and bisection with fixed budget B; all three components execute on CPU (GPU not required).

7.6 Discussion and Potential Defenses

Discussion: Our results show that offline-trained, static UAV IDSs remain vulnerable when an attacker can approximate the score distribution from reconnaissance data. Across AE, clustering, one-class, and holistic controller/actuator-aware IDSs, SLEIGHT produces small, smooth, multi-sensor perturbations that cause trajectory drift and landing offsets while staying below the phase-aware threshold τ_{eff}. This persists across flight plans, sensor choices, moderate environmental variation, HITL firmware execution, and swarm runs with only a subset of vehicles perturbed. The main driver is closed-loop feedback: perturbed telemetry changes estimator state and controller actions, amplifying small per-step changes into cumulative drift that offline/open-loop tests miss. The results also show a structural tradeoff: higher impact lies near the plausible-dynamics boundary, and residual checks such as PID-Piper reduce but do not remove fully stealthy manipulation space. HITL confirms that SITL stealth–impact trends persist under hardware timing, sensor noise, and estimator dynamics, suggesting that log-based open-loop evaluations can overestimate IDS robustness.

Countermeasures: The following approaches highlight directions for potential defenses. (1) *Adaptive or time-varying thresholds* (with mild randomization) to make boundary tracking harder. Our threshold estimation analysis demonstrates that as the defender tightens the threshold online, SLEIGHT's stealth degrades and the drift rate becomes spiky. Even basic adaptation therefore reduces impact, though it does not fully eliminate successful attacks. More sophisticated adaptive strategies and lightweight threshold randomization can be evaluated within this framework as future work. (2) *Cross-sensor and physics checks* (e.g., EKF innovation/residual consistency between GPS, IMU, magnetometer, estimator outputs, and control commands) to surface coordinated multi-channel perturbations [9, 31]. (3) *Online/periodic updates and ensembles* (poisoning-aware) to avoid a single fixed boundary. (4) *Closed-loop adversarial calibration*: incorporate SLEIGHT-like traces for calibration or adversarial training to target the specific failure modes exposed in this evaluation.

Limitations and Scope: SLEIGHT evaluates IDS robustness against an in-path telemetry-integrity adversary at the autopilot–offboard MAVLink mediation point (Section 4). This matches PX4 offboard/companion-relay deployments where limited navigation fields can be rewritten without modifying firmware, missions, or safety logic. It does not model firmware compromise, mission reprogramming, RF denial, or physical sensor reconfiguration. Our SITL/HITL experiments use the same mediation proxy and runtime enforcement loop. Different stacks, estimator tuning, or offboard controllers may shift the numerical stealth–impact boundary, but the methodology transfers by re-instantiating x_t, M, and Π_Ω for the target telemetry interface. We focus on bounded, temporally coherent, message-valid perturbations at configured update rates; stronger channel integrity reduces exposure to in-path rewriting, while redundant sensing may require coordinated multi-channel manipulation.

8 Conclusion

We proposed SLEIGHT, a closed-loop attack analytics framework for stress-testing IDS robustness that characterizes the stealth-impact frontier of UAV IDSs under physically plausible telemetry manipulation. It separates the deployed detection $F(\cdot)$ (evaluation-only) from the attacker-side plausibility $S(\cdot)$ and phase thresholds $\tau_A(\phi)$ (synthesis and runtime enforcement), enabling real-time perturbations without querying the IDS outputs. Using benign reconnaissance telemetry, SLEIGHT learns attacker-side models, generates phase-conditioned perturbation directions, and applies per-step maximal-feasible scaling under plausibility, amplitude, smoothness, and domain constraints. Multi-sensor and multi-UAV settings, and multiple IDS families (including controller- and actuator-aware designs), SLEIGHT maintains high no-alarm rates while inducing accumulating mission deviation (drift and landing displacement). Pixhawk HITL confirms the same constraint-limited behavior seen in SITL under firmware timing and estimator dynamics. These results show that offline or open-loop evaluations can overestimate IDS robustness in closed-loop operation, motivating IDS calibration and design for adaptive, mission-aware real-time adversaries.

Acknowledgements

This research was supported in part by the Department of Energy (DOE) under awards DE-CR0000024 and DE-CR0000046.

Ethical Considerations

This work evaluates UAV IDS robustness using an adversarial testing framework.
Stakeholders: Relevant stakeholders include UAV manufacturers, IDS developers, security researchers, potential attackers, and the public affected by UAV safety. ***Minimization of harm:*** All evaluations were conducted in simulation or contained environments. We did not target outside vehicles, infrastructure, operational networks, live deployments, or production devices, so the research posed no physical risk. ***Respect for privacy:*** The work involved no human subjects, recruitment, or user interaction. All data were author-generated or publicly available. We also avoid singling out specific vendors, platforms, or organizations; the evaluation is platform-agnostic and focused on methodological insights, not claims about commercial systems. ***Legal compliance and public interest:*** All experiments complied with applicable laws and institutional drone-research policies. No unauthorized access, interference, or circumvention of safeguards was performed. The goal is to strengthen defensive technologies and support safer autonomous-system design. ***Disclosure and potential misuse:*** Although adversarial evaluation methods could be misused, the primary value is defensive: identifying weaknesses before exploitation. We therefore weigh dissemination risks against the societal benefit of improved UAV security. This work supports rigorous evaluation of safety-critical IDSs and provides proactive defensive value.

Open Science

For reproducibility and to encourage further research, our work is available at [39].

References

[1] Ark cannode. https://docs.px4.io/v1.13/en/uavcan/ark_cannode. Last updated 2025-07-10.

[2] I2c bus (development overview). https://docs.px4.io/main/en/sensor_bus/i2c_development.

[3] I2c bus peripherals. https://docs.px4.io/main/en/sensor_bus/i2c_general.

[4] Mavlink message signing. https://docs.px4.io/main/en/mavlink/message_signing.

[5] Azza Allouch, Omar Cheikhrouhou, Anis Koubâa, Mohamed Khalgui, and Tarek Abbes. Mavsec: Securing the mavlink protocol for ardupilot/px4 unmanned aerial systems. In *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 621–628. IEEE, 2019.

[6] Alejandro Almeida, Muneeba Asif, Md Tauhidur Rahman, and Mohammad Ashiqur Rahman. Side-channel-driven intrusion detection system for mission critical unmanned aerial vehicles. In *2024 25th International Symposium on Quality Electronic Design (ISQED)*, pages 1–9. IEEE, 2024.

[7] Ebtihaj Alshahrani, Daniyal Alghazzawi, Reem Alotaibi, and Osama Rabie. Adversarial attacks against supervised machine learning based network intrusion detection systems. *Plos one*, 17(10):e0275971, 2022.

[8] Alvaro Alva, Luis Martinez Moreno, Muneeba Asif, Alvi Ataur Khalil, Mohammad Ashiqur Rahman, Alfredo Cuzzocrea, and Shahriar Hossain. Secured uav navigation: A novel intrusion detection system based on pwm signal analysis. In *2024 11th IEEE Swiss Conference on Data Science (SDS)*, pages 174–180. IEEE, 2024.

[9] Mohammed Y Alzahrani. Enhancing drone security through multi-sensor anomaly detection and machine learning. *SN Computer Science*, 5(5):651, 2024.

[10] Muneeba Asif, Mohammad Ashiqur Rahman, Kemal Akkaya, and Ahmad Mohammad. Confide: A pwm-driven control-fused intrusion detection system for hardware security in unmanned aerial vehicles. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, pages 886–901, 2024.

[11] Muneeba Asif, Mohammad Ashiqur Rahman, Kemal Akkaya, Hossain Shahriar, and Alfredo Cuzzocrea. Adversarial data-augmented resilient intrusion detection system for unmanned aerial vehicles. In *2023 IEEE International Conference on Big Data (BigData)*, pages 5428–5437. IEEE, 2023.

[12] Muhammad Zohaib Butt, Nazri Nasir, and Rozeha Bt A Rashid. A review of perception sensors, techniques, and hardware architectures for autonomous low-altitude uavs in non-cooperative local obstacle avoidance. *Robotics and Autonomous Systems*, 173:104629, 2024.

[13] Paolo Cerracchio, Stefano Longari, Michele Carminati, Stefano Zanero, et al. Investigating the impact of evasion attacks against automotive intrusion detection systems. In *Symposium on Vehicles Security and Privacy (VehicleSec) 2024*, pages N–A, 2024.

[14] Ozlem Ceviz, Sevil Sen, and Pinar Sadioglu. A survey of security in uavs and fanets: Issues, threats, analysis of attacks, and solutions. *IEEE Communications Surveys & Tutorials*, 2024.

[15] Aniruddh Chandratre, Tomas Hernandez Acosta, Tanmay Khandait, Giulia Pedrielli, and Georgios Fainekos. Stealthy attacks formalized as stl formulas for falsification of cps security. In *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control*, pages 1–8, 2023.

[16] Pritam Dash, Mehdi Karimibiuki, and Karthik Pattabiraman. Out of control: stealthy attacks against robotic vehicles protected by control-based techniques. In *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC '19, page 660–672, New York, NY, USA, 2019. Association for Computing Machinery.

[17] Pritam Dash, Guanpeng Li, Zitao Chen, Mehdi Karimibiuki, and Karthik Pattabiraman. Pid-piper: Recovering robotic vehicles from physical attacks. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 26–38. IEEE, 2021.

[18] Alessandro Erba, John H Castellanos, Sahil Sihag, Saman Zonouz, and Nils Ole Tippenhauer. Confusense: Sensor reconfiguration attacks for stealthy uav manipulation. In *3rd USENIX Symposium on Vehicle Security and Privacy (USENIX Vehicle Security 25)*, 2025.

[19] Jeffrey Greer IV. *MITRE Attack framework adaptation in UAV usage during surveillance and reconnaissance missions*. Mississippi State University, 2024.

[20] J Dinal Herath, Ping Yang, and Guanhua Yan. Real-time evasion attacks against deep learning-based anomaly detection from distributed system logs. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*, pages 29–40, 2021.

[21] Rune Hylsberg Jacobsen and Ali Marandi. Security threats analysis of the unmanned aerial vehicle system. In *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*, pages 316–322. IEEE, 2021.

[22] Seonghoon Jeong, Eunji Park, Kang Uk Seo, Jeong Do Yoo, and Huy Kang Kim. Muvids: False mavlink injection attack detection in communication for unmanned vehicles. In *Workshop on automotive and autonomous vehicle security (AutoSec)*, volume 2021, page 25, 2021.

[23] Shixiong Jiang, Mengyu Liu, and Fanxin Kong. Vulnerability analysis for safe reinforcement learning in cyber-physical systems. In *2024 ACM/IEEE 15th International Conference on Cyber-Physical Systems (ICCPS)*, pages 77–86, 2024.

[24] Amir Khazraei, Spencer Hallyburton, Qitong Gao, Yu Wang, and Miroslav Pajic. Learning-based vulnerability analysis of cyber-physical systems. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, pages 259–269. IEEE, 2022.

[25] Amir Khazraei, Haocheng Meng, and Miroslav Pajic. Stealthy perception-based attacks on unmanned aerial vehicles. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3346–3352. IEEE, 2023.

[26] Amir Khazraei, Haocheng Meng, and Miroslav Pajic. Black-box stealthy gps attacks on unmanned aerial vehicles. *arXiv preprint arXiv:2409.11405*, 2024.

[27] Taegyu Kim, Chung Hwan Kim, Altay Ozen, Fan Fei, Zhan Tu, Xiangyu Zhang, Xinyan Deng, Dave Jing Tian, and Dongyan Xu. From control model to program: Investigating robotic aerial vehicle accidents with {MAYDAY}. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 913–930, 2020.

[28] Taegyu Kim, Chung Hwan Kim, Junghwan Rhee, Fan Fei, Zhan Tu, Gregory Walkup, Xiangyu Zhang, Xinyan Deng, and Dongyan Xu. {RVFuzzer}: Finding input validation bugs in robotic vehicles through {Control-Guided} testing. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 425–442, 2019.

[29] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[30] Asif Ali Laghari, Awais Khan Jumani, Rashid Ali Laghari, Hang Li, Shahid Karim, and Abudllah Ayub Khan. Unmanned aerial vehicles advances in object detection and communication security review. *Cognitive Robotics*, 2024.

[31] Hunje Lee, Sujeong Han, Jeong-Il Byeon, Seoulgyu Han, Rangun Myung, Jingon Joung, and Jihoon Choi. Cnn-based uav detection and classification using sensor fusion. *IEEE Access*, 11:68791–68808, 2023.

[32] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 6235–6240. IEEE, 2015.

[33] Yilin Mo and Bruno Sinopoli. Secure control against replay attacks. In *2009 47th annual Allerton conference on communication, control, and computing (Allerton)*, pages 911–918. IEEE, 2009.

[34] Juhwan Noh, Yujin Kwon, Yunmok Son, Hocheol Shin, Dohyun Kim, Jaeyeong Choi, and Yongdae Kim. Tractor beam: Safe-hijacking of consumer drones with adaptive gps spoofing. *ACM Transactions on Privacy and Security (TOPS)*, 22(2):1–26, 2019.

[35] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[36] Raul Quinonez, Jairo Giraldo, Luis Salazar, Erick Bauman, Alvaro Cardenas, and Zhiqiang Lin. {SAVIOR}: Securing autonomous vehicles with robust physical invariants. In *29th USENIX security symposium (USENIX Security 20)*, pages 895–912, 2020.

[37] Sara Salim, Nour Moustafa, and Martin Reisslein. Cybersecurity of satellite communications systems: A comprehensive survey of the space, ground, and links segments. *IEEE Communications Surveys & Tutorials*, 2024.

[38] Burcu Sönmez Sarikaya and Şerif Bahtiyar. Generative adversarial networks for synthetic jamming attacks on uavs. In *2024 9th International Conference on Computer Science and Engineering (UBMK)*, pages 760–765. IEEE, 2024.

[39] SLEIGHT. Home. https://sites.google.com/view/sleight-/home.

[40] Caroline Strickland, Muhammad Zakar, Chandrika Saha, Sareh Soltani Nejad, Noshin Tasnim, Daniel J Lizotte, and Anwar Haque. Drl-gan: A hybrid approach for binary and multiclass network intrusion detection. *Sensors*, 24(9):2746, 2024.

[41] Fadhila Tlili, Samiha Ayed, and Lamia Chaari Fourati. Advancing uav security with artificial intelligence: A comprehensive survey of techniques and future directions. *Internet of Things*, page 101281, 2024.

[42] Mehmed Kerem Uludag, Maryna Veksler, Yasin Yilmaz, and Kemal Akkaya. Deceptive skies: Leveraging gans for drone sensor data falsification. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, pages 1606–1613, 2024.

[43] Tarun Kumar Vashishth, Vikas Sharma, Kewal Krishan Sharma, Bhupendra Kumar, Sachin Chaudhary, and Shahanawaj Ahamad. Unmanned aircraft systems (uass) technology, applications, and challenges. *Unmanned Aircraft Systems*, pages 1–63, 2024.

[44] Xiaojie Wang, Zhonghui Zhao, Ling Yi, Zhaolong Ning, Lei Guo, F Richard Yu, and Song Guo. A survey on security of uav swarm networks: Attacks and countermeasures. *ACM Computing Surveys*, 57(3):1–37, 2024.

[45] Yuan Xu, Xingshuo Han, Gelei Deng, Jiwei Li, Yang Liu, and Tianwei Zhang. Sok: Rethinking sensor spoofing attacks against robotic vehicles from a systematic view. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 1082–1100. IEEE, 2023.

[46] Nian Xue, Liang Niu, Xianbin Hong, Zhen Li, Larissa Hoffaeller, and Christina Pöpper. Deepsim: Gps spoofing detection on uavs using satellite imagery matching. In *Proceedings of the 36th Annual Computer Security Applications Conference*, pages 304–319, 2020.

Appendix

A SLEIGHT Hyperparameters Specifications

Table 7 lists the hyperparameters used in SLEIGHT's deployed runtime (`sleight_var.py`) which generated all SITL, HITL, and swarm results in this paper. Configuration is released in our repository [39]. **On λ_C and λ_F.** Both are zero because the cap and feasibility constraints are enforced as *hard* operations rather than soft penalties: $\alpha \in [0, \alpha_{\max}]$ is clamped at every step, and the runtime bisection in Alg. 1 enforces Π_Ω and the ℓ_∞ bounds before injection. This design choice means feasibility violations never reach the reward signal and soft penalties are unnecessary; we retain λ_C, λ_F in Eq. (19) for generality so practitioners can substitute soft penalties when hard projection is unavailable. **On the high**

Table 7: SLEIGHT hyperparameters.

Symbol	Value	Description
Representation & runtime		
W	50	Window length (time steps)
d	11	Feature dimensionality
f_{ctrl}	10 Hz	Control / injection rate
Stealth & feasibility enforcement		
B	8	Bisection iteration budget
$\alpha_{\max}$	0.25	Max scaling factor (per-step ℓ_∞ bound)
α_{init}	0.005	Initial scale
κ	0.01	Stealth margin ($\tau_{\text{eff}} = 0.99\,\tau_A$)
Policy training (REINFORCE)		
γ	0.99	Discount factor
λ_I	1.0	Impact reward weight
λ_R	1.0	Realism ($\log D$) shaping weight
λ_S	50.0	Stealth-violation penalty
λ_C	0	Cap penalty (clipped via $\alpha_{\max}$)
λ_F	0	Feasibility penalty (clipped via Π_Ω)

$\lambda_S = 50$. The stealth weight dominates the reward by design. Drift values r_{drift} typically lie in $[0,5]$ m per step, so $\lambda_S = 50$ ensures any stealth violation strictly outweighs the largest plausible single-step impact gain, biasing the policy toward conservative exploration of the feasible region. This is consistent with our empirical observation that SLEIGHT sustains >95% no-alarm rates across all evaluated IDSs.

B ConFIDe IDS Implementation

ConFIDe [10] is a control-fused IDS that monitors consistency across control commands, estimator outputs, and PWM actuation signals. Our implementation follows ConFIDe's autoencoder-based design: a fixed feature vector is extracted from benign flights, an autoencoder is trained to reconstruct nominal feature windows, and a threshold is set from benign reconstruction error statistics. During deployment, ConFIDe computes a reconstruction error score and raises an alarm when the score exceeds the deployed threshold.

Let $x_t \in \mathbb{R}^d$ denote ConFIDe's feature vector at time t after scaling, and let $A(\cdot)$ denote the trained autoencoder. The deployed score used for evaluation is

$$F_{\text{ConFIDe}}(x_t) = \varepsilon(x_t) = \|x_t - A(x_t)\|_2^2,$$

with a deployed threshold T_F selected from benign validation errors (e.g., a high quantile). Algorithm 2 summarizes training and inference.

Algorithm 2: ConFIDe IDS Implementation

Input: Benign flight logs, feature extractor $\mathcal{F}$, scaler $\phi(\cdot)$, quantile level q
Output: Trained autoencoder A, deployed threshold T_F
Data preparation: extract features $\mathcal{F}$ from logs, align streams, and form dataset D
Scale: $D_{\text{scaled}} \leftarrow \phi(D)$
Training: train autoencoder A by minimizing $\frac{1}{|D|}\sum_{x \in D_{\text{scaled}}} \|x - A(x)\|_2^2$
Thresholding: compute $\varepsilon(x) = \|x - A(x)\|_2^2$ on benign validation data and set $T_F \leftarrow \text{Quantile}_q(\{\varepsilon(x)\})$
Inference: for each incoming scaled x_t, compute $\varepsilon(x_t)$ and raise an alarm if $\varepsilon(x_t) > T_F$

C PID-Piper IDS Implementation

PID-Piper [17] detects actuator-level and control-integrity anomalies using LSTM predictors for benign roll, pitch, and yaw dynamics. Online prediction residuals are accumulated with a one-sided CUSUM detector, triggering recovery when the statistic exceeds a threshold. Let $x_t \in \mathbb{R}$ denote an attitude axis value (roll, pitch, or yaw) at time t, and let $\hat{x}_t$ be the one-step prediction from the corresponding LSTM. The per-step residual is $\delta_t = |x_t - \hat{x}_t|$ and the CUSUM update is

$$S(t+1) = \max\big(0,\ S(t) + \delta_t - b\big),$$

with drift parameter b and threshold τ. Algorithm 3 summarizes the pipeline.

Algorithm 3: PID-Piper IDS Implementation

Input: Benign flight logs, attitude sequences, window length w
Output: Trained LSTM models, CUSUM parameters (b, τ)
Training: for each axis (roll, pitch, yaw), train an LSTM to predict x_{t+1} from $\{x_{t-w}, \ldots, x_t\}$ by minimizing MSE
Select (b, τ) from benign residual statistics
Inference: initialize $S(0) = 0$
For each time step t: predict $\hat{x}_t$, compute $\delta_t = |x_t - \hat{x}_t|$, update $S(t)$, and trigger recovery if $S(t) > \tau$

Residual threshold-crossing evidence. To provide evidence that PID-Piper's residual signal separates benign from GPS-manipulated behavior in our setup, we compare residual norms from benign flights to residual norms from GPS-manipulated flights under the same mission. We use the strict maximum benign residual as an operational reference bound. Fig. 16 shows benign residuals remaining below this bound, while the GPS-manipulated flight crosses it repeatedly (36 crossings out of 846 samples, 4.26%), indicating repeated deviations that would contribute to CUSUM accumulation in the deployed pipeline.

D Evaluation Against ConFIDe

We evaluate SLEIGHT against ConFIDe as a deployed alarm-scoring detector, using $F_{\text{ConFIDe}}(\cdot)$ and threshold T_F only to compute no-alarm rate. Runtime injection follows Algorithm 1 with attacker-side $S_\theta(\cdot)$ and $\tau_A(\phi)$, without online access to $F_{\text{ConFIDe}}(\cdot)$ or T_F.

Metrics. We report (i) deployed stealth as the fraction of timesteps/windows with $F_{\text{ConFIDe}}(x_t) \le T_F$ and (ii) mission impact as trajectory deviation and landing displacement from closed-loop execution.

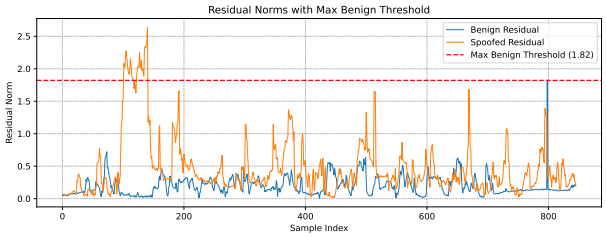


Figure 16: Residual norms over time against strict maximum benign residual bound. GPS-manipulated flight crosses bound repeatedly, while the benign flight remains below.

Algorithm 4: SLEIGHT: ConFIDe Evaluation

Input: x_t (scaled), window w_t, generator/policy g, phase estimate $\phi(t)$, plausibility S_θ, threshold $\tau_A(\cdot)$, mask M, bounds (ε, σ), projection Π_Ω, previous perturbation δ_{t-1}, max scale $\bar{\alpha}$

Output: x_t^{adv}

sample $z_t \sim \mathcal{N}(0, I)$; $\Delta x_t \leftarrow g(x_t, z_t, \phi(t))$; clamp Δx_t; $\alpha \leftarrow \bar{\alpha}$

while $\alpha > 0$ **do**

  $x^{\text{cand}} \leftarrow \Pi_\Omega\big(x_t + \alpha(M \odot \Delta x_t)\big)$; $\delta_t \leftarrow \alpha(M \odot \Delta x_t)$

  $w_t' \leftarrow \textsc{UpdateWindow}(w_t, x^{\text{cand}})$

  if $S_\theta(\text{vec}(w_t')) \le \tau_A(\phi(t))$ ***and*** $\|\delta_t\|_\infty \le \varepsilon$ ***and*** $\|\delta_t - \delta_{t-1}\|_\infty \le \sigma$ **then**

    break

  end

  $\alpha \leftarrow \alpha/2$

end

$x_t^{\text{adv}} \leftarrow (\alpha > 0)\ ?\ x^{\text{cand}} : x_t$

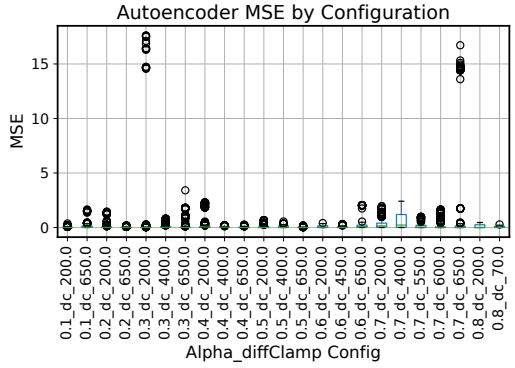


Figure 17: ConFIDe reconstruction-error distributions across $(\alpha, \texttt{diffClamp})$ configurations. Most settings maintain low error; aggressive settings produce high-error outliers.

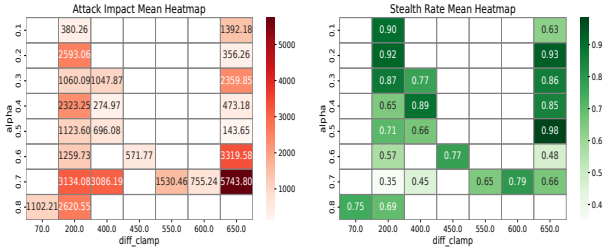


Figure 18: ConFIDe configuration sweeps: (a) mean attack impact across $(\alpha, \texttt{diffClamp})$; (b) mean stealth rate across $(\alpha, \texttt{diffClamp})$.

Configuration-wise ConFIDe reconstruction error. Fig. 17 reports ConFIDe reconstruction error distributions across $(\alpha, \texttt{diffClamp})$ configurations. Most settings remain in ConFIDe's operating region, while aggressive settings yield higher-error outliers.

Impact and stealth tradeoff across configurations. Fig. 18 summarizes mean impact and mean stealth across $(\alpha, \texttt{diffClamp})$. Conservative settings yield higher stealth

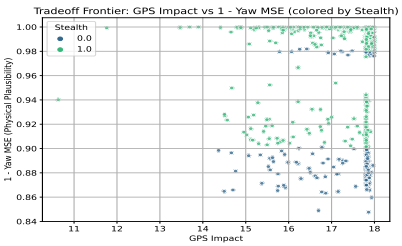


Figure 19: Tradeoff between GPS impact and a yaw-consistency proxy, colored by stealth outcome under PID-Piper.

with lower impact; more aggressive settings increase impact while lowering stealth.

E Evaluation Against PID-Piper

We evaluate SLEIGHT against PID-Piper by treating PID-Piper as a deployed detector used only for scoring alarms. PID-Piper's LSTM residual and CUSUM logic is evaluated on the resulting closed-loop telemetry to determine whether recovery would be triggered. SLEIGHT's runtime injection uses Algorithm 1 with attacker-side plausibility $S_\theta(\cdot)$ and $\tau_A(\phi)$, and does not depend on PID-Piper model parameters or thresholds online.

Algorithm 5: SLEIGHT: PID-Piper Evaluation

Input: x_t (scaled), window w_t, generator/policy g, phase estimate $\phi(t)$, plausibility S_θ, threshold $\tau_A(\cdot)$, mask M, bounds (ε, σ), projection Π_Ω, previous perturbation δ_{t-1}, max scale $\bar{\alpha}$

Output: x_t^{adv}

sample $z_t \sim \mathcal{N}(0, I)$; $\Delta x_t \leftarrow g(x_t, z_t, \phi(t))$; clamp Δx_t; $\alpha \leftarrow \bar{\alpha}$

while $\alpha > 0$ **do**

  $x^{\text{cand}} \leftarrow \Pi_\Omega\big(x_t + \alpha(M \odot \Delta x_t)\big)$; $\delta_t \leftarrow \alpha(M \odot \Delta x_t)$

  $w_t' \leftarrow \textsc{UpdateWindow}(w_t, x^{\text{cand}})$

  if $S_\theta(\text{vec}(w_t')) \le \tau_A(\phi(t))$ ***and*** $\|\delta_t\|_\infty \le \varepsilon$ ***and*** $\|\delta_t - \delta_{t-1}\|_\infty \le \sigma$ **then**

    break

  end

  $\alpha \leftarrow \alpha/2$

end

$x_t^{\text{adv}} \leftarrow (\alpha > 0)\ ?\ x^{\text{cand}} : x_t$

Metrics. We report stealth as the fraction of timesteps/windows in which PID-Piper's CUSUM statistic remains below threshold, and we report impact as trajectory deviation and landing displacement.

Tradeoff evidence. Fig. 19 visualizes a tradeoff between GPS impact and a yaw-consistency proxy, colored by stealth outcome under PID-Piper's detection logic. High-impact points cluster near the edge of plausible yaw behavior, and a subset maintain stealth under CUSUM-based detection.

ARTIFACT EVALUATED
USENIX
AVAILABLE

E-Trojans: Ransomware, Tracking, DoS, and Data Leaks on the Xiaomi E-Scooter Ecosystem

Marco Casagrande
EURECOM, Biot, France

Riccardo Cestaro
University of Padua, Padua, Italy

Mauro Conti
University of Padua, Padua, Italy

Eleonora Losiouk
University of Padua, Padua, Italy

Daniele Antonioli
EURECOM, Biot, France

Abstract

The popularity of electric scooters has grown in recent years, both for personal use and through rental apps. E-scooters are typically managed via a mobile app and powered by internal components, such as the battery, the battery management system (BMS), the driving system (DRV), and the Bluetooth Low Energy system (BTS). Despite their safety implications, there is little research on the security and privacy of these proprietary and undocumented internals and their associated attacker models. To fill this gap, we present the first security and privacy assessment of the internal components of two popular Xiaomi e-scooters. We cover the M365 (2016) and Mi3 (2023) e-scooters, and their Mi Home companion app.

Via reverse-engineering (RE), we uncover four critical design vulnerabilities on the e-scooter internals, such as arbitrary code execution on the BMS. Based on these issues, we develop E-Trojans, five novel attacks that flash a malicious BMS firmware to exploit the e-scooter's internal components. The attacks can be performed physically via the debug port of the BMS, wirelessly over Bluetooth Low Energy (BLE), and remotely from a malicious app on the victim's phone. They have a critical and real-world impact as they violate the safety, security, availability, and privacy of e-scooters and their users. For instance, they can permanently damage the e-scooter battery by undervolting it below the safety threshold or track the user by utilizing e-scooter internal values as a fingerprint.

We implement our attacks and RE findings in a modular and low-cost toolkit. Our toolkit binary patches BMS firmware by adding malicious capabilities, such as disabling battery safety thresholds, which enables our attacks and the creation of new ones. We test our attacks on real M365 and Mi3 e-scooters, empirically confirming their effectiveness and practicality. We propose four practical countermeasures that improve the security, privacy, and safety of the Xiaomi e-scooter ecosystem and its millions of users. We responsibly disclosed our findings to Xiaomi, which acknowledged and addressed them.

1 Introduction

Electric scooters are pervasive embedded systems used for smart and sustainable mobility. Their market is valued at USD 37 billion, with an estimated annual growth of 9.9% [51, 52]. They carry sensitive data, such as geolocation and telemetry, and can be wirelessly controlled by the user from a mobile companion app. A vulnerable e-scooter poses serious security, privacy, and safety risks, such as theft by defeating the motor-locking mechanism or physical injuries by tampering with the braking system.

E-scooters have complex and proprietary *internals* typically including a battery and its management system, an electric motor and its controller, and a wireless module supporting a low-power communication such as Bluetooth Low Energy (BLE). Moreover, the architecture includes internal buses, such as Universal Asynchronous Receiver-Transmitter (UART) and Inter-Integrated Circuit (I2C), allowing the modules to exchange data.

E-scooters' internals represent an attractive *attack surface*. Prior research explored internal attack surfaces in domains such as automotive, targeting electric control units (ECUs) [12] and internal communication buses [30, 72], drones, flashing malicious flight controller firmware to remotely control them [55, 56], and laptops' batteries [41]. In the context of e-scooters, researchers focused on external attack surfaces, such as the proprietary over-the-air communication protocols [11] and e-scooter rental mobile apps [69].

However, the security and privacy of *e-scooters' internals* have received no attention from the scientific community and, despite the safety risks, there is no knowledge regarding the consequences of an attacker breaking into internals, like the battery management firmware. We fill this gap by presenting a security and privacy assessment of the internals of two popular e-scooters sold by Xiaomi and its subsidiary Segway-Ninebot, which are market leaders [17]. To systematically analyze the whole attack surface, we consider three real-world attacker models: physical, proximity, and remote. We target the *M365* and *Mi3* models, covering two e-scooter generations

from 2016 to 2024. These e-scooters are used by millions of people [62] who either own an e-scooter or rent it using services like Bird [6,71]. We also cover *Mi Home*, the Xiaomi e-scooter companion app available for Android and iOS.

We reverse-engineer (RE) the e-scooters' internals using static and dynamic techniques, including firmware disassembly and decompilation and packet analysis of internal and external communications. During our RE experiments we uncover *four design vulnerabilities* (V1–V4). For instance, the presence of an unsigned BMS firmware allowing for arbitrary code execution in case of a rogue firmware update. Being design flaws, they are effective on *any* M365 and Mi3 e-scooter. Other e-scooters, such as the Pro, Pro2, 1S, and Essential, are also vulnerable to our attacks because they share the same internals as the M365 and Mi3. Through RE, we also reveal new information about the e-scooter internals, such as how to disable the hardware safety protections in the BMS, prevent the battery from charging, and alter brake settings.

We consider three realistic attacker models: *physical*, *proximity*, and *remote*. The physical attacker has one-time access to the e-scooter and the BMS debug port, the proximity attacker communicates over BLE with a nearby e-scooter, and the remote attacker leverages a malicious BLE-enabled app installed on the victim's smartphone to remotely connect to the e-scooter.

We propose **E-Trojans**, *five novel attacks on the e-scooter internals*. Our attacks flash a malicious battery management system (BMS) firmware and demonstrate what an attacker can accomplish given arbitrary code execution on the BMS. For instance, we present an *overvoltage battery destruction* attack that permanently damages the battery and an *undervoltage battery ransomware* attack that irreversibly degrades the e-scooter's battery until the victim pays a ransom, even if the e-scooter is turned off and not plugged into a charger. Both attacks can cause hazards, including fire and explosions due to physical phenomena like battery thermal runaway and polarity inversion [40]. Moreover, we show a novel way to track an e-scooter and its driver using a fingerprint computed from the e-scooter's internals (e.g., motor serial number) and advertising it over BLE. Our attacks have a critical and large-scale impact on the Xiaomi e-scooter ecosystem, breaking the safety, security, and privacy of millions of e-scooters (and their users).

We release the `E-Trojans` toolkit, implementing the E-Trojans attacks using low-cost and open-source software and hardware. Our toolkit contains a binary patcher to generate malicious BMS firmware. This patcher can inject in the BMS a set of eleven (low-level and malicious) firmware capabilities required to deploy our attacks. Such capabilities include turning off safety-critical voltage protections, disabling firmware updates, and altering the e-scooter's braking behaviour. The firmware patcher allows to design and deploy new attacks (i.e., beyond the E-Trojans attacks) by picking different combinations of capabilities, making it a generalizable tool to assess the security of e-scooters. While our toolkit targets STM8 chips, it can be extended to others, like STM32, with low engineering effort [65].

We confirm that the attacks are practical and effective by ethically testing them on 365 and Mi3 e-scooters in a controlled environment. Among others, we manage to overheat the Mi3 battery above 80°C (i.e., the thermal stability threshold [47]), degrade the M365 battery's autonomy by 50% in less than four hours via undervoltage, and track the e-scooters using BLE. To fix the presented vulnerabilities and attacks, we propose *four countermeasures* (C1–C4). These add confidentiality and integrity to the BMS (controller) firmware and protect the internal (UART) bus against spoofing and Denial-of-Service (DoS). Our fixes are *backward-compatible* and *low-cost*, as they reuse available cryptographic primitives and require minimal computational overhead. We summarize our **contributions** as follows:

- We identify the lack of security and privacy assessments on e-scooters' internals and present an analysis of two popular Xiaomi e-scooters (M365 and Mi3). Via extensive RE, we uncover four critical design vulnerabilities, including arbitrary code execution on a BMS by flashing a malicious and unsigned firmware.
- We present five novel attacks on e-scooter internals, that we call E-Trojans. They explore the consequences of running a compromised BMS on the e-scooter. For instance, they exploit battery overvoltage and undervoltage to damage the e-scooter battery, or track an e-scooter via a fingerprint computed from the e-scooter's internal components. The attacks are practical and deployable by a real-world physical, proximity-based, or remote attacker.
- We release `E-Trojans`, a toolkit implementing our attacks with open-source software and cheap hardware. The toolkit is usable to further RE the proprietary Xiaomi e-scooter ecosystem, target other non-Xiaomi e-scooters, and compose new attacks.
- We deploy our five attacks on actual M365 and Mi3 e-scooters, empirically confirming their practicality and stealthiness. We propose four effective, low-cost, and legacy-compliant countermeasures to fix the E-Trojans vulnerabilities and attacks.

Responsible Disclosure

We followed standard practices for responsible disclosure by contacting Xiaomi and the manufacturers of the BMS chips we target: STMicroelectronics (ST) and Texas Instruments (TI). We notified Xiaomi in November 2023 via their official HackerOne bug bounty program [74]. Xiaomi assigned a Medium (5.8) CVSS severity score and closed our report as informative in March 2024. In April 2024, we contacted TI and ST. TI could not comment on the Xiaomi third-party

firmware and highlighted that their STM8L151K6 chip does not claim resistance to physical attack (we note that our attacks can also be conducted remotely and in BLE proximity. ST did not respond to our report.

After a second round of disclosure in June 2025, Xiaomi acknowledged our attacks, awarding us with the highest bounty for our tier and a Medium severity CVE. They also stated that the M365 and Mi3 models have reached the end of their lifecycle, as described in [75], and that the E-Trojans vulnerabilities have been mitigated in all subsequent Xiaomi electric scooter models, which now incorporate enhanced security measures. Hence, our findings concretely contributed to a more secure Xiaomi e-scooter ecosystem.

2 Background, Motivation, and Threat Model

In this section, we provide background information, motivate our work, and present our system and attacker models.

2.1 Background

Figure 1 shows the block diagram for a Xiaomi e-scooter. An e-scooter is typically composed of a Bluetooth Low Energy system (BTS), a Driving system (DRV), and a Battery Management system (BMS). The BTS provides low-power and reliable wireless communication, the DRV controls the electric motor, and the BMS manages the battery. Each system has a dedicated System-On-Chip (SoC) running closed-source and undocumented firmware. These three systems communicate using a proprietary protocol over a UART bus that provides a serial, asynchronous, and full-duplex digital communication channel with TX/RX wires.

Mi Home App. The user interacts with a Xiaomi e-scooter via the *Mi Home* app for Android or iOS. The e-scooter and Mi Home discover each other using BLE scanning and advertising, and communicate using a proprietary protocol on top of BLE [11]. The user logs in with their Xiaomi account on Mi Home, pairs the app and the e-scooter, and then uses the app to interact with the e-scooter. Once paired, the devices automatically connect when they are in BLE range without requiring user interaction. Mi Home offers valuable features for controlling the e-scooter, such as setting a password to lock it, performing over-the-air (OTA) firmware updates, and monitoring the battery status.

E-Scooter Internals. There is limited information about Xiaomi e-scooters' internals. In [11], the authors focus on external communications with Mi Home and do not discuss internal components. Research on the Xiaomi 1S e-scooter reports that Xiaomi encrypts the BTS, DRV, and BMS firmware using a custom version of the TEA cipher featuring a leaked symmetric encryption key [7], potentially allowing the creation of custom firmware. However, the BTS and DRV have been found to be protected by ECDSA-based signature verification and include a certificate chain with a public key used to validate integrity [43].

Firmware Update. During a firmware update, Mi Home transmits firmware to the BTS, which decrypts, verifies, and distributes them to the DRV and the BMS over the UART bus. The developers of the ScooterHacking Utility [59, 61] Android app reversed an old Xiaomi firmware update protocol to enable flashing a modded BTS, DRV, or BMS firmware on e-scooters, but did not publicly disclose the protocol details. Moreover, there is no information about recent BTS, DRV, and BMS firmware and their update protocols.

Battery. The M365 and Mi3 e-scooters are equipped with a lithium-ion (Li-Ion INR) battery pack. The battery contains ten individual cells in a 5x2 configuration: two parallel groups, each made of five cells connected in series. Their nominal range goes from 42V (100% charge) to 36V (0% charge).

2.2 E-Trojans Motivation

Our research is motivated by the severe *security*, *privacy*, and *safety* risks posed by vulnerabilities and attacks on e-scooters' internals. For example, a compromised BMS can hijack the lithium-ion battery or the driving subsystem to cause physical damage to an e-scooter and its owner. Existing research in this area is limited. The author of [41] compromised a laptop's battery management system without evaluating the feasibility of battery overcharging and overheating on a real device. In [15], the authors modified office printers' firmware but did not explore attacks on the battery.

While prior work on internal attack surfaces exists for automotive [29] and drones [8, 56], the *internal attack surfaces for e-scooters* is largely unexplored, with only a few papers studying their external threats. In [11], the authors reverse-engineered and compromised proprietary communication protocols used by Xiaomi e-scooters and Mi Home over BLE. In [9], researchers performed a threat analysis on the Xiaomi M365 and reproduced known attacks on its authentication. The authors of [28] investigated physical and DoS attacks on e-scooter sharing platforms. Privacy issues have also been found in e-scooters' apps [69] that shared user identity, location, and phone data with the manufacturer and third parties.

To address this gap, the paper explores the internal attack surface of an e-scooter and its exploitability using real-world physical, proximity-based, and remote attacker models.

2.3 Threat Model

Our system model follows the block diagram in Figure 1 and includes an e-scooter, a charger, a smartphone, and a user. The e-scooter is either owned by the user or rented from a fleet and runs up-to-date firmware. The smartphone is owned by the user and runs Mi Home. The user paired their Mi Home app and account with the e-scooter. Moreover, they activate

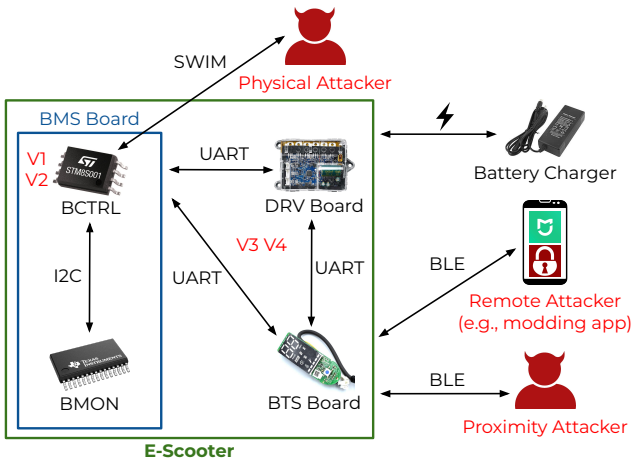


Figure 1: E-scooter block diagram and E-Trojans attacker models. The green rectangle shows e-scooter internals (BMS, DRV, and BTS boards, and UART and I2C buses). The blue rectangle shows BMS components (BCTRL and BMON connected via I2C). We consider: a physical attacker with access to the BMS debug (SWIM) port, a proximity-based attacker in the BLE range of the e-scooter and a remote attacker who installed a rogue app (e.g., an e-scooter modding app) on the victim's smartphone. We show four design vulnerabilities affecting the BCTRL (V1, V2) and the UART bus (V3, V4).

the software lock feature, allowing users to lock and unlock the e-scooter from the app.

As shown in Figure 1, we consider three attacker models: (i) a *physical attacker* with direct access to the e-scooter and the Single Wire Interface Module (SWIM) debug port of its BMS, (ii) a *proximity-based attacker* who interacts with the e-scooter wirelessly over BLE (e.g., using their device in BLE range with the victim's e-scooter); and (iii) a *remote attacker* who connects to the victim's e-scooter over the Internet through a malicious app installed on the victim's smartphone. These models systematically address the whole attack surface and map to real-world attack scenarios. For example, the proximity-based attacker can place a BLE device in a train station or a parking spot to target e-scooters in range, while the remote attacker can install a malicious BLE-enabled app on a victim's smartphone.

The attacker goal is to violate the *safety*, *security*, *privacy*, and *availability* of the e-scooter and its driver by attacking its internal components. For example, the attacker might want to attack the BMS or the DRV subsystems via a malicious firmware update over BLE. Achieving these goals leads to a large-scale and critical impact on the Xiaomi e-scooter ecosystem by affecting millions of e-scooters and their users. The attacker reuses tools, knowledge, and techniques from prior

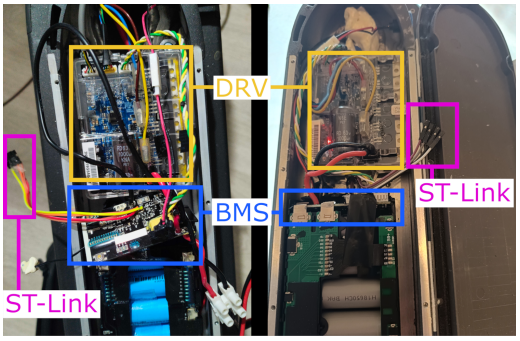


Figure 2: Disassembled M365 (left) and Mi3 (right). We color-coded the boxes to visualize the DRV (orange), the BMS (blue), and the soldered ST-Link wires (fuchsia).

work [11, 43, 59, 76]. For example, they can use the malicious BLE device and app from the `E-Spoofer` toolkit [10].

3 Xiaomi E-Scooters RE and Vulnerabilities

This section summarizes our RE findings on the M365 and Mi3 with an emphasis on their BMS. Then, it describes the four E-Trojans vulnerabilities. Refer to the Appendix for a description of our RE methodology, and more information on the internal (UART and I2C) communication protocols.

3.1 RE Findings

In 2023, we dedicated approximately nine months of RE effort on the Mi3 and M365 e-scooters. We picked these two e-scooters because the Mi3 was the latest e-scooter from Xiaomi, and the M365 was the most popular one. These models represent two different e-scooter generations. However, they share the same internal architecture, BCTRL firmware, and communication protocols over UART and I2C.

We disassembled the M365 and Mi3 e-scooters by tearing down their lower deck, gaining access to the DRV, BMS, UART bus, and battery pack, as shown in Figure 2. We extracted and inspected the BMS board of the e-scooters, finding a *battery controller (BCTRL)* and a *battery monitor (BMON)*. The BCTRL and BMON work together to ensure safe battery management. They communicate through an I2C bus, which is a synchronous two-wire serial interface (SDA and SCL) used for communication between integrated circuits.

Our security assessment focuses on three unsafe conditions that impact battery health and user safety: *overvoltage (OV)*, which can lead to overheating, fire, and thermal runaway; *undervoltage (UV)*, which can lead to polarity inversion and permanent battery damage; and voltage *imbalance* between battery cells, which contributes to both undervoltage and overvoltage, accelerating battery degradation [23, 31, 63]. We now detail new RE findings on the BCTRL and BMON of the

Mi3 and M365, as they are needed to describe the E-Trojans attacks and relative root causes.

BMON. The BMON is a TI BQ76930 chip [25]. It measures battery parameters (i.e., voltage and temperature) via dedicated sensors, reports faults to the BCTRL, and stores safety thresholds (i.e., OV and UV thresholds). These thresholds are kept in dedicated BMON registers and are configurable by the BCTRL with read and write commands sent using proprietary I2C messages. When a safety threshold is surpassed, the BMON relies on hardware protections, such as a hardware load balancer for the battery cells, or charge and discharge MOSFETs to stop charging and discharging. These protections are used to prevent or mitigate hazardous situations, like battery overvoltage or undervoltage. Being a battery monitoring integrated chip, the BMON does not run firmware or support hardware debugging [26].

BCTRL. The BCTRL is an ST STM8L151K6 chip [64]. It manages the battery's health with the BMON and communicates with the DRV and BTS. For example, it powers off the DRV and BTS when the battery undervolts or overvolts, initializes BMON registers during startup, and processes battery data received from the BMON. We identified the power supply (VDD), Single Wire Interface Module (SWIM), Ground (GND), and Reset (RST) pins used for powering, programming, and resetting the BCTRL. We soldered them to an *ST-Link v2* programmer for firmware extraction and debugging, finding *no hardware debug (ST-Link) protection*.

We RE the BCTRL firmware initialization and main loop by decompiling its stripped binary and debugging it via ST-Link. During initialization, the BCTRL loads its default settings, including serial number and firmware version. It also configures the BMON over I2C, including its overvoltage and undervoltage thresholds and the voltage balancing delta. In the main loop, the BCTRL firmware: communicates over UART with the BTS and DRV, reads the voltage of individual battery cells from BMON read-only registers, and checks these voltages against the overvoltage and undervoltage thresholds and the voltage balancing delta. If a safety check fails, the BCTRL sends an OV or UV fault message over UART, which powers off the BTS and DRV, enters a low energy consumption mode, and performs voltage load balancing on imbalanced cells.

Battery Overvoltage. A fully charged cell in a Xiaomi e-scooter battery pack has 4.2V nominal voltage. This is also the default value of the *overvoltage threshold*, stored in the BMON's `OV_TRIP` register and used to protect the battery. Overvoltage occurs when the voltage of a cell exceeds the overvoltage threshold and goes above 100% charging state. This is because 0% and 100% are relative values depending on the chemistry of the battery and not absolute ones. An overvolted cell is a serious safety hazard, capable of causing permanent battery damage, cell voltage imbalance, battery overheating, and even fire or an explosion [31]. In case of overvoltage, the BMON raises a fault (OV fault bit =1, `sys_stat` register) that can be cleared (bit set back to 0) by the BCTRL.

Battery Undervoltage. An e-scooter battery cell is considered discharged (0% charge) when its voltage is 3.6V, the nominal value defined by the manufacturer. However, cell voltage can drop as low as 2.75V, which is the default *undervoltage threshold*, below which the cell is considered undervolted. This threshold is stored in the BMON's `UV_TRIP` register and applies to all battery cells. Undervoltage reduces battery longevity and health [23,27,63], prevents safe recharging, and can result in safety hazards. For example, it can trigger polarity inversion, which can cause safety risks such as internal short circuits, voltage imbalance, swelling, fires, or an explosion. In case of undervoltage, the BMON raises a fault (UV fault bit =1, `sys_stat` register) and the fault can be cleared (bit set back to 0) by the BCTRL.

3.2 E-Trojans Vulnerabilities

Through RE, we uncovered *four design vulnerabilities* on the M365 and Mi3 internals:

V1: BCTRL firmware is weakly (or not) encrypted. An attacker can retrieve the BCTRL binary from Mi Home (at rest) or during a BLE firmware update (in transit). The M365 BCTRL firmware is not encrypted, while the Mi3 BCTRL can be decrypted through a leaked key [43].

V2: BCTRL firmware is unsigned. An attacker can modify a BCTRL firmware or craft a new one, and flash it on the BCTRL without having to authenticate it.

V3: UART bus lacks integrity protection, encryption, and authentication. An attacker on the UART bus can eavesdrop on the UART messages, replay or forge them, and impersonate or MitM the DRV, BMS, and BTS.

V4: UART bus lacks protection against DoS. An attacker accessing the UART bus can DoS the BMS, BTS, and DRV.

These issues stem from the *insecure design* of the Xiaomi e-scooter internals and firmware management, and are independent from the hardware and software implementation details of the e-scooters. V1–V4 exist in the M365, Mi3, and *all* other models that share the same BCTRL firmware and internal architecture, like the Pro, Pro2, 1S, and Essential. As a consequence, our attacks affect a wide range of e-scooters.

4 E-Trojans Attacks

We present *E-Trojans*, five novel attacks on the internals of Xiaomi e-scooters exploiting V1–V4 from Section 3.2:

1. Overvoltage Battery Destruction (OBD)
2. Undervoltage Battery Ransomware (UBR)

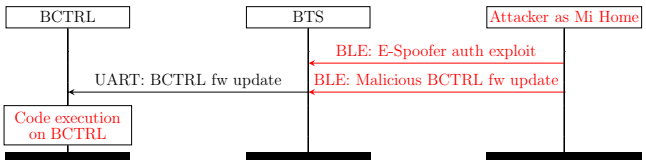


Figure 3: E-Trojans attack technique. The attacker, impersonating Mi Home, utilizes the Malicious Pairing or Session Downgrade technique from E-Spoofer [11] to authenticate to the BTS of the victim's e-scooter. Then, they perform a rogue firmware update that installs a malicious BCTRL, resulting in arbitrary code execution.

3. User Tracking via Internals (UTI)
4. Denial of E-Scooter Services (DES)
5. Password Leak and Recovery (PLR)

The attacker executes each E-Trojans attack by deploying a malicious BCTRL firmware on the victim's e-scooter. This firmware is specifically crafted to provide the attacker with arbitrary code execution and direct access to the e-scooter's internal attack surface. The delivery of the rogue BCTRL firmware can be achieved in multiple ways. For example, Figure 3 shows how to break authentication in the Xiaomi BLE proprietary protocol to flash BCTRL firmware over BLE. A physical attack with one-time access to the e-scooter can access the BCTRL debug port and flash firmware over SWIM. A software supply chain attack on the Mi Home firmware distribution server constitutes another viable remote attack vector. Table 4 in the Appendix maps the E-Trojans attacks and vulnerabilities.

Our attacks raise severe safety concerns, including the risk of fires and explosions, by overvolting and undervolting the e-scooter battery beyond the maximum overvoltage threshold or the minimum undervoltage threshold. We further discuss these safety hazards in Section 6.4. The attacks compromise the security of the Xiaomi e-scooter ecosystem by, for example, disabling firmware updates and spoofing internal components. They also violate privacy by tracking the e-scooter and its user, and break availability via DoS. Next, we describe each attack in detail.

4.1 Overvoltage Battery Destruction (OBD)

OBD exploits overvoltage to damage and destroy the e-scooter battery, resulting in a DoS attack which threatens the user's safety.

OBD employs a malicious BCTRL firmware to disable overvoltage and voltage balancing protections in the BMON, overvolting and ultimately destroying the e-scooter battery cells, as shown in Figure 4. OBD activates upon reaching the default overvoltage threshold of 4.2V, when the BMON

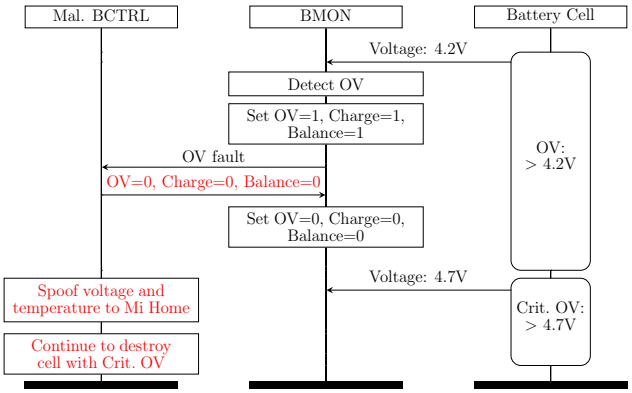


Figure 4: Overvoltage Battery Destruction (OBD). The Malicious BCTRL (the attacker) sets the OV threshold to the highest value (i.e., 4.7V). When the BMON detects overvoltage and tries to mitigate it, the attacker stops it by modifying the overvoltage fault bit, the MOSFET charge bit, and the voltage balancing bits. It also maintains stealthiness by spoofing fake sensor readings to Mi Home.

overvoltage safety protections are triggered (e.g., OV fault and stop charging). However, OBD defeats them by clearing the OV fault (OV fault bit =0, `sys_stat` register), keeping the charger connected to the battery (MOSFET charge bit =0, `sys_stat` register), and disabling load balancing (voltage balancing bits =0, `CELLBAL` registers).

This technique enables what we define as *critical overvoltage* (i.e., voltage above 4.7V). OBD remains undetected by spoofing voltage and temperature measurements to Mi Home. Since our firmware disables voltage balancing, the battery cells remain in an overvolted and unbalanced state that degrades and destroys them over time.

OBD reveals, for the first time, a *software-induced safety attack* on e-scooters, performed by a malicious BCTRL that can cause critical overvoltage without physical access or custom hardware.

4.2 Undervoltage Battery Ransomware (UBR)

UBR is an *undervoltage ransomware* that progressively and irreversibly damages the e-scooter's battery. As opposed to classical ransomware, which targets files, it asks for a ransom to stop ongoing physical damage to the e-scooter battery. It can also be deployed as a DoS attack that undervolts and destroys the battery without asking for a ransom.

UBR disables the BMON undervoltage and voltage balancing protections to undervolt the e-scooter battery cells, as shown in Figure 5. First, UBR prevents charging (MOSFET charge bit =1, `sys_stat` register), locks the e-scooter

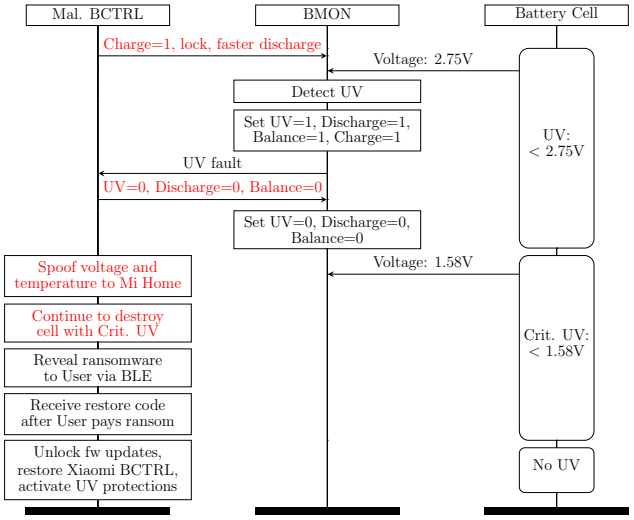


Figure 5: Undervoltage Battery Ransomware (UBR). The Malicious BCTRL (the attacker) sets the UV threshold to the lowest value (i.e., 1.58V), locks the e-scooter, and prevents it from charging (MOSFET charge bit). When the BMON detects undervoltage, the attacker breaks safety protections by altering the undervoltage fault bit, the MOSFET discharge bit, and the voltage balancing bits. UBR maintains stealthiness by spoofing sensor data, but, when triggered, will reveal its presence via BLE advertising. If the user pays the ransom, the ransom payment app submits the restore code that enables firmware updates and flashes the legitimate BCTRL firmware.

to keep it powered on, and forces discharging by ignoring sleep instructions and, optionally, turning on the headlight and taillight. When cell voltage reaches 2.75V, the BMON undervoltage safety protections activate (e.g., UV fault and voltage balancing). However, UBR turns them off by clearing the UV fault (UV fault bit =0, `sys_stat` register), restoring current flow from the battery to other internals (MOSFET discharge bit =0, `sys_stat` register), and disabling voltage load balancing.

This technique allows voltage to drop below 1.58V, putting cells into a condition that we call *critical undervoltage*. The ransomware maintains stealthiness by spoofing safe battery parameters to Mi Home until a programmable trigger activates. UBR then reveals the ransomware to the user, rebooting the e-scooter to broadcast a short link via BLE advertising (e.g., *t.ly/AaBbCc*). This link enables the victim to download an app to pay the ransom and see real-time battery degradation. To stop the damage, the victim has to transfer cryptocurrency to the attacker's crypto wallet.

Once the attacker receives the payment, they share with the user through the ransom app a restore code unique to

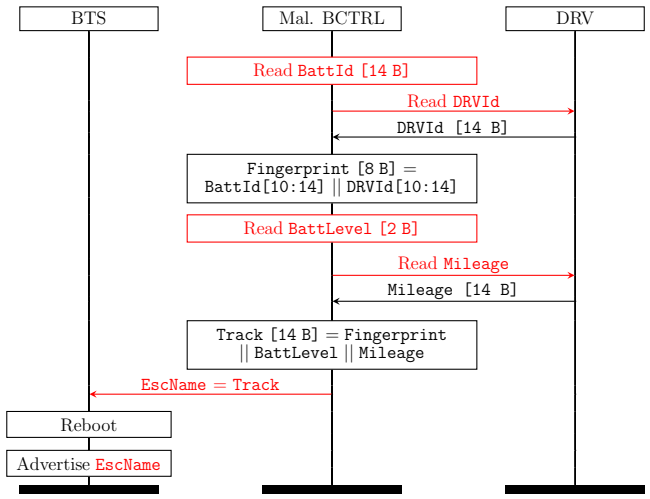


Figure 6: User Tracking via Internals (UTI). The Malicious BCTRL (the attacker) builds an 8-byte e-scooter fingerprint (`Fingerprint`) from the electric motor serial number. Then, they append other telemetry data like mileage and battery level to the fingerprint, creating a 14-byte tracking message (`Track`). Finally, they enforce a BLE name change to `Track`, allowing anyone to track the user and read their private data in real-time.

that e-scooter. The ransom app sends a special BLE packet containing that code to the e-scooter and then restores a legitimate BCTRL firmware. Undervoltage is fixed by reactivating the safety protections and charging the battery to a safe level.

UBR introduces the *first persistent, hardware-level ransomware* that monetizes physical degradation instead of data encryption. Attackers can rely on firmware vulnerabilities to cause irreversible hardware damage for profit.

4.3 User Tracking via Internals (UTI)

UTI is a privacy attack that tracks an e-scooter and its driver via a hardware-based fingerprint computed from the e-scooter's internals. Moreover, it can be used to leak the driver's data, such as travel mileage.

Figure 6 shows how UTI tracks users and leaks their e-scooter data. First, UTI reads the 14-byte battery serial number (`BattId`), and retrieves the 14-byte DRV serial number (Read `DRVId`) by impersonating the BTS and spoofing a read operation to the DRV. The attacker-controlled BCTRL builds a `Fingerprint` from the last 8 bytes of `DRVId` representing the year, month, and day of production, the revision, and the unit identifier (i.e., the nth unit produced that day).

UTI optionally retrieves via UART or I2C additional private data from the DRV, BMON, and BTS, like the real-time mileage and battery level (Read `BattLevel` and Read `Mileage`). Then, UTI appends the data to the fingerprint and

creates a 14-byte tracking message (`Track`). It sends a command to the BTS while spoofing the DRV to set the tracking message as the BLE name (`EscName = Track`). The e-scooter reboots and advertises its new name (`Track`) over BLE. The attacker can now track the user using pre-installed BLE sniffers at fixed locations, retrieving the user's position and telemetry data, which help infer travel route characteristics (e.g., if the user took a longer or shorter path).

UTI unveils a new *hardware-fingerprinting vector* based on internal serial numbers and telemetry leaked through manipulated BLE advertisements. Unlike traditional BLE-address-based tracking, this approach persists across factory resets and BLE randomization. The attack demonstrates that DRV internal identifiers can uniquely and permanently tag devices, introducing a previously unknown privacy risk.

4.4 Denial of E-Scooter Services (DES)

DES is a collection of eight DoS sub-attacks:

DES1: drops all UART and I2C packets sent to the BCTRL.

DES2: forces a supplier-only mode (SHIP mode) that disables the BMON by issuing a proprietary I2C command.

DES3: floods the UART bus with dummy packets.

DES4: periodically issues commands that reboot the e-scooter, causing a bootloop, or lock its motor, preventing the user from unlocking it.

DES5: disables battery charging unsetting the `canCharge` flag in the BCTRL flash.

DES6: exploits error handling and raises fake errors that cause the e-scooter to lock the motor or become unresponsive.

DES7: is a denial-of-sleep (DoSL) that keeps the BCTRL in a resource-intensive mode (other than low-energy mode) to discharge the battery faster.

DES8: alters braking settings and turns on/off regenerative braking (i.e., BCTRL slowing the e-scooter by converting kinetic energy into battery power, typically downhill).

DES expands the concept of DoS beyond network availability to embedded denial. By exploiting internal buses and control logic, it shows that low-cost firmware faults can disable propulsion, braking, charging, or entire subsystems. This attack highlights the need to examine *internal system communication (UART/I2C) as a DoS surface*.

4.5 Password Leak and Recovery (PLR)

PLR enables recovering the Mi Home password set by the victim user by leaking its hash from the e-scooter memory and brute forcing it offline.

On the Mi Home app, the user can set a password to prevent unauthorized access to their e-scooter via their smartphone. The password is stored on the phone, but we realized that its SHA256 hash is also stored in the DRV memory and is readable by the BTS via UART. PLR (BCTRL) spoofs the BTS to retrieve the password hash from the DRV, sets the e-scooter BLE name equal to the hash, and exfiltrates it using the e-scooter BLE name via BLE advertising.

PLR cycles through three BLE 14-byte BLE names containing portions of the hash. Each time it changes the BLE name, the e-scooter reboots. The password is six digits long (e.g., `123456`), and its search space is 10^6 [38]. Techniques such as password lists and rainbow tables can be used to brute force offline the hash to reconstruct the password. The attacker utilizes it to access the e-scooter and any other device using the same password (e.g., Android screen lock PIN).

PLR exposes a new leakage channel involving a secret stored in the DRV that can be exfiltrated over BLE advertisements. This highlights how the weak separation between the e-scooter subsystems allows recovery of authentication information, underlining the need for secure firmware boundaries and encrypted internal buses.

5 Implementation

We present `E-Trojans`, a new toolkit to conduct the E-Trojans attacks and create new ones. It includes code to send a rogue BCTRL firmware update and a binary patcher to build a firmware based on eleven malicious capabilities. While our attacks utilize fixed combinations (e.g., UTI uses C1, C2, C3, and C9), these eleven capabilities can be recombined to generate new attacks. The toolkit can be extended to work with other devices with internals and chips similar to the M365 and Mi3 e-scooters.

5.1 BCTRL Patching and Capabilities

The toolkit includes `payload-patcher.py`, a Python 3 script to binary patch a BCTRL firmware using STM8 assembly code. The script patches a vanilla BCTRL firmware, updating its CRC, and outputs a valid BCTRL firmware. Moreover, `payload-patcher.py` offers an interactive shell to craft the malicious BCTRL firmware. The user must select one of our five attacks and a supported e-scooter model (i.e., M365, Pro, Essential, Pro2, and Mi3). We support models *outside* of the M365 and Mi3 because they rely on the same BCTRL firmware and internal architecture. Thus, our malicious BCTRL is compatible with the Pro, Essential, and Pro2 models, and our techniques to flash the firmware (e.g., rogue over-the-air firmware update) remain effective.

The binary patcher includes the *eleven capabilities (C1–C11)* that compose the E-Trojans attacks, and can be combined in different ways to construct new attacks. Next, we describe each capability and how we use them.

C1 Disable Firmware Updates disables firmware updates on the BCTRL with a `canFwUpdate` flag (address 0x04CB) set to `false`. It changes the `HandleUART()` function (address 0x94C5) to accept or reject firmware update packets (starting with 0x07, 0x08, 0x09, or 0x0A), depending on `canFwUpdate`.

This flag is set to `true` when the BCTRL receives our new unlock packet (0xEE followed by the unlock code).

C2 Manage UART Bus sends arbitrary UART packets on the internal bus. It utilizes a USART1 peripheral mapped to RAM (address 0x5230). Due to V3, C2 crafts UART messages by altering fields such as the sender and receiver (e.g., 0x22 corresponds to BCTRL), packet type (e.g., read or write), command, payload, and CRC. It changes the `sendUART()` function (address 0xB06C) to send forged packets, stored from address 0xD3 onward. For example, Password Leak and Recovery uses C2 to spoof the BTS and retrieve the SHA256 hash of the e-scooter password from the DRV, but it can be used to spoof the DRV and BCTRL as well.

C3: Manage I2C Bus sends custom packets to the BMON through the I2C bus. These packets are used to read and write registers on the BMON and control it. For example, it can be used to read battery cells voltages from `VC1`, ..., `VC10` and alter the BMON undervoltage threshold in `UV_TRIP`. C3 rewrites `writeToI2C()` (address 0x90BE) to send forged packets on I2C.

C4: Modify Safety Thresholds alters the BMON undervoltage and overvoltage thresholds. It changes the BCTRL firmware boot logic, and initializes the BMON's `UV_TRIP` (address 0x923D) and `OV_TRIP` (address 0x923E) registers to 1.58V (lowest settable value) and 4.7V (highest settable value), or any other valid value. We remind that the default overvoltage and undervoltage safety thresholds are 2.75V and 4.2V. For example, C4 is used in OBD and UBR to force an overvoltage or undervoltage state that physically damages the battery, while not triggering any BMON safety protection.

C5: Disable Safety Protections disables extra safety protections in the BMON, allowing critical overvoltage (above 4.7V) and undervoltage (below 1.58V). The patch sets to 0 the BMON's OV fault, UV fault, MOSFET charge, and MOSFET discharge bits in the `sys_stat` register to prevent safety protections, such as preventing further charging, from triggering. It also rewrites the BCTRL error handler with NOPs to delete its OV and UV fault checks (address 0xB5A9) and ignore any fault message received from the BMON. For example, we use C5 in OBD to disable overvoltage protections in order to damage the battery via critical overvoltage.

C6: Disable Load Balancing speeds up cell deterioration, as load balancing is essential to compensate for voltage discrepancies across cells. It rewrites manageLoadBalancing() (address 0xA81A) to unset the three `CELLBAL` registers regulating balancing. For example, C6 is used by UBR to forcefully imbalance and kill one specific cell at a time.

C7: Disable Battery Charging disallows charging by rewriting `controlCharge()` (address 0x945C) and setting `canCharge` (address 0x42C) to `false`, regardless if the charger is connected. For instance, DES5 uses C7 to prevent the battery from charging.

C8: Fast Battery Discharge accelerates battery consumption by altering the BCTRL main loop. It sets to `false` the `sleepMode` variable (address 0x400) and forces the BCTRL to stay in a more resource-intensive mode. It can also turn on the e-scooter's headlight and taillight to further increase battery consumption. UBR uses C8 to accelerate battery depletion and critical undervoltage.

Table 1: Mapping between the E-Trojans attacks and the eleven capabilities offered by the `E-Trojans` binary patcher (`payload-patcher.py`). ✓*: OBD and UBR use either C4 or C5, depending on the attack scenario.

BCTRL Capability	OBD	UBR	UTI	DES	PLR
C1: Disable Fw Updates	✓	✓	✓	✓	✓
C2: Manage UART Bus	✓	✓	✓	✓	✓
C3: Manage I2C Bus	✓	✓	✓	✓	✗
C4: Modify Safety Thr.	✓*	✓*	✗	✗	✗
C5: Disable Safety Prot.	✓*	✓*	✗	✗	✗
C6: Disable Load Bal.	✓	✓	✗	✗	✗
C7: Disable Batt. Charge	✗	✓	✗	✓	✗
C8: Fast Batt. Discharge	✗	✓	✗	✓	✗
C9: Change BLE Adv.	✗	✓	✓	✗	✓
C10: Spoof Cell Voltage	✓	✓	✗	✗	✗
C11: Alter Brake Settings	✗	✗	✗	✓	✗

C9: Change BLE Advertising changes the e-scooter's BLE advertisements, including its BLE device name. It sends a proprietary message to the BTS containing the new BLE advertisement data. The message comprises a destination field (0x20), an opcode (0x50), and a payload containing the new advertisement. Once the message is received by the BTS, the e-scooter reboots and starts advertising over BLE with the new data. In UTI, the attacker tracks the e-scooter and its user via a custom BLE advertisement containing their fingerprint using C9.

C10: Spoof Cell Voltage fakes voltage measurements of battery cells to Mi Home to add stealthiness to an attack. It rewrites the BCTRL main loop to ignore the actual measurements from the BMON and reports arbitrary spoofed values. C10 sends these values to the BTS, which in turn sends them to Mi Home, which displays them to the user. Faking voltage allows UBR to hide the real voltage from the user while keeping undervolting the battery cells under the hood.

C11: Alter Brake Settings modifies brake-related settings managed by the BCTRL. It rewrites the default values for the brake lever pull (linear response between pull and braking force), brake phase current (regulates braking force), regenerative braking activation, and current raising coefficient (braking force in relation to speed). C11 can modify brake values in real-time depending on the context (e.g., downhill or crossing a road), maximizing the chances of injuring the user.

C1–C11 Usage. Table 1 shows how we combine the eleven capabilities to perform UBR, OBD, UTI, DES, and PLR. C1, C2, and C3 are common to all attacks, as each of them pre-

vents the user from reflashing the BCTRL, and send arbitrary packets over the UART and I2C bus (except PLR, which does not interact with I2C). C4, C5, C6, and C10 are leveraged by OBD and UBR to induce battery overvoltage and undervoltage by disabling voltage thresholds, protections, turning off load balancing, and sharing fake cell voltage. C7 and C8 are needed for UBR and DES, as they tamper with battery charge. C9 enables information disclosure from the e-scooter in UBR, UTI, and PLR, while C11 is required by DES8 to change downhill acceleration and mess with brake responsiveness.

5.2 UBR Ransom Android App and Backend

Our toolkit contains an Android app and a Django/MongoDB backend to test the UBR ransomware in a controlled, but realistic environment. These components generate unique ransom unlock codes, manage cryptocurrency payments, and handle BCTRL recovery.

ScooterToolkit. We developed *ScooterToolkit*, an Android app given to the victim so that they can pay the ransom in cryptocurrency and stop the undervoltage attack on their battery. The app is written in Kotlin and requires Bluetooth, Location, and Internet permissions. It initiates the BCTRL recovery when the back-end returns the unlock code that enables firmware updates (i.e., the user paid the ransom).

Django and MongoDB Backend. Using Django, we developed a RESTful web service that exposes two APIs to the ransom app through the APIView class. The `simulatePayment()` API simulates a user payment, setting the payment status to `Paid`. The `unlockFirmware(serial)` API requires the e-scooter's motor serial as its input, and, if the ransom has been paid, the unlock code is returned. We also created a model serializer that converts e-scooter objects into JSON to interface with our MongoDB database. This database stores the e-scooter's motor serial and model, 128-bit ransomware unlock codes, and payment status for each compromised device.

6 Evaluation

We evaluate the E-Trojans attacks (presented in Section 4) by using the `E-Trojans` toolkit (described in Section 5) on M365 and Mi3 e-scooters with up-to-date BTS, DRV, and BCTRL firmware. At the time of our experiments (2023), the M365 was the most popular model, while the Mi3 was Xiaomi's latest e-scooter. We also provide a video demonstration of UTI and DES4 at [19]. We describe our setup and results.

6.1 Evaluation Setup

We use the `payload-patcher.py` script in our toolkit to generate our malicious BCTRL firmware. The setup then varies depending on the attacker model of choice (physical, proximity-based, and remote). We test the physical attacker by connecting (only one time) an ST-Link debugger to the BCTRL SoC, as shown in Figure 2), and by directly flashing the malicious BCTRL firmware over the SWIM debug port.

For the proximity-based attacker, we run a Python script to connect over BLE from our machine (Dell Inspiron 15 3000 running a Linux-based OS) to a nearby e-scooter. We gain authorized access (e.g., via malicious pairing or session downgrade) and perform a rogue firmware update to flash the malicious BCTRL firmware.

As the remote attacker, we install the malicious *ScooterToolkit* Android app, found in our toolkit, on the victim's smartphone. We wait for the e-scooter to enter BLE range of the malicious app and then use the same technique as the proximity-based attacker to gain authorized access and perform a firmware update. We test the app on two different smartphones: a Realme GT (Android 13) and a OnePlus 3 (Android 9). We then set up each attack differently.

6.2 E-Trojans Attacks Setup

OBD Setup. To induce overvoltage, we connect two 30V power supplies in series and set them up to output 50V (i.e., above the 4.7V maximum overvoltage threshold). We monitor the voltage, temperature, and other battery parameters directly from the BMON by using our dynamic debugging setup. We plan to stop the experiment when a cell reaches either 50V or 80°C, to guarantee our safety (see Section 6.4).

UBR Setup. We configure UBR to activate when the e-scooter battery is charging at 5% or lower. We discharge the battery to 5% and connect the e-scooter to a charger, simulating a user who recharges their device overnight. Using our dynamic debugging setup, we observe and time the battery dropping from 5% charge to 0% (3.6V), and going beyond the minimum undervoltage threshold (below 1.58V). To assess how much permanent damage could be dealt, we maintain the battery in an undervolted state while monitoring parameters such as voltage, current, and temperature.

UTI Setup. We deploy BLE sniffers in four separate locations inside a private parking lot owned by our institution. These e-scooters travel on a predetermined route along the four locations, while the sniffers log their passing and the live data leaked by their BLE advertisements. The logs are then fed to a tracking system, updating the users' movements in real-time, and processed by scripts that attempt to de-anonymize them, based on where and for how long the users stopped during the current trip.

DES Setup. For each DES attack, we measure the time-to-failure and effect on user safety by observing their behaviour after the attack was triggered. We test DES8 on a mild downward slope while wearing protective equipment to mitigate fall injuries.

PLR Setup. We test the feasibility of retrieving the six-digit e-scooter password when deploying PLR. We consider the seven most common six-digit patterns [70] and randomly

Table 2: Evaluation results. All five attacks are effective on the M365 and Mi3 e-scooters. ✓[1]: OBD with dangerous overvoltage (4.2V). ✓[2]: UBR with dangerous undervoltage (1.58V).

Attack	M365	Mi3	Details
OBD	✓	✓[1]	Mi3 battery overheating in 5min
UBR	✓	✓[2]	M365 battery autonomy -50% in 3.5h
UTI	✓	✓	Tracks and de-anonymizes the user
DES	✓	✓	Sudden downhill deceleration
PLR	✓	✓	Leaks user password

generated seventy Xiaomi-compliant passwords (ten for each pattern). Then, we implement a rainbow table [35] and populate it with 3 million randomly generated Xiaomi-compliant passwords, simulating a basic attacker setup.

6.3 E-Trojans Attacks Results

Table 2 shows our evaluation results. The M365 and Mi3 e-scooters are vulnerable to the five attacks. Our experimental results demonstrate that overvoltage and undervoltage are relevant and practical threats, as the attacker can turn off safety protections on the BMS by programming the BMON from the rogue BCTRL firmware. However, they are less effective on the newer Mi3 e-scooter, whose DRV performs additional voltage checks and turns their power off if critical overvoltage or undervoltage is detected. We prove that the M365 and Mi3 e-scooters can be tracked over BLE, their password hashes can be leaked over BLE and brute forced offline, and their internals can be DoSed. We note that our empirical results represent a *lower bound* as other e-scooters might be vulnerable to our attacks For example, the Xiaomi Pro e-scooter runs BCTRL v1.2.6 (like the M365), and the 1S, Essential, and Pro2 run BCTRL v1.4.1 (like the Mi3). Next, we provide attack-specific insights.

OBD Eval. OBD overvolts and overheats the M365 battery within five minutes. We surpass the maximum overvoltage threshold and monitor the rising internal temperature of the battery. When it reaches the point of venting and electrolyte decomposition (i.e., 80°C), we halt the experiment. On the Mi3, we could only reach a dangerous overvoltage condition, as the DRV would power off the e-scooter when detecting critical overvoltage, preventing significant overheating.

UBR Eval. UBR irreversibly reduces the battery autonomy of the M365 by ∼50% in three hours and thirty minutes. We force a critical undervoltage state and stop the experiment after a few cells reach 0V (i.e., the most critical undervoltage condition). These cells now have limited charging capacity (i.e., 75%) and discharge 4x faster than healthy cells.

Similarly, we reduce the battery autonomy of the Mi3 by ∼10% in ten hours, spending six of them depleting the battery to 0% (three hours for the M365). We could not drop below the minimum undervoltage threshold due to the DRV independently measuring an abnormal voltage in the motor (RAM register with offset 0x47) and raising *Error 24 - Supply Voltage out of range*. As such, the Mi3 is better protected than the M365 against UBR, even though it still sustained significant damage.

UTI Eval. UTI successfully tracks users, exfiltrates their private data over BLE, and de-anonymizes them. We identify the two e-scooters from their fingerprint (e.g., motor serial number), allowing us to track users across the four areas where we deployed the BLE sniffers. Our tracking system, for example, reports up-to-date mileage (e.g., 0x01B1 represents 433km) and battery charge (e.g., 0x2D represents 45%), while identifying the locations where the user stopped along a predefined route.

We confirm that, by leaking the battery charge, total mileage, last travel time and mileage, and average speed, we can infer where they stopped, for how long, and whether they recharged the e-scooter (e.g., at home). We also verify that the user is still trackable after an e-scooter factory reset.

DES Eval. All DES attacks render the e-scooter unresponsive and not drivable within milliseconds. DES1 and DES3 cause the DRV to raise *Error 21 - No Communication with BMS*, halting the motor and powering off the e-scooter after ten seconds. DES2 cuts power from all internals, permanently powering off the e-scooter. DES4 prevents battery charging regardless of the charger hardware or e-scooter status (e.g., powered on/off). DES5 induces an endless lock state or reboot cycle, rendering the e-scooter unable to move or power off.

DES6 raises *Error 23 - Internal BMS not activated*, which induces a constant beeping noise, and *Error 24 - Supply Voltage out of range*, which powers off the e-scooter. DES7 ignores all sleep cycles even when the e-scooter is off, thus accelerating battery consumption. DES8 arbitrarily disables and re-enables regenerative braking, causing sudden and unpredictable deceleration and acceleration, putting a downhill driver at risk of falling off and injuring themselves. It also makes it harder to initiate braking by altering the brake lever responsiveness and modifies braking force by increasing or decreasing the brake current. By setting these parameters incorrectly on purpose, the attacker can generate too high a current and destroy the BMS board.

The DES attacks can target a variety of internal components and deny them. To restore the e-scooter to its original state, the user needs to flash the original BMON firmware via debug ports without the option of an over-the-air firmware update. This operation is complex, as it requires disassembling the entire battery compartment.

PLR Eval. PLR leaks the e-scooter password hash and enables offline brute forcing. We sniff three rotating BLE advertisements broadcasted by a compromised e-scooter, each containing part of the hash. Since the e-scooter resets after changing its advertisement, we rotate it in rapid succession, tricking the user into thinking a single reboot happened instead of three. We also confirm that retrieving the e-scooter

password from the hash is feasible. Our non-optimized rainbow table cracks the seventy passwords, following the most common six-digit patterns, in less than one second on average.

6.4 Experimental Safety Concerns

According to our experiments, OBD and UBR can be used to set a battery on fire and cause it to explode due to *induced hazards of overvoltage and undervoltage, and stealthiness*. We emphasize that the attacks are feasible from any distance, as shown in our evaluation of physical, proximity-based, and remote attackers, and that OBD and UBR do not require a charger. These aspects further increase the safety implications of the attacks. Next, we summarize the experimental evidence we collected to prove our claims.

During our UBR tests, we undervolt the battery *below the lowest BMS undervoltage threshold* (i.e., below 1.58V down to 0V). This dangerous condition is complex for users to detect, as the BCTRL spoofs valid battery levels to Mi Home and the user. During the OBD tests, we overvolt the battery *above the highest BMS overvoltage threshold* (i.e., above 4.7V up to 4.9V). The battery reached 80°C, whereas the recommended temperature range is between 0 °C and 45°C.

7 Countermeasures

We discuss how to address the E-Trojans attacks by design with *four fixes (F1–F4)*. Each countermeasure addresses one of the four vulnerabilities (i.e., F1 fixes V1 and so on) presented in Section 3.2.

F1: *Encrypt the BCTRL firmware with TEA*. The BCTRL firmware should be encrypted at rest (e.g., when stored by Mi Home) and in transit (e.g., during a BLE firmware update). We recommend reusing the custom Xiaomi implementation of the TEA block cipher already used to encrypt the DRV firmware (since v0.1.7).

F2: *Sign and verify the BCTRL firmware with ECDSA*. The BCTRL firmware should be digitally signed by Xiaomi and verified by the BCTRL to prevent rogue firmware updates. We recommend reusing ECDSA, which is FIPS-compliant [46] and already employed to sign/verify the firmware of the DRV (since DRV v0.1.7) and BTS (since BTS v1.5.2).

F3: *Protect the UART bus with SCP03*. The UART communication should be encrypted, authenticated, and integrity-protected. We suggest using standard embedded protocols such as *Secure Channel Protocol 03 (SCP03)* [49]. SCP03 provides confidentiality, integrity, authenticity, and replay protection using a lightweight scheme based on pre-shared symmetric keys, authenticated encryption, and AES. Since SCP03 is transport-agnostic, it can be adapted to UART with low engineering effort by adding message framing and flow control.

F4: *Protect the UART bus with rate limiting*. The UART bus should also be protected against DoS attacks. We recommend implementing an efficient rate-limiting mechanism for UART, such as the leaky bucket algorithm [34], where excess packets are discarded if the incoming packet rate exceeds the outgoing rate. Combining F4 with F3 enhances DoS protection, as F3 discards messages failing the integrity checks.

The fixes are *practical* and *backward-compatible*. They introduce negligible overhead, as they leverage lightweight cryptography (e.g., TEA and SCP03), while providing needed security guarantees, including confidentiality, authentication, and DoS protection. For instance, encrypting the 14KB BCTRL firmware with TEA would require ~135ms on Xiaomi's 16MHz BTS (i.e., nRF51822). They are backward-compatible with the firmware and internals of the e-scooters.

8 Related Work

E-Scooters' Security and Privacy. Research on e-scooters has studied their proprietary protocols and rental ecosystems, but no work has focused on their *internals*. A recent paper analyzed the Xiaomi e-scooter ecosystem, identifying issues in the proprietary BLE protocols used by the e-scooter and Mi Home [11]. Researchers identified vulnerabilities in the Bird e-scooter sharing platform [3] and discovered MitM attacks on the rental e-scooters of the Lime company [44]. Other studies focused on the privacy of user-related data in e-scooter rental Android apps [69] and the safety risks of riding e-scooters [37]. In 2019, researchers from Zimperium revealed flaws in the M365 locking system that could halt a running e-scooter [76]. The same year, Lanrat found that a M365 e-scooter did not enforce authentication over BLE [21].

E-Scooters Modding. There is a vibrant e-scooter modding community around the ScooterHacking [60] website. The community released useful tools such as ones to interact with Xiaomi e-scooters [58, 59] and perform firmware updates [7]. Other modding tools, such as [57], focus on e-scooter performance by, for example, increasing the maximum speed value. Our toolkit focuses on the previously unexplored BCTRL firmware and can add new malicious capabilities to the e-scooter BMS.

Attacks on Battery. The likelihood of battery overheating, thermal runaway, overcharging, and mechanical damage in electric vehicles has been investigated in [16, 32, 42]. Researchers have proposed a battery drain attack on gas and electric vehicles [13], an electromagnetic interference attack against automotive batteries [67], and a physical access rogue firmware update to a Tesla BMS [29]. Theoretical vulnerabilities in MacBook batteries, capable of overheating the

laptop, have been discussed in [41]. The E-Trojans attacks are *orthogonal* as they target e-scooters' battery and BMS.

Attacks on Embedded Firmware. Several papers focused on malicious embedded firmware, but none discussed a malicious BCTRL firmware causing battery overvoltage and undervoltage, or tracking e-scooter users. Researchers attacked embedded firmware on programmable logic control (PLC) [22], object trackers [53], printers [15], mice [39], OS management systems [24], mobile bootloaders [50], network equipment [14], botnets [4] and industrial wrenches [48].

Attacks on Battery-Powered Embedded Systems. Security research on battery-powered embedded systems has identified vulnerabilities across various domains. Automotive researchers exploited the insecure CAN bus by compromising an Electronic Control Unit (ECU) [30]. The authors of [2] document several CAN bus vulnerabilities, including the lack of authentication or traffic encryption, while [1] proposes an intrusion detection system in a vehicle's CAN bus to detect attacks such as battery degradation and explosion. Similar works have exposed wireless charging stations and EV infrastructure to attacks such as MitM, DoS, and privacy leakage [5, 33, 45, 54, 73]. Studies on drones have revealed issues in firmware updates, mobile apps, and wireless traffic [8, 18, 36, 56].

9 Conclusion

E-scooters are interesting attack targets as their exploitation can result in security, privacy, and safety issues. This paper focuses on their internals, which have received limited attention so far, mainly due to their complexity and proprietary nature. We focus on two popular Xiaomi e-scooter models, the M365 and Mi3. We RE their BMS, DRV, BTS subsystems, and their UART and I2C communication channels. We uncover four critical design flaws (V1–V4) enabling arbitrary code execution on the BCTRL firmware via a rogue firmware update over BLE. We use this attack primitive to design five novel attacks we call E-Trojans. Our attacks are executed with an attacker-controlled device in BLE proximity to the e-scooter or via a malicious app installed on the victim's phone.

OBD puts the battery in an overvoltage state by disabling the BMS overvoltage protections and damages it until destruction without being noticed by the end user. UBR disables undervoltage protections on the BMS and forces the battery into an undervoltage state, progressively and silently damaging it. Then, it asks the user for a ransom. UTI generates a unique e-scooter fingerprint from the e-scooter internals and broadcasts it over BLE to track the e-scooter and its user. It can optionally exfiltrate telemetry data. DES is a collection of attacks capable of DoS-ing the e-scooter's internal systems and communication in several ways.

We develop `E-Trojans`, a new toolkit that can conduct the E-Trojans attacks and create new ones using a binary patcher script supporting eleven malicious capabilities (C1–C11). We carried out the attacks on actual devices, proving their real-world impact and practicality. We discuss four fixes (F1–F4) to address the attacks at the design level in an efficient and backward-compatible way. We responsibly disclosed our contributions to Xiaomi, who acknowledged them, assigned us a CVE and the highest bounty for our category, and, most importantly, fixed the issues on newer e-scooter models.

Acknowledgments

This work is funded by the European Union – NextGenerationEU and by the 2023 STARS Grants@Unipd programme (Acronym and title of the project: "PatchThemAll: A Virtualization-Based Approach for Distributing Android Security Patches on Any Custom AndroidOS"). Besides funding this work, the above funding source had no involvement in the conduct of the research and in the preparation of the article. This work was partially supported by the French National Research Agency (ANR) under the France 2030 label, with the REV (ANR-22-PECY-0009) and NF-HiSec (ANR-22-PEFT-0009) research grants. We thank Professor Aurélien Francillon for his assistance in reverse-engineering the internals of the Xiaomi e-scooter.

Availability

We provide our toolkit and attack demos at https://doi.org/10.5281/zenodo.20529875. We release it as open-source, but, for safety reasons, we redacted the code responsible for generating malicious BMS firmware, which took months of RE to develop. The toolkit implements the E-Trojans attacks and shows our full attack chain in a video demonstration. It also contains a patched BCTRL firmware that blocks BMS firmware updates to prevent our attacks and our RE Ghidra projects for the M365 and Mi3 BMS firmware. Additionally, we release the undervoltage battery ransomware proof of concept, including a modified e-scooter firmware, an Android ransom payment app, and a Django/MongoDB backend managing the payment.

References

[1] S. Sai Akhil, K. Dhananjay Rao, T. Mansa, S. Tharun, and B. Dilip. A Brief Review of Cyber Attacks on Electric Vehicle Battery Management System. In *Innovations in Energy Management and Renewable Resources*, pages 87–98, 2024.

[2] Emad Aliwa, Omer Rana, Charith Perera, and Peter Burnap. Cyberattacks and Countermeasures for In-Vehicle Networks. *ACM Computing Surveys*, 54(1), 2021.

[3] App Analyst. App Analysis: Bird. https://theappanalyst.com/bird.html/, 2019.

[4] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the Mirai Botnet. In *26th USENIX security symposium (USENIX Security 17)*, pages 1093–1110, 2017.

[5] Richard Baker and Ivan Martinovic. Losing the Car Keys: Wireless PHY-Layer Insecurity in EV Charging. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 407–424, 2019.

[6] Brian Benchoff. Security Engineering: Inside the Scooter Startups. https://hackaday.com/2019/02/12/security-engineering-inside-the-scooter-startups/, 2019.

[7] BotoX. Xiaomi M365 Firmware Patcher. https://github.com/BotoX/xiaomi-m365-firmware-patcher, 2022.

[8] Pedro Cabrera. Parrot Drones Hijacking. https://www.rsaconference.com/Library/presentation/USA/2018/parrot-drones-hijacking, 2018.

[9] Louis Cameron Booth and Matay Mayrany. IoT Penetration Testing: Hacking an Electric Scooter, 2019.

[10] Marco Casagrande. E-Spoofer (GitHub). https://github.com/Skiti/ESpoofer, 2024.

[11] Marco Casagrande, Riccardo Cestaro, Eleonora Losiouk, Mauro Conti, and Daniele Antonioli. E-Spoofer: Attacking and Defending Xiaomi Electric Scooter Ecosystem. In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2023.

[12] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *20th USENIX security symposium (USENIX Security 11)*, 2011.

[13] Kyong-Tak Cho, Yuseung Kim, and Kang G Shin. Who Killed my Parked Car? *arXiv preprint arXiv:1801.07741*, 2018.

[14] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. A Large-scale Analysis of the Security of Embedded Firmwares. In *23rd USENIX security symposium (USENIX Security 14)*, pages 95–110, 2014.

[15] Ang Cui, Michael Costello, and Salvatore Stolfo. When Firmware Modifications Attack: A Case Study of Embedded Exploitation. In *NDSS*, 2013.

[16] Yan Cui, Jianghong Liu, Xin Han, Shaohua Sun, and Beihua Cong. Full-scale Experimental Study on Suppressing Lithium-ion Battery Pack Fires from Electric Vehicles. *Fire Safety Journal*, page 103562, 2022.

[17] Univ Datos. Folding Electric Scooter Market: Current Analysis and Forecast (2023-2030). https://univdatos.com/report/folding-electric-scooter-market/, 2022.

[18] Eddy Deligne. ARDrone Corruption. https://link.springer.com/article/10.1007/s11416-011-0158-4, 2011.

[19] E-Trojans. User Tracking and DoS demonstration on the Xiaomi Mi3 e-scooter. https://www.youtube.com/watch?v=Y2R46yeCXOQ, 2024.

[20] Alexey Esaulenko. Ghidra STM8 Processor Module (GitHub). https://github.com/esaulenka/ghidra_STM8, 2021.

[21] Ian D. Foster. Xiaomi M365 Scooter Authentication Bypass. https://lanrat.com/xiaomi-m365/, 2019.

[22] Luis Garcia, Ferdinand Brasser, Mehmet Hazar Cintuglu, Ahmad-Reza Sadeghi, Osama A Mohammed, and Saman A Zonouz. Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit. In *NDSS*, pages 1–15, 2017.

[23] Rui Guo, Minggao Ouyang, Languang Lu, and Xuning Feng. Mechanism of the Entire Overdischarge Process and Overdischarge-induced Internal Short Circuit in Lithium-ion Batteries. *Scientific Reports*, page 30248, 2016.

[24] Duha Ibdah, Nada Lachtar, Abdulrahman Abu Elkhail, Anys Bacha, and Hafiz Malik. Dark Firmware: a Systematic Approach to Exploring Application Security Risks in the Presence of Untrusted Firmware. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 413–426, 2020.

[25] Texas Instrument. 6 to 10-Series Cell Li-Ion and Li-Phosphate Battery Monitor. https://www.ti.com/product/BQ76930, 2022.

[26] Texas Instrument. BQ769x0 Datasheet. https://www.ti.com/lit/ds/symlink/bq76930.pdf, 2022.

[27] Texas Instruments. How To Protect 48-V Batteries from Overcurrent and Undervoltage. https://www.ti.com/lit/an/snoaa65/snoaa65.pdf, 2020.

[28] Gizay Kisa Isik, Theo Tryfonas, and George Oikonomou. E-scooter Sharing Platforms: Understanding Their Architecture and Cybersecurity Threats. In *2023 IEEE*

26th International Conference on Intelligent Transportation Systems (ITSC), pages 5909–5916, 2023.

[29] Patrick Kiley. Reverse Engineering the Tesla Battery Management System to Increase Power Available. https://www.youtube.com/watch?v=UV2zvgyIF0I, 2020.

[30] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental Security Analysis of a Modern Automobile. In *2010 IEEE symposium on security and privacy*, pages 447–462. IEEE, 2010.

[31] Berkeley Lab. Overcharge Protection Prevents Exploding Lithium Ion Batteries IB-3263. https://ipo.lbl.gov/lbnl3263, 2016.

[32] Binghe Liu, Yikai Jia, Chunhao Yuan, Lubing Wang, Xiang Gao, Sha Yin, and Jun Xu. Safety Issues and Mechanisms of Lithium-ion Battery Cell Upon Mechanical Abusive Loading: A Review. *Energy Storage Materials*, pages 85–112, 2020.

[33] Jianwei Liu, Xiang Zou, Leqi Zhao, Yusheng Tao, Sideng Hu, Jinsong Han, and Kui Ren. Privacy Leakage in Wireless Charging. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[34] Dimitris Logothetis and K Trivedi. The Leaky Bucket as a Policing Device: Transient Analysis and Dimensioning. In *INFOCOM*, volume 94, 1994.

[35] Charles Lu and Jialin Ding. Rainbow Table Attack on 6-Digit PINs (GitHub). https://github.com/clu8/RainbowTable, 2015.

[36] Aaron Luo. Multidimensional Attack Vectors and Countermeasures. https://infocondb.org/con/def-con/def-con-24/drones-hijacking-multi-dimensional-attack-vectors-and-countermeasures, 2016.

[37] Qingyu Ma, Hong Yang, Alan Mayhue, Yunlong Sun, Zhitong Huang, and Yifang Ma. E-Scooter safety: The Riding Risk Analysis Based on Mobile Sensing Data. *Accident Analysis and Prevention*, 2021.

[38] Philipp Markert, Daniel V Bailey, Maximilian Golla, Markus Dürmuth, and Adam J Aviv. This Pin Can Be Easily Guessed: Analyzing the Security of Smartphone Unlock Pins. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 286–303. IEEE, 2020.

[39] Jacob Maskiewicz, Benjamin Ellis, James Mouradian, and Hovav Shacham. Mouse Trap: Exploiting Firmware Updates in USB Peripherals. In *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, 2014.

[40] Carla Menale, Stefano Constà, Vincenzo Sglavo, Livia Della Seta, and Roberto Bubbico. Experimental Investigation of Overdischarge Effects on Commercial Li-Ion Cells. *Energies*, 15(22), 2022.

[41] Charlie Miller. Battery Firmware Hacking. *Black Hat USA*, pages 3–4, 2011.

[42] Shravan Murlidharan, Varsha Ravulakole, Jyothi Karnati, and Hafiz Malik. Battery Management System: Threat Modeling, Vulnerability Analysis, and Cybersecurity Strategy. *IEEE Access*, 13:37198–37220, 2025.

[43] Daljeet Nandha. Exploring Xiaomi's New Firmware Security Measures. https://robocoffee.de/?p=193, 2022.

[44] BBC News. Scooters Hacked To Play Rude Messages To Riders. https://www.bbc.com/news/technology-48065432, 2019.

[45] Tao Ni, Xiaokuan Zhang, Chaoshun Zuo, Jianfeng Li, Zhenyu Yan, Wubing Wang, Weitao Xu, Xiapu Luo, and Qingchuan Zhao. Uncovering User Interactions on Smartphones via Contactless Wireless Charging Side Channels. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 3399–3415. IEEE, 2023.

[46] NIST. FIPS 186-5 – Digital Signature Standard (DSS). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf, 2023.

[47] Gabriel Oltean, Nareerat Plylahan, Charlotte Ihrfors, Wei Wei, Chao Xu, Kristina Edström, Leif Nyholm, Patrik Johansson, and Torbjörn Gustafsson. Towards Li-Ion Batteries Operating at 80 °C: Ionic Liquid versus Conventional Liquid Electrolytes. *Batteries*, 4(1), 2018.

[48] PCmag. Network-Connected Torque Wrench Used in Factories Is Vulnerable to Ransomware. https://www.pcmag.com/news/network-connected-torque-wrench-used-in-factories-is-vulnerable-to-ransomware, 2024.

[49] Global Platform. SCP03. https://globalplatform.org/specs-library/secure-channel-protocol-03-amendment-d-v1-2, 2020.

[50] Nilo Redini, Aravind Machiry, Dipanjan Das, Yanick Fratantonio, Antonio Bianchi, Eric Gustafson, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. BootStomp: on the Security of Bootloaders in Mobile Devices. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 781–798, 2017.

[51] Grand View Research. Electric Scooters Market Size, Share & Trends Analysis Report (2024-2030). https://www.grandviewresearch.com/industry-analysis/electric-scooters-market, 2024.

[52] Grand View Research. Micro-mobility Market Size, Share & Trends Analysis Report By Vehicle Type (Electric Kick Scooters, Electric Skateboards, Electric Bicycles), By Battery, By Voltage, By Region, And Segment Forecasts, 2025-2030. https://www.grandviewresearch.com/industry-analysis/micro-mobility-market-report, 2025.

[53] Thomas Roth, Fabian Freyer, Matthias Hollick, and Jiska Classen. AirTag of the Clones: Shenanigans with Liberated Item Finders. In *2022 IEEE Security and Privacy Workshops (SPW)*, pages 301–311. IEEE, 2022.

[54] Khaled Sarieddine, Mohammad Ali Sayed, Sadegh Torabi, Ribal Attallah, Danial Jafarigiv, Chadi Assi, and Mourad Debbabi. Uncovering Covert Attacks on EV Charging Infrastructure: How OCPP Backend Vulnerabilities Could Compromise Your System. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, page 977–989, 2024.

[55] Nico Schiller, Merlin Chlosta, Moritz Schloegel, Nils Bars, Thorsten Eisenhofer, Tobias Scharnowski, Felix Domke, Lea Schönherr, and Thorsten Holz. Drone Security and the Mysterious Case of DJI's DroneID. In *NDSS*, 2023.

[56] Moritz Schloegel and Nico Schiller. Unchained Skies: A Deep Dive into Reverse Engineering and Exploitation of Drones. https://mschloegel.me/slides/slides_recon23_drone_security.pdf, 2023.

[57] ScooterHacking. ScooterHacking Firmware Toolkit. https://mi.cfw.sh/, 2022.

[58] ScooterHacking. ScooterHacking (GitHub). https://github.com/orgs/scooterhacking/repositories, 2022.

[59] ScooterHacking. ScooterHacking Utility App. https://play.google.com/store/apps/details?id=sh.cfw.utility, 2022.

[60] ScooterHacking. ScooterHacking Website. https://scooterhacking.org/, 2022.

[61] ScooterHacking. ScooterHacking Utility Homepage. https://utility.cfw.sh/, 2023.

[62] Ma Si. Segway-Ninebot Eyes Bigger Market Share in Country. https://global.chinadaily.com.cn/a/202204/28/WS6269ee14a310fd2b29e59d1f.html, 2022.

[63] Shashank Sripad, Sekar Kulandaivel, Vikram Pande, Vyas Sekar, and Venkatasubramanian Viswanathan. Vulnerabilities of Electric Vehicle Battery Packs to Cyberattacks on Auxiliary Components. *arXiv preprint arXiv:1711.04822*, 2017.

[64] STMicroelectronics. Ultra-low-power 8-bit MCU with 32 Kbytes Flash, 16 MHz CPU, integrated EEPROM. https://www.st.com/en/microcontrollers-microprocessors/stm8l151k6.html, 2018.

[65] STMicroelectronics. Migration of Applications from the STM8L and STM8S Series to the STM32C0 Series Microcontrollers. https://www.st.com/resource/en/application_note/an5775-migration-of-applications-from-the-stm8l-and-stm8s-series-to-the-stm32c0-series-microcontrollers-stmicroelectronics.pdf, 2022.

[66] STMicroelectronics. ST Visual Develop. https://www.st.com/en/development-tools/stvd-stm8.html, 2024.

[67] Marcell Szakály, Sebastian Köhler, Martin Strohmeier, and Ivan Martinovic. Assault and Battery: Evaluating the Security of Power Conversion Systems Against Electromagnetic Injection Attacks. *arXiv preprint arXiv:2305.06901*, 2023.

[68] ST-Link V2. ST-Link In-Circuit Debugger and Programmer for STM8 and STM32. https://www.st.com/en/development-tools/st-link-v2.html, 2024.

[69] Nisha Vinayaga-Sureshkanth, Raveen Wijewickrama, Anindya Maiti, and Murtuza Jadliwala. An Investigative Study On The Privacy Implications Of Mobile E-Scooter Rental Apps. In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, page 125–139, 2022.

[70] Ding Wang, Qianchen Gu, Xinyi Huang, and Ping Wang. Understanding Human-Chosen PINs: Characteristics, Distribution and Security. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS'17)*, 2017.

[71] Tech We Want. Who Makes Bird and Lime Scooters? https://techwewant.com/this-is-who-makes-bird-lime-and-jump-scooters-review-b3f6be32221e, 2018.

[72] Marko Wolf, André Weimerskirch, and Christof Paar. Security in Automotive Bus Systems. In *Workshop on Embedded Security in Cars*, pages 1–13. Bochum, 2004.

[73] Yi Wu, Zhuohang Li, Nicholas Van Nostrand, and Jian Liu. Time to Rethink the Design of Qi Standard? Security and Privacy Vulnerability Snalysis of Qi Wireless Charging. In *Annual Computer Security Applications Conference*, pages 916–929, 2021.

[74] Xiaomi. Xiaomi Bug Bounty Program (HackerOne). https://hackerone.com/xiaomi, 2024.

[75] Xiaomi. Xiaomi – Trust Center EOL Product List. https://trust.mi.com/misrc/updates/iot, 2025.

[76] Rani Idan (Zimperium). Don't Give Me A Brake – Xiaomi Scooter Hack Enables Dangerous Accelerations And Stops For Unsuspecting Riders. https://blog.zimperium.com/dont-give-me-a-brake-xiaomi-scooter-hack-enables-dangerous-accelerations-and-stops-for-unsuspecting-riders/, 2019.

Appendix

RE Challenges and Methodology

It took us around nine months to RE the M365 and ES3 internals and the Mi Home mobile app. This process was challenging for several factors. First, the e-scooter ecosystem is sophisticated, combining black-box hardware components with mobile application logic and opaque operations performed by the Xiaomi backend. Second, we started with no prior knowledge of the BCTRL firmware.

We had to physically inspect hardware components to identify them, and consult their datasheet before we could properly RE the BCTRL source code. The firmware lacks debug symbols and documentation, requiring extensive manual analysis of the decompiled code. Third, our experimental setup is non-trivial. It includes soldering wires to connect them to debug interfaces, installing proprietary software for runtime debugging, unbricking and re-flashing the e-scooters multiple times while testing the malicious BCTRL firmware. We also had to work within strict safety guidelines since we were messing with the voltage of dangerous lithium-ion batteries.

Our RE methodology combines static and dynamic analysis techniques, applied across different parts of the Xiaomi e-scooter ecosystem. We now describe each technique and then discuss how this methodology can be generalized to other e-scooter models and manufacturers.

Firmware static analysis. We used `binwalk` to scan the firmware and see if they were encrypted (i.e., entropy analysis) or contained certificates and public keys. Appended to the encrypted firmware, we found ECDSA signatures that we verified with `openssl`. We decrypted the BTS and DRV firmware using TEA and a leaked Xiaomi symmetric key (i.e., `UPDKEY=FE801CB2D1EF41A6A41731F5A06824F0`) [7].

We decompiled the BCTRL firmware using `Ghidra`. Since the BCTRL runs on an STM8L151K6 and `Ghidra` does not natively support the STM8 instruction set architecture, we installed the STM8 processor module extension [20]. We selected STM8 16-bit Big Endian as our Language and Compiler Specification and memory mapped the firmware.

We identified the RAM (0x0000 to 0x07FE), EEPROM (0x1000 to 0x13FF), FLASH (0x8000 to 0xFFFF), USART and I2C communication interfaces by consulting the microprocessor's datasheet. Then, we statically reversed the firmware, naming functions and variables based on their behavior. We started our RE from functions referencing the USART and I2C interfaces, tracing them back to the main loop. We also searched for values used by the internal protocols, like the UART packet header (i.e., 0x5AA5).

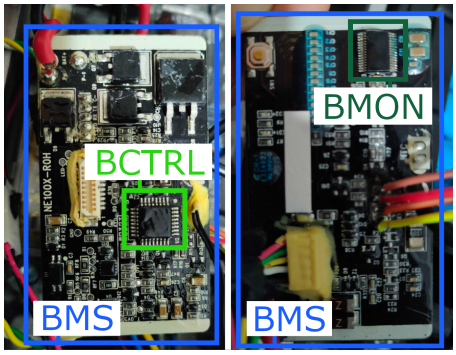


Figure 7: M365 BMS components on the M365. The BCTRL STM8 SoC (left) and the BMON BQ73930 SoC (right).

Firmware dynamic analysis. To perform the dynamic experiments, we disassembled the e-scooter chassis (as shown in Figure 2), protecting the BMS, DRV, and battery pack. Then, we extracted the BMS board and identified the BCTRL and BMON SoCs (as shown in Figure 7). The BCTRL exposed unlocked debugging ports, connectable through a *ST-Link v2* [68] debugger/programmer. After figuring out from the datasheet the pin layout, which was unknown for the ES3, we soldered four wires to the BCTRL's VDD, SWIM, GND, and RST pins. We used ST-Link onboard `gdb` server to debug a running BCTRL firmware through the *ST Visual Develop* IDE provided by ST [66].

We debugged the firmware at runtime with common practices. We set breakpoints and watchpoints to read and write to memory and inspect the CPU state. For instance, we set a breakpoint at 0xB06C to capture inbound UART packets and at 0x00D3 for outbound ones. Similarly, we set a breakpoint at 0xA24B to capture inbound I2C packets and we read the latest cell voltage measurement in the `X` register.

Local and backend traffic analysis. We captured local (BLE) traffic between Mi Home and an e-scooter by enabling *Android HCI snooplog* in the Developer Options. We opened the btsnoop port (8872) on our rooted smartphones, allowing us to monitor live BLE traffic through an interface in `Wireshark`. We retrieved the Xiaomi pairing key by performing session downgrade or malicious pairing as in [11].

Then, we triggered a BCTRL firmware update, using a script to decrypt the BLE traffic, from which we extracted the stripped BCTRL firmware. We eavesdropped on the backend (Wi-Fi or cellular) traffic during a legitimate Xiaomi firmware update using mitmproxy. To do so, we installed their CA certificate on our smartphone, set a proxy, and monitored the requests and responses from Mi Home. We also discovered

that the Xiaomi backend authenticates a firmware download request using its MD5 hash. This confirms that our BTS, DRV, and BCTRL binaries are legitimate and up-to-date.

Mi Home and ScooterHacking Android apps analysis. We studied the *closed-source* and *undocumented* Mi Home and ScooterHacking Utility Android apps to RE the Xiaomi proprietary firmware update protocol and the BLE packets that convey data from the BCTRL to the app. We performed static analysis using `JADX` and discovered that the code managing the communication with the e-scooter is obfuscated. We manually RE the decompiled code, finding the function responsible for generating BLE traffic, crypto primitives (encryption, decryption, authentication via challenge-response), and downloading BTS, DRV, and BCTRL firmware from the Xiaomi cloud. We utilized `Frida` for dynamic binary instrumentation to check functions' inputs and outputs and modify them at runtime during our RE experiments.

Methodology generalization. Our RE methodology is generalizable to other e-scooters since most use similar architectures. Our static and dynamic firmware analysis approach applies to any embedded firmware, which all feature RAM, EEPROM, FLASH memory, and internal UART and I2C bus. We target a systemic issue of battery management systems in embedded systems, showing how their components are not secure against an internal attacker. The same applies to the analysis we deploy on mobile e-scooter companion apps.

I2C and UART Binary Protocols

We RE the proprietary UART protocol used by the BCTRL, DRV, and BTS. The BCTRL uses a USART1 peripheral mapped to RAM (address 0x5230) to read and write UART. A UART packet has a constant 2 bytes header (0x5AA5), 2 bytes denoting the packet length, 2 bytes specifying the sender and receiver (e.g., 0x22 corresponds to BTS), 2 bytes identifying the packet type (e.g., read or write), 2 bytes indicating the command to execute, an N-byte payload, and a 2-byte CRC.

The BCTRL and BMON communicate using a proprietary protocol over I2C, which is neither encrypted nor authenticated. I2C is a synchronous serial digital protocol running on two wires (SDA, SCL) that supports master and slave devices, message addressing, and framing. I2C has a higher bandwidth than UART. The BCTRL can directly read and write the BMON registers via I2C. For example, we can set the overvoltage threshold by writing into `OV_TRIP` or read cell voltages via `VC1`, ..., `VC10`.

Table 3 presents a selection of UART and I2C commands we utilize in our attacks. We represent UART commands with their Xiaomi proprietary format and the I2C commands by their target register and accepted values. The *Lock*, *Reboot*, and *Shutdown* commands target the DRV and affect the e-scooter by preventing its movement, rebooting it or turning it off. We send the *Read PwdHash*, *WritePwdHash*, and *ReadDRVSerial* to the DRV to leak private or confidential data, like the hashed password or the DRV serial number.

Table 3: UART and I2C commands (column one) and content (column two). We represent UART commands as Xiaomi-compliant HEX packets, for example Change Adv includes two 1-byte fields (0x21 and 0x50) and the new advertising data. We represent I2C commands through the registers (R) they read or write. For example, the Set UVT command writes a value between 0 and 255 to R[0x9].

Src, Dst	Command	Content
UART bus		
BTS, BCTRL	Fw Update Init	0x220700‖ size
BTS, BCTRL	Fw Update Chunk	0x220800‖chunk
BTS, BCTRL	Fw Update Chksm	0x220900‖chksm
BTS, BCTRL	Fw Update End	0x220A00
DRV, BTS	Change Adv	0x2150‖advdata
BTS, DRV	Lock	0x20037001
BTS, DRV	Reboot	0x20037801
BTS, DRV	Power OFF	0x20037901
BTS, DRV	Read DRV Serial	0x20011006
I2C bus		
BCTRL, BMON	Read Voltage	0x0C-0x1F
BCTRL, BMON	Set OVT	0x09‖value
BCTRL, BMON	Set UVT	0x0A‖value
BCTRL, BMON	No Balancing	0x01‖0x00, 0x02‖0x00
BCTRL, BMON	Set Ship Mode	0x04‖0x01, 0x04‖0x02

We interact with I2C by calling `writeToI2C(register, value)` and `readFromI2C(register)`. We *Read Voltage* from the `VC1-10` registers (address 0x0C to 0x1F). We *Set OVT* and *Set UVT* by writing to the `OV_TRIP` and `UV_TRIP` registers (address 0x9 and 0xA). For example, writing 0 to `UV_TRIP` sets the overvotlage threshold to 1580mV, its lower possible value. We force *No Balancing* on the battery by writing 0 to the `CELLBAL1` and `CELLBAL2` registers.

Mapping Attacks and Vulnerabilities

Table 4 shows the mapping between the E-Trojans attacks and vulnerabilities.

Table 4: Mapping between the five attacks and the four vulnerabilities. All attacks exploit V1, V2, and V3. Only the DES attacks exploit V4.

E-Trojans Attack	V1	V2	V3	V4
Overvoltage Battery Destruction (OBD)	✓	✓	✓	✗
Undervoltage Battery Ransomware (UBR)	✓	✓	✓	✗
User Tracking via Internals (UTI)	✓	✓	✓	✗
Denial of E-Scooter Services (DES)	✓	✓	✓	✓
Password Leak and Recovery (PLR)	✓	✓	✓	✗

ARTIFACT EVALUATED
USENIX
AVAILABLE

USENIX VehicleSec '26 Artifact Appendix: E-Trojans: Ransomware, Tracking, DoS, and Data Leaks on the Xiaomi E-Scooter Ecosystem

Marco Casagrande
EURECOM, Biot, France

Riccardo Cestaro
University of Padua, Padua, Italy

Mauro Conti
University of Padua, Padua, Italy

Eleonora Losiouk
University of Padua, Padua, Italy

Daniele Antonioli
EURECOM, Biot, France

A Artifact Appendix

A.1 Abstract

E-Trojans is a toolkit that implements five attacks on the Xiaomi M365 and ES3 (Mi3) electric scooters. We release a set of malicious BCTRL firmware that can be flashed on the e-scooter to alter its behaviour. For example, it can infect the e-scooter with a ransomware that induces undervoltage to permanently damage the battery. Each malicious firmware adds multiple malicious capabilities that we implement on the BMS after RE the stock Xiaomi firmware. The firmware speaks UART and I2C to communicate with the other entities on the UART (i.e., BTS and DRV) and I2C (i.e., BMON) bus. We also offer a proof-of-concept Android app that simulates ransom payment and its backend server, that includes a database and two web APIs. The toolkit works with minimal resources: a computer that supports BLE, Python (+ libraries), Java (+ libraries), MongoDB, Django, an Android phone (no root required), and a Xiaomi e-scooter (M365 or ES3).

A.2 Description & Requirements

The artifact consists of five folders.

Binary Tools. Contains Python scripts to analyzie firmware. No special requirements for testing. The binary-diff.py script identifies differences between two e-scooter binary firmware. This tool diffs firmware from different e-scooter models. It also helps reverse-engineering more recent and/or malicious firmware versions by directly comparing them to a baseline version. The binary-patcher script generates custom firmware compatible with most Xiaomi e-scooters (only M365 and ES3 have been properly tested). We also inlcude two legitimate BCTRL binaries for testing.

BMS Ghidra. Contains the reverse-engineered Ghidra projects for the Xiaomi 365 and ES3 BMS firmware. Requires Ghidra, and the installation of the STM8 Processor Modules (https://github.com/esaulenka/ghidra_STM8). The file can simply be opened and studied in Ghidra.

Malicious Payloads. Contains the malicious BMS firmware payloads. Requires an actual Xiaomi 365 or ES3 e-scooter to test. The payloads are firmware binary files that can be flashed on the e-scooter by our Ransom App (also released in this repository) or any other third-party app that allows for firmware updates, such as M365 DownG (https://play.google.com/store/apps/details?id=com.m365downgrade). As soon as the firmware is flashed on the e-scooter, the attack begins automatically. **FLASHING OUR FIRMWARE WILL LIKELY DAMAGE THE E-SCOOTER'S BATTERY AND SHOULD ONLY BE DONE ON TEST DEVICES. RESTORING IT WILL ALSO REQUIRE A COMPLICATED HARDWARE SETUP INVOLVING A ST-LINK DEBUGGER. DO IT AT YOUR OWN RISK.**

Ransom App. Contains the Android app that deploys our attacks, including the Undervoltage Battery Ransomware. Requires an Android smartphone. The app can scan for nearby Xiaomi e-scooters, flash a malicious firmware on them, perform our E-Trojans attacks, and restore an undervolted battery.

Ransom Server. Contains the files for the ransomware webserver and its database, to be used together with the ransom app. The webserver stores ransomware unlock keys and submits them to the ransom app if the user has the correct password (obtained by paying our ransom).

A.2.1 Security, privacy, and ethical concerns

Flashing our malicious firmware on an e-scooter leads to the safety hazards described in our paper, including battery overvoltage, undervoltage, rsik of fire and explosion, DoS, and permanent damage to the battery's health. If the reviewers want to test the attacks, they should contact us authors first. Experimenting with the is a safety-risk and should not be taken lightly. For this reason, we redacted parts of the malicious payload's code, making it not usable in this state.

This also means that people can not just clone our repo and perform dangerous attacks in the wild. They will need months of reverse-engineering. The other folders do not include dangerous code. They can be installed and ececuted freely. The ransom app and server perform basic non-malicious utility operations, but do not run any ransomware. The ransomware is contained inside the malicious firmware payload only.

A.2.2 How to access

We provide full and public access to our artifact in the following Zenodo DOI (https://doi.org/10.5281/zenodo.20529875).

A.2.3 Badges

We request the Available, Functional, and Reproduced badge.

A.2.4 Hardware dependencies

An Android smartphone. Bluetooth does not work on emulators, so emulated devices will have limited functionality. Optionally, if testing the attacks, a throwaway Xiaomi 365 or ES3 (Mi3) electric scooter.

A.2.5 Software dependencies

Standard requirements: Python and dependencies, Ghidra (with STM8 Processor Modules).

A.2.6 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

Installation of software dependencies is straightforward and described in the repo. There are no particular challenges.

A.3.2 Basic Test

Clone the Github repo and check all files are there.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Our toolkit can diff Xiaomi BCTRL firmware, and patch it to create malicious versions.
(C2): Our toolkit contains the reverse-engineering Ghidra projects for the Xiaomi M365 and ES3 BCTRL firmware.
(C3): Our toolkit contains the malicious BCTRL firmware that can be flashed into a real M365 and ES3 e-scooter.
(C4): Our toolkit contains the infrastructure for the ransomware attack, including an Android app and webserver.

A.4.2 Experiments

(E1): BCTRL Firmware Diff [2 human-minutes].
Preparation: Open the *Binary Tools* folder.
Execution: Run `python binary-diff.py` to diff two BCTRL firmware versions. You can use default files, or modify them within the script.
Results: Binary diffs are shown in the console.
(E2): BCTRL Firmware Patching [5 human-minutes].
Preparation: Open the *Binary Tools* folder.
Execution: Run `python payload-patcher.py` to create a new patched and malicious firmware from the list. In the menu, we suggest to select e-scooter version=1 or version=6, and attack=4 or attack=5. We redacted the code for the other listed attacks for safety reasons.
Results: A new firmware is created in the folder.
(E3): Ghidra Reverse-engineering [15 human-minutes].
Preparation: Install Ghidra and the STM8 Processor Modules (https://github.com/esaulenka/ghidra_STM8). Open the *BMS Ghidra* folder.
Execution: Open the M365 or ES3 project files with Ghidra.
Results: The files will be correctly analyzed. Function names will be shown, our comments in relevant parts of the code (e.g., firmware updates) will be visible, and the memory map will be interpreted correctly (e.g., RAM, ROM, and Peripherals).
(E4): Malicious Payloads [10 human-minutes].
Preparation: Open the *Malicious Payload* folder.
Execution: Check the presence of the malicious BCTRL firmware. Check the User Tracking + Denial-of-Service attack video (https://www.youtube.com/watch?v=Y2R46yeCXOQ).
Results: The malicious firmware has been released and there is video proof of it working.
(E5): Ransomware App and Webserver [10 human-minutes].
Preparation: Open the *Ransom App and Ransom Server* folders and use Python.
Execution: Deploy them by following the instructions in the repo. First, install the app on the Android phone, scan for nearby Xiaomi e-scooters, and see if there are any results (of course, there might not be Xiaomi e-scooters nearby). Second, install the webserver and its requirements.
Results: Android app and Python scripts have been released and work.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.

ARTIFACT EVALUATED
USENIX
AVAILABLE

RVDebloater: Mode-based Adaptive Firmware Debloating for Robotic Vehicles

Mohsen Salehi
The University of British Columbia

Karthik Pattabiraman
The University of British Columbia

Abstract

Embedded firmware is becoming increasingly larger, thereby expanding its attack surface. Despite the increase in firmware size, many embedded devices, such as robotic vehicles (RVs), operate in distinct modes, each requiring only a small subset of the firmware code at runtime. Debloating is an approach to reduce attack surfaces by removing or restricting unneeded code. Unfortunately, existing techniques suffer from significant limitations, such as coarse granularity and irreversible code removal, limiting their applicability. We propose *RVDebloater*, a novel adaptive debloating technique for *mode-based embedded devices* such as RVs that automatically identifies unneeded firmware code for each mode using either static or dynamic analysis, and dynamically debloats the firmware for each mode at the function level at runtime. *RVDebloater* uses a software-based enforcement approach to ensure that only the required functionalities are accessible from each mode and supports diverse *mode-based embedded devices*.

We implemented *RVDebloater* using the LLVM compiler and evaluated its efficiency and effectiveness on six different RVs, including both simulated and real ones, with different real-world missions. We find that device requirements change throughout its lifetime for each mode, and that many critical firmware functions can be restricted in other modes, with an average of 85% of functions not being required. The results showed that none of the missions failed after debloating with *RVDebloater*, indicating that it neither incurred false positives nor false negatives. Further, *RVDebloater* prunes the firmware call graph by an average of 45% across different firmware. Finally, *RVDebloater* incurred an average performance overhead of 3.9% and memory overhead of 4% (approximately 0.25 MB) on real RVs, and also prevented three real attacks.

1 Introduction

Robotic vehicles (RVs) such as unmanned aerial vehicles (UAVs), unmanned ground vehicles (UGVs), and remotely operated underwater vehicles (ROVs), are becoming increasingly popular. A wide range of features (e.g., system calls and device drivers [43]) and hardware functionalities is being incorporated into these devices to support the requirements of different users, even though each user may only require a fraction of these features. This expansion enhances potentially buggy code, even in unused features, thereby increasing the devices' attack surface [17, 40, 50]. For instance, UAVs are vulnerable to various security vulnerabilities, such as buffer overflows [40, 45], which can allow attackers to hijack the control flow and manipulate critical programs like the flight control program, causing crashes.

One approach to reduce attack surfaces is minimizing or restricting unwanted features (e.g., security-critical system calls) through a technique known as *debloating* [31, 33, 37, 38, 41, 42, 45, 49]. Debloating *keeps* a *small subset* of firmware code necessary for specific workloads while *restricting* the execution of the remaining code [31, 36, 45, 52], or *removes* unwanted features such as peripherals (e.g., USB and UART [41, 42]). Although these techniques have gained popularity in desktop [31, 37] and server applications [33, 44, 52], embedded systems have not received sufficient attention [45].

Debloating techniques can be categorized into two main classes: *fixed* and *adaptive* techniques. *Fixed debloating techniques* such as SHARD [31], Minion [45], and FACE-CHANGE [37] analyze applications throughout the entire lifetime, without considering how applications' requirements change during different execution phases. For instance, critical firmware code or system calls like `execve` or `fork` are usually executed only during the initial moments of an application's lifetime. However, *fixed debloating techniques* often fail to disable such system calls or code, leaving these available to be used in exploits [36].

In contrast, *adaptive debloating techniques* [36, 52] consider the changing requirements of an application through different execution phases. For example, Ghavamnia et al. [36] and Rajagopalan et al. [52] proposed temporal system call specialization techniques for server applications, which restrict available system calls based on the application's execution phase. They use static analysis techniques to identify the required system calls for two distinct phases—*initialization*

and *serving*—and block unneeded system calls, such as `fork` and `execve`, after transitioning to the *serving* phase.

Unfortunately, adaptive debloating techniques [36,52] have two limitations. First, similar to fixed debloating techniques, they focus on disabling security-critical system calls such as `execve`. However, embedded devices' firmware, particularly those used in RVs (e.g., ArduPilot [9], PX4 [23]), contains critical functions, such as `disarm_motors`, which should only be executed in specific modes[1], e.g., `Turtle` mode (i.e., flipping the copter). If not, they can lead to serious safety violations.

Second, these techniques use a Linux mechanism known as Seccomp-BPF to restrict the set of available system calls for each phase [25]. However, the Seccomp-BPF mechanism for filtering system calls is irreversible, meaning that once a system call is filtered out, it cannot be reallowed. In contrast to server applications, which typically have a simple control flow and do not revert to the previous phase, RVs often experience frequent mode changes, including transitions back to earlier modes (e.g., `AUTO <-> LOITER`). Therefore, debloating techniques that cannot *dynamically* adjust access restrictions on firmware functions at runtime, such as those relying on Seccomp-BPF, are unsuitable for embedded devices.

Our key insight is that many embedded devices-such as RVs-operate in distinct modes (e.g., `LAND` or `TAKEOFF`) at runtime, with each mode requiring a specific set of functionalities, implemented as functions in their firmware. We call these devices mode-based embedded devices. With this insight, we propose *RVDebloater*[2], an adaptive debloating approach that automatically identifies the set of functionalities required by each mode (e.g., `TAKEOFF`) at the function-level and then monitors the device at runtime to ensure that only those firmware functions are allowed to execute when the device operates in the corresponding mode. For instance, when RVs are in `AUTO` mode, travelling from point A to B, they do not require the motor disarming function designed for `LAND` mode; misuse of this function can result in crashes. *Thus, we can restrict these functionalities (firmware functions) to their respective modes at runtime.* In this paper, we focus on RVs as a prominent class of embedded devices and evaluate *RVDebloater* on six RVs; however, *RVDebloater* can be generalized to other *mode-based embedded devices*, which we leave to future work.

RVDebloater has two main goals: 1) Specializing firmware code by *dynamically* adapting it to changes in the device's requirements at the mode level throughout its execution lifetime, and 2) Enforcing restrictions on firmware code at *function-level granularity*, providing finer specialization than system call-level [31,36,52] or hardware-level [41,42] (e.g., disabling unused peripherals) techniques.

RVDebloater addresses the above goals through two innovations. First, it uses either *static* or *dynamic analysis* to identify per-mode firmware code coverage, while considering changes in the device's requirements throughout its execution (i.e., the set of firmware functions required for each mode). Second, *RVDebloater* uses a reversible, software-based enforcement technique that statically instruments indirect control flow transfer instructions (e.g., indirect call sites and function return instruction) and the mode-switching function to redirect control flow, monitor function execution and mode transitions, and restrict access to functions belonging to other modes. Upon a mode switch, *RVDebloater* loads the required firmware functions for the new mode by applying per-mode configurations (goal 1), including *required* functions (goal 2) identified by either *static* (*RVDebloater-static*) or *dynamic analysis* (*RVDebloater-dynamic*). Henceforth, *RVDebloater* refers to both variants unless stated otherwise.

To the best of our knowledge, RVDebloater is the first fine-grained, adaptive debloating framework for mode-based embedded devices that automatically identifies and specializes the firmware code at the function-level for each mode.

Our main contributions are summarized as follows:

- Propose two automated analysis techniques (static and dynamic) to identify the required firmware code at the function-level, tailored to the operational requirements of RVs across different modes (e.g., `LAND`, `TAKEOFF`).
- Propose a runtime adaptive debloating technique that tracks mode switches and a novel reversible software-only enforcement technique to dynamically specialize the firmware code for each mode at runtime, thereby supporting frequent mode changes.
- Design *RVDebloater*, a fine-grained debloating framework to integrate the above techniques for mode-based embedded devices such as RVs. *RVDebloater* is implemented using the LLVM compiler [18], making it portable across different devices. Further, it requires no special hardware support (e.g., MPU or MMU), unlike prior compartmentalization and hardware-based isolation techniques [35,59], making it suitable for resource-constrained RVs.
- Evaluate *RVDebloater* on four simulated and two real RVs, all running one of two widely used open-source autopilot systems, ArduPilot or PX4, across a diverse range of scenarios and missions. We also evaluated it against three realistic code-reuse attacks on the RVs.

The results demonstrate that (1) *RVDebloater-static* reduces the attack surface by an average of 41% of the firmware functions. (2) *RVDebloater-dynamic* identifies all *required* functions for each mode by running RVs with only up to 10 missions, and achieves an average attack surface reduction of 85%. (3) Neither *RVDebloater-static* nor *RVDebloater-dynamic* incurred any false-positives (i.e., no mission fails due to a missing function), (4) The average performance overhead and power consumption overhead of *RVDebloater* on

[1] We use *modes* (e.g., `RTL`, `AUTO`) to refer to the *phases* of the RV.

[2] RVDebloater is available at: https://github.com/DependableSystemsLab/RVDebloater.git

real RVs are 3.9% and 0.47%, respectively. Further, the overall mission time was not increased due to *RVDebloater*. (5) The firmware instrumented by *RVDebloater* incurred an average memory overhead of about 4%, which is about 0.25 MB, and (6) *RVDebloater* successfully prevented all three attacks.

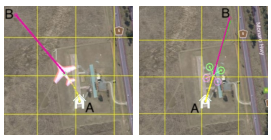
2 Background

2.1 Memory Corruptions

The majority of embedded devices' software, including firmware, is developed in low-level programming languages (i.e., C or C++) due to their efficiency and ability to provide full control over the underlying hardware. However, these languages are type-unsafe, requiring developers to carefully manage memory accesses and ensure their validity to prevent issues like dereferencing invalid pointers. The lack of memory safety in these languages, combined with developers' failure to meet these responsibilities, leads to memory corruption vulnerabilities (e.g., buffer overflow) that attackers can exploit to manipulate device behavior maliciously or even gain full control over the system's execution flow [57] (e.g., control flow hijacking attacks). Control flow hijacking attacks occur when a code pointer, such as a return address (backward edge) or a function pointer (forward edge), on the backward edge (e.g., return addresses) or forward edge (e.g., function pointers) becomes corrupted.

2.2 Mode-based Embedded Devices

Robotic vehicles (RVs) switch between different modes such as `AUTO` and `RTL` while performing missions like delivering a package from point A to B. Examples of simple missions for the ArduPilot firmware are shown on the right for ArduPlane and ArduCopter.



These modes are predefined in RVs' autopilot software such as ArduPilot [9] and PX4 [23]. Further, to manage their tasks, mode-based embedded devices often rely on real-time operating systems (RTOS) such as NuttX [3]. All these software components—the autopilot software, RTOS, and other essential modules like device drivers—are integrated into the device's firmware. As an example, Figure 1 presents mode switching in *ArduCopter* [4,10], a sub-platform of the ArduPilot software. The list of ArduPilot and PX4 modes is presented in the extended version of this paper available on arXiv [55].

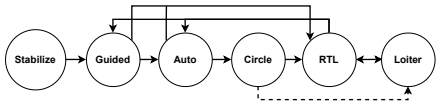


Figure 1: Mode switching process in ArduCopter.

Many embedded devices, including RVs, often lack hardware security features, such as Memory Management Unit (MMU), which results in all applications being mapped into a single memory space [45]. Therefore, these applications have unrestricted access to the entire shared memory in the privileged mode, enabling attackers to exploit a single memory corruption vulnerability to take over the entire system.

Furthermore, these devices may not support Memory Protection Units (MPUs) [53]. Even when an MPU is available, configuring and controlling memory access presents challenges due to its limitations. Unlike the MMU, which enforces access restrictions through page tables and virtual memory, an MPU operates with a limited number of hardware registers, typically supporting only 8 to 16 memory regions (e.g., 8 regions in ARMv7-M [35]). Each region is protected using read, write, and execute permissions, configurable for both privileged and unprivileged modes. Also, MPU regions must be a power of two in size, with a minimum size of 32 bytes, and their starting addresses must be a multiple of their size. This severely restricts MPUs' use in these devices.

3 Motivation and Threat Model

3.1 Debloating Techniques' Limitations

L1: Treating Applications as Monolithic Entities with Fixed Requirements. *Fixed techniques* in attack surface reduction [31, 38, 45, 48] specialize or remove functionality based on the entire application's lifetime requirements. However, *adaptive debloating techniques* [36,52] have shown that considering the changing requirements during the application's execution phases is more effective. In particular, many devices and programs, such as mode-based embedded devices and server applications, operate in distinct modes, with their requirements changing based on the execution phase.

L2: Removing Functionalities. *Fixed debloating techniques* [41,42] often remove functionalities, such as hardware features or kernel source code, that are not required at any point during the applications' lifetime. On the other hand, these techniques retain critical functionalities used for legitimate purposes during the application's operation, while some of these functionalities are only executed for a short period of time. Therefore, to prevent over-debloating and potential system crashes, these techniques often overestimate the functionalities required. Retaining such functionalities would inflate the attack surface.

L3: Using MPU. Some *Fixed debloating techniques* [35, 45] use MPUs to control memory access. However, this hardware may not be available on all devices, particularly embedded systems [53]. Moreover, due to MPU limitations (§2.2), these methods must rely on clustering techniques to group code for multiple processes. Thus, processes may gain access to unrequired code, thus leading to security risks.

L4: Considering System Calls. Existing debloating techniques, both *fixed* and *adaptive*, only focus on system calls to reduce attack surfaces. However, other functionalities implemented as firmware functions, such as those in RV autopilot software, may also pose security risks if accessible throughout the device's execution lifetime. For instance, Listing 1 and Listing 2 show two security-critical functions in ArduPilot for the `Land` and `Turtle` modes, respectively. The purpose of the function shown in Listing 1 is to land the RV when it is in specific states, such as upon reaching its destination. Also, the function in Listing 2 is executed to disarm the motors if the RV lands upside down, and enters the `Turtle` mode to flip the drone upright. However, executing `disarm_motors` during flight immediately kills the motors, causing a crash. Thus, these functions should not be accessible in other modes, such as `AUTO` or `GUIDED`, as attackers could exploit vulnerabilities to execute these functions.

```
1 void ModeLand::run () {
2   if (control_position){
3         gps_run();
4    }else {
5         nogps_run();
6    }
7 }
```

Listing 1: Landing function for ArduCopter's Land mode.

```
1 void ModeTurtle::disarm_motors(){
2    //... omit
3    motors->armed(false); // disarm
4     //... omit
5    hal.util->set_soft_armed(false);
6 }
```

Listing 2: ArduCopter's function executed in Turtle mode to disarm the vehicle motors if it lands upside down.

L5: Irreversible Mechanism. Although *adaptive debloating techniques* like SysPart [52] and TSCP [36] are more effective at reducing attack surfaces compared to *fixed techniques*, they employ an irreversible mechanism to disable system calls after transitioning to the *serving* phase. Specifically, these techniques leverage a Linux kernel feature called Seccomp-BPF, which is a one-way process, meaning the filtered system calls cannot be re-enabled. As a result, they are suitable for applications with simple control flow, such as *server programs*, but not for devices like *mode-based embedded devices* that frequently switch between modes, and transition back to earlier modes (Figure 1).

3.2 Our Approach: *RVDebloater*

RVDebloater proposes two solutions (i.e., S1 and S2) to address the limitations of existing debloating techniques.

S1: Addressing L1 and L4. We propose an adaptive debloating technique that specializes firmware code based on the device's requirements during its mode of execution (L1). This technique, detailed in §4.2, employs either a static or dynamic analysis to identify the required firmware code for each mode at the function-level (L4).

S2: Addressing L2, L3, and L5. We use a software-based technique that does not rely on hardware features to enforce firmware debloating based on the device's current execution mode throughout its execution lifetime (L5). Specifically, *RVDebloater* monitors mode switches and controls memory access by instrumenting the firmware's mode-switching functions and indirect control flow transfer instructions to restrict access to required firmware functions based on the device's current mode (L2 and L3).

3.3 Assumptions and Threat Model

Assumptions. We make the following two assumptions. First, the list of autopilot software modes (e.g., `TAKEOFF` and `AUTO`) and mode-switching functions responsible for changing modes at runtime are known to *RVDebloater* and provided as inputs to it. This information can be found in the firmware's source code or autopilot software's documentation (e.g., ArduPilot documentation [4, 10]). Although determining mode-switching function could be automated through dynamic analysis, we did not invest the effort to develop this capability, as it only needs to be performed once per firmware model (e.g., ArduPlane). It took us only a few minutes to identify this function in six different firmware models.

Second, similar to prior debloating techniques [31, 36, 45], we assume that *RVDebloater* has access to the firmware's source code. With that said, since *RVDebloater* operates on the LLVM IR [18], it can also work if the firmware is available in the LLVM IR format instead of source code.

Threat Model. We consider an adversary who is capable of performing remote attacks on the system without having root or physical access to it. Furthermore, we assume that the core system software is benign and developed by honest developers; however it may contain memory corruption vulnerabilities (e.g., buffer overflows). An adversary can exploit such memory corruption vulnerabilities, to hijack the firmware's control flow and call unneeded critical firmware functionalities via code-reuse attacks (e.g., through ROP or JOP gadgets). Hardware-level corruptions, physical attacks such as sensor spoofing, and side-channel attacks are outside the scope of this work. Our threat model is consistent with those of prior debloating techniques [31, 45].

4 Design: *RVDebloater*

This section presents the system design of *RVDebloater*, the first adaptive debloating technique for *mode-based embedded devices*. We start with an overview of *RVDebloater* and its key innovations (§4.1), before describing the different phases of the system in detail (§4.2, §4.3 and §4.4).

4.1 Overview of *RVDebloater*

Figure 2 shows *RVDebloater*'s workflow, which includes three main phases: (1) the analysis phase, (2) the guard insertion phase, and (3) the monitoring phase. Our approach aims to identify and dynamically specialize firmware code[3] running on mode-based embedded devices at the function-level granularity (i.e., firmware functions), depending on the device's execution mode. To accomplish this goal, we propose two key innovations. The first innovation is using either a static analysis or dynamic analysis to identify the firmware functions used in a specific mode (see S1 in §3.2). We leverage this method in the *analysis* phase.

The second main innovation of our technique is that, unlike previous debloating methods that rely on hardware components (MPU) [45], virtualization mechanisms (VMX) [31], or irreversible mechanisms (Seccomp-BPF) [36, 52], our approach uses a completely *software-based enforcement technique* (see S2 in §3.2). In particular, our *software-based enforcement technique* uses *static analysis* to instrument indirect control flow transfer instructions (e.g., indirect function call or function return instruction) and redirect the control flow to the *RVDebloater* monitor, which specializes firmware code for each mode using a mode-specific configuration file that includes the required functions and their addresses generated in the previous phase (i.e., *analysis* phase). This is achieved in two phases: (1) the *guard insertion* phase, which modifies the firmware code, and (2) the *monitoring* phase, which tracks the device's execution to check indirect control flow transfer instructions and specialize the firmware for each mode. As shown in Figure 2, the *analysis* and *guard insertion* phase are both performed offline, while the *monitoring* phase is performed online.

After generating the mode-based configuration file (*analysis* phase) and executing the instrumented firmware (*guard insertion* phase) on the device, the *RVDebloater* monitor is triggered when an indirect control flow transfer instruction is executed (*monitoring* phase). The monitor checks whether the jump and execution of the target address are allowed based on the configuration file. If the target is not present in the static output (*RVDebloater-static*) or in the dynamic output (*RVDebloater-dynamic*) (depending on which is used), the monitor can either activate safety mechanisms such as `Fail-Safe` mode or take other actions based on device maintainers' preferences (e.g., mechanisms they have developed). `Fail-Safe` mechanisms are commonly employed in various detection and recovery techniques [34, 45, 46, 51] to safely control the vehicle by returning it to the launch point or performing an emergency landing when an attack is detected or control is lost. ArduPilot provides 11 well-defined fail-safe mechanisms [6], and PX4 also includes a fail-safe landing [22]. Thus, we use fail-safe mechanisms in our evaluation.

[3] Embedded firmware consists of code from various software modules, such as libraries and applications, compiled into a single executable file.

We provide a detailed discussion of the process in the following sections (§4.2, §4.3 and §4.4). Note that, for simplicity, the examples are presented in the C language, even though *RVDebloater* operates on LLVM IR. Further, to describe the examples, we treat each line of C code (i.e., program statement) as a single instruction, even though, in practice, each of these would correspond to multiple LLVM instructions.

4.2 Analysis Phase

The first phase of the *RVDebloater* process is the offline analysis phase, which automatically identifies the *required functions* for each mode using either *static* or *dynamic* analysis. Since static analysis tends to overestimate the required functions for each mode, while dynamic analysis tends to underestimate them, *RVDebloater* leverages *both techniques*, giving device maintainers the flexibility to choose the most appropriate option based on their level of risk-tolerance.

Specifically, the offline analysis phase consists of either: (1) executing the device with different missions to dynamically identify the functions used in each mode at runtime (*RVDebloater-dynamic* §4.2.1), OR (2) generating a pruned call graph of the firmware to statically identify an overestimated set of required functions for each mode (*RVDebloater-static* §4.2.2). Subsequently, *RVDebloater-dynamic* uses the dynamic output to enforce firmware specialization based on the device's current mode, while *RVDebloater-static* uses the identified function lists for each mode from the static output.

4.2.1 *RVDebloater-dynamic*: Dynamic Analysis

To enforce debloating, *RVDebloater* leverages dynamic analysis to identify the required firmware functions for each mode (*RVDebloater-dynamic*). To do so, *RVDebloater* employs static instrumentation to generate a modified firmware version that records (1) mode switches, and (2) the firmware functions executed in each mode at runtime. Afterwards, *RVDebloater* runs the instrumented firmware on the device with various missions to determine the executed (*required*) firmware functions for each mode. The following outlines each step.

(1) Instrumentation. This step inserts trampolines[4] into the mode-switching function(s)[5] and firmware functions. *RVDebloater* provides an instrumentation component that automates this process. This component takes the firmware and mode-switching function(s) (§3.3) as input and outputs the instrumented firmware. Specifically, the instrumentation component statically analyzes the firmware to locate the given mode-switching function(s) and other firmware functions, then inserts the trampolines. To ensure that the mode has changed successfully, the component inserts a trampoline at

[4] A single instruction, like a jump or branch instruction, inserted into the program to redirect its original control flow.

[5] Although *RVDebloater* supports multiple mode-switching functions, in the tested firmware (e.g., ArduCopter, PXCopter), instrumenting a single mode-switching function was sufficient to monitor all mode changes.

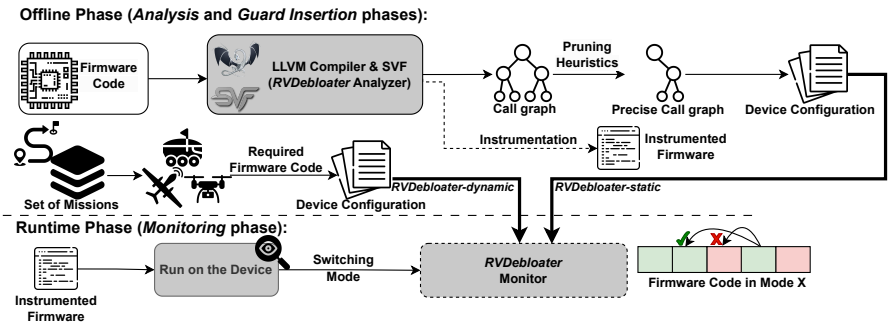


Figure 2: Overview of *RVDebloater*'s operation, including both the offline phase (top) and runtime phase (bottom).

the end of mode-switching function(s). If a mode switch fails, the firmware handles the error independently of *RVDebloater*.

```
1 bool Plane::set_mode_by_number(const Mode::
    Number new_mode_number, const ModeReason
    reason){
2     Mode *new_mode = plane.mode_from_mode_num(
    new_mode_number);
3     if (new_mode == nullptr) {
4         notify_no_such_mode(new_mode_number);
5         return false;
6     }
7     Trampoline
8     return set_mode(*new_mode, reason); }
```

Listing 3: Output of the instrumentation step for the mode-switching function in ArduPlane (ArduPilot).

Listing 3 shows an example of the mode-switching function in ArduPlane. This function first (line 2 in Listing 3) retrieves the name of the new mode (e.g., `CIRCLE` or `RTL`) by passing the input number (`new_mode_number`). Then, it checks the returned mode name to validate whether the new mode exists (lines 3 to 6). Finally, it calls a function to switch the current mode to the new mode (line 8). As seen in Listing 3, the component has inserted a trampoline at the end of the function (line 7) after the mode change has been successfully validated (lines 3 to 6). When the inserted trampoline is triggered at runtime, it passes the current and new mode to *RVDebloater*'s profiler, records the executed functions in the current mode, and changes the profiler's mode to the new mode.

Further, the component inserts trampolines at the beginning of all firmware functions (except the mode-switching function) to ensure that every function executed in the current mode triggers *RVDebloater*'s profiler. Algorithm 1 demonstrates the workflow of the instrumentation step. For the mode-switching function, the component inserts a trampoline at the end (lines 2 to 5). For all other functions, it adds trampolines to the beginning (lines 6 to 9).

(2) Dynamic Profiling. In the next step, *RVDebloater* runs the device with the instrumented firmware (e.g., ArduPlane or ArduCopter) across different missions and scenarios (e.g., with and without obstacles) to identify the firmware functions

Algorithm 1: Instrumentation for Profiling

```
1  instrumentationComponent (Functions)
       inputs : Firmware Functions and Mode-switching Function Set
                (modeFuncs)
       output : Instrumented Firmware
2      if Function ∈ modeFuncs then
3          if End of Function then
4              location := findLocation();
5              insertTrampoline(location);
6      else
7          if Function Entrance then
8              location := findLocation();
9              insertTrampoline(location);
10     return instrumentedFirmware;
```

for each mode at runtime by capturing functions with *RVDebloater*'s profiler. The instrumented firmware ensures that each function for the current mode is logged at most once when the device switches to a new mode. We assume that *RVDebloater* runs this step (dynamic profiling) in a benign environment, free from any security attacks.

4.2.2 *RVDebloater-static*: Static Analysis

RVDebloater also statically analyzes the firmware to construct its call graph. Since the firmware is written in C/C++, it often relies on indirect function calls via function pointers. Statically resolving the targets of these calls is both challenging and critical for accurately identifying the functions required by each mode. To address this, *RVDebloater* employs Andersen's points-to analysis [32], a static technique for resolving pointer targets in C/C++ programs. This analysis can have different sensitivities depending on how objects in memory are modeled: field sensitivity, path sensitivity, and context sensitivity. Higher levels of sensitivity yield more accurate results but also increase the analysis time and the implementation complexity. For instance, field sensitivity distinguishes and models each field of a structured variable (e.g., a `struct`) separately, while context sensitivity distinguishes between different call sites of a function, meaning that when the same

function is called from different places, each call gets its own context and is analyzed independently of the others. While a detailed discussion of these topics is beyond the scope of this paper, more details about these sensitivities can be found in previous work [36, 39]. Specifically, *RVDebloater* leverages SVF [56], which implements Andersen's analysis with field sensitivity but without path or context sensitivity.

As a result, SVF tends to over-approximate the call graph, as it associates indirect function calls with a large number of potential targets—only a small subset of which is actually executed, while the rest are spurious. Although this over-approximation increases the number of functions identified per mode, introducing unneeded functions that increase the attack surface and could be exploited by attackers to subvert the system, the approach is sound in determining which functions are not required (unreachable) for each mode. However, to improve precision, *RVDebloater* applies two pruning heuristics, inspired by the TSCP technique [36], to remove spurious targets caused by the lack of path and context sensitivity and generate a more accurate call graph. We modified these heuristics to support embedded firmware and refine call graphs further. The call graph is generated once per firmware model (e.g., ArduCopter), independent of the number of modes. The two heuristics are as follows. More details are §A in the Appendix.

(1) Signature-based Pruning Heuristic. SVF only prunes the spurious targets by checking the number of arguments, whereas *RVDebloater* uses two additional filtering methods: considering argument types and matching return types when resolving indirect call targets.

(2) Address-based Pruning Heuristic. In contrast to Andersen's algorithm, which assumes that all functions are reachable from the entry point, *RVDebloater* identifies and removes edges that point to functions whose addresses are not reachable from the entry point.

After refining the call graph, *RVDebloater* identifies the entry functions of each mode to determine their reachable (required) functions. To do so, it employs an instrumentation component similar to the one used in dynamic analysis (§4.2.1), which instruments both the firmware functions and the mode-switching function(s). *RVDebloater* then executes the device through a mission that includes all modes intended for monitoring and control at runtime by *RVDebloater-static*. During execution, when the device switches modes, *RVDebloater* tracks the function(s) invoked by the mode-switching function to initialize the parameters required for the new mode and marks them as the entry function(s) of that mode.

RVDebloater also performs mode-name pattern matching to confirm the validity of the identified entry function(s). In particular, we observed that these firmware models (e.g., ArduCopter or ArduPlane) have entry function(s) responsible for managing mode-specific parameters, and their names typically include the mode name itself. Therefore, *RVDebloater* uses the list of mode names provided as input (§3.3) to verify and confirm the correctness of these functions. We found that all identified entry functions were correct. Note that this pattern can be adapted by developers to support other firmware. However, in all the firmware models evaluated in this paper, this pattern was sufficient. Finally, *RVDebloater* uses the generated call graph to identify the reachable and unreachable functions for each mode, starting from its entry function(s).

Both *dynamic* and *static* analyses only need to be performed *once* for each firmware model (e.g., ArduPlane or ArduCopter) to identify and store the functions required for each mode at runtime. These identified functions are then used during the runtime (device execution) monitoring phase to enforce debloating by checking whether they are permitted in the current mode of the device.

4.3 Guard Insertion Phase

The next phase of the *RVDebloater* process is the *guard insertion* phase, which is performed offline and only once per firmware. The *RVDebloater monitoring* phase is designed to: (1) track mode switches, (2) load the appropriate list of identified functions for the new mode when a mode switch occurs, and (3) monitor indirect control flow transfer instructions and ensure they are allowed; otherwise, *RVDebloater* logs the violation and switches to `Fail-Safe` mode. The *RVDebloater* process has two main phases: **guard insertion** and **monitoring** (§4.4).

Similar to the previous instrumentation step for offline analysis phase, *RVDebloater* provides a component that automatically identifies mode-switching function(s) and inserts a trampoline at the end of each. However, in this instrumentation step, *RVDebloater* replaces *all* indirect control flow transfer instructions in the firmware code with calls to the *RVDebloater* monitor. This allows *RVDebloater* to observe all function calls and returns, and verify them against the list of allowed functions identified in the previous phase.

This component takes a list of mode-switching function(s) and the firmware code as input (§3.3), and outputs an instrumented firmware where trampolines are statically added to the mode-switching functions, and indirect control flow transfer instructions are replaced accordingly.

Algorithm 2 describes the workflow of this instrumentation component for monitoring. Similar to the prior instrumentation component (Algorithm 1), this component inserts trampolines at the end of mode-switching functions (lines 2 to 5). For other firmware functions, it iterates over their instructions, and replaces each indirect control flow transfer instruction (e.g., indirect call site or return instruction) with an instruction (i.e., `call "monitor_fn"`) that redirects the control to *RVDebloater*'s monitor (lines 6 to 15). *RVDebloater*'s monitor checks the call target before calling the intended target address directly. Section 4.4 provides details. Note that *RVDebloater* restricts enforcement to indirect call sites and return instructions; direct calls are inherently safe, as their targets are statically fixed in the binary, and cannot be redirected

Algorithm 2: Instrumentation for Monitoring

```
1  instrumentationComponent (Functions)
       inputs : Firmware Functions (Insts = inst_1 ... inst_n) and
                Mode-switching Function Set (modeFuncs)
       output : Instrumented Firmware
2      if Function ∈ modeFuncs then
3          if End of Function then
4              location := findLocation();
5              insertTrampoline(location);
6      else
7          foreach instruction inst_i ∈ Insts do
8              if inst_i.type is a indirect_cotrol_flow_instruction
                 then
9                  location := findLocation();
10                 replaceInst(location);
11             else if inst_i.type is a return_instruction then
12                 location := findLocation();
13                 replaceInst(location);
14             else
15                 continue;
16     return instrumentedFirmware;
```

through memory corruption attacks (§2.1).

4.4 Monitoring Phase

The final phase of the *RVDebloater* process is the *monitoring phase*, which operates after the *analysis* and *guard insertion* phases have determined the required functions for each mode (either static or dynamic output) and instrumented the firmware. The purpose of the monitoring phase is to *monitor* the device and *dynamically debloat* (specialize) the firmware functions for each mode based on the identified lists at runtime, such as during RV missions. To do so, *RVDebloater* uses the instrumented firmware from the previous phase (§4.3) and the list of functions identified for each mode, which is generated during the analysis phase (§4.2.2 or §4.2.1). The procedure of monitoring call sites and returns is illustrated in Figure 3. The figure consists of five steps (labelled (1) to (5)) and four phases: the process of switching modes in the RV, an access control table for each mode, an example of Mode A at runtime, and the *RVDebloater* monitor component.

Once a mode switch occurs, the mode-switching function jumps to *RVDebloater*'s monitor (step (1)) to update the access control tables for the new mode (step (2)). For example, as shown in Figure 3 (steps (1) and (2)), when the device mode changes from *Mode B* to *Mode A*, the access control table containing the required functions for *Mode A* is loaded from the generated configuration file.

As mentioned, *RVDebloater* rewrites call instructions to redirect them to its monitor (step (3) in Figure 3). When Function X calls Function Z, control is transferred to the *RVDebloater* monitor, which checks whether the call is permitted using the loaded access control table for the current mode (based on either the static or dynamic output).

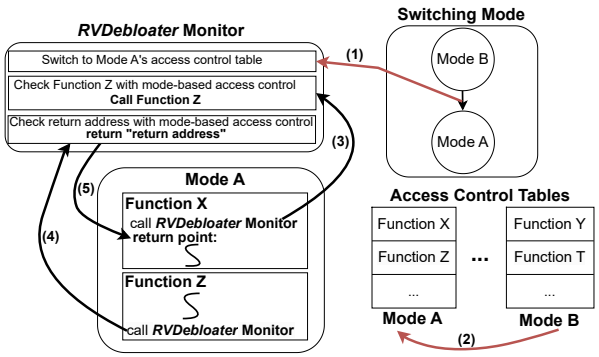


Figure 3: *RVDebloater* Runtime System.

If calling Function Z is permitted, the monitor saves the return address (e.g., the "return point" in Function X, as shown in Figure 3) and then invokes the function. Otherwise, if the function call is not authorized based on the static output for *RVDebloater-static* or the dynamic output for *RVDebloater-dynamic*, *RVDebloater* triggers an alert and switches the device to `Fail-Safe` mode, as the call is unauthorized according to the respective analysis.

After finishing the execution of Function Z, the return instruction jumps back to the monitor (step (4) in Figure 3), which verifies the stored return address. If the address matches the expected return address, control flow returns to Function X (step (5) in Figure 3). Otherwise, the monitor performs the same approach as for an unauthorized function call (i.e., switching to `Fail-Safe` mode).

5 Implementation

We implemented *RVDebloater*'s analysis phase using two LLVM module passes, which can be executed on firmware containing different functions [29]. Furthermore, we developed two functions that are linked to the firmware to support the instrumentation component and *RVDebloater*'s profiler. As mentioned in §4.2, this phase takes the mode-switching function(s) and firmware code as input and generates an instrumented firmware along with a list of required functions for each mode (configuration file). We also utilized SVF [27] to generate the firmware's call graph. However, we modified and added two LLVM passes to refine the generated call graph (i.e., signature-based and address-based pruning heuristics).

Further, the guard insertion phase is implemented as two LLVM module passes, and is executed on the firmware. This generates an instrumented firmware that tracks mode changes and controls function execution based on the identified required functions. Finally, the runtime monitoring phase monitors mode switching and indirect control flow transfer instructions to *dynamically* specialize the firmware code for each mode according to the configuration file. The implementation

details are discussed in §B in the Appendix.

6 Evaluation

This section describes the evaluation of different aspects of *RVDebloater* including the effectiveness and efficiency of the system. Effectiveness is the ability of *RVDebloater* to specialize the firmware code for each mode at runtime and reduce the attack surface, while the efficiency is the runtime and memory overheads of *RVDebloater*.

6.1 Experimental Setup

We evaluated *RVDebloater* on four simulated RV systems using either ArduPilot [9] or PX4 [23]—two popular open-source autopilot platforms—to test whether *RVDebloater* can work across different platforms without modification: (1) ArduPilot's quadcopter [5] (ArduCopter), (2) ArduPilot's quadplane [7] (ArduPlane), (3) ArduPilot's ground rover [8] (ArduRover), and (4) PX4's quadcopter [21] (PXCopter). We used Gazebo [15], QGroundControl [24], and APM SITL (Software-In-The-Loop) [9] platforms for simulations.

We also used two real RVs, shown to the right: **(Left)** Aion R1 ground rover (Aion rover) [2] and **(Right)** DIY drone (Pixhawk drone) [47].



Both are commodity RV systems based on the Pixhawk [19]. Pixhawk is a platform built on an ARM Cortex architecture that integrates ArduPilot or PX4 firmware, a flight management unit (FMU), memory, sensors, and I/O interfaces.

To generate the firmware call graph using the *RVDebloater*'s modified SVF, statically extract the required (reachable) firmware functions for each mode (*analysis* phase), and perform instrumentation during both the *analysis* and *guard insertion* phases, we used a desktop machine equipped with an Intel Xeon E-224 CPU (4 cores, 3.4 GHz) running Ubuntu 22.04 (64-bit). After identifying the required functions for each mode and generating the instrumented firmware, we tested *RVDebloater* using simulations and real devices.

Note that the Aion rover and the Pixhawk drone use the same platforms as ArduRover in ArduPilot and PXCopter in PX4, respectively; therefore, the evaluation results for ArduRover and the Aion rover, and for PXCopter and the Pixhawk drone, are similar. Thus, we do not report them separately, except when evaluating the efficiency in §6.3.

RV Missions: To evaluate the effectiveness of *RVDebloater*, we execute a diverse set of 40 missions (for each RV) with varying durations and distances, randomly selected to cover a range of scenarios. Each mission consists of a navigation path that begins at a starting location, passes through several intermediate waypoints, and ends at a specific destination. These missions emulate a range of real-world RV missions, including straight-line paths for last-mile delivery drones [28], polygonal paths for rovers in warehouse management [11], and both polygonal and circular paths for agriculture and surveillance [14]. The types of missions used for each RV are summarized in Table 1.

During each mission, the RV switches between different modes, such as `CIRCLE` and `LOITER` for ArduCopter and ArduRover, or `QHOVER` and `QLOITER` for ArduPlane. The main RVs' modes are outlined in the extended version of this paper available on arXiv [55]. Although our mission set includes other modes, this section presents the results for the main RV modes that we observed are used in real-world missions (Tables 2 and 3).

Table 1: Mission sets for each RV firmware type used to evaluate *RVDebloater*. In the "Missions" column, we use SL: straight line, MW: multiple waypoints, HFE: hover at fixed elevation, CP: circular paths, PP: polygonal path.

RV (Firmware) Type	Missions					
	Total	SL	MW	HFE	PP	CP
ArduCopter	40	10	12	5	5	8
ArduRover	40	12	13	-	7	8
ArduPlane	40	11	13	3	3	10
PXCopter	40	10	12	5	5	8

Metrics: We used four metrics to evaluate *RVDebloater*, namely (1) ***missed functions*** are those that *RVDebloater*'s dynamic profiler fails to identify during profiling missions using dynamic analysis. (2) ***false positive rate (FPR)*** is the percentage of mission failures caused by the absence of required functions in the configuration file (either static or dynamic results) for the current mode, despite the RV needing them for the mission's success. (3) ***false negative rate (FNR)*** is the percentage of functions that *RVDebloater* allows during missions, even though these functions are not required by the RV's current mode. (4) ***precision*** of the call graph, which refers to the reduction in the number of spurious call graph edges achieved by applying various pruning heuristics. This metric is similar to what prior work used [36, 52].

6.2 Effectiveness

Monitoring Phase. *RVDebloater* reduces the attack surface by restricting each mode's access to only the *required* firmware functions at runtime. In the following, we evaluate the effectiveness of *RVDebloater*'s profiler in identifying the required firmware functions for each mode, and measure the resulting attack surface reduction achieved by *RVDebloater*.

***RVDebloater-dynamic* Profiler.** To evaluate the *RVDebloater* dynamic analysis output, we measured the number of missed functions across varying numbers of profiling and testing missions on different RV types, and determined how many profiling missions are needed to identify all the *required* functions for each mode without missing any. To do so, we used the RV missions discussed in §6.1 to dynamically determine

the *required* firmware functions for each mode and RV. Specifically, we first randomly chose two out of the 20 test missions as profiling missions for *dynamic analysis* for each RV. We then used the remaining 18 missions to test the identified *required* functions for each mode at runtime. We further evaluated the dynamic analysis output by gradually increasing the number of profiling missions, adding 2 missions each time to the profiling set from the testing set (e.g., 4 profiling and 16 testing missions), all the way to 20 missions.

The results are shown in Figure 4. As can be seen, all RVs converged to *zero missed functions* after profiling within only 10 missions (i.e., 10 profiling and 10 testing missions). Although the *RVDebloater* profiler identified all the required functions for each mode using 10 profiling missions, we continued the dynamic analysis evaluation by incrementally adding two new missions at a time to the profiling mission set and testing on the remaining missions (e.g., 12 profiling and 8 testing missions) to ensure that no missed functions appeared after convergence. The results confirmed that the number of missed functions for each RV remained at zero.

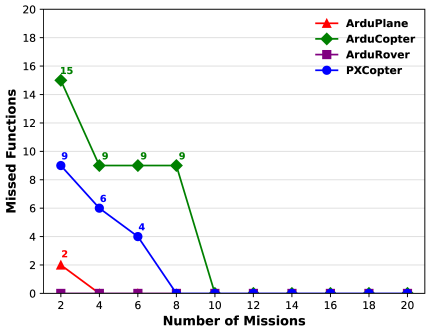


Figure 4: Number of missed functions across different missions for various platforms during dynamic analysis with *RVDebloater*. All platforms converge to 0 missed functions after 10 missions.

To further confirm this result, we evaluated the *RVDebloater* profiler on a totally different set of 20 different missions. The results, shown in Figure 4 (20 missions), demonstrate that *RVDebloater*'s dynamic profiler did not miss any required functions. Therefore, all *required* functions for each mode can be identified within 10 missions in all tested RVs.

Although the number of profiling missions used to identify all required functions for each mode at runtime is small, the *RVDebloater-dynamic* profiler may still miss some functions and produce false positives. To address this, *RVDebloater-static* can be used, which also includes the missing functions. However, the device maintainer may choose to rely on the dynamic analysis results (*RVDebloater-dynamic*) to achieve greater specialization compared to *RVDebloater-static*, which uses static analysis technique (§4.2 and §4.4).

Attack Surface Reduction. To evaluate the attack surface reduction, we used *RVDebloater*'s dynamic and static analysis to identify the required firmware functions for each mode. As we observed, after profiling 10 missions, the dynamic analysis converged to zero missing functions, and the number of functions remained unchanged with additional profiling. Hence, Tables 2 and 3 (*Dynamic Analysis* row) shows the number of firmware functions required at runtime for each mode, as identified by *RVDebloater*'s profiler with 10 missions. Furthermore, the *Static Analysis* row in Tables 2 and 3 presents the number of *reachable* functions for each mode, as determined by static analysis on the generated call graph. As discussed earlier (§4.2 and §4.4), *RVDebloater-static* and *RVDebloater-dynamic* use the outputs of static and dynamic analysis, respectively, to specialize the firmware code.

The results reveal that, on average, 85% and 41% of firmware functions are not required according to dynamic and static analysis, respectively. In particular, when *RVDebloater* uses static analysis outputs, it reduces the firmware functions by an average of 41% (Reduction *RVDebloater-static* in Tables 2 and 3), while achieving a 0% **FPR** and an average **FNR** of 44%, as this technique tends to overestimate the functions required for each mode (§4.2). Attempts to access debloated functions lead to *RVDebloater-static* activating `Fail-Safe` mode, and raising an alarm to the device maintainer (§4.4).

On the other hand, when the device maintainer uses *RVDebloater-dynamic*, which uses dynamic analysis—generated by profiling 10 missions—no missions fail, resulting in a 0% **FPR** and an 85% reduction in the attack surface (Reduction *RVDebloater-dynamic* in Tables 2 and 3). As in the case of *RVDebloater-static*, any attempt to access the debloated functions leads to *RVDebloater-dynamic* switching the RV to `Fail-Safe` mode, thereby restricting the attacker's access to only 15% of the firmware's functions with *RVDebloater-dynamic*. Furthermore, all functions allowed by *RVDebloater-dynamic* are actually required during execution (identified by profiling different missions), resulting in a 0% **FNR** for *RVDebloater-dynamic*.

From the results (Table 2 and 3), we can infer three main trends. First, each mode requires only a subset of the firmware functions rather than the entire firmware. For instance, when the RV is in `AUTO` mode, it requires approximately 2800 firmware functions, whereas in `RTL` mode, it requires a different set of around 3000 firmware functions (addressing L1, L2, and L4 in §3.1). Second, because RVs switch between different modes, including returning to earlier modes, they need an adaptive debloating technique that is reversible, in contrast to previous debloating techniques [36, 52] that use irreversible mechanisms to disable unwanted features (addressing L5 in §3.1). This is why we need a technique like *RVDebloater*. Third, *RVDebloater-dynamic* can debloat more functions, with only a few missions (fewer than 10), without introducing either *false negatives* or *false positives*. This is because embedded devices typically require a fixed set of functions (functionalities) for each mode, which can be

Table 2: Summary of firmware functions reduction (i.e., attack surface reduction through debloating) across different modes in ArduCopter, ArduPlane, and ArduRover (sub-platforms of ArduPilot). Reduction represents the percentage decrease in the number of firmware functions accessible to each mode when using *RVDebloater* with dynamic (*RVDebloater-dynamic*) or static analysis (*RVDebloater-static*), compared to the baseline firmware where all functions remain accessible.

Technique \ RV Type (modes)	ArduCopter						ArduPlane							ArduRover					
	AUTO	CIRCLE	STABILIZE	GUIDED	RTL	LOITER	MANUAL	AUTO	CIRCLE	GUIDED	QLOITER	RTL	QHOVER	AUTO	CIRCLE	MANUAL	RTL	GUIDED	LOITER
Dynamic Analysis	2869	3106	7054	3271	3153	2757	6891	2823	2756	2692	3120	2692	2700	2513	2435	6246	2433	2647	2255
Static Analysis	13520	13450	13447	13469	13470	13452	13235	13253	13232	13238	13252	13239	13238	12637	12558	12549	12566	12583	12550
w/o *RVDebloater*	22653	22653	22653	22653	22653	22653	22514	22514	22514	22514	22514	22514	22514	20990	20990	20990	20990	20990	20990
Reduction: ***RVDebloater-dynamic***	87.33%	86.28%	68.86%	85.56%	86.08%	87.83%	69.39%	87.46%	88.75%	88.04%	86.14%	88.04%	88%	88.02%	88.40%	70.24%	88.40%	87.39%	89.26%
RVDebloater-static	40.31%	40.62%	40.64%	40.54%	40.53%	40.61%	41.21%	41.13%	41.22%	41.20%	41.14%	41.19%	41.20%	39.80%	40.17%	40.21%	40.13%	40.05%	40.20%

Table 3: Summary of firmware functions reduction in PX-Copter across different modes (continuation of Table 2).

Technique	TAKEOFF	MISSION	LOITER	RTL
Dynamic Analysis	4320	2765	2660	2853
Static Analysis	12244	12238	12208	12227
w/o *RVDebloater*	20805	20805	20805	20805
Reduction: ***RVDebloater-dynamic***	79.23%	86.71%	87.21%	86.29%
RVDebloater-static	41.15%	41.18%	41.32%	41.23%

identified with just a few missions.

As discussed in §4.2, dynamic analysis may underestimate the required firmware functions — this is a known limitation of dynamic analysis techniques. Since the exact required functions per mode cannot be determined a priori, code coverage metrics cannot be used here, as they quantify execution depth rather than whether all functions required by each mode were captured. Nevertheless, for cases where rare or unanticipated conditions cause a function to be missed, *RVDebloater-dynamic* will trigger a transition to `Fail-Safe` mode, returning the device to its launch location or taking other actions based on the device maintainers' preferences (§4.1). Notably, in our evaluation across diverse missions (§6.1), *RVDebloater-dynamic* did not miss any required functions after converging to zero, and thereby never triggered `Fail-Safe` mode.

In contrast, *RVDebloater-static* eliminates fewer unneeded firmware functions for each mode, while employing a more careful and conservative debloating approach. Thus, the choice between these two approaches involves a trade-off: *static analysis* provides more cautious restrictions with lower protection, whereas *dynamic analysis* offers higher protection but has potential false positives.

Attack Evaluation. According to the list of most dangerous software weaknesses from CWE [1], out-of-bounds write is one of the most prevalent vulnerabilities. Thus, to evaluate *RVDebloater*'s effectiveness (*RVDebloater-dynamic*, *RVDebloater-static*) in restricting firmware access, we injected a buffer-overflow vulnerability—inspired by real-world vulnerabilities [13, 26]—into a function that processes user input, following prior work [45, 46]. An attacker who exploits this vulnerability can hijack control flow and perform a code-reuse attack. In our evaluation, we exploited this vulnerability to hijack the control flow and then performed three attacks. Each attack executed sensitive firmware functions, such as `disarm_motors` (§3.1), while the device was not in the correct mode. We discuss the three attacks in detail below.

(A1) Disarming RVs. ArduCopter provides a safety-critical function, `disarm_motors` (Listing 2), which disarms the motors during benign tasks such as flipping the device upright in `Turtle` mode (§3.1). However, after hijacking the control flow, our attack reused the `disarm_motors` function during a mission (e.g., while the device is in `GUIDED` mode), causing the device to crash. The attack succeeded because the firmware allowed unrestricted access to any function, including `disarm_motors`, regardless of the current mode.

When *RVDebloater* was deployed, it detected the attack and switched the device mode to `Fail-Safe`, allowing the device to land safely or return to the launch point (§4.1), because `disarm_motors` is restricted and is not accessible from other modes, such as `GUIDED`. The *disarming* functionality also exists in other platforms, such as ArduPlane and ArduRover, and is used during specific events like parachute release or crash detection. An attacker can perform the same attack to disarm the motors and cause physical damage. Again, *RVDebloater* detected such attacks because disarming should not be allowed in modes such as `QHOVER`.

(A2) Stopping Motors. ArduCopter and ArduPlane provide a function called `output_min`, which stops all motors and cuts engine power when specific events occur, such as when the device has landed. Similar to the previous attack, our stopping motors attack hijacks the control flow of the victim device (firmware) and calls `output_min` to stop the motors at an incorrect time (e.g., during mission while the device is in `GUIDED`), causing the device to crash. In our experiments, we confirmed that an attacker can easily disrupt a mission and damage the device by invoking this function. This attack was also detected and prevented by *RVDebloater* by switching the RV to the `Fail-Safe` mode. This is because `output_min` is not reachable from modes such as `GUIDED` or `QHOVER`.

(A3) Disarming Aion Rover. This attack case demonstrates the effectiveness of *RVDebloater* on a real-world device, the Aion Rover, discussed in §6.1. Similar to the previous attack cases, a buffer overflow vulnerability is intentionally inserted into the function that processes user input, inspired by previous work [46]. During the mission, this vulnerability is triggered, allowing the attacker to hijack the control flow and invoke a safety-critical function, `disarm`, from an incorrect mode (i.e., `MANUAL`). As a result, the motors are disarmed, and the rover stops abruptly, which may cause collisions with other nearby rovers or other objects.



(a) Boot Firmware (b) Runtime

Figure 5: Real-world attack (A3) on Aion Rover.

Again, when *RVDebloater* was deployed, it successfully detected this attack because the `disarm` function is not included in the list of required functions for modes such as `MANUAL`. As before, it switched the rover to the `Fail-Safe` mode upon detecting the attack. Figure 5 shows the setup for this attack on the Aion Rover in our lab.

Analysis Phase. As discussed in §4.2.2 and §B.1, *RVDebloater* uses a modified version of SVF along with two pruning heuristics—signature-based and address-based—to generate and refine the firmware call graph, identifying the *reachable* (*required*) functions for each mode (*RVDebloater-static*). To assess the effectiveness of these heuristics in reducing spurious edges in the call graph, we evaluated them on ArduCopter, ArduPlane, ArduRover, and PXCopter. As shown in Table 4, the number of edges in the original call graph generated by SVF ranges from 104K to 193K. By applying our pruning heuristics, we reduced the number of spurious edges in the call graph, achieving a precision of approximately 36% (66.8K), 45% (67.1K), 48% (60.1K), and 49% (98.5K) for ArduCopter, ArduPlane, ArduRover and PXCopter, respectively. Thus, our enhancement enables *RVDebloater-static* to more effectively restrict the functions accessible in each mode, ensuring that it only identifies the functions actually required by each mode.

Table 4: Precision after applying different heuristics. The third column shows the number of edges after applying the signature-based heuristic, and the fourth column shows the number after both heuristics.

RV (Firmware) Type	SVF	+ Signature-based	+ Addressed-based	Precision
ArduCopter	104K	87.4K	66.8K	36%
ArduRover	115.3K	80.3K	60.1K	48%
ArduPlane	122K	86.8K	67.1K	45%
PXCopter	193K	130.2K	98.5K	49%

The complexity of the input (i.e., firmware) affects the duration of the above process. On our platform, generating the call graph for each RV type (firmware) took approximately 3 to 4 hours on average, based on 10 executions of the SVF. However, as mentioned earlier, this step is performed offline, and is done only once per firmware.

6.3 Efficiency

Performance and Power Overhead. As discussed in §4.1, the only component of *RVDebloater* that incurs overhead on the RV is the monitoring component, which monitors the mode-switching function and checks indirect control flow transfer instructions to specialize the firmware code for each mode. Therefore, since both *RVDebloater-static* and *RVDebloater-dynamic* use this component, their results are similar. To evaluate the performance overhead, we used the real RVs mentioned in §5, i.e., the Aion rover and Pixhawk drone, as performance overheads in simulated RVs are dependent on the computing platform used for simulation. We used the scheduler in the RV autopilot platforms (ArduPilot and PX4) to track total CPU time and measured *RVDebloater*'s overhead by analyzing the additional CPU time incurred when *RVDebloater* was deployed.

We found that *RVDebloater* incurs modest performance overheads of 3.5% on the Aion rover and 4.3% on the Pixhawk drone, with an average of 3.9%. We also found that there was *no increase* in the overall mission time of the RVs with *RVDebloater* deployed for the missions discussed in §6.1.

Furthermore, we estimated the power consumption of *RVDebloater* running on those RVs, which are powered by a 5000 mAH battery. Since the processor typically accounts for 12% of the total power consumption [20] and *RVDebloater* adds an average overhead of 3.9%, we estimate that the RV's total power and energy consumption increases by approximately 0.47% due to *RVDebloater*, which is negligible.

Memory Overhead. To measure *RVDebloater*'s memory overhead, only the configuration files generated during the *analysis* phase and the instrumented firmware produced during the *guard insertion* phase are stored on the device. Thus, we focus on the overhead introduced by these two phases, as they are the only ones that incur overhead to memory usage on the device. As discussed in §B.2 and §B.3, *RVDebloater* builds the firmware using the LLVM compiler toolchain (i.e., gclang, gclang++, and clang). This process involves extracting the bitcode from the original firmware file, and then generating a new executable file from the instrumented bitcode for deployment on the device.

Table 5: Firmware size with and without using *RVDebloater* guard insertion phase.

Technique \ RV Type	ArduCopter	ArduPlane	ArduRover	PXCopter
w/o *RVDebloater*	6.2 MB	6.1 MB	5.7 MB	6.2 MB
RVDebloater	6.5 MB	6.4 MB	5.9 MB	6.4 MB
Overhead	**0.3 MB (4.8%)**	**0.3 MB (4.9%)**	**0.2 MB (3.5%)**	**0.2 MB (3.2%)**

We find that the instrumented firmware introduces an average memory overhead of 0.25 MB across different platform types, corresponding to approximately a 4% increase over the original firmware. A summary of firmware sizes is presented in Table 5. Further, the mode-based configuration files generated by *RVDebloater-static* and *RVDebloater-*

dynamic (§4.2) incur an average memory overhead of 0.65 MB and 0.17 MB, respectively. These files are not included in the firmware and thus do not contribute to the 4% size increase. Overall, the total memory overhead (i.e., instrumented firmware+configuration files) is negligible ($< 4\%$), as the memory available on the real RVs is over 20 MB.

Note that, like other debloating (specialization) techniques [31,36,45] that retain the firmware code while restricting access, *RVDebloater* also does not reduce the memory usage. As discussed earlier, mode-based embedded devices require the entire firmware, but only the functions relevant to the current mode should be accessible in it.

7 Related Work

Fixed debloating techniques. fall into two types based on whether they *retain* or *remove* unneeded features, such as security-critical system calls or hardware features.

1. Retaining Features. Abubakar et al. [31] proposed SHARD, which leverages context-aware hardening to specialize the Linux kernel at the application and system call levels. SHARD uses both static and dynamic analysis to identify the required kernel code for executing the system call invoked by an application. Afterwards, at runtime, SHARD ensures that only the identified kernel code is allowed to execute when the application calls the same system call. However, SHARD switches all kernel code pages for each system call invocation, making it unsuitable for resource-constrained embedded devices like RVs.

Minion [45] proposed a thread-level memory isolation technique for real-time microcontroller systems (MCS) that isolates the memory space for each process through offline static analysis of firmware and runtime memory access control using the Memory Protection Unit (MPU). However, Minion remains vulnerable to control flow hijacking attacks, as attackers can exploit memory corruption vulnerabilities in the shared memory between processes to hijack control flow and construct ROP or JOP chains. Further, since Minion uses the MPU to enforce memory isolation, it applies clustering techniques to group functions. However, this may allow processes to access functions that they do not require (§3.1).

Gu et al. proposed FACE-CHANGE [37], a kernel debloating technique that employs dynamic analysis to generate kernel profiles for each application, identifying the required kernel code at the application level. However, FACE-CHANGE's reliance on runtime profiling leads to incomplete identification of required kernel code (i.e., customized kernel view), limiting its analysis to the subset of the program executed during profiling. Furthermore, FACE-CHANGE suffers from coarse specialization granularity (i.e., single kernel view for the entire application), and hence attackers can still hijack the control flow of the sensitive programs (e.g., flight control program in RVs) and manipulate the devices' behavior.

2. Removing Features. Hu et al. proposed IRQDebloat [41], a debloating technique that leverages dynamic analysis to identify interrupt handlers and disable undesired hardware functionalities by rewriting their handlers with a no-op function in the target firmware. Hu et al. proposed Hacksaw [42], a kernel specialization technique that uses a list of hardware components attached to the target machine, called the device inventory, to remove kernel device drivers related to unnecessary hardware components.

Wu et al. proposed LightBlue [58], a profile-aware debloating technique that uses static analysis to identify the Bluetooth profiles (e.g., Advanced Audio Distribution Profile (A2DP)) utilized by an application (e.g., an Android app). Then, it removes unused code from the Bluetooth stack, including Bluetooth host code and firmware, thereby reducing the attack surface. However, LightBlue is limited to debloating of Bluetooth stacks and does not consider other features, such as firmware functionalities (e.g., `disarm_motors` in RVs).

Jahanshahi et al. proposed Minimalist [44], a semi-automated debloating technique for PHP web applications (e.g., WordPress and PhpMyAdmin) that uses static analysis to generate a call graph for these applications. Minimalist then removes unreachable and unused code based on the generated call graph and recorded access-log files, which include users' interactions with the web applications. However, Minimalist has two limitations as follows. First, users need to interact with the web server for each application to generate the access-log files, which is both time-consuming and potentially incomplete. Second, since Minimalist relies on user requests, it may remove essential parts of the code that are not captured in the log files, potentially causing system crashes.

All of these debloating techniques, which *remove unwanted features* such as hardware functionalities or kernel code, are irreversible, meaning that once unneeded features are removed, they cannot be restored. However, embedded devices may require different functionalities during various phases of execution. Therefore, these techniques are only suitable when certain features are guaranteed to remain unused during runtime; hence, our work is orthogonal to this line of work.

Adaptive Debloating Techniques. As mentioned in §1, *fixed debloating techniques* do not consider changes in application requirements at runtime, and as a result, they cannot disable critical features, such as the `fork` and `execve` system calls, which are necessary during the initial stages of an application. To address this issue, Ghavamnia et al. [36] and Rajagopalan et al. [52] proposed adaptive system call specialization techniques for server applications, which are discussed in detail,in Sections §1 and §3.1.

Unlike existing adaptive debloating techniques that rely on irreversible mechanisms to disable system calls, *RVDebloater* dynamically specializes firmware code at runtime for each mode. This is particularly important for mode-based embedded devices, such as RVs, where firmware functions may need to be re-enabled depending on the active mode (§6.2). Fur-

ther, while prior techniques focus on system call disabling, *RVDebloater* performs function-level specialization. This is important for embedded systems such as RVs.

Other Techniques. In addition to the two main categories of debloating techniques discussed above, several complementary security techniques have been proposed. (i) Compartmentalization techniques [35] partition a program into isolated compartments based on a specialization policy, such as filename. However, due to hardware constraints, MPUs typically provide limited number of configurable regions (e.g., four in ACES [35]), and as a result, compartments that should be isolated must be merged together, allowing them to access each other's code and data unintentionally [35]. (ii) Hardware-based isolation techniques [45, 59], some of which are discussed in fixed debloating techniques, require specific hardware features or hardware modifications for reducing the attack surface for control-flow hijacking. However, both (i) and (ii) rely on hardware support or modification that may not be available on resource-constrained mode-based embedded devices such as RVs (§2.2, §3.1). In contrast to these, *RVDebloater* uses a software-only enforcement mechanism to specialize the firmware based on *requirements of each operational mode*, without relying on hardware support. (iii) CFI techniques [30] monitor indirect control flow transfers according to a statically determined control flow graph, but do not consider the operational mode requirements of these devices. CFI is orthogonal to *RVDebloater*, and combining both techniques can provide stronger security guarantees, where *RVDebloater* reduces the required function set per mode while CFI further constrains transfers within that reduced set. (iv) Hotpatching techniques such as AutoPatch [54] automatically generate and apply patches to real-time embedded devices at runtime, without requiring a reboot. Since *RVDebloater* reduces the attack surface rather than patching known vulnerabilities, these two lines of work are orthogonal and can complement each other.

8 Conclusion and Future Work

We propose *RVDebloater*, the first adaptive debloating framework for *mode-based* embedded devices such as robotic vehicles (RVs). *RVDebloater* addresses limitations of existing debloating techniques by using static or dynamic analysis to *automatically* identify the *required* firmware functions for each mode and reduce the attack surface by restricting access to unneeded functions at runtime. Furthermore, *RVDebloater* uses a software-only technique to track mode-switches and specialize the firmware code at runtime, without relying on an MPU. *RVDebloater* is implemented using the LLVM compiler, making it portable across different embedded platforms.

We evaluated *RVDebloater* using real missions with various scenarios on six different RVs running either ArduPilot or PX4, two of the most widely used RV firmware. The results reveal that *RVDebloater* can reduce the RVs' attack surface by an average of 41% with static analysis and 85% with dynamic analysis. Furthermore, *RVDebloater* incurred a performance overhead of 3.9%, and negligible power and memory overheads on those RVs ($< 4\%$). Finally, *RVDebloater* successfully mitigated three attacks on the RVs.

Although *RVDebloater* has addressed limitations in existing debloating techniques, three possible directions remain for future work. First, although this paper focuses on RVs for evaluation, we believe *RVDebloater* can be applied to other types of *mode-based embedded devices*. Evaluating *RVDebloater* on such devices is a direction for future work.

Second, we required the mode-switching function(s) and list of modes to be provided to *RVDebloater* by device maintainers (§3.3). These functions can be easily identified from the firmware documentation or source code, as device maintainers are generally familiar with the supported modes since they are required for controlling RVs. However, automated techniques such as dynamic analysis can be used to monitor the device and identify which function is responsible for changing the mode at runtime. Further, our instrumentation component (§4.2.1) can assist such analysis by enabling the monitoring of firmware functions.

Third, although we evaluated *RVDebloater* using static or dynamic analysis separately, these techniques can be combined as a hybrid approach. *RVDebloater-hybrid* can first check using the dynamic output; if the target is not found, it can then check the static output. Since static analysis tends to overestimate, control flow integrity (CFI) techniques can then be applied to ensure that all control flow transfers adhere to the call graph.

Acknowledgments

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Department of National Defense (DND), Canada, and a Four Year Fellowship from UBC. We also thank the anonymous reviewers of USENIX VehicleSec 2026.

References

[1] 2024 cwe top 25 most dangerous software weaknesses. https://cwe.mitre.org/top25/archive/2024/2024_cwe_top25.html, Last Accessed, May 2026.

[2] Aion robotics. https://www.aionrobotics.com/, Last Accessed, May 2026.

[3] Apache nuttx. https://nuttx.apache.org/, Last Accessed, May 2026.

[4] Arducopter flight modes. https://ardupilot.org/copter/docs/flight-modes.html, Last Accessed, May 2026.

[5] Ardupilot copter. https://ardupilot.org/copter/index.html, Last Accessed, May 2026.

[6] Ardupilot fail-safe. https://ardupilot.org/copter/docs/failsafe-landing-page.html, Last Accessed, May 2026.

[7] Ardupilot plane. https://ardupilot.org/plane/index.html, Last Accessed, May 2026.

[8] Ardupilot rover. https://ardupilot.org/rover/index.html, Last Accessed, May 2026.

[9] Ardupilot software in the loop. https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html, Last Accessed, May 2026.

[10] Arduplane flight modes. https://ardupilot.org/plane/docs/flight-modes.html, Last Accessed, May 2026.

[11] Bot-in-time delivery. https://www.forbes.com/forbes/2009/0316/040_bot_time_saves_nine.html, Last Accessed, May 2026.

[12] Clang: a c language family frontend for llvm. https://clang.llvm.org/index.html, Last Accessed, May 2026.

[13] Cve-2022-28711. https://nvd.nist.gov/vuln/detail/CVE-2022-28711, Last Accessed, May 2026.

[14] Dji agras t50. https://ag.dji.com, Last Accessed, May 2026.

[15] Gazebo robot simulation. https://gazebosim.org/home, Last Accessed, May 2026.

[16] Gllvm. https://github.com/SRI-CSL/gllvm, Last Accessed, May 2026.

[17] Hijacking drones with a mavlink exploit. https://diydrones.com/profiles/blogs/hijacking-quadcopters-with-a-mavlink-exploit, Last Accessed, May 2026.

[18] The llvm compiler. https://llvm.org/, Last Accessed, May 2026.

[19] Pixhawk. https://ardupilot.org/copter/docs/common-pixhawk-overview.html, Last Accessed, May 2026.

[20] Pixhawk power consumption. https://diydrones.com/profiles/blogs/pixhawk-and-apm-power-consumption, Last Accessed, May 2026.

[21] Px4 airframes. https://docs.px4.io/v1.12/en/airframes/airframe_reference.html, Last Accessed, May 2026.

[22] Px4 fail-safe configuration. https://docs.px4.io/main/en/config/safety.html, Last Accessed, May 2026.

[23] Px4 open source autopilot. https://docs.px4.io/main/en/, Last Accessed, May 2026.

[24] Qgroundcontrol. https://qgroundcontrol.com/, Last Accessed, May 2026.

[25] Seccomp bpf (secure computing with filters). https://www.kernel.org/doc/html/v4.16/userspace-api/seccomp_filter.html, Last Accessed, May 2026.

[26] Stack buffer overflow. https://github.com/PX4/PX4-Autopilot/issues/5643, Last Accessed, May 2026.

[27] Svf. https://github.com/SVF-tools/SVF, Last Accessed, May 2026.

[28] Ups will use drones to deliver medical supplies in north carolina. https://www.theverge.com/2019/3/26/18282291/ups-drone-delivery-hospital-nc-matternet, Last Accessed, May 2026.

[29] Writing an llvm pass. https://llvm.org/docs/WritingAnLLVMPass.html, Last Accessed, May 2026.

[30] Martín Abadi, Mihai Budiu, Ulfar Erlingsson, and Jay Ligatti. Control-flow integrity principles, implementations, and applications. *ACM Transactions on Information and System Security (TISSEC)*, 13(1):1–40, 2009.

[31] Muhammad Abubakar, Adil Ahmad, Pedro Fonseca, and Dongyan Xu. {SHARD}:{Fine-Grained} kernel specialization with {Context-Aware} hardening. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2435–2452, 2021.

[32] Lars Ole Andersen. Program analysis and specialization for the c programming language. 1994.

[33] Babak Amin Azad, Pierre Laperdrix, and Nick Nikiforakis. Less is more: quantifying the security benefits of debloating web applications. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1697–1714, 2019.

[34] Hongjun Choi, Wen-Chuan Lee, Yousra Aafer, Fan Fei, Zhan Tu, Xiangyu Zhang, Dongyan Xu, and Xinyan Deng. Detecting attacks against robotic vehicles: A control invariant approach. In *Proceedings of the 2018 ACM*

SIGSAC Conference on Computer and Communications Security, pages 801–816, 2018.

[35] Abraham A Clements, Naif Saleh Almakhdhub, Saurabh Bagchi, and Mathias Payer. {ACES}: Automatic compartments for embedded systems. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 65–82, 2018.

[36] Seyedhamed Ghavamnia, Tapti Palit, Shachee Mishra, and Michalis Polychronakis. Temporal system call specialization for attack surface reduction. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1749–1766, 2020.

[37] Zhongshu Gu, Brendan Saltaformaggio, Xiangyu Zhang, and Dongyan Xu. Face-change: Application-driven dynamic kernel view switching in a virtual machine. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 491–502. IEEE, 2014.

[38] Kihong Heo, Woosuk Lee, Pardis Pashakhanloo, and Mayur Naik. Effective program debloating via reinforcement learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 380–394, 2018.

[39] Michael Hind. Pointer analysis: Haven't we solved this problem yet? In *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 54–61, 2001.

[40] Michael Hooper, Yifan Tian, Runxuan Zhou, Bin Cao, Adrian P Lauf, Lanier Watkins, William H Robinson, and Wlajimir Alexis. Securing commercial wifi-based uavs from common security attacks. In *MILCOM 2016-2016 IEEE Military Communications Conference*, pages 1213–1218. IEEE, 2016.

[41] Zhenghao Hu and Brendan Dolan-Gavitt. Irqdebloat: Reducing driver attack surface in embedded devices. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1608–1622. IEEE, 2022.

[42] Zhenghao Hu, Sangho Lee, and Marcus Peinado. Hacksaw: Hardware-centric kernel debloating via device inventory and dependency analysis. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1994–2008, 2023.

[43] Yongzhe Huang, Vikram Narayanan, David Detweiler, Kaiming Huang, Gang Tan, Trent Jaeger, and Anton Burtsev. {KSplit}: Automating device driver isolation. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 613–631, 2022.

[44] Rasoul Jahanshahi, Babak Amin Azad, Nick Nikiforakis, and Manuel Egele. Minimalist: Semi-automated debloating of {PHP} web applications through static analysis. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5557–5573, 2023.

[45] Chung Hwan Kim, Taegyu Kim, Hongjun Choi, Zhongshu Gu, Byoungyoung Lee, Xiangyu Zhang, and Dongyan Xu. Securing real-time microcontroller systems through customized memory view switching. In *NDSS*, 2018.

[46] Ao Li, Jinwen Wang, and Ning Zhang. Software availability protection in cyber-physical systems. In *34nd USENIX Security Symposium (USENIX Security 25)*, 2025.

[47] Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. Pixhawk: A system for autonomous flight using onboard computer vision. In *2011 ieee international conference on robotics and automation*, pages 2992–2997. IEEE, 2011.

[48] Chenxiong Qian, Hong Hu, Mansour Alharthi, Pak Ho Chung, Taesoo Kim, and Wenke Lee. {RAZOR}: A framework for post-deployment software debloating. In *28th USENIX security symposium (USENIX Security 19)*, pages 1733–1750, 2019.

[49] Anh Quach, Aravind Prakash, and Lok Yan. Debloating software through {Piece-Wise} compilation and loading. In *27th USENIX security symposium (USENIX Security 18)*, pages 869–886, 2018.

[50] Davide Quarta, Marcello Pogliani, Mario Polino, Federico Maggi, Andrea Maria Zanchettin, and Stefano Zanero. An experimental security analysis of an industrial robot controller. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 268–286. IEEE, 2017.

[51] Raul Quinonez, Jairo Giraldo, Luis Salazar, Erick Bauman, Alvaro Cardenas, and Zhiqiang Lin. {SAVIOR}: Securing autonomous vehicles with robust physical invariants. In *29th USENIX security symposium (USENIX Security 20)*, pages 895–912, 2020.

[52] Vidya Lakshmi Rajagopalan, Konstantinos Kleftogiorgos, Enes Göktas, Jun Xu, and Georgios Portokalidis. Syspart: Automated temporal system call filtering for binaries. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1979–1993, 2023.

[53] Majid Salehi, Danny Hughes, and Bruno Crispo. {μSBS}: Static binary sanitization of bare-metal embedded devices for fault observability. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 381–395, 2020.

[54] Mohsen Salehi and Karthik Pattabiraman. Autopatch: Automated generation of hotpatches for real-time embedded devices. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 2370–2384, 2024.

[55] Mohsen Salehi and Karthik Pattabiraman. RVDebloater: Mode-based adaptive firmware debloating for robotic vehicles, 2026. arXiv preprint arXiv:2602.00270.

[56] Yulei Sui and Jingling Xue. Svf: interprocedural static value-flow analysis in llvm. In *Proceedings of the 25th international conference on compiler construction*, pages 265–266, 2016.

[57] Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. Sok: Eternal war in memory. In *2013 IEEE Symposium on Security and Privacy*, pages 48–62. IEEE, 2013.

[58] Jianliang Wu, Ruoyu Wu, Daniele Antonioli, Mathias Payer, Nils Ole Tippenhauer, Dongyan Xu, Dave Jing Tian, and Antonio Bianchi. {LIGHTBLUE}: Automatic {Profile-Aware} debloating of bluetooth stacks. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 339–356, 2021.

[59] Jie Zhou, Yufei Du, Zhuojia Shen, Lele Ma, John Criswell, and Robert J Walls. Silhouette: Efficient protected shadow stacks for embedded systems. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1219–1236, 2020.

A Pruning Heuristics

(1) Signature-based Pruning Heuristic. SVF resolves indirect call sites using a basic pruning heuristic that considers only the number of arguments. However, this approach does not effectively prune spurious targets, as it considers functions with the correct number of arguments but incorrect argument types as valid targets.

To improve precision, we additionally use two filtering methods. First, *RVDebloater* considers the argument types when resolving indirect call targets, discarding those that do not match. In particular, *RVDebloater* analyzes all indirect call sites and prunes call edges to functions whose argument types do not match the expected types at the call site. Second, *RVDebloater* enhances call graph accuracy by also considering return types. Specifically, functions with a `void` return type are identified as spurious targets for indirect call sites that expect a return value. It effectively prunes spurious targets and reduces the number of call edges in the generated call graph.

(2) Address-based Pruning Heuristic. To further prune spurious targets, *RVDebloater* applies an additional heuristic (address-based pruning heuristic) based on “taken" addresses in the reachable path to the entry point. Andersen’s algorithm assumes that *all functions in the program* are reachable from the entry point. However, a function may not have its address stored in a variable or may be unreachable through the entry point, making it an invalid target for an indirect call site.

Thus, *RVDebloater* identifies all functions whose addresses appear in paths reachable from the entry point. Then, it examines all indirect call sites and removes edges that point to functions that are not in this list—i.e., functions whose addresses are not in any reachable path from the entry point.

After refining the call graph, *RVDebloater* identifies the entry functions of each mode to determine their reachable (required) functions. To do so, it employs an instrumentation component similar to the one used in dynamic analysis (§4.2.1), which instruments both the firmware functions and the mode-switching function(s). *RVDebloater* then executes the device through a mission that includes all modes intended for monitoring and control at runtime by *RVDebloater-static*. During execution, when the device switches modes, *RVDebloater* tracks the function(s) invoked by the mode-switching function to initialize the parameters required for the new mode and marks them as the entry function(s) of that mode.

RVDebloater also performs mode-name pattern matching to confirm the validity of the identified entry function(s). In particular, we observed that these firmware models (e.g., ArduCopter or ArduPlane) have entry function(s) responsible for managing mode-specific parameters, and their names typically include the mode name itself. Therefore, *RVDebloater* uses the list of mode names provided as input (§3.3) to verify and confirm the correctness of these functions. We found that all identified entry functions were correct. Note that this pattern can be adapted by developers to support other firmware. However, we used the same pattern in all the firmware models evaluated in this paper, as it was sufficient. Finally, *RVDebloater* uses the generated call graph to identify the reachable and unreachable functions for each mode, starting from its entry function(s).

B Implementation Details

A summary of the *RVDebloater* implementation across its five main components, corresponding to the different phases (§5, §B.1, §B.2, and §B.3) is presented in Table 6.

Table 6: Lines of code in *RVDebloater* components.

Component	Lines of Code
Pruning Heuristics	220 C++
Instrumentation	1000 C++ (two LLVM passes)
***RVDebloater* Profiler**	500 C
Guard Insertion	1100 C++ (two LLVM passes)
***RVDebloater* Monitor**	400 C
Total	3220

B.1 Analysis Phase

As mentioned earlier, the instrumentation component of the analysis phase takes the firmware source code as input. In the first step, the firmware is compiled to LLVM bitcode (*bc* file) using the *gclang* and *gclang++* compilers [16]. Then, if *RVDebloater-static* is selected, it uses the modified version of SVF with two pruning heuristics—signature-based and address-based—to generate a pruned call graph. We modified the TSCP implementation [36] by adding new heuristics to refine the call graph. A Python script then analyzes this call graph to generate lists of reachable functions for each mode through static analysis.

If *RVDebloater-dynamic* is selected, the instrumentation component links the firmware bitcode with *RVDebloater*'s profiler functions (i.e., *mode_entry* and *log_fn*). This component then takes the mode-switching function as input and iterates through the firmware's instructions. Specifically, it inserts *mode_entry* at locations within the mode-switching function where the return value is `true`, indicating a successful mode change. Also, it inserts *log_fn* at the entry points of other firmware functions.

```
define dso_local zeroext i1 @AP_Arming_Plane(%
    class.AP_Arming_Plane*, i32, i1 zeroext) #0
    align 2 !dbg !97833 {
// Allocation Variables
call void @log_fn(arguments)
// ... omit
}
```

Listing 4: Inserting *log_fn* at the entry of functions.

In the example shown in Listing 4, the instrumentation component inserts *log_fn* at the beginning of the function, after the allocation instructions. Finally, *RVDebloater* automatically executes the instrumented firmware with different missions in a benign environment and tracks its execution using profiler functions to identify the required functions for each mode at runtime.

B.2 Guard Insertion Phase

In the next phase, a different instrumentation component takes the firmware and mode-switching function(s) as input. Once the firmware is compiled to LLVM bitcode using *gclang* and *gclang++* compilers and linked with *RVDebloater*'s monitor functions (i.e., *mode_entry_runtime* and *monitor_fn*), *RVDebloater* first identifies the mode-switching function(s). It then inserts *mode_entry_runtime*, which monitors mode changes and loads the required functions for the new mode from the configuration file (either static or dynamic output) at locations where the mode successfully changes. Next, the component iterates over the firmware functions and replaces indirect control flow transfer instructions (e.g., indirect call sites represented by *CallInst* in LLVM IR) and return instructions (*ReturnInst* in LLVM IR) with *monitor_fn*, which controls target execution, raises an alarm, and switches to `Fail-Safe` mode if needed.

```
1 define dso_local zeroext i1
    @Plane18set_mode_by_number(%class.Plane*, i8
    zeroext, i8 zeroext) #0 align 2 !dbg !566193 {
// ... omit
2 call void @mode_entry_runtime(arguments)
3 ret i1 %22, !dbg !566216
4 }
```

Listing 5: Inserting *mode_entry_runtime* before return instruction (Listing 3).

For instance, this component inserts *mode_entry_runtime* before the return instruction when its value is true, as shown in Listing 5. Finally, it stores the instrumented file.

B.3 Monitoring Phase

In the final phase, *RVDebloater* builds the instrumented firmware executable using the *Clang* compiler [12]. At runtime (during an RV mission), the *RVDebloater* monitor is triggered when a mode change occurs. The monitor component then loads the corresponding configuration file, which includes the required functions for the new mode identified by either static or dynamic analysis. Furthermore, the *RVDebloater* monitor is triggered whenever an indirect control flow transfer or a return instruction is executed. In such cases, the component checks permissions based on the loaded configuration for the current mode and logs the event, raising an alarm or switching to `Fail-Safe` mode if an unauthorized execution is detected.

USENIX VehicleSec '26 Artifact Appendix: RVDebloater: Mode-based Adaptive Firmware Debloating for Robotic Vehicles

A Artifact Appendix

A.1 Abstract

The RVDebloater artifacts provide a complete, reproducible evaluation framework for mode-based adaptive firmware debloating of robotic vehicles (RVs). The artifact includes a modified SVF pointer analysis framework, pre-generated call graphs for each RV platform (e.g., ArduCopter, ArduPlane, ArduRover, and PX4), a set of missions, profiling and monitoring tools and results, and all scripts necessary to run the RVDebloater static and dynamic analysis pipeline in a simulation environment (SITL). In addition, the artifact introduces the first approach to LLVM bitcode generation for RV firmware, enabling researchers to generate `.bc` files and apply custom LLVM passes and third-party analysis tools to real-world firmware such as ArduPilot and PX4.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Because our technique operates in a simulation environment, it poses no risk to evaluators. RVDebloater operates without requiring the disabling of any security mechanisms.

A.2.2 How to access

The artifact source code and all supporting data are hosted on GitHub and Zenodo: https://github.com/DependableSystemsLab/RVDebloater.git , https://doi.org/10.5281/zenodo.20513118

A.2.3 Hardware dependencies

The main contributions and results of RVDebloater—its effectiveness, including the identification of required functions via static and dynamic analysis and the reduction of the attack surface at runtime (Section 6.2 in paper)—can be evaluated without hardware access using a simulation environment (SITL). Performance overhead (efficiency) is assessed on real hardware platforms, specifically the Aion Rover R1 and the Pixhawk drone.

PC-based analysis and simulation (E1, E2). All artifact experiments (E1) and (E2) run entirely in simulation (SITL) and do not require any physical robotic vehicle hardware.

Minimum requirements for running the simulation-based experiments: Disk: 30 GB of free storage (after cloning, installing dependencies, and building firmware) OS: Ubuntu 22.04 LTS (64-bit). *Machine used to generate the paper's analysis results* (call graph generation, static reachability analysis, and instrumentation during analysis and guard-insertion phases): CPU: Intel Xeon E-2224 (4 cores, 3.4 GHz base), OS: Ubuntu 22.04.5 LTS (64-bit).

A.2.4 Software dependencies

Required OS: Ubuntu 22.04 LTS. The artifact has been developed and tested exclusively on Ubuntu 22.04 LTS. Other Linux distributions are not automatically supported and may fail during the build or simulation steps. Our artifact includes a prerequisite script at `scripts/pre-req.sh`, which installs and configures all required software dependencies.

A.2.5 Benchmarks

The artifact includes pre-defined mission files (in the `missions` directory) for each evaluated RV type (ArduCopter, ArduPlane, ArduRover, and PX4). These missions are used as workloads during simulation-based evaluation.

A.3 Set-up

For more details, please refer to the README in repository.

A.3.1 Installation

Step 1 — Create the project directory. Change `${HOME}/rvd-project` to a different path if needed, and update the `$RVD_PROJ` variable accordingly.

Step 2 — Check or remove an existing `go` installation. If `go` is already installed, verify the version: If the version is 1.17 or higher, skip to Step 3. Otherwise, remove the existing installation before proceeding.

Step 3 — Clone the repository in $RVD_PROJ.

```
git clone
    ↪ git@github.com:DependableSystemsLab/RVDebloater.git
```

A.3.2 Basic Test

The scripts are pre-configured to run RVDebloater in **Rover (profiling) mode**. To verify a successful installation, run `run-simulation.sh`. Once the simulation has loaded, you can control the RV (check README). A successful run will produce profiling output in the `profiles` directory and simulation logs confirming mission completion. The current setup uses an automatic script to execute the mission without further manual intervention. By default, run-simulation.sh (in RVDebloater/scripts/) runs the simulation automatically.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): RVDebloater effectively identifies required functions through combined static and dynamic analysis and reduces the firmware's attack surface at runtime. This is proven by experiments (E1) and (E2) described in Section 6.2, whose results are reported in the corresponding tables and figures.

(C2): RVDebloater supports multiple RV types and operates correctly in both profiling and monitoring modes. This is proven by experiments (E1) and (E2).

A.4.2 Experiments

(E1): *Effectiveness — Profiling Mode* [15 human-minutes + 1 compute-hour]: This experiment demonstrates that RVDebloater correctly identifies required functions using profiling-based analysis for each RV type.
Preparation: Edit `scripts/run-simulation.sh` to select the desired RV type.
Step 1 — Select RV type. Uncomment the appropriate mission line (keep only one uncommented).
Step 2 — Configure vehicle type. In the same file, uncomment the vehicle and frame configuration (keep only one uncommented). Also, uncomment one of the lines 168, 169, or 170 based on your vehicle type in `scripts/run-simulation.sh`.
Step 3 — Set analysis mode to profiling. Edit `scripts/dependent/instrument-ap.sh` and uncomment the profiling passes (ensure the monitoring passes are commented out). Also, based on your vehicle type, you need to uncomment the lines in `scripts/dependent/instrument-ap.sh`.
Sub-Step 3 — Set Vehicle Type. Edit `Instrumentation/AddTrack.cpp` and uncomment the vehicle type that is set in previous steps (ensure the other lines are commented out).
Also, uncomment the correspond lines based on your vehicle type in `Instrumentation/AddTrack.cpp` (*Keep exactly one of the lines above uncommented at a time*).
Copter: Lines 128-170, **Plane or Rover:** Lines 176-218.
Results: The experiment generates a file called `mode_profile` in the `profiles` directory. Run `FunctionsName_to_Mode.py` to produce per-mode files listing the required functions for each mode.
(E2): *Effectiveness — Monitoring Mode* [15 human-minutes + 1 compute-hour]: This experiment shows RVDebloater's runtime debloating in monitoring mode.
Preparation: Follow Steps 1 and 2 from (E1) to select the RV type. Then set the analysis mode to monitoring.
Step 3 — Set analysis mode to monitoring. Edit `scripts/dependent/instrument-ap.sh` and uncomment the monitoring passes.
Sub-Step 3 — Set Vehicle Type. Edit `Instrumentation/CallOverwrite.cpp` and uncomment the vehicle type that is set in previous steps (ensure the other lines are commented out)
Also, edit `Instrumentation/AddEnforce.cpp` and uncomment the vehicle type that is set in previous steps (ensure the other lines are commented out).
Copter: Lines 96-138, **Plane or Rover:** Lines 142-185.
Execution: Before running, you need to change few places in `run-simulation.sh` (check README).

A.5 Notes on Reusability

Beyond the contributions of RVDebloater discussed in the paper, we address a fundamental gap in the RV firmware ecosystem: the lack of an automated technique for generating bitcode files, which are essential for leveraging analysis frameworks such as LLVM. To overcome this limitation, we modified the infrastructure of **gllvm** and its underlying **Go** implementation, enabling the automated generation of LLVM bitcode (BC) during the firmware build process. Our artifact supports complex, real-world firmware systems such as ArduPilot and PX4. With this capability, developers can seamlessly apply LLVM-based analyses, including custom LLVM passes and third-party tools such as DataFlowSanitizer (DFSan). Moreover, because our changes extend the core infrastructure of `gllvm` and Go, the approach generalizes beyond ArduPilot and PX4 and can support a broader range of firmware.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.



CANBERT+: An Optimized Multi-Layer BERT-Based Intrusion Detection for CAN Bus

Jun Yeon Won
The Ohio State University
won.126@osu.edu

Qadeer Ahmed
The Ohio State University
ahmed.358@osu.edu

Zhiqiang Lin
The Ohio State University
lin.3021@osu.edu

Abstract

Controller Area Network (CAN) buses in modern vehicles lack fundamental security mechanisms, leaving them vulnerable to message-injection attacks. While Transformer-based Intrusion Detection Systems (IDS) are a promising solution, existing designs struggle to balance performance with efficiency, either discarding payload context or incurring high computational costs and vocabulary explosion when modeling full CAN frames. We present CANBERT+, an optimized BERT-based IDS designed to consider realistic deployment viability without sacrificing message context making inference 6 times faster. CANBERT+ represents each CAN message as a single token augmented with multi-layer embeddings for the identifier, data length, and payload bytes with the elimination of redundant layers. This approach differentiates payload nuances while maintaining short sequence lengths to allow the model to capture byte-level context. Following a two-stage training process, self-supervised pre-training on benign traffic and supervised fine-tuning on attacks, we evaluated CANBERT+ on eight public datasets covering spoofing, fuzzing, DoS, and replay attacks. The model consistently outperforms traditional machine learning baselines and prior BERT-based IDSs; notably, it improves the F1 score on replay attacks from 0.60 to 0.72, which is 20% improvement. Furthermore, we introduce two compact variants that reduce model size by up to 205 times while preserving high detection accuracy, making the approach suitable for resource-constrained automotive hardware.

1 Introduction

Modern vehicles have evolved into sophisticated cyber-physical systems composed of dozens of electronic control units (ECUs), a dense array of sensors, and multiple wireless and wired interfaces. Beyond basic drivability, these platforms deliver advanced driver-assistance features, comfort functions, and infotainment services that continuously monitor and respond to the driver, passengers, and surrounding environment. As a result, in-vehicle communication networks routinely carry rich behavioral and contextual information, including acceleration and braking patterns. When aggregated over time, such data can reveal sensitive attributes such as driving style and daily routines. Protecting both safety and privacy in this data-rich ecosystem has therefore become a central requirement for modern automotive systems.

At the heart of in-vehicle networking is the Controller Area Network (CAN) protocol, which enables low-latency, priority-based communication among ECUs. CAN's broadcast design and simple arbitration mechanism make it well suited for real-time coordination, but they also introduce fundamental security and privacy challenges. The protocol provides neither built-in authentication nor confidentiality: any node attached to the bus can transmit messages, and all nodes can observe all traffic. Consequently, an attacker who compromises an ECU or connects a rogue device can inject forged messages to manipulate vehicle behavior while simultaneously monitoring traffic to collect detailed records of in-vehicle activity. Such attacks threaten safety and reliability, and they also create serious privacy risks by enabling long-term tracking, profiling, and misuse of sensitive vehicular data.

To mitigate these risks, a wide range of intrusion detection systems (IDS) for in-vehicle networks has been proposed. Traditional approaches primarily rely on rule-based or statistical monitoring, such as detecting abnormal message frequencies [45] or abrupt changes in component states [55]. As vehicle functionality and attack surfaces have grown more complex, machine learning-based IDSs have gained popularity due to their ability to model subtle patterns in high-volume CAN traffic. Deep neural networks (DNNs) [34] and long short-term memory (LSTM) models [27] have demonstrated improved detection accuracy, and more recently, transformer-based models, originally developed for natural language processing (NLP), have been successfully adapted to in-vehicle IDS settings [4, 43]. Transformers are particularly attractive because they can capture contextual dependencies across message sequences more effectively than many recurrent architectures.

Among transformer models, Bidirectional Encoder Representations from Transformers (BERT) is a compelling backbone for CAN intrusion detection because it can learn context-rich representations and model long-range dependencies in sequential data. However, existing BERT-based IDS solutions [4, 43] still face two key limitations. First, they often apply BERT with minimal adaptation, treating CAN messages as generic strings rather than structured, field-based records—an approach that is not tailored to the semantics of in-vehicle communications. Second, their input representations and data handling strategies can introduce substantial computational overhead (*e.g.*, long token sequences or inefficient payload processing), which makes deployment on resource-constrained automotive platforms challenging.

To address these limitations, we propose CANBERT+, a computationally efficient and customized IDS for in-vehicle networks. Built on BERT, CANBERT+ introduces multi-layer input embeddings that integrate the CAN identifier, data length, and byte-wise payload information into a single token per message. This design preserves contextual information while keeping the sequence length compact, improving efficiency without sacrificing expressiveness. In addition, CANBERT+ explicitly distinguishes individual data bytes within a CAN message to better capture fine-grained payload semantics relevant to attack detection. Recognizing that different vehicles employ heterogeneous sets of CAN identifiers and payload formats, CANBERT+ follows a two-stage training strategy: (1) self-supervised pre-training on benign CAN traffic to learn structural and contextual properties of normal messages, and (2) supervised fine-tuning on labeled datasets containing attack traffic to enable accurate malicious message classification.

We validate CANBERT+ through an extensive experimental evaluation on eight open-source in-vehicle attack datasets comprising 258 data logs. Compared to both traditional machine learning approaches (*e.g.*, support vector machines and random forests) and prior BERT-based IDSs that either ignore payload context or process it inefficiently, CANBERT+ usually achieves superior detection performance. In replay attack scenarios, for example, it attains an F1 score of 0.72, substantially improving over the 0.60 F1 score of the previous BERT-based IDS. Moreover, CANBERT+ is 6 times faster than the previous BERT-based IDS. Last, we design two compact variants, S-CANBERT+ and T-CANBERT+, which significantly reduce model size and memory footprint while retaining high detection accuracy, making them better suited for deployment on resource-constrained automotive hardware.

Contributions. This work makes the following contributions:

- **Novel Model.** We develop CANBERT+, an optimized IDS leveraging BERT-based multilayer embeddings to enhance the understanding of CAN messages.
- **Empirical Evaluation.** We conduct an extensive evaluation across eight datasets (258 logs), covering diverse attack scenarios including spoofing, fuzzing, and replay attacks.
- **Better Performance.** We demonstrate that CANBERT+ usually achieves higher F1 scores than both traditional machine learning techniques and prior BERT-based IDS methods.

2 Background and Related Works

2.1 Controller Area Network

The CAN protocol is the de facto standard for real-time, low-latency communication among ECUs in modern automotive architectures. CAN operates as a multi-master broadcast bus, meaning any node can transmit data onto the shared network without a central coordinator.

To manage simultaneous transmissions, CAN employs a non-destructive bitwise arbitration mechanism based on the Message Identifier (CAN ID). When multiple nodes transmit at once, the message with the lowest numeric ID value, indicating the highest priority, wins arbitration. The winning node continues transmission seamlessly, while lower-priority nodes automatically back off and switch to a receiving state to prevent data collisions.

Structurally, a standard CAN frame consists of the arbitration field (CAN ID), a Data Length Code (DLC), and a data payload ranging from 0 to 8 bytes, alongside fields for cyclic redundancy checks (CRC) and acknowledgment. While ECUs use hardware acceptance filters to process only messages relevant to their function, the protocol itself lacks intrinsic security measures. CAN was designed without support for message authentication or payload encryption. Consequently, any adversary with access to the bus, whether physical or remote, can indiscriminately monitor traffic or inject malicious frames without restriction.

2.2 Types of CAN Attacks

The inherent lack of authentication and encryption in the CAN protocol exposes vehicles to various message injection attacks [3, 7, 9, 36]. These attacks exploit the broadcast nature of the bus to manipulate vehicle behavior. Common classes include:

- **Spoofing**: Injecting forged messages with legitimate IDs to impersonate ECUs (*e.g.*, altering instrument cluster outputs or actuator commands).
- **Fuzzing**: Sending malformed or random payloads to trigger software faults or unexpected ECU behavior.
- **Denial-of-Service (DoS)**: Flooding the bus with high-priority frames to block legitimate traffic and disrupt functions such as braking or steering.
- **Replay**: Recording legitimate messages and retransmitting them later to cause repeated or out-of-context actions.

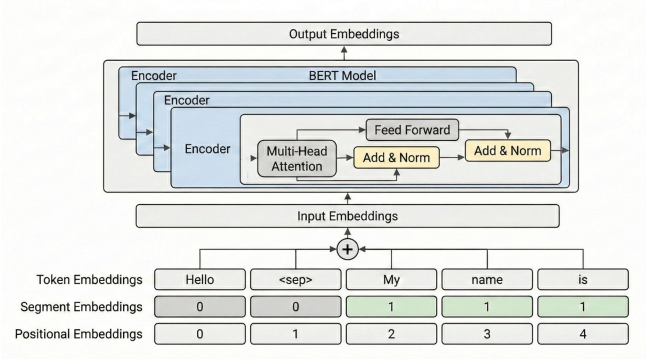


Figure 1: **High-level Architecture of the BERT Model.**

Sophisticated adversaries may also combine these techniques, for instance, launching a DoS attack to suppress legitimate sensor data while simultaneously Spoofing false readings, to evade simple rule-based intrusion detection systems.

2.3 BERT Model

BERT [17] is a deep learning framework originally developed for Natural Language Processing (NLP). Unlike traditional directional models (such as RNNs) or encoder-decoder Transformer architectures [48], BERT is composed exclusively of a stack of Transformer encoder layers. It is optimized to learn rich, contextual representations of input sequences through the following mechanisms:

Bidirectional Contextual Learning. Unlike LSTMs, which typically process sequences in a single direction (left-to-right), BERT utilizes bidirectional self-attention. This allows every token in a sequence to attend to both preceding and succeeding tokens simultaneously, capturing the long-range dependencies necessary for understanding complex contexts.

Architectural Components. As illustrated in Figure 1, each encoder layer in BERT consists of three primary modules:

- **Multi-head Attention**: This mechanism runs multiple self-attention operations in parallel, allowing the model to focus on different parts of the sequence simultaneously and model intricate relationships between tokens.
- **Feed-Forward Network (FFN)**: A fully connected network applied to each position independently and identically. It introduces non-linearity and enhances the model's representational power.
- **Add & Normalization Layers**: Each sub-layer (Attention and FFN) is followed by a residual connection (Add) and Layer Normalization. This structure mitigates vanishing gradients and ensures stable convergence during training.

Training Methodology. BERT is pre-trained using two self-supervised objectives that do not require labeled data:

- **Masked Language Modeling (MLM)**: A percentage of input tokens are randomly masked, and the model is trained to predict the original token based solely on its context. This enforces deep bidirectional learning.
- **Next Sentence Prediction (NSP)**: The model determines if two input segments naturally follow one another, helping the model capture inter-sentence coherence.

Embedding Mechanism. To process inputs, BERT represents sequences as the sum of three distinct embedding layers: 1) Token Embeddings to encode individual words or subwords; 2) Segment Embeddings to distinguish between different sentence inputs; and 3) Position Embeddings to encode the sequential order of tokens, compensating for the Transformer's lack of recurrence.

Context-aware Predictions and Applications. By generating a probability distribution for each token based on its entire surrounding context, BERT excels at complex pattern recognition. While originally designed for text, its ability to capture nuanced dependencies makes it highly effective for non-text domains, including bioinformatics and cybersecurity, where it can be adapted to detect anomalies in CAN traffic sequences.

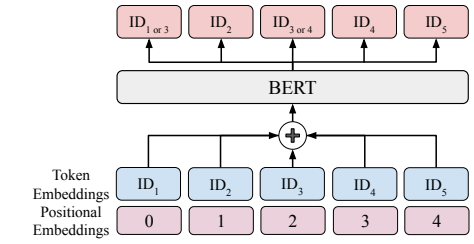
2.4 Related Works

Research on in-vehicle IDS is generally categorized into two primary domains: non-machine learning approaches based on physical or statistical rules, and machine learning-based approaches that learn behavioral patterns from data.

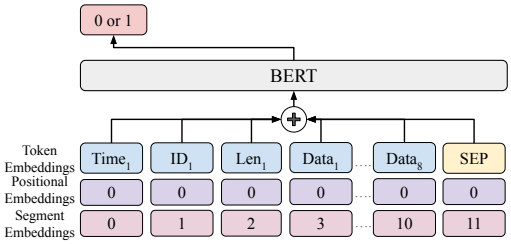
Non-Machine Learning Techniques. Traditional IDS techniques detect anomalies by monitoring the physical layer or statistical properties of CAN communication. These methods are typically computationally efficient but rely on static thresholds, making them brittle against sophisticated or evolving attacks. Key methodologies include:

- **Physical Fingerprinting:** Identifies intrusions by analyzing unique electrical characteristics of transmitting ECUs, such as clock skew [11] and voltage signal variations [12, 13].
- **Frequency-based Analysis:** Monitors the periodicity of message transmissions. Since many CAN messages are cyclic, deviations in arrival time or frequency often indicate malicious injection [20, 45].
- **Statistical Analysis:** Applies information-theoretic measures, such as entropy [38, 54] and Hamming distance [47] to detect irregularities in the distribution of message bits.
- **Vehicle State Analysis:** Cross-references various signals to evaluate the physical plausibility of the vehicle's state (*e.g.*, speed vs. wheel rotation) [55].

Machine Learning Techniques. Machine learning (ML) methods leverage data-driven models to learn normal traf-



(a) **CAN-BERT [4]**



(b) **CANBERT [43]**

Figure 2: **Input Embeddings and Output of Two Existing BERT Model-based IDSes.**

fic patterns, enabling the detection of subtle anomalies that escape static rules.

- **Classical Algorithms:** Methods such as support vector machines (SVMs) [1, 6], decision trees [39, 56], and extreme gradient boosting (XGBoost) [5, 19] are widely used for the binary classification of messages as either normal or malicious.
- **Deep Learning Models:** Deep neural architectures, including CNNs [16, 28], RNNs [2, 29], LSTMs [27], and DNNs [34] are employed to capture complex temporal and spatial dependencies within CAN streams.

Input Representation Strategies. A critical differentiator among ML-based IDSs is how they represent input data. Some approaches aggregate multiple messages (*e.g.*, a window of 29 frames) into a single input vector [35], while others classify attacks based on partial information from a single message [4].

Other Context-aware IDS. Several works incorporate CAN data bytes rather than relying solely on IDs. For instance, some methods process raw payload bytes directly [27, 31], while others operate on interpreted signals [57]. These approaches show that payload information is valuable for detecting replay attacks, but they often result in additional engineering overhead or computational complexity.

BERT-based IDS Solutions. Recent studies have extended BERT to the automotive domain, leveraging its Transformer architecture to model sequential dependencies in CAN traffic. As illustrated in Figure 2, two notable approaches are CANBERT [43] and CAN-BERT [4], which adopt distinct strategies.

	Context	# Tokens Per Msg	Model Update
CAN-BERT [4]	✗	1	✗
CANBERT [43]	✓	11	✗
CANBERT+	✓	1	✓

Table 1: **Comparison with Other BERT Model IDS.**

- **CAN-BERT (ID-Only)**: A lightweight model that processes only the CAN ID sequence. As shown in Figure 2a, it predicts a ranked list of probable next IDs. Detection is threshold-based, if the observed ID falls outside the predicted set, the message is flagged as anomalous.
- **CANBERT (Full-Frame)**: A comprehensive model that treats each CAN message as a sentence, tokenizing the ID, data fields, and timestamps, as described in Figure 2b. It uses a "SEP" token to delimit messages and performs direct classification, eliminating the need for post-processing thresholds.

Limitations of Existing BERT-based IDS. Despite their novelty, both architectures suffer from critical limitations that hinder practical deployment, as summarized in Table 1.

- **Lack of Context in CAN-BERT**: By restricting input to CAN IDs, CAN-BERT ignores the payload (data length and data bytes). Consequently, it very likely fails to detect sophisticated attacks where valid IDs are paired with malicious payloads (*e.g.*, replay or fuzzy attacks with correct headers).
- **Computational Inefficiency in CANBERT**: CANBERT achieves context awareness by tokenizing every component of a message, requiring approximately 11 tokens per message (8 data bytes with 3 additional metadata fields). This drastically reduces the number of messages that can be analyzed in a single pass. Given a 256-token limit, CAN-BERT (ID-only) can process 256 messages, whereas CANBERT is limited to roughly 21 messages, which a 92% reduction in throughput. This overhead is prohibitive for real-time automotive applications.
- **Vocabulary Instability (Timestamps)**: Incorporating raw timestamps into the vocabulary, as done in CANBERT, introduces severe Out-Of-Vocabulary (OOV) issues. In real-world scenarios, timestamps differ continuously from training data, leading to a sparse embedding space and poor generalization.

3 Threat Model

Our threat model operates within the standard automotive security framework, assuming the CAN lacks intrinsic security mechanisms such as message authentication or encryption. Consequently, the bus functions as a broadcast medium where any connected node can read or transmit messages. We align our assumptions with established literature, including the framework utilized in CANBERT [43], positing that an

adversary has successfully compromised at least one ECU or an external gateway.

3.1 Adversary Capabilities and Access

We assume the adversary possesses full control over a compromised node, enabling them to eavesdrop on bus traffic, fabricate messages, and manipulate transmission timing. Such attacks could be executed through various attack surfaces, including internal vehicle components [30, 37, 51] and connected accessories [32, 50, 52, 53]. Access to the vehicle network may be established through two primary vectors:

- **Physical Access:** The adversary connects directly to the On-Board Diagnostics (OBD-II) port or physically taps into the CAN wiring harness to inject malicious traffic.
- **Remote Access:** As modern vehicles increasingly rely on connectivity, adversaries may exploit vulnerabilities in telematics units, infotainment systems, or over-the-air (OTA) update mechanisms. Moreover, emerging Vehicle-to-Everything (V2X) interfaces expose the CAN bus to long-range remote exploitation [10].

3.2 Attack Scenarios

Once access is established, the adversary utilizes the compromised node to launch CAN message injection attacks. The goal is to destabilize vehicle operations or override legitimate control commands. We consider the following specific attack types discussed in §2.2.

- **Spoofing (Impersonation):** The adversary injects messages with high-priority IDs (*e.g.*, 0x000) or legitimate IDs (*e.g.*, RPM or Gear status) to deceive other ECUs into actuating unsafe behaviors, such as sudden braking or steering manipulation.
- **Replay Attack:** The adversary captures valid, time-sensitive CAN frames and re-transmits them at a later time. This acts as a logical Man-in-the-Middle (MitM) attack, intended to trigger specific vehicle states without requiring knowledge of the proprietary signaling protocol.
- **Fuzzing:** The adversary injects frames with randomized arbitration IDs and payloads. This technique is used to discover undocumented vulnerabilities or to cause unpredictable behavior in receiving ECUs.
- **Denial-of-Service (DoS):** The adversary floods the bus with high-priority messages to monopolize bandwidth. This prevents legitimate critical messages from reaching their destinations, effectively paralyzing vehicle communication.

We also consider hybrid attack strategies, where sophisticated adversaries combine techniques (*e.g.*, executing a DoS attack to mask a simultaneous Spoofing attempt), thereby raising the complexity for intrusion detection systems.

3.3 Scope and Exclusions

Our model explicitly excludes complete ECU shutdown or "bus-off" attacks, where an ECU is silenced entirely. While dangerous, we omit these for two specific reasons:

- **Nature of Detection:** Our proposed IDS is designed to analyze traffic semantics and timing anomalies (active intrusions), whereas a shutdown represents a total cessation of signal (passive failure).
- **Mitigation Strategy:** Countering node failures typically falls under the domain of hardware redundancy and fault-tolerant network design, which is distinct from the software-centric traffic analysis focus of this work.

4 Objectives

This work primarily targets the detection of CAN message injection attacks, wherein adversaries introduce unauthorized frames to mimic or disrupt legitimate vehicle communications. As these attacks allow for the manipulation of vehicle behavior, often without requiring physical access, they represent a critical safety vulnerability. To effectively mitigate this threat, our proposed IDS is guided by four core objectives:

- **High Detection Accuracy:** The system must reliably distinguish between benign traffic and malicious injections by modeling complex contextual dependencies and identifying anomalous deviations in message sequences.
- **Performance Efficiency:** Given the resource-constrained nature of automotive ECUs, the detection mechanism must be computationally lightweight to ensure low-latency performance suitable for real-time deployment.
- **Model Universality (Generalizability):** Recognizing the diversity of CAN traffic across different vehicle manufacturers and models, the IDS must be architecture-agnostic and capable of generalizing across varied datasets without requiring extensive retraining.
- **Granular (Single-Message) Detection:** Unlike window-based approaches that flag large blocks of time, our system aims to detect threats at the single-message level, enabling precise identification of specific malicious frames for rapid forensic analysis and mitigation.

4.1 Insights and Design Choices

Since vehicles are safety-critical and conduct their tasks in real time, we need to customize and optimize IDS to make it applicable as much as possible. To meet these objectives, we introduce several novel insights and design choices informed by the strengths and limitations of the BERT architecture:

- **Redundant Layer Elimination:** After completely understanding the usage of each layer of original BERT, we discard the segmentation layer, since the role of the segment layer and the positional layer is identical in our design.

- **OOV Prevention via Input Layer:** Since sequential information is critical for intrusion detection, a mechanism is required to represent message order without causing OOV issues. As illustrated in Figure 1, BERT inherently captures sequential relationships through positional embeddings, making explicit timestamps redundant. By replacing timestamps with positional embeddings, we eliminate the OOV problem while allowing the model to learn message ordering directly from positional cues.
- **Multiple Input Layers Utilization with Context Awareness of Data:** Capturing message context is essential for detecting complex attacks such as spoofing and replay. However, naively augmenting inputs with additional context is computationally inefficient, which is applied by CANBERT previously. To address this, we exploit BERT's ability to incorporate multiple embedding layers, thereby enriching the context representation of tokens. Inspired by prior work in reverse engineering [33], we design separate embedding layers for distinct CAN message components with additional embeddings for each data information. This design enables each input token to encapsulate more comprehensive contextual information, as summarized in Table 1.
- **Unique Tokens for Context Awareness of Data Bytes:** Although integrating all message components into single tokens improves representation, context ambiguity may persist if identical embeddings are applied to different byte positions (*e.g.*, the first and second data bytes). To preserve context integrity, we assign unique identifiers to each data byte within its embedding layer. This ensures clear differentiation across byte positions and enhances the model's ability to capture the true context of message data.

5 CANBERT+ Design

5.1 Data Pre-processing

To ensure consistency and computational efficiency during learning, all datasets undergo a rigorous standardization process. This normalization allows CANBERT+ to interpret CAN traffic uniformly, regardless of the original source or logging format.

Following the architecture established by CANBERT, our model processes the three primary components of a CAN frame: the CAN ID, the DLC, and the 8-byte data field. Notably, timestamps are deliberately excluded from the input features. Since timestamps are specific to the recording epoch of a dataset, including them would introduce OOV issues and hinder the model's ability to generalize across different vehicle platforms or temporal contexts.

The data requirements differ between the two training phases:

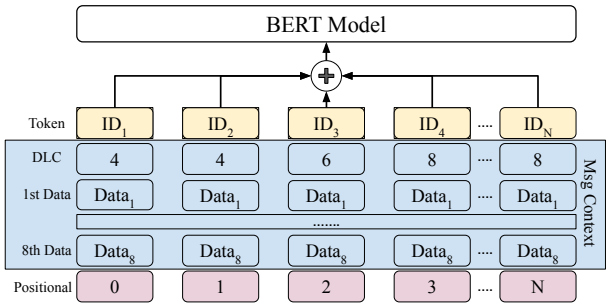


Figure 3: **Input Embeddings of CANBERT+.**

- **Pre-training:** The model utilizes a massive corpus consisting exclusively of benign CAN messages. This unlabeled approach forces the model to learn the intrinsic structural properties and contextual dependencies of legitimate vehicle network traffic.
- **Fine-tuning:** Each message is annotated with a binary label, benign (0) or malicious (1), to enable supervised learning for intrusion detection tasks.

The standardized message format follows a specific sequence: `0316 8 05 20 ea 0a 20 1a 00 7f`. In this example, `0316` represents the CAN ID, `8` denotes the DLC, and the subsequent hexadecimals represent the 8-byte payload. To maintain a fixed-length input vector, messages with data fields shorter than 8 bytes are padded with a reserved integer token (-1). This padding ensures that the transformer architecture receives a consistent structural representation across all frames.

5.2 Pre-training Phase

The pre-training phase allows CANBERT+ to develop a foundational understanding of the vehicle's communication context by learning from unlabeled, benign CAN traffic. This stage is critical for model generalization, ensuring robustness even when the system is deployed on smaller datasets or previously unseen vehicle architectures. This phase is divided into three core components: optimized input embeddings, model pre-training via MLM, and the loss function.

Optimized Multi-layer Input Embeddings. As illustrated in Figure 3, CANBERT+ utilizes a specialized embedding layer that integrates 10 distinct features alongside a positional embedding. These features encompass the CAN ID, the DLC, and each of the eight data bytes.

Unlike standard NLP Transformers, CANBERT+ omits segment embeddings. Because each input token represents a single, self-contained CAN message, the traditional need for sentence-pair separation markers is eliminated, thereby reducing computational overhead and improving tokenization efficiency.

Data Byte Differentiation For Context Awareness. A significant challenge in CAN data representation is that different bytes within a payload often share identical values (*e.g.*, 0x00) while representing entirely different sensor signals or states. Without differentiation, two distinct messages, such as `0316 8 00 00 00 00 00 00 00 10` and `0316 8 00 00 00 00 00 00 10 00`, might result in identical aggregated embeddings, causing the model to lose critical spatial context.

To preserve the positional significance of each byte, CANBERT+ implements a prefix-based differentiation scheme:

- **Byte Indexing:** Each data byte is prepended with a "D + index" identifier. For example, a value of 0 in the first byte becomes `D10`, while a 0 in the eighth byte becomes `D80`.
- **Length Tagging:** The DLC is prepended with an "L" identifier (*e.g.*, `L8`) to distinguish it from numerical values found in the CAN ID or data fields.
- **Collision Avoidance:** We use alphabetic prefixes (`D` and `L`) rather than pure integers to prevent token collisions with CAN IDs, which are natively represented as integers.

After differentiation, the 10 separate tokens (ID, DLC, and 8 bytes) are aggregated into a single high-dimensional vector that encapsulates the complete state of a unique CAN message.

Pre-training via MLM. Since the pre-training corpus is unsupervised, we employ a MLM strategy. In this setup, 15% of the CAN IDs are randomly masked. The model is then tasked with reconstructing these missing IDs based on the surrounding message context.

We deliberately focus the masking process on the CAN ID rather than the aggregated data tokens. As the CAN ID serves as the primary identifier for the message's function, forcing the model to predict it ensures a deeper understanding of the relationship between message frequencies and payload patterns. Moreover, we omit the NSP task, as the sequential nature of CAN traffic makes next-ID prediction redundant when MLM is already being performed on a per-message basis.

Loss Function. To evaluate performance and guide parameter updates, we utilize the Negative Log Likelihood (NLL) loss function:

$$L = -\sum_{n=1}^{N} \log(p(y_n|x_n))$$

where N represents the total number of CAN messages, y is the ground-truth CAN ID, x is the input context, and p is the predicted probability distribution over the possible CAN IDs. The optimizer iteratively minimizes this loss, progressively refining the model's ability to predict message patterns with high precision.

5.3 Fine-tuning Phase

Following the pre-training stage, CANBERT+ undergoes fine-tuning using a labeled dataset containing both benign and malicious CAN messages. This phase specializes the model for IDS by refining its ability to distinguish legitimate traffic from attack patterns learned during the unsupervised phase.

Architecture Adaptation. To maintain consistency in data representation, CANBERT+ retains the same input embedding structure used during pre-training. However, the output head is modified for supervised classification. Instead of predicting masked tokens, the model is appended with a dedicated classification head that maps the learned high-dimensional representations to a binary output.

The classification head typically consists of a fully connected (Linear) layer followed by specific activation stages.

1. **Non-linearity:** A ReLU (Rectified Linear Unit) activation is applied within the hidden layers of the classification head to capture complex, non-linear relationships between CAN message features.

2. **Probability Mapping:** For the final output, we employ a Sigmoid activation function (for a single output node) or a Softmax function (for a dual-node output). This ensures the model generates a probability distribution between 0 and 1, representing the confidence level that a given message is malicious.

The Value of Pre-training. While it is possible to train the model directly on labeled data (supervised learning only), this often leads to suboptimal performance, particularly with limited datasets. Direct training may fail to capture the deep contextual nuances of the CAN bus protocol. Pre-training on a large corpus of benign data provides a robust "weight initialization" that allows the model to generalize effectively, even when fine-tuned on a relatively small number of attack samples.

Loss Function. Since fine-tuning is framed as a binary classification task, we utilize the Binary Cross-Entropy (BCE) Loss function to guide the training process:

$$L = -\frac{1}{N}\sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

where N is the number of CAN messages in the batch, y_i is the ground-truth label, $\hat{y}_i$ is the predicted probability of the message being malicious. The optimizer iteratively updates the model parameters to minimize this loss, penalizing high-confidence incorrect predictions. This process ensures CANBERT+ achieves high-confidence, real-time detection of cyber-attacks within the vehicle network.

Dataset	# Logs	# Msgs	
		Benign	Attack
2019 Challenge [22]	16	4,829,571	1,086,989
CanML [8]	192	146,677,755	1,297,569
CSU [15]	4	82,724	14,089
Hacking [21]	4	14,237,958	2,331,517
Mcan [25]	2	1,635,080	500,000
Bcan [24]	2	5,019,845	500,000
Road [18]	29	3,203,951	62,513
Survival [23]	9	1,105,903	183,314
Total	258	176,792,787	5,975,991

Table 2: **List of Datasets and The Number of Logs in The Corresponding Dataset.**

Parameter	Value
Hidden size	768
Number of transformer layers	6
Attention heads	12
Dropout	0.1
Learning rate	0.001
Optimizer	AdamW
Adam beta	0.9, 0.999
Patience	10
Number of epochs	200
Batch size	128

Table 3: CANBERT+ **Parameter Configuration.**

6 Evaluation

6.1 Evaluation Setup

Metrics. To evaluate CANBERT+, we primarily employ the F1 score, the gold standard for assessing IDS due to its robustness against class imbalance. However, because our large-scale evaluation includes datasets where malicious messages occur sporadically, certain test subsets may contain zero attack samples. In these instances, Precision, Recall, and F1 scores are mathematically zero (or undefined), regardless of model performance. To account for these edge cases and demonstrate model stability, we report accuracy for these specific scenarios in Appendix B.5.

Datasets. Previous CAN IDS research has often been constrained by limited evaluation scopes; for instance, CANBERT and CANBERT were validated on only 9 and 4 data logs, respectively. To ensure a more rigorous assessment, we evaluate CANBERT+ on a massive corpus of 258 data logs spanning eight distinct sources, as described in Table 2. These logs encompass a wide array of attack methodologies, including replay, masquerade, and fuzzy attacks.

Evaluation Environment. All experiments were performed on a workstation equipped with an NVIDIA GTX 1080 Ti GPU and 64GB of RAM. We acknowledge that this setup is not aligned with in-vehicle hardware constraints. Hence, we describe potential application to real-world in the later section.

Parameter Configuration. The BERT architecture within CANBERT+ utilizes the hyperparameters detailed in Table 3. To prevent overfitting and optimize training efficiency, we implement an early stopping mechanism: fine-tuning terminates if the training loss fails to improve for ten consecutive epochs.

Availability. The scripts and datasets are available at https://github.com/OSUSecLab/CanBERTplus and https://zenodo.org/records/20380113.

Baselines. We benchmark CANBERT+ (denoted as OUR) against two categories of established models:

- **BERT-based IDS:** We compare against modified versions of CAN-BERT and CANBERT. We refer CAN-BERT and CANBERT as B1 and B2, respectively, to differentiate them clearly. For a fair comparison, these models were adapted for binary classification and utilized our standardized pre-processing (*e.g.*, timestamp removal). Hyperparameters, such as the number of layers, were synchronized across all BERT-based models. Notably, B2 was limited to three epochs during fine-tuning due to significant computational overhead, which we detail in the §6.7.
- **Machine Learning Baselines:** We evaluate performance against five traditional techniques: Support Vector Machine (SVM), Random Forest (RF), XGBoost (XGB), Logistic Regression (Reg), and Long Short-Term Memory (LSTM). LSTM hyperparameters were configured following established benchmarks in the field [27].

Evaluation Methodology. To prevent data leakage, we employ a strict 80/20 train-test split for each data log. Given our scale, we trained and independently evaluated 258 distinct models. For the 2019 Challenge dataset, we adhered to the official training and testing splits to ensure an unbiased assessment of the model's performance on unseen data.

Presentation of Results. Given the volume of results, we present our findings as follows:

- **Bar Charts:** Used for datasets with fewer logs (*e.g.*, Bcan) to emphasize performance margins.
- **Tables:** Used for high-volume datasets to ensure density and readability.
- **Appendix:** As noted, if a test subset contains zero malicious messages (rendering the F1 score zero), we move those accuracy-based stability results to the Appendix to avoid skewing the primary performance discussion.

6.2 Single Analysis Performance

In the single-analysis evaluation, we independently fine-tune models from a common pre-trained baseline and evaluate them on each individual data log. This granular approach ensures a rigorous and detailed assessment of model performance across diverse scenarios.

	SVM	RF	XGB	Reg	LSTM	B1	B2	OUR
	1st dataset							
File_1	0.93	0.99	0.99	0.94	0.99	0.46	0.99	0.99
File_2	0.81	0.99	0.42	0.80	0.93	0.45	0.99	0.99
File_3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
File_4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Fuzzy_Sonata	0.93	0.99	0.99	0.94	0.98	0.56	0.99	0.99
Fuzzy_Soul	0.80	0.99	0.44	0.81	0.98	0.48	0.99	0.99
Malfunc_Sonata	0.00	1.00	0.80	0.00	0.66	0.00	1.00	1.00
Malfunc_Soul	0.00	0.65	0.00	0.00	0.00	0.00	0.66	0.70
	2nd dataset							
File_1	1.00	1.00	0.66	0.99	1.00	0.89	1.00	1.00
File_2	1.00	1.00	0.66	1.00	1.00	0.87	1.00	1.00
File_3	0.00	0.63	0.51	0.00	0.26	0.42	0.62	0.63
File_4	0.00	0.68	0.58	0.00	0.42	0.34	0.68	0.73
Malfunc_Sonata	1.00	1.00	0.00	1.00	1.00	0.00	1.00	1.00
Malfunc_Soul	1.00	1.00	0.99	0.99	1.00	0.00	1.00	1.00
Replay_Sonata	0.00	0.60	0.49	0.00	0.26	0.43	0.60	0.72
Replay_Soul	0.00	0.67	0.53	0.00	0.25	0.41	0.67	0.71

Table 4: **F1 Score on 2019 Challenge Dataset.**

	SVM	RF	XGB	Reg	LSTM	B1	B2	OUR
signal_1	0.99	0.99	0.88	0.99	0.99	0.00	0.99	0.99
signal_2	0.99	0.99	0.98	0.99	0.99	0.00	0.99	1.00
signal_3	1.00	1.00	0.00	1.00	1.00	0.00	1.00	1.00
signal_spoof_1	0.99	0.99	0.89	0.99	0.99	0.00	0.99	0.99
signal_spoof_2	0.99	0.99	0.98	0.99	0.99	0.00	0.99	1.00
signal_spoof_3	0.99	0.99	0.00	0.99	0.99	0.00	0.99	1.00
engine	1.00	1.00	0.00	0.00	0.00	0.00	1.00	0.62
engine_spoof	1.00	1.00	0.00	0.00	0.00	0.00	1.00	0.18
speed_3	0.99	0.99	0.99	0.99	0.99	0.97	0.32	0.99
speed_spoof_3	0.99	0.99	0.99	0.98	0.99	0.12	0.19	0.99
light_off_1	0.00	0.99	0.93	0.00	0.00	0.00	0.99	0.99
light_off_2	0.98	0.99	0.37	0.99	0.99	0.00	0.95	0.99
light_off_spoof_1	0.00	0.99	0.12	0.00	0.00	0.00	0.99	0.99
light_off_spoof_2	0.98	0.99	0.30	0.99	0.29	0.00	0.84	0.99

Table 5: **F1 Score on Road Dataset.**

	SVM	RF	XGB	Reg	LSTM	B1	B2	OUR
forester								
combined-tri-2	0.83	0.96	0.41	0.90	0.94	0.00	0.99	1.00
fuzzing-1	0.67	0.99	0.71	0.69	0.95	0.00	0.99	0.99
interval-1	0.99	1.00	0.00	0.99	1.00	0.99	1.00	1.00
interval-2	0.00	1.00	0.00	0.00	0.00	0.66	1.00	1.00
rpm-1	0.00	0.98	0.00	0.00	0.00	0.00	0.99	0.99
speed-1	0.99	1.00	0.00	0.94	0.99	0.00	1.00	1.00
speed-2	0.99	1.00	0.00	0.00	1.00	0.00	1.00	1.00
impala								
fuzzing-1	0.35	0.99	0.19	0.34	0.32	0.98	0.99	0.99
interval-2	0.97	1.00	0.00	0.97	1.00	0.75	1.00	1.00
interval-3	0.89	1.00	0.00	0.89	0.99	0.71	1.00	0.99
system-1	0.88	0.94	0.93	0.88	0.94	0.99	0.99	0.99
silverado								
combined-tri-1	0.00	0.99	0.00	0.00	0.97	0.02	0.99	0.40
gear-1	0.00	1.00	1.00	0.00	0.00	0.00	1.00	1.00
interval-1	0.88	1.00	0.00	0.91	0.99	0.94	1.00	1.00
interval-2	0.00	1.00	0.00	0.00	0.99	0.00	1.00	1.00
interval-3	0.98	1.00	0.00	0.98	1.00	0.48	1.00	1.00
speed-accessory-1	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.97
speed-accessory-2	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.29
system-3	0.44	0.99	0.38	0.54	0.87	0.00	0.99	0.99
traverse								
interval-1	0.00	0.99	0.00	0.00	0.00	0.99	0.99	0.99
standstill-2	0.00	1.00	0.00	0.00	1.00	0.00	1.00	1.00

Table 6: **F1 Score on CanML Dataset.**

2019 Challenge Dataset. As shown in Table 4, CANBERT+ demonstrates superior performance, outperforming all baseline techniques across every log. Notably, CANBERT+ significantly improves the detection of replay attacks, achieving F1 scores of 0.72 and 0.71. This clearly surpasses the next-best model, B2, which obtained 0.60 and 0.67, respectively. While these results represent a new state-of-the-art, the detection of sophisticated replay attacks remains an area for future refinement.

Road Dataset. On the 14 Road logs as described in Table 5, CANBERT+ outperformed all methods with two exceptions: the `engine` and `engine_spoof` logs. In these specific cases, RF, SVM, and B2 achieved near-perfect F1 scores. We attribute the lower initial performance of CANBERT+ to the small log sizes, which triggered premature early stopping. By increasing the training limit to 1,000 epochs, CANBERT+ eventually reached F1 scores of 0.83 on `engine` and 0.77 on engine_spoof, though it still trailed the some baselines in these low-data regimes.

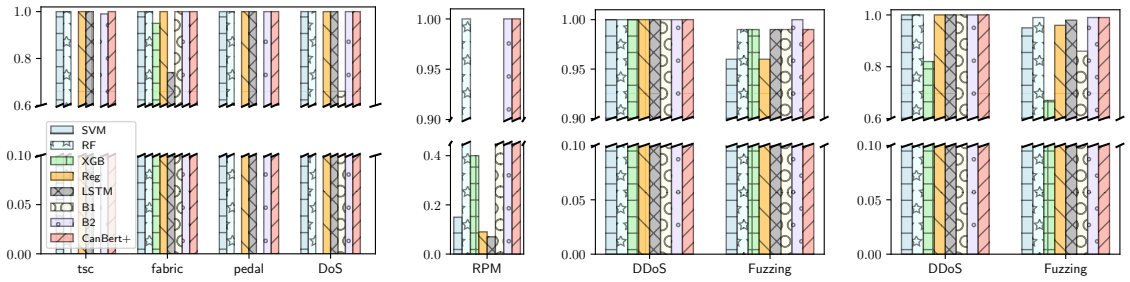
CanML Dataset. This dataset encompasses 21 logs from four vehicle models: Chevrolet Silverado, Subaru Forester, Chevrolet Traverse, and Chevrolet Impala, as illustrated in Table 6. CANBERT+ consistently achieved F1 scores near 1.0 across the majority of the logs. It was only outperformed on `combined-triple-1` (by RF, LSTM, and B2) and `speed-accessory-1` (by B2). Similar to the Road dataset, extending the fine-tuning process for `combined-triple-1` allowed CANBERT+ to reach an F1 score of 0.99, matching the top-performing baselines.

CSU and Hacking Datasets. In the CSU dataset, CANBERT+ achieved perfect F1 scores on every log. While SVM, RF, and logistic regression also reached perfect scores, XGB and B1 failed on several logs, as shown in Figure 4a. Similarly, for the RPM log in the Hacking dataset, CANBERT+, RF, and B2 all attained 1.0 F1 scores, significantly distancing themselves from other techniques, as described in Figure 4b.

Mcan and Bcan Datasets. Nearly every evaluated IDS, with the exception of XGB, achieved a 1.0 F1 score on DoS attacks. In the more challenging fuzzing scenarios, CANBERT+ maintained a high F1 score of 0.99, as shown in Figure 4c and Figure 4d. Only B2 achieved a marginal lead with a 1.0 F1 score on the Mcan Fuzzy log.

Survival Dataset. On the Survival dataset, CANBERT+ consistently ranked among the top performers, as illustrated in Figure 5, often matching the performance of the B2 baseline.

Summary. Overall, CANBERT+ surpasses both traditional machine learning techniques and the two BERT-based baselines in the single-analysis setting. Its advantages are most pronounced in replay attack detection and scenarios where payload context is critical for identifying anomalies.



(a) CSU (b) Hack (c) Mcan (d) Bcan

Figure 4: **F1 Score on CSU, Hacking, Mcan, and Bcan Datasets.**

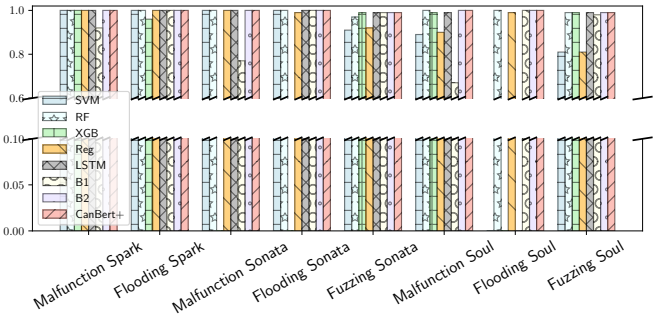


Figure 5: **F1 Score on Survival Dataset.**

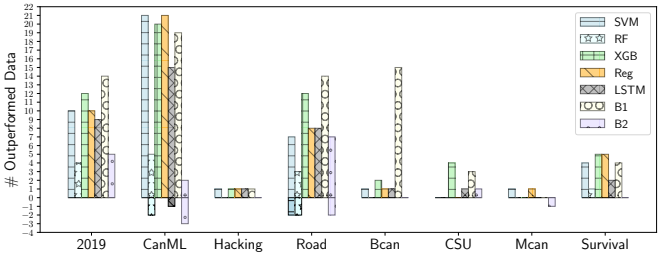


Figure 6: **The Number of Datasets Each Model Outperforms Except for The Tie in Terms of F1 Score. Positive and Negative mean the Number CANBERT+ Outperforms and Underperforms, Respectively.**

	SVM	RF	XGB	Reg	LSTM	B1	B2	OUR
1st	0.81	0.93	0.89	0.82	0.98	0.00	0.98	0.99
2nd	0.33	0.76	0.63	0.57	0.63	0.46	0.78	0.99

Table 7: **F1 Score on 2019 Challenge Dataset with Train and Test.**

	# Hidden Dimensions	# Layers	# Attention Heads
CANBERT+	768	6	12
S-CANBERT+	60	2	12
T-CANBERT+	12	2	4

Table 8: **Configurations of Different Size Models.**

Figure 6 visualizes the comparative performance across all datasets containing at least one malicious message. The chart illustrates the "performance margin":

- **Positive values:** The number of logs where CANBERT+ outperforms the competitor.
- **Negative values:** The number of logs where the competitor outperforms CANBERT+.

Our analysis shows that while B2 and RF are the closest competitors, outperforming CANBERT+ on 6 and 4 logs in total, respectively, CANBERT+ maintains the highest aggregate performance and stability across the widest variety of attack vectors.

6.3 Training and Testing Performance

For the 2019 Challenge dataset, we strictly adhere to the original experimental setup to evaluate the model's generalization capabilities. Models are fine-tuned on the provided training logs (grouped by attack type) and subsequently evaluated on the separate release logs (Files 1–4). This process yields two distinct models corresponding to the first and second release datasets.

Table 7 summarizes the resulting F1 scores. On the first release dataset, CANBERT+ achieves an F1 score of 0.99, while LSTM and B2 both follow closely at 0.98, significantly outperforming traditional machine-learning baselines. On the second release dataset, which proved considerably more challenging, CANBERT+ maintains an F1 score of 0.99. In contrast, no other baseline IDS surpassed the 0.8 F1 threshold. These results demonstrate that CANBERT+ generalizes exceptionally well across official data splits and remains highly competitive with full BERT architectures by utilizing a representation specifically tailored for CAN bus traffic.

	Param	Model	Ckpt	Mem	Time
CANBERT+	61.7M	246M	708M	702.95MB	0.227*ms*/msg
S-CANBERT+	1.6M	6.3M	50.7M	281.6M	0.199*ms*/msg
T-CANBERT+	315K	1.2M	16.62M	274.5M	0.192*ms*/msg

Table 9: **Statistics of Different Model Configurations.**

6.4 Different Configuration Model Performance

To assess the feasibility of deployment in resource-constrained automotive environments, we evaluate two compact variants: S-CANBERT+ and T-CANBERT+ following the configuration described in Table 8. Both variants significantly reduce the total parameter count; notably, T-CANBERT+ is approximately 205 times smaller in model size than the full CANBERT+ architecture.

Comparative Efficacy. All three variants were trained and evaluated on the 2019 Challenge dataset using the standardized pre-training and fine-tuning procedures illustrated in Figure 7. We excluded `File_3` and `File_4` from the first dataset analysis, as all evaluated methods yielded an F1 score of 0. These compact variants remain remarkably competitive:

- CANBERT+ typically maintains the strongest performance on sophisticated replay attacks.
- S-CANBERT+ achieved the highest F1 score on the `File_3` log in the second release dataset.
- T-CANBERT+ offers a "high-efficiency" alternative, providing comparable F1 scores at a fraction of the computational cost.

Hardware and Latency Metrics. As detailed in Table 9, the reduction in footprint is substantial. Peak memory usage during testing, measured via the `max_memory_allocated` function, dropped from 703*MB* (Full) to 280*MB* (Compact). Moreover, processing latency is significantly improved. While the CANBERT+ requires 0.227*ms* to process a single CAN message, S-CANBERT+ reduces this to 0.199*ms*. This 13% increase in throughput is critical for real-time intrusion detection, where meeting the timing constraints of high-speed CAN networks is paramount.

6.5 Performance Comparison with Other IDS

We benchmark T-CANBERT+ against prior state-of-the-art IDSs [35, 40] using the hacking dataset. Because the source code for these systems is not publicly available, we utilize the performance metrics reported in their respective publications. To ensure a direct comparison, we adopt the F1 score as the primary metric. Notably, because the Hacking dataset is significantly imbalanced and prior works do not specify their data-splitting logic, we employed a shuffled, batch-level split for our training and test sets.

Model	F1 Score			
	DoS	Fuzzy	Gear	RPM
DCNN ResNet [46]	0.9985	0.9979	0.9988	0.9989
CE ResNet [26]	0.9987	0.9987	0.9993	0.9993
SupCon ResNet [26]	0.9999	0.9997	0.9997	0.9997
Rec-CNN [16]	0.9862	0.8619	0.8763	0.8604
LSTM [40]	0.9919	0.9670	0.9833	0.9650
CNN-LSTM [40]	0.9969	0.9949	0.9945	0.9913
Transformer [40]	1.0000	0.9999	0.9953	0.9996
Time IDS [35]	**1.0000**	**1.0000**	**1.0000**	**1.0000**
T-CANBERT+	**1.0000**	0.9999	**1.0000**	**1.0000**

Table 10: **Comparison with Existing IDS on Hacking Dataset.**

Table 10 summarizes the results. T-CANBERT+ achieves perfect F1 scores across three of the four attack scenarios (DoS, Gear, and RPM), matching the performance of Time IDS. Other Transformer-based IDSs only reached a perfect score on the DoS case, showing overall lower performance.

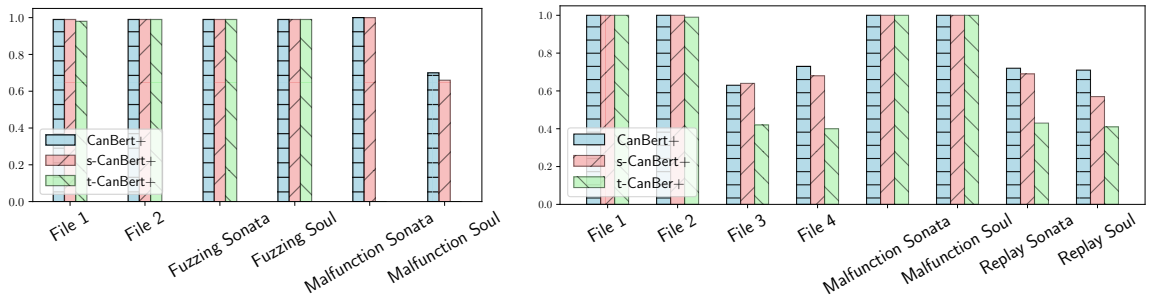
While Time IDS appears to slightly outperform T-CANBERT+ on the fuzzy attack, the comparison is not equivalent. Time IDS aggregates 29 CAN messages into a single frame and labels the entire frame as malicious if any constituent message is anomalous. This aggregation makes the data low entropy and detection simple. For example, their aggregated fuzzy dataset has a 2:1 normal-to-attack ratio, whereas the raw dataset contains over 1.1 million messages where attacks represent a tiny fraction. Moreover, multiple attack messages within a single frame can collectively trigger a detection, even if individual messages are subtle.

In contrast, T-CANBERT+ operates at the single-message level on highly imbalanced, raw traffic. Detecting attacks at this granularity is substantially more difficult but far more practical for real-vehicle deployment, where security actions must be tied to specific, individual messages.

6.6 Ablation Study

To evaluate the necessity of data differentiation, we conducted an ablation study using the `Replay_Soul` log from the 2019 challenge dataset. In the ablated variant, identical payload values are assigned the same token regardless of their byte position (*e.g.*, a value of 0xFF at byte 3 and byte 5 would share the same token).

As shown in Figure 8, while all three models (B2, CANBERT+ without differentiation, and the full CANBERT+) achieve F1 scores near 0.7, the full model consistently maintains the lead. The slight performance drop in the ablated version confirms that precise positional context within the CAN payload is vital for detecting sophisticated replay attacks that exploit specific signal offsets.



(a) 1st Dataset

(b) 2nd Dataset

Figure 7: **F1 Score on 2019 Challenge Dataset of Different Configuration Setups.**

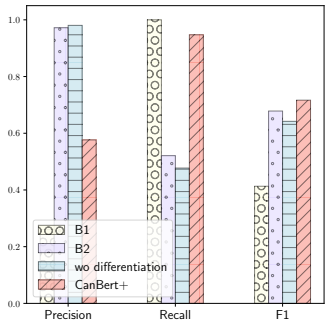


Figure 8: **F1 Score of CANBERT, CAN-BERT, and CANBERT+ With and Without Data Differentiation.**

6.7 Computational Overhead and Efficency with Other BERT IDS

A primary contribution of CANBERT+ is the optimization of input embeddings to resolve the significant overhead found in previous BERT-based models like B1 and B2.

Training Efficiency. Original B2 architectures utilize multiple tokens for a single CAN message, leading to explosive growth in sequence length. Our tests on the `Replay_Soul` log demonstrate a dramatic difference in training speed:

- B2 requires around $2,000$ seconds per epoch.
- CANBERT+ requires around 150 seconds per epoch, which is 13 times faster.

At a standard 200-epoch training cycle, the original B2 would require approximately 111 hours, whereas CANBERT+ completes the same task in just 8 hours. Due to this massive disparity, we limited the original B2 to 3 epochs for our broader multi-dataset evaluation to make the study feasible.

Inference and Deployment Latency. The overhead of original BERT models also hinders real-time vehicle integration. In a test involving 25,502 messages, B2 and CANBERT+ takes 34.52 and 5.8 seconds, respectively. That is, CANBERT+ is 6 times faster compared to B2. By avoiding the "flattening" of messages into massive token sequences, CANBERT+ achieves the throughput necessary for high-speed automotive networks without sacrificing the deep contextual learning of the Transformer architecture.

6.8 Environment Setup Analysis

While our evaluation was conducted on a workstation-grade NVIDIA GTX 1080 Ti, this hardware represents a research-oriented baseline rather than a prerequisite for deployment. A critical requirement for any in-vehicle IDS is the ability to process traffic in real time without introducing bus latency.

Real-Time Performance. At a standard CAN baud rate of 500kbit/s, the worst-case transmission time for an 8-byte message is approximately $0.25ms$, with an average inter-message interval often exceeding $1ms$ in non-saturated networks [49]. Our CANBERT+ model achieves an average inference latency of $0.227ms$ per message. When considering our compact variants, S-CANBERT+ and T-CANBERT+, latency is reduced by an additional 13%, bringing processing time well within the safety margins required for real-time monitoring on active CAN buses.

Automotive-Grade Hardware Compatibility. Current automotive computing trends are shifting toward centralized, high-performance platforms that match or exceed our experimental setup. For example, the Nvidia Jetson Xavier and Orin delivers up to 254 TOPS (Tera Operations Per Second), providing significantly more AI throughput than the 1080 Ti baseline [41, 42]. Moreover, Qualcomm Snapdragon Ride features dedicated Transformer accelerators specifically designed to handle models like CANBERT+ with minimal power consumption [44]. These platforms demonstrate that the computational requirements of CANBERT+ are fully compatible with production-grade hardware found in modern Level 2 and 3 autonomous vehicles.

Resource Optimization and Scalability. Beyond raw speed, our compact variants address the strict memory and power

budgets of embedded systems. By reducing the parameter count and memory footprint (dropping from 703*MB* to 280*MB* for testing), CANBERT+ can reside alongside other critical ECU functions. Moreover, we envision a hybrid deployment life-cycle:

- **Offline Training:** The resource-intensive fine-tuning process is performed on high-compute servers.
- **OTA Deployment:** Once optimized, the lightweight, fine-tuned model is delivered to the vehicle via Over-the-Air (OTA) updates, a standard practice in modern software-defined vehicles [14].

This architecture ensures that the vehicle benefits from state-of-the-art Transformer-based security without the need for constant, high-powered training hardware on board.

7 Discussion

7.1 Limitations

Despite the performance gains of CANBERT+, several challenges remain for production-scale deployment:

- **Resource Contention:** While our compact variants (S-CANBERT+ and T-CANBERT+) significantly reduce footprint, production ECUs must simultaneously manage safety-critical tasks and diagnostic services. Coexistence with these high-priority tasks requires further optimization of memory scheduling.
- **Stochastic Stability:** Deep learning training is inherently stochastic. Minor variations in fine-tuning can lead to performance outliers. For safety-critical automotive applications, moving toward deterministic or verifiable training pipelines is necessary to ensure consistent protection across a fleet.
- **Domain Adaptation:** Our results show that Transformer-based models are sensitive to specific vehicle platforms. Differences in CAN ID dictionaries and signal offsets across manufacturers (*e.g.*, Tesla vs. Ford) can hinder cross-vehicle generalization. Moreover, existing public datasets often contain simplified attack traces that may not capture the complexity of zero-day exploits in modern, highly connected vehicles.

7.2 Future Work

Our future research will focus on evolving both the model architecture and the underlying data quality:

- **Hybrid Defense-in-Depth:** We plan to integrate CANBERT+ with state-aware mechanisms that monitor physical vehicle dynamics (*e.g.*, abrupt changes in wheel speed or steering angle). This multimodal fusion will allow the IDS to distinguish between network noise and high-consequence cyber-physical attacks.
- **Minimalist Architectures:** We will continue to refine the hyperparameter space of CANBERT+ to identify "extreme" compact models that maintain Transformer-level context while competing with the low latency of CNNs or Gated Recurrent Units (GRUs).
- **Synthetic Data Generation:** To address dataset limitations, we aim to develop a Generative Adversarial Network (GAN)-based augmentation framework. This will generate realistic, balanced traffic that includes complex masquerade and timing-based attacks, providing a more rigorous testbed than currently available public logs.

Last, we try to apply IDS into actual vehicle hardware to measure its overhead and performance.

7.3 Ethical Considerations

This research utilizes only publicly available, anonymized CAN datasets that contain no personally identifiable information. Our system is designed solely for security monitoring and does not track, infer, or store any driver behavior or passenger data. The goal is to enhance the safety and integrity of the vehicle platform without compromising user privacy.

8 Conclusion

We present CANBERT+, a context-aware and efficient BERT-based IDS for in-vehicle CAN networks. CANBERT+ encodes each CAN message as a single token with multi-layer embeddings for identifier, data length, and byte-wise payload, together with a data differentiation scheme that preserves positional information for payload bytes. This design allows the model to exploit payload context while keeping sequence length small and avoiding timestamp-related vocabulary issues. We evaluate CANBERT+ on eight public datasets with 258 logs covering spoofing, fuzzing, DoS, and replay attacks. Across these datasets, CANBERT+ generally outperforms traditional machine-learning baselines and a standard BERT model, with the largest gains on replay attacks where message context is critical. To support deployment on resource-constrained platforms, we further introduced two compact variants, S-CANBERT+ and T-CANBERT+, which reduce model size and memory footprint by up to two orders of magnitude while retaining competitive detection performance. These results indicate that context-aware transformer models with carefully designed embeddings are a promising direction for practical IDS deployment in modern vehicles.

LLMs Usage. A LLM was utilized solely to polish the grammar, technical terminology, and stylistic flow of this manuscript. All conceptual frameworks, experimental designs, data analyses, and technical conclusions are the original work of the authors.

Acknowledgments

We sincerely thank the anonymous reviewers for their insightful comments and the shepherd for their invaluable guidance throughout the revision process. This work was supported in part by the U.S. Department of Transportation under Grant 69A3552348327 for the CARMEN+ University Transportation Center.

References

[1] Mamdooh Al-Saud, Ali M Eltamaly, Mohamed A Mohamed, and Abdollah Kavousi-Fard. An intelligent data-driven model to secure intravehicle communications based on machine learning. *IEEE Transactions on Industrial Electronics*, 67(6):5112–5119, 2019.

[2] Abdullatif Albaseer and Mohamed Abdallah. Fine-tuned lstm-based model for efficient honeypot-based network intrusion detection system in smart grid networks. In *2022 5th international conference on communications, signal processing, and their applications (ICCSPA)*, pages 1–6. IEEE, 2022.

[3] Emad Aliwa, Omer Rana, Charith Perera, and Peter Burnap. Cyberattacks and countermeasures for in-vehicle networks. *ACM computing surveys (CSUR)*, 54(1):1–37, 2021.

[4] Natasha Alkhatib, Maria Mushtaq, Hadi Ghauch, and Jean-Luc Danger. Can-bert do it? controller area network intrusion detection system based on bert language model. In *2022 IEEE/ACS 19th International Conference on Computer Systems and Applications (AICCSA)*, pages 1–8. IEEE, 2022.

[5] Afia Anjum, Paul Agbaje, Sena Hounsinou, and Habeeb Olufowobi. In-vehicle network anomaly detection using extreme gradient boosting machine. In *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–6. IEEE, 2022.

[6] Omid Avatefipour, Ameena Saad Al-Sumaiti, Ahmed M El-Sherbeeny, Emad Mahrous Awwad, Mohammed A Elmeligy, Mohamed A Mohamed, and Hafiz Malik. An intelligent secured framework for cyberattack detection in electric vehicles' can bus using machine learning. *Ieee Access*, 7:127580–127592, 2019.

[7] Omid Avatefipour and Hafiz Malik. State-of-the-art survey on in-vehicle network communication (can-bus) security and vulnerabilities. *arXiv preprint arXiv:1802.01725*, 2018.

[8] can ml. can-ml github. https://bitbucket.org/brooke-lampe/can-ml/src/master/, 2023. Accessed: 2026-04-06.

[9] Paul Carsten, Todd R Andel, Mark Yampolskiy, and Jeffrey T McDonald. In-vehicle networks: Attacks, vulnerabilities, and proposed solutions. In *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, pages 1–8, 2015.

[10] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX security symposium (USENIX Security 11)*, 2011.

[11] Kyong-Tak Cho and Kang G Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 911–927, 2016.

[12] Kyong-Tak Cho and Kang G Shin. Viden: Attacker identification on in-vehicle networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1109–1123, 2017.

[13] Wonsuk Choi, Kyungho Joo, Hyo Jin Jo, Moon Chan Park, and Dong Hoon Lee. Voltageids: Low-level communication characteristics for automotive intrusion detection system. *IEEE Transactions on Information Forensics and Security*, 13(8):2114–2129, 2018.

[14] Thomas Chowdhury, Eric Lesiuta, Kerianne Rikley, Chung-Wei Lin, Eunsuk Kang, BaekGyu Kim, Shinichi Shiraishi, Mark Lawford, and Alan Wassyng. Safe and secure automotive over-the-air updates. In *International Conference on Computer Safety, Reliability, and Security*, pages 172–187. Springer, 2018.

[15] Colorado State University System Cyber. Colorado state university j1939 attack data. https://projects-web.engr.colostate.edu/cybersystems/j1939-attacks/, 2022. Accessed: 2026-04-06.

[16] Araya Kibrom Desta, Shuji Ohira, Ismail Arai, and Kazutoshi Fujikawa. Rec-cnn: In-vehicle networks intrusion detection using convolutional neural networks trained on recurrence plots. *Vehicular Communications*, 35:100470, 2022.

[17] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Miki E. Real ornl automotive dynamometer (road) can intrusion dataset. https://0xsam.com/road/, 2022. Accessed: 2026-04-06.

[19] Abdallah R Gad, Ahmed A Nashat, and Tamer M Barkat. Intrusion detection system using machine learning for vehicular ad hoc networks based on ton-iot dataset. *IEEE Access*, 9:142206–142217, 2021.

[20] Mee Lan Han, Jin Lee, Ah Reum Kang, Sungwook Kang, Jung Kyu Park, and Huy Kang Kim. A statistical-based anomaly detection method for connected cars in internet of things environment. In *Internet of Vehicles-Safe and Intelligent Mobility: Second International Conference, IOV 2015, Chengdu, China, December 19-21, 2015, Proceedings 2*, pages 89–97. Springer, 2015.

[21] HCRL. Car-hacking dataset. https://ocslab.hksecurity.net/Datasets/car-hacking-dataset, 2018. Accessed: 2026-04-06.

[22] HCRL. In-vehicle network intrusion detection challenge. https://ocslab.hksecurity.net/Datasets/datachallenge2019/car, 2018. Accessed: 2026-04-06.

[23] HCRL. Survival analysis dataset for automobile ids. https://ocslab.hksecurity.net/Datasets/survival-ids, 2018. Accessed: 2026-04-06.

[24] HCRL. B-can intrusion dataset. https://ocslab.hksecurity.net/Datasets/b-can-intrusion-dataset, 2022. Accessed: 2026-04-06.

[25] HCRL. M-can intrusion dataset. https://ocslab.hksecurity.net/Datasets/m-can-intrusion-dataset, 2022. Accessed: 2026-04-06.

[26] Thien-Nu Hoang and Daehee Kim. Supervised contrastive resnet and transfer learning for the in-vehicle intrusion detection system. *Expert Systems with Applications*, 238:122181, 2024.

[27] Md Delwar Hossain, Hiroyuki Inoue, Hideya Ochiai, Doudou Fall, and Youki Kadobayashi. Lstm-based intrusion detection system for in-vehicle can bus communications. *Ieee Access*, 8:185489–185502, 2020.

[28] Rong Hu, Zhongying Wu, Yong Xu, and Taotao Lai. Multi-attack and multi-classification intrusion detection for vehicle-mounted networks based on mosaic-coded convolutional neural network. *Scientific Reports*, 12(1):6295, 2022.

[29] Abdul Rehman Javed, Saif Ur Rehman, Mohib Ullah Khan, Mamoun Alazab, and Thippa Reddy. Canintelliids: Detecting in-vehicle intrusion attacks on a controller area network using cnn and attention-based gru. *IEEE transactions on network science and engineering*, 8(2):1456–1466, 2021.

[30] Seonghoon Jeong, Minsoo Ryu, Hyunjae Kang, and Huy Kang Kim. Infotainment system matters: Understanding the impact and implications of in-vehicle infotainment system hacking with automotive grade linux. In *Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy*, pages 201–212, 2023.

[31] Yeonseon Jeong, Hyunghoon Kim, Seyoung Lee, Wonsuk Choi, Dong Hoon Lee, and Hyo Jin Jo. In-vehicle network intrusion detection system using can frame-aware features. *IEEE Transactions on Intelligent Transportation Systems*, 25(5):3843–3853, 2023.

[32] Jake Jepson, Rik Chatterjee, and Jeremy Daily. Commercial vehicle electronic logging device security: Unmasking the risk of truck-to-truck cyber worms. In *Symposium on Vehicles Security and Privacy (VehicleSec)*, volume 2024, pages 1–12, 2024.

[33] Xin Jin, Kexin Pei, Jun Yeon Won, and Zhiqiang Lin. Symlm: Predicting function names in stripped binaries via context-sensitive execution-aware code embeddings. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1631–1645, 2022.

[34] Min-Joo Kang and Je-Won Kang. Intrusion detection system using deep neural network for in-vehicle network security. *PloS one*, 11(6):e0155781, 2016.

[35] Tien-Dat Le, Hoang Bao Huy Truong, Daehee Kim, et al. Multi-classification in-vehicle intrusion detection system using packet-and sequence-level characteristics from time-embedded transformer with autoencoder. *Knowledge-Based Systems*, 299:112091, 2024.

[36] Jiajia Liu, Shubin Zhang, Wen Sun, and Yongpeng Shi. In-vehicle network attacks and countermeasures: Challenges and future directions. *IEEE Network*, 31(5):50–58, 2017.

[37] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015(S 91):1–91, 2015.

[38] Michael Müter and Naim Asaj. Entropy-based anomaly detection for in-vehicle networks. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 1110–1115. IEEE, 2011.

[39] Iqra Nazakat and Khurram Khurshid. Intrusion detection system for in-vehicular communication. In *2019 15th International Conference on Emerging Technologies (ICET)*, pages 1–6. IEEE, 2019.

[40] Trieu Phong Nguyen, Heungwoo Nam, and Daehee Kim. Transformer-based attention network for in-vehicle intrusion detection. *IEEE Access*, 11:55389–55403, 2023.

[41] Nvidia. End-to-end solutions for autonomous vehicles, 2025.

[42] Nvidia. Nvidia jetson xavier, 2025.

[43] Ebelechukwu Nwafor and Habeeb Olufowobi. Canbert: A language-based intrusion detection model for in-vehicle networks. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 294–299. IEEE, 2022.

[44] Qualcomm. A smoother automated drive, 2025.

[45] Hyun Min Song, Ha Rang Kim, and Huy Kang Kim. Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network. In *2016 international conference on information networking (ICOIN)*, pages 63–68. IEEE, 2016.

[46] Hyun Min Song, Jiyoung Woo, and Huy Kang Kim. In-vehicle network intrusion detection using deep convolutional neural network. *Vehicular Communications*, 21:100198, 2020.

[47] Dario Stabili, Mirco Marchetti, and Michele Colajanni. Detecting attacks to internal vehicle networks through hamming distance. In *2017 AEIT International Annual Conference*, pages 1–6. IEEE, 2017.

[48] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[49] Wilfried Voss. Can bus tutorial - can bus main characteristics, 2021.

[50] Haohuang Wen, Qi Alfred Chen, and Zhiqiang Lin. {Plug-N-Pwned}: Comprehensive vulnerability analysis of {OBD-II} dongles as a new {Over-the-Air} attack surface in automotive {IoT}. In *29th USENIX security symposium (USENIX Security 20)*, pages 949–965, 2020.

[51] Haohuang Wen and Zhiqiang Lin. Egg hunt in tesla infotainment: A first look at reverse engineering of qt binaries. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3997–4014, 2023.

[52] Haohuang Wen, Qingchuan Zhao, Qi Alfred Chen, and Zhiqiang Lin. Automated cross-platform reverse engineering of can bus commands from mobile apps. In *Proceedings 2020 Network and Distributed System Security Symposium (NDSS'20)*, 2020.

[53] Jun Yeon Won, Wenzhuo Wang, Keith Redmill, and Zhiqiang Lin. {CarPlay} at risk: Unveiling security threats of {Third-Party} infotainment adapters. In *3rd USENIX Symposium on Vehicle Security and Privacy (VehicleSec 25)*, pages 143–159, 2025.

[54] Wufei Wu, Yizhi Huang, Ryo Kurachi, Gang Zeng, Guoqi Xie, Renfa Li, and Keqin Li. Sliding window optimized information entropy analysis method for intrusion detection on in-vehicle networks. *Ieee Access*, 6:45233–45245, 2018.

[55] Lei Xue, Yangyang Liu, Tianqi Li, Kaifa Zhao, Jianfeng Li, Le Yu, Xiapu Luo, Yajin Zhou, and Guofei Gu. {SAID}: State-aware defense against injection attacks on in-vehicle network. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1921–1938, 2022.

[56] Li Yang, Abdallah Moubayed, Ismail Hamieh, and Abdallah Shami. Tree-based intelligent intrusion detection system in internet of vehicles. In *2019 IEEE global communications conference (GLOBECOM)*, pages 1–6. IEEE, 2019.

[57] Jiayan Zhang, Fei Li, Haoxi Zhang, Ruxiang Li, and Yalin Li. Intrusion detection system using deep learning for in-vehicle security. *Ad Hoc Networks*, 95:101974, 2019.

A Accuracy on the Rest Dataset

We describe the accuracy of the rest logs from datasets that do not contain malicious CAN messages in the test dataset.

	SVM	RF	XGB	Reg	LSTM	B1	B2	OUR
DoS	0.78	1.00	0.93	0.96	0.98	1.00	1.00	1.00
fuzzy	0.83	1.00	0.16	0.81	0.67	1.00	1.00	1.00
gear	0.89	1.00	0.99	0.98	0.98	0.99	1.00	1.00

Table 11: **Accuracy on Hacking Dataset.**

	SVM	RF	XGB	Reg	LSTM	B1	B2	OUR
fuzzy_1	1.00	1.00	0.95	0.99	1.00	1.00	1.00	1.00
speed_1	1.00	1.00	1.00	0.99	0.99	1.00	1.00	1.00
speed_2	0.99	1.00	1.00	0.99	0.99	1.00	0.97	1.00
speed_spoof_1	1.00	1.00	1.00	0.99	1.00	1.00	1.00	1.00
speed_spoof_2	0.99	1.00	1.00	0.99	0.99	0.77	0.98	1.00
light_off_3	0.97	1.00	1.00	0.97	0.98	1.00	0.99	1.00
light_off_spoof_3	0.97	1.00	0.99	0.97	0.96	1.00	1.00	1.00
light_on_1	0.99	1.00	1.00	0.99	0.99	0.99	1.00	1.00
light_on_2	0.98	1.00	1.00	0.97	0.99	0.99	0.99	1.00
light_on_3	0.99	1.00	1.00	0.99	0.99	1.00	1.00	1.00
light_on_spoof_1	0.99	1.00	1.00	0.99	0.99	1.00	0.99	1.00
light_on_spoof_2	0.97	1.00	1.00	0.97	0.97	0.85	0.99	1.00
light_on_spoof_3	0.99	1.00	1.00	0.99	1.00	1.00	1.00	1.00

Table 12: **Accuracy on Road Dataset.**

	SVM	RF	XGB	Reg	LSTM	B1	B2	OUR
Fuzzy Spark	0.94	1.00	0.06	0.94	0.93	0.07	0.99	1.00

Table 13: **Accuracy on Survival Dataset**

	SVM	RF	XGB	Reg	LSTM	B1	B2	OUR		SVM	RF	XGB	Reg	LSTM	B1	B2	OUR
forester																	
DoS-1	1.00	1.00	0.93	1.00	1.00	1.00	1.00	1.00	rpm-2	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00
DoS-2	1.00	1.00	0.90	0.99	1.00	0.98	1.00	0.99	rpm-3	0.99	1.00	0.99	0.99	0.99	1.00	1.00	1.00
DoS-3	1.00	1.00	0.93	1.00	1.00	1.00	1.00	1.00	rpm-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
DoS-4	0.99	1.00	1.00	0.99	0.99	1.00	1.00	1.00	rpm-accessory-1	0.99	1.00	1.00	0.99	0.99	1.00	1.00	1.00
combined-double-1	0.99	1.00	1.00	0.99	1.00	1.00	1.00	1.00	rpm-accessory-2	1.00	1.00	0.93	1.00	1.00	1.00	1.00	1.00
combined-double-2	0.99	0.99	0.99	0.99	0.98	1.00	0.99	1.00	rpm-accessory-3	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00
combined-double-3	0.99	0.99	0.99	0.99	0.98	1.00	0.99	1.00	rpm-accessory-4	1.00	1.00	0.96	0.99	0.99	1.00	1.00	1.00
combined-double-4	0.99	1.00	0.99	0.99	1.00	1.00	1.00	1.00	speed-3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
combined-triple-1	0.95	0.99	0.97	0.91	0.97	1.00	0.99	1.00	speed-4	0.99	1.00	0.99	1.00	0.99	1.00	1.00	1.00
combined-triple-3	1.00	0.99	0.99	1.00	0.99	1.00	0.99	1.00	speed-accessory-1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
combined-triple-4	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00	speed-accessory-2	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00
fuzzing-2	0.99	1.00	0.80	0.99	0.99	1.00	1.00	1.00	speed-accessory-3	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00
fuzzing-3	1.00	1.00	0.69	1.00	1.00	1.00	1.00	1.00	speed-accessory-4	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00
fuzzing-4	1.00	1.00	0.92	1.00	1.00	1.00	1.00	1.00	standstill-1	0.99	1.00	0.99	0.99	0.99	1.00	1.00	1.00
gear-1	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00	standstill-2	0.99	1.00	0.99	0.99	0.99	1.00	1.00	1.00
gear-2	1.00	1.00	1.00	0.99	1.00	1.00	1.00	1.00	standstill-3	1.00	1.00	0.62	0.99	1.00	1.00	1.00	1.00
gear-3	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00	standstill-4	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00
gear-4	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00	systematic-1	0.99	1.00	0.71	0.99	0.99	1.00	1.00	1.00
interval-3	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00	systematic-2	1.00	1.00	0.95	0.99	1.00	1.00	1.00	1.00
interval-4	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00	systematic-3	0.99	1.00	0.68	0.99	0.99	1.00	1.00	1.00
									systematic-4	0.99	1.00	0.85	0.99	0.99	1.00	1.00	1.00
impala																	
DoS-1	1.00	1.00	0.66	1.00	1.00	1.00	1.00	1.00	rpm-2	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00
DoS-2	0.99	1.00	0.99	0.99	0.99	1.00	1.00	1.00	rpm-3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
DoS-3	0.99	1.00	0.99	0.98	0.99	1.00	1.00	1.00	rpm-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
DoS-4	0.99	1.00	0.96	0.99	0.99	1.00	1.00	1.00	rpm-accessory-1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
combined-double-1	0.97	1.00	1.00	0.99	0.99	1.00	1.00	1.00	rpm-accessory-2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
combined-double-2	1.00	0.99	1.00	1.00	0.99	1.00	0.99	1.00	rpm-accessory-3	1.00	1.00	1.00	0.99	1.00	1.00	1.00	1.00
combined-double-3	1.00	0.99	0.95	1.00	0.99	1.00	0.99	1.00	rpm-accessory-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
combined-double-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	speed-1	0.99	1.00	1.00	0.99	1.00	1.00	1.00	1.00
combined-triple-1	0.99	0.99	1.00	0.99	0.98	1.00	0.99	1.00	speed-2	0.99	1.00	0.99	0.99	1.00	1.00	1.00	1.00
combined-triple-2	1.00	1.00	1.00	0.99	0.99	1.00	0.99	1.00	speed-3	0.99	1.00	1.00	0.99	0.99	1.00	1.00	1.00
combined-triple-3	1.00	1.00	0.99	1.00	1.00	1.00	0.99	1.00	speed-4	0.99	1.00	1.00	0.99	0.99	1.00	1.00	1.00
combined-triple-4	1.00	1.00	0.99	1.00	0.99	1.00	0.99	1.00	speed-accessory-1	0.99	0.99	0.79	0.99	0.99	1.00	0.99	1.00
fuzzing-2	0.99	1.00	0.67	0.99	0.99	1.00	1.00	1.00	speed-accessory-2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
fuzzing-3	0.99	1.00	0.59	0.99	0.99	1.00	1.00	1.00	speed-accessory-3	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00
fuzzing-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	speed-accessory-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
gear-1	0.98	1.00	1.00	0.99	0.99	1.00	1.00	1.00	standstill-1	0.99	1.00	0.93	0.99	1.00	1.00	1.00	1.00
gear-2	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00	standstill-2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
gear-3	1.00	1.00	0.97	1.00	0.99	1.00	0.99	1.00	standstill-3	0.99	1.00	1.00	0.99	1.00	1.00	1.00	1.00
gear-4	1.00	1.00	0.99	1.00	0.99	1.00	1.00	1.00	standstill-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
interval-1	0.99	1.00	0.99	0.99	1.00	1.00	1.00	1.00	systematic-2	1.00	1.00	0.91	1.00	1.00	1.00	0.99	1.00
interval-4	1.00	1.00	0.99	1.00	0.99	1.00	1.00	1.00	systematic-3	1.00	1.00	0.86	1.00	1.00	1.00	1.00	1.00
rpm-1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	systematic-4	1.00	1.00	0.98	1.00	1.00	1.00	1.00	1.00

Table 14: **Accuracy on CanML Dataset with Forester and Impala**

	SVM	RF	XGB	Reg	LSTM	B1	B2	OUR		SVM	RF	XGB	Reg	LSTM	B1	B2	OUR
silverado																	
DoS-1	0.92	1.00	0.65	0.98	0.94	1.00	1.00	0.99	rpm-2	1.00	0.99	0.76	1.00	0.99	1.00	0.99	1.00
DoS-2	0.94	1.00	0.93	0.99	0.99	1.00	1.00	1.00	rpm-3	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00
DoS-3	0.96	1.00	0.99	0.98	0.98	1.00	1.00	1.00	rpm-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
DoS-4	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00	rpm-accessory-1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
combined-double-1	0.99	0.99	0.98	0.99	0.99	1.00	0.99	1.00	rpm-accessory-2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
combined-double-2	0.99	0.99	0.99	0.99	0.99	1.00	1.00	1.00	rpm-accessory-3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
combined-double-3	0.99	1.00	0.99	0.99	0.98	1.00	1.00	1.00	rpm-accessory-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
combined-double-4	1.00	1.00	0.99	1.00	0.99	1.00	1.00	1.00	speed-1	1.00	1.00	0.80	0.99	1.00	1.00	1.00	1.00
combined-triple-1	0.98	0.99	0.98	0.98	0.99	0.01	0.99	0.98	speed-3	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00
combined-triple-3	1.00	0.99	0.99	1.00	0.99	1.00	0.99	1.00	speed-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
combined-triple-4	1.00	1.00	0.99	1.00	0.99	1.00	1.00	1.00	speed-accessory-3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
fuzzing-1	0.99	1.00	0.63	0.98	0.95	1.00	0.99	1.00	speed-accessory-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
fuzzing-2	1.00	1.00	0.98	1.00	0.99	1.00	1.00	1.00	standstill-1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
fuzzing-3	0.99	1.00	0.77	0.99	0.99	1.00	1.00	1.00	standstill-2	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00
fuzzing-4	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00	standstill-4	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00
gear-2	1.00	1.00	1.00	1.00	0.99	1.00	0.99	1.00	systematic-1	1.00	1.00	0.85	1.00	1.00	1.00	0.99	1.00
gear-3	1.00	1.00	1.00	1.00	1.00	1.00	0.98	1.00	systematic-2	1.00	1.00	0.93	0.99	1.00	1.00	0.99	1.00
gear-4	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	systematic-4	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00
interval-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00									
traverse																	
DoS-1	0.99	1.00	0.81	0.99	0.99	1.00	1.00	1.00	rpm-1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
DoS-2	1.00	1.00	0.99	0.99	0.99	1.00	1.00	1.00	rpm-2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
DoS-3	0.99	1.00	1.00	0.99	0.99	1.00	1.00	1.00	rpm-3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
DoS-4	1.00	1.00	0.99	0.99	1.00	1.00	1.00	1.00	rpm-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
combined-double-1	0.99	1.00	0.99	0.99	0.99	1.00	1.00	1.00	rpm-accessory-1	1.00	1.00	0.96	1.00	1.00	1.00	1.00	1.00
combined-double-2	0.99	1.00	0.99	0.99	0.99	0.97	1.00	1.00	rpm-accessory-2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
combined-double-3	1.00	1.00	0.99	1.00	0.99	1.00	1.00	1.00	rpm-accessory-3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
combined-double-4	1.00	1.00	0.99	1.00	0.99	1.00	1.00	1.00	rpm-accessory-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
combined-triple-1	0.97	0.99	1.00	0.97	0.99	1.00	0.99	1.00	speed-1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
combined-triple-2	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00	speed-2	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00
combined-triple-3	1.00	0.99	0.99	1.00	1.00	1.00	0.99	1.00	speed-3	1.00	1.00	0.99	0.99	1.00	1.00	1.00	1.00
combined-triple-4	1.00	1.00	1.00	0.99	0.99	1.00	1.00	1.00	speed-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
fuzzing-1	0.95	0.99	0.42	0.95	0.93	1.00	1.00	1.00	speed-accessory-1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
fuzzing-2	0.99	0.99	0.73	0.99	0.99	1.00	1.00	1.00	speed-accessory-2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
fuzzing-3	0.99	0.99	0.58	0.99	0.98	1.00	1.00	1.00	speed-accessory-3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
fuzzing-4	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00	speed-accessory-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
gear-1	1.00	1.00	0.99	1.00	0.99	1.00	0.99	1.00	standstill-1	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00
gear-3	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	standstill-3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
gear-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	standstill-4	0.98	0.99	1.00	1.00	1.00	1.00	1.00	1.00
interval-2	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00	systematic-1	0.96	0.99	0.46	0.96	0.85	1.00	1.00	1.00
interval-3	0.99	1.00	0.99	1.00	0.99	1.00	0.99	1.00	systematic-2	1.00	1.00	0.76	0.99	0.99	1.00	1.00	1.00
interval-4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	systematic-3	1.00	1.00	0.97	1.00	1.00	1.00	1.00	1.00
									systematic-4	0.99	0.97	0.89	0.99	0.99	1.00	0.99	1.00

Table 15: **Accuracy on CanML Dataset with Silverado and Traverse**

B Artifact Appendix

B.1 Abstract

We propose a BERT-based in-vehicle intrusion detection system named CANBERT+. It utilizes a data differentiation process and multi-layer input embeddings to improve and optimize performance. The artifact contains the source code and datasets required to run CANBERT+. The objective of the artifact is to release the code of CANBERT+ to contribute to open science. It includes one mid-sized dataset (mcan), which best demonstrates the system's detection performance and overhead. The corresponding result is shown in Figure 4(c) of our paper.

B.2 Description and Requirements

- **Security, privacy, and ethical concerns:** All data used in this artifact is publicly available and does not contain any private information, raising no ethical concerns. The data distributors confirm that it was collected without any violations.
- **Access to the artifact:** `https://zenodo.org/records/20380113`
- **Required badges:** Available, Functional, and Reproduced
- **Hardware dependencies:** A GPU is recommended to run CANBERT+. We used an NVIDIA GTX 1080 Ti for our evaluation. The GPU may require more than 8GB of memory. Our machine is equipped with 64GB RAM and an Intel Xeon CPU E5-1650 v4 at 3.60GHz. However, this specific hardware configuration is not required to run CANBERT+.
- **Software dependencies:** CANBERT+ runs on Ubuntu 24.04 with Python 3.12. All required Python library dependencies are listed in the included `requirements.txt` file.
- **Benchmarks:** We evaluate CANBERT+ using publicly available datasets. Since the datasets were collected by third parties, we provide the original source reference for each dataset used.

B.3 Set-up

Download the `CanBERTplus.zip` file from the provided link and extract it. The following steps set up the required environment.

1. conda create -n canbert python=3.12
2. conda activate canbert
3. pip install -r requirements.txt

Steps 1 and 2 are optional if you prefer to use an existing Python environment. Step 3 is required to install all necessary dependencies before running CANBERT+.

B.4 Evaluation workflow

- **Major claims:** The source code implements our two core design contributions. First, the data differentiation process is implemented in `model/CanTokenizer.py`. Second, the multi-layer input embeddings are implemented in `model/bert.py`. For validation, we include the mid-sized mcan dataset. The results should be comparable to Figure 4(c) in the paper.
- **Experiments:** There are three steps to conduct experiments. The first step is to prepare the input tokens. Executing the following command will generate required tokens from every dataset.
  - python gen_vocab.py -i /path/to/data/

  The second step is to execute the pre-training process.
  - python can_ids.py -i /path/to/data/orig.csv

  It will generate `orig_ids.ckpt` file.

  Then, the last step is to fine-tune the model.
  - python can_finetune.py -i /path/to/data/mcan/g80_mcan_fuzzing_data.csv
  - python can_finetune.py -i /path/to/data/mcan/g80_mcan_ddos_data.csv

  It is possible to run experiments on all datasets. However, this may take more than a few weeks due to the size of the full dataset collection. In our recommended setup, the first step takes a few minutes and the second step takes approximately 3 hours. The last step may also take a few hours. Note that running without a GPU will significantly increase the execution time.

  The configurations for small and tiny variants of CANBERT+ are defined in `canlm.py`. However, changing the model configuration requires re-running the pre-training step from scratch.

  Due to the inherent randomness in machine learning training, results may differ slightly from those reported in the paper. To improve reproducibility, you may increase the patience of the early stopping criterion, or use the test set as a validation set to monitor model performance at each epoch.
- **Pre-trained weights.** This repository contains the experimental weights presented in the main body of the paper. To reproduce the results, first update the data and checkpoint paths in the can_test.py script. Next, move orig.ckpt, can_test.py, and run.sh into the original working directory. Finally, execute the run.sh script to calculate the F1 score for each experiment.

B.5 Notes on Reusability

This artifact can be reused as a baseline for comparison with other in-vehicle intrusion detection systems. Moreover, the source code is structured to allow straightforward modification for performance improvements or adaptation to other CAN bus datasets.



MIRAGE: Detecting Fake Emergency Electronic Brake Light Attacks in V2X Networks via Event-Gated Behavioral Analysis

Eunhan Ka
Purdue University

Doguhan Yeke
Purdue University

Z. Berkay Celik
Purdue University

Satish V. Ukkusuri
Purdue University

Abstract

The emergency electronic brake light (EEBL) application in Vehicle-to-Everything (V2X) networks warns upstream vehicles of hard braking. However, recent studies show that a compromised vehicle with valid credentials can broadcast fabricated EEBL messages, triggering unnecessary emergency braking and creating significant traffic safety risks. To detect such attacks, existing misbehavior detection systems apply general-purpose plausibility checks to every basic safety message, resulting in false alarms during normal driving and missed detections when the attacker generates a kinematically self-consistent trajectory. To address this problem, we propose MIRAGE (Multi-stage Inspection and Response Against Ghost-vehicle EEBL), an event-gated detector grounded in the Intelligent Driver Model (IDM). MIRAGE activates only under EEBL conditions, validates post-event trajectory consistency using position-derived speed, detects post-attack position freezing artifacts, and bounds the ego vehicle's response via IDM even on missed detections. We implement MIRAGE within the V2X Application Spoofing Platform (VASP) and evaluate against two fake EEBL attack variants alongside three existing defenses. MIRAGE achieves an F1 score of 0.922 across five random seeds with 88.3% detection rate at 0.9% false positive rate, and is the only detector that maintains high detection across both attack variants. We also show that MIRAGE reduces mean braking severity by 75%.

1 Introduction

Vehicle-to-Everything (V2X) communication enables vehicles to exchange real-time kinematic data via Basic Safety Messages (BSMs) at 10 Hz [22], supporting cooperative safety applications that extend awareness beyond the on-board sensor range [16]. Among these, the Emergency Electronic Brake Light (EEBL) is one of the first applications to be standardized for deployment: it notifies upstream vehicles of hard braking ahead, enabling earlier reaction when a direct line of sight is blocked [11, 35]. Given that rear-end crashes account for a large fraction of multi-vehicle incidents [16], EEBL is recognized as one of the highest-impact V2X safety applications and is widely included as a Day-1 service in C-ITS deployment studies [35].

Every BSM is digitally signed under the vehicular Public Key Infrastructure (PKI), commonly the Security Credential Management System (SCMS), [49], which authenticates the sender as a legitimately enrolled vehicle [14, 19, 33], preventing external devices from injecting messages into the V2X channel. However, the PKI verifies only sender identity, not the contextual correctness of message content, so it cannot prevent an enrolled device from reporting fabricated position, speed, or brake status [7]. Independent verification is infeasible precisely because EEBL targets non-line-of-sight scenarios where receivers cannot directly observe the sender [37]. Prior work confirms the feasibility of these vectors through full remote attack chains on production vehicles [30], link-layer Controller Area Network (CAN) bus control from compromised ECUs [42], and hardware key extraction via voltage glitching [48]. Similarly, an industry report recorded 409 automotive cyber incidents in 2024, with 92% executed remotely [44]. An attacker who compromises a legitimate on-board unit (OBU) can pass PKI authentication using valid credentials and issue arbitrary BSM content under an apparently valid identity [49]. The attacker can forge brake flags, deceleration, and position fields to fabricate an EEBL event [5, 25]. The V2X Application Spoofing Platform (VASP) [5] rates the two EEBL-targeting attack variants implemented in its framework as high-risk attacks overall. In a related collaborative vehicular perception setting, Zhang et al. [52] demonstrated that crafted data can induce hard braking or increase collision risk, achieving over 86% success in high-fidelity simulation, with feasibility shown in real-world experiments. Together, these studies show that authenticated data-fabrication attacks are practical enough to require application-level validation beyond cryptographic authentication.

Existing Misbehavior Detection Systems (MDS) address content-level attacks through plausibility checks on BSM fields [7, 45], including range-and-threshold checks [6, 24],

cross-field physics validation [3, 36], and machine-learning (ML) classifiers trained on misbehavior datasets [15, 20, 21, 38, 46]. However, when applied to fake EEBL, these methods face two problems. First, always-on detectors, such as threshold checks [24] and F2MD [20], evaluate every BSM regardless of the application context. Sensor noise during normal car-following triggers false alarms [39], and at the application level, a false positive that suppresses a genuine EEBL warning may be more dangerous than the attack itself. Second, cross-field validators such as VCADS [3] detect inconsistencies across reported fields. However, VASP's FakeEEBL_WithStop attack [5] produces a kinematically self-consistent stopping trajectory. This evasion strategy is well-established: the SixPack attack [32] and its successor [53] generate consistent fake braking trajectories across all six BSM kinematic fields to bypass cross-field validators.

We address these limitations with MIRAGE (Multi-stage Inspection and Response Against Ghost-vehicle EEBL), grounded in the Intelligent Driver Model (IDM). The detector activates only when EEBL trigger conditions are met, suppressing false alarms during routine driving. Upon activation, four stages evaluate contextual plausibility (Stage 1), deceleration consistency against physical and kinematic constraints (Stage 2), trajectory evidence through position-derived speed (Stage 3), and post-stop position consistency to catch frozen-position artifacts from artifact-bearing attack variants (Stage 4). The key insight is that position-derived speed, computed from consecutive position reports, serves as an independent kinematic witness that an attacker can suppress only by falsifying the position field. Failed checks accumulate into a weighted anomaly score for threshold-based detection, while an IDM-based mitigation layer bounds the ego vehicle's braking response to limit the impact of missed detections. We evaluated MIRAGE within the VASP framework [5] against both attack variants and three baselines [3, 5, 20, 24]. MIRAGE achieves an F1 of 0.922 (true positive rate (TPR) = 0.883, false positive rate (FPR) = 0.009) and is the only detector maintaining high rates across both types.

In summary, we make the following contributions:

- We present an *integration of event-gated, multi-stage behavioral inspection with IDM-based response bounding* for fake EEBL attack defense. The contribution is this integration rather than any single check in isolation. The detector reuses established ingredients (plausibility checks, position-derived speed, post-event trajectory analysis), but couples them to an application-specific event gate that activates only under EEBL conditions and to a controller-level response layer that bounds the ego vehicle's deceleration even when detection fails.
- We design the *event-gated, multi-stage detector* with four verification stages: contextual relevance of the EEBL claim (Stage 1), kinematic plausibility of the reported deceleration (Stage 2), post-event trajectory consistency through position-derived speed (Stage 3), and post-stop position consistency (Stage 4). The architecture treats Stages 1–3 as defense-in-depth against kinematically inconsistent attacks, while Stage 4 is a targeted check against attack variants that produce kinematically self-consistent BSMs during the braking phase.
- We implement and evaluate MIRAGE within the *VASP framework* [5], benchmarking against three baselines under two fake EEBL attack variants. MIRAGE achieves an F1 of 0.922 with 77.0% $\mathcal{A}_1$ and 99.7% $\mathcal{A}_2$ detection rates; the $\mathcal{A}_2$ result reflects VASP's current FakeEEBL_WithStop implementation and is discussed as a primary limitation in Section 8.3. The IDM bounding layer reduces peak braking severity by 50% compared to unprotected emergency braking, limiting deceleration to -4.0 m/s^2 even for undetected attack BSMs.

MIRAGE is publicly available, including the code, SUMO scenario files, Veins configurations, analysis scripts, and random seeds for reproducibility, and is archived at `https://doi.org/10.5281/zenodo.19639252` (see Section 10).

2 Background

2.1 EEBL Application

Emergency Electronic Brake Light (EEBL) is standardized in SAE J2945/1 [35] and is widely listed as a Day-1 C-ITS safety service. When a vehicle decelerates beyond a hard-brake threshold (SAE J2945 specifies $\geq 0.4g \approx 3.92$ m/s^2; VASP's implementation uses 4.5 m/s^2), it sets the `eventHardBraking` flag in its BSM [35]. A receiver triggers an EEBL warning if three conditions hold: (1) the sender's brake flag is set, (2) the sender is longitudinally ahead and in the same or adjacent lane, and (3) the sender is within a safety-relevant distance [5, 18]. This protocol extends hazard awareness beyond direct line of sight, enabling earlier braking reactions and helping prevent chain-reaction rear-end collisions [40]. As receivers verify only the sender's PKI enrollment rather than the physical existence of the claimed vehicle [25, 49], EEBL is vulnerable to ghost-node attacks in which an adversary creates a virtual entity that broadcasts fabricated hard-brake BSMs from a position where no real vehicle exists. The VASP platform [5] implements two types of EEBL attacks, formalized as $\mathcal{A}_1$ and $\mathcal{A}_2$:

- $\mathcal{A}_1$ **(FakeEEBL_NoStop):** A ghost vehicle j^* appears ahead of victim i (offset $d_0 = 60$ m), sets $\beta_{j^*} = 1$, and reports $a_{j^*} = -5$ m/s^2. The reported speed decreases to half the claimed rate while the position remains at constant velocity, so the position field implies that the ghost continues moving rather than decelerating as claimed.
- $\mathcal{A}_2$ **(FakeEEBL_WithStop):** A ghost vehicle appears ahead ($d_0 = 55$ m) and decelerates at $a_{j^*} = -3.5$ m/s^2 to

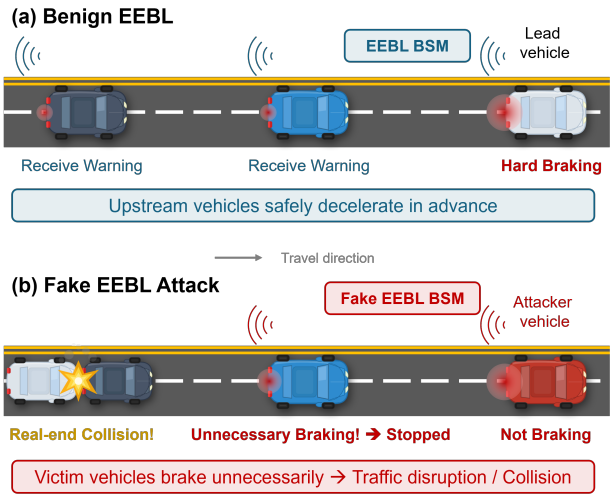


Figure 1: Comparison between legitimate EEBL protocol operation (top) and a fake EEBL attack (bottom).

a full stop, with all BSM fields kept mutually consistent so the trajectory is kinematically self-consistent and is the hardest case for BSM-only detection [32].

Figure 1 illustrates the contrast between benign EEBL operation and a fake EEBL attack. In the benign case (top), the leading vehicle undergoes genuine hard braking and broadcasts an EEBL BSM; upstream vehicles receive the warning and decelerate safely in advance, preventing a chain-reaction collision. In the attack case (bottom), an attacker broadcasts a fabricated hard-brake BSM while continuing at constant speed. The victim vehicle brakes unnecessarily and comes to a stop, while a following vehicle that cannot react in time collides with it, demonstrating that a single fake EEBL can cascade into real physical harm.

The ego vehicle runs an on-board MDS that processes incoming BSMs in real time. The defender has access only to BSM content (no radar/lidar confirmation in the non-line-of-sight (NLOS) scenario), must prioritize specificity since suppressing a genuine EEBL could cause a collision [45], and must process each BSM within one BSM interval ($\Delta t = 0.1$ s); the time from the first attack BSM to first detection is a separate quantity reported in Section 8.

2.2 Intelligent Driver Model (IDM)

The Intelligent Driver Model (IDM) [43] is a continuous-time car-following model that computes the acceleration of a following vehicle i as a function of its current speed, the gap to the leader, and the closing speed:

$$a_{\text{IDM}} = a_{\max}\left[1 - \left(\frac{v_i}{v_0}\right)^{\delta} - \left(\frac{s^*(v_i, \Delta v_{ij})}{s_{ij}}\right)^2\right], \qquad (1)$$

where v_0 is the desired speed, $a_{\max}$ the maximum acceleration, δ the acceleration exponent (typically 4), s_{ij} the current gap to leader j, $\Delta v_{ij} = v_i - v_j$ the closing speed, and $s^*(v, \Delta v) = s_0 + vT + \frac{v\Delta v}{2\sqrt{a_{\max} b}}$ is the desired minimum gap with jam distance s_0, safe time headway T, and comfortable deceleration b.

The IDM has two properties that make it suitable for MIRAGE. First, its output is *gap-dependent*: the deceleration scales with the current gap s_{ij} and closing speed Δv_{ij}, so the ego's response naturally adapts to the actual traffic situation rather than applying a fixed emergency deceleration. When a claimed braking event is far away or the closing speed is low, the IDM prescribes mild deceleration; only when the gap is genuinely small does the output approach the comfortable deceleration limit b. Second, the model has been extensively calibrated against empirical traffic data and is widely adopted in traffic simulation [23, 43], ensuring that the mitigation layer's behavior is consistent with real driving dynamics. We use the IDM in two roles: (1) to generate realistic platoon dynamics in the Simulation of Urban MObility (SUMO) simulation [23, 26], and (2) to bound the ego vehicle's response to a claimed EEBL event (Section 5.7), with calibration parameters detailed in Section 6.2.

2.3 Time-to-Collision (TTC)

Time-to-Collision (TTC) [17] is a surrogate safety metric widely used in collision warning design [18] that estimates the time remaining before vehicle i would reach the position of leader j at the current closing speed:

$$\text{TTC}_{ij}^{(t)} = \begin{cases} \dfrac{s_{ij}^{(t)}}{\Delta v_{ij}^{(t)}}, & \text{if } \Delta v_{ij}^{(t)} > 0 \\ \infty, & \text{otherwise.} \end{cases} \qquad (2)$$

TTC plays two roles in MIRAGE. First, it serves as a *context filter*: a genuine hard-brake event produces low TTC for nearby followers, whereas a ghost vehicle placed far ahead or receding from the ego yields high or undefined TTC, indicating that the claimed emergency lacks physical urgency. Second, the sign of the closing speed conveys directionality: $\Delta v_{ij} \leq 0$ indicates that the ego is not closing on the sender, so the EEBL has no safety relevance regardless of whether it is genuine. We exploit both properties in Check 1a (Section 5.2).

3 Threat Model

We follow the same threat model as in the V2X security literature [7, 28, 37], adapted to fake EEBL attacks.

The adversary aims to trigger false EEBL warnings that cause unwarranted hard-braking responses from victim vehicles. A successful attack degrades safety by inducing sudden deceleration and increasing rear-end collision risk, disrupts

traffic by propagating braking shockwaves through the platoon [2], and reduces trust in V2X warnings if false events occur repeatedly [31].

We consider an internal, authenticated adversary consistent with the taxonomy in prior work [7, 28]. The attacker holds legitimate V2X certificates obtained through a compromised OBU [9, 27], supply-chain compromise during device provisioning [31], or insider access at fleet operators [37]. Firmware-level exploits have been demonstrated on production vehicles, from remote wireless-to-CAN-bus attack chains [30] to link-layer message manipulation by software-compromised ECUs [42]. With this level of access, the attacker can set attacker-chosen values in BSM fields, including position $\mathbf{x}_j$, speed v_j, acceleration a_j, and brake flag β_j [5]. The attacker can inject a ghost vehicle, a virtual entity broadcasting BSMs from a position where no physical vehicle exists [14, 25]. The attacker operates within a typical Dedicated Short-Range Communications (DSRC) communication range ($\leq$ 300 m [22]) and broadcasts at the standard 10 Hz rate. Sybil attacks, RF jamming, and combined multi-vector attacks are outside the scope and have been addressed by a separate line of V2X security work: foundational eviction and revocation mechanisms [34] were subsequently formalized into the Security Credential Management System (SCMS) [49] and have been revisited in recent V2X cybersecurity surveys [37], which document the gap that content-level misbehavior detection (the focus of this paper) is designed to fill.

The insider threat described above is not hypothetical. The VASP threat analysis rates the two EEBL-targeting attack variants implemented in its framework as *high risk* overall [5]. Data fabrication attacks on collaborative vehicular perception have achieved over 86% success in high-fidelity simulation, can induce hard braking or increased collision risk, and have been shown feasible in real-world experiments [52]. Meanwhile, misbehavior detection standards do not yet cover all V2X services [8, 29]: vehicular PKI authenticates senders but does not validate message content [49], as receivers lack an independent channel to verify the sender's physical state in the non-line-of-sight conditions where EEBL operates [37]. An insider with valid certificates can pass all cryptographic checks yet still fabricate EEBL events. A gap remains between deployed authentication and content-level validation.

The attacker is aware of the BSM format and EEBL trigger conditions per SAE J2735/J2945 [35], common MDS approaches such as range checks and speed/acceleration bounds [20, 24], and the victim's approximate position and speed from overheard BSMs. This knowledge enables the attacker to craft attacks that remain within plausibility bounds and to place the ghost vehicle at a realistic distance ahead. In this paper, the attacker conducts two primary attacks, $\mathcal{A}_1$ (FakeEEBL_NoStop) and $\mathcal{A}_2$ (FakeEEBL_WithStop), introduced in the VASP framework [5].

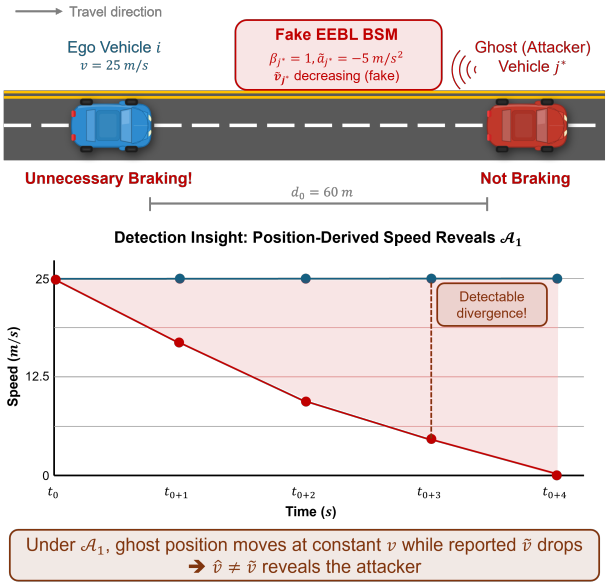


Figure 2: Motivating example under $\mathcal{A}_1$. Position-derived speed $\hat{v}_{j^*}$ remains near 25 m/s while reported $\tilde{v}_{j^*}$ drops to zero, revealing a detectable divergence.

4 Motivating Example

We illustrate the attack and the detection challenge through a concrete real-world scenario (shown in Figure 2). Ego vehicle i travels at $v_i^{(t_0)} = 25$ m/s. At time t_0, a ghost vehicle j^* begins broadcasting BSMs claiming a position 60 m ahead, with $\beta_{j^*} = 1$ and $\tilde{a}_{j^*} = -5$ m/s^2. Vehicle i has no prior BSM history for j^*. All EEBL trigger conditions are met. A threshold detector (B1 in Section 6.3) verifies $|\tilde{a}_{j^*}| < 11$ m/s^2 and $\tilde{v}_{j^*} < 70$ m/s. Since the manipulated values remain within normal ranges, the detector does not flag the BSM. A cross-field detector works better under $\mathcal{A}_1$: it computes $\hat{a}_{j^*}^{(t)} = (\tilde{v}_{j^*}^{(t)} - \tilde{v}_{j^*}^{(t-1)})/\Delta t$ and finds $\hat{a}_{j^*} \approx -2.5$ vs. $\tilde{a}_{j^*} = -5$ m/s^2, yielding a discrepancy of 2.5 m/s^2 that meets the detection threshold τ_a. However, under $\mathcal{A}_2$, all reported fields satisfy kinematic equations, and the detector finds nothing anomalous.

The key observation is that *position-derived speed* $\hat{v}_{j^*}^{(t)} = |\mathbf{x}_{j^*}^{(t)} - \mathbf{x}_{j^*}^{(t-1)}|/\Delta t$, computed from consecutive position reports, reveals the attacker under $\mathcal{A}_1$. Since the ghost's position continues at constant velocity even as $\tilde{v}_{j^*}$ drops, $\hat{v}_{j^*}$ remains near 25 m/s while $\tilde{v}_{j^*}$ approaches zero. The attacker cannot suppress this divergence without faking a consistent stopping trajectory, which requires $\mathcal{A}_2$-level sophistication. We note that this check derives speed solely from the position field, bypassing the reported speed and acceleration fields. It is robust to manipulation of speed and acceleration in particular. However, since position is itself a reported BSM field, an attacker who controls position evolution consistently (as in $\mathcal{A}_2$) can defeat this check during the braking phase. We address this

through a post-stop position-consistency check (Stage 4) that detects frozen-position artifacts after the attack completes, complemented by an IDM mitigation layer that bounds the ego's response regardless of the detection outcome.

Finally, gating these checks to EEBL-active periods avoids false alarms during routine driving, where transient sensor noise would trigger cross-field checks on every BSM.

5 MIRAGE: Multi-stage Inspection and Response Against Ghost-vehicle EEBL

5.1 Overview

Figure 3 illustrates the end-to-end detection pipeline of MIRAGE. Unlike always-on MDS [20, 24, 45], MIRAGE performs deep inspection only when an incoming BSM from sender j satisfies two conditions. First, $\beta_j^{(t)} = 1$ and the sender is within communication range, $|\mathbf{x}_j^{(t)} - \mathbf{x}_i^{(t)}| \le R_{\max} = 300$ m (regardless of relative direction; directional context is assessed in Stage 1). Second, at least two consecutive BSMs from the same sender carry the brake flag (the *consecutive-brake requirement*). The consecutive-brake filter ensures that a single transient brake flag, which may occur during normal car-following deceleration in SUMO, does not open the gate. We call this the **EEBL gate**. The gate remains open for up to $T_{\text{gate}} = 5$ s after the initial trigger, allowing post-event checks to accumulate evidence; the spatial and directional context of the EEBL event is then assessed in Stage 1. When the gate is open, four stages of evidence are evaluated. Each failed check k contributes a calibrated weight w_k to a cumulative anomaly score $S_j^{(t)}$. A BSM is flagged as malicious when $S_j^{(t)} \ge \Theta$. Table 1 summarizes the notation used throughout the paper.

5.2 Stage 1: Context Assessment

When a BSM passes the EEBL gate, the detector evaluates if the traffic context is consistent with a genuine emergency.

Check 1a (TTC plausibility). A genuine EEBL event typically involves low TTC for nearby followers. Van der Horst [47] identified a TTC of 4 s as the boundary between comfortable and uncomfortable car-following situations, a value widely adopted in collision warning design. We use $T_{\text{warn}} = 4$ s and flag when TTC exceeds twice this value:

$$\text{TTC}_{ij}^{(t)} > 8 \text{ s} \implies \text{flag} \quad (w_{1a} = 0.15). \tag{3}$$

If the ego is more than 8 seconds from reaching the claimed braking vehicle, the EEBL context lacks urgency. A stronger variant (Check 1a′) flags when the sender is not longitudinally ahead of the ego *or* when $\text{TTC}_{ij}^{(t)} < 0$ (i.e., the sender is moving faster), indicating the EEBL has no safety relevance ($w_{1a'} = 0.35$). Checks 1a and 1a′ are mutually exclusive.

Table 1: Summary of notation.

Symbol	Description
i	Ego (detecting) vehicle
j, j^*	Sender vehicle; attacker / ghost vehicle
t, Δt	Discrete time step, BSM interval (0.1 s)
$\mathbf{x}_j^{(t)}$, $v_j^{(t)}$, $a_j^{(t)}$	Position, speed, acceleration of j at t
$\tilde{v}_j^{(t)}$, $\tilde{a}_j^{(t)}$	Reported speed and acceleration in BSM
$\hat{v}_j^{(t)}$	Position-derived speed: $\lvert\mathbf{x}_j^{(t)} - \mathbf{x}_j^{(t-1)}\rvert / \Delta t$
$\hat{a}_j^{(t)}$	Speed-derived accel.: $(\tilde{v}_j^{(t)} - \tilde{v}_j^{(t-1)}) / \Delta t$
$\beta_j^{(t)}$	Hard-brake flag ($\beta_j = 1$: braking event)
$s_{ij}^{(t)}$	Gap between i and j: $\lvert\mathbf{x}_j^{(t)} - \mathbf{x}_i^{(t)}\rvert$
$\Delta v_{ij}^{(t)}$	Closing speed: $v_i^{(t)} - v_j^{(t)}$
$\text{TTC}_{ij}^{(t)}$	Time-to-collision (Eq. 2)
$\mathcal{H}_j$	BSM history buffer for sender j
$S_j^{(t)}$	Anomaly score for j at time t
w_k, Θ	Weight of check k; detection threshold
T_{gate}	EEBL gate timeout (5 s)
$\mathcal{A}_1$, $\mathcal{A}_2$	FakeEEBL_NoStop, FakeEEBL_WithStop

Check 1b (New-sender detection). A ghost vehicle that appears and claims hard braking has minimal BSM history. We flag if the number of previously received BSMs satisfies:

$$|\mathcal{H}_j| \le N_{\min} = 3. \tag{4}$$

In normal traffic, a vehicle is observed for several seconds before a braking event [41]. A sender with ≤ 3 prior BSMs (≤ 0.3 s of history at 10 Hz) that immediately triggers EEBL is anomalous ($w_{1b} = 0.20$). We note that a legitimate vehicle entering communication range during a genuine hard brake would also trigger this check; however, since $w_{1b} = 0.20 < \Theta = 0.55$, this alone does not cause a false positive.

5.3 Stage 2: Deceleration Plausibility

Check 2a (Physical limit). On dry pavement, standard vehicles achieve peak sustained braking of approximately 0.8–0.9g [13]; the American Association of State Highway and Transportation Officials (AASHTO) road design uses a conservative deceleration rate of 3.4 m/s^2 ($\approx 0.35g$) for stopping sight distance calculations [1]. Deceleration at or above $1g$ is unattainable without specialized surfaces, making it physically implausible for standard vehicles:

$$|\tilde{a}_j^{(t)}| > g = 9.81 \text{ m/s}^2 \implies \text{flag} \quad (w_{2a} = 0.20). \tag{5}$$

Check 2b (Cross-field consistency). We compare speed-derived acceleration $\hat{a}_j^{(t)}$ against reported $\tilde{a}_j^{(t)}$:

$$|\hat{a}_j^{(t)} - \tilde{a}_j^{(t)}| > \tau_a = 2.5 \text{ m/s}^2 \implies \text{flag} \quad (w_{2b} = 0.15). \tag{6}$$

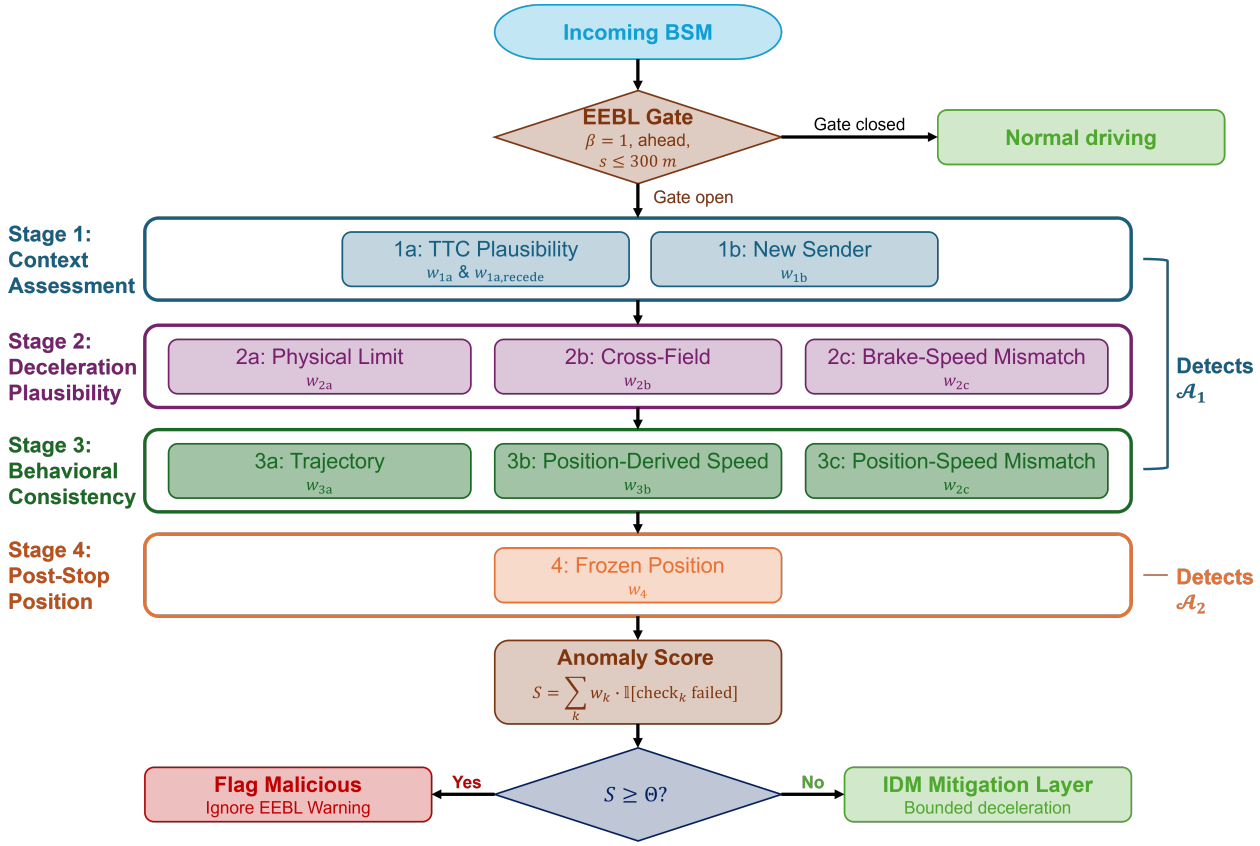


Figure 3: Overview of MIRAGE: Stages 1–3 target $\mathcal{A}_1$; Stage 4 targets $\mathcal{A}_2$; the IDM mitigation bounds the ego vehicle's response on missed detections.

The tolerance τ_a accommodates SUMO's discrete-time integration artifacts, which produce per-step acceleration discrepancies exceeding the nominal sensor noise ($\sigma_a = 0.15$ m/s^2). This check catches $\mathcal{A}_1$, where reported speed drops at half the claimed rate, but not $\mathcal{A}_2$, where $\hat{a}_j \approx \tilde{a}_j$ by construction.

Check 2c (Brake–speed mismatch). If $\beta_j = 1$ but speed-derived deceleration is mild:

$$\beta_j^{(t)} = 1 \wedge |\hat{a}_j^{(t)}| < \tfrac{1}{2} a_{\text{hard}} = 1.96 \text{ m/s}^2 \implies \text{flag}, \quad (7)$$

where $a_{\text{hard}} = 0.4g = 3.92$ m/s^2 is the SAE J2945 hard-brake threshold [35] ($w_{2c} = 0.15$). Neither $\mathcal{A}_1$ nor $\mathcal{A}_2$ consistently triggers this check, as both attacks reduce reported speed sufficiently to exceed the half-threshold. However, it provides defense-in-depth against simpler attack variants, such as a naive attacker that sets the brake flag to trigger EEBL without modifying the speed or acceleration fields. Because $w_{2c} = 0.15 < \Theta$, it contributes only as supporting evidence alongside other checks.

5.4 Stage 3: Behavioral Consistency

This stage evaluates post-event trajectory evidence, which is a capability that general-purpose per-BSM detectors do not leverage. Stage 3 requires at least $N_{\min} = 3$ received BSMs per sender to activate, because position-derived speed requires two consecutive positions and trajectory consistency requires a baseline kinematic state; Check 3a then operates on a sliding window of $W = 6$ BSMs, and Check 4 (Stage 4) uses $N_{\text{freeze}} = 5$ post-stop samples. Under sparse communication or high packet-loss conditions, the time to fill these windows grows accordingly, and we revisit this dependency in Section 8.3.

Check 3a (Trajectory consistency). Over a sliding window of $W = 6$ BSMs (0.6 s at 10 Hz, covering the initial braking phase during which position divergence becomes measurable), we compare expected displacement from reported kinematics against actual position change:

$$\delta_j^{(t)} = \left| |\mathbf{x}_j^{(t)} - \mathbf{x}_j^{(t-1)}| - \left(\tilde{v}_j^{(t-1)} \Delta t + \tfrac{1}{2} \tilde{a}_j^{(t-1)} \Delta t^2 \right) \right|. \quad (8)$$

Let $\tau_x^{(t)} = \max(3.0,\ 0.20 \cdot \tilde{v}_j^{(t)})$ m be a speed-scaled tolerance, with the base value of 3.0 m accommodating SUMO's discrete-time position integration artifacts. If more than 50% of intervals in the window satisfy $\delta_j^{(t)} > \tau_x^{(t)}$, the trajectory is flagged ($w_{3a} = 0.20$). The 50% threshold requires persistent inconsistency across at least 4 of 6 BSMs, filtering out transient noise while catching sustained trajectory divergence.

Check 3b (Post-EEBL position-derived speed). This is the primary check for $\mathcal{A}_1$. After an EEBL trigger at t_{trigger}, we monitor the position-derived speed:

$$\hat{v}_j^{(t)} = \frac{|\mathbf{x}_j^{(t)} - \mathbf{x}_j^{(t-1)}|}{\Delta t}, \quad t > t_{\text{trigger}}. \tag{9}$$

If $\hat{v}_j^{(t)} > 0.8 \cdot \hat{v}_j^{(t_{\text{trigger}})}$ for more than 60% of the post-trigger BSMs within the observation window, the vehicle never actually slowed ($w_{3b} = 0.30$). The threshold of 0.8 reflects the expected speed reduction under genuine hard braking: at the EEBL trigger threshold of $0.4g = 3.92$ m/s^2 [35], a vehicle traveling at 25 m/s would reduce its speed by 20% within approximately 1.3 s, making this a conservative bound that avoids flagging genuine braking while still detecting the sustained high-speed movement characteristic of $\mathcal{A}_1$. As discussed in Section 4, this check exploits the fact that position-derived speed bypasses reported speed and acceleration fields. However, it does not discriminate when the attacker also controls position evolution ($\mathcal{A}_2$).

Check 3c (Position–speed mismatch). Per-step comparison of position-derived and reported speed:

$$|\hat{v}_j^{(t)} - \tilde{v}_j^{(t)}| > \tau_v = 6.0 \text{ m/s} \implies \text{flag} \quad (w_{3c} = 0.15). \tag{10}$$

Position-derived speed noise is bounded by $2\sigma_x/\Delta t = 2 \times 0.5\,/\,0.1 = 10$ m/s in the worst case. In SUMO's discrete-time simulation, consecutive-position differencing exhibits empirical noise of approximately 2 m/s due to sub-step integration artifacts. The tolerance $\tau_v = 6.0$ m/s provides a conservative 3σ margin over this empirical noise, ensuring that simulation dynamics do not trigger false positives while still catching the sustained divergence produced by $\mathcal{A}_1$, where $\hat{v}_{j^*}$ remains near the pre-attack speed ($\approx$25 m/s) while $\tilde{v}_{j^*}$ approaches zero.

5.5 Stage 4: Post-Stop Position Consistency

Stages 1–3 are effective when the attacker's trajectory is kinematically inconsistent ($\mathcal{A}_1$), but they cannot distinguish $\mathcal{A}_2$ during its braking phase because all reported fields satisfy kinematic equations. However, VASP's $\mathcal{A}_2$ implementation reveals a post-attack artifact. After the ghost vehicle completes its simulated stop, the attack logic ceases updating the position field. The ghost's reported position remains frozen at the final stop location across subsequent BSMs, lacking the sub-meter global positioning system (GPS) jitter ($\sigma_x = 0.5$ m) that a physically present vehicle would exhibit.

Check 4 (Frozen position). We monitor whether a sender's position remains unchanged after it claims to have stopped. Let $\varepsilon_x = 0.01$ m be a position-change threshold and $N_{\text{freeze}} = 5$ the monitoring window. If more than 80% of position updates within the window show near-zero change:

$$\frac{\#\{t : |\mathbf{x}_j^{(t)} - \mathbf{x}_j^{(t-1)}| < \varepsilon_x\}}{N_{\text{freeze}}} > 0.8 \implies \text{flag}. \tag{11}$$

Table 2: Detection check weights.

Check	Description	w_k
1a	TTC plausibility (high TTC)	0.15
1a′	TTC plausibility (receding)	0.35
1b	New sender	0.20
2a	Physical deceleration limit	0.20
2b	Cross-field consistency	0.15
2c	Brake–speed mismatch	0.15
3a	Trajectory consistency	0.20
3b	Post-EEBL position-derived speed	0.30
3c	Position–speed mismatch	0.15
4	Frozen position after stop	0.60

The sender is flagged ($w_4 = 0.60$) if its average reported speed exceeds 0.5 m/s or the brake flag remains set. This additional condition prevents flagging a legitimately parked vehicle. The tight threshold $\varepsilon_x = 0.01$ m targets VASP's attack implementation, where the ghost's position field freezes entirely after the simulated stop, producing position changes well below the sub-meter GPS jitter ($\sigma_x = 0.5$ m) expected from a physically present vehicle. This check does not trigger during the $\mathcal{A}_2$ braking phase (where position updates are kinematically consistent), but activates once the attacker's trajectory generator stops and the position field freezes. Because Check 4 triggers on every subsequent BSM from the frozen ghost, it provides sustained detection of $\mathcal{A}_2$ after the initial braking transient. The weight $w_4 = 0.60$ exceeds Θ, enabling immediate detection: a confirmed position freeze is a high-confidence indicator of a fabricated trajectory.

5.6 Scoring and Decision

Failed checks contribute to a cumulative anomaly score:

$$S_j^{(t)} = \sum_{k \in \mathcal{K}_{\text{active}}} w_k \cdot \mathbf{1}[\text{check}_k \text{ failed}], \tag{12}$$

where $\mathcal{K}_{\text{active}} \subseteq \{1a, 1a', 1b, 2a, 2b, 2c, 3a, 3b, 3c, 4\}$ is the set of checks applicable at time t, with the constraint that exactly one of $\{1a, 1a'\}$ appears in $\mathcal{K}_{\text{active}}$ for any given BSM (mutually exclusive on the sender's longitudinal/TTC context). Table 2 lists all weights w_k. Stages 1–2 require the EEBL gate to be open; Stage 3 additionally requires $|\mathcal{H}_j| \geq N_{\min}$ to ensure sufficient trajectory history; Stage 4 requires the sender to have reported a near-zero speed. Since Checks 1a and 1a′ are mutually exclusive, the maximum achievable score is 2.30. With the exception of Check 4 ($w_4 = 0.60 > \Theta$), all individual checks have weights below Θ, so detection generally requires corroborating evidence from multiple checks. Check 4 can independently trigger detection: a confirmed position freeze is a high-confidence indicator warranting immediate flagging. A BSM is flagged as malicious if:

$$S_j^{(t)} \geq \Theta = 0.55. \tag{13}$$

We calibrate $\Theta = 0.55$ empirically to balance detection against false-positive cost; sensitivity to this choice is analyzed in Section 7. The individual check weights w_k map to semantic confidence: $w_4 = 0.60$ (confirmed freeze, high confidence), $w_{3b} = 0.30$ (sustained high speed, strong but not conclusive), with the remaining $w \in [0.15, 0.20]$ providing supporting evidence. Tolerance thresholds and window lengths are likewise empirically selected rather than optimized, with the full per-parameter rationale (standards-based thresholds, physics- and noise-derived tolerances, BSM-rate-driven windows) provided in the project repository (`docs/parameter_derivation.md`).

5.7 IDM Mitigation Layer

When a BSM passes the EEBL gate but is *not* flagged as malicious ($S_j^{(t)} < \Theta$), the ego decelerates according to the IDM-prescribed acceleration rather than executing maximum emergency braking:

$$a_i^{\text{mitigated}} = \max\left(a_{\text{IDM}}\left(v_i^{(t)}, s_{ij}^{(t)}, \Delta v_{ij}^{(t)}\right), -2b\right). \quad (14)$$

The lower bound $-2b = -4.0$ m/s^2 caps the IDM output at twice the comfortable deceleration, corresponding to the SAE J2945 hard-brake threshold of $0.4g$ [35]. If the EEBL is flagged ($S_j^{(t)} \geq \Theta$), the ego ignores the warning and continues normal driving. The IDM layer thus serves as a *false-negative safety net*: when a fake EEBL evades detection, the ego's response is still bounded by IDM rather than an unconstrained emergency brake. If the event is genuine, IDM produces appropriate braking for the current gap and speed [43]. This is especially important for $\mathcal{A}_2$, where detection is inherently limited: even on a missed detection, IDM bounds deceleration to a level consistent with the current gap rather than commanding a full emergency stop.

5.8 Algorithm

Algorithm 1 presents the end-to-end detection and mitigation pipeline. For each incoming BSM, the detector first evaluates the EEBL gate conditions (line 3). BSMs that do not satisfy the gate are discarded without further processing. When the gate is open, Stages 1–4 are executed sequentially, each contributing weighted evidence to the anomaly score $S_j^{(t)}$. If the score exceeds Θ, the sender is flagged and its EEBL warning is suppressed (line 16); otherwise, the ego vehicle responds with IDM-bounded deceleration rather than unconstrained emergency braking (line 18).

Algorithm 1 MIRAGE: Multi-stage Inspection and Response Against Ghost-vehicle EEBL

Require: BSM stream; Θ, T_{warn}, $N_{\min}$, W, N_{freeze}

1: **for each** incoming BSM $m_j^{(t)}$ from vehicle j **do**
2: Append $m_j^{(t)}$ to history $\mathcal{H}_j$; set $S_j^{(t)} \leftarrow 0$
3: **if** $\beta_j^{(t)} = 1$ **and** $|\mathbf{x}_j^{(t)} - \mathbf{x}_i^{(t)}| \leq R_{\max}$ **and** consecBrake$(j) \geq 2$ **then** ▷ **EEBL Gate**
4: **Stage 1:** Checks 1a–1b (Eqs. 3–4)
5: **Stage 2:** Checks 2a–2c (Eqs. 5–7)
6: **if** $|\mathcal{H}_j| \geq N_{\min}$ **then**
7: **Stage 3:** Checks 3a–3c (Eqs. 8–10)
8: **end if**
9: **if** $\tilde{v}_j^{(t)} \approx 0$ **and** position frozen **then**
10: **Stage 4:** Check 4 (Eq. 11)
11: **end if**
12: **end if**
13: **if** j ahead of i **and** $s_{ij}^{(t)} > 0$ **then**
14: Compute a_{IDM} via Eq. 14
15: **end if**
16: **if** $S_j^{(t)} \geq \Theta$ **then**
17: **Flag** j; ignore EEBL from j
18: **else if** EEBL active:
19: Respond with $a_i^{\text{mitigated}} = a_{\text{IDM}}$ (Eq. 14)
20: **end if**
21: **end for**

6 Evaluation

6.1 Simulation Platform

We implement MIRAGE within the VASP framework [5], which couples the OMNeT++ discrete-event network simulator with the SUMO traffic simulator via the Veins V2X simulation framework. VASP provides native implementations of 68 BSM attack types, including the two fake EEBL variants evaluated in this work. BSMs are generated at 10 Hz ($\Delta t = 0.1$ s) and processed by each vehicle's on-board detector. This integrated evaluation captures realistic vehicle dynamics and BSM exchange while isolating detection logic from physical-layer effects by running under ideal channel conditions. This is an idealization; packet loss, latency, and congestion effects on the windowed checks are discussed in Section 8.3. By evaluating all detectors within the same framework using VASP's native attack implementations, we ensure a fair comparison: every method processes identical BSM streams under identical traffic conditions, eliminating confounds from differing attack models or simulation fidelity.

6.2 Scenario Configuration

We simulate a single-lane highway with $N = 20$ SUMO-controlled vehicles over 120 s. Vehicle dynamics are governed by SUMO's IDM car-following model with driver imperfection $\sigma = 0.5$ and speed deviation $\pm 10\%$, and BSMs

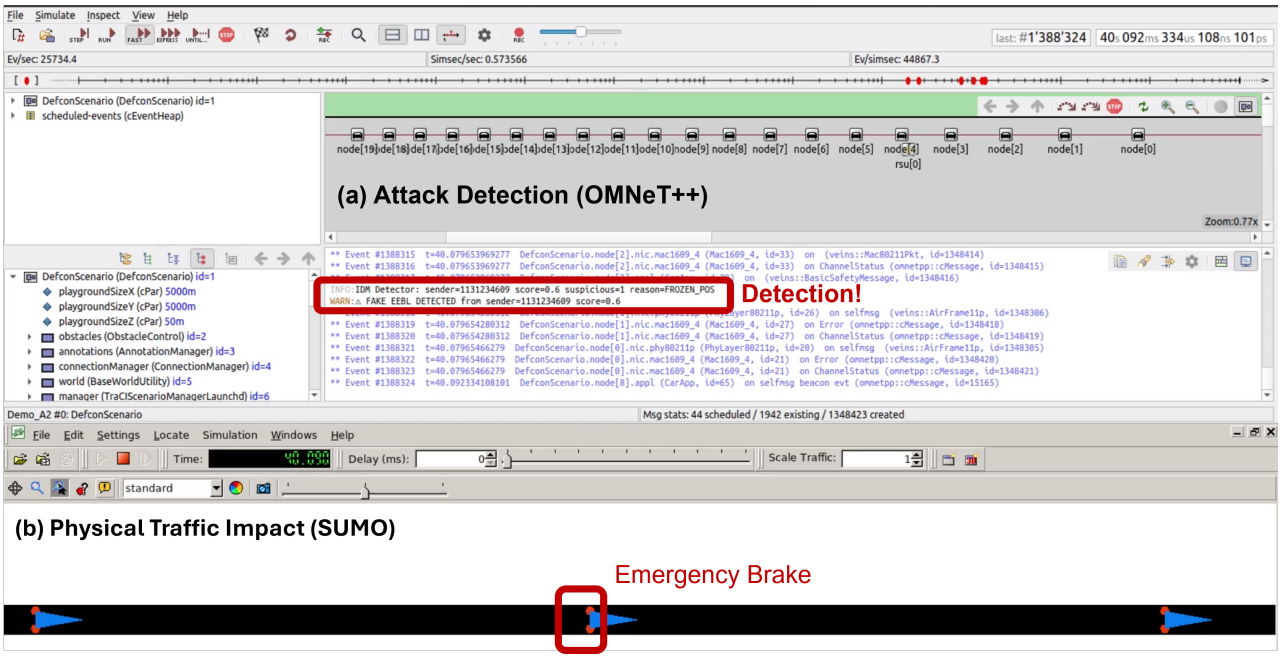


Figure 4: Visualization of the Fake EEBL attack scenario. **(Top)** The OMNeT++ log window shows the detection of a malicious "Fake EEBL" message from the attacker vehicle. **(Bottom)** The SUMO simulation view illustrates the consequence, where trailing vehicles perform emergency braking (highlighted in red), disrupting normal traffic flow.

are exchanged via Veins at 10 Hz. IDM parameters for the mitigation layer follow standard calibrations [23, 43]: $v_0 = 33.33$ m/s, $T = 1.5$ s, $a_{\max} = 1.4$ m/s^2, $b = 2.0$ m/s^2, $s_0 = 2.0$ m. To establish statistical reliability, we run each configuration across five seeds with randomized initial vehicle speeds (25 ± 2 m/s) and inter-vehicle gaps (30 ± 3 m), varying both the OMNeT++ random seed and the SUMO route file per run.

One designated vehicle (node 19) acts as the attacker, as shown in Figure 4. In attack scenarios, it executes VASP's native $\mathcal{A}_1$ or $\mathcal{A}_2$ implementation, injecting a ghost vehicle that broadcasts fabricated EEBL BSMs to all vehicles within communication range. Each of the remaining 19 vehicles independently evaluates incoming BSMs using its on-board detector. We run separate simulations for baseline (no attack), $\mathcal{A}_1$, and $\mathcal{A}_2$ for each detector configuration.

6.3 Baselines

We compare against three baselines spanning the range of existing MDS approaches. Table 3 summarizes the differences. We use each baseline's published default thresholds and configuration; tuning baselines specifically for the fake-EEBL setting could shift their per-attack tradeoffs but is unlikely to remove the cross-attack imbalance documented in RQ2 (Threshold and F2MD detect $\mathcal{A}_1$-style decelerations but miss $\mathcal{A}_2$, while VCADS exhibits the opposite pattern). Event-gated variants of these baselines were not evaluated and are an interesting direction for follow-up work.

Table 3: Comparison of detection approaches.

	B1 [24]	**B2** [3]	**B3** [5, 20]	**MIRAGE**
Activation	Always	Always	Always	Event-gated
Check type	Threshold	Cross-field	Multi-check	Multi-stage
Post-event traj.	×	×	×	✓
Post-stop check	×	×	×	✓
Weighted scoring	×	×	×	✓
Mitigation	×	×	×	IDM-based

B1: Threshold [24]. Range-and-threshold plausibility checks [6, 24] that evaluate individual BSM fields against fixed bounds ($\tilde{v}_j < 70$ m/s, $|\tilde{a}_j| < 11$ m/s^2, $s_{ij} < 300$ m), augmented with a basic brake-deceleration consistency rule that compares the brake flag against reported deceleration.

B2: VCADS [3]. Physics-based cross-field validation using a sliding window of $W = 5$ BSMs. VCADS computes three cross-checks between reported and derived kinematics: acceleration, position-displacement, and speed consistencies. Although it uses a temporal window, all checks operate on instantaneous field pairs within the window rather than analyzing post-event trajectory patterns.

B3: F2MD/VASP [5, 20]. A multi-check plausibility chain combining six checks: communication and speed range verification, position jump rate, BSM frequency, and position-speed/acceleration consistency. The checks span both per-

field thresholds and cross-field comparisons but are applied uniformly to every BSM without application-specific gating.

6.4 Metrics

We evaluate detection performance using four metrics at the per-BSM granularity: true positive rate (TPR), false positive rate (FPR), precision, and F1 score. A true positive is an attack BSM correctly flagged by the detector, and a false positive is a benign BSM incorrectly flagged.

In the EEBL context, FPR carries asymmetric cost: a false positive suppresses a genuine emergency warning, whereas a false negative results in unnecessary but recoverable braking [45]. We also report per-attack breakdowns to separate the performance of $\mathcal{A}_1$ and $\mathcal{A}_2$, since aggregate metrics can mask important asymmetries.

7 Results

Each detector processes approximately 4.7 million BSM evaluations per seed (3.7M benign, 1.0M attack) across five seeds with randomized traffic conditions (σ=0.5, varied initial speeds and gaps). We evaluate MIRAGE to answer the following research questions (RQ):

RQ1 (Effectiveness of MIRAGE): How accurately does MIRAGE identify fake EEBL messages?
RQ2 (Baseline Comparison): How does MIRAGE perform relative to existing detectors?
RQ3 (Per-Attack Analysis): How does detection performance differ between the two attack variants $\mathcal{A}_1$ and $\mathcal{A}_2$?
RQ4 (Parameter Sensitivity): How robust is MIRAGE's detection accuracy to variations in the decision threshold Θ?
RQ5 (Response Mitigation): How much does the IDM-based mitigation limit braking severity on undetected attack BSMs?

7.1 RQ1: Effectiveness of MIRAGE

Table 4 summarizes aggregate performance across five seeds. MIRAGE achieves an F1 score of 0.922 ± 0.001 at the operating point $\Theta = 0.55$, with TPR = 0.883, FPR = 0.009, and precision = 0.965. The low FPR is particularly noteworthy in the EEBL context: Van der Heijden et al. [45] emphasize that false positives in safety applications carry asymmetric cost, as suppressing a genuine emergency warning may directly cause a collision. Several baselines also achieve low per-BSM FPR (Threshold and F2MD both at 0.009; see Table 4); the distinguishing property of MIRAGE is that it achieves this low FPR while simultaneously maintaining high TPR across both attack variants, where Threshold and F2MD trade FPR for substantial recall loss on at least one variant. Event-gating contributes to this profile by restricting deep inspection to EEBL-active periods, which keeps false alarm pressure low even as the detector applies more sensitive multi-stage checks. The balanced per-attack coverage (77.0% for $\mathcal{A}_1$, 99.7% for $\mathcal{A}_2$ on VASP's current $\mathcal{A}_2$ implementation; see RQ3) confirms that the aggregate F1 reflects consistent detection rather than strength on a single variant.

Table 4: Aggregate detection performance averaged over 5 seeds. Standard deviations in parentheses.

Method	TPR	FPR	Precision	F1 score
B1: Threshold	0.597	0.009	0.950	0.733
	(0.003)	(0.001)	(0.009)	(0.004)
B2: VCADS	0.499	0.040	0.778	0.607
	(0.000)	(0.011)	(0.059)	(0.018)
B3: F2MD/VASP	0.110	0.009	0.770	0.193
	(0.003)	(0.001)	(0.037)	(0.006)
MIRAGE	**0.883**	**0.009**	**0.965**	**0.922**
	(0.003)	**(0.001)**	**(0.007)**	**(0.001)**

7.2 RQ2: Baseline Comparison

Table 4 compares MIRAGE against three baselines. MIRAGE substantially outperforms VCADS (F1 = 0.607), Threshold (F1 = 0.733), and F2MD/VASP (F1 = 0.193). Threshold achieves moderate TPR (0.597) by detecting primarily $\mathcal{A}_2$ through its brake-deceleration consistency, but misses most $\mathcal{A}_1$ instances. VCADS detects $\mathcal{A}_1$ effectively through cross-field validation but fails on the kinematically consistent $\mathcal{A}_2$, yielding low aggregate TPR (0.499) despite high per-attack accuracy on $\mathcal{A}_1$. F2MD/VASP achieves the lowest detection rate (TPR = 0.110) because it was designed for VeReMi-style misbehavior [21, 46], where attacks produce extreme BSM field values. Fake EEBL attacks create kinematically plausible values that pass its checks. MIRAGE's advantage stems from combining trajectory-based checks (effective against $\mathcal{A}_1$) with post-stop position consistency (effective against $\mathcal{A}_2$), a complementarity that no single baseline achieves.

7.3 RQ3: Per-Attack Detection Analysis

Figure 5 separates results by attack type. Each baseline detector shows a pronounced specialization toward one attack variant, whereas MIRAGE is the only detector to maintain high TPR across both.

For $\mathcal{A}_1$ (NoStop), VCADS achieves the highest per-attack TPR (0.997) through its cross-field kinematic validation, which detects the speed-position divergence inherent in this attack. MIRAGE reaches TPR = 0.770, driven primarily by Check 3b (post-EEBL position-derived speed). Threshold and F2MD both achieve approximately 0.22, as $\mathcal{A}_1$'s fabricated values remain within their plausibility bounds.

For $\mathcal{A}_2$ (WithStop), the situation reverses. VCADS achieves TPR = 0.000 because the ghost's fields are mutually consistent during the braking phase, leaving no cross-field discrepancy to detect. Threshold detects $\mathcal{A}_2$ at 0.976 through its brake-

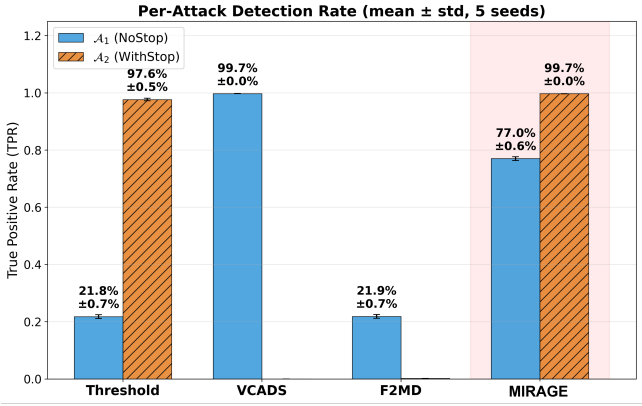


Figure 5: Per-attack detection rate (TPR). MIRAGE is the only detector that maintains high TPR across both $\mathcal{A}_1$ and $\mathcal{A}_2$.

deceleration consistency rule in the post-stop phase. After the ghost completes its simulated stop, the brake flag remains set while the speed-derived deceleration drops to zero, triggering the brake-without-deceleration check. This detection, similar to our Stage 4, depends on post-stop behavior rather than the kinematically consistent braking phase. MIRAGE achieves TPR = 0.997, with detection driven by Stage 4: after the ghost completes its simulated stop, its position field freezes, and Check 4 flags the absence of expected GPS jitter.

This complementarity is the core result. Existing detectors are each effective against one attack variant but fail against the other, whereas MIRAGE's multi-stage architecture provides balanced coverage. The F1 score of 0.922 reflects this balanced performance rather than strength in a single variant.

We note that MIRAGE's $\mathcal{A}_1$ TPR (0.770) is lower than VCADS (0.997) because our position-derived speed check (Check 3b) requires a short observation window before the divergence becomes statistically distinguishable from sensor noise. During the first few BSMs after the EEBL trigger, the position displacement difference is within the noise margin, and these early BSMs are scored below Θ. This reflects a deliberate design choice: we prioritize low FPR over maximizing per-attack TPR for $\mathcal{A}_1$, since our IDM mitigation layer bounds the ego's response even for undetected attack BSMs during this transient window. In contrast, VCADS achieves near-perfect $\mathcal{A}_1$ detection but with an FPR of 0.040, more than four times ours (0.009), and zero detection of $\mathcal{A}_2$.

7.4 RQ4: Parameter Sensitivity

Figure 6 shows TPR, FPR, and F1 as functions of Θ (mean $\pm$ std across five seeds). The best F1 (0.925) occurs at $\Theta = 0.52$, close to our chosen operating point of 0.55. Performance is stable across $\Theta \in [0.45, 0.59]$: TPR remains above 0.88 and F1 above 0.92, indicating that the detector is robust to the exact threshold choice.

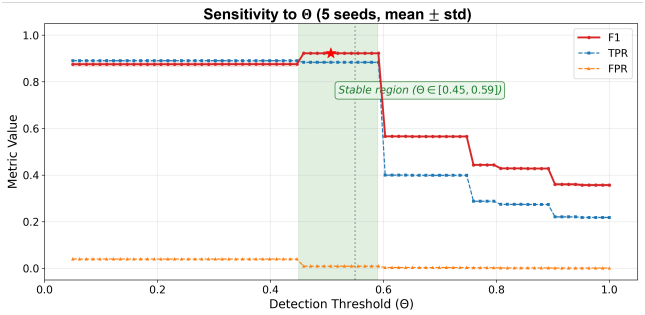


Figure 6: Detection performance as a function of Θ. The shaded region marks the stable operating range. The star marks the best F1 score.

Below $\Theta = 0.45$, FPR increases to approximately 0.05 as lower-confidence check combinations begin to trigger false alarms. At $\Theta = 0.60$, TPR drops from 0.883 to 0.41 and F1 drops to 0.57 because Check 4's weight ($w_4 = 0.60$) no longer exceeds the threshold, collapsing $\mathcal{A}_2$ detection entirely. Above $\Theta = 0.75$, TPR falls below 0.30 as even multi-check attack evidence falls below the threshold. We selected $\Theta = 0.55$ as it lies within the stable region and slightly favors detection coverage over the F1-optimal 0.52. The marginal FPR difference between 0.52 and 0.55 is acceptable in our deployment model. MIRAGE is intended as a complement to PKI-level revocation rather than a sole decision authority. False positives produce ignored EEBL warnings rather than safety actions, and the IDM mitigation layer is invoked when detection *misses* an attack, not when detection produces a false alarm.

7.5 RQ5: Response Mitigation

Table 5 summarizes the IDM-bounded deceleration for EEBL-relevant BSMs. For attack BSMs, the IDM-bounded deceleration averages -2.03 m/s^2 (median -1.86 m/s^2), with a maximum magnitude of -4.0 m/s^2. This compares to the -8.0 m/s^2 ($\approx 0.8g$) maximum emergency braking that an unprotected EEBL response would command, representing a 75% reduction in mean braking severity and a 50% reduction in peak deceleration magnitude. For benign BSMs in EEBL-relevant contexts, IDM output averages -1.75 m/s^2 (median -1.36 m/s^2), consistent with normal car-following deceleration. Both distributions are bounded well above the emergency braking threshold, confirming that IDM limits the ego's response to physics-reasonable levels regardless of whether the triggering BSM is genuine or fabricated.

The IDM layer provides defense in depth: for the small fraction of attack BSMs that evade detection (primarily during $\mathcal{A}_2$'s initial braking phase before Stage 4 activates), the ego's response is bounded to a level consistent with the observed

Table 5: IDM-bounded deceleration for EEBL BSMs.

BSM Type	Mean	Median	Min
Attack (n=194,100)	-2.03 m/s^2	-1.86 m/s^2	-4.00 m/s^2
Benign (n=1,292,323)	-1.75 m/s^2	-1.36 m/s^2	-4.00 m/s^2
Unprotected EEBL	-8.0 m/s^2 (emergency braking)		

gap and closing speed rather than an unconstrained emergency stop. This fail-safe property is particularly valuable given the safety-critical nature of EEBL, where false negatives directly translate to unnecessary hard braking.

To contextualize this reduction, the IDM-bounded peak of -4.0 m/s^2 ($\approx 0.4g$) corresponds to the SAE J2945 hard-brake threshold itself [35], meaning that even in the worst case, the ego's response does not exceed the deceleration level that would normally trigger an EEBL warning. By contrast, an unprotected response at -8.0 m/s^2 ($\approx 0.8g$) approaches the physical braking limit of production vehicles [13], substantially increasing rear-end collision risk. This quantitative bound distinguishes our approach from binary accept/reject detectors, which provide no mitigation on missed detections.

8 Discussion

We discuss MIRAGE's detection behavior on the evaluated attacks (Section 8.1), its deployment considerations and generalization potential beyond EEBL (Section 8.2), and the explicit limitations that scope its claims (Section 8.3).

8.1 Detection Analysis

Detecting $\mathcal{A}_2$ via Post-Attack Artifacts. FakeEEBL_WithStop produces kinematically consistent BSMs during its braking phase, making it indistinguishable from a valid hard-brake trajectory at the field level, a well-known evasion principle [5, 32]. Our Stage 4 check instead targets the ghost's post-attack trajectory artifact: in VASP's implementation, the position field freezes after the simulated stop, producing changes below $\varepsilon_x = 0.01$ m across consecutive BSMs. Check 4 detects this with 99.7% TPR and carries the highest weight ($w_4 = 0.60 > \Theta$), enabling immediate flagging once the freeze is confirmed. An adaptive attacker could instead emit plausible post-stop updates (see Adaptive Attacker Analysis below); the IDM mitigation serves as the safety net in that case.

Per-Check Contribution Analysis. An ablation shows that Check 4 (position freeze, w=0.60) drives $\mathcal{A}_2$ detection almost exclusively (ΔTPR $= -0.974$ when removed), while Check 3b (post-EEBL no stop, w=0.30) is the primary contributor for $\mathcal{A}_1$ (ΔTPR $= -0.222$). The remaining checks have a marginal impact against the two evaluated variants, since these attacks do not trigger the anomalies those checks target (physical-limit violations, cross-field inconsistencies); we retain them as defense-in-depth layers for more adaptive future attacks. The full ablation figure (ΔF1 and ΔTPR across Overall, $\mathcal{A}_1$, $\mathcal{A}_2$ for each removed check) is provided in the project repository (`docs/extended_discussion.md`).

Adaptive Attacker Analysis. Defenses validated against fixed attack implementations may fail against adaptive adversaries [5, 32], so we make MIRAGE's threat envelope explicit through three adaptive-attacker classes beyond $\mathcal{A}_1$, $\mathcal{A}_2$. *Path 1 (GPS noise)* injects jitter above $\varepsilon_x = 0.01$ m to defeat Check 4, but no fixed threshold resolves this (values below the genuine jitter envelope $\sigma_x \approx 0.5$ m are replicable; above it causes false positives), so discrimination needs evidence beyond a single sender's stream [3, 39]. *Path 2 (continued plausible updates)* never triggers Check 4 and is the practical upper bound for content-only detection, though Checks 3b and 3c ($\tau_v = 6.0$ m/s) still narrow the evasion. *Path 3 (history-building, pseudonym change)* does not directly evade MIRAGE but maps to the SCMS/revocation layer, out of scope [34,49]. Because content-only detection cannot close Path 2, the IDM layer caps ego deceleration at -4.0 m/s^2 regardless of detection [4,37], reducing worst-case braking from -8.0 m/s^2 to the SAE J2945 threshold [35] as a mitigation, not a guarantee. None of these undermine the event-gating design; a per-path breakdown is in the project repository (`docs/extended_discussion.md`).

8.2 Deployment Considerations

Safety vs. Security Tradeoff. Suppressing a genuine EEBL is potentially more dangerous than acting on a fake one [45]. Our design reflects this asymmetry: $\Theta = 0.55$ tolerates some missed attacks while keeping a low FPR, and IDM mitigation ensures graceful degradation rather than binary accept/reject, aligning with fail-safe recommendations in V2X standardization [4, 37].

Authentication Without Authorization. The vehicular PKI and SCMS [49] authenticate that a BSM originates from an enrolled device but impose no authorization on message content, so any enrolled OBU may broadcast any kinematic values. Proposed content-level validation mechanisms (sensor-aided cross-checking [3], cooperative witness protocols [4], reputation systems [10,12]) are not yet adopted in deployment standards [8,35]. MIRAGE operates within this gap, requiring no infrastructure changes, extra message exchanges, or SCMS modifications, and performing all checks locally on standard BSM fields.

Generalization Beyond EEBL. The event-gated architecture is a design pattern, not a result specific to EEBL: it requires a well-defined trigger condition, application-specific behavioral invariants a genuine event must satisfy, and a controller-level layer that bounds the ego's reaction under uncertain detection. Other longitudinal V2X safety applications (e.g., Forward

Collision Warning) fit the same template, whereas lateral applications (Intersection Movement Assist, Lane Change Warning) would require different invariants and steering-rate or lateral-acceleration bounds rather than the IDM deceleration cap. The transferable elements (gating principle, windowed-evidence aggregation, weighted scoring) and the application-specific ones that must be re-derived per standard (trigger conditions, invariants, tolerances, weights) are detailed in the project repository (`docs/extended_discussion.md`). We therefore present event-gating as a transferable design pattern while restricting empirical claims strictly to the evaluated EEBL variants ($\mathcal{A}_1$, $\mathcal{A}_2$).

Scalability in Dense Traffic. MIRAGE's per-BSM checks are closed-form arithmetic, so per-sender state grows linearly with the number of neighbors N and processing remains $O(1)$ per message ($O(N)$ per second at 10 Hz). Full Stage 1–4 evaluation runs only when the EEBL gate opens, so the full-pipeline rate is bounded by the (sparse) rate of genuine hard-brake events rather than the total BSM rate. Channel congestion and packet loss in dense traffic degrade the sliding-window checks (Section 8.3, item 3) rather than the algorithmic scalability, and each sender is evaluated independently. A detailed treatment of all four scaling concerns is provided in the project repository (`docs/extended_discussion.md`); quantifying performance under dense, lossy-channel operation is left for future work.

Simulation Realism and Computational Overhead. A potential concern with simulation-based evaluation is whether the results transfer to real-world V2X deployments. We address this through three design choices. First, SUMO's car-following models are extensively calibrated against real-world trajectory data [23, 26], and the IDM parameters we adopt ($v_0 = 33.33$ m/s, $T = 1.5$ s, $b = 2.0$ m/s^2) follow standard calibrations that reproduce empirical speed-flow relationships [43]. Second, the sensor noise parameters ($\sigma_x = 0.5$ m, $\sigma_a = 0.15$ m/s^2) reflect typical GPS and IMU error bounds reported in V2X field operational tests [18, 22]. Third, evaluating all methods on VASP's native attack implementations ensures the comparison is not biased by our own attack modeling. Ideal channel conditions represent a best-case assumption; the impact of packet loss and channel congestion is discussed under Section 8.3.

All detection checks are closed-form arithmetic (comparisons, subtractions, divisions) with no iterative solvers, cryptographic operations, or ML inference, and the full pipeline runs only when the EEBL gate is open; during normal driving, incoming BSMs are dismissed by the gate test with no further processing. In our evaluation, each of the 19 non-attacker vehicles ran the full detector independently in OMNeT++, processing about 4.7 million BSM evaluations per seed without simulation delays or relaxation of the 10 Hz rate, confirming that per-BSM processing fits within the $\Delta t = 0.1$ s BSM interval. MIRAGE flags attacks within 0.14 s for $\mathcal{A}_1$ and 0.29 s for $\mathcal{A}_2$ (mean, across the 19 ego vehicles over 5 seeds), faster than all baselines: VCADS detects $\mathcal{A}_1$ comparably (0.24 s) but fails entirely on $\mathcal{A}_2$, while Threshold and F2MD require 1.5 to 9.7 s on average. Even during the pre-detection window, the IDM mitigation layer caps the ego's deceleration at -4.0 m/s^2. The full end-to-end detection-latency breakdown (mean, median, and missed-BSM counts per detector and attack variant) is provided in the project repository (`docs/extended_discussion.md`).

8.3 Limitations

MIRAGE is an application-specific detector evaluated on VASP's two fake-EEBL implementations under single-lane, ideal-channel simulation. We make explicit what we do and do not claim. *We claim:* the event-gated architecture is effective against the two VASP-implemented fake-EEBL variants ($\mathcal{A}_1$, $\mathcal{A}_2$) under the evaluated conditions; the IDM bounding layer reduces worst-case ego deceleration even when detection misses. *We do not claim:* robustness against adaptive trajectory generators that defeat Check 4; generalization to multi-lane or urban traffic; performance under lossy or congested wireless channels; generalization to V2X services beyond EEBL; or optimality of the chosen weights and thresholds. The six items below scope each of these areas.

1. Artifact dependency and adaptive attackers. The 99.7% detection rate on $\mathcal{A}_2$ is driven by a specific implementation artifact in VASP's FakeEEBL_WithStop generator: the reported position field freezes at the claimed stop location. This headline number should not be taken as reliable evidence of physically plausible $\mathcal{A}_2$ instances; an adaptive attacker that injects synthetic GPS noise or continues plausible position updates can defeat Check 4 (Adaptive Attacker Analysis, Paths 1 and 2 in Section 8.1). Closing this gap requires sensor-aided cross-validation [3], cooperative corroboration [4], or physical-layer features [39]; the IDM layer complements these as a safety net rather than as a defense.

2. Single-lane evaluation. Our evaluation is restricted to a single-lane highway. Multi-lane traffic introduces three effects: lateral position ambiguity (adjacent-lane braking can open the EEBL gate, likely raising the false-positive rate), lane-association errors during lane changes (genuine post-braking trajectories may appear inconsistent with Check 3a), and sender- vs. ego-side lane-change transients (perturbing Check 3a, TTC, and relevance gating). Multi-lane extension likely requires map-aware lane association.

3. Ideal channel assumption. Our simulations assume ideal V2X channels. Deployed communication can be lossy (multipath fading, NLOS shadowing, collisions on the shared 5.9 GHz spectrum, congestion) [20]: packet loss reduces the BSM history available to Stages 3 and 4, delaying window fills and potentially raising false negatives for short attacks, while latency and out-of-order arrival degrade Check 3c's position-derived speed estimate. Quantifying these effects us-

ing realistic channel models or hardware-in-the-loop testbeds is left for future work.

4. BSM history requirement. Stage 3 requires $N_{\min} = 3$ BSMs per sender to activate. Under high packet loss, window-fill times increase. Senders that appear briefly and then disappear (e.g., pseudonym-change attackers) may not accumulate sufficient history; addressing this requires SCMS-level integration [49] outside our content-level scope.

5. Parameter calibration. Weights, tolerances, and thresholds are empirically selected rather than jointly optimized, and the calibration is valid for the evaluated single-lane highway scenarios with ideal channels. Each parameter has a principled rationale (physical limits, sensor noise envelopes, BSM rate, braking dynamics), detailed in the project repository (`docs/parameter_derivation.md`), and we anticipate that some thresholds (notably τ_v, τ_x, and the speed-scaled term in Check 3a) require recalibration for multi-lane or lossy-channel deployment. Systematic joint sensitivity analysis or learning-based optimization [15, 38] could improve robustness and remains an important direction for future work.

6. Scope of empirical claims. Empirical results come from a single simulator (VEINS 5.2 + SUMO + VASP), a single highway scenario, and a single attack family (fake EEBL); we have not evaluated MIRAGE on hardware testbeds, urban/intersection scenarios, or other V2X services, and coordinated multi-attacker, Sybil, and pseudonym-chain attacks [34, 49] are out of scope. Metrics are reported per-BSM, which can understate the safety impact (event-level safety depends on time-to-first-flag; see `docs/extended_discussion.md`), and the IDM mitigation bounds deceleration rather than guaranteeing collision avoidance. Specific numbers should not be extrapolated beyond this envelope without re-evaluation.

9 Related Work

V2X Data Attacks and EEBL. False safety messages have been studied since early VANET research [14, 19, 33]. Lo and Tsai [25] formalized illusion attacks, and Petit and Shladover [31] cataloged cyberattacks on connected vehicles. Amoozadeh et al. [2] showed how falsified messages cause string instability in platoons. VeReMi [46] and its extension [21] provide labeled misbehavior datasets, while VASP [5] implements 68 BSM attacks, including two targeting EEBL. Beyond the vehicular domain, systematic attack discovery frameworks have also been developed for adjacent autonomous-agent systems, such as Raven [51], which uses signal temporal logic and stochastic optimization to discover semantic false-data injection attacks against multi-robot collision-avoidance algorithms. Despite these advances, detection work specifically addressing fake EEBL remains sparse.

Misbehavior Detection Systems. Plausibility-based detection compares BSM fields against physical bounds [6, 24, 36]. Stübing et al. [41] proposed path prediction for two-stage Car-to-X verification. VCADS [3] uses kinematic equations for cross-field validation, and F2MD [20] chains multiple plausibility checks. ML approaches include support vector machine and neural network classifiers [15], consecutive BSM feature analysis [38], and RSSI-based physical-layer checks [39]. Recent surveys [7, 37, 45] classify these into data-centric and node-centric categories.

Our work differs from existing MDS not by introducing individual components but by integrating application-specific event gating, trajectory-level behavioral analysis, and physics-grounded response bounding into a single EEBL-specific defense. Three aspects of this integration are specific to our work. First, we validate the EEBL *event* rather than individual BSM fields; this application-specific gating reduces the false-alarm surface that plagues always-on detectors in safety-critical contexts [45]. Second, we check *post-event behavioral consistency* through position-derived speed, a quantity that existing cross-field validators do not exploit because they operate on instantaneous BSM pairs rather than post-trigger trajectory windows. Third, we incorporate an IDM-based mitigation layer that bounds the ego's physical response, providing a fail-safe mechanism absent from all prior detection-only approaches [7, 37]. To our knowledge, this integration of application-specific event gating, trajectory-level behavioral analysis, and physics-grounded response bounding has not been explored in the V2X misbehavior detection literature.

Resilient V2X Response. Beyond detection, the literature has explored credential revocation [34, 49], reputation-based trust [10, 12], blockchain-based consensus [50], and sensor-aided validation [3, 39]. In platooning, bounded filtering limits the impact of malicious commands [2]. Our IDM-based mitigation extends this principle to ad hoc traffic by bounding the ego's response using a car-following model [43]. To our knowledge, this integration of detection and response bounding within a single EEBL-specific framework has not been previously proposed.

10 Conclusion

We presented MIRAGE, an event-gated multi-stage misbehavior detector for fake EEBL attacks grounded in the Intelligent Driver Model. The detector restricts inspection to EEBL-active contexts, validates post-event trajectory consistency through position-derived speed, and detects post-stop position-freezing artifacts. Across five randomized seeds, it achieves an F_1 of 0.922 ± 0.001 at 0.9% FPR, with 77.0% TPR on $\mathcal{A}_1$ and 99.7% on $\mathcal{A}_2$ (the latter driven by an implementation-specific position-freezing artifact in VASP, discussed in Section 8.3), making it the only evaluated detector that maintains high detection across both kinematically inconsistent and self-consistent attacks. The IDM mitigation layer further reduces mean braking severity by 75% and limits peak deceleration to -4.0 m/s^2 for undetected attack BSMs.

Our results confirm that application-specific, event-driven detection outperforms general-purpose BSM plausibility checks for EEBL and suggest that the V2X security community should move toward application-aware detection architectures that exploit the semantics and temporal structure of each safety application. Defending against adaptive trajectory generators that continue to produce plausible position updates remains an open problem. Future work will evaluate the detector under wireless channel impairments, extend event-gating to other V2X services such as Intersection Movement Assist and Forward Collision Warning, explore learning-based weight optimization using larger misbehavior datasets [7, 21], and pursue staged hardware validation from closed-track V2X testbeds to instrumented-vehicle deployments.

Acknowledgements

This work is based upon the work supported by the National Center for Transportation Cybersecurity and Resiliency (TraCR) (U.S. Department of Transportation National University Transportation Center) headquartered at Clemson University, Clemson, South Carolina, USA. Any opinions, findings, conclusions, and recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of TraCR, and the U.S. Government assumes no liability for the contents or use thereof.

Ethical Considerations

This work studies fake EEBL attacks exclusively in a closed-loop simulation environment (VASP/VEINS/SUMO) with no real vehicles, real roadways, or human participants involved at any stage of the research. All experiments were conducted on synthetic traffic scenarios generated by the SUMO microscopic traffic simulator, and no data from real drivers or production vehicle systems were collected or used.

The two attack variants evaluated ($\mathcal{A}_1$ and $\mathcal{A}_2$) are implemented using VASP's existing native attack modules, which are publicly available as open-source software to the research community [5]. No new attack tools, exploits, or vulnerability disclosures were developed as part of this work; the attack descriptions in Section 3 are limited to what is necessary to explain the detection methodology and reproduce the experimental evaluation. The primary goal of this research is defensive: to improve the detection and mitigation of safety-critical V2X threats that have been identified as high risk in prior threat assessments [5,52]. We believe MIRAGE and comparative evaluations against existing baselines, substantially outweigh the marginal risk of describing attack vectors that are already thoroughly documented in prior work [25, 32, 52].

Availability

To support reproducibility and independent re-evaluation, the complete MIRAGE artifact (detector source code, automated install pipeline, SUMO/Veins scenarios, analysis scripts, and reference results) is permanently archived on Zenodo at https://doi.org/10.5281/zenodo.19639252 (concept DOI resolving to the latest version; camera-ready snapshot is version 2.1, https://doi.org/10.5281/zenodo.20398530), released under the MIT license, and awarded the Artifacts Available, Functional, and Reproduced badges. Supplementary documentation cited in this paper (in `docs/`) is maintained in the project repository at https://github.com/eunhanka/mirage-eebl-detector.

Use of Generative AI

In accordance with the VehicleSec 2026 policy on the use of generative AI, we disclose that ChatGPT (OpenAI) was used during manuscript preparation for grammar checking and LaTeX formatting assistance. No generative AI tool was used to generate technical content, design the detection methodology, implement the detector, conduct experiments, analyze data, or interpret results. All scientific contributions are entirely the work of the authors. The authors have verified all content and assume full responsibility for its originality.

References

[1] American Association of State Highway and Transportation Officials. *A Policy on Geometric Design of Highways and Streets*, 7th ed. AASHTO, Washington, DC, USA, 2018.

[2] Amoozadeh, M., Raghuramu, A., Chuah, C., Ghosal, D., Zhang, H. M., Rowe, J., and Levitt, K. Security vulnerabilities of connected vehicle streams and their impact on cooperative driving. *IEEE Communications Magazine 53*, 6 (2015).

[3] Andrade Salazar, A. A., McDaniel, P. D., Sheatsley, R., and Petit, J. Physics-based misbehavior detection system for V2X communications. *SAE International Journal of Connected and Automated Vehicles 5*, 3 (2022).

[4] Ansari, M. R., Monteuuis, J.-P., Petit, J., and Chen, C. V2x misbehavior and collective perception service: Considerations for standardization. In *2021 IEEE Conference on Standards for Communications and Networking (CSCN)* (2021), IEEE.

[5] Ansari, M. R., Petit, J., Monteuuis, J.-P., and Chen, C. VASP: V2X application spoofing platform. In *Proc. of NDSS VehicleSec Workshop* (2023).

[6] Bissmeyer, N., Mauthofer, S., Bayarou, K. M., and Kargl, F. Assessment of node trustworthiness in VANETs using data plausibility checks with particle filters. In *Proc. of IEEE Vehicular Networking Conference (VNC)* (2012).

[7] Boualouache, A., and Engel, T. A survey on machine learning-based misbehavior detection systems for 5g and beyond vehicular networks. *IEEE Communications Surveys & Tutorials 25*, 2 (2023).

[8] Bouchouia, M. L., Labiod, H., Jelassi, O., Monteuuis, J.-P., Jaballah, W. B., Petit, J., and Zhang, Z. A survey on misbehavior detection for connected and autonomous vehicles. *Vehicular Communications 41* (2023).

[9] CHECKOWAY, S., MCCOY, D., KANTOR, B., ANDERSON, D., SHACHAM, H., SAVAGE, S., KOSCHER, K., CZESKIS, A., ROESNER, F., AND KOHNO, T. Comprehensive experimental analyses of automotive attack surfaces. In *Proc. of USENIX Security Symposium* (2011).

[10] DHURANDHER, S. K., OBAIDAT, M. S., JAISWAL, A., TIWARI, A., AND TYAGI, A. Vehicular security through reputation and plausibility checks. *IEEE Systems Journal 8*, 2 (2014).

[11] FALLAH, Y. P., HUANG, C.-L., SENGUPTA, R., AND KRISHNAN, H. Design of cooperative vehicle safety systems based on tight coupling of communication, computing, and physical vehicle dynamics. In *ACM/IEEE Int. Conf. on Cyber-Physical Systems (ICCPS)* (2010).

[12] GHALEB, F. A., ZAINAL, A., MAROOF, M. A., RASSAM, M. A., AND SAEED, F. Detecting bogus information attack in vehicular ad hoc network: a context-aware approach. *Procedia Computer Science 163* (2019).

[13] GILLESPIE, T. *Fundamentals of vehicle dynamics*. SAE international, 2021.

[14] GOLLE, P., GREENE, D., AND STADDON, J. Detecting and correcting malicious data in vanets. In *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks* (2004).

[15] GYAWALI, S., AND QIAN, Y. Misbehavior detection using machine learning in vehicular communication networks. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)* (2019), IEEE.

[16] HARDING, J., POWELL, G., YOON, R., FIKENTSCHER, J., DOYLE, C., SADE, D., LUKUC, M., SIMONS, J., AND WANG, J. Vehicle-to-vehicle communications: Readiness of V2V technology for application. Tech. Rep. DOT HS 812 014, NHTSA, 2014.

[17] HAYWARD, J. C. Near-miss determination through use of a scale of danger. In *Highway Research Record, HRR 384* (1972).

[18] HOWE, G., XU, G., HOOVER, D., ELSASSER, D., AND BARICKMAN, F. Commercial connected vehicle test procedure development and test results – emergency electronic brake light. Tech. Rep. DOT HS 812 327, NHTSA, 2016.

[19] HUBAUX, J.-P., ČAPKUN, S., AND LUO, J. The security and privacy of smart vehicles. *IEEE Security & Privacy 2*, 3 (2004).

[20] KAMEL, J., ANSARI, M. R., PETIT, J., KAISER, A., JEMAA, I. B., AND URIEN, P. Simulation framework for misbehavior detection in vehicular networks. *IEEE Transactions on Vehicular Technology 69*, 6 (2020).

[21] KAMEL, J., WOLF, M., VAN DER HEIJDEN, R. W., KAISER, A., URIEN, P., AND KARGL, F. Veremi extension: A dataset for comparable evaluation of misbehavior detection in vanets. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)* (2020), IEEE.

[22] KENNEY, J. B. Dedicated short-range communications (DSRC) standards in the United States. *Proceedings of the IEEE 99*, 7 (2011).

[23] KESTING, A., TREIBER, M., AND HELBING, D. Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 368*, 1928 (2010).

[24] LEINMÜLLER, T., SCHOCH, E., AND KARGL, F. Position verification approaches for vehicular ad hoc networks. *IEEE Wireless Communications 13*, 5 (2006).

[25] LO, N.-W., AND TSAI, H.-C. Illusion attack on vanet applications-a message plausibility problem. In *2007 IEEE globecom workshops* (2007), IEEE.

[26] LOPEZ, P. A., BEHRISCH, M., BIEKER-WALZ, L., ERDMANN, J., FLÖTTERÖD, Y.-P., HILBRICH, R., LÜCKEN, L., RUMMEL, J., WAGNER, P., AND WIESSNER, E. Microscopic traffic simulation using SUMO. In *2018 21st international conference on intelligent transportation systems (ITSC)* (2018), IEEE.

[27] MILLER, C., AND VALASEK, C. Remote exploitation of an unaltered passenger vehicle. Black Hat USA, 2015.

[28] MONTEUUIS, J., PETIT, J., ZHANG, J., LABIOD, H., MAFRICA, S., AND SERVEL, A. Attacker model for connected and automated vehicles. In *Proc. of ACM Computer Science in Cars Symposium (CSCS)* (2018).

[29] MONTEUUIS, J.-P., PETIT, J., CHEN, C., DAS, S., NEKOUI, M., AND YANG, S. V2X misbehavior in decentralized notification basic service: Considerations for standardization. In *Proceedings of the 2024 Cyber Security in CarS Workshop* (2024).

[30] NIE, S., LIU, L., AND DU, Y. Free-fall: Hacking tesla from wireless to can bus. Black Hat USA, 2017.

[31] PETIT, J., AND SHLADOVER, S. E. Potential cyberattacks on automated vehicles. *IEEE Transactions on Intelligent transportation systems 16*, 2 (2015).

[32] POLLICINO, F., STABILI, D., BELLA, G., AND MARCHETTI, M. Sixpack: Abusing abs to avoid misbehavior detection in vanets. In *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)* (2021), IEEE.

[33] RAYA, M., AND HUBAUX, J.-P. The security of vehicular ad hoc networks. In *Proc. of the 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)* (2005).

[34] RAYA, M., PAPADIMITRATOS, P., AAD, I., JUNGELS, D., AND HUBAUX, J.-P. Eviction of misbehaving and faulty nodes in vehicular networks. *IEEE Journal on Selected Areas in Communications 25*, 8 (2007).

[35] SAE INTERNATIONAL. On-board system requirements for V2V safety communications. Tech. Rep. J2945/1_202004, SAE International, 2020.

[36] SCHMIDT, R. K., LEINMÜLLER, T., SCHOCH, E., HELD, A., AND SCHÄFER, G. Vehicle behavior analysis to enhance security in vanets. In *Proceedings of the 4th IEEE Vehicle-to-Vehicle Communications Workshop (V2VCOM2008)* (2008).

[37] SEDAR, R., KALALAS, C., VÁZQUEZ-GALLEGO, F., ALONSO, L., AND ALONSO-ZARATE, J. A comprehensive survey of v2x cybersecurity mechanisms and future research paths. *IEEE Open Journal of the Communications Society 4* (2023).

[38] SHARMA, A., AND JAEKEL, A. Machine learning based misbehaviour detection in vanet using consecutive bsm approach. *IEEE Open Journal of Vehicular Technology 3* (2022).

[39] SO, S., PETIT, J., AND STAROBINSKI, D. Physical layer plausibility checks for misbehavior detection in v2x networks. In *Proceedings of the 12th conference on security and privacy in wireless and mobile networks* (2019).

[40] SPEIRAN, J., AND SHAKSHUKI, E. Emergency electronic brake lights for the smartphone VANET. *Procedia Computer Science 203* (2022).

[41] STÜBING, H., FIRL, J., AND HUSS, S. A. A two-stage verification process for car-to-x mobility data based on path prediction and probabilistic maneuver recognition. In *2011 IEEE Vehicular Networking Conference (VNC)* (2011), IEEE.

[42] TANG, Z., SERAG, K., ZONOUZ, S., CELIK, Z. B., XU, D., AND BEYAH, R. Eracan: Defending against an emerging can threat model. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (2024).

[43] TREIBER, M., HENNECKE, A., AND HELBING, D. Congested traffic states in empirical observations and microscopic simulations. *Physical review E 62*, 2 (2000).

[44] UPSTREAM SECURITY. 2025 global automotive and smart mobility cybersecurity report. Tech. rep., Upstream Security Ltd., Feb. 2025.

[45] VAN DER HEIJDEN, R. W., DIETZEL, S., LEINMÜLLER, T., AND KARGL, F. Survey on misbehavior detection in cooperative intelligent transportation systems. *IEEE Communications Surveys & Tutorials 21*, 1 (2019).

[46] VAN DER HEIJDEN, R. W., LUKASEDER, T., AND KARGL, F. Veremi: A dataset for comparable evaluation of misbehavior detection in vanets. In *International conference on security and privacy in communication systems* (2018), Springer.

[47] VAN DER HORST, R. Time-to-collision as a cue for decision making in braking. *Vision in vehicles* (1991).

[48] WERLING, C., KÜHNAPFEL, N., JACOB, H. N., AND DROKIN, O. Jailbreaking an electric vehicle in 2023 or what it means to hotwire tesla's x86-based seat heater. Black Hat USA, 2023.

[49] WHYTE, W., WEIMERSKIRCH, A., KUMAR, V., AND HEHN, T. A security credential management system for V2V communications. In *Proc. of IEEE Vehicular Networking Conference (VNC)* (2013).

[50] YANG, Z., YANG, K., LEI, L., ZHENG, K., AND LEUNG, V. C. M. Blockchain-based decentralized trust management in vehicular networks. *IEEE Internet of Things Journal 6*, 2 (2019).

[51] YEKE, D., PANT, K. A., OZMEN, M. O., KIM, H., GOPPERT, J. M., HWANG, I., BIANCHI, A., AND CELIK, Z. B. Automated discovery of semantic attacks in Multi-Robot navigation systems. In *34th USENIX Security Symposium (USENIX Security 25)* (2025).

[52] ZHANG, Q., JIN, S., ZHU, R., SUN, J., ZHANG, X., CHEN, Q. A., AND MAO, Z. M. On data fabrication in collaborative vehicular perception: Attacks and countermeasures. In *33rd USENIX Security Symposium (USENIX Security 24)* (2024).

[53] ZOCCOLI, G. G., POLLICINO, F., STABILI, D., AND MARCHETTI, M. Sixpack v2: enhancing sixpack to avoid last generation misbehavior detectors in vanets. In *2022 IEEE 21st International Symposium on Network Computing and Applications (NCA)* (2022), IEEE.

A Artifact Appendix

A.1 Abstract

This artifact contains the source code, install pipeline, experimental scenarios, reference results, and automated claim verifier for MIRAGE, an event-gated misbehavior detector for fake Emergency Electronic Brake Light (EEBL) attacks in Vehicle-to-Everything (V2X) networks. From a fresh Ubuntu 20.04 host, a single command installs all dependencies (OMNeT++ 5.6.2, SUMO 1.8.0, Veins 5.2, VASP), compiles MIRAGE, and produces a ready-to-run environment; a smoke test, single-seed reproduction, and an automated verifier cover the path from installation to claim validation in under two hours on commodity hardware.

It reproduces the paper's five quantitative claims: aggregate effectiveness ($F_1 = 0.922$, TPR $= 0.883$, FPR $= 0.009$, Precision $= 0.965$), balanced per-attack coverage (77.0 % on $\mathcal{A}_1$, 99.7 % on $\mathcal{A}_2$), superiority over three baselines (Threshold, VCADS, F2MD), threshold robustness across $\Theta \in [0.45, 0.59]$, and IDM mitigation reducing mean braking severity by approximately 75 %. It was verified end-to-end in three clean-room runs on a fresh `ubuntu:20.04` container, all producing 4/4 PASS within 1 h 55 min.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact is purely simulation-based: it does not interact with any real vehicle, live network, or physical radio, does not disable any host security mechanism, and writes only under `/opt/mirage` and the scenario directory. Reviewers preferring stricter isolation can run it inside an `ubuntu:20.04` Docker container, as the authors did for clean-room verification.

A.2.2 How to access

- **Archival DOI (recommended):** https://doi.org/10.5281/zenodo.19639252 (Zenodo concept DOI; always resolves to the latest published version).
- **Version DOI of this submission:** https://doi.org/10.5281/zenodo.20398530 (immutable snapshot, version 2.1). The archived zip has MD5 `a0633472ad2c33a39fa4a3914dadca99` and size approximately 6.7 MB. It contains the source code, reference figures and tables, and the reference verifier outputs; raw simulation logs are *not* included and are not required, as every claim is reproducible from a fresh install.
- **GitHub (working copy, for AE phase only):** https://github.com/eunhanka/mirage-eebl-detector, branch `artifact-evaluation`, tag `vehiclesec26-ae-v2.1` (commit `2981dd0`).
- **License:** MIT, consistent with the upstream VASP project.

A.2.3 Hardware dependencies

Commodity x86_64 workstation; minimum 4 CPU cores, 8 GB RAM, 20 GB free disk; recommended 16 cores, 16 GB RAM. No GPU, FPGA, or specialized hardware is required. The artifact was verified on a 16-core Ubuntu 20.04 host.

A.2.4 Software dependencies

Host OS: **Ubuntu 20.04 LTS**. All dependencies are pinned and installed automatically by `scripts/install.sh`: GCC 9.4.0 (Ubuntu 20.04 default); OMNeT++ 5.6.2 (upstream tarball); SUMO 1.8.0 (git tag `v1_8_0`); Veins 5.2 (tag `veins-5.2`, commit `c5b4d7c4`); VASP (git commit `0ec4af32`); Python 3.8 with `numpy`, `pandas`, `matplotlib`, `scikit-learn` (via `pip`); Qt5, Boost, Xerces-C (via `apt`). Network access is required during installation for `apt`, `pip`, and `git` against `github.com`; air-gapped evaluation is not supported without prior source staging.

A.2.5 Benchmarks

The evaluation uses the synthetic scenario of Section 7 of the paper: a single-lane highway with 20 SUMO-controlled vehicles over 120 s, governed by the SUMO IDM model with driver imperfection $\sigma = 0.5$, speed deviation ± 10 %,

initial speeds 25 ± 2 m/s, and inter-vehicle gaps 30 ± 3 m. Five random seeds $\{0,1,2,3,4\}$ are used for the aggregate results. Per-seed traffic variations are generated at runtime by `generate_route_variants.py`, which produces `highway_seed{0..4}.rou.xml` inside the scenario directory. `run_single_seed.sh` invokes this generator automatically on first use; reviewers running `run_multi_seed.sh` as a standalone entry point must invoke it once beforehand (see experiment E1). Attacker behavior is the unmodified VASP implementation of `FakeEEBLJustAttack` ($\mathcal{A}_1$, NoStop) and `FakeEEBLStopPositionUpdateAfterAttack` ($\mathcal{A}_2$, WithStop). No external dataset is required.

A.3 Set-up

From a fresh Ubuntu 20.04 host, cloning the `artifact-evaluation` branch and running `sudo ./scripts/install.sh all` builds the full pinned stack through five idempotent stages (approximately 7 min on 16 cores, 25 to 30 min on 4 cores; any stage is re-runnable). A smoke test, `./scripts/quick_test.sh`, then runs MIRAGE against $\mathcal{A}_2$ for 120 s and, on success, reports 327868 total and 96990 suspicious rows and exits 0. Exact commands, expected console output, and troubleshooting are in the repository `README.md`.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): MIRAGE achieves $F_1 = 0.922$ at $\Theta = 0.55$ (TPR $=$ 0.883, FPR $= 0.009$, Precision $= 0.965$), averaged over five seeds (per-metric SD below 0.01; tolerance bands in `CLAIMS.md`). Proven by (E1); paper Table 4.

(C2): MIRAGE is the only detector sustaining high TPR on both variants (0.770 on $\mathcal{A}_1$, 0.997 on $\mathcal{A}_2$); no baseline exceeds TPR $= 0.25$ on both. Proven by (E1); paper Figure 5.

(C3): MIRAGE outperforms three baselines: Threshold ($F_1 = 0.733$), VCADS (0.607), F2MD/VASP (0.193). Proven by (E1); paper Table 4.

(C4): Detection is stable across $\Theta \in [0.45, 0.59]$ ($F_1 > 0.92$), with a sharp transition at $\Theta = 0.60$ (TPR drops from 0.883 to 0.41). Proven by (E2); paper Figure 6.

(C5): The IDM mitigation layer bounds ego braking to $-4.0\,\mathrm{m/s^2}$, reducing mean braking severity by approximately 75 % (mean bounded deceleration approximately $-2.03\,\mathrm{m/s^2}$). Proven by (E1) via the `mitigated_a` column; paper Table 5.

A.4.2 Experiments

The full reviewer path takes approximately 1 h 55 min on a 16-core host; a minimum-validation path (install, smoke test, E1, verifier) completes in approximately 30 min.

(E1): *Single- or five-seed reproduction.* `run_single_seed.sh 0` followed by `verify_results.py --seeds 0` runs the 18 detector-by-attack configurations and reports a PASS/FAIL verdict against the tolerance bands in `CLAIMS.md`, validating (C1), (C2), (C3), (C5); the five-seed protocol (`run_multi_seed.sh 0 1 2 3 4`, approximately 80 min more) reproduces the paper averages.

(E2): *Threshold sensitivity.* `analyze_multi_seed.py results/` sweeps Θ over the E1 logs without recompilation and regenerates Figure 6, validating (C4).

Exact invocations, expected verifier output, per-claim tolerance bands, and the reference figures/tables for diff-based comparison are documented in `CLAIMS.md`, `reference_results/README.md`, and the repository `docs/`.

A.5 Notes on Reusability

The five installation stages run independently, and the detector factory is in `src/vasp/mdm/MisbehaviorDetectors.h` exposes a runtime-configurable `detectorType`; adding a detector needs only a `BaseDetector` subclass and a `case` in `createDetector()` (log schema, verifier, and figure pipeline unchanged), and adding a claim is a one-entry change in the `CLAIMS` table of `verify_results.py`.

WIP: On the Real-World Applicability of Automotive CAN Intrusion Detection Systems

Kristijan Lazeski[1,2], Michael Zohner[4], Jannis Hamborg[1,2], Christoph Krauß[1,2,3], and Alexander Wiesmaier[1,2,3]

[1]Darmstadt University of Applied Sciences, Germany
[2]European University of Technology, European Union
[3]National Research Center for Applied Cybersecurity ATHENE, Germany
[4]University of Applied Sciences Fulda, Germany

Abstract

Machine Learning (ML)-based Controller Area Network (CAN) Intrusion Detection Systems (IDS) are widely studied and used to detect and mitigate attacks on in-vehicle communication. Despite frequent reports on high detection performance for well-established datasets, the practical applicability and robustness against real-world attackers of the trained solutions is rarely discussed. This work presents a preliminary analysis of the practical robustness of ML-based CAN IDS solutions, with a focus on per-frame, binary intrusion detection approaches. Our approach consists of three steps: (1) We analyze common datasets used for training and evaluating ML-based CAN IDS, (2) train human-interpretable decision trees on the datasets and investigate their learned classification rules[1], and (3) critically evaluate the models against a practical attacker model. We observe that many commonly used CAN IDS datasets only include simple and easily distinguishable attacks. Consequently, ML-based models trained on these datasets tend to learn dataset-specific statistical artifacts or shortcuts rather than robust or semantically meaningful detection logic, a finding we demonstrate specifically within the scope of per-frame, binary detection. While our results are preliminary and limited to this detection paradigm, they show that high detection performance is not inherently indicative of practical robustness. We therefore argue that new detection strategies founded upon vehicle specifications and domain knowledge, in combination with ML tools, may offer more transparent and reliable protection than purely ML-driven approaches, and we call for broader evaluation across more diverse detection settings in future work.

Intrusion Detection System, Automotive, CAN, Machine Learning, Datasets

1 Introduction

The Controller Area Network (CAN) bus is the central technology for in-vehicle communication between Electronic Control Units (ECUs), but was designed without encryption or authentication, making it a well-known attack vector [5, 19, 32]. Numerous machine Learning (ML)-based CAN Intrusion Detection Systems (IDSs) have been proposed to detect attacks such as Denial of Service (DoS), fuzzing, and spoofing [26], with many reporting precision and recall near or at 100% [6, 10, 11, 15–17, 20–22, 24, 36, 37, 41, 46]. However, it remains unclear whether these results reflect models learning robust behavioral invariants or merely exploiting dataset-specific artifacts.

In this paper, we present an initial study aimed at answering this question. We systematically analyze five widely used CAN IDS datasets [10, 16, 21, 22, 41], examining their structural properties and the classification rules learned by interpretable models to assess whether observed separability reflects genuine attack characteristics or dataset-induced biases. Our analysis follows a two-step approach. First, we examine the distinguishing characteristics between benign and malicious traffic, where a *trace* is a recorded CAN message with a timestamp and a benign or malicious label. Second, we train a Decision Tree (DT) model per dataset and analyze its learned classification rules. DTs are chosen for their transparency, enabling direct inspection of which features drive classification [12]. We note that this study is scoped to per-frame, binary intrusion detection, and our findings should be understood within that limitation. Our preliminary findings indicate that many attacks in the datasets analyzed can be distinguished from benign traffic based on features unique to the dataset-specific attack implementation rather than inherent properties of malicious CAN messages. Although our DT models often achieve a precision and recall of over 99%, an analysis of the learned rules shows that they are based on simple features that are easy to circumvent. An attacker who slightly adapts timing or payload values can evade such an IDS without fundamentally changing the attack strategy. These observations raise broader questions about current evaluation methodology that warrant further investigation. In practice, manufacturers often deploy rule-based IDS that explicitly encode domain knowledge such as expected message frequencies, signal ranges, and

[1]https://tudatalib.ulb.tu-darmstadt.de/handle/tudatalib/5167

ECU communication patterns. ML-based IDS, by contrast, are evaluated on publicly available datasets using standard metrics. If these datasets contain artifacts or unrealistic attack characteristics, models may learn shortcuts rather than robust security-relevant properties and reported high performance may not translate into real-world protection. This preliminary study has three main contributions: (1) a systematic analysis of popular CAN IDS datasets to assess the separability of malicious from benign traffic, (2) an investigation into features learned by ML-based CAN IDS using interpretable DT models, and (3) a demonstration that many learned rules are fragile and easily bypassed under realistic attacker models. From these findings, we derive implications for dataset design and outline challenges for more robust evaluation and deployment of ML-based CAN IDS.

The remainder if this paper is structured as follows: Section 2 presents background and related work (including a typical attacker model assumed in our study). Our study is described in Section 3. Finally, we conclude the paper and discuss future work in Section 4.

2 Background and Related Work

2.1 Controller Area Network (CAN)

Introduced in the 1980s [4], the CAN bus remains a core technology for in-vehicle communication, enabling deterministic, fault-tolerant, real-time data exchange between ECUs at low cost. While newer protocols such as Automotive Ethernet [33] are gaining ground for high-bandwidth applications, CAN continues to dominate safety-critical domains such as braking systems [28]. This work focuses exclusively on classical CAN, acknowledging but excluding CAN FD and other protocol variants.

2.1.1 CAN Frame Structure

CAN encapsulates data in frames, most commonly the *Base Frame Format* (11-bit identifier) or the *Extended Frame Format* (29-bit identifier), consisting of the following fields (cf. Figure 1): Start of Frame (SOF), CAN ID / arbitration field, Remote Transmission Request (RTR) bit, Control field (Identifier Extension (IDE) bit and Data Length Code (DLC)), data payload (0–8 bytes), CRC, Acknowledgment field, and End of Frame (EOF) [4]. CAN uses a multi-master architecture in which bus conflicts are resolved via *non-destructive bitwise arbitration*: each transmitting node continuously monitors the bus and relinquishes transmission if it detects a dominant bit (logical '0') while asserting a recessive bit (logical '1'), thereby granting bus access to the frame carrying the lowest CAN ID without loss of data or time.

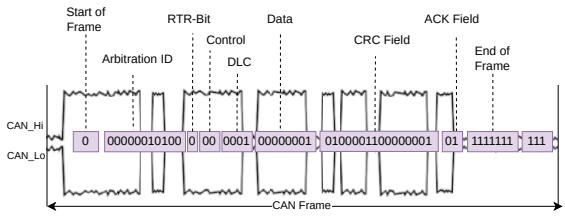


Figure 1: The structure of a CAN frame.

2.1.2 CAN Data Interpretation and Frame Timing

The 8-byte CAN payload is interpreted through signals defined in a Database CAN (DBC) file [7, 8], which is authored by the vehicle manufacturer and distributed to subcontractors for ECU configuration. Each signal definition specifies its bit position and length, endianness, a linear scaling function (scale and offset), a valid value range, and an optional unit. Figure 2 shows an exemplary DBC entry from the OpenDBC project[2]: the steering angle is encoded in bits 0–15 (scaled by 0.0428316886, Most Significant Bit (MSB) as sign, valid range −600 to 600), a counter in bits 20–23, and a complement value in bits 24–34, illustrating that only a subset of all bit-encodable values carries semantic meaning.

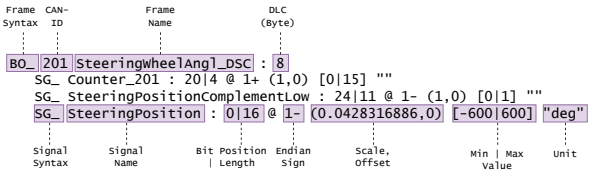


Figure 2: Example DBC entry for a BMW E9x/E8x.

Beyond payload structure, CAN frame timing follows three distinct patterns [31, 47]: safety-critical frames are transmitted periodically with tight, well-defined intervals and minimal jitter [2, 47]; non-critical synchronization frames are periodic but tolerate higher jitter [2]; and event-driven frames (e.g., door events, diagnostics) are transmitted only upon specific triggers and may be absent during normal operation [31, 43]. Both signal definitions and timing specifications are proprietary and not publicly disclosed, making it necessary for academic CAN IDS research to reverse-engineer this information from observed traffic or available datasets [10, 16, 21, 22, 41].

2.2 CAN bus Attacks

Attacker Model. Following Hamborg et al. [9], we distinguish two attacker classes. A *standard-compliant attacker* operates within the CAN protocol specification and, after gaining electrical bus access (e.g., via OBD-II or a compromised ECU), can passively observe traffic, inject crafted

[2] https://github.com/commaai/opendbc

frames, replay recorded messages, and generate DoS conditions, but cannot modify frames already in transmission. A *physical-layer attacker* additionally manipulates the electrical signaling of the bus, enabling suppression or corruption of legitimate frames beyond protocol-compliant injection, for instance by forcing dominant states or pushing ECUs into error-passive or bus-off conditions. In this work, we assume an adversary with physical bus access who can observe, inject, and replay CAN frames and adapt injected messages based on observed CAN IDs, payload ranges, and timing behavior, without requiring cryptographic keys or proprietary specifications.

Attack Types. The assumed attacker model enables several well-established attack classes [39]. **DoS** attacks exploit CAN's priority-based arbitration by flooding the bus with high-priority frames, starving lower-priority communication. **Fuzzing** injects random or malformed frames to trigger unintended ECU behavior or expose implementation weaknesses. **Spoofing** exploits the absence of sender authentication to impersonate legitimate ECUs by injecting fabricated frames with valid CAN IDs, enabling targeted manipulation of vehicle functions. **Replay attacks** retransmit previously recorded valid frames to reproduce specific actions such as door unlocking or component activation. Attackers with additional physical-layer capabilities can further perform selective frame suppression, arbitration manipulation, and fault injection, allowing not only malicious traffic injection but also targeted shaping or suppression of legitimate communication.

2.3 Machine Learning

ML has become the dominant approach for intrusion detection due to its ability to learn behavioral characteristics directly from data, where traditional rule-based systems fail against unseen attack patterns [25,45]. Detection performance is strongly dependent on dataset quality and representativeness [40], and high numerical results alone do not guarantee robustness. Explainable AI (XAI) and interpretable models are therefore critical for bridging automated detection and human understanding [30].

Decision Trees (DTs). We employ DTs [44] as interpretable proxies to examine which features drive classification decisions. Following Quinlan [34], DTs recursively partition data by maximizing information gain via Shannon entropy until a stopping criterion is met. Crucially, dataset artifacts or trivially separable features that dominate DT decisions are likely to be exploited equally, or more aggressively, by opaque models such as random forests or neural networks, which may achieve even higher numerical performance without learning more semantically meaningful or robust rules.

Evaluation Metrics. We address binary classification between benign (negative) and malicious (positive) traffic, with True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN) defined accordingly. Since the analyzed datasets are highly class-imbalanced, with malicious traces often constituting only a small fraction of all data, accuracy alone is insufficient, as a trivial majority-class predictor would appear performant. We therefore evaluate using False Positive Rate (FPR), False Negative Rate (FNR), precision, recall, and F1-score [3, 27, 29].

2.4 Related Work

Several surveys structure the CAN IDS landscape: Karopoulos et al. [18] and Rajapaksha et al. [35] both highlight inconsistencies in evaluation practices and dataset realism, while Sharmin et al. [38] emphasize the lack of standardized benchmarking protocols. Stabili et al. [42] and Abgaje et al. [1] provide structured evaluations of detection methods, with the latter reporting strong ML performance on the Car Hacking dataset [41], yet neither investigates whether this performance stems from robust behavioral modeling or dataset-specific artifacts. Jacobs et al. [12] raise analogous concerns for network-based IDS, arguing that many reported results rely on unrealistic assumptions; we transfer this critical perspective directly to the CAN domain. On dataset quality, Lee et al. [23] conclude that no existing dataset fully satisfies representativeness and diversity requirements, and Lampe et al. [21] propose improved construction methodologies, yet neither examines how dataset properties shape learned classification rules. The Car Hacking Competition [16] further corroborates our findings: the best-performing final-round solution was rule-based, suggesting that semantic signal understanding may outweigh statistical learning.

In contrast to prior work, we neither propose a new detection method nor a new dataset. Instead, we systematically analyze five widely used CAN IDS datasets through the lens of interpretable ML, examining what decision tree models actually learn and evaluating learned rules under a realistic attacker model, a gap that performance-focused surveys and benchmarks have not addressed.

3 Analysis of Datasets for ML-Based CAN IDS

We assess the practical robustness of ML-based CAN IDS trained on commonly used datasets, analyzing dataset properties (Section 3.2), empirically verifying findings via DTs (Section 3.3), evaluating robustness (Section 3.4), and discussing implications (Section 3.5). Our goal is not to propose a novel IDS, but to understand what drives near-perfect reported performance and whether learned rules remain meaningful under the attacker model described in Section 2.2.

3.1 Methodology and Limitations

Our methodology proceeds in two steps. First, we analyze widely used CAN IDS datasets (cf. Section 3.2) with respect to the assumed attacker model (cf. Section 2.2), examining trace organization, recorded fields, attack strategies, class distribution, and CAN-ID–payload overlap between benign and malicious traffic. Second, DTs are trained on these datasets (cf. Section 3.3) and their learned classification logic is inspected to determine whether decision rules reflect robust behavioral constraints or dataset-specific regularities.

Limitations.

1. **Decision Trees Only.** We restrict our empirical evaluation to DTs, which provide native interpretability and directly expose structural shortcuts in the data. While tools such as Trustee [12] can convert complex models into interpretable surrogates, tree-based models achieve performance comparable to deep learning in CAN IDS settings [16]. Since all supervised models trained on the same data share the same feature distributions and separability properties, DT analysis generalizes qualitatively, though subtle differences in learned boundaries across model types cannot be excluded.

2. **Binary Classification.** Models are trained separately per attack type to isolate decision criteria and prevent dominant attack patterns from masking subtler ones. Mixed-attack training can obscure whether performance is driven by robust behavioral properties or a single easily separable class; single-attack models serve as a controlled baseline.

3. **Scope of Detection Paradigm.** This study is scoped exclusively to per-frame, binary intrusion detection, in which individual CAN frames are classified as either benign or malicious.

3.2 Dataset-Level Structural Analysis

We analyze five publicly available CAN IDS datasets, Car Hacking [41], OTIDS [22], Survival [10], CT&T [21], and Car Hacking Attack & Defense [16], covering DoS, fuzzing, spoofing, and replay attacks. We focus on whether malicious messages share CAN IDs and payload patterns with benign traffic, since attack messages whose CAN ID and payload combinations never occur in benign traffic are trivially easy to classify. Benign traffic is concentrated on a fixed set of CAN IDs with characteristic payload distributions, reflecting regular ECU communication (cf. Figure 3). Dataset properties are summarized in Appendix Table 3, and empirical results are reported in Table 1.

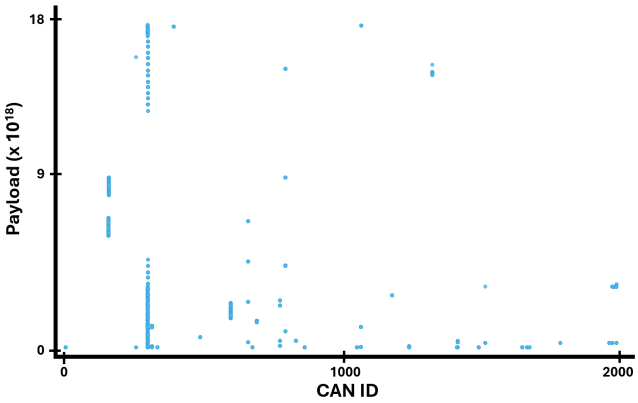
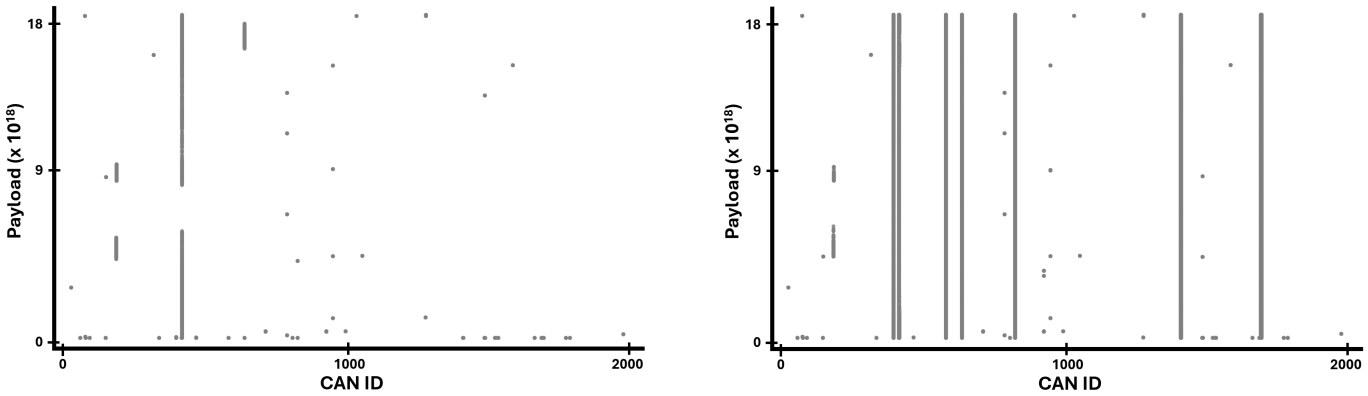


Figure 3: Distribution of benign traces for Car Hacking dataset on different CAN IDs (x-axis) and payload (y-axis).

DoS Attacks. Across all datasets, DoS attacks use a small number of static, uniquely identifiable combinations of CAN ID and payload that do not occur in normal traffic, making them trivially easy to detect. Car Hacking [41] (3 078 250/587 521), Survival [10] (148 760/33 141), OTIDS [22] (320 957/335 623), and Car Hacking Challenge [16] (3 207 280/171 824) all use CAN ID `0x000` with an all-zero payload, which is absent from benign traffic. CT&T [21] (2 246 099/51 798) deviates slightly, using CAN IDs `0x000` and `0x002` with three distinct constant payload variants, likewise absent from benign traffic. In all cases, detecting DoS attacks reduces to identifying a fixed injection pattern. Under the attacker model, however, an adversary can preserve the DoS effect while selecting a different high-priority CAN ID or varying the payload, thereby bypassing any detector that relies on static signatures.

Spoofing Attacks. Spoofing attacks are also implemented as fixed patterns consisting of CAN IDs and payload data that are not present in normal traffic, meaning that classification relies more on memorization than on the detection of semantic inconsistencies. In Car Hacking [41], gear spoofing (3 845 890/597 252) uses CAN ID `1087` with a static payload, and RPM spoofing (3 966 805/654 897) injects a single fixed payload for the targeted CAN ID. In Survival [10] (166 035/7 401), the malfunction attack targets CAN ID `0x153` with payloads outside the benign range. OTIDS [22] modifies RTR frames for CAN ID `0164`: benign requests carry no payload (DLC=0), injected frames carry 8 bytes (DLC=8). In CT&T [21], speed spoofing targets CAN ID `212` with three fixed payload values encoding 0,mph, 10,mph, and 60,mph, none of which appear in the benign traces for that CAN ID; script-based attacks (interval: `0x0D4`/`3501340134013501`; standstill: `0x161`/`2823FFFF002000F0`) and triple-spoofing follow the same static pattern structure. The Car Hacking Challenge [16] uses 2 patterns in training (1 648 114/3 877), 3



(a) Distribution of traces in the benign file.

(b) Distribution of traces in the fuzzing file.

Figure 4: Trace distributions in the benign and fuzzing files of OTIDS [22] (CAN ID on x-axis, payload on y-axis). Dots are grey as frames are unlabeled and thus indistinguishable.

in submission (1 559 166/20 093), and 4 in the final round (1 090 312/149 999), with one final-round pattern overlapping benign traffic. Under the attacker model, an adversary can observe valid payload ranges and inject within-range values, bypassing detectors that rely on fixed static signatures.

Fuzzing Attacks. Fuzzing attacks inject random CAN IDs and payloads, deliberately producing CAN ID and payload combinations that also occur in normal traffic, which forces models to rely on secondary artifacts such as recording structure or temporal boundaries rather than the content of individual frames. In Car Hacking [41] (3 347 013/491 847), eleven benign CAN IDs appear exclusively in the fuzzing file, likely triggered as ECU responses to injected frames. The benign-only file therefore does not cover all observable CAN IDs, risking misclassification of legitimate condition-triggered traffic at deployment. Survival [10] (210 178/39 812) follows the same principle without this discrepancy. OTIDS [22] targets nine CAN IDs (`153`, `164`, `1F1`, `220`, `2C0`, `4B0`, `4B1`, `5A0`, `5A2`) also present in benign traffic (cf. Figure 4). For IDs with near-constant benign payloads (e.g., `5A0`) fuzzed frames are visually distinguishable, while for variable IDs (e.g., `164`) they are not. Without frame-level labels, OTIDS does not support principled per-frame fuzzing detection. CT&T [21] provides standard fuzzing (2 413 372/37 391) with random CAN IDs and payloads, and systematic fuzzing (3 374 832/23 814) whose temporally structured progression introduces strong time-locality that enables models to exploit absolute timestamp boundaries. The Car Hacking Challenge [16] injects all possible payloads only for CAN IDs present in benign traffic (training: 1 648 112/44 405, submission: 1 559 163/51 923, final: 1 090 311/29 999).

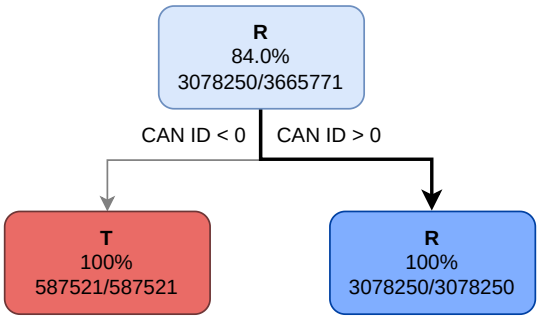
Replay. Replay attacks appear only in the Car Hacking Challenge [16]. Malicious traces consist of replayed benign messages. They fall within normal CAN-ID–payload value ranges and are timestamped consistently with the overall communication flow, making them difficult to distinguish. The training file contains 1 468 112/23 818 and the submission file (the only test split) 1 559 163/25 012 benign/malicious traces.

Dataset-Level Structural Anomalies and Artifacts. Several irregularities warrant attention. In Car Hacking [41], eleven benign CAN IDs appear only in the fuzzing file, meaning the benign recording does not fully represent observable CAN traffic. OTIDS [22] lacks frame-level labels in attack files, preventing reliable per-frame supervised learning. Several datasets exhibit strong class imbalance — DoS in CT&T constitutes 2% of traces, in Survival 4% — which can inflate accuracy metrics by rewarding majority-class predictions. Finally, attack scenarios in several datasets form contiguous time blocks with absolute timestamps, making temporal boundaries directly visible and constituting recording artifacts unrelated to vehicle behavior.

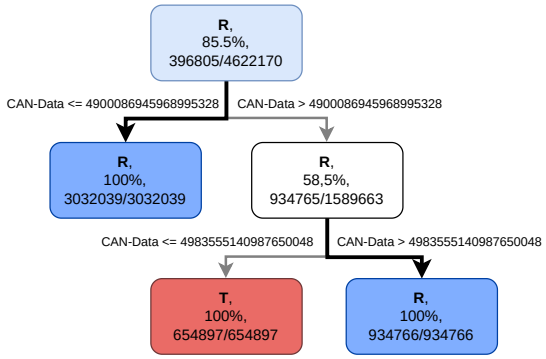
3.3 Decision Tree Training and Evaluation

We train DTs on all datasets except OTIDS, whose attack files lack frame-level labels. Prior work [13] incorrectly labeled all frames in these files as malicious, yielding no meaningful insight. We extend all datasets with interval features (time since the CAN ID last appeared) and represent payloads both as a single integer and as individual bytes. DTs are trained in Orange [44] with at least two instances per leaf, no splits on fewer than three traces, and a 99% stopping criterion (raised to 100% for strongly imbalanced subsets or fuzzing attacks). Datasets with dedicated test sets (Car Hacking Challenge) are evaluated on provided test files; others use 10-fold random sampling (66%/34%, averaged). Results are in Table 1.

Overall, DT models achieve F1-scores close to or equal to 100% across nearly all datasets and attack types. Lower performance arises where benign and malicious traces over-



(a) DT for the DoS attack, classifying on a static CAN ID unique to malicious traces.



(b) DT for the RPM spoofing attack, classifying on a static payload value unique to malicious traces.

Figure 5: DTs trained on DoS and RPM spoofing in the Car Hacking dataset (blue: benign, red: malicious). Arrow labels indicate decision criteria; box percentages indicate the class ratio.

lap in CAN-ID–payload space or where test files contain spoofing patterns unseen during training. High performance consistently coincides with static, uniquely identifiable attack patterns that would not survive adversarial adaptation.

DoS and Spoofing. Models for DoS and single spoofing attacks achieve 100% across all metrics for nearly every dataset, directly exploiting the unique static patterns identified in the data analysis (cf. Figure 5a). DoS classification relies solely on CAN ID `0x000`, and RPM spoofing (cf. Figure 5b) on a static payload not present in benign traffic. The exception is the Car Hacking Challenge: as the submission and final-round files contain different spoofing patterns than the training file, the model fails to detect any attack, highlighting the dependence on observing the specific pattern during training.

Fuzzing. Fuzzing models achieve F1-scores between 93.7% (CT&T) and 99.8% (Car Hacking and Survival), with tree sizes from 25 nodes (99%-split, Survival) to 389 nodes (100%-split, Car Hacking). Across all DTs, the interval feature is used first, followed by byte-level payload patterns and, less frequently, the CAN ID. Some nodes classify on absolute UNIX timestamps, which we attribute to overfitting on temporal recording boundaries rather than learned behavioral invariants.

Replay. Replay detection yields the poorest performance, ranging from 54.7% (100%-split, submission file) to 76.4% F1-score (100%-split, training file), with the largest models overall (3 809 nodes for 99%-split, 10 707 for 100%-split). This reflects overfitting: no general distinguishing rule can be learned, and unlike fuzzing where the timestamp appears only near the leaves, here it is already used at the second level of the tree.

3.4 Assessment Under the Attacker Model

We reassess the identified dataset characteristics against the attacker model in Section 2.2, examining whether learned decision rules remain robust when an adversary adapts CAN IDs, payload values, or timing behavior.

DoS Attacks. In Car Hacking [41], Survival [10], OTIDS [22], and the Car Hacking Challenge [16], DoS detection reduces to a binary check on the CAN ID `0x000`–all-zero-payload pair. Under the attacker model, an adversary can inject high-frequency frames using a different high-priority CAN ID or a modified payload while preserving the DoS effect, bypassing any detector tied to this static pattern. CT&T's slightly more complex instantiation (CAN IDs `000`/`002` with constant payload variants) likewise falls to an attacker using alternative CAN IDs, since the model only learns the specific observed combinations.

Static Spoofing Attacks. In all evaluated spoofing scenarios, learned rules memorize fixed CAN-ID–payload combinations or value ranges: RPM and gear spoofing in Car Hacking [41] use fixed payloads absent from benign traffic; the malfunction attack in Survival [10] targets payloads outside the benign range; speed spoofing in CT&T [21] uses three static payload values (0 mph, 10 mph, 60 mph); and the Car Hacking Challenge [16] uses 9 distinct static traces with varying patterns across files. Under the attacker model, an adversary can observe valid payload ranges and inject within-range values while still influencing vehicle behavior, rendering all static learned rules ineffective.

Fuzzing and Timestamp Dependence. For fuzzing in Car Hacking [41], Survival [10], CT&T [21], and the Car Hack-

ing Challenge [16], benign and malicious traces overlap in CAN-ID–payload space, so DTs frequently resort to absolute timestamp splits. These effectively encode dataset-specific temporal boundaries (e.g., a specific date in the Car Hacking dataset) rather than behavioral properties. Under the attacker model, an adversary can adapt injection timing to invalidate such splits, causing the IDS to misclassify adapted attacks or mislabel benign traffic recorded outside the original temporal window.

Replay Attacks. Replay attacks re-inject previously observed legitimate frames, meaning replay traffic can be indistinguishable in CAN-ID–payload space from benign operation. ML-based IDS models consequently overfit on timestamps, yielding very poor generalization (cf. Table 1). Under the attacker model, an adversary can replay a carefully selected frame subset and align timing to the current bus schedule, bypassing any detector that relies on recording-specific artifacts rather than semantic consistency.

OTIDS Label Ambiguity. OTIDS [22] attack files mix benign and malicious frames without frame-level labels. Prior work [13] labels entire attack files as malicious, causing models to learn file-level differences rather than frame-level attack characteristics, with timestamps serving as the primary discriminating feature. Under the attacker model, an adversary injecting frames during normal operation would not reproduce file-level structural differences, rendering such detection meaningless.

Training Coverage and Behavioral Completeness. In Car Hacking [41], additional benign CAN IDs appear only in attack files due to ECU responses triggered by injected frames, a behavior typical of event-based signals. If training does not capture the full range of legitimate signals, an IDS may misclassify valid but previously unseen communication. Under the attacker model, an adversary may deliberately trigger such rare legitimate signals to increase false positives or mask malicious activity. Overall, when evaluated against the assumed attacker model, the majority of learned decision rules reduce to recognizing static, dataset-specific artifacts that no adaptive adversary would reproduce.

3.5 Insights and Limitations

The combined analysis reveals three central limitations. First, many datasets implement attacks using static, uniquely identifiable CAN-ID–payload combinations, reducing detection to recognizing dataset-specific constants rather than genuinely malicious behavior. Second, incomplete benign coverage leaves models vulnerable to misclassifying rare but valid signals. Third, file-level separation, timestamp discontinuities, and labeling ambiguities introduce structural artifacts that models exploit as shortcuts, properties no adaptive attacker would reproduce in live vehicle communication.

Although DT-based models achieve between 90% and 100% F1-score across nearly all datasets, inspection of their learned decision logic reveals that this performance largely stems from structural regularities that do not persist under realistic adversarial adaptation. Against an attacker capable of adjusting CAN ID selection, payload values, and timing, many learned rules lose their security relevance, rendering the IDS ineffective against real-world attacks.

4 Conclusion & Future Work

In this paper, we show our analysis of ML-based CAN IDS through dataset structure and model interpretability. Using DTs, we show that widely used benchmark datasets contain systematic artifacts in CAN ID selection, payload values, and timing, allowing models to achieve near-perfect detection by learning dataset-specific regularities rather than robust attack signatures. More fundamentally, training ML models directly on raw CAN traffic is inherently limited: models tend to reverse-engineer static ID and payload ranges already specified in the manufacturer's DBC file, a task more efficiently handled by rule-based pre-filtering. A more effective architecture would apply such pre-filtering first, then train ML models on the residual traffic to target genuinely anomalous behavioral patterns, an approach that also scales better to real-world multi-bus, multi-gateway in-vehicle networks. High benchmark performance therefore does not imply practical security; the bottleneck is dataset realism, not model choice.

Future Work. Future work should establish clear criteria for security-relevant automotive IDS datasets: realistic attacker capabilities and injection strategies, proper frame-level labeling, representative benign coverage across operating conditions, avoidance of structural artifacts such as file-level or timestamp-based separability, and multi-bus topologies reflecting real in-vehicle networks. Only once such criteria are in place can datasets meaningfully support robustness evaluation. Extending this analysis to additional datasets, such as X-CANIDS [14] which may better reflect realistic attack characteristics, represents a natural next step and window based IDS approaches. More broadly, ML-based IDS becomes most valuable in architecturally realistic settings, e.g., multi-bus environments, gateway-level monitoring, cross-domain anomaly correlation, where complex interdependencies are difficult to express manually, requiring a shift from isolated single-bus trace classification toward system-level security modeling.

Acknowledgments

This work was supported by the German Federal Ministry of Research, Technology and Space (BMFTR), under the project

PARFAIT (Grant:16KIS2143).

References

[1] Paul Agbaje, Afia Anjum, Arkajyoti Mitra, Gedare Bloom, and Habeeb Olufowobi. A Framework for Consistent and Repeatable Controller Area Network IDS Evaluation. In *Proceedings Fourth International Workshop on Automotive and Autonomous Vehicle Security*, San Diego, CA, USA, 2022. Internet Society.

[2] Zixiang Bi, Guoai Xu, Guosheng Xu, Miaoqing Tian, Ruobing Jiang, and Sutao Zhang. Intrusion detection method for in-vehicle can bus based on message and time transfer matrix. *Security and Communication Networks*, 2022(1):2554280, 2022.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, USA, 2006.

[4] R. Bosch. Controller area network (can): An in-vehicle serial communication protocol. In *Proceedings of the SAE International Congress and Exposition*, number SAE Technical Paper 860391, Detroit, Michigan, USA, 1986. SAE International.

[5] Mehmet Bozdal, Mohammad Samie, and Ian Jennions. A survey on can bus protocol: Attacks, challenges, and potential solutions. In *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*, pages 201–205, 2018.

[6] G. Dupont et al. Automotive controller area network (can) bus intrusion dataset v2, 2019.

[7] Daniel Frassinelli, Sohyeon Park, and Stefan Nürnberger. I know where you parked last summer: Automated reverse engineering and privacy analysis of modern cars. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1401–1415. IEEE, 2020.

[8] Robert Bosch GmbH. Can specification. 1991.

[9] Jannis Hamborg, Timm Lauser, and Christoph Krauß. RollCAN – CAN-bus based RollJam-Attack. CSCS '25, New York, NY, USA, 2025. Association for Computing Machinery.

[10] Mee Lan Han, Byung Il Kwak, and Huy Kang Kim. Anomaly intrusion detection method for vehicular networks based on survival analysis. *Vehicular communications*, 14:52–63, 2018.

[11] Markus Hanselmann, Thilo Strauss, Katharina Dormann, and Holger Ulmer. Canet: An unsupervised intrusion detection system for high dimensional can bus data. *Ieee Access*, 8:58194–58205, 2020.

[12] Arthur S. Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A. Ferreira, Arpit Gupta, and Lisandro Z. Granville. AI/ML for Network Security: The Emperor has no Clothes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, pages 1537–1551, New York, NY, USA, 2022. Association for Computing Machinery.

[13] Abdul Rehman Javed, Saif Ur Rehman, Mohib Ullah Khan, Mamoun Alazab, and Thippa Reddy G. CANintelliIDS: Detecting In-Vehicle Intrusion Attacks on a Controller Area Network Using CNN and Attention-Based GRU. *IEEE Transactions on Network Science and Engineering*, 8(2):1456–1466, April 2021.

[14] Seonghoon Jeong, Sangho Lee, Hwejae Lee, and Huy Kang Kim. X-canids: Signal-aware explainable intrusion detection system for controller area network-based in-vehicle network. *IEEE Transactions on Vehicular Technology*, 73(3):3230–3246, 2024.

[15] Christian Kaiser, Alexander Stocker, and Andreas Festl. Automotive can bus data: An example dataset from the aegis big data project. *OpenAIRE*, 2019.

[16] Hyunjae Kang, Byung Il Kwak, Young Hun Lee, Haneol Lee, Hwejae Lee, and Huy Kang Kim. Car hacking: Attack & defense challenge 2020 dataset. *Third International Workshop on Automotive and Autonomous Vehicle Security*, 2021.

[17] Min-Joo Kang and Je-Won Kang. Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security. *PLOS ONE*, 11(6):e0155781, June 2016.

[18] Georgios Karopoulos, Georgios Kambourakis, Efstratios Chatzoglou, José L Hernández-Ramos, and Vasileios Kouliaridis. Demystifying in-vehicle intrusion detection systems: A survey of surveys and a meta-taxonomy. *Electronics*, 11(7):1072, 2022.

[19] Mohamed Karrouchi et al. Analysis of current attacks on the can bus and countermeasures. In *2023 ICEGC Conference Proceedings*, 2023. Accessed: 2025-11-04.

[20] C. Lab. Datasets. https://www.crysys.hu/research/vehicle-security/, 2017. Accessed: 2024-10-10.

[21] Brooke Lampe and Weizhi Meng. can-train-and-test: A Curated CAN Dataset for Automotive Intrusion Detection, August 2023. arXiv:2308.04972 [cs].

[22] Hyunsung Lee, Seong Hoon Jeong, and Huy Kang Kim. Otids: A novel intrusion detection system for in-vehicle network by using remote frame. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 57–5709. IEEE, 2017.

[23] Seyoung Lee, Wonsuk Choi, Insup Kim, Ganggyu Lee, and Dong Hoon Lee. A Comprehensive Analysis of Datasets for Automotive Intrusion Detection Systems. *Computers, Materials & Continua*, 76(3):3413–3442, 2023.

[24] Seyoung Lee, Hyo Jin Jo, Aram Cho, Dong Hoon Lee, and Wonsuk Choi. Ttids: Transmission-resuming time-based intrusion detection system for controller area network (can). *IEEE Access*, 10:52139–52153, 2022.

[25] Hongyu Liu and Bo Lang. Machine learning and deep learning methods for intrusion detection systems: A survey. *Applied Sciences*, 9(20):4396, 2019.

[26] Yangyang Liu, Lei Xue, Sishan Wang, Xiapu Luo, Kaifa Zhao, Pengfei Jing, Xiaobo Ma, Yajuan Tang, and Haiying Zhou. Vehicular intrusion detection system for controller area network: A comprehensive survey and evaluation. *IEEE Transactions on Intelligent Transportation Systems*, 26(7):10979–11009, 2025.

[27] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, USA, 1997.

[28] Alberto Morato and Federico Tramarin. Bridging real-time can networks with in-vehicle single pair ethernet: A time-sensitive networking approach. *Acta IMEKO*, 14(3):1–10, 2025.

[29] Andreas C. Müller and Sarah Guido. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, Sebastopol, CA, USA, 2016.

[30] Alfredo Nascita, Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, Valerio Persico, and Antonio Pescapé. A survey on explainable artificial intelligence for internet traffic classification and prediction, and intrusion detection. *IEEE Communications Surveys & Tutorials*, 2024. early access.

[31] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert. Trends in automotive communication systems. *Proceedings of the IEEE*, 93(6):1204–1223, 2005.

[32] Damilola Oladimeji et al. Canattack: Assessing vulnerabilities within controller area network. *PMC / Scientific Reports(?)*, 2023. Accessed: 2025-11-04.

[33] Anonymous or CSE570-19 Course Material. A survey of automotive ethernet technologies and protocols. 2019. Available as open PDF overview: https://www.cse.wustl.edu/~jain/cse570-19/ftp/auto_eth.pdf.

[34] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986.

[35] Sampath Rajapaksha, Harsha Kalutarage, M. Omar Al-Kadri, Andrei Petrovski, Garikayi Madzudzo, and Madeline Cheah. Ai-based intrusion detection systems for in-vehicle networks: A survey. *ACM Comput. Surv.*, 55(11), February 2023.

[36] Muhammad Sami, Matthew Ibarra, Anamaria C Esparza, Saleh Al-Jufout, Mehrdad Aliasgari, and Mohammad Mozumdar. Rapid, multi-vehicle and feed-forward neural network based intrusion detection system for controller area network bus. In *2020 IEEE Green Energy and Smart Systems Conference (IGESSC)*, pages 1–6. IEEE, 2020.

[37] Eunbi Seo, Hyun Min Song, and Huy Kang Kim. GIDS: GAN based Intrusion Detection System for In-Vehicle Network. In *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, pages 1–6, August 2018. arXiv:1907.07377 [cs].

[38] Shaila Sharmin, Hafizah Mansor, Andi Fitriah Abdul Kadir, and Normaziah A. Aziz. Benchmarking frameworks and comparative studies of controller area network (can) intrusion detection systems: A review, 2024.

[39] Craig Smith. *The Car Hacker's Handbook: A Guide for the Penetration Tester*. William Pollock, 2016. Last accessed on 2022.05.01.

[40] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, pages 305–316. IEEE, 2010.

[41] Hyun Min Song, Jiyoung Woo, and Huy Kang Kim. In-vehicle network intrusion detection using deep convolutional neural network. *Vehicular Communications*, 21:100198, 2020.

[42] Dario Stabili, Francesco Pollicino, and Alessio Rota. A benchmark framework for CAN IDS. *ITASEC21: Italian Conference on Cyber Security*.

[43] Adrian Taylor, Nathalie Japkowicz, and Sylvain Leblanc. Frequency-based anomaly detection for the automotive can bus. In *2015 World Congress on Industrial Control Systems Security (WCICSS)*, pages 45–49. IEEE, 2015.

[44] University of Ljubljana – Bioinformatics Lab. Orange data mining (version 3.x) — a visual programming environment for data analysis, 2024.

[45] Abhishek Verma and Virender Ranga. Machine learning based intrusion detection systems for iot applications. *Wireless Personal Communications*, 111(4):2287–2310, 2020.

[46] Miki E Verma, Michael D Iannacone, Robert A Bridges, Samuel C Hollifield, Pablo Moriano, Bill Kay, and Frank L Combs. Addressing the lack of comparability & testing in can intrusion detection research: A comprehensive guide to can ids data & introduction of the road dataset. *arXiv preprint arXiv:2012.14600*, 2020.

[47] Peter Waszecki, Philipp Mundhenk, Sebastian Steinhorst, Martin Lukasiewycz, Ramesh Karri, and Samarjit Chakraborty. Automotive electrical and electronic architecture security via distributed in-vehicle traffic monitoring. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(11):1790–1803, 2017.

Table 1: Model Performance Metrics

Dataset	Attack	FPR	FNR	Precision	Recall	F1-score	Comments
HCRL Car Hacking	DoS	0.000	0.000	1.000	1.000	1.000	1 Unique Attack
	Fuzzing (99%-Tree)	0.059	0.001	0.994	0.941	0.967	51 nodes
	Fuzzing (100%-Tree)	>0.001	>0.001	0.996	1.000	0.998	389 nodes
	Spoofing (Gear & RPM)	0.000	0.000	1.000	1.000	1.000	1 Unique Attack
Survival	DoS	0.000	0.000	1.000	1.000	1.000	1 Unique Attack
	Fuzzing	0.024	>0.001	0.997	0.976	0.987	25 nodes
	Fuzzing (100%)	0.002	0.001	0.997	0.998	0.998	165 nodes
	Malfunction	0.003	0.000	0.998	0.995	0.997	5 nodes
Can-Train-and-Test	DoS / Standstill (100%)	0.000	0.000	1.000	1.000	1.000	1 Unique Attack
	Fuzzing (100%)	0.014	0.002	0.893	0.986	0.937	235 nodes
	Systematic (100%)	0.010	>0.001	0.999	0.990	0.995	127 nodes
	Speed-Spoofing (100%)	0.000	0.000	1.000	1.000	1.000	3 Unique Attacks
	Triple (100%)	0.002	0.000	1.000	0.998	0.999	115 nodes
Car-Hacking Attack & Defense	DoS Train / Prelim	0.000	0.000	1.000	1.000	1.000	1 Unique Attack Trace
	Fuzzing Train	0.101	>0.001	0.988	0.899	0.942	101 nodes
	Fuzzing Train (100%)	0.003	>0.001	0.994	0.996	0.995	387 nodes
	Fuzzing Prelim	0.098	>0.001	0.999	0.902	0.948	101 nodes
	Fuzzing Prelim (100%)	0.004	0.001	0.980	0.996	0.988	387 nodes
	Fuzzing Final	0.068	0.001	0.967	0.932	0.949	101 nodes
	Fuzzing Final (100%)	0.006	0.004	0.871	0.994	0.928	387 nodes
	Spoofing Train (100%)	0.000	>0.001	0.999	1.000	1.000	25 nodes
	Spoofing Prelim (100%)	1.000	>0.001	0.000	0.000	0.000	25 nodes
	Spoofing Final (100%)	1.000	>0.001	0.000	0.000	0.000	25 nodes
	Replay Train	0.454	0.002	0.837	0.546	0.661	3 809 nodes
	Replay Train (100%)	0.255	0.003	0.784	0.745	0.764	10 707 nodes
	Replay Prelim	0.555	0.003	0.715	0.445	0.548	3 809 nodes
	Replay Prelim (100%)	0.555	0.003	0.709	0.445	0.547	10 707 nodes

Table 2: Abbreviations used in dataset overview tables.

Category	Abbreviations and Meanings
Attack Types	D – DoS, F – Fuzzing, S – Spoofing, M – Masquerade, R – Replay, Su – Suppression, Di – Diagnostic, RPMt – RPM Tampering.
Data Source	R – Real, S – Synthetic, M – Mixed (Real/Synthetic).
Message Structure	T – Timestamp, ID – CAN ID, DLC – Data Length Code, Data – Payload Data, Sig – Signals.

Table 3: Overview of available CAN intrusion datasets with criteria for selection (real source, labels exist, access is provided)

Dataset	Environment	Attack Types	Source	Attributes	Labels	Access	Analyzed
OTIDS [22]	1 vehicle	D, F	R	T, ID, DLC, Data	✗	✓	✓
Car-Hack. [37]	1 vehicle	D, F, S	R	T, ID, DLC, Data	✓	✗	✗
Survival [10]	3 vehicles	D, F	R	T, ID, DLC, Data	✓	✓	✓
Aut. CAN [6]	2 vehicles; 1 testbed	D, F, R, Di, Su	S	T, ID, Data	-	✓	✗
SynCAN [11]	1 simulation	S, Su, M	S	T, ID, Sig	✓	✓	✗
CarChall. [16]	1 vehicle	D, F, R, S	R	T, ID, DLC, Data	✓	✓	✓
ROAD [46]	1 vehicle	S, F, M	M	T, ID, Data	✗	✓	✗
TTIDS [24]	1 vehicle	M	R	T, ID, DLC, Data	✗	✓	✗
ct&t [21]	4 vehicles; 1 testbed	D, S, F, M, R, Su	R	T, ID, DLC, Data	✓	✓	✓
Sim CAN [17]	1 testbed	D, F	S	T, ID, DLC, Data	✗	✗	✗
CrySyS [20]	1 vehicle; 26 recordings	S, M	R	T, ID, DLC, Data	-	✓	✗
HCRL [41]	2 vehicles	D, S, F, RPMt	R	T, ID, Data	✓	✓	✓
AEGIS [15]	1 vehicle; 3 drivers	None	R	T, ID, DLC, Data	✗	✓	✗
ML350 [36]	1 vehicle	Unknown	R	T, ID, DLC, Data	✓	✗	✗

Short: CARdea: Two-Phase Plausibility-Based Vehicle-to-Vehicle Anomaly Detection System

Mert D. Pesé[1], Arman Tabaddor[2], Teja Guruvelli[1], Roland Varriale[3], Marco De Vincenzi[4], Kang G. Shin[2]

[1]*Clemson University* [2]*The University of Michigan* [3]*Argonne National Laboratory* [4]*IIT-CNR*

[1]*{mpese, gguruve}@clemson.edu* [2]*{armantab, kgshin}@umich.edu*

[3]*rvarriale@anl.gov* [4]*marco.devincenzi@iit.cnr.it*

Abstract

Secure Vehicle-to-Vehicle (V2V) communication is threatened by internal attackers broadcasting anomalous Basic Safety Messages (BSMs). Existing anomaly detection methods often neglect deployment feasibility in resource-constrained vehicles, relying heavily on computationally expensive machine learning (ML) or multi-modal sensor fusion. To address this, we propose `CARdea`, a practical, real-time two-phase anomaly detection framework relying solely on standard BSM kinematics. Phase 1 performs a lightweight, in-vehicle statistical plausibility check, while Phase 2 offloads complex edge cases to a robust, opportunistically deployed ML classifier. Evaluated on 108 hours of simulated traffic, `CARdea` achieves an ultra-low 0.10 ms detection latency on edge hardware alongside an average 95.35% TPR and 0.55% FPR across seven attack types.

1 Introduction

Connected and automated vehicles (CAVs) rely on Vehicle-to-Vehicle (V2V) communication, exchanging Basic Safety Messages (BSMs) [18] to share kinematic state data such as position, speed, and acceleration. While authentication protocols effectively mitigate threats from external actors lacking valid credentials [14], a critical vulnerability remains: internal attackers. Compromised vehicles can broadcast legitimate but anomalous BSMs to launch false data injection, Sybil, or replay attacks [19], posing severe safety risks to the control systems of surrounding vehicles.

Despite recent advances in BSM anomaly detection, existing solutions largely neglect the deployment feasibility within resource-constrained Electronic Control Units (ECUs) [17]. Current methods propose either computationally heavy Machine Learning (ML) approaches or rely on expensive multi-modal sensor fusion (e.g., radar, Angle-of-Arrival) that is not yet economically viable for standard vehicles [10]. Crucially, prior work frequently ignores the ultra-low detection latency required to mitigate safety risks in real time.

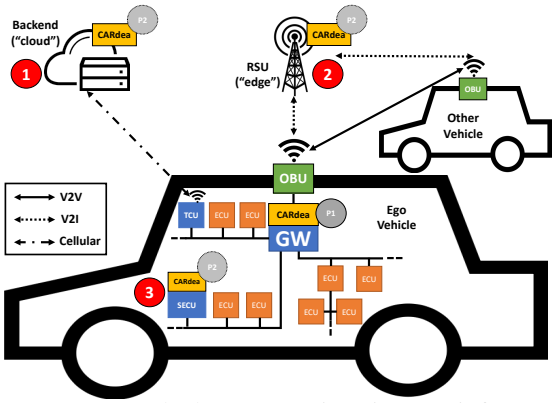


Figure 1: `CARdea` deployment options in V2X infrastructure

To address this disconnect, we propose `CARdea`[1], a practical, real-time anomaly detection scheme for V2V communication. `CARdea` bridges the gap between performance and practicality through a novel two-phase architecture. Phase 1 performs a lightweight, data-centric statistical plausibility check locally inside the vehicle, evaluating inherent correlations among BSM kinematic variables. Phase 2 acts as a robust ML classifier deployed opportunistically (e.g., edge or cloud) to resolve complex edge cases that Phase 1 cannot definitively classify. This paper makes the following main contributions:

- **Novel Two-Phase Architecture**: A hybrid system combining in-vehicle statistical checks with a remote ML classifier to optimize V2V safety and computational usability.
- **Ultra-Low Latency**: Achieves 0.10 ms detection latency on resource-constrained hardware (Raspberry Pi 3), meeting real-time in-vehicle requirements.
- **High Detection Efficacy**: Validated via 108 hours of simulation [25], achieving 95.35% TPR and 0.55% FPR across seven attack types using only standard BSM data. Code and datasets are publicly available [2].

[1]Cardea was the ancient Roman goddess of the hinge, Roman doors being hung on pivot hinges.

[2]https://github.com/tigerseclab/VehicleSec26_CARdea

Table 1: Comparison with related work. Notation: Position p, speed v, acceleration a, heading h, yaw rate ω.

Work	So19 [23]	Sha24 [20]	Yao17 [32]	Yav17 [33]	Bis12 [3]	Sun17 [28]	Van19 [29]	**CARdea**
BSM Data	p	p, v, a, h, ω	N/A	p, v, a	p, h, v	p, a	p, v, a	p, v, a
Other Data	RSSI	None	RSSI	Map	Radar	AoA, DS	None	None
Anomalies	Constant (+Offset) Random (+Offset) Eventual Stop	Constant (+Offset) Random (+Offset) High/Low Value Opposite Rotating Perpendicular	Sybil	Random Error	Sybil	Constant Offset	Instant Constant Gradual Drift Bias	Constant (+Offset) Random (+Offset) Sybil Replay
Approach	Physical Layer Consistency Check	Wasserstein GAN (WGAN)	Dynamic Time Warping (DTW)	Vehicle Physics (VP)	Particle Filter (PF)	Extended Kalman Filter (EKF)	CNN+KF	VP + (RF,SVM,DNN)
Performance (%)	TPR: 83.7	AUROC: 89 FPR: < 5	TPR: > 90 FPR: < 10	TPR: 90 FPR: 2.7	N/A	TPR: 100 FPR: 5	TPR: 51.5-99.2	Phase 1 TPR: 98 FPR: 0.02 Phase 2 TPR: 99 FPR: 0.16
Detection Time (ms)	N/A	40	630	N/A	5-25	3.2	KF: 5.1	Phase 1: 0.09-0.11 Phase 2: 5.1-15.5

2 Threat Model

Based on adversary incentives [26], V2X attacks fall into two taxonomic dimensions [10]: (i) *Active* vs. *Passive*: active attackers interact with the system; passive attackers eavesdrop. (ii) *Internal* vs. *External*: internal attackers hold valid credentials; external ones do not. Most V2X attacks assume an active, internal adversary, including DoS, Sybil, replay, and false data injection [19]. DoS exhausts resources, Sybil creates ghost vehicles via fake identities, replay resends stale BSMs, and false data injection broadcasts spoofed sensor data. We focus on falsification of kinematic state data (Sybil, replay, false data injection), as these are validatable against physical laws. High-level semantic attacks (e.g., bogus traffic warnings) are excluded, as they require context-aware verification beyond CARdea's kinematic plausibility checks.

False data may stem from faulty sensors or deliberate spoofing via a (i) compromised OBU or (ii) compromised IVN. Miller *et al.* [13] demonstrated that the CAN bus is vulnerable to both physical and remote tampering. Anomalies should be caught before BSM broadcast or before data reaches actuator ECUs. While CAN-based IDS detects IVN compromise [31], it misses OBU-level tampering; thus, false data is best detected upon BSM receipt before IVN forwarding. We assume a compromised vehicle broadcasting contiguous spoofed BSMs to our ego vehicle, with the attacker controlling all BSM field values. In contrast, faulty sensors typically cause issues in only a few frames rather than sustained segments. Note that CARdea is agnostic to the European equivalent of BSM, the *Cooperative Awareness Message* (CAM) [9].

We propose CARdea as a data-centric, plausibility-based V2V anomaly detection module running *inside* the central gateway. Detected BSM anomalies trigger data discarding; anomalies persisting beyond a defined threshold result in full suspension of communication with the offending vehicle.

Generalization to Multi-Sensor Spoofing. One drawback of CARdea's current system design is that it only comprises *single*-sensor spoofing in AT1–4, where one consistent sensor is anomalous per BSM. Another case is *multi*-sensor spoofing, where more than one sensor may be anomalous. If two sensors are compromised, Phase 1 detects inconsistency via inter-sensor correlation. If all three are spoofed consistently, Phase 1 may fail, an inherent limitation of physics-based detection also noted by So *et al.* [23]. Adding RSSI as a parallel plausibility check mitigates this: even when a smart attacker maintains kinematic consistency across all three sensors, RSSI can still flag position spoofing. A BSM is marked anomalous if *either* module triggers. Multi-sensor spoofing experiments are left for future work.

3 Related Work

Existing Approaches. As summarized in Table 1, previous V2X anomaly detection largely relies on either statistical plausibility checks or Machine Learning (ML). Statistical methods leverage vehicle dynamics [33], particle or Kalman filters [3, 28], and RSSI [23, 32]. While computationally lighter, pure data-centric statistical checks frequently fail against sophisticated GPS spoofing unless augmented by physical-layer signals [23] or expensive multi-modal sensor fusion, such as radar and Angle-of-Arrival [28]. Conversely, ML-based solutions employing Convolutional Neural Networks [29] or Generative Adversarial Networks (VEHIGAN) [20] achieve highly robust detection against complex attacks. However, these models introduce computational and memory overhead in the Electronic Control Units (ECUs).

Comparison to Previous Work. While recent advancements in BSM anomaly detection have demonstrated promising detection rates, they frequently overlook the practical and economic constraints of actual in-vehicle deployment. CARdea distinguishes itself from existing literature through three novel contributions tailored for real-world automotive feasibility:

Ultra-Low Latency on Resource-Constrained Hardware. Existing ML-based approaches incur high latency. CARdea

achieves 0.10 ms detection latency on ECU-like hardware, at least 46× faster than Kalman Filter (KF)-based detectors. Moreover, most prior work does not report latency [7, 30, 33].

Cost-Effective Data Requirements. Prior work relies on costly multi-modal sensing (e.g., AoA, Doppler Shift) [1, 22]. CARdea instead uses only existing kinematic data (position, speed, acceleration) from standard BSMs, eliminating the need for additional sensors and reducing deployment cost.

Strategic Two-Phase Hybrid Architecture. Prior work trades off latency (statistical models) and accuracy (ML). CARdea bridges this via a two-phase design: a lightweight in-vehicle statistical check (Phase 1) and an opportunistic remote ML classifier (Phase 2), preserving real-time safety while maintaining high detection performance.

4 System Design

4.1 Overview

Considering the limitations of related work, we propose `CARdea` to provide real-time, accurate, and light-weight detection of BSM anomalies. It uses a hybrid of statistical (Phase 1, Sec. 4.3) and ML-based (Phase 2, Sec. 4.4) anomaly detection. Phase 1 runs locally *inside* the vehicle to provide very low detection latency that only statistical approaches can provide. Another benefit of `CARdea` is the low memory consumption which is important due to automotive resource constraints [17]. In contrast, Phase 2 can run *outside* the vehicle, e.g., on the car-maker's backend or the roadside edge or even cloud, and can rely on more computing resources.

Phase 1 performs real-time detection of both *highly-likely anomalous frames* and *highly-likely non-anomalous frames*. We define a *frame* f as a sequence of contiguous sensor readings/samples from a transmitter vehicle. A frame is classified as *highly-likely anomalous* if the mean of the predicted conditions for each (anomalous or non-anomalous) sample in the frame is above a certain per-frame threshold $\theta_{f,a}$. Similarly, a frame is said to be *highly-likely non-anomalous* if the mean of the predicted conditions for each sample in the frame is smaller than $\theta_{f,na}$. Frame predictions between these two thresholds will be sent to Phase 2 for fine-grained classification. Phase 1 will temporarily consider such frames *potentially anomalous* if the frame prediction is greater than 0.5, and *potentially non-anomalous* otherwise, until a classification result from Phase 2 becomes available. Thus, the goal in Phase 1 is to maximize the safety-critical True Positive Rate (TPR) as well as to minimize the False Positive Rate (FPR). However, Phase 1 is optimized for frames that consist of mainly anomalous samples (i.e., from an attacker vehicle) or mainly non-anomalous samples (i.e., from a benign vehicle). Frames consisting of a mixture of anomalous and non-anomalous samples (i.e., a 50/50 split) are sent to Phase 2 for *per-sample detection*. Our ego vehicle can temporarily suspend its communication with vehicles marked as anomalous as a result of anomalies in consecutive BSMs.

Phase 2 provides computationally powerful per-sample detection strictly for frames where Phase 1 is uncertain, preserving V2V usability without sacrificing safety. Both phases exploit inter-sensor correlations: Phase 1 via vehicle dynamics, Phase 2 via ML, relying exclusively on standard BSM data and RSSI to eliminate expensive sensor fusion. This hybrid architecture of `CARdea` satisfies OEM constraints by allowing flexible Phase 2 deployment (cloud, edge, or super-ECU) to balance cost, bandwidth, and performance.

4.2 Anomalies under Consideration

It is challenging to evaluate anomaly detection schemes for CAVs due to the lack of anomaly datasets for BSMs. Thus, we evaluate `CARdea` using the Vehicular in Network Simulation (`Veins`) framework [25]. By using a simulation instead of generating anomalies on top of a static dataset, we ensure that false data may propagate to the control system over time and affect the behavior of the ego vehicle and neighboring vehicles. A simulation can thus account for vehicle dynamics.

We evaluate `CARdea` on 7 attack types. AT1–AT4 falsify position (lat/lon), speed, and acceleration, extended from [11,12] and consistent with literature [2, 21]. AT5 (Sybil), AT6 (Data Replay), and AT7 (Stealthy) target real-world scenarios; AT5–AT6 are extended from [11], while AT7 is motivated by stealthy attacks on robotic vehicles [5, 6, 16]. AT7 specifically maximizes spoofing damage while evading detection, stress-testing `CARdea` against a sophisticated adversary consistent with the threat model in Sec. 2. The superscript i denotes the sensor; x_t^i is the true measurement at time t: **(AT1) Constant:** broadcasts a fixed $x_t^i = x_c$; **(AT2) Constant Offset:** adds a fixed offset, $x_t^i = x_t^i + \Delta x_c$; **(AT3) Random:** broadcasts $x_t^i = \mathcal{U}([x_{min}, x_{max}])$ bounded by the simulation space; **(AT4) Random Offset:** adds random noise, $x_t^i = x_t^i + \mathcal{U}([-x_c, x_c])$; **(AT5) Sybil:** generates ghost vehicles via $x_t^i = S_x \cdot (x_l^i + x_r^i)$ for map grid length S_x, local measurement x_l^i, and random measurement x_r^i; **(AT6) Data Replay:** resends delayed target BSM data $x_t^i \in_R [x_0^i, x_{t-1}^i]$; and **(AT7) Stealthy:** injects the maximum relative false data without crossing the optimal per-sample threshold θ_s^i via $x_t^i = x_t^i + (1 - \theta_s^i \cdot b_i)$ for $b_i > 0$.

4.3 Phase 1: Local Anomaly Detection

Overview. Phase 1 performs real-time detection of anomalies inside the vehicle by leveraging statistical models of inherent sensor correlations. Among the sensor data included in the BSM [18], we select three fundamental sensors to express vehicle dynamics: (GPS) position $\vec{p}$, velocity $\vec{v}$, and longitudinal acceleration a. BSMs are transmitted by vehicles every 100 ms, indicated by dt. Our ego vehicle will receive BSMs from M surrounding vehicles which vary with traffic and driving behavior of the other cars. The objective of Phase 1 is to perform the plausibility verification

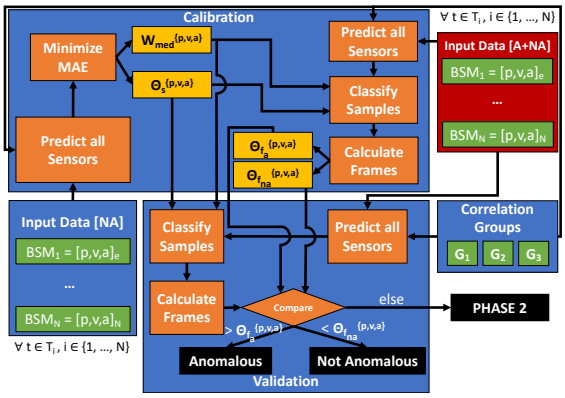


Figure 2: Phase 1 (A: Anomalous, NA: Non-Anomalous)

of each BSM upon its reception by the OBU and to determine if the analyzed BSM is anomalous. Let t be the absolute timestamp; the correlated sensor groups follow basic physics equations: $\vec{p}_{t+dt} = f(p_t, \vec{v}_t) = \vec{p}_t + \int_t^{t+dt} \vec{v}_t dt$, $v_t = f(\vec{p}_t, \vec{p}_{t+dt}) = \frac{|\vec{p}_{t+dt} - \vec{p}_t|}{dt}$, and $a_t = f(v_t, v_{t+dt}) = \frac{v_{t+dt} - v_t}{dt}$.

As shown in Fig. 2, Phase 1 is decomposed into threshold calibration and validation modules. The *correlation groups* G1–G3 characterize the correlation between the three sensors as expressed above, respectively. Namely, G1 describes the relationship between position $\vec{p}$ and velocity vector $\vec{v}$; G2 the relationship between speed v and position $\vec{p}$; and G3 the relationship between longitudinal acceleration a and speed v. Speed v is defined as the magnitude of two-dimensional velocity $\vec{v} = (v_x, v_y)^T$. Below we elaborate on the threshold calibration and validation of Phase 1.

Calibration. The calibration module is configured to operate offline, presumably at the time of vehicle manufacturing. The objective of calibration is to compute four thresholds for each sensor $i \in \{p, v, a\}$: the median window size W^i_{med}, per-sample threshold θ^i_s, per-frame anomaly threshold $\theta^i_{f,a}$, and per-frame non-anomaly threshold $\theta^i_{f,na}$. A single anomalous sensor reading in a BSM can be detected by using the aforementioned correlation groups. For example, if only the speed v is anomalous, G2 can be used to calculate the correct speed from the GPS readings. If v is not anomalous, G2 should yield a value similar to the speed in the received BSM. Since GPS readings can be noisy/erroneous (like any other sensor reading), the difference between *calculated* and *received* values will have a small error. The mean of all errors across all non-anomalous datasets is characterized by the per-sample threshold θ^i_s for each correlation group. This threshold is determined by computing the *Mean Absolute Error* (MAE) for each sample t between the received signal x_t and estimated signal $\hat{x}_t$. Ideally, estimated and received signals should be near-identical for non-anomalous data. However, estimation is prone to noise, which a rolling mean filter of length W mitigates by smoothing the raw signal. The optimal window W^i_{med} for each sensor i minimizes MAE across all non-anomalous trips; the averaged minimized MAEs yield θ^i_s. Using only past samples up to the current frame, the filter preserves real-time detection.

These two calibrated parameters W^i_{med} and θ^i_s may be sufficient to detect the anomaly in a BSM. However, we introduce further statistical techniques and design choices to reduce the false alarm rate in Phase 1. Similar to a cumulative sum, we compute an average $\theta^i_{f,a}$ for all anomalous frames, for each sensor group. A frame prediction is the mean of predicted conditions (1 for anomalous or 0 for non-anomalous) based on θ^i_s for samples in a frame f. This same calculation is computed for non-anomalous frames, $\theta^i_{f,na}$. The per-frame threshold computation can be generalized in Eq. (1), where N is the number of samples and $|f|$ the frame size:

$$\theta^i_{f,*} = \frac{1}{N} \sum_{j=1}^{N} \frac{1}{|f|} \sum_{k=j\cdot|f|}^{j\cdot|f|+|f|} x^i_k. \quad (1)$$

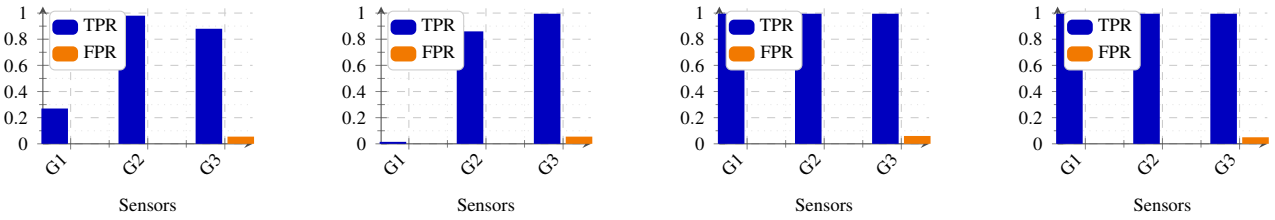
The introduction of the above thresholds helps ensure truly anomalous or truly non-anomalous frames are detected immediately by Phase 1. Therefore, Phase 1 considers frames, and the computed frame value is compared to the respective $\theta^i_{f,*}$ for anomalous and non-anomalous vehicles to make a decision. Trustworthiness of calibration is ensured through continuous backend validation. OEMs can aggregate non-anomalous data from the fleet to periodically refine W_{med} and θ values, pushing optimized thresholds to vehicles via secure over-the-air (OTA) updates to account for aging sensors or changing road conditions.

Validation. The anomaly detection of Phase 1 operates online and extracts the relevant signal data from each BSM. Using G1-G3 and the four calibration parameters W^i_{med}, θ^i_s, $\theta^i_{f,a}$, and $\theta^i_{f,na}$, it first calculates the MAE for the three correlation groups and compares it with the threshold θ^i_s computed during training. If the computed MAE is larger than the threshold, the predicted condition for the respective sample is marked as *anomalous*. Likewise, if the computed MAE is smaller than the threshold, the predicted condition for the sample is marked as *non-anomalous*. After collecting W^i_{med} samples, the samples are smoothed by a rolling mean filter of window size W^i_{med}, and the MAE is re-calculated and compared again with the threshold to update the appropriate markings.

We then compute the mean of all predicted conditions for samples in a frame, such that if this computed frame prediction is greater than, or equal to $\theta^i_{f,a}$, we mark all samples within this frame as *highly-likely anomalous*, and if this computed frame prediction is less than, or equal to $\theta^i_{f,na}$, this frame is marked as *highly-likely non-anomalous*. If the frame prediction lies between these two thresholds, the affected frames are sent to Phase 2 for further investigation.

4.4 Phase 2: Remote Anomaly Detection

Overview. Phase 2 employs ML-based anomaly detection designed for opportunistic, remote deployment (e.g., edge



(a) AT1 (b) AT2 (c) AT3 (d) AT4

Figure 3: Phase 1 per-frame performance with frame size $f = 10$

or cloud, see Fig. 1), operating without strict real-time constraints. Its primary goal is to resolve uncertain frames forwarded from Phase 1, specifically those falling within the interval $[\theta^i_{f,na}, \theta^i_{f,a}]$. For each sensor group, an associated ML model executes in parallel. Similar to Phase 1, a final logical-OR is applied, meaning an anomaly is flagged if any single model detects it.

Feature Extraction. Motivated by [24], the features extracted are dependent on a sensor of interest. More precisely, for each correlation group, we extract the rolling mean $\overline{X^i_t}$, rolling mean displacement, estimated displacement and plausibility check for each vehicle communicating with the ego vehicle.

Training and Validation. We evaluate three models for Phase 2: Random Forest (RF) uses 5-fold cross-validation with 100 estimators; Support Vector Machine (SVM) standardizes features to zero mean and unit variance, utilizing an RBF kernel ($C = 10$) to balance performance with training time; and a Deep Neural Network (DNN) relies on a 60/20/20 train/val/test split, trained for 25–50 epochs (batch size 32) with early stopping. The DNN architecture comprises 6 hidden layers (32–128 nodes, ReLU activations), a 0.2 dropout rate, and a sigmoid output, optimized via Adam using binary cross-entropy loss.

5 Evaluation

5.1 Experimental Setup

We evaluate CARdea using the Veins simulation framework [26] with the Luxembourg SUMO Traffic (LuST) scenario [4] over a 6.63 km^2 topology. Our dataset comprises 108 hours of simulations encompassing over 49,000 vehicles. Attacker vehicles were spawned with a probability $\alpha = 0.05$ and broadcast contiguous sequences of anomalous messages (30–70 messages) with probability $\beta = 0.01$ [27]. These adversaries executed the seven defined attack types (AT1–AT7) manipulating position, speed, and longitudinal acceleration. To demonstrate deployment feasibility, Phase 1 was evaluated in Python 3 on a resource-constrained Raspberry Pi 3 Model B (1.2 GHz, 1 GB RAM) representative of an in-vehicle ECU. Phase 2 was evaluated on a desktop server (64-core Intel Xeon, 128 GB RAM) representing an edge or cloud backend.

5.2 Phase 1

Performance. We first calculated the ROC (see Fig. 4a in Appendix) using our training data for each correlation group to determine a satisfactory θ^i_s. To select an optimal θ^i_s, we selected the first threshold with at least 90% TPR, in accordance with the ROC curve. Then we used θ^i_s to determine $\theta^i_{f,*}$ values for each correlation group. We calculated the means over all different configurations of the anomalous datasets. The calibrated thresholds for each correlation group are as follows: for G_1, $\theta^1_{f,a} = 0.95$ and $\theta^1_{f,na} = 0.015$; for G_2, $\theta^2_{f,a} = 0.83$ and $\theta^2_{f,na} = 0.021$; for G_3, $\theta^3_{f,a} = 0.90$ and $\theta^3_{f,na} = 0.04$.

Furthermore, when computing the $\theta^i_{f,a}$ and $\theta^i_{f,na}$ values, we evaluated them on frames of size $3, 5, 7$, and 10. As frame size increases, performance generally improves, as shown in Figs. 4b and 4c (in Appendix). G3 FPR, for example, decreases from $\approx 9\%$ for a frame size of 3 to $\approx 5.5\%$ for a frame size of 7, and then plateaus. We selected a frame size of 10 for performance, latency, and resource metrics.

Phase 1 targets high TPR for anomalous frames and low FPR for non-anomalous ones. As Fig. 3 shows, confident Phase 1 classifications can trigger mitigation without engaging Phase 2. However, G1 AT1 and AT2 are a known limitation: because G1 uses its previous (potentially spoofed) value to compute estimates, MAEs spike at anomalous/non-anomalous transitions and fail to stay consistently below the determined threshold θ^i_s. These position spoofing attacks are better addressed via RSSI-based plausibility checks [23], which can boost TPR by up to 40% when combined with Phase 1 via logical OR. For AT5 and AT6, Phase 1 achieves 99% TPR and 0.005% FPR. Stealthy attacks, however, are explicitly designed to evade Phase 1, making `CARdea` vulnerable to them, if undetected in Phase 1, all frames are forwarded to Phase 2, negating the hybrid system's efficiency benefits.

Detection Latency. Phase 1 mean latency was evaluated on a Raspberry Pi 3 Model B (comparable to in-vehicle ECUs). As Table 6 (in Appendix) shows, processing one frame takes 0.10 ms across all attack types and correlation groups, negligible given BSMs arrive every 100 ms. In an AV platooning scenario, Phase 1 latency is $\approx$2000x smaller than the delay required to restore full autonomy after compromise, which ranges from 247 ms (2 vehicles) to 1457 ms (8 vehicles) [8].

Memory Consumption. Table 6 (in Appendix) shows an

Table 2: Phase 2 per-sample performance for frame size $f = 10$. AT5 and AT6 are independent of correlation group G* and are displayed for G1 merely for visualization purposes. Best values per attack type are bolded.

	G1			G2			G3		
Attack Type	TPR	FPR	F-1	TPR	FPR	F-1	TPR	FPR	F-1
AT1	93.46%	3.24%	90.30%	**96.90%**	0.57%	**94.77%**	91.36%	**0.02%**	91.44%
AT2	**76.06%**	**0.53%**	**77.81%**	68.62%	1.30%	69.35%	68.30%	0.83%	69.60%
AT3	94.53%	9.06%	87.45%	97.14%	**1.00%**	**94.03%**	75.04%	13.02%	75.06%
AT4	**91.16%**	1.21%	**89.50%**	88.53%	1.26%	87.04%	52.16%	0.89%	52.02%
AT5	**92.75%**	**0.23%**	**93.80%**	-	-	-	-	-	-
AT6	**87.43%**	2.89%	**86.16%**	-	-	-	-	-	-
AT7, $b = 0.1$	62.93%	1.08%	64.57%	**86.37%**	1.71%	**84.29%**	70.29%	**0.50%**	71.15%
AT7, $b = 0.01$	60.01%	**0.32%**	61.35%	**78.99%**	1.87%	**78.20%**	64.68%	0.46%	65.49%
AT7, $b = 0.001$	61.12%	**0.25%**	62.02%	**83.56%**	0.83%	**83.74%**	67.70%	0.40%	68.41%

Table 3: Phase 2 results on frames from Phase 1

	G1				G2				G3				**Average**
	AT1	**AT2**	**AT3**	**AT4**	**AT1**	**AT2**	**AT3**	**AT4**	**AT1**	**AT2**	**AT3**	**AT4**	
TPR	100%	99.75%	99.67%	99.51%	100%	85.8%	99.06%	76.5%	98.64%	98.37%	95.9%	91.0%	95.35%
FPR	0.0%	0.015%	0.0%	0.0%	0.0%	1.3%	0.16%	1.9%	0.09%	0.3%	1.06%	1.8%	0.55%

average RAM usage of 39 MB across the four attack types. While this may seem high for in-vehicle ECUs with limited RAM [17], the current Python implementation inflates usage. A C-based version would require far less memory, given the minimal parameters and equations involved.

5.3 Phase 2

We evaluate TPR, FPR, and F-1 score of Phase 2, both independently and jointly with Phase 1. We first evaluate Phase 2 in a scenario where it receives *all* frames from Phase 1.

Performance. Compared to Phase 1's *per-frame* performance evaluation, Phase 2 focuses on *per-sample* classification, which explains certain lower TPR values, despite its generally favorable performance. For data spoofing attacks (AT1—AT4), RFs almost exclusively display the best performance, particularly for G1 and G2, frequently achieving high TPR and F-1 scores (e.g., $>$ 90% F-1 for AT1, AT3, AT4). Compared to Van *et al.* [29] who inject anomalies on a static dataset, instead of conducting a simulation that makes their evaluation less realistic, they report performance comparable to `CARdea`. Additionally, Sun *et al.* [28] implemented the constant offset attacker, yet relied on AoA and Doppler shift to achieve comparable results. One limitation of Phase 1 was the need to incorporate RSSI to detect such attacks, which might also be required for Phase 2, as AT2 performance suggests. However, AT1 performance is significantly better than in Phase 1, even without considering RSSI. For Sybil (AT5) and Data Replay (AT6) attacks, RFs outperform SVMs and DNNs with $\approx$ 90%+ average F-1 score. For stealthy attacks (AT7), there was no observed correlation in detection performance and the attacker's b_i selection, with RFs being the superior ML model again. The performance numbers for AT7 were only slightly lower than the other attack types, demonstrating that `CARdea` succeeded in thwarting this advanced attack type. Table 2 summarizes the performance of the best performing model (Random Forest) while Table 4 depicts the full performance of all three models for all attack types.

Computation Time. Table 6 (in Appendix) reports Phase 2 inference times for RF, SVM, and DNN across frame sizes 3–10 on both the RPi and desktop. RPi latency substantially exceeds desktop: e.g., RF at f=10 takes 101.9 ms on the RPi vs. 16.0 ms on the desktop, exceeding the BSM period. Compared to Phase 1's 0.10 ms, Phase 2's ML overhead is not ideal for direct ECU deployment.

Memory Consumption. Table 6 (in Appendix) shows that ML models consume significantly more RAM than statistical methods, about 5× more in Phase 1, making them less suitable for ECUs. RFs and SVMs exceed 100 MB, and deep learning models require up to 3× more.

Interactions between Phase 1 and 2. We now perform a combined evaluation of Phases 1 and 2 jointly to evaluate Phase 2 on the frames sent from Phase 1. We can express this real-time interaction between the two phases where the input X are Phase 1's estimated sensor readings of frame size N. Phase 2 implements an RF model, and we select a frame size of 10, consistent with our preceding evaluation. Phase 2 reports an average 95.35% TPR and 0.55% FPR (see Table 3).

6 Conclusion

`CARdea` demonstrates that combining a lightweight, in-vehicle statistical check with a powerful ML-based approach offers a highly performant, practical solution for real-time V2V anomaly detection. By bridging the gap between high detection accuracy and deployment feasibility on resource-constrained ECUs, our two-phase architecture proves that security and latency need not be a compromise. Future work will explore adaptive thresholding, dynamically shrinking the uncertain interval $[\theta_{f,na}, \theta_{f,a}]$ as Phase 1 learns from Phase 2. This will progressively reduce offloading rates and further minimize cellular bandwidth over the vehicle's lifecycle.

Acknowledgments

This work was supported in part by NSA H98230-21-1-0317 through INSuRE+C, as well as the US National Science Foundation under grants 2245223 and 2608885.

References

[1] Ahmed Abdo, Sakib Md Bin Malek, Zhiyun Qian, Qi Zhu, Matthew Barth, and Nael Abu-Ghazaleh. Application level attacks on connected vehicle protocols. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*, pages 459–471, 2019.

[2] Monowar H Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal K Kalita. Network anomaly detection: methods, systems and tools. *Ieee communications surveys & tutorials*, 16(1):303–336, 2013.

[3] Norbert Bißmeyer, Sebastian Mauthofer, Kpatcha M Bayarou, and Frank Kargl. Assessment of node trustworthiness in vanets using data plausibility checks with particle filters. In *2012 IEEE Vehicular Networking Conference (VNC)*, pages 78–85. IEEE, 2012.

[4] Lara Codeca, Raphaël Frank, and Thomas Engel. Luxembourg sumo traffic (lust) scenario: 24 hours of mobility for vehicular networking research. In *2015 IEEE Vehicular Networking Conference (VNC)*, pages 1–8. IEEE, 2015.

[5] Pritam Dash, Mehdi Karimibiuki, and Karthik Pattabiraman. Out of control: stealthy attacks against robotic vehicles protected by control-based techniques. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 660–672, 2019.

[6] Pritam Dash, Guanpeng Li, Zitao Chen, Mehdi Karimibiuki, and Karthik Pattabiraman. Pid-piper: Recovering robotic vehicles from physical attacks. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 26–38. IEEE, 2021.

[7] Stefan Dietzel, Rens van der Heijden, Hendrik Decke, and Frank Kargl. A flexible, subjective logic-based framework for misbehavior detection in v2v networks. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pages 1–6. IEEE, 2014.

[8] Jeremy Erickson, Shibo Chen, Melisa Savich, Shengtuo Hu, and Z Morley Mao. Commpact: Evaluating the feasibility of autonomous vehicle contracts. In *2018 IEEE Vehicular Networking Conference (VNC)*, pages 1–8. IEEE, 2018.

[9] TCITS ETSI. Intelligent transport systems (its); vehicular communications; basic set of applications; part 2: Specification of cooperative awareness basic service. *Draft ETSI TS*, 20(2011):448–51, 2011.

[10] M. Hasan, S. Mohan, T. Shimizu, and H. Lu. Securing vehicle-to-everything (v2x) communication platforms. *IEEE Transactions on Intelligent Vehicles*, 5(4):693–713, 2020.

[11] Joseph Kamel, Mohammad Raashid Ansari, Jonathan Petit, Arnaud Kaiser, Ines Ben Jemaa, and Pascal Urien. Simulation framework for misbehavior detection in vehicular networks. *IEEE transactions on vehicular technology*, 69(6):6631–6643, 2020.

[12] Joseph Kamel, Michael Wolf, Rens Wouter van der Heijden, Arnaud Kaiser, Pascal Urien, and Frank Kargl. VeReMi Extension: A Dataset for Comparable Evaluation of Misbehavior Detection in VANETs. In *IEEE International Conference on Communications (ICC)*, Dublin (virtual), Ireland, June 2020. Virtual conference.

[13] Charlie Miller and Chris Valasek. Adventures in automotive networks and control units. *Def Con*, 21(260-264):15–31, 2013.

[14] Mujahid Muhammad and Ghazanfar Ali Safdar. Survey on existing authentication issues for cellular-assisted v2x communication. *Vehicular Communications*, 12:50–65, 2018.

[15] Orange Travel. How much data does spotify use?, March 28 2025. Accessed: 2026-02-24.

[16] Raul Quinonez, Jairo Giraldo, Luis Salazar, Erick Bauman, Alvaro Cardenas, and Zhiqiang Lin. {SAVIOR}: Securing autonomous vehicles with robust physical invariants. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 895–912, 2020.

[17] ResearchAndMarkets.com. Automotive microcontroller unit (mcu) industry report 2023: Automotive high-end mcus will still be in short supply, and how oems can break the situation, March 2023. Accessed: 2026-01-28.

[18] SAE. *Dedicated Short Range Communications (DSRC) Message Set Dictionary*, March 2016.

[19] Fatih Sakiz and Sevil Sen. A survey of attacks and detection mechanisms on intelligent transportation systems: Vanets and iov. *Ad Hoc Networks*, 61:33–50, 2017.

[20] Md Hasan Shahriar, Mohammad Raashid Ansari, Jean-Philippe Monteuuis, Cong Chen, Jonathan Petit, Y Thomas Hou, and Wenjing Lou. Vehigan: Generative adversarial networks for adversarially robust v2x

misbehavior detection systems. In *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*, pages 1294–1305. IEEE, 2024.

[21] Abhishek B Sharma, Leana Golubchik, and Ramesh Govindan. Sensor faults: Detection methods and prevalence in real-world datasets. *ACM Transactions on Sensor Networks (TOSN)*, 6(3):1–39, 2010.

[22] Junjie Shen, Jun Yeon Won, Zeyuan Chen, and Qi Alfred Chen. Drift with devil: Security of multi-sensor fusion based localization in high-level autonomous driving under {GPS} spoofing. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 931–948, 2020.

[23] Steven So, Jonathan Petit, and David Starobinski. Physical layer plausibility checks for misbehavior detection in v2x networks. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, pages 84–93, 2019.

[24] Steven So, Prinkle Sharma, and Jonathan Petit. Integrating plausibility checks and machine learning for misbehavior detection in vanet. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 564–571. IEEE, 2018.

[25] Christoph Sommer, David Eckhoff, Alexander Brummer, Dominik S Buse, Florian Hagenauer, Stefan Joerer, and Michele Segata. Veins: The open source vehicular network simulation framework. In *Recent Advances in Network Simulation*, pages 215–252. Springer, 2019.

[26] Jan Peter Stotz, Norbert Bißmeyer, Frank Kargl, Stefan Dietzel, Panos Papadimitratos, and Christian Schleiffer. Preserve d1. 1 security requirements of vehicle security architecture. *Deliverable, PRESERVE consortium*, 2011.

[27] Beshr Sultan and Mike McDonald. Assessing the safety benefit of automatic collision avoidance systems (during emergency braking situations). In *Proceedings of the 18th International Technical Conference on the Enhanced Safety of Vehicle.(DOT HS 809 543)*, 2003.

[28] Mingshun Sun, Ming Li, and Ryan Gerdes. A data trust framework for vanets enabling false data detection and secure vehicle tracking. In *2017 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9. IEEE, 2017.

[29] Franco van Wyk, Yiyang Wang, Anahita Khojandi, and Neda Masoud. Real-time sensor anomaly detection and identification in automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):1264–1276, 2019.

[30] Xiaoyang Wang, Ioannis Mavromatis, Andrea Tassi, Raul Santos-Rodriguez, and Robert J Piechocki. Location anomalies detection for connected and autonomous vehicles. In *2019 IEEE 2nd Connected and Automated Vehicles Symposium (CAVS)*, pages 1–5. IEEE, 2019.

[31] Wufei Wu, Renfa Li, Guoqi Xie, Jiyao An, Yang Bai, Jia Zhou, and Keqin Li. A survey of intrusion detection for in-vehicle networks. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):919–933, 2019.

[32] Yuan Yao, Bin Xiao, Gaofei Wu, Xue Liu, Zhiwen Yu, Kailong Zhang, and Xingshe Zhou. Multi-channel based sybil attack detection in vehicular ad hoc networks using rssi. *IEEE Transactions on Mobile Computing*, 18(2):362–375, 2018.

[33] Chaitanya Yavvari, Zoran Duric, and Duminda Wijesekera. Vehicular dynamics based plausibility checking. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8. IEEE, 2017.

A Appendix

A.1 Bandwidth

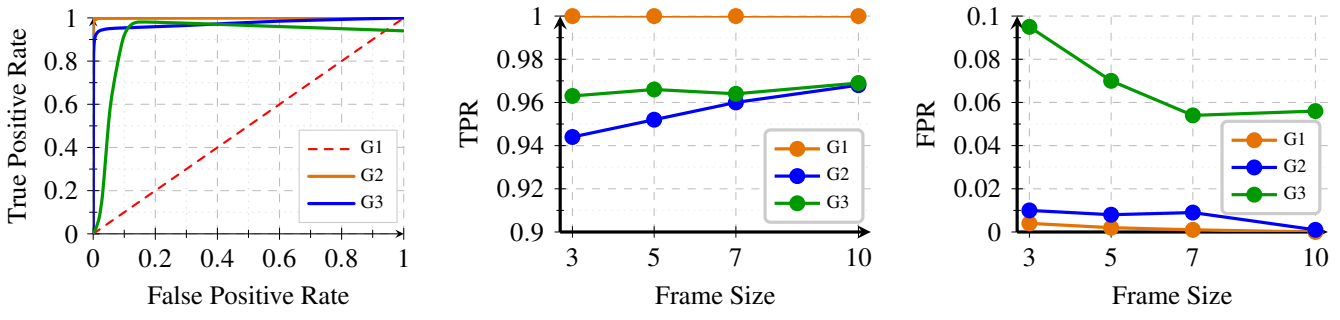
We would like to discuss how many of the *potentially anomalous*-flagged frames from Phase 1 have to be sent to Phase 2. This will have an impact on bandwidth which is another important metric for an OEM since it drives up cost. The car will very likely use the built-in cellular connection. Each BSM has a default size of 254 bytes and will be sent using the Transmission Control Protocol (TCP). This is necessary since we cannot assume that the network connection during a trip will always be reliable. Together with the TCP/IP header overhead, each packet including the BSM will have a total size of 318 bytes. Table 5 (in Appendix) summarizes the mean percentage of frames in each experiment to be sent to Phase 2 (a mean number of frames to be sent to Phase 2 d_{p2} over total number of frames d_{total} for given vehicle), as well as the required bandwidth b.

$$b = \frac{d_{p2} \cdot 318bytes}{d_{total} \cdot 0.1s} \tag{2}$$

The required bandwidth is always < 0.56 kB/s if only every flagged frame is transmitted. At most, 1.8 MB is transmitted per hour, less than one minute of high-quality Spotify streaming (320 kbps) [15]. Table 5 uses $f = 10$; smaller frame sizes reduce bandwidth by sending fewer frames to Phase 2. Thus, the table shows an upper bound. Larger frames improve performance but increase bandwidth, making frame size a trade-off between performance and cost.

Table 4: Phase 2 per-sample performance for frame size $f = 10$. AT5 and AT6 are independent of correlation group G* and are displayed for G1 merely for visualization purposes. Best values per attack type are bolded.

Model	Attack Type	G1			G2			G3		
		TPR	FPR	F-1	TPR	FPR	F-1	TPR	FPR	F-1
RF	AT1	93.46%	3.24%	90.30%	**96.90%**	0.57%	**94.77%**	91.36%	**0.02%**	91.44%
	AT2	**76.06%**	**0.53%**	**77.81%**	68.62%	1.30%	69.35%	68.30%	0.83%	69.60%
	AT3	94.53%	9.06%	87.45%	97.14%	**1.00%**	**94.03%**	75.04%	13.02%	75.06%
	AT4	**91.16%**	1.21%	**89.50%**	88.53%	1.26%	87.04%	52.16%	0.89%	52.02%
	AT5	**92.75%**	**0.23%**	**93.80%**	-	-	-	-	-	-
	AT6	**87.43%**	2.89%	**86.16%**	-	-	-	-	-	-
	AT7, $b = 0.1$	62.93%	1.08%	64.57%	**86.37%**	1.71%	**84.29%**	70.29%	**0.50%**	71.15%
	AT7, $b = 0.01$	60.01%	**0.32%**	61.35%	**78.99%**	1.87%	**78.20%**	64.68%	0.46%	65.49%
	AT7, $b = 0.001$	61.12%	**0.25%**	62.02%	**83.56%**	0.83%	**83.74%**	67.70%	0.40%	68.41%
SVM	AT1	72.56%	2.01%	71.79%	77.91%	1.39%	77.14%	62.66%	0.98%	62.45%
	AT2	68.65%	9.76%	61.23%	56.12%	3.61%	56.76%	64.25%	1.02%	64.41%
	AT3	**97.79%**	1.79%	91.85%	95.61%	1.03%	90.86%	69.44%	14.56%	69.98%
	AT4	88.70%	1.20%	86.59%	84.55%	1.52%	82.85%	50.12%	**0.0009%**	49.37%
	AT5	76.40%	3.99%	76.72%	-	-	-	-	-	-
	AT6	77.38%	**0.36%**	82.03%	-	-	-	-	-	-
	AT7, $b = 0.1$	51.54%	0.94%	50.56%	66.27%	11.22%	56.75%	54.03%	1.15%	53.27%
	AT7, $b = 0.01$	50.41%	0.72%	50.24%	64.83%	7.73%	61.88%	52.60%	1.09%	51.15%
	AT7, $b = 0.001$	50.50%	0.49%	50.08%	68.18%	8.23%	62.83%	55.19%	0.84%	54.12%
DNN	AT1	82.90%	2.85%	78.94%	85.31%	1.25%	83.71%	62.32%	2.23%	61.54%
	AT2	69.52%	5.75%	65.08%	55.01%	3.32%	55.14%	59.88%	0.56%	60.71%
	AT3	94.65%	1.58%	90.85%	95.37%	**1.00%**	90.72%	73.38%	2.74%	70.35%
	AT4	83.20%	3.80%	76.37%	86.09%	2.72%	80.06%	50.00%	**0.0009%**	49.05%
	AT5	76.68%	6.14%	76.39%	-	-	-	-	-	-
	AT6	79.91%	**0.36%**	84.56%	-	-	-	-	-	-
	AT7, $b = 0.1$	51.26%	0.51%	50.46%	66.07%	20.82%	58.38%	54.81%	1.67%	55.45%
	AT7, $b = 0.01$	50.21%	0.47%	49.71%	66.68%	8.65%	62.57%	55.02%	1.98%	53.93%
	AT7, $b = 0.001$	50.20%	**0.25%**	49.94%	66.68%	8.65%	62.57%	54.05%	0.48%	54.67%



(a) Combined ROC Curves for θ_s^i

(b) TPR vs Frame Size

(c) FPR vs Frame Size

Figure 4: Phase 1 ROC curves and per-frame performance based on frame size.

Table 5: Bandwidth considerations

Sensor Group G_i	**Attack Type** AT^*	**Mean Percentage (%)**	**Bandwidth (kB/s)**
G_1	AT_1	5.3	0.17
	AT_2	5.2	0.17
	AT_3	5.2	0.17
	AT_4	5.3	0.17
	AT_5	8.7	0.28
	AT_6	7.6	0.24
G_2	AT_1	3.9	0.12
	AT_2	4.2	0.13
	AT_3	4.5	0.14
	AT_4	4.7	0.15
	AT_5	3.0	0.09
	AT_6	2.7	0.08
G_3	AT_1	14.5	0.46
	AT_2	14.0	0.45
	AT_3	15.0	0.48
	AT_4	15.6	0.50
	AT_5	17.9	0.56
	AT_6	17.8	0.56

Table 6: Detection Latency and RAM Usage

Phase	**Model**	**Frame Size**	**Latency (ms)**		**RAM Usage (MB)**	
			RPi	Desktop	RPi	Desktop
Phase 1	N/A	1	0.10	N/A	38.8	N/A
Phase 2	RF	3	98.9	15.1	106	153
		5	95.1	15.2	106	153
		7	99.4	15.6	106	153
		10	101.9	16.0	106	153
	SVM	3	30.1	3.8	102	147
		5	31.9	4.2	102	147
		7	33.5	4.6	102	147
		10	38.8	4.9	102	147
	DNN	3	570	41.5	357	485
		5	532	39.5	357	485
		7	528	39.9	357	485
		10	538	40.4	357	485

WIP: SNFuzz: Gateway-Centric State-Oriented Fuzzing for MQTT-SN

Christian de la Pena
Florida International University

Samin Y. Chowdhury
Florida International University

Mohammad Rahman
Florida International University

Weidong Zhu
Florida International University

Ruimin Sun
Florida International University

Abstract

Modern vehicles are increasingly evolving into complex IoT systems that rely on lightweight messaging protocols for Vehicle-to-Everything (V2X) communication. MQTT-SN, a UDP-based variant of MQTT designed for low-power and resource-constrained devices, is widely adopted in such environments. However, MQTT-SN's reliance on stateless transport and gateway-centric design introduces additional attack surfaces, particularly in the form of implicit, state-dependent protocol logic maintained at gateways. This paper presents SNFuzz, a gateway-centric and state-oriented fuzzing framework designed to systematically explore vulnerabilities in MQTT-SN gateway implementations. SNFuzz models gateway-managed protocol states and enables exploration of deep, state-dependent behaviors through response-driven execution. This design allows SNFuzz to uncover vulnerabilities that are difficult to detect using existing fuzzers designed for connection-oriented protocols. We implement a prototype of SNFuzz and evaluate it on three MQTT-SN gateway implementations. Compared with state-of-the-art network fuzzers, SNFuzz achieves higher coverage of gateway-managed states and systematically explores deep protocol paths that prior approaches fail to reach. Our evaluation identifies a high-severity denial-of-service vulnerability in EMQX, which has been reported to the developers. Beyond MQTT-SN, our findings highlight the importance of state-aware fuzzing for gateway-centric IoT protocols operating over stateless transports.

1 Introduction

Modern vehicles are increasingly evolving into software-defined and IoT-enabled systems that rely on distributed communication across sensors, roadside infrastructure, and cloud services. Lightweight messaging protocols play a critical role in supporting Vehicle-to-Everything (V2X) communication under constrained network conditions. MQTT for Sensor Networks (MQTT-SN), a UDP-based variant of MQTT designed for low-power and resource-constrained devices, reduces protocol overhead and replaces topic strings with compact identifiers, making it well-suited for vehicular environments [8, 14, 16].

Unlike TCP-based MQTT, MQTT-SN shifts session, topic, and subscription management logic from a centralized broker to an intermediate gateway operating over connectionless transport. As summarized in Table 1, this client–gateway–broker architecture introduces key differences, including the use of UDP and the absence of persistent connections. Consequently, gateways must maintain implicit client state, manage topic translation tables, and handle sleep-state transitions without strong transport-layer guarantees. These design characteristics expand the attack surface of MQTT-SN gateways and introduce additional sources of implementation complexity [23, 26].

While prior work has examined MQTT-SN security from attack modeling, and the design of lightweight security enhancements, these efforts primarily focus on communication security and protocol semantics rather than systematic vulnerability discovery in implementations [3, 11, 13, 19, 22]. In parallel, a large body of research has explored fuzzing techniques for network protocols, spanning specification-driven, learning-based, and multi-party/stateful approaches [2, 6, 17, 20, 27, 29]. Many of these techniques have been successfully applied to MQTT over TCP, where persistent connections and well-defined session semantics simplify state modeling. However, these assumptions do not directly extend to MQTT-SN gateways, which operate over stateless transport and maintain protocol state implicitly across message sequences. As a result, existing approaches often exercise shallow parsing logic while failing to explore deeper gateway-managed states involving topic registration, subscription handling, and session transitions.

Core Insight. The key challenge in securing MQTT-SN arises from a broader class of gateway-centric, stateful protocols operating over stateless transports. In such systems, vulnerabilities are not triggered by isolated malformed packets, but by sequences of state-dependent interactions that manipu-

Table 1: Comparison between MQTT and MQTT-SN

Feature	MQTT	MQTT-SN
Transport	TCP	UDP / BLE / Non-TCP
Broker Connection	Direct	Via Gateway
Topic Representation	Full Topic String	Short Topic ID
Designed For	General IoT	Sensors
Sleep Support	Limited	Native Support

late gateway-maintained session, topic, and subscription state. Therefore, effective vulnerability discovery requires modeling the evolution of gateway-side state across message sequences, rather than treating inputs as independent events.. This observation motivates a state-oriented fuzzing strategy that explicitly captures gateway-managed protocol behavior.

To address this challenge, we design `SNFuzz`, a gateway-centric, state-oriented fuzzing framework instantiated for MQTT-SN. While building on existing coverage-guided fuzzing techniques, `SNFuzz` introduces a design that models gateway-maintained session establishment, topic registration, and subscription management as explicit state transitions. Instead of batching inputs, it performs single-message, response-driven execution, where each transmitted message advances exploration based on the gateway's response. `SNFuzz` further incorporates protocol-aware mutation strategies conditioned on the current protocol state, as well as response-type-aware feedback to prioritize inputs that lead to previously unexplored gateway states. This design enables systematic exploration of deep protocol behaviors that are difficult to reach with existing fuzzers designed for connection-oriented or broker-centric protocols.

We implement a prototype of `SNFuzz` and evaluate it on three MQTT-SN gateway implementations, including an open-source simulator and widely used deployments. We compare `SNFuzz` with representative state-of-the-art network fuzzers, including AFLNet and Boofuzz. Across all implementations, `SNFuzz` achieves the highest coverage of gateway-managed states and systematically reaches deep protocol paths involving topic registration and subscription management that prior approaches fail to reach. Our evaluation identifies a high-severity denial-of-service vulnerability in EMQX, which has been responsibly disclosed to the developers.

Although our evaluation focuses on MQTT-SN, the proposed design applies more broadly to gateway-centric IoT protocols that maintain application-layer state over stateless transports. These include systems where protocol correctness depends on multi-step message sequences and where state must be reconstructed without transport-layer guarantees.

This paper makes the following contributions:

- **State-Oriented Fuzzing Design.** We present a state-oriented fuzzing framework for gateway-centric, UDP-based protocols, instantiated for MQTT-SN, enabling systematic exploration of state-dependent protocol logic.
- **Response-Driven Exploration.** We introduce a response-driven execution model and protocol-aware mutation strategies tailored to stateful protocols operating over stateless transports.
- **Empirical Security Evaluation.** We demonstrate that `SNFuzz` achieves higher gateway-state coverage than existing fuzzers and identify a high-severity vulnerability in a real-world MQTT-SN gateway implementation.

2 Background and Related Work

2.1 MQTT-SN Architecture and Security

MQTT is a lightweight publish–subscribe protocol widely used in IoT and vehicular systems, where clients connect to a centralized broker over TCP. The broker maintains all session and subscription states. MQTT-SN extends MQTT to support highly constrained and low-power wireless environments [8]. In automotive settings, MQTT-SN enables telemetry collection from Electronic Control Units (ECUs), tire pressure monitoring systems, and battery management systems in electric vehicles. It also supports vehicle-to-infrastructure (V2I) communication, where roadside units collect sensor data from vehicles, as well as fleet management applications for real-time tracking and diagnostics. In these deployments, gateways are often embedded within telematics control units (TCUs) or edge computing modules, making them critical security boundaries.

Unlike MQTT, MQTT-SN operates primarily over UDP and introduces an intermediate gateway, forming a client–gateway–broker architecture. We refer to MQTT-SN as a *gateway-centric protocol*, where the gateway acts as a stateful and active participant responsible for session management, protocol translation, and coordination, in contrast to MQTT's broker-centric architecture where these responsibilities reside primarily at the broker.

The gateway maintains client session states, translates message formats, and manages dynamic topic registrations by mapping topic strings to numeric identifiers between MQTT-SN (UDP) clients and MQTT (TCP) brokers. Since UDP lacks transport-layer reliability, the gateway also explicitly handles application-layer retransmissions, duplicate detection, and timeout management. Its operation involves multiple classes of state, including session lifecycle states (e.g., connect, active, sleep, disconnect), dynamic topic registration states for identifier mappings, and message reliability states for retransmissions and QoS handling. Additional protocol messages, such as `ADVERTISE`, `SEARCHGW`, and `REGISTER`, further expand the gateway's state machine compared to a standard MQTT broker, as summarized in Table 2. These interactions create deep states, where correct behavior depends on multi-step message sequences and cross-dependent state transitions that persist across packets rather than being triggered by a single input.

These design characteristics significantly enlarge the protocol state compared to MQTT, increasing the likelihood of parser flaws, state desynchronization, and resource exhaustion vulnerabilities. Accordingly, prior work has examined the security posture and deployment risks of MQTT and MQTT-SN [3, 11, 13, 19, 23], including reflection-based denial-of-service attacks exploiting MQTT-SN's use of UDP [26], and secure protocol design with cryptographic enhancements [15, 24, 25]. However, while these studies identify structural weaknesses, such as the lack of built-in authentication, they do not provide systematic methods to discover implementation-level flaws in gateway software.

2.2 Fuzzing Techniques

Fuzz testing is a widely used technique for assessing software security by generating unexpected or malformed inputs to uncover security vulnerabilities. General-purpose network fuzzers such as AFLNet [21] and its extensions [5, 9, 28] have successfully extended coverage-guided fuzzing to stateful network services.

In the IoT ecosystem, fuzzing campaigns have uncovered hundreds of vulnerabilities in popular MQTT implementations [6]. Numerous tools have been developed to target MQTT brokers. These approaches generally fall into three categories: specification-driven fuzzers that leverage protocol grammars, learning-based fuzzers that infer input structures or state transitions, and multi-party or stateful fuzzers that model interactions across multiple protocol entities. Representative examples include specification-driven fuzzers such as FUME [20] and MQueez [17], learning-based approaches [2, 29], and multi-party fuzzers such as MBFuzzer [27] and MultiFuzz [30]. However, these tools fundamentally assume TCP-based communication and broker-centric state management. As a result, they are not directly applicable to MQTT-SN, which relies on UDP-based transport and gateway-managed protocol state. In this setting, state is no longer enforced by the transport but must be reconstructed at the application level, while existing fuzzers rely on connection-oriented ordering guarantees. Notably, a stateless transport does not imply a stateless protocol: MQTT-SN correctness depends on multi-step message sequences (e.g., CONNECT → REGISTER → PUBLISH → ACK), which introduce state dependencies across packets. While UDP introduces transport-level challenges such as loss and reordering, MQTT-SN further amplifies these issues through protocol-specific state maintained at the gateway.

Edge gateways and protocol translation layers form critical security boundaries in vehicular and IoT systems [12]. Recent work emphasizes the necessity of robust testing of edge middleware components, as demonstrated by tools such as OCPPStorm for EV charging protocols [7] and SHiFT for embedded applications [18]. Despite the central role of gateways in MQTT-SN deployments, they remain underexplored with respect to automated fuzzing and implementation-level vulnerability discovery. In this work, we focus on protocol-level fuzzing of MQTT-SN gateway implementations, rather than over-the-air wireless fuzzing. To the best of our knowledge, no prior work systematically applies coverage-guided fuzzing to MQTT-SN gateway implementations under these constraints.

3 Threat Model

The MQTT-SN gateway serves as the critical bridge between resource-constrained edge devices and the centralized MQTT Broker, maintaining protocol state and managing communication across a large number of client nodes. As a result, it represents a critical point for security analysis. In this work, we target the MQTT-SN gateway implementation, focusing on its message parsing and state-handling logic. We consider two primary adversary objectives: (1) to trigger a Denial of Service (DoS) attack that disrupts telemetry and message routing for connected clients [23] and (2) to exploit implementation flaws to achieve remote code execution on the gateway. Successful exploitation enables manipulation of translation tables, interception or spoofing of sensor data, and broader compromise of system availability and integrity.

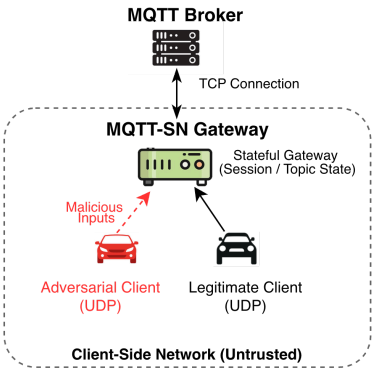


Figure 1: Gateway-facing Message Flows and Adversarial Inputs.

We assume an adversary that interacts with the gateway as a malicious MQTT-SN client. MQTT-SN operates over connectionless transports without requiring authentication by default [16], allowing an attacker to send arbitrary datagrams directly to the gateway's client-facing interface. As illustrated in Figure 1, the attacker does not require control over the underlying network but can inject protocol-compliant or malformed datagrams that influence gateway behavior [23]. This attack surface includes protocol parsers for connection establishment (`CONNECT`), topic registration (`REGISTER`), and state management (`PINGREQ`, `DISCONNECT`) [23].

Since the gateway must parse incoming messages and allocate resources prior to applying external validation or filtering,

it is particularly susceptible to pre-authentication vulnerabilities. These include memory exhaustion, malformed message handling, and inconsistencies in state transitions. Such vulnerabilities are especially critical in MQTT-SN due to the gateway's responsibility for maintaining implicit session state, buffering messages, and managing sleep-state transitions. As a result, errors in these mechanisms can lead to severe denial-of-service conditions or corruption of internal data structures [10].

4 Design

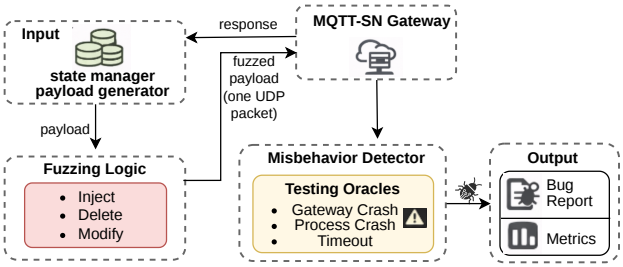


Figure 2: SNFuzz Workflow.

Figure 2 illustrates the overall workflow of our MQTT-SN gateway fuzzer SNFuzz. The design is guided by three principles: (1) model gateway-centric protocol state, (2) respect UDP single-message processing semantics, and (3) use response feedback for state exploration.

4.1 Gateway-Centric State Model

MQTT-SN gateways maintain persistent, structured state across client interactions. Specifically, a gateway tracks (1) client bindings via the (source IP, source port) tuple and ClientID, (2) a topic translation table mapping topic names to short identifiers, (3) subscription state governing publish delivery, and (4) sleep/awake status and buffered messages for battery-optimized clients.

To systematically exercise this logic, our fuzzer models gateway state progression explicitly. Each fuzzing round follows a protocol session path beginning with session establishment and advancing through stateful transitions. Rather than treating fuzzing as a stochastic sequence of independent inputs, we represent execution as protocol-aware state exploration.

State Structure. The model consists of four principal states:

- S_0 **(Session Initialization)** establishes a client context by selecting a transport tuple (source port, source IP address) and issuing CONNECT message, ensuring the gateway binds the clientID to the transport tuple.

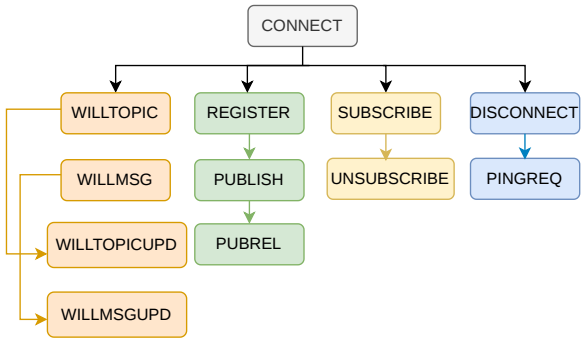


Figure 3: Message Generator Types and Sequences

- S_1 **(Valid Message Construction)** generates a syntactically valid MQTT-SN message selected from the protocol-defined message set (Figure 3).
- S_2 **(Controlled Perturbation)** applies protocol-aware mutation to the constructed message while preserving surrounding session semantics.
- S_f **(Round Termination)** concludes the current session round.

State transitions correspond to observable protocol actions. Message types act as triggers for gateway-side state changes. For example, REGISTER allocates topic-table entries, SUBSCRIBE updates topic subscriptions, DISCONNECT transitions the client toward termination or sleep, and PINGREQ exercises keep-alive or wake-up handling. The corresponding response types (Table 2) confirm whether the intended transition occurred by parsing the response from the gateway. By aligning fuzzing states with gateway-maintained structures, each round becomes a structured exploration of stateful behavior rather than a sequence of independent packet mutations. This enables systematic coverage of state-dependent logic that is difficult to reach using stateless or purely message-centric fuzzing [21].

Table 2: Message Parser Types

Name (ID)	Description
ADVERTISE (00)	Gateway broadcast to announce presence.
GWINFO (02)	GW ID/address response to SEARCHGW.
CONNACK (05)	Response to connection request.
WILLTOPICREQ (06)	Gateway request for client Will topic.
WILLMSGREQ (08)	Gateway request for client Will message.
REGACK (0B)	Acknowledges topic registration.
PUBACK (0D)	Acknowledges PUBLISH or reports error.
PUBREC (0E)	QoS 2 publish receipt acknowledgment.
PUBCOMP (10)	QoS 2 publish exchange completion.
SUBACK (13)	Acknowledges SUBSCRIBE and Topic ID.
UNSUBACK (15)	Acknowledges UNSUBSCRIBE request.
PINGRESP (17)	PINGREQ response or buffer empty.
WILLTOPICRESP (1B)	Acknowledges Will topic update.
WILLMSGRESP (1D)	Acknowledges Will message update.

4.2 Single-message, Response-Driven Execution

Single-message execution. During preliminary experimentation across multiple MQTT-SN gateways, we observed that UDP payloads containing multiple MQTT-SN messages are rejected with a "malformed length" error. In practice, gateways process exactly one MQTT-SN message per UDP packet. This constraint fundamentally shapes our execution model. Accordingly, we adopt a strictly atomic execution strategy: each mutation is transmitted as a single MQTT-SN message and evaluated independently based on the gateway's response. We do not batch messages. Each execution step proceeds as follows: (1) Generate a syntactically valid MQTT-SN message based on the current modeled state; (2) Optionally apply protocol-aware mutation; (3) Send exactly one MQTT-SN message to the gateway; (4) Parse the gateway response (if any).

Response-aware feedback. The gateway's responses determine whether: (1) a new protocol state has been reached (e.g., successful registration, subscription acknowledgment, sleep transition), (2) the input was ignored (no response), or (3) a failure condition occurred (connection reset or abnormal termination). Only after this evaluation does the engine decide whether to continue exploring the current session path or terminate the round.

This single-message, response-driven execution model provides two key advantages. First, it ensures precise attribution between an input mutation and its observed effect, which is critical for debugging state-dependent vulnerabilities. Second, it mirrors the actual operational constraints of MQTT-SN gateways, preventing unproductive exploration patterns. As a result, fuzzing becomes a controlled exploration of gateway state transitions, where each input is an atomic protocol step.

4.3 State-Aware Mutation and Feedback

Our fuzzer incorporates state-aware mutation sequence and feedback to guide exploration toward semantically distinct gateway behaviors.

Response-Aware Mutation. We implement three mutation operators: inject, delete, and modify, parameterized by configurable weights which enables controlled adjustment of structural perturbation choices. **Inject** inserts additional bytes or protocol fields. **Delete** removes selected bytes or field segments. **Modify** alters existing byte values. Unlike blind byte-level fuzzing, mutations are conditioned on the current protocol stage. For example, during `REGISTER`, mutations target topic identifiers; during `SUBSCRIBE`, they focus on topic IDs and QoS flags. Conditioning mutations on gateway state increases the likelihood of triggering state-dependent parsing and logic faults in topic-table management, subscription handling, and sleep-state transitions.

Response-Based Novelty Detection To interpret gateway behavior, we implement parsers for each MQTT-SN response type (e.g., `CONNACK`, `REGACK`, `SUBACK`, `PUBACK`). Responses are normalized by removing highly volatile fields such as `TopicId`, enabling semantic deduplication. A response is considered novel if its normalized field values have not been observed. Messages leading to novel responses are added to a response corpus, which seeds subsequent rounds. This mechanism prioritizes inputs that expand reachable gateway states rather than merely increasing syntactic diversity.

Crash Inference and Failure Classification Because gateways may not explicitly report internal errors, we classify failures based on observable behavior: (1) No response: potential silent drop; (2) Explicit error response: protocol-level rejection; and (3) Connection reset or abnormal termination: inferred crash. All messages sent during a round that ends in abnormal termination are preserved to enable deterministic reproduction. Together, response-aware mutation and semantic feedback transform fuzzing into structured state exploration. Gateway responses determine which paths are promising, which mutations expand reachable states, and which inputs expose stability violations. Over time, the fuzzer constructs a corpus of state-distinguishing inputs that systematically exercise deep, stateful gateway logic.

5 Preliminary Evaluation

5.1 Experimental Setup

Targets. We evaluated three widely used MQTT-SN gateway implementations: EMQX (v6.1.0), Eclipse Paho (v1.3), and HiveMQ Edge (v2026.2). Each gateway was deployed as a standalone service exposing MQTT-SN client-facing interfaces over UDP and broker-facing interfaces over TCP. These gateways maintain complex internal state, including client bindings, topic translation tables, subscription state, and sleep-state handling, making them suitable targets for evaluating stateful fuzzing techniques.

Baselines. We compare `SNFuzz` against two established network fuzzers: **AFLNet** (a coverage-guided network protocol fuzzer built on AFL), and **Boofuzz** (a generation-based black-box protocol fuzzer). To ensure a fair comparison, all fuzzers interact with the gateway through the same client-facing interface and operate under identical assumptions. Because of the black-box assumption, we align comparisons with AFLNet at the level of input mutation and protocol exploration strategies.

AFLNet Configuration. AFLNet requires seed inputs to bootstrap protocol exploration. We seeded AFLNet with approximately 50 valid MQTT-SN packet sequences captured from client–gateway interactions using the Eclipse Paho Gateway and an open-source MQTT-SN client [14]. These seeds were transmitted as UDP datagrams to the gateway's client-facing interface and include representative sequences such

as CONNECT, REGISTER, SUBSCRIBE, and PUBLISH. The same seed corpus was used across all gateway implementations to ensure consistency.

BooFuzz Configuration. The BooFuzz configuration was constructed based on the MQTT-SN v1.2 protocol specification. It included static fields such as message type and length (which were not fuzzed to avoid excessive packet drops) and dynamic fields such as ClientId, TopicName, and payload, which were used as fuzzing candidates. BooFuzz also modeled protocol session graphs to enable exploration of valid message sequences (e.g., CONNECT → REGISTER → PUBLISH). Similar to SNFuzz and AFLNet, BooFuzz interacts with the gateway via socket-based message injection over the client-facing interface.

Execution Model. Our evaluation models the adversary as a malicious MQTT-SN client interacting directly with the gateway's client-facing interface. Fuzzing inputs are injected as UDP datagrams over a local network interface, emulating the behavior of a client without requiring over-the-air wireless communication. While Figure 1 illustrates a conceptual deployment scenario, our experiments focus on protocol-level behavior and do not depend on physical network conditions. This abstraction enables controlled and reproducible evaluation of gateway parsing and state-handling logic.

Environment. All experiments were conducted under identical time budgets (3 hours per target) on the same hardware platform (i7-8650U [4 cores], 16GB RAM, Ubuntu 24.04.3) in a native (non-virtualized) environment. All gateways were executed locally.

Metrics. Because no standardized fuzzing benchmarks exist for MQTT-SN, we approximate protocol coverage using unique normalized network responses. A response is considered unique based on packet type (e.g., CONNACK, REGACK, PUBACK), flag configuration, and relevant field values (excluding highly volatile fields). This metric reflects the diversity of protocol states explored.

5.2 Coverage Results

Table 3 compares unique response counts across all gateways. Across all targets, SNFuzz consistently achieved the highest response diversity. Specifically, SNFuzz was able to generate 22, 13, and 17 unique network responses for EMQX, Eclipse Paho, and HiveMQ Edge, respectively. On average, SNFuzz improves total unique response coverage by approximately 104.2% over AFLNet and 29.4% over Boofuzz. Importantly, SNFuzz uniquely triggered WILLTOPICREQ and WILLMSGREQ states in Eclipse Paho, which were unreachable by both AFLNet and Boofuzz. These states require coordinated multi-step session sequences and cannot be reached through stateless mutation. Further, SNFuzz generated the maximum number of unique PUBACK and SUBACK responses across all implementations. Since deep-state network responses like an accepted PUBACK requires the successful execution of prerequisite message sequences like CONNECT, REGISTER, etc. This indicates SNFuzz's ability to reach deep exploration of topic registration and subscription state.

Table 3: Network Response Coverage Comparison Across three MQTT-SN Gateway Implementations

Impl.	Fuzzer	CONNACK	WILLTOPICREQ	WILLMSGREQ	REGACK	PUBACK	SUBACK	UNSUBACK	Total
EMQX	AFLNet	3	0	0	3	2	3	1	12
	Boofuzz	5	1	1	4	3	4	1	18
	SNFuzz	5	1	1	4	5	5	1	**22**
Paho	AFLNet	2	0	0	1	1	2	0	6
	Boofuzz	2	0	0	2	2	2	1	9
	SNFuzz	3	1	1	2	3	2	1	**13**
Hive	AFLNet	2	0	0	2	2	1	1	8
	Boofuzz	3	1	1	2	3	3	1	14
	SNFuzz	4	1	1	3	4	3	1	**17**

The performance of the baseline tools is consistent with their ability to handle black-box scenarios. AFLNet had the worst performance due to its inability to pass initial syntax and state tests. Boofuzz performed better due to its generation-based approach and its ability to utilize predefined protocol templates. This made it easy for Boofuzz to pass initial CONNECT states. However, Boofuzz is limited in dynamically satisfying the complex state dependencies.

5.3 Execution Rate

We measure execution throughput as packets transmitted per second. As Table 4 shows, SNFuzz has lower throughput due to atomic, response-gated execution and protocol-state tracking. This is due to the stochastic gateway state transitions, which is necessary to penetrate into protocol session states. Despite reduced raw execution speed, it achieves higher state coverage. This demonstrates that structured state exploration can outperform high-throughput stateless fuzzing.

Table 4: Execution Throughput Comparison

Fuzzer	Executions per Second
AFLNet	150
BooFuzz	62
SNFuzz	22

5.4 A Case Study of an MQTT-SN Bug

Attack Mechanism. The deeper state exploration capabilities of SNFuzz uncovered a critical and state-dependent vulnerability in the EMQX MQTT-SN gateway. Our method

achieves this by injecting a valid CONNECT packet into an already-established session with CONNECTED state. Since MQTT-SN operates over connectionless UDP without an authentication method, gateway implementations associate session states primarily with plaintext client IDs. Consequently, an unauthenticated attacker can easily spoof a legitimate device and inject this anomalous state transition using a learned or guessed client ID [23].

Root Cause and Architectural Insight. The underlying flaw is not a traditional memory corruption or encoding error, but a semantic gap in the gateway's state management architecture. When the expected CONNECT packet arrives, the protocol can correctly identify the illegal transition and emits a structured error. However, the gateway's underlying connection dispatch layer has no logic to handle this error gracefully. This exception can cause the process to crash, destroying the established communication channel and all correlated states. This indicates an architectural vulnerability in protocol brokers: insecure error-handling boundaries between the isolated protocol state machine and underlying session management can lead to minor state anomalies, leading to fatal process failures.

Impact on Sensors and Vehicular Networks. For resource-constrained sensors and connected vehicles relying on MQTT-SN, the real-world implications of this vulnerability are critical. It gives adversaries an exceptionally cheap and asymmetric DoS primitive; a trivially spoofed UDP datagram is sufficient to sever an active connection. Since an attacker can continuously transmit these state-confusing datagrams, they can disrupt telemetry pipelines, effectively blinding the gateway and neutralizing edge devices in safety-critical environments.

6 Challenges

Despite its success in expanding deep-state exploration, SNFuzz still has several outstanding challenges that provide us with future opportunities to improve its performance and refine the tool.

Executions vs Coverage Optimization One of the biggest challenges identified during the evaluation of SNFuzz was a significantly lower execution throughput, averaging approximately 22 executions per second compared to 150 for AFLNet. While the fuzzer state transition weights could be adjusted to favor faster sequences and increase raw speed, such changes would likely reduce coverage. Therefore, a critical future objective is to determine optimal weight values to maximize execution speed without sacrificing code penetration.

Expanding Deep-State Exploration. Another opportunity for future iterations is to expand the SNFuzz state model to target even deeper MQTT-SN states. This includes: (1) *Sleep-State Management.* Assessing the gateway's logic in buffering and later delivering messages for sensors that are in sleep mode. (2) *Gateway Translation Tables.* Extending the exploration of how the gateway handles translations between MQTT-SN topic identifiers and MQTT topic strings.

Physical Hardware Validation. All testing performed in this study was conducted using native software on commodity hardware, specifically a Lenovo ThinkPad running Ubuntu 24.04.3. While this provided a controlled environment for assessing fuzzer performance, it does not account for all environmental factors present in real-world deployments. Consequently, there is a possibility that bugs found in a native or non-virtualized OS environment may not be reproducible on industry-standard hardware, such as the constrained edge devices used in SDVs or Intelligent Transportation Systems (ITSs). Future validation of these results should be conducted on a variety of physical sensors and vehicle edge devices to ensure the findings map accurately to field deployments.

Future Works. The automotive world is experiencing a revolution to SDVs, where modern vehicles are now IoT networks on wheels. However, our research reveals a critical weakness in the current state of SDVs. We show that the MQTT-SN gateways used in these vehicles' sensor communications are alarmingly prone to network attacks. As vehicles become increasingly sensor-dependent, communication protocols are now a critical component of vehicle operation and safety. The implications of our research are far-reaching and go beyond software flaws. We have shown the feasibility of a simple fuzzer to crash the infrastructure of vehicle telemetry and safety monitoring. In a real-world ITS scenario, such vulnerabilities could allow an attacker to disable critical connections with a single, spoofed packet. Looking to the future, we urge a shift in focus. The stability of our future transport infrastructure depends on our ability to secure communication protocols today. Otherwise, theoretical exploits in labs will become real attacks on the road.

AI Policy

AI tools were used solely for language polishing and grammar refinement. All technical content, experiments, and conclusions were conceived, executed, and validated by the authors.

Ethical Considerations

In this paper, we analyze publicly available MQTT-SN gateway implementations. The research does not involve humans, animals, private user data, environmental systems or healthcare and military applications. MQTT-SN gateways serve as critical communication gateways in IoT deployments. A successful attack exploiting gateway vulnerabilities could sever active connections and disrupt safety-critical communication channels. We therefore consider it essential to proactively

identify and report such vulnerabilities so that vendors can issue fixes before adversaries discover and exploit them.

Ethical Principles. We recognize that our work for vulnerability search in MQTT-SN gateway can potentially involve ethical concerns. We have carefully considered the ethical implications and potential risks throughout the process. Our research adhered to the four principles of the Menlo Report [4], which includes Beneficence, Respect for Persons, Justice and Respect for Law and Public Interest. Beneficence is the core ethical principle guiding our decisions since the key ethical question is whether the benefits of disclosure and publication outweigh the risks. To minimize potential harm, we follow coordinated disclosure norms reporting vulnerabilities to the responsible vendor before public release.

Stakeholders. We identify the following stakeholders impacted by our research and its publication: MQTT-SN gateway vendors (e.g., EMQX, Eclipse Paho, HiveMQ), developers and maintainers of gateway software, operators of IoT and vehicular communication infrastructure, end users who depend on reliable sensor communication, and potential adversaries. During the research process, we respected individuals and avoided any interference with real end users or production deployments. All experiments were conducted on locally deployed test instances and we did not test against affected targets in public environments. We did not collect or disclose any private data from vendors, operators, or users. We complied with each vendor's published security policy and followed coordinated disclosure norms, contacting the EMQX security team through their official channel before any public release and committing to withhold detailed exploitation artifacts until remediation. We did not target any specific vendor unfairly; gateway implementations were selected based on adoption and availability rather than to single out particular manufacturers. Our work aims to proactively identify vulnerabilities in MQTT-SN gateways to assist vendors in remediation and to strengthen the overall security of the IoT communication ecosystem.

Potential Harms and Impacts. The primary risk arising from publishing this research is that adversaries could learn of weaknesses in MQTT-SN gateway implementations and attempt attacks against unpatched deployments potentially causing operational disruption and degraded service availability in IoT and vehicular settings that rely on MQTT-SN telemetry. The research itself imposes no harm as all experiments were performed on locally deployed test instances with no interaction with production systems on real-world devices. This research involves no human subjects, no private user data and no informed-consent or privacy considerations.

Mitigation. We reduce the likelihood of harm by avoiding operational exploit details and by not releasing exploit code or detailed reproduction artifacts prior to remediation. We disclosed the discovered EMQX vulnerability to the EMQX security team. The disclosure was made via EMQX's official disclosure channel per their published security policy [1]. As of May 08, 2026, the EMQX security team has acknowledged the vulnerability and confirmed the crash is reproducible. Discussions regarding severity assessment and remediation are ongoing with the concerned party. We will not publicly release detailed exploit information until EMQX has had a reasonable opportunity to issue a fix. We will coordinate the public release timing of any artifacts with the vendor's patch schedule.

Decision Process to Publish. The purpose of this research is to identify vulnerabilities in MQTT-SN gateway implementations and support timely remediation by vendors. Before conducting this study, we carefully evaluated the potential ethical risks and expected benefits of disclosure and publication. We proceeded with the research because we believe the work provides practical defensive value to vendors, operators, and the broader research community, while avoiding technical details that could introduce operational risk. Based on this assessment, we concluded that the benefits of responsible disclosure and publication outweigh the remaining risks. We further confirm that our disclosure decisions were made with careful consideration of all relevant stakeholders, the potential impacts of publication, and the assumptions underlying our findings.

References

[1] emqx/emqx, May 2026. original-date: 2012-12-17T10:07:44Z. URL: https://github.com/emqx/emqx.

[2] Bernhard K. Aichernig, Edi Muškardin, and Andrea Pferscher. Learning-Based Fuzzing of IoT Message Brokers. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 47–58, April 2021. ISSN: 2159-4848. URL: https://ieeexplore.ieee.org/abstract/document/9438590, doi:10.1109/ICST49551.2021.00017.

[3] Wael Alsabbagh, Bahij Sayegh, Chaerin Kim, Samuel Amogbonjaye, Nitin Patil, M. Omar Al-Kadri, and Peter Langendoerfer. *MQTT Protocol in Industrial Internet of Things: Today Challenges and Tomorrow Solutions*. October 2025. doi:10.13140/RG.2.2.18668.42885.

[4] Michael Bailey, David Dittrich, Erin Kenneally, and Doug Maughan. The Menlo Report. *IEEE Security & Privacy*, 10(2):71–75, March 2012. URL: https://ieeexplore.ieee.org/document/6173001/, doi:10.1109/MSP.2012.52.

[5] Yuanliang Chen, Yu Jiang, Fuchen Ma, Jie Liang, Mingzhe Wang, Chijin Zhou, Xun Jiao, and Zhuo Su. EnFuzz: Ensemble Fuzzing with Seed Synchronization among Diverse Fuzzers. pages 1967–1983, 2019. URL: https://www.usenix.org/

conference/usenixsecurity19/presentation/chen-yuanliang.

[6] Samin Y. Chowdhury, Ruimin Sun, and Brandon Dudley. A Survey for MQTT Fuzzing. In *Proceedings of the 2025 Workshop on Re-design Industrial Control Systems with Security*, RICSS '25, pages 16–25, New York, NY, USA, October 2025. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/3733823.3764515, doi:10.1145/3733823.3764515.

[7] Gaetano Coppoletta. OCPPStorm: A Comprehensive Fuzzing Tool for OCPP Implementations. Master's thesis, University of Illinois at Chicago, United States – Illinois, 2024. URL: https://www.proquest.com/docview/3075474023/abstract/87FDE40895E245F7PQ/1.

[8] Mahmoud Elkhodr, Seyed Shahrestani, and Hon Cheung. Emerging Wireless Technologies in the Internet of Things : A Comparative Study. *International Journal of Wireless & Mobile Networks*, 8, November 2016. doi:10.5121/ijwmn.2016.8505.

[9] Shuitao Gan, Chao Zhang, Xiaojun Qin, Xuwen Tu, Kang Li, Zhongyu Pei, and Zuoning Chen. CollAFL: Path Sensitive Fuzzing. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 679–696, May 2018. ISSN: 2375-1207. URL: https://ieeexplore.ieee.org/document/8418631, doi:10.1109/SP.2018.00040.

[10] Xiang Gong, Ting Kou, and Yan Li. Enhancing MQTT-SN Security with a Lightweight PUF-Based Authentication and Encrypted Channel Establishment Scheme. *Symmetry*, 16(10):1282, October 2024. URL: https://www.mdpi.com/2073-8994/16/10/1282, doi:10.3390/sym16101282.

[11] Hemant Gupta and Amiya Nayak. Addressing IoT Security Challenges: A Framework for Determining Security Requirements of Smart Locks Leveraging MQTT-SN. In *2023 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–7, October 2023. ISSN: 2768-0940. URL: https://ieeexplore.ieee.org/abstract/document/10323864, doi:10.1109/ISNCC58260.2023.10323864.

[12] Hemant Gupta and Amiya Nayak. Publish Subscribe System Security Requirement: A Case Study for V2V Communication. *IEEE Open Journal of the Computer Society*, 5:389–405, 2024. URL: https://ieeexplore.ieee.org/document/10636299/, doi:10.1109/OJCS.2024.3442921.

[13] Hemant Gupta and Amiya Nayak. Security Vulnerabilities and Countermeasures in MQTT-SN Communication. In Mayank Singh, Vipin Tyagi, P. K. Gupta, Jan Flusser, Tuncer Ören, Vladimir Tomberg, and Alex Norta, editors, *Advances in Computing and Data Sciences*, pages 117–131, Cham, 2026. Springer Nature Switzerland. doi:10.1007/978-3-032-13757-9_9.

[14] Nicholas Humfrey. njh/mqtt-sn-tools, December 2025. original-date: 2013-12-02T15:13:48Z. URL: https://github.com/njh/mqtt-sn-tools.

[15] T L Kao, H C Wang, and J E Li. Safe MQTT-SN: a lightweight secure encrypted communication in IoT. *Journal of Physics: Conference Series*, 2020(1):012044, September 2021. doi:10.1088/1742-6596/2020/1/012044.

[16] Wei Lin, Sini Chen, and Huibiao Zhu. Formalization and Verification of MQTT-SN Communication Using CSP. In Jan Kofroň, Tiziana Margaria, and Cristina Seceleanu, editors, *Engineering of Computer-Based Systems*, pages 115–132, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-49252-5_10.

[17] Xinpeng Liu, Qinying Wang, Peiyu Liu, Wenhai Wang, and Shouling Ji. MQueez: Specification-Driven Fuzzing for MQTT Broker (Registered Report). In *Proceedings of the 34th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA Companion '25, pages 133–142, New York, NY, USA, June 2025. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/3713081.3731724, doi:10.1145/3713081.3731724.

[18] Alejandro Mera, Changming Liu, Ruimin Sun, Engin Kirda, and Long Lu. SHiFT: Semi-hosted Fuzz Testing for Embedded Applications. pages 5323–5340, 2024. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/mera.

[19] Chang-Seop Park and Hye-Min Nam. Security Architecture and Protocols for Secure MQTT-SN. *IEEE Access*, 8:226422–226436, 2020. URL: https://ieeexplore.ieee.org/document/9296847/, doi:10.1109/ACCESS.2020.3045441.

[20] Bryan Pearson, Yue Zhang, Cliff Zou, and Xinwen Fu. FUME: Fuzzing Message Queuing Telemetry Transport Brokers. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pages 1699–1708, May 2022. ISSN: 2641-9874. URL: https://ieeexplore.ieee.org/abstract/document/9796755, doi:10.1109/INFOCOM48880.2022.9796755.

[21] Van-Thuan Pham, Marcel Böhme, and Abhik Roychoudhury. AFLNET: A Greybox Fuzzer for

Network Protocols. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 460–465, October 2020. ISSN: 2159-4848. URL: https://ieeexplore.ieee.org/abstract/document/9159093, doi:10.1109/ICST46399.2020.00062.

[22] José Roldán-Gómez, Javier Carrillo-Mondéjar, Juan Manuel Castelo Gómez, and José Luis Martínez Martínez. Security Assessment of the MQTT-SN Protocol for the Internet of Things. *Journal of Physics: Conference Series*, 2224(1):012079, April 2022. URL: https://iopscience.iop.org/article/10.1088/1742-6596/2224/1/012079, doi:10.1088/1742-6596/2224/1/012079.

[23] José Roldán-Gómez, Javier Carrillo-Mondéjar, Juan Manuel Castelo Gómez, and Sergio Ruiz-Villafranca. Security Analysis of the MQTT-SN Protocol for the Internet of Things. *Applied Sciences*, 12(21):10991, January 2022. URL: https://www.mdpi.com/2076-3417/12/21/10991, doi:10.3390/app122110991.

[24] Ousmane Sadio, Ibrahima Ngom, and Claude Lishou. Lightweight Security Scheme for MQTT/MQTT-SN Protocol. In *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pages 119–123, October 2019. URL: https://ieeexplore.ieee.org/abstract/document/8939177, doi:10.1109/IOTSMS48152.2019.8939177.

[25] Meena Singh, M.A. Rajan, V.L. Shivraj, and P. Balamuralidhar. Secure MQTT for Internet of Things (IoT). In *2015 Fifth International Conference on Communication Systems and Network Technologies*, pages 746–751, April 2015. URL: https://ieeexplore.ieee.org/abstract/document/7280018, doi:10.1109/CSNT.2015.16.

[26] Hannes Sochor, Flavio Ferrarotti, and Rudolf Ramler. Exploiting MQTT-SN for Distributed Reflection Denial-of-Service Attacks. In Gabriele Kotsis, A. Min Tjoa, Ismail Khalil, Lukas Fischer, Bernhard Moser, Atif Mashkoor, Johannes Sametinger, Anna Fensel, and Jorge Martinez-Gil, editors, *Database and Expert Systems Applications*, pages 74–81, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-59028-4_7.

[27] Xiangpu Song, Jianliang Wu, Yingpei Zeng, Hao Pan, Chaoshun Zuo, Qingchuan Zhao, and Shanqing Guo. MBFuzzer: A Multi-Party Protocol Fuzzer for MQTT Brokers. pages 6179–6197, 2025. URL: https://www.usenix.org/conference/usenixsecurity25/presentation/song-xiangpu.

[28] Sören Tempel, Vladimir Herdt, and Rolf Drechsler. Specification-Based Symbolic Execution for Stateful Network Protocol Implementations in IoT. *IEEE Internet of Things Journal*, 10(11):9544–9555, June 2023. URL: https://ieeexplore.ieee.org/abstract/document/10016470, doi:10.1109/JIOT.2023.3236694.

[29] Zhiqiang Wei, Xijia Wei, Xinghua Zhao, Zongtang Hu, and Chu Xu. SGANFuzz: A Deep Learning-Based MQTT Fuzzing Method Using Generative Adversarial Networks. *IEEE Access*, 12:27210–27224, 2024. URL: https://ieeexplore.ieee.org/abstract/document/10433531, doi:10.1109/ACCESS.2024.3365712.

[30] Yingpei Zeng, Mingmin Lin, Shanqing Guo, Yanzhao Shen, Tingting Cui, Ting Wu, Qiuhua Zheng, and Qiuhua Wang. MultiFuzz: A Coverage-Based Multiparty-Protocol Fuzzer for IoT Publish/Subscribe Protocols. *Sensors*, 20(18):5194, January 2020. URL: https://www.mdpi.com/1424-8220/20/18/5194, doi:10.3390/s20185194.

Digital Eyes on the Road: Using Street Cameras to Verify Traffic Integrity and Detect Sybil Attacks at City Scale

Jhonatan Tavori, Mehmet Kerem Turkcan, Zoran Kostic, Salvatore Stolfo and Gil Zussman
Columbia University
New York, NY, USA
{j.tavori,mkt2126,zk2172}@columbia.edu, sal@cs.columbia.edu, gil.zussman@columbia.edu

Abstract

Crowdsourced navigation platforms infer congestion from GPS-derived telemetry, enabling real-time traffic estimation and control at city scale. However, because these systems rely on device-reported location traces, they are vulnerable to *Sybil attacks*, where an adversary injects fabricated "vehicles" and synthetic slowdowns using emulators, scripted clients, or device farms. Such attacks have been publicly demonstrated to create "ghost" cars and phantom congestion that distort the displayed traffic layer and influence routing decisions.

We propose using street cameras as a real-time validation layer for GPS-based traffic inference. Street cameras add an independent signal grounded in physical observation ("eyes on the road"), rather than device-reported telemetry alone. They also sit on a different attack surface than mobile devices: manipulating camera evidence targets physical infrastructure rather than the mobile client, and is often more expensive to scale than client-side Sybil injection. By cross-checking camera-observed vehicle activity against GPS-based mobile traces, camera feeds provide a conservative verification layer, and can be used to raise integrity alerts.

In this paper, we implement an end-to-end GPS-camera pipeline that (i) matches cameras to nearby road links, (ii) aligns and stabilizes the streams, (iii) scores mismatch using a residual-based detector to produce alerts, and (iv) extends coverage to blind spots using nearby intersections.

We evaluate the system using New York City DOT street cameras (200+ active in Manhattan) together with GPS-based traffic levels collected via the TomTom API. The pipeline achieves 91.5% detection under a moderate injected attack ($\delta = +0.05$) and reaches $\approx$99% detection for stronger attacks. We further address blind spots: using only neighbors, we retain a high-load consensus signal in 96.68% of Manhattan tested locations.

1 Introduction

Mobile navigation services estimate traffic from real-time telemetry reported by end-user devices (e.g., mobile phones) [15, 23]. Platforms aggregate location and speed samples, map-match them to road segments, and combine historical baselines to produce congestion overlays, ETAs, and routing recommendations. As these outputs shape navigation decisions for large numbers of users, and underpin driver-assistance systems and autonomous vehicles, the integrity of their inferred traffic state is a critical security and safety concern [4, 12].



(a) Actual state of the road.

(b) Reflected state (jammed) due to the attack.

Figure 1: Illustration of a Sybil-driven congestion attack. By injecting manipulated probe reports, the adversary forces the platform to account for congestion that is decoupled from the physical traffic conditions on the roadway.

Navigation services infer traffic from location and speed reports submitted by user devices, which creates an Sybil attack surface: an adversary can manipulate the traffic layer by controlling what devices report. The attacker can either (i) compromise devices (e.g., via malware or instrumentation) or (ii) avoid compromising anyone by generating many reporting identities using emulators, scripted clients, or device farms that replay or synthesize GPS traces. Prior work has demonstrated both stealthy GPS manipulation of a victim's reports [39], as well as Sybil attacks that inject coordinated 'ghost' vehicles to bias aggregate congestion inference on platforms such as *Waze* and *Google Maps* [3, 29, 33].

Figure 1 illustrates this threat: an attacker can inject 'ghost cars': reported vehicles that do not exist physically. The

'ghost cars' will make the platform infer congestion that is not present on the road. If the service believes a segment is jammed, it may reroute drivers away from it, shifting demand and reshaping traffic patterns beyond the targeted street. At city scale, this manipulation can impose large cumulative costs in wasted travel time, fuel, and operational disruption, and can undermine trust in routing infrastructure. These concerns become sharper as smart cities and autonomous vehicles increasingly rely on data-driven traffic layers for control, routing,and safety-critical planning.

Existing defenses largely focus on identifying or preventing *fake* traffic reports. In the Sybil setting, this typically means (i) hardening the reporting endpoint so an attacker cannot easily control what a device submits (e.g., platform/app attestation and anti-tampering), (ii) strengthening user or device authentication to raise the cost of creating many identities using emulators or device farms, and (iii) detecting Sybil behavior from structure in the reported telemetry, such as co-location graphs, trajectory consistency, or behavioral anomaly detection [12]. These mechanisms can raise attacker cost, but they still lack an external notion of truth: attestation does not prove physical location; authentication can still be bypassed at scale; and behavior-based filters can be evaded or confused by legitimate disruptions.

As report-based defenses can be bypassed, integrity checks should rely on evidence *grounded in the physical world*, not only on the same stream of device reports the attacker can shape. This motivates adding an independent sensing channel to corroborate the inferred traffic state.

In this paper, we demonstrate how to fuse street-camera sensing with GPS-derived traffic flow to improve the integrity of traffic state estimation. Figure 2 summarizes the end-to-end workflow: camera and GPS inputs are associated, aggregated and smoothed to reduce measurement noise, and then compared via residual-based scoring to produce an integrity signal (i.e., attack alerts).

We implement an end-to-end GPS-camera verification pipeline. The inputs are different modalities: camera-based vehicle counts and GPS-based traffic flow. We spatially match cameras to nearby road links and temporally align the asynchronous streams, which allows us to learn typical co-variation patterns. When the GPS-based congestion signal is artificially increased by an injected attack, the resulting cross-source mismatch grows and is detected by our residual-based monitor.

A key challenge is that the two sources measure different things: cameras yield vehicle-count time series, while the GPS feed reports a congestion-intensity value (analogous to the green/yellow/red traffic layer in navigation apps). Despite this mismatch, we associate each camera with nearby road links and align the asynchronous streams in time, allowing us to learn what "normal" agreement looks like at each location (using a residual-based monitor). When an injected attack

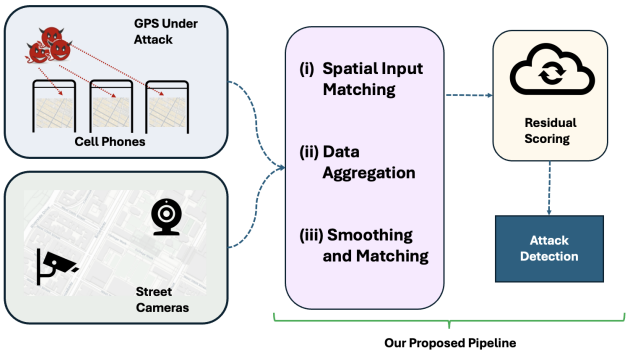


Figure 2: GPS and camera fusion workflow. We match each camera to nearby road links, align timestamps and signals if needed, and score mismatches to raise alerts.

inflates the GPS-reported congestion without a corresponding change in observed vehicle activity, the disagreement grows and is surfaced by our residual-based monitor.

To demonstrate feasibility at city scale, we use measurements from New York City and focus on the Manhattan borough, where dense camera coverage and complex road topology create a challenging urban setting. We use NYC Department of Transportation (DOT) traffic cameras [25], with over 200 active cameras in Manhattan providing vehicle-count time series at 15 s granularity. We pair these counts with GPS-derived traffic measurements from *TomTom* via the TomTom Traffic Flow API [31], which reports timestamped, link-level traffic indicators (`traffic_level`) over road geometries.

In the Manhattan setting, our pipeline achieves 91.5% detection under a moderate injected manipulation ($\delta = +0.05$) and reaches ≈99% detection for stronger injections. We also evaluate robustness to camera blind spots: in Manhattan 'leave one camera out' tests, neighboring intersections preserve a high-load consensus signal in 96.68% of time bins, providing a conservative integrity mechanism when a center camera is unavailable.

Paper structure. Section 2 summarizes how navigation platforms infer traffic and reviews Sybil and spoofing attacks, including public demonstrations. It further surveys prior defenses for location integrity, contributor authentication, and multi-source verification. Section 3 formalizes the threat model, mitigation goals, research questions, and practical challenges for camera-based integrity verification. Section 4 presents our cross-modal verification approach: data sources, camera processing, spatial association of cameras and road links, gap-aware temporal alignment, and the alerting logic used to surface sustained disagreement. Section 5 describes our implementation and experimental analysis. In particular, Section 5.1 details the GPS-camera verification pipeline (spatial matching, temporal alignment, and mismatch scoring) and demonstrates injected-manipulation detection, and Sec-

tion 5.2 evaluates robustness under partial camera coverage via a neighbor-consensus analysis, including case studies at two Manhattan intersections. Section 6 then evaluates both GPS-camera mismatch detection and blind-spot consensus at Manhattan scale, across over 250 cameras. Section 7 discusses limitations, deployment considerations, and privacy implications. Section 8 concludes with closing remarks and directions for future work.

2 Background and Related Work

In this section, we provide background on navigation platforms' reliance on GPS-based telemetry, Sybil attacks on traffic inference, and real-world demonstrations of these attacks. We then discuss related work on detecting and preventing Sybil manipulation in crowdsourced navigation systems.

2.1 Background: Navigation Platforms and Sybil Attacks

GPS telemetry in navigation platforms. GPS-derived telemetry is central to many smart-city applications and to emerging autonomous and connected-vehicle systems. Navigation platforms such as Google Maps and Waze estimate congestion by aggregating location and speed samples from large sets of devices [19]. They also combine recent observations with historical baselines and incidents. The result is a per-segment traffic state (often colored on map as green/yellow/red congestion) that impacts the estimation of ETAs and routing recommendations. Because this is driven by device-reported telemetry, an adversary who can inject, replay, or distort reports can bias the inferred traffic state and influence routing decisions.

Sybil injection and GPS manipulation. Navigation platforms can be manipulated either by injecting fabricated device data at the application layer or by distorting the location layer that produces those reports. In an application-layer Sybil attack, an adversary creates many identities (e.g., emulated devices, scripted clients, or coordinated "ghost" vehicles) that submit consistent location and speed traces so the service infers congestion on targeted segments [9]. Prior work has shown that such virtual attempts can cause 'phantom' congestion and trigger rerouting in systems such as Waze [36, 37], and similar manipulation opportunities have been analyzed for Google Maps' speed-based congestion inference [10]. GPS manipulation (including GNSS spoofing) targets the location layer directly by causing devices to report false positions and speeds. This then propagate through map-matching and can aggregate into the traffic layer [17, 30]. An attacker can pair Sybil-generated traces with GPS spoofing to make fabricated trajectories more realistic and harder to filter [39].

Real-world demonstrations of sybil attacks. These threats have been demonstrated outside controlled simulations. In March 2014, a student demonstration induced Waze to display a nonexistent traffic jam and to suggest alternate routes by creating many fake Android devices via emulation, generating coordinated slow trajectories, and submitting reports consistent with being stuck in traffic [29, 33]. The demonstration was conducted on campus roads to limit external impact, but it showed that large-scale aggregation of device reports can be influenced by an adversary who can scale credible identities. A subsequent academic study (MobiSys 2016) provided explicit before/after evidence: Waze initially recommended the target segment as the fastest route, then rerouted users once the injected congestion increased the segment's inferred travel time [36]. This linkage, from manipulated reports to routing decisions, is the central risk: false congestion can divert traffic, waste capacity, and create secondary congestion elsewhere.

In February 2020, the "99 phones" experiment demonstrated a different but operationally similar mechanism. Rather than emulation, an artist moved dozens of physical smartphones at walking speed, producing telemetry consistent with heavy slow traffic. After roughly an hour, Google Maps rendered an extended red congestion segment on an otherwise lightly trafficked road [3, 14]. Google publicly noted that traffic visualization is refreshed from aggregated anonymous data and did not (at that time) explicitly filter out that configuration [3]. Together, these demonstrations show that both virtual Sybil fleets and physical device multiplicity can bias GPS-based traffic inference, motivating defenses that corroborate inferred traffic state using independent, infrastructure-grounded sensing.

2.2 Related Work: Sybil Detection

Prior work has explored a range of defenses for improving the integrity of GPS-based and crowd-sourced traffic inference. These include mechanisms that make it harder to generate or sustain fake traffic reports (e.g., endpoint hardening and authentication), methods that detect Sybil behavior from reported patterns, and techniques that identify manipulation at the GPS/GNSS layer for such spoofing attacks. A recent survey of vehicular traffic security similarly organizes defenses into trust schemes, learning-based misbehavior detection, and authentication mechanisms, highlighting that most protection efforts focus on identifying or excluding malicious senders within the reporting channel [12]. These approaches make different assumptions and target different parts of the pipeline. We summarize the most relevant directions and use them to motivate our focus on street cameras as an additional, infrastructure-grounded source for validation.

A common direction is to filter *fake* traffic reports by raising the cost of producing many credible identities. Thus, server-side account controls, or platform operator, can reduce low-

effort attacks by requiring attackers to use real devices or more sophisticated bypasses [36, 37]. However, without trusted certification, an adversary can still present many identities and confuse the system [9], as attestation alone does not prove where a device actually is.

A complementary class of defenses tries to identify 'ghost' users by investigating the reported data. SybilRank [6]) apply graph-based techniques to separate Sybil clusters when they have limited connections to honest users. Waze-focused work proposed constructing such graphs from short-range encounters and using them for detection [36, 37]; [1] note these assumptions can break under real-device usage of the attacker and thus are not sufficient.

Other work detects manipulation from reported movement patterns. Trajectory- and behavior-based models look for what appears as implausible motion, coordination signatures, or inconsistent contribution patterns, including learning-based approaches applied to crowdsourced navigation [38]. These methods can catch naive attacks, but may be evaded by adaptive attackers. Further, legitimate disruptions (e.g., parades, storms, or closures) that naturally deviate from historical baselines, may also raise false alarms.

Regarding GNSS spoofing, which were mentioned as one of the attacking schemes, the literature mostly focuses on receiver- and signal-layer detection and cross-checking (e.g., anomaly detection at the GNSS signal level or redundancy across receivers) [18]. These techniques target location-layer manipulation, but they do not address application-layer Sybil fleets that fabricate traffic reports without relying on GNSS reception.

Limits of report-based defenses. Real-device farms can scale despite attestation, and rate limits can be averaged across many accounts and over time. Behavior-based detectors can be evaded by tuning traces to match expected distributions (e.g., plausible speeds, dispersion, and timing). A particularly challenging case is gradual manipulation that stays within the range of normal variability caused by rush hour dynamics, weather, or events, making it difficult to separate attacks from legitimate disruptions [1, 27].

Thus, in practice, a determined adversary can often bypass defenses that operate only on device-reported traffic data (perhaps at higher cost), since the attacker can shape the very signals those defenses rely on for detection.

These limitations motivate a complementary requirement: attacks alarms should be triggered by evidence that is independent of mobile device reports. When report-based controls are bypassed, the system needs a mechanism that is grounded in the physical world and harder to manipulate remotely at scale. Traffic-poisoning studies emphasize that effective countermeasures depend on an explicit attacker model and thresholds calibrated to realistic capabilities [27].

Multi-source verification and our contribution. To overcome the limitations of defenses that rely only on device-reported traffic data, we suggest cross-checking traffic estimates using independent sources so that manipulating one stream does not determine the inferred traffic state. In practice, this can mean suggest several implementations, including comparing GPS-based congestion with infrastructure telemetry or incident feeds. The value of such fusion depends on true independence: if multiple inputs ultimately derive from the same end-devices or report ecosystem, the manipulation can still propagate and bypass the comparison [18, 27].

In this work, we use street cameras as an infrastructure-grounded verification channel and compare camera-observed vehicle activity against GPS-derived traffic flow to flag sustained mismatches consistent with manipulation. In the following section, we describe the camera-based verification layer and the city-scale GPS-camera verification pipeline in detail.

Prior work on fusing GPS-derived probe signals with physical sensors has primarily focused on improving traffic estimation and operations [20]. In contrast, we use camera–GPS comparison in real time as an integrity mechanism: we explicitly monitor for disagreement to detect manipulation. We further demonstrate and evaluate this use at city scale across Manhattan.

3 Threat Model and System Goals

In this section, we describe the threat model considered by our pipeline, and state the operational goals of the proposed integrity-verification architecture. We then describe the main research questions that guide our design and evaluation.

Threat model. We consider an adversary whose objective is to artificially inflate *reported* congestion on targeted road segments. The attacker injects fake GPS telemetry at scale, for example, via emulator fleets, device farms, or scripted clients. By doing so, it poisons the platform's traffic inference and creates phantom congestion that is not present on the road.

The systemic risk extends beyond individual misroutes. Because algorithmic navigation influences large populations of drivers, and is increasingly incorporated into emerging autonomous-driving systems, poisoning the inferred traffic state can create cascading effects. Manipulation may cause visible disruption (e.g., widespread rerouting and localized jams), but it can also induce subtle, hard-to-detect inefficiencies; at city scale and over time, even small biases can accumulate into substantial societal costs in time and fuel.

This Sybil threat can be executed remotely and at relatively low cost, without physically affecting the roadway. Consequently, we focus on infrastructure-grounded "ground-truth" validation that remains effective even when application-layer GPS defenses are bypassed.

Mitigation goal and scope. Our goal is to build a city-scale integrity pipeline that can detect Sybil-style manipulation of mobile GPS reports by flagging disagreement between GPS-reported traffic conditions and camera-observed vehicle activity. Thus, we do not aim to replace primary Sybil defenses (e.g., attestation, identity friction, or behavior-based filtering). Instead, we provide an independent, infrastructure-grounded verification signal that supports conservative monitoring decisions when persistent cross-source inconsistencies arise.

Research questions. Our implementation is organized around three questions that together test whether the proposed verification pipeline is practical and effective at city scale.

- **(RQ1)** Can camera-derived *vehicle-count* signals, obtained from low-rate/resolution camera interfaces cities typically expose (i.e., periodic frames, not continuous video), be spatially matched and aligned with GPS-based traces well enough to enable cross-source validation?
- **(RQ2)** Can residual-based GPS-camera mismatch scoring detect targeted Sybil-style false congestion at city scale?
- **(RQ3)** If mismatch can be detected where cameras exist, can we extend verification to camera blind spots by using consensus from neighboring intersections when a center camera is missing or not installed?

4 Our Approach: Real-Time Pipeline for Traffic Verification with Street Cameras

In this section, we describe the inputs to our real-time verification pipeline and the practical challenges it must handle. Our design is motivated by the threat model in Section 3.

As reviewed in Section 2, most prior defenses operate within the reporting ecosystem (e.g., authentication or anomaly detection at the report-level). These mechanisms can raise the attacker's cost, but they still lack *physical* evidence of what is happening on the road. Since an adversary can manipulate GPS-derived reports at scale, integrity checks benefit from a validation signal that is not produced by the same reporting channel.

We therefore use street cameras as an infrastructure-based verification layer in a city-compatible setting: periodic camera frames paired with a commercial GPS-based traffic feed. Our pipeline cross-checks the two sources and raises an alert when disagreement persists beyond normal variability, producing a practical integrity signal for monitoring.

Pipeline Sources: Cameras and GPS Our goal is to build a verification pipeline that plugs into existing municipal sensing infrastructure. Because bandwidth constraints and privacy-oriented policies often limit what cities can expose, we restrict ourselves to the minimal public interface [24, 25]: periodic traffic-camera frames, rather than assuming continuous, high-rate streaming access. This design choice also defines a core technical challenge: the camera input yields vehicle *counts*, not direct speed estimates (as full video stream tracking might provide), so verification must relate count-based activity to GPS-derived congestion intensity.

The first input to our pipeline is, for each camera location, a timestamped vehicle-count time series paired with the camera's coordinates. To obtain these counts, we rely on the collection and processing pipeline of [11, 34], built over NYC Department of Transportation (DOT) public traffic cameras [25], and the COSMOS NSF testbed [11, 28]. The system of [34] ingests the DOT feeds and performs vehicle counting using YOLO-LR, an adaptation of YOLO optimized for low-resolution footage. This design is important for generality: NYC DOT streams are 352×240 pixels, well below standard object-detection dataset resolutions. Upstream processing in [34] was performed on a single compute node equipped with dual NVIDIA A100 40 GB GPUs, producing per-camera count series at 15 s granularity (with class-level counts) that we use as the cameras' input.

The second input is a GPS-derived traffic feed from the TomTom Developer Traffic Flow API [31], which provides real-time, link-level measurements aligned to road geometries. For each road segment, TomTom reports a timestamped traffic level value in the interval $[0, 1]$ that indicates the *relative* traffic level: it is calculated as the current speed on the segment expressed as a fraction of its free-flow speed, highlighting areas of congestion when the fraction drops. TomTom further classifies traffic conditions based on how much slower vehicles are moving compared to free-flow conditions [32]. We ingest and store the API stream per segment as provided, while preserving the original timestamps.

These two inputs are heterogeneous: cameras provide vehicle counts from fixed viewpoints, while TomTom provides a link-level congestion indicator derived from relative speed. Bridging this gap lies in the core design problem. In the remainder of this section, we describe (i) the sensing challenges of working with periodic frames and relating those counts to GPS-based congestion; and (ii) the spatial challenges of aligning cameras to the road map.

Sensing Challenge: Working From Frames and Matching Counts to GPS Because the camera interface is frame- or count-based, the main challenge is extracting a stable activity signal and making it comparable to the GPS congestion indicator through aggregation, smoothing, and careful missing-data handling.

Another sensing challenge is camera availability. DOT feeds are not uniformly continuous: outages and missing intervals vary across locations and days. In our pipeline, missing camera periods are treated as *no observation* (not as zero traffic), and we can exclude those timestamps from cross-source

scoring and, when needed, concatenate contiguous observation blocks rather than forcing a fully dense time series. This keeps the detector from being biased by gaps and allows the baseline relationship to be learned from the data that are actually observed. In addition, as we show in Section 5.2, neighboring intersections can often provide corroborating evidence when a given camera is unavailable, offering a conservative fallback under partial coverage.

At fine granularity, camera counts are noisy. We thus stabilize the camera signal before comparison by aggregating into coarser bins and applying rolling smoothing when needed.

Spatial Challenges: Camera Viewpoint and Area-Based Association A second challenge is spatial alignment. A street camera observes a particular viewing direction and a partial set of lanes, which often does not map cleanly to a single road segment, especially near intersections. Rather than assuming a one-to-one camera-road mapping, we treat each camera as representing a local *area* around its fixed coordinates and associate it with all nearby road links within a radius.

This area-based association is robust to viewpoint uncertainty and road geometry, and it provides a practical basis for comparing camera-observed vehicle activity with GPS-derived congestion on nearby links. Section 5 and Figure 3 detail the spatial association procedure.

In the next section, we show that these challenges can be handled at city scale: despite the mismatch between camera vehicle counts and GPS-derived congestion intensity, the fused signals support detection of injected Sybil-style manipulation. We also address camera blind spots by using neighboring intersections for consensus when direct camera observation is missing.

5 Implementation and Analysis

In this section we present the implementation and experimental analysis. In Section 5.1 we describe the GPS-camera verification pipeline and attack detection; in Section 5.2 we evaluate robustness to missing cameras via spatial neighbor consensus.

5.1 Cross-Modal (GPS-Camera) Fusion and Manipulation Detection

We next describe how we connect street cameras to GPS-derived traffic signals and use the resulting cross-check to detect an attack. Building on the challenges in Section 4, we suggest spatial association, stabilization, and residual-based mismatch scoring.

5.1.1 GPS-Camera Spatial Matching and Alignment

Our first step is to make the camera and GPS streams comparable in space and time. We associate each camera with nearby road links using TomTom road geometries: for each camera's latitude/longitude, we consider road polylines[1] in the traffic database and retain links whose geometry falls within the camera's local neighborhood.

As illustrated in Figure 3, we match a road link to camera i if a point along the link's polyline falls within a fixed radius of the camera's location. In our baseline configuration, we use a radius of 0.01 miles. This rule does not assume that the camera observes the entire link; Rather, it defines a conservative spatial neighborhood for comparison. This can be useful when the camera direction is unclear, just its location.

Given the matched link set for camera i, $\mathcal{L}(i)$, we compute a corresponding GPS signal by aggregating link-level values at each timestamp using the mean of the traffic level across the associated links:

$$\overline{\texttt{traffic_level}}_i(t) = \frac{1}{|\mathcal{L}(i)|} \sum_{\ell \in \mathcal{L}(i)} \texttt{traffic_level}_\ell(t).$$

In the Manhattan configuration, this procedure yields more than 50,000 unique associated links across 265 cameras.

After spatial aggregation, each camera i is paired with a corresponding GPS-derived signal summarizing traffic on its nearby road links. Yet the two sources are sampled very differently: camera counts arrive every 15 seconds, whereas TomTom actual values updates are asynchronous and may vary across links and time.

We therefore align the streams explicitly. We first transform the aggregated TomTom traffic level into a congestion-intensity signal,

$$g_i(t) = 1 - \overline{\texttt{traffic_level}}_i(t),$$

so that larger values correspond to heavier congestion. We treat missing intervals as *no observation* (not as zero congestion) and align each camera count series with its corresponding $g_i(t)$ using nearest-neighbor (as-of) matching under a bounded time tolerance. Finally, to account for noise and differing update rates, we evaluate a range of uniform temporal aggregation and smoothing windows before computing cross-source agreement and mismatch scores.

5.1.2 Mismatch Scoring and Manipulation Detection

After the spatial matching and temporal alignment, we have for camera i, a paired observations $(c_i(t), g_i(t))$, where $c_i(t)$ is the camera-derived vehicle-activity signal and $g_i(t)$ is the corresponding GPS-based congestion-intensity signal aggregated from nearby matched road links at time t.

[1] A *polyline* is a piecewise linear curve which is defined by an ordered sequence of coordinates. Road networks, including TomTom's, use polylines to encode roads and routes [16].

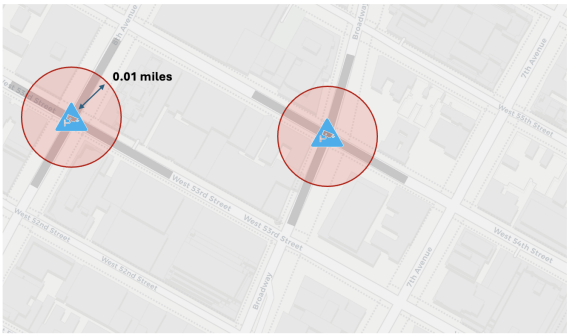


Figure 3: Camera-to-road (spatial) matching: camera (blue), spatial radius (red), associated road links (grey). A link is matched if its polyline point lies within 0.01 miles of the camera; the camera's GPS signal we treat is the aggregate `traffic_level` over matched links.

Because the two sources arrive at different cadences (camera counts at 15 seconds when available, TomTom may update asynchronously), we first aggregate the camera counts to time bins before comparison. We choose the smoothing window empirically: across Manhattan, longer smoothing windows produce more stable cross-source agreement, and we therefore use a long-minutes operating window in our evaluation.

For each camera, we fit a pre-attack baseline that captures the typical relationship between camera-observed vehicle counts and the corresponding GPS-derived traffic level. Let $z_i(t)$ denote the normalized residual at time t, i.e., the deviation between the observed GPS traffic level $g_i(t)$ and the counts observed by the cameras $c_i(t)$. Large positive $z_i(t)$ corresponds to where reported GPS congestion is higher than what the camera observations would support.

We then use a one-sided local (windowed) accumulation inspired by CUSUM/MOSUM control to detect mismatch between the two sources [13, 22, 26]. [2] We first filter the normalized residual signal $z_i(t)$ to retain only positive deviations above the noise floor (dictated by κ, the slack parameter):

$$u_i(t) = \max\big(0, z_i(t) - \kappa\big).$$

We then accumulate u_i over a rolling window (of length W):

$$S_i^{(W)}(t) = \sum_{\tau=t-W+1}^{t} u_i(\tau).$$

We use a finite window to limit long-memory effects. Because both inputs are noisy (even after binning and smoothing), the detector should emphasize recent persistent mismatch, and not to accumulate deviations that can cause delayed alarms based on history charts, nor to let short-lived visual glitches trigger alarms.

Because different cameras may experience different levels of natural variance, we normalize the accumulated result using a threshold h_i learned from pre-attack behavior, for each camera. The resulting alarm score is

$$A_i(t) = \frac{S_i^{(W)}(t)}{h_i}.$$

High $A_i(t)$ indicates persistent mismatch between the reported GPS congestion and camera-observed vehicle counts, that exceeds the camera's baseline. An alert is raised when $A_i(t)$ bypass a threshold (set to 1.2). Under our threat model, this pattern is consistent with Sybil-attack false congestion or related manipulation of GPS traffic reports, and we treat it as an integrity signal requiring inspection.

5.1.3 Evaluation Under Injected Sybil-Style Congestion

To model Sybil-driven false congestion, we inject an artificial increase into the GPS-derived congestion intensity after a fixed start time, while leaving the camera stream unchanged. This captures our target attack mode: the GPS-based feed is manipulated through fabricated reports, so it indicates elevated congestion even though camera-observed vehicle activity in the surrounding area does not change accordingly.

We use April 21–29 (2025) as the evaluation window, since this period provided consecutive camera coverage from the NYC DOT feeds. We activate the injected Sybil manipulation at April 26. To model the effect of false congestion on the GPS-derived signal, we add an offset δ to the traffic leveled provided by the GPS metrics (and bounded by the maximal value of 1):

$$g_i^{\text{attack}}(t) = \min\big(1,\, g_i(t) + \delta\big), \qquad t \geq t_0,$$

where t_0 is the attack start time and $\delta \in \{0.01, 0.025, 0.05, 0.10\}$ support several attack values, from moderate to severe. We illustrate the detector behavior on two representative Manhattan intersections, *8 Ave @ 14 St* and *1 Ave @ E Houston St*.

We compute two score trajectories under the same learned baseline and per-camera threshold: an *attacked* score (comparing the cameras to the manipulated GPS after the attack activation) and a *non-attacked* score (using the original GPS measures). Figure 4 reports results for two representative Manhattan intersections, *8 Ave @ 14 St* and *1 Ave @ E Houston St*. We use a decision threshold is of $(A_i(t) = 1.2)$. In both cases, the attack score (i.e., residual signals) increased after the attack activation, where larger attack magnitude leads to an earlier and stronger alarm triggering.

Overall, the results for both intersections are monotone in the attack magnitude, as post-activation attacked scores

[2] We use this detector because it accumulates only positive residual evidence (above a certain value), emphasizing the attack impact of upward shifts. The finite window avoids unbounded history effects and keeps pre/post behavior comparable [13, 22, 26].

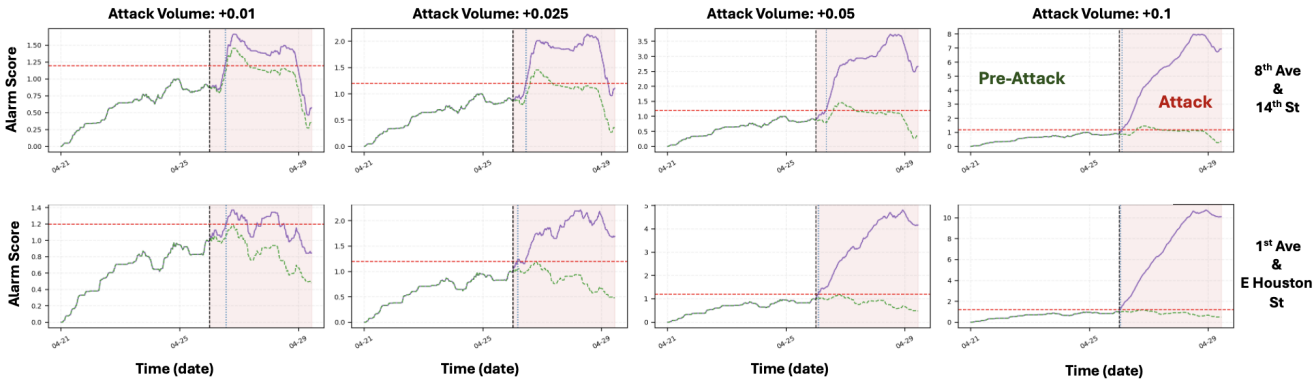


Figure 4: Pipeline detection results for injected Sybil attacks (Apr 21–29; attack activated Apr 26). Rows: *8 Ave @ 14 St* (top) and *1 Ave @ E Houston St* (bottom). Columns: attack volume $\delta = 0.01, 0.025, 0.05, 0.10$ (applied to GPS congestion and clipped at 1.0). Solid purple: attacked normalized score (relative residual signal), $A_i(t)$. Green dashed: non-attacked score under the same baseline and threshold. Red: alarm threshold (of 1.2). Larger values yield earlier and stronger alarm triggering.

increase when δ grows, while the non-attacked trajectories remain lower. This supports the pipeline's ability to distinguish manipulated regimes under injected Sybil GPS inflation.

5.2 Robustness to Missing Cameras via Spatial Neighbor Consensus

As discussed, road-camera coverage is incomplete and its availability can vary over time. This creates blind segments where GPS-reported traffic cannot be directly cross-checked against a physical camera. We consider an attacker who manages to manipulate GPS-reported congestion on such a 'blind segment'. Our solution is therefore a neighbor-consensus check: when the center camera is missing or unavailable, we use nearby intersections as corroborating evidence. If GPS indicates sustained congestion while surrounding cameras show no corresponding increase in vehicle activity, we flag the report as suspicious.

Neighbor consensus uses a common reliability principle in networked sensing: an isolated anomaly is treated as suspicious unless supported by nearby sensors within a spatiotemporal neighborhood. In wireless sensor networks, fault and outlier detection exploits local spatial correlation, using neighborhood agreement to separate real events from noise, faults, or adversarial injections [5, 40]. A similar concept appears in smart-grid security, where redundancy and cross-checking is carried to indicate a false-data injection [8, 21]. Transportation sensing applies similar logic for data-quality diagnostics and imputation: isolated anomalies are often treated as sensor faults unless corroborated by neighboring detectors [7, 35].

Our objective is slightly different: we use neighboring consensus as a mismatch integrity mechanism to validate GPS-reported congestion.

5.2.1 Neighbor Consensus Method Demonstration

We study two center locations: (i) *5 Ave @ 57 St* and (ii) *Amsterdam @ 72 St*. These cases illustrate two neighborhood structures. For *5 Ave @ 57 St*, we use nearby cameras along 57th Street and the adjacent avenues. For *Amsterdam @ 72 St*, we use a broader set of surrounding cameras spanning the local intersection neighborhood. Figure 5 depicts the intersections and selected neighbors.

Note that we choose center locations that *do* have a working camera, so we can evaluate whether a neighbor-based consensus would have provide the same load magnitude which we can use to detect Sybil attacks. If neighbor agreement is strong in this setting, it supports using neighbors as a mismatch mechanism when the center feed is missing.

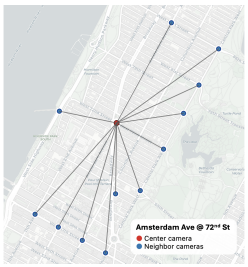
For each camera i, we smooth the vehicle-count stream and aggregate it into fixed-length bins. We then label a bin as high-load when its value is unusually high relative to that camera's baseline over the analysis window. Concretely, we apply this camera threshold scaled by a factor of $\alpha = 0.75$.

We then mark a center bin as *unique* if the center is considered high, but none of its neighboring cameras are considered high in the same bin or the immediately preceding bin (to allow for small timing offsets). Clearly, a low fraction of unique bins means that high-load events at the center are usually experienced by the nearby cameras, which supports using neighbors as an integrity signal when the center camera is unavailable.

Table 1 reports the number of center high traffic bins and the number of unique center bins. The observed unique rates are low. Thus, our pipeline will flag a blind-segment GPS report as suspicious if it indicates of high traffic level while neighboring cameras remain non-high around that timestamp.



(a) Case A: 5 Ave @ 57 St and surrounding cameras.



(b) Case B: Amsterdam @ 72 St and expanded surrounding set.

Figure 5: Two blind-spot case studies used for neighbor consensus analysis.

Intersection	# Nbrs	High Traffic	Unique	Rate
5 Ave @ 57 St	5	580	34	5.86%
Amsterdam @ 72 St	12	636	1	0.16%

Table 1: Neighbor consensus (5-min bins; 60-sample window; camera threshold scaled by a factor of $\alpha = 0.75$).

6 City-Scale Evaluation: Pipeline Demonstration across Manhattan

We next demonstrate that the two methods developed (in Sec. 5.1 and Sec. 5.2) scale beyond specific intersection examples: (i) GPS-camera mismatch can be used for Sybil attacks detection across hundreds of locations, and (ii) neighboring cameras across Manhattan can provide a conservative integrity signal when a camera is missing (blind spots). Within New York City, we focus on Manhattan as its camera density and road complexity provide a challenging urban setting.

6.1 Manhattan-Wide Attack Detection

We scale the GPS-camera mismatch detector and carry a Manhattan-wide run of the pipeline. We use the same pipeline and parameters as in Section 5.1: cameras are matched to nearby TomTom road links using a 0.01-mile radius, counts are resampled and smoothed, and mismatch is scored using the one-sided normalized windowed accumulation signal. We evaluate all Manhattan cameras with sufficient coverage before and after the attack activation during Apr 21–29.

A practical issue at this scale is that a small subset of locations is chronically unstable in this dataset (e.g., fragmented availability, blank intervals, or varied count behavior), which can trigger alarms even in the non-attacked baseline. Since such feeds are not usable for integrity monitoring without site-specific handling, we exclude such cameras. This yields a filtered evaluation set of 223 cameras out of 265 initially usable cameras (about 15% removed). We report the Manhattan-wide results on this filtered set.

We model a Sybil-style manipulation by inflating the GPS-derived congestion signal for each camera i out of the 223 relevant cameras. Concretely, we add an offset δ to the per-camera fused GPS congestion intensity $g_i(t)$ after the attack start time, creating an attacked metrics:

$$g_i^{\text{attack}}(t) = \min\big(1,\, g_i(t) + \delta\big), \qquad t \geq t_0,$$

where $t_0 =$ Apr 26, 2025 00:00. We range the attack volume $\delta \in \{0.01, 0.02, \ldots, 0.20\}$ to assess the impact of its volume on the detectability, and compute detection rates over the filtered camera set.

Figures 6 and 7 summarize the Manhattan-wide results. Figure 6 shows per-camera outcomes for a representative attack magnitude ($\delta = 0.05$), with 91.5% detection rate. Many of the remaining non-detections are spatially concentrated rather than uniformly distributed. For example, several cluster near the Queensboro Bridge corridor on the east side of Manhattan, which may reflect location-specific variability. Notably, as the injection magnitude increases, these areas also begin to trigger, and the non-detection region shrinks accordingly. Per Figure 7, which reports detection rates across the full range of the magnitudes, detection increases monotonically with attack magnitude: it rises from 18.8% at $\delta = 0.01$ and reaches $\approx 99\%$ by $\delta \approx 0.10$.

Overall, detection follows the same monotonic trend and saturates as the injected manipulation grows. These Manhattan-wide results show that GPS–camera mismatch monitoring remains effective when applied at city scale, across hundreds of intersections.

6.2 Manhattan-Wide Neighbor Consensus for Blind Spots

We next generalize the spatial neighbor consensus demonstration beyond the two intersections we studied in Section 5. We treat each camera i in Manhattan as a potentially missing center. We measure how often its high-load bins are *not* corroborated by surrounding cameras.

For each camera i, we take its five nearest neighboring cameras (by geographic distance) as the comparison set. We smooth each camera's vehicle-count signal (running average over samples) and aggregate it into bins. As before, a bin is marked high-load when the center camera's binned value is high relative to its own baseline.

We then mark a red center bin as *unique* if none of the neighbors are red in the same bin or in the immediately previous bin (to allow for small timing offsets). Finally, for each camera we summarize neighbor consensus using the fraction of high-traffic bins that are unique:

$$\text{UniqueRate}(c) = \frac{\#\text{ unique high-traffic bins at } i}{\#\text{ high-traffic bins at } i}.$$

Lower unique rates indicate *stronger* neighbor agreement.[3]

[3]Intuitively, a high-traffic bin is unique only when the center camera

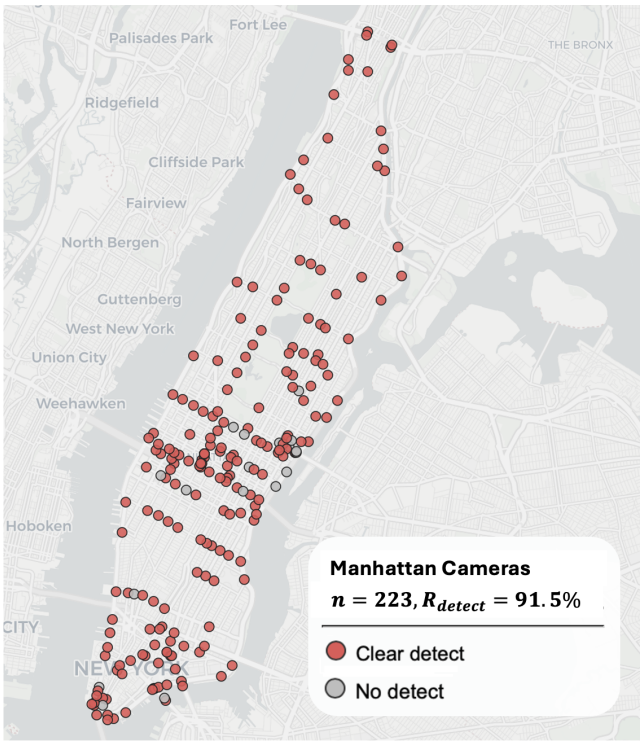


Figure 6: Per-camera outcomes at additive attack $\delta = 0.05$ on filtered Manhattan cameras ($n = 223$). Gray: no detection. red: (clear) detection.

Figures 8 and 9 summarize results for Apr 21–29, 2025 over the active Manhattan cameras. The city map in Fig. 8 shows that most locations are in the low-unique-rate regime, with a smaller set of outliers. Note that the remaining outliers are not uniformly distributed. Several appear near boundary regions such as bridge approaches and major tunnel entrances (e.g., the Midtown Tunnel), where the 'nearest neighbors' within Manhattan may not capture the same traffic dynamics. In these areas, the natural neighborhood for corroboration can span across borough boundaries.

The distribution/CDF view in Figure 9 confirms this quantitatively: the median unique rate is 3.32% (mean 8.67%), 73.62% of cameras have unique rate at most 10%, and 84.66% have unique rate at most 20%.

From a security perspective, these Manhattan-wide results support using nearby cameras as a mismatch signal when a center camera is missing. Locations with low unique rates can use stricter neighbor-based validation rules.

indicates high load but none of its neighbors do; thus, fewer unique bins means high-load periods are likely to be observed by at least one neighbor camera.

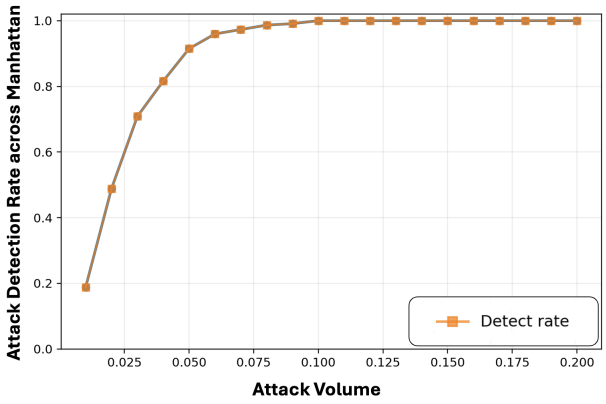


Figure 7: Manhattan-wide detection and clear-detection rates versus additive attack magnitude $\delta \in [0.01, 0.20]$ (filtered cameras, $n = 223$). Detection rates increase monotonically with attack magnitude and saturate for larger δ.

7 Discussions and Limitations

The Multi-Modal Approach to Integrity Validation Existing Sybil defenses (including endpoint attestation, behavioral filtering, and identity analysis; see Section 2) provide an important defense layer. The defense schemes can raise attacker cost and block low-effort manipulation. Yet, because these defenses still reason primarily over device-generated reports, they remain constrained by the fact that the attacker can control the reporting channel. A determined adversary is likely to still bypass these mechanisms using many real devices, or mimic benign mobility patterns closely enough to confuse detectors.

Our pipeline adds an independent verification layer using multi-modal approach. Street cameras provide infrastructure-grounded evidence from outside the GPS reporting channel: they observe actual vehicles at fixed locations and allow us to derive aggregate signals (vehicle counts) that can compare to GPS-derived congestion.

Combining the two data inputs increases attack cost directly. Now, to create false congestion without triggering an alert, an adversary must either manipulate device reports while also producing consistent physical effects, compromise infrastructure sensing, or exploit the cases where inputs are not available. While this does not guarantee perfect security, it provides a stronger mechanism than relying on device reports alone. Our Manhattan-wide evaluation further shows that this mechanism scales and remains useful even with natural camera coverage gaps.

Note that an attacker could also, in principle, target camera infrastructure or stage physical effects. Thus, this design on its own is not an absolute source of trust. However, this represents a different class of attack than Sybil injection, typically requiring localized presence and/or infrastructure intrusion rather than scalable remote data injection. In this paper, we

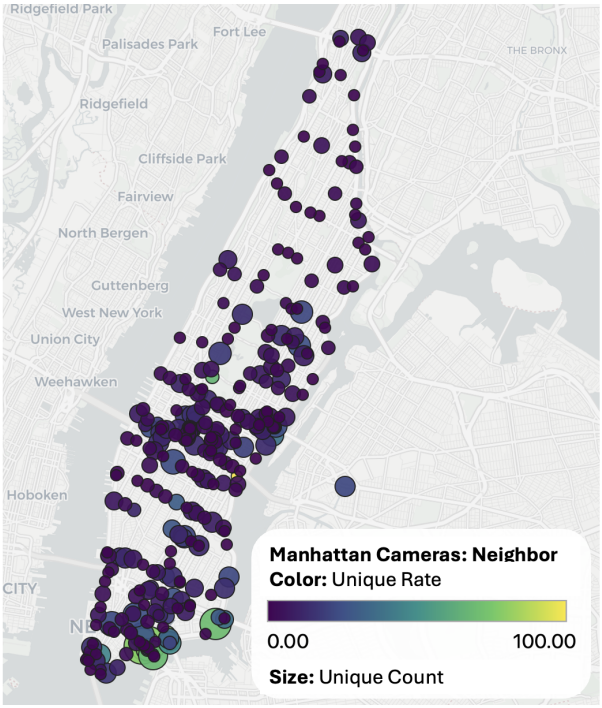


Figure 8: Manhattan-wide neighbor cross-validation ($k = 5$): per-camera unique-rate map. Color encodes unique rate, and marker size encodes unique-bin count. Most locations are in the low-unique regime, with a smaller set of spatial outliers.

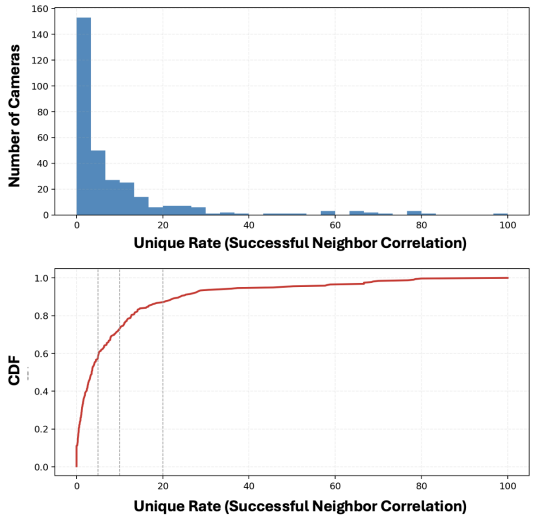


Figure 9: Manhattan-wide neighbor cross-validation ($k = 5$): distribution of unique rates (top) and cumulative distribution function (bottom). The mass near low values indicates that neighbor cross-validation is strong for most cameras, with a long but limited high-unique tail.

do not attempt to defend against a strong adversary who can compromise municipal camera infrastructure, tamper with video feeds, or systematically stage physical traffic to match a fabricated congestion narrative. Those remain outside the scope of our threat model.

Privacy, Anonymity, and Data Handling. Camera-based sensing raises privacy concerns, particularly in municipal deployments where data may cross organizational boundaries, and especially where pedestrians are part of the environment. Our pipeline operates on aggregate vehicle counts; it requires neither storing identifiable imagery nor tracking vehicles across a sequence of cameras views. Nevertheless, real-world deployment requires clear data governance, including access controls, auditing, and transparency about what is collected and how it is used.

A practical requirement is that privacy protections must be enforced *before* any video or derived data leaves the local sensing domain. This aligns with prior work on the NSF PAWR COSMOS testbed, which demonstrated a real-time anonymization pipeline for smart-city intersections that removes sensitive identifiers (e.g., pedestrian faces and vehicle license plates) at the edge, enabling downstream analytics without exporting raw identifiable video [2, 11].

Accordingly, we recommend an 'edge-first' deployment model for our pipeline. I.e., process raw camera feeds locally, and export only privacy-preserving aggregated data (in our case, per-camera vehicle-count time series) to the downstream mismatch detector.

Why not rely on cameras alone? A natural question is whether we could avoid integrity risks posed by mobile devices at all, by relying entirely on camera-based sensing. While street cameras provide strong, infrastructure-grounded evidence at specific locations, they do not replace GPS-derived telemetry. GPS feeds offer broader spatial coverage and different operational value: they support end-to-end routing and trip planning, and in many settings they provide demand-side context that cameras cannot directly observe (e.g., where vehicles intend to go, as captured within navigation apps). They also enable city-scale mobility statistics without depending on the availability of camera streams across jurisdictions.

For these reasons, our goal is not to replace GPS-based traffic inference, but to treat cameras as an additional verification source when integrity is in doubt.

8 Conclusions

In this paper, we addressed the integrity risk posed by Sybil attacks against GPS-based traffic inference in navigation platforms, including routing and planning for automated-driving systems. To add infrastructure-grounded evidence, we proposed using fixed street cameras as a real-time verification

layer for GPS-derived traffic signals.

We presented and implemented a city-scale GPS–camera pipeline that (i) associates each camera with nearby road links, (ii) aligns camera-derived vehicle counts with GPS-based congestion metrics, and (iii) scores cross-source mismatch using residual-based detection. We also addressed camera blind spots: since coverage can be incomplete or intermittent, we evaluated a neighbor-consensus test showing that surrounding intersections can often support conservative detection even when direct camera evidence at a target location is missing.

We evaluated the pipeline at city scale using New York City data (NYC DOT cameras and TomTom traffic levels). Across Manhattan, with more than 200 active cameras, the system achieves strong detection under injected GPS inflation: it reaches 91.5% clear detection at a moderate attack level and approaches 99% under stronger manipulations.

Overall, these results suggest that "digital eyes on the road" can complement report-based defenses by providing a deployable multi-modal detection signal for traffic inference. Future work includes evaluating additional attack strategies, extending coverage beyond dense urban cores, and integrating additional infrastructure signals to account for attacks on the cameras themselves.

Ethical Consideration This work does not raise any ethical concerns.

Acknowledgments

This research was supported in part by the NSF Center for Smart Streetscapes (CS3) under Grant EEC-2133516, NSF grant CNS-2450567, NSF grant CNS-2038984 and corresponding support from the Federal Highway Administration (FHWA), NSF grant CNS-2148128, funds from federal agency and industry partners as specified in the Resilient & Intelligent NextG Systems (RINGS) program, by the Fulbright Scholars Program, by the Fulbright Scholar Program, a program of the United States Department of State Bureau of Educational and Cultural Affairs and by the Zuckerman STEM Leadership Program.

References

[1] Lorenzo Alvisi, Allen Clement, Alessandro Epasto, Silvio Lattanzi, and Alessandro Panconesi. Sok: The evolution of sybil defense via social networks. In *Proc. of IEEE SP'13*, May 2013.

[2] Alex Angus, Zhuoxu Duan, Gil Zussman, and Zoran Kostić. Real-time video anonymization in smart city intersections. In *Prof. of IEEE MASS'22*, Oct. 2022.

[3] Brian Barrett. An artist used 99 phones to fake a Google maps traffic jam. Wired, February 2020. https://www.wired.com/story/99-phones-fake-google-maps-traffic-jam/.

[4] Neal E. Boudetter. Building a road map for the self-driving car. The New York Times, 2017.

[5] Joel W Branch, Chris Giannella, Boleslaw Szymanski, Ran Wolff, and Hillol Kargupta. In-network outlier detection in wireless sensor networks. *Knowledge and information systems*, 34(1):23–54, 2013.

[6] Qiang Cao, Michael Sirivianos, Xiaowei Yang, and Tiago Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *Proc. of NSDI'12*, Apr. 2012.

[7] Chao Chen, Jaimyoung Kwon, John Rice, Alexander Skabardonis, and Pravin Varaiya. Detecting errors and imputing missing data for single-loop surveillance systems. *Transportation Research Record*, 1855(1):160–167, 2003.

[8] Ruilong Deng, Gaoxi Xiao, Rongxing Lu, Hao Liang, and Athanasios V Vasilakos. False data injection on state estimation in power systems—attacks, impacts, and defense: A survey. *IEEE transactions on industrial informatics*, 13(2):411–423, 2016.

[9] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[10] Cihan Eryonucu and Panos Papadimitratos. Sybil-based attacks on google maps or how to forge the image of city life. In *Proc. of ACM WiSec'22*, May 2022.

[11] Mahshid Ghasemi, Yongjie Fu, Xinyu Ouyang, Peiran Wang, Mehmet Kerem Turkcan, Jhonatan Tavori, Sofia Kleisarchaki, Thomas Calmant, Levent Gürgen, Zoran Kostic, et al. Real-time video analytics for urban safety: Deployment over edge and end devices. In *Proc. of ACM/IEEE SEC'25*, Dec. 2025.

[12] Amber Hankins, Tapadhir Das, Shamik Sengupta, and David Feil-Seifer. Eyes on the road: A survey on cyber attacks and defense solutions for vehicular ad-hoc networks. In *Proc. of IEEE CCWC'23 (Computing and Communication Workshop and Conference)*, Mar. 2023.

[13] Douglas M Hawkins and David H Olwell. *Cumulative sum charts and charting for quality improvement*. Springer Science & Business Media, 2012.

[14] Alex Hern. Berlin artist uses 99 phones to trick Google maps traffic jam alert. The Guardian, February 2020.

[15] Juan C Herrera, Daniel B Work, Ryan Herring, Xuegang Jeff Ban, Quinn Jacobson, and Alexandre M Bayen.

Evaluation of traffic data obtained via gps-enabled mobile phones: The mobile century field experiment. *Transportation Research Part C: Emerging Technologies*, 18(4):568–583, 2010.

[16] Albert S Huang, David Moore, Matthew Antone, Edwin Olson, and Seth Teller. Finding multiple lanes in urban road networks with vision and lidar. *Autonomous Robots*, 26(2):103–122, 2009.

[17] Todd E Humphreys, Brent M Ledvina, Mark L Psiaki, Brady W O'Hanlon, Paul M Kintner, et al. Assessing the spoofing threat: Development of a portable gps civilian spoofer. In *Proc. of ION GNSS'08 (International technical meeting of the satellite division of the ION)*, Sep. 2008.

[18] Kai Jansen, Matthias Schäfer, Daniel Moser, Vincent Lenders, Christina Pöpper, and Jens Schmitt. Crowd-gps-sec: Leveraging crowdsourcing to detect and localize gps spoofing attacks. In *Proc. of IEEE SP'18*, 2018.

[19] Tobias Jeske. Floating car data from smartphones: What google and waze know about you and how hackers can control traffic. *Proc. of the BlackHat Europe'13*, 2013.

[20] Kichun Jo, Keounyup Chu, and Myoungho Sunwoo. Interacting multiple model filter-based sensor fusion of gps with in-vehicle sensors for real-time vehicle positioning. *IEEE Transactions on Intelligent Transportation Systems*, 13(1):329–343, 2011.

[21] Yao Liu, Peng Ning, and Michael K Reiter. False data injection attacks against state estimation in electric power grids. *ACM Transactions on Information and System Security (TISSEC)*, 14(1):1–33, 2011.

[22] Alexander Meier, Claudia Kirch, and Haeran Cho. mosum: A package for moving sums in change-point analysis. *Journal of Statistical Software*, 97:1–42, 2021.

[23] Sashank Narain, Aanjhan Ranganathan, and Guevara Noubir. Security of gps/ins based on-road location tracking systems. In *Proc. of IEEE SP'19*, May 2019.

[24] NYC Department of Transportation. New engalnd 511 cameras. http://www.newengland511.org/cctv/.

[25] NYC Department of Transportation (DOT). Real-time traffic cameras (rtti). https://nyctmc.org/.

[26] Ewan S Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

[27] Simone Raponi, Savio Sciancalepore, Gabriele Oligeri, and Roberto Di Pietro. Road traffic poisoning of navigation apps: Threats and countermeasures. *IEEE Security & Privacy*, 20(3):71–79, 2021.

[28] Dipankar Raychaudhuri, Ivan Seskar, Gil Zussman, Thanasis Korakis, Dan Kilper, Tingjun Chen, Jakub Kolodziejski, Michael Sherman, Zoran Kostic, Xiaoxiong Gu, et al. Challenge: Cosmos: A city-scale programmable testbed for experimentation with advanced wireless. In *Proc. of ACM MOBICOM'20*, Sep. 2020.

[29] Meital Ben Sinai, Nimrod Partush, Shir Yadid, and Eran Yahav. Exploiting social navigation. In *BlackHat Asia 2015*.

[30] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. On the requirements for successful gps spoofing attacks. In *Proc. of ACM CCS'11*, Oct. 2011.

[31] TomTom. Tomtom traffic flow api. https://developer.tomtom.com/.

[32] TomTom. Traffic flow segment definition. https://developer.tomtom.com/traffic-api/documentation/tomtom-maps/traffic-flow/flow-segment-data.

[33] Nicholas Tufnell. Students hack Waze, send in army of traffic bots. Wired, March 2014. https://www.wired.com/story/waze-hacked-fake-traffic-jam/.

[34] Mehmet Kerem Turkcan, Jhonatan Tavori, Javad Ghaderi, Gil Zussman, Zoran Kostic, and Andrew Smyth. A vision-based analysis of congestion pricing in new york city. *arXiv preprint arXiv:2602.03015*, 2026.

[35] Lelitha Vanajakshi and LR Rilett. Loop detector data diagnostics based on conservation-of-vehicles principle. *Transportation research record*, 1870(1):162–169, 2004.

[36] Gang Wang, Bolun Wang, Tianyi Wang, Ana Nika, Haitao Zheng, and Ben Y Zhao. Defending against sybil devices in crowdsourced mapping services. In *Proc. of ACM MobiSys'16*, Jun. 2016.

[37] Gang Wang, Bolun Wang, Tianyi Wang, Ana Nika, Haitao Zheng, and Ben Y Zhao. Ghost riders: Sybil attacks on crowdsourced mobile mapping services. *IEEE/ACM transactions on networking*, 26(3), 2018.

[38] James J. Q. Yu. Sybil attack identification for crowdsourced navigation: A self-supervised deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 22(7):4622–4634, 2021.

[39] Kexiong Curtis Zeng, Shinan Liu, Yuanchao Shu, Dong Wang, Haoyu Li, Yanzhi Dou, Gang Wang, and Yaling Yang. All your {GPS} are belong to us: Towards stealthy manipulation of road navigation systems. In *Proc. of USENIX Security'18*, Aug. 2018.

[40] Yang Zhang, Nirvana Meratnia, and Paul Havinga. Outlier detection techniques for wireless sensor networks: A survey. *IEEE communications surveys & tutorials*, 12(2):159–170, 2010.

Charge It to My Neighbor: A Relay Attack on ISO 15118 Plug and Charge Payment

Jakob Löw
Technische Hochschule Ingolstadt

Vishwa Vasu
Technische Hochschule Ingolstadt

Thomas Hutzelmann
Technische Hochschule Ingolstadt

Hans-Joachim Hof
Technische Hochschule Ingolstadt

Abstract

ISO 15118, the leading standard for DC fast charging in Europe, includes a plug-and-charge mechanism that allows electric vehicles to handle payment automatically via contract certificates. We present a novel relay attack against this mechanism: an attacker builds a fake charging station, plugs it into a victim's vehicle, and relays the cryptographic authentication to a real charging station — charging the attacker's vehicle while billing the victim. The attack exploits the absence of station-identifying information in the plug-and-charge signature, combined with weaknesses in how ISO 15118 handles TLS certificates. We provide a proof-of-concept implementation demonstrating the full attack chain and discuss possible mitigations and alternatives. As plug-and-charge adoption grows, addressing this vulnerability is critical before it becomes widely exploitable.

1 Introduction

Electric vehicles are gaining in popularity. Especially in countries with corresponding incentives to buy electric over combustion engines, like China and Norway, they reach nearly 100% market share in newly registered vehicles [1]. With the increasing number of electric vehicles on the road also comes a growing demand for fast charging stations, allowing electric vehicles to quickly recharge before continuing their trip.

In Europe, the leading standard for charging communication between an electric vehicle and a charging station is ISO 15118 [13–15]. This standard not only describes the required communication between the two parties but also includes additional features, such as plug and charge payment, allowing the user to handle payment directly after plugging in a vehicle without further user interaction. A recent study found that plug and charge is the most preferred yet least used payment method among electric vehicle drivers, while mobile app and RFID card based payment are the most used but least preferred [23]. This significant discrepancy suggests that plug and charge adoption will continue to grow. Handling payment directly within the charging communication, however, also increases the attack surface and, especially, the reward potential that attackers could have when attacking charging communication.

In this paper, we present a novel relay attack against the plug and charge payment mechanism of ISO 15118. An attacker builds a fake charging station, plugs it into a victim's vehicle, and relays the cryptographic payment authentication to a real charging station — charging the attacker's vehicle while billing the victim. The attack exploits the absence of station-identifying information in the plug and charge signature, combined with weaknesses in how ISO 15118 handles TLS certificates. We provide a proof-of-concept implementation demonstrating the full attack chain and discuss possible mitigations, including an alternative out-of-band payment approach that avoids the vulnerability entirely.

While, to the best of our knowledge, the relay attack itself has not been described before, our proof-of-concept implementation builds on shortcomings in charging communication security identified in related work [11, 20–22]. Conti et al. [6] have previously demonstrated an attack against plug and charge payment; however, their approach relies on opaque forwarding, which introduces several limitations discussed later.

Section 2 provides background on the plug and charge mechanism and TLS certificate handling in ISO 15118. Section 3 discusses related work on charging communication security. Section 4 introduces the adversary model used for evaluating the vulnerability. Section 5 presents the plug and charge relay vulnerability. Section 6 describes our proof-of-concept implementation. Finally, Section 7 discusses mitigations and an alternative payment approach.

2 Background

ISO 15118 [13–15] defines the communication protocol between electric vehicles and charging stations for DC fast charging in Europe. The standard uses powerline communication over the charging cable, with IPv6 for network addressing and optionally TLS for session encryption. Before the main

charging communication begins, several initialization steps take place: low-level signaling per IEC 61851 [12], powerline network formation via the Signal Level Attenuation Characterization (SLAC) protocol, IPv6 address assignment, and TCP connection parameter exchange using the Service Discovery Protocol (SDP). Previous work has identified vulnerabilities in many of these initialization steps [9, 11, 16, 20]. For this paper, the relevant aspects of ISO 15118 are the plug and charge payment mechanism and the use of TLS certificates, both described below.

2.1 Plug and Charge

After the SDP request and response described in the previous section, the two communication parties establish a TCP connection. When TLS is supported by both parties, a standard TLS handshake follows. Within this established connection, the main charging communication is performed. The vehicle sends a `ServiceDiscoveryReq` to which the charging station responds with a `ServiceDiscoveryRes`. This response includes a list of supported payment methods. As of today, mostly the `ExternalPayment` method is used [18, 20]. With this method, the user has to handle payment directly with the charging station, for example, using a credit card.

Alternatively, the charging station may offer `ContractCertificate` payment. If the vehicle selects this payment method in its `PaymentServiceSelectionReq`, it follows up with its certificate in a `PaymentDetailsReq`. The charging station responds with a `PaymentDetailsRes`, including a unique challenge consisting of random bytes. The vehicle has to sign the challenge bytes using the private key belonging to the given certificate, sending the signature in a `AuthorizationReq`. This way, the vehicle proves to the charging station that it has ownership of the private key belonging to the contract certificate presented in the previous step. Figure 1 illustrates this communication flow, when plug and charge payment is used.

According to the ISO 15118 standard [13], plug and charge shall only be used when the connection is encrypted using TLS. For establishing a TLS connection with ISO 15118-2 [13], only the server needs to authenticate itself. With the newer ISO 15118-20 [15], mutual authentication is required for establishing an encrypted connection. With ISO 15118, there are two separate public key infrastructures (PKIs). One provides certificates to charging stations and vehicles for establishing TLS communication. While ISO 15118-2 requires only the charging station to possess a certificate, the newer ISO 15118-20 demands that both sides authenticate using a valid certificate obtained from the first PKI. The second PKI is responsible for handing out contract certificates used for payment. On a technological level, contract certificates used for payment are identical to certificates used for session authentication. While a vehicle typically has only one fixed certificate for session authentication, this separation enables it to have multiple, possibly user-configurable contract certificates.

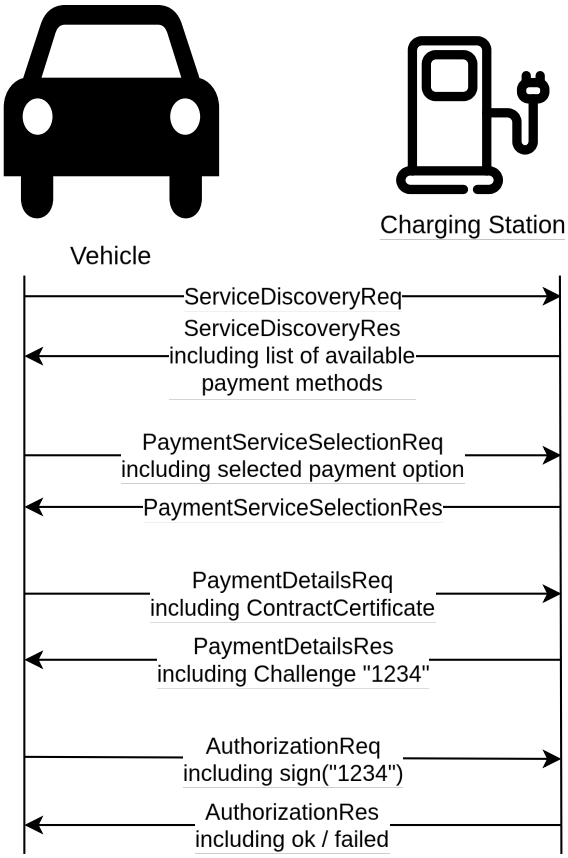


Figure 1: Plug and Charge Flow

2.2 TLS Certificate Handling in ISO 15118

For securing the main charging communication, the ISO 15118 standard uses Transport Layer Security. While TLS is a well-established protocol for securing and authenticating sessions, the way ISO 15118 employs it leaves gaps that are critical enablers for the relay attack presented in this paper.

TLS uses certificate-based authentication with a public key infrastructure for creating and signing certificates. On the web, where TLS is commonly used, this means there are a number of root certificate authorities. Website operators can acquire a certificate for their website from any of the commonly trusted certificate authorities. Web browsers usually bundle a large number of certificate authority root certificates and update them regularly. In the vehicle ecosystem, where storage space and internet connectivity are sparse, managing large certificate stores and especially updating them is significantly harder. As of today, there are multiple certificate authorities for charging communication; however, it is not clear to the public which OEMs ship their vehicles with trust for which authorities.

The most significant problem for the attack presented in this paper is the lack of verifiable charging station identification.

When on the web connecting to a website `example.com` using TLS, the web browser ensures the presented certificate was issued to `example.com`. For charging communication, the ISO 15118 standard requires the charging station ID to be encoded as the certificate identity. While this identity is unique per charging station, there is no way for the vehicle to check if the provided certificate belongs to the charging station currently connected to [20]. A potential attacker could acquire one certificate from any charging station and use this certificate to impersonate any other charging station. While certificate compromise can happen on the web as well, the impact is different: on the web, when a certificate for `example.com` is known to an attacker, the attacker can only impersonate this one website. With ISO 15118, when a certificate for one charging station is compromised, an attacker can impersonate any charging station, including stations from other vendors, operators or in other countries. This creates a weakest-link problem: the security of the entire ecosystem is bounded by the least secure charging station. Real-world evidence suggests that obtaining certificates from charging stations is feasible. Even Alpitronic, the largest DC fast charging hardware vendor in Europe, has been affected by vulnerabilities that could enable certificate extraction. In 2024, it was discovered that the configuration panel of Alpitronic Hyperchargers is accessible from the powerline communication network, meaning any connected vehicle can reach the charging station's administrative interface [24]. Additionally, a CISA advisory revealed that Alpitronic Hyperchargers shipped with default credentials for this administrative interface [5] (CVE-2024-4622). While neither vulnerability has been publicly demonstrated to lead to certificate extraction, they illustrate the kind of access an attacker could leverage, and the fact that they affect the market-leading vendor underscores the weakest-link concern.

3 Related Work

With the increasing adoption of electric vehicles, more and more cybersecurity researchers have uncovered vulnerabilities in ISO 15118. Most related work has focused on the initialization phases of charging communication, such as vulnerabilities in powerline communication [8, 11, 16], the SLAC process [9, 11], and service discovery [3, 19]. While these lower-layer vulnerabilities can facilitate man-in-the-middle positions that complement our attack (see Section 6.4), the relay vulnerability we present operates independently at the application layer.

The most closely related work is EVExchange by Conti et al. [6], which presents a relay attack on ISO 15118 charging communication. In EVExchange, an attacker places two malicious devices at two adjacent charging stations managed by the same backend infrastructure. These devices forward the entire network traffic between the two EVSEs, effectively swapping which vehicle communicates with which charging station. Since the relay operates at the network layer, it does not need to decrypt TLS-encrypted traffic; instead, it opaquely forwards all packets. The attacker can then control when to stop each vehicle's charge, making the victim pay for the attacker's energy consumption. EVExchange requires both the attacker and victim vehicles to be physically present at the same charging location, as the malicious devices must be installed between the vehicles and adjacent EVSEs. As a countermeasure, Conti et al. propose a distance bounding protocol that measures round-trip times of rapid packet exchanges to detect the additional latency introduced by the relay.

Because EVExchange relays the entire charging session at the network layer, the charging parameters sent by each vehicle must be consistent with the physical charging station it is actually connected to. In particular, during DC charging, the vehicle continuously reports its present battery voltage in `CurrentDemandReq` messages. Both vehicles must therefore have the same battery voltage for the relayed session to proceed without errors. Even when two vehicles start at the same voltage, the attacker's vehicle will see its battery voltage increase as it charges, causing a growing mismatch with the victim's reported values. The only way to avoid this inconsistency would be to charge both vehicles simultaneously, which eliminates the attacker's economic gain.

Table 1 summarizes the key differences between EVExchange and the vulnerability presented in this paper. In contrast to EVExchange, the plug and charge relay vulnerability presented in this paper (Section 5) operates at the application layer, specifically targeting the plug and charge authentication mechanism. Rather than forwarding entire network flows, our attack selectively relays the cryptographic messages used for payment authentication: the contract certificate, the challenge, and the signed response. This allows the attacker vehicle and the fake charging station to be at entirely different physical locations, communicating over a cellular network, similar to keyless go relay attacks on vehicles. Furthermore, while EVExchange requires tampering with existing charging infrastructure, our attack only requires the attacker to build a portable fake charging station that is plugged into the victim's vehicle.

4 Adversary Model

Bao et al. [3] have published an extensive threat analysis regarding ISO 15118, including the modeling of various adversaries. We utilize their adversary models for evaluating the impact of the described vulnerabilities and their potential exploitation by attackers. Of the adversaries defined by Bao et al., two are particularly relevant to the plug and charge relay vulnerability presented in this paper. Additionally, we introduce a third adversary, the *equipment supplier*, who is not defined by Bao et al. but is relevant for assessing the real-world scalability of the attack.

Table 1: Comparison of EVExchange and this work

Property	EVExchange [6]	This paper
Relay layer	Network (IP)	Application (ISO 15118)
Relayed data	All network traffic	Payment authentication messages
Vehicle Requirements	Same battery voltage	None
Actual Charging	Both vehicles	Only attacker vehicle
TLS broken	No (opaque forwarding)	No (separate sessions)
Countermeasure	Distance bounding	Context binding / backend

The *freeloader* aims to charge their own vehicle for free by making someone else pay for the charged energy. The freeloader has basic technical knowledge of electric vehicle charging and physical access to publicly accessible vehicles and charging stations. However, the freeloader does not have the capability to develop custom hardware or software and instead relies on readily available tools or equipment obtained from third parties. Using the relay attack described in Section 5, a freeloader could build or acquire a fake charging station, plug it into a victim's vehicle, and relay the payment authentication to a real charging station where the freeloader's vehicle is charging.

The *contract-sharer* aims to share one charging contract with a large number of other users, possibly abusing flat charging fees. Similar to the freeloader, the contract-sharer has basic technical knowledge and physical access to charging infrastructure, but does not necessarily possess advanced hardware or software development skills. The contract-sharer's distinguishing characteristic is the organizational capability to coordinate multiple users and distribute relay equipment or access credentials among them. By relaying the plug and charge authentication from a single contracted vehicle, a contract-sharer could enable multiple unauthorized vehicles to charge under the same contract.

The *equipment supplier* manufactures and sells ready-to-use relay equipment to customers such as freeloaders and contract-sharers. Unlike the other two adversaries, the equipment supplier possesses advanced technical knowledge of charging communication protocols, powerline communication, and embedded systems development. The equipment supplier is capable of developing custom hardware and software, such as portable fake charging stations and man-in-the-middle devices that combine known powerline communication vulnerabilities [9, 11, 16] with the plug and charge relay vulnerability described in Section 5. The equipment supplier's motivation is financial profit from selling attack equipment rather than directly exploiting the vulnerability. Similar to the market for keyless go relay devices used in vehicle theft, the existence of an equipment supplier would significantly lower the barrier to entry for freeloaders and contract-sharers, as they would no longer need any technical expertise to carry out the attack.

5 Plug and Charge Relay Vulnerability

As described in Section 2.1, ISO 15118 supports contract-based payment, also known as plug and charge. With this method, the vehicle transmits its contract certificate to the charging station, which responds with a random challenge. The vehicle then signs this challenge using the private key belonging to the contract certificate, proving ownership. Assuming the charging station uses a good random number generator and generates a new challenge for each session, this technique effectively prevents replay attacks. However, the shortcoming with the current mechanism defined by ISO 15118 is the absence of any identifying credential of the charging station for the vehicle to check. The vehicle does not include any additional information in the signed data, such as the current timestamp or information about the charging station it is connected to. Thus, while the challenge prevents replay attacks, it does not effectively prevent relay attacks. While Szakály et al. [20] already discussed the underlying TLS identification shortcomings that enable this attack, to the best of our knowledge, the plug and charge relay attack itself has not been previously described or implemented.

Normally, the use of TLS would prevent an attacker from creating a custom charging station emulator, as the lack of a valid certificate would prevent the victim vehicle from accepting it. However, due to the certificate shortcomings described in Section 2.2, an attacker could use certificates and private keys obtained from any compromised charging station to make the emulator impersonate any real charging station. Since a certificate from any vendor is accepted by all vehicles, compromising a single poorly secured charging station is sufficient to attack vehicles at any other station, regardless of that station's own security posture. This creates a weakest-link problem: the security of the entire ecosystem is bounded by the least secure charging station vendor. With ISO 15118-20, which mandates mutual TLS authentication, the attacker vehicle must additionally present a valid vehicle TLS certificate when connecting to the *regular charging station*. This means the attacker must also have compromised at least one vehicle's TLS certificate and private key. While this raises the bar compared to ISO 15118-2, payment handling is identical across both standard versions, so the fundamental relay vulnerability remains: the plug and charge signature still contains no

station-identifying information, and the attack proceeds in the same way once valid certificates for both sides are available.

To perform such a relay attack, an attacker could build a charging station emulator and plug it into a victim's vehicle. Simultaneously, the attacker vehicle is plugged into a regular charging station. After initializing the charging communication with the victim's vehicle, the vehicle sends its charging contract certificate. The attacker forwards this certificate to the charging station that the attacker's vehicle is plugged into, acting as if this certificate belongs to the attacker's vehicle. The charging station then generates a random challenge and sends it to the attacker's vehicle. Normally, the attacker would need the private key belonging to the victim's contract certificate in order to sign the challenge. However, the attacker simply forwards the challenge to the fake charging station, which sends it to the victim vehicle. The victim signs the challenge and returns the signature, which the attacker relays back to the regular charging station. Since the signed data contains no station-specific information, there is no way for the regular charging station to detect that this signature was originally created by a different vehicle for a different charging station. Figure 2 visualizes the communication flow of this relay attack.

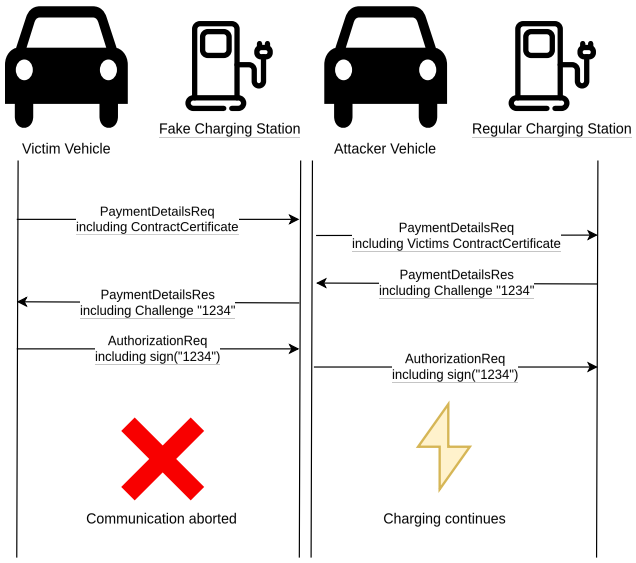


Figure 2: Plug and Charge Relay Attack

6 Implementing and Combining Vulnerabilities for a Payment Fraud Proof of Concept

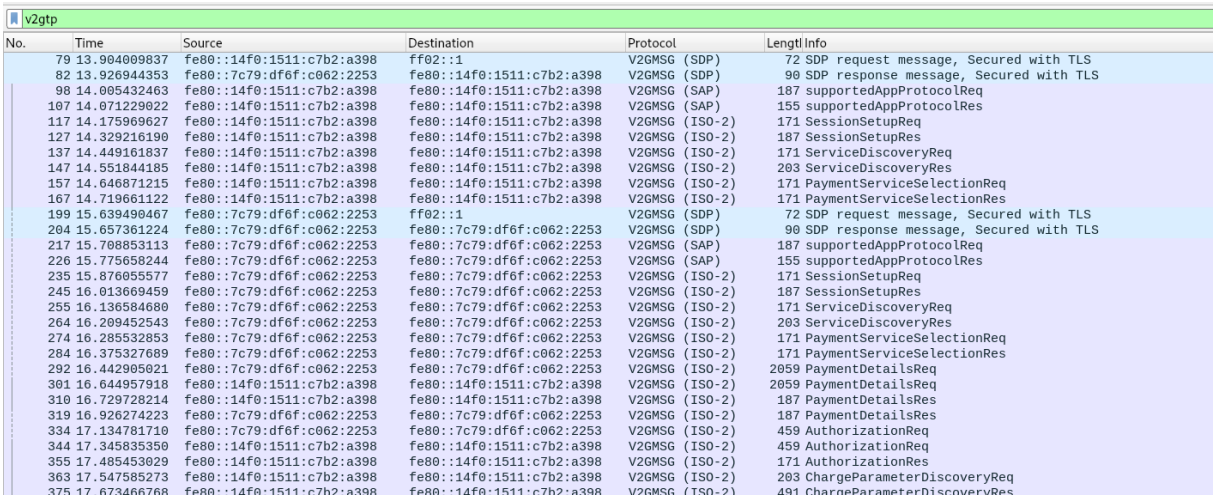
The previous section described a plug and charge relay vulnerability allowing an attacker to charge their vehicle while making a second vehicle, the victim vehicle, pay for the charging session. By combining this vulnerability with the vulnerability described in section 2.2, this section describes a possible proof of concept proving the feasibility of the attack. The software stacks, published as open source alongside this paper, can run entirely in a virtual environment, communicating with each other using local network interfaces. Additionally, the proof of concept was successfully tested using real hardware: our own charging station as the *regular charging station*, our EV simulator as the *victim vehicle*, and two powerline modems acting as *attacker vehicle* and *fake charging station*.

6.1 Communication Participants and Requirements

For this proof of concept, a custom PKI, as well as a set of virtual vehicles and virtual charging stations, is created. The following lists the created communication participants, which are also present in figure 2:

- The *victim vehicle*, which in reality could be any vehicle found parked on the street, after forcefully opening its charge port.
- The *attacker vehicle*. The goal of the attack is to charge the *attacker vehicle* with energy being billed to the *victim vehicle*.
- The *regular charging station*. This is the charging station that will be tricked into charging the *attacker vehicle* while billing the *victim vehicle* for the provided energy.
- The *compromised charging station* does not actually participate in the communication during the attack, but the attacker was previously able to extract this charging station's certificate and private key.
- The *fake charging station* is a purpose-built device. It does not need to actually provide power but merely needs to communicate with the *victim vehicle* to perform the plug and charge handshake.

In order to create the *attacker vehicle* as well as the *fake charging station*, a customized ISO 15118 stack implementation is required. Normally, when sending a `PaymentDetailsReq`, the vehicle includes its own certificate from its certificate store. For the attack to be successful, the *attacker vehicle* instead needs to send the certificate of the *victim vehicle*, which was received by the *fake charging station*. Similarly, when the *fake charging station* sends the `PaymentDetailsRes` to the *victim vehicle* the included challenge needs to be identical to the original challenge received by the *attacker vehicle* from the *regular charging station*. Additionally, the *attacker vehicle* will need to include the signature received by the *fake charging station* from the *victim vehicle* when sending its `AuthorizationReq` to the *regular charging station*. Therefore, a real-time communication between the *fake charging station* and the *attacker vehicle* is required, allowing them to exchange the certificate, challenge, and signature.



v2gtp

No.	Time	Source	Destination	Protocol	Lengtl	Info
79	13.904009837	fe80::14f0:1511:c7b2:a398	ff02::1	V2GMSG (SDP)	72	SDP request message, Secured with TLS
82	13.926944353	fe80::7c79:df6f:c062:2253	fe80::14f0:1511:c7b2:a398	V2GMSG (SDP)	90	SDP response message, Secured with TLS
98	14.005432463	fe80::14f0:1511:c7b2:a398	fe80::14f0:1511:c7b2:a398	V2GMSG (SAP)	187	supportedAppProtocolReq
107	14.071229022	fe80::14f0:1511:c7b2:a398	fe80::14f0:1511:c7b2:a398	V2GMSG (SAP)	155	supportedAppProtocolRes
117	14.175969627	fe80::14f0:1511:c7b2:a398	fe80::14f0:1511:c7b2:a398	V2GMSG (ISO-2)	171	SessionSetupReq
127	14.329216190	fe80::14f0:1511:c7b2:a398	fe80::14f0:1511:c7b2:a398	V2GMSG (ISO-2)	187	SessionSetupRes
137	14.449161837	fe80::14f0:1511:c7b2:a398	fe80::14f0:1511:c7b2:a398	V2GMSG (ISO-2)	171	ServiceDiscoveryReq
147	14.551844185	fe80::14f0:1511:c7b2:a398	fe80::14f0:1511:c7b2:a398	V2GMSG (ISO-2)	203	ServiceDiscoveryRes
157	14.646871215	fe80::14f0:1511:c7b2:a398	fe80::14f0:1511:c7b2:a398	V2GMSG (ISO-2)	171	PaymentServiceSelectionReq
167	14.719661122	fe80::14f0:1511:c7b2:a398	fe80::14f0:1511:c7b2:a398	V2GMSG (ISO-2)	171	PaymentServiceSelectionRes
199	15.639490467	fe80::7c79:df6f:c062:2253	ff02::1	V2GMSG (SDP)	72	SDP request message, Secured with TLS
204	15.657361224	fe80::7c79:df6f:c062:2253	fe80::7c79:df6f:c062:2253	V2GMSG (SDP)	90	SDP response message, Secured with TLS
217	15.708853113	fe80::7c79:df6f:c062:2253	fe80::7c79:df6f:c062:2253	V2GMSG (SAP)	187	supportedAppProtocolReq
226	15.775658244	fe80::7c79:df6f:c062:2253	fe80::7c79:df6f:c062:2253	V2GMSG (SAP)	155	supportedAppProtocolRes
235	15.876055577	fe80::7c79:df6f:c062:2253	fe80::7c79:df6f:c062:2253	V2GMSG (ISO-2)	171	SessionSetupReq
245	16.013669459	fe80::7c79:df6f:c062:2253	fe80::7c79:df6f:c062:2253	V2GMSG (ISO-2)	187	SessionSetupRes
255	16.136584680	fe80::7c79:df6f:c062:2253	fe80::7c79:df6f:c062:2253	V2GMSG (ISO-2)	171	ServiceDiscoveryReq
264	16.209452543	fe80::7c79:df6f:c062:2253	fe80::7c79:df6f:c062:2253	V2GMSG (ISO-2)	203	ServiceDiscoveryRes
274	16.285532853	fe80::7c79:df6f:c062:2253	fe80::7c79:df6f:c062:2253	V2GMSG (ISO-2)	171	PaymentServiceSelectionReq
284	16.375327689	fe80::7c79:df6f:c062:2253	fe80::7c79:df6f:c062:2253	V2GMSG (ISO-2)	171	PaymentServiceSelectionRes
292	16.442905021	fe80::7c79:df6f:c062:2253	fe80::7c79:df6f:c062:2253	V2GMSG (ISO-2)	2059	PaymentDetailsReq
301	16.644957918	fe80::14f0:1511:c7b2:a398	fe80::14f0:1511:c7b2:a398	V2GMSG (ISO-2)	2059	PaymentDetailsReq
310	16.729728214	fe80::14f0:1511:c7b2:a398	fe80::14f0:1511:c7b2:a398	V2GMSG (ISO-2)	187	PaymentDetailsRes
319	16.926274223	fe80::7c79:df6f:c062:2253	fe80::7c79:df6f:c062:2253	V2GMSG (ISO-2)	187	PaymentDetailsRes
334	17.134781710	fe80::7c79:df6f:c062:2253	fe80::7c79:df6f:c062:2253	V2GMSG (ISO-2)	459	AuthorizationReq
344	17.345835350	fe80::14f0:1511:c7b2:a398	fe80::14f0:1511:c7b2:a398	V2GMSG (ISO-2)	459	AuthorizationReq
355	17.485453029	fe80::14f0:1511:c7b2:a398	fe80::14f0:1511:c7b2:a398	V2GMSG (ISO-2)	171	AuthorizationRes
363	17.547585273	fe80::14f0:1511:c7b2:a398	fe80::14f0:1511:c7b2:a398	V2GMSG (ISO-2)	203	ChargeParameterDiscoveryReq
375	17.673466768	fe80::14f0:1511:c7b2:a398	fe80::14f0:1511:c7b2:a398	V2GMSG (ISO-2)	491	ChargeParameterDiscoveryRes

Figure 3: Wireshark Log of the two charging sessions during the attack

6.2 Proof of Concept Implementation Details

In reality, this communication is likely to be handled via a cellular network, allowing the *fake charging station* to be at a different location than the *attacker vehicle*. In the proof of concept, the *attacker vehicle* and *fake charging station* exchange the certificate, challenge, and signature via a TCP connection, but run on the same machine. If the *attacker vehicle* reaches the step where it should send a `PaymentDetailsReq` before the certificate is available from the *fake charging station*, it simply stalls and waits for it to arrive over the network. For `PaymentDetailsReq` and `AuthorizationReq`, ISO 15118 specifies a response timeout of 2 s [13]. Since the relay only needs to forward small application-layer messages, a cellular network with typical round-trip times well below 100 ms would provide more than enough margin for the relay to complete within the protocol's timeout constraints.

Implementation-wise, the proof of concept is based on the ISO 15118 stack formerly developed and published by SwitchEV, now EcoG [10]. EcoG is a company selling commercial communication controllers and software for charging stations. While the software stack is published as open source, allowing for easy modifications and attack implementation, it is also deployed in real charging stations. In addition to the required changes for handling `PaymentDetailsReq`, `PaymentDetailsRes` and `AuthorizationReq` packets, the customized software stack also creates SSL key logs, allowing for easy inspection of encrypted traffic in Wireshark [25]. Our modified software stack is published under the original software license, allowing others to easily reproduce the attacks or try them against their own ISO 15118 implementations: https://github.com/securityinmobility/plug-and-charge-relay-poc

In order to run the proof of concept, in addition to the customized software stacks used for the *fake charging station* and *attacker vehicle*, two more ISO 15118 instances are required for the *regular charging station* and *victim vehicle*. In a purely virtual setup, the same ISO 15118 stack can be used in its original unmodified version. For our hardware-based test, a real charging station serves as the *regular charging station* and our EV simulator acts as the *victim vehicle*, while two powerline modems running the customized stack act as *attacker vehicle* and *fake charging station*. Since the relevant vulnerabilities described in section 2.2 and section 5 are not specific to one implementation, but rather apply to all implementations of the standard, in theory, any compliant ISO 15118 implementation could be used.

6.3 Proof of Concept Evaluation

Running the attack proof-of-concept produces a traffic log similar to the one shown in figure 3. In this screenshot, the IPv6 address of the *attacker vehicle* ends on `a398`, while the IPv6 address of the *victim vehicle* ends on `2253`. At first, the *attacker vehicle* initiates a charging session with the *regular charging station*, but stalls after the `PaymentServiceSelectionRes`. Normally, the *attacker vehicle* would follow up with a `PaymentDetailsReq`, but it has to wait for the content from the *victim vehicle* to relay the message. The charging session between *victim vehicle* and *fake charging station* is started slightly later. As soon as the *fake charging station* receives the `PaymentDetailsReq` the included certificate is also used by the *attacker vehicle* for its packet. Similarly, as soon as the *attacker vehicle* receives the `PaymentDetailsRes` including the challenge required to sign, the *fake charging station* sends exactly this challenge to the *victim vehicle*. After the signed challenge is received from the *victim vehicle* the communication with the victim stops. Only the charging session with the *attacker vehicle* continues.

6.4 From Proof of Concept to the Real World

As described in Section 4, the *freeloader* and the *contract-sharer* are the two adversaries which could directly benefit from abusing the described plug and charge relay vulnerability.

We successfully executed the attack both in a purely virtual setup and on real powerline hardware, as described in Section 6. In this subsection, we discuss the additional engineering that would be required to extend the attack from our controlled hardware setup to unmodified, off-the-shelf production vehicles encountered in the wild. While our hardware-based test already demonstrates the attack using real charging infrastructure, a real-world scenario requires targeting actual vehicles rather than an EV simulator. Building the *fake charging station* which performs high-level communication with the *victim vehicle* can easily be done using openly available charging communication evaluation kits [4, 17]. Modifying the charging communication stack inside a vehicle, to build the *attacker vehicle*, however, is a more complicated task. A simpler approach would be to utilize known powerline communication vulnerabilities to design a man-in-the-middle device, which modifies the communication between an unmodified *attacker vehicle* and a *regular charging station*. Previous work has demonstrated wireless powerline interception [16], network infiltration via SLAC sniffing [11], and man-in-the-middle attacks on charging communication [9, 22]. Combining these techniques allows designing a wireless charging communication man-in-the-middle device. This device could then perform plug and charge authentication with the charging station on one side and behave like a free charging station towards the *attacker vehicle*. Similar to keyless go relaying hardware utilized by car thieves, these known lower-layer vulnerabilities could be combined with the relay vulnerability presented in this paper to develop a pair of devices able to perform plug and charge relay attacks. As described in Section 4, this development and packaging of attack equipment into a consumer-ready product is the role of the *equipment supplier*, whose existence would make the attack accessible to freeloaders and contract-sharers without any technical expertise.

7 Possible Mitigations

This section discusses approaches for mitigating the plug and charge relay vulnerability presented in Section 5, as well as an alternative payment mechanism that avoids the vulnerability entirely.

7.1 Improving TLS Certificates for Charging Communication

For the plug and charge relay to work, the attacker needs to be able to impersonate a valid charging station towards the victim vehicle. Due to the TLS weakness described in Section 2.2, a single compromised charging station certificate is sufficient to impersonate any station worldwide. Fixing this does not directly address the relay vulnerability itself, but it would prevent this attack and potential future vulnerabilities from being exploited at scale, as the ability to impersonate arbitrary charging stations is a prerequisite for most attack scenarios. Szakály et al. [20] already proposed two possible remedies for the TLS vulnerability: stricter OCSP checking and encoding geo-coordinates into charging station TLS certificates.

OCSP provides a mechanism to revoke compromised certificates, which would limit the window during which an attacker can abuse a stolen charging station certificate. However, OCSP checking is not mandatory for TLS leaf certificates in the plug and charge specification, and vehicles are not required to perform it. Making OCSP verification mandatory for vehicles would ensure that revoked charging station certificates can no longer be used for impersonation, provided that compromises are detected and reported in a timely manner.

Another way to strengthen the TLS trust model would be to encode geo-coordinates into charging station TLS certificates. Vehicles can then check the distance between their last GNSS position and the coordinates in the certificate. When an attacker obtains a compromised certificate, it would only allow impersonation of stations near the compromised station's location, rather than any station worldwide. This mitigation could be carried out as a TLS certificate extension, allowing backwards compatibility with vehicles that do not yet support the verification of geo-coordinates.

7.2 Timing-Based Detection

Distance bounding protocols that detect relay attacks through round-trip time measurements are well established in domains such as NFC and passive keyless entry. Conti et al. propose a similar approach as a countermeasure to EVExchange [6], measuring the latency of rapid challenge-response exchanges over the charging cable to detect the additional delay introduced by a relay. In principle, such timing analysis could also be applied to detect the plug and charge relay presented in this paper. However, production EV and EVSE controllers vary widely in processing power and system load, and the PLC channel over the charging cable is subject to variable latency from electrical noise and cable quality. This makes it difficult to establish reliable timing thresholds that distinguish relay-induced delay from normal variation across the diverse ecosystem of charging hardware.

7.3 Context-Binding the Plug and Charge Signature

The relay attack succeeds because the plug and charge signature covers only the random challenge issued by the charging station. Since the signed data contains no information about

which charging station the vehicle believes it is connected to, the signature is valid at any charging station, enabling the relay.

A direct countermeasure is to bind the signature to the context of the current session by including the charging station's EVSE ID in the signed data. During the TLS handshake, the charging station presents its certificate, which contains the EVSE ID as the certificate subject. The vehicle can extract this identifier and concatenate it with the challenge bytes before computing the signature. On the charging station side, the station concatenates its own EVSE ID with the challenge it issued and verifies the signature against the result.

This defeats the relay attack as follows: When the *victim vehicle* signs the challenge, it includes the EVSE ID of the *fake charging station* as presented during the TLS handshake. When the *attacker vehicle* forwards this signature to the *regular charging station*, the station verifies the signature against its own EVSE ID concatenated with the challenge. Since the two EVSE IDs differ, the signature verification fails and the authentication is rejected. The attacker cannot correct this mismatch without access to the victim's private contract certificate key.

This mitigation can be implemented in a backwards-compatible manner without changing the message format defined in ISO 15118. The challenge issued by the charging station is 16 bytes long. Instead of filling these bytes entirely with random data, an updated charging station embeds its country code and operator identifier (5 bytes) within the challenge, using the remaining 11 bytes for random data. Updated vehicles extract the EVSE ID from the charging station's TLS certificate and verify that the corresponding country and operator bytes in the challenge match; if they do not, the vehicle rejects the session, defeating the relay attack.

This approach is backwards compatible in both directions: vehicles that have not been updated treat the challenge as opaque random data and sign it as before, so they continue to function with updated charging stations. Conversely, updated vehicles connecting to a legacy charging station that issues fully random challenges can detect the absence of the expected EVSE identifier and reject the session. As a result, updated vehicles are protected regardless of whether the charging station has been updated, while legacy vehicles retain full functionality with both legacy and updated infrastructure.

7.4 Replacing Plug and Charge with Out-of-Band Payment

An alternative to fixing the plug and charge mechanism is to bypass it entirely. After establishing a TLS connection with the charging station, the vehicle forwards the EVSE ID to a manufacturer-provided backend, which initiates the charging session via the OCPP `RemoteStartTransaction` command — the same mechanism already used by mobile charging apps today. This eliminates the need for the cross-vendor PKI required by plug and charge, and requires no changes to ISO 15118, OCPP, or the charging station software. Additionally, this approach would greatly reduce complexity, as it removes the need for enrolling contract certificates and designing custom challenge-response procedures for plug and charge.

This approach has two drawbacks: it makes the vehicle manufacturer a gatekeeper for payment, and it requires an internet connection, unlike plug and charge which can operate offline. However, ISO 15118 already gives manufacturers control over which contract certificates are enrolled, and the trend towards connected charging infrastructure — reinforced by regulations such as the EU Alternative Fuels Infrastructure Regulation [2] — makes the offline limitation increasingly marginal. A full evaluation of this alternative payment approach, including its security properties, deployment considerations, and impact on existing roaming agreements, is beyond the scope of this paper and should be explored in more comprehensive studies in future work.

8 Conclusion

In this paper, we presented a novel relay attack against the plug and charge payment mechanism of ISO 15118. The attack exploits the absence of station-identifying information in the plug and charge signature, combined with the inability of vehicles to verify charging station identity through TLS certificates. As shown in our proof-of-concept implementation, these weaknesses make it possible to build a relay device, similar to keyless go relaying, that allows adversaries to charge a vehicle while a different vehicle is billed for the charged energy.

We discussed direct mitigations for the relay vulnerability, including binding the plug and charge signature to the charging station identifier and timing-based detection of relay delays. Additionally, we proposed an alternative out-of-band payment approach that shifts payment handling to the vehicle manufacturer's backend, reusing existing remote start mechanisms. This approach removes the need for the complex cross-vendor PKI required by plug and charge and can be implemented without changes to ISO 15118 or charging station software.

As plug and charge adoption grows, addressing this vulnerability is critical before it becomes widely exploitable.

9 Ethical Considerations

We disclosed the plug and charge relay vulnerability to CharIN, the organization managing the Combined Charging System and the ISO 15118 standard, prior to publication. In addition, we contacted several major charging station manufacturers directly; while they acknowledged the disclosure, they did not consider immediate mitigation necessary on their

side. As plug and charge is not widely deployed today [18,20], the immediate risk to users is limited. Furthermore, performing the attack in practice requires obtaining a valid TLS certificate trusted by the victim vehicle, which requires exploiting and extracting one from a real-world charging station, potentially using a zero-day exploit.

10 Open Science

Our proof-of-concept implementation is publicly available at: https://github.com/securityinmobility/plug-and-charge-relay-poc Apart from the modified charging communication stack, this repository also includes network dumps of successful demonstrations of the plug and charge payment fraud attack. By using the free and open source dissector for charging communication "dsV2Gshark" [7], these files can be viewed using Wireshark [25].

Acknowledgment

This work was created in the research project "Elektromobiles Sicheres Laden" (ESiLa) funded by the Bavarian Ministry of Economic Affairs, Regional Development and Energy under grant DIK0512/01.

References

[1] Open EV Charts. https://open-ev-charts.org/. Accessed: 30 June 2025.

[2] Regulation (EU) 2023/1804 of the European Parliament and of the Council of 13 September 2023 on the deployment of alternative fuels infrastructure, and repealing Directive 2014/94/EU (Text with EEA relevance), September 2023. Legislative Body: CONSIL, EP.

[3] Kaibin Bao, Hristo Valev, Manuela Wagner, and Hartmut Schmeck. A threat analysis of the vehicle-to-grid charging protocol ISO 15118. *Computer Science - Research and Development*, 33(1):3–12, February 2018.

[4] Chargebyte GmbH. Iso 15118 and din 70121 compliant dc charging controller for electric vehicle charging stations (evse) and electric vehicles (ev). https://chargebyte.com/controllers-and-modules/evse-controllers/evacharge-se. Accessed: 2025-03-06.

[5] CISA. Alpitronic Hypercharger EV Charger, May 2024. ICSA-24-130-02, CVE-2024-4622.

[6] Mauro Conti, Denis Donadel, Radha Poovendran, and Federico Turrin. EVExchange: A Relay Attack on Electric Vehicle Charging System. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian D. Jensen, and Weizhi Meng, editors, *Computer Security – ESORICS 2022*, pages 488–508, Cham, 2022. Springer International Publishing.

[7] dSpace Group. Free wireshark plugin to analyze iso15118 v2g communication between electric vehicles (ev) and charging stations (evse). https://github.com/dspace-group/dsV2Gshark. Accessed: 2025-03-06.

[8] Sébastien Dudek. HomePlugAV PLC: Practical attacks and backdooring. *Netw. Anal.*, 2015.

[9] Sébastien Dudek, Jean-Christophe Delaunay, and Vincent Fargues. V2g injector: Whispering to cars and charging units through the power-line. In *Proceedings of the SSTIC (Symposium sur la sécurité des technologies de l'information et des communications)*, pages 1–26, Rennes, France, 2019. SSTIC.

[10] EcoG GmbH. Implementation of the iso 15118 communication protocol (-2, -20, -8). https://github.com/EcoG-io/iso15118. Accessed: 2025-03-06.

[11] Lukas Eder, Jakob Löw, and Hans-Joachim Hof. Charging Communication Sniffing and Man-in-the-Middle Attacks. In *Proceedings of the 16th ACM International Conference on Future and Sustainable Energy Systems*, E-Energy '25, pages 799–804, New York, NY, USA, June 2025. Association for Computing Machinery.

[12] IEC. IEC 61851-1 ed2.0: Electric vehicle conductive charging system - Part 1: General requirements, 2010.

[13] ISO/IEC. ISO/IEC DIS 15118-2: Road vehicles - Vehicle to grid communication interface – Part 2: Network and application protocol requirements, 2012.

[14] ISO/IEC. ISO/IEC DIS 15118-3: Road vehicles - Vehicle to grid communication interface – Part 3: Physical and data link layer requirements, 2012.

[15] ISO/IEC. ISO/IEC DIS 15118-20: Road vehicles - Vehicle to grid communication interface – part 20: 2nd generation network layer and application layer requirements, 2022.

[16] Sebastian Köhler, Richard Baker, Martin Strohmeier, and Ivan Martinovic. Brokenwire: Wireless Disruption of CCS Electric Vehicle Charging. In *Proceedings 2023 Network and Distributed System Security Symposium*, San Diego, CA, USA, 2023. Internet Society.

[17] Jakob Löw, Dominik Bayerl, Kevin Mayer, and Hans-Joachim Hof. DrainDead: Emptying Batteries of Parked Electric Vehicles. In *3rd USENIX Symposium on Vehicle Security and Privacy*, pages 233–241, 2025.

[18] Jakob Löw, Kevin Mayer, and Hans-Joachim Hof. *Fast Charging Communication and Cybersecurity: A Technology Review*. November 2024.

[19] Marc Mültin. ISO 15118 as the Enabler of Vehicle-to-Grid Applications. In *2018 International Conference of Electrical and Electronic Technologies for Automotive*, pages 1–6, July 2018.

[20] Marcell Szakály, Sebastian Köhler, and Ivan Martinovic. Current Affairs: A Security Measurement Study of CCS EV Charging Deployments, February 2025. arXiv:2404.06635 [cs].

[21] Marcell Szakály, Sebastian Köhler, and Ivan Martinovic. Short: PIBuster: Exploiting a Common Misconfiguration in CCS EV Chargers. pages 243–249, 2025.

[22] Marcell Szakály, Martin Strohmeier, Ivan Martinovic, and Sebastian Köhler. DCeption: Real-world Wireless Man-in-the-Middle Attacks Against CCS EV Charging, January 2026. arXiv:2601.15515 [cs].

[23] USCALE. eMobility Charging Payment Study 2026, 2025.

[24] Wilco van Beijnum and Sebastiaan Laro-Tol. Hacking EV charging stations via the charging cable, 2024. Hardwear.io Netherlands 2024, The Hague.

[25] Wireshark Foundation. The world's most popular network protocol analyzer. https://www.wireshark.org/. Accessed: 2025-03-06.

CIAplates: Covert, Intelligent, Adversarial Attack on Automated License Plate Recognition Systems using Digital License Plates

Namo Asavisanu
University of Southern California

Tina Khezresmaeilzadeh
University of Southern California

Rohan Sequeira
University of Southern California

Konstantinos Psounis
University of Southern California

Abstract

Digital license plates are increasingly being introduced as an option for motorists and fleet operators alike. Naturally, turning such an important identification device into a digitally controlled, internet-connected device opens up the possibility for *abuse* of such devices. In this paper, we introduce CIAPLATES, an attack created by combining physical adversarial attack generation and *fast* electrophoretic display (EPD) technology to turn commercially available digital license plates into a covert adversarial attack delivery vector for automated license plate recognition (ALPR) systems, showing that such abuse is indeed possible. We investigate, implement, and evaluate multiple adversarial attack generation methods on physical hardware, both in test bed and real-world conditions, achieving successful attacks against OpenALPR, Plate Recognizer, and Qwen3-VL while using less than 6.2% of the total plate area for perturbations, highlighting the plausibility of real-world attacks. Furthermore, we show that countermeasures to a CIAPLATES attack can be practically implemented, paving the way to further research and implementation of defenses in future systems.

1 Introduction

In recent years, digital license plates (as shown in Figure 1) are increasingly being legalized and introduced as an alternative to traditional metal license plates in multiple U.S. states, including California [31], Texas [62], and Arizona [2]. Acting as both a traditional identification mechanism (as metal license plates have been doing so for centuries) and an IoT device, they introduce features such as detachment sensors (in an attempt to prevent plates from being stolen and then reused) and personalization (within certain limits).

However, being an IoT device that is connected to the cloud, digital license plates are not impervious to common pitfalls of such devices, as demonstrated by the recent cloud privilege escalation exploit [13] which allows an attacker to have full administrative access to all plates (and perform actions such as remotely updating an arbitrary plate), as well as an embedded software exploit [28] allowing for a software swap that then allows for arbitrary content to be displayed instead of the actual license plate information.

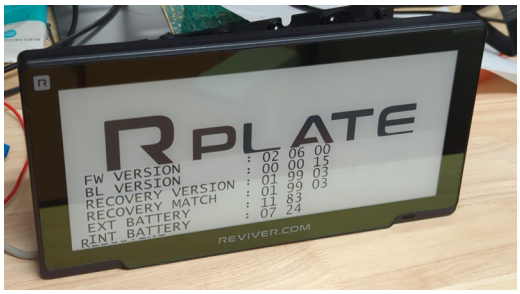


Figure 1: A digital license plate.

Many traffic management applications today (from parking management [22, 23], open road tolling [32, 61], automated traffic enforcement [63, 65], to automated hotlist notifications [7, 36] and more) rely on automated license plate recognition (ALPR) technology to detect and extract data from vehicle license plates, given an input image from a camera system, which can be either stationary or mobile. The general assumption is that license plates tend to be an accurate, visible, and unique identifier of vehicles (as it is legally required to be in most jurisdictions). *Intentional* malicious avoidance of ALPR systems generally requires visible changes to the license plate itself (*i.e.,* defacement), which tends to be easily detectable by observation and at least semi-permanent, resulting in relatively few *in-the-wild* attempts at doing so.

In CIAPLATES, we ask: "Is it possible for a malicious actor to stealthily avoid ALPR systems while maintaining plausible deniability by *abusing* digital license plates?"[1]

To answer this question, we first investigate the hardware behind these digital license plates in detail, identifying the

[1]Our goal is not to provide malicious actors with a turn-key method to evade law enforcement, but rather to demonstrate that such attacks are plausible in the real world in order to raise awareness and foster the development of countermeasures against them, see §6 for more details.

electrophoretic display (EPD) component responsible for their functionality. Then, we adapted an open-source, high-framerate EPD controller to work with the plates' EPD, showing the possibility of running the plate display at a much higher effective refresh rate (allowing for *dynamic* perturbations). Then, we implemented and experimented with multiple adversarial attack generation methods, evaluating them in the real world against 3 different systems: OpenALPR (a popular open-source ALPR system) [43], Plate Recognizer [46] (an off-the-shelf, commercially available ALPR system), and Qwen3-VL [8] (a state-of-the-art vision-language model (VLM) that can perform text recognition tasks that we envision will be integrated into future ALPR systems). Our experiments show that CIAPLATES attacks are effective in our physical test bed (over 82.3% success rate with OpenALPR on 429 license plate samples) and can transfer to other, previously-unseen ALPR systems in both laboratory and outdoor tests, while using less than 6.2% of the plate area for perturbations.

To the best of our knowledge, CIAPLATES is the first physical adversarial attack on ALPR systems delivered through digital license plates. We highlight our contributions:

- We design and implement three *physical adversarial attack* methods to target ALPR systems with digital license plates. First, we adapt a prior character-level heuristic ALPR attack for use with digital license plates (with minor modifications for compatibility). In addition, we introduce two new attacks: a character-level method based on a generative model, and a whole-plate-level heuristic approach. We demonstrate the feasibility of such attacks in different cases, and provide insights on transferability.
- We conduct extensive experiments in the real world with multiple ALPR systems (including a *black-box* commercial system and a modern VLM-based system) plus off-the-shelf digital license plate hardware, showing that CIAPLATES attacks are plausible and effective.
- We provide an open dataset of labeled electronic license plate images (for both *benign* and *perturbed* cases) to foster development of mitigations against CIAPLATES.

We hope that our findings will serve to raise awareness on such attacks and motivate stakeholders who rely on ALPR systems to develop and deploy countermeasures.

2 Background and Related Work

2.1 Automated license plate recognition

Automated license plate recognition (ALPR) or automated number plate recognition (ANPR) systems utilize a combination of computer vision (and sometimes machine learning) technologies [58] to detect and extract data from images of vehicle license plates. Typically (as with OpenALPR), a multi-stage pipeline (such as the one shown in Figure 2) is used. In this model, the plate detection algorithm first detects and extracts a license plate from the input image. Then, a segmentation algorithm detects and separates individual characters from the license plate. Finally, a character recognition system recognizes individual characters, the results of which are combined to create an output of license plate information.

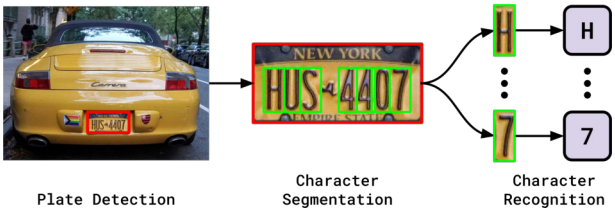


Figure 2: Multi-stage ALPR pipeline.

An alternative, but emerging, approach is a single-stage system (such as [67]), where an end-to-end neural network is trained to directly translate input images into license plate information in a single inference step.

2.2 Attacks on ALPR systems

[35, 64] apply contemporary computer vision attacks (*e.g.,* the fast gradient sign method (FGSM) [25]) to perform subtle perturbations, causing ALPR systems to omit, misclassify, or insert phantom characters into the output. While these attacks are generally successful in *digital* environments, they can be difficult to scale into the *physical* world, as inserting FGSM-like perturbations such that they can be picked up by cameras (with varying placement locations and optical properties) can be very difficult for an attacker who does not have control over the cameras themselves.

[49] uses an optimization (genetic algorithm) based method to create a small perturbation adversarial example on a per-character basis to lead an ALPR system into reading a character incorrectly, showing an attack that may be more realizable from the perspective of an adversary who only has access to the license plate, but not the ALPR camera or pipeline (*i.e.,* by applying surreptitiously placed stickers to their own license plates). This also results in a smaller perturbation which may be easier to overlook, but is still semi-permanent and time-consuming to properly install.

In the real world (and on the much simpler side), some motorists in areas where open road tolling or camera-based traffic enforcement systems are widely deployed have applied many *low-tech* methods in an attempt to disrupt ALPR systems (with varying degrees of plausible deniability and success), from plate covers [34], "stuck" foliage [57], intentionally damaged [3] license plates, mechanical plate flippers [30], and many more than we can possibly cover in this section, despite the fact that such acts are generally illegal in most jurisdic-

tions, highlighting a gap in enforcement.

2.3 Digital license plates

Envisioned since 2001 [39], digital (or electronic) license plates aim to provide additional features, such as visual customization and theft deterrence, by replacing traditional metal license plates with a display coupled to an electronic control system. They are required to conform to certain requirements, such as meeting the same levels of legibility as traditional license plates and being readable by ALPR systems, among other requirements [1, 12].

In order to allow digital license plates to always be able to display the license plate image without requiring a constant source of power (*i.e.,* even when the car is off), current implementations [56] of digital license plates utilize electrophoretic display (EPD) technology (similar to those found in e-book readers and sometimes also called electronic paper displays), which itself is capable of retaining images without a power source. This enables the controller to *sleep* in a low-power mode until an external action (such as update requests by the user) or trigger (such as an anti-tamper switch trip) happens. Commonly, they are also equipped with one or more wireless communication mechanisms (either Bluetooth or Cellular) and a battery (for operation independent of the host vehicle's power), and may also be equipped with a GNSS receiver.

In an attempt to ensure that only the correct license plate image according to the vehicle registration database is shown, it is typical for digital license plates to require an *activation* by the digital license plate manufacturer, ostensibly including manual verification of registration documents, before an activation message is *pushed* from the manufacturer to the associated plate over a wireless connection, enabling it to display the plate in which the owner is entitled to [55].

To date, there has been no publicly available documentation on the inner workings of these devices. We acquired two commercially available digital license plates and provide more information on them in Appendix A.

As recent work [28] has shown that an attack to bypass the readout and overwrite prevention mechanisms on Reviver digital license plates is possible (allowing an attacker to arbitrarily display any content on the plate's display), in this work we direct our focus into "could we stage an attack that exploits the unique capabilities of EPD-powered digital license plates to adversarially attack ALPR systems?"

2.4 High refresh rate electrophoretic displays

While the EPD technology has been historically associated with slow refresh rates, recent improvements in display technology as well as research has shown that it is possible to drive these displays at a much faster *apparent* refresh rate by driving each pixel individually in a pipelined fashion, instead of updating the entire screen. This has led to the commercial introduction of EPD-based computer monitors [10, 16], as well as open hardware efforts [38].

3 Threat Model

We aim to realize a proof-of-concept physical adversarial attack on ALPR systems delivered through the vector of EPD-powered digital license plates to show the plausibility of such attacks in the real world, focusing on achieving misrecognition of plates with minimal, plausibly deniable perturbations, delivered with precise timing, for maximum *stealthiness*.

In this work, we primarily use OpenALPR [43] as a proxy for real-world ALPR systems, as it is a popular open-source, feature-complete ALPR system (with a commercial derivative [53] utilized by many governmental agencies as well as police departments [54]).

To further explore attack transferability to other and future ALPR systems, we use Plate Recognizer [46], a commercial ALPR software that has also seen real-world deployments [47], and Qwen3-VL [8], a modern, open-weight VLM that performs well on text recognition tasks.

We do not perform any training or attack optimization that is specific to either Plate Recognizer or Qwen3-VL, which we use as proxies for a black-box, *unseen* ALPR system that may be deployed in the real world.

3.1 Attack scenario

In this attack, we assume a *fixed-installation, single-camera* ALPR deployment, where a camera connected to an ALPR system is deployed on a stationary mount (*e.g.,* traffic signal pole or toll gantries), pointed to the rear license plate of passing vehicles. We consider an individual adversary that exploits digital license plates to maliciously interfere with ALPR systems, for purposes such as toll/ticket avoidance or interference with ALPR-based surveillance systems (such as [21]), on a fully-automated system where most, if not all, license plate images are automatically processed on a massive scale (rendering human-based feedback generally infeasible such that it is only reserved for extreme edge-case scenarios).

3.2 Adversary's objective

In this attack, an adversary intends to modify their own digital license plate to briefly display a perturbed version of their own license plate (we call this an *obfuscated* license plate), to mislead ALPR systems into reading the wrong license plate information. The attack should be *covert* (*i.e.,* not easily detectable by automated systems as suspicious), not simply be the act of replacing one valid license plate image with another (as doing so could introduce large visual changes which impacts visual stealthiness as well as plausible deniability), and not result in ALPR systems not detecting any license plate at all (as many ALPR systems today can flag vehicles

with missing or unreadable license plates for human review or action [41], which can result in loss of covertness as human reviewers are more likely to be able to spot obfuscated plates).

3.3 Adversary's capability

We assume that the adversary can only influence the system by perturbing their own digital license plate and not by interfering with the ALPR cameras or systems themselves. We also assume that the adversary either has prior knowledge of ALPR camera locations (which can be obtained by manual mapping, or from crowd-sourced databases such as [44]), or is able to detect upcoming ALPR cameras on the road and approximate the timing when they will pass within the camera's view (we discuss this more in §4). In addition, we assume that the adversary has the ability to either modify the software on the digital license plate or swap its controller out to allow for arbitrary content display (similar to what we have done).

4 The CIAplates Attack

We envision an attack as visualized in Figure 3, where the attacker pre-generates their obfuscated plate (possibly even multiple ones) in advance based on their actual license plate information. Then, as they pass in front of an ALPR camera's field of view (red dashed *attack window* as shown), the attack is triggered by quickly switching the displayed image on the attacker's digital license plate from the original plate to the obfuscated one, then quickly switching back again as soon as they leave the calculated camera field of view, in order to increase the stealthiness of the attack.

For use cases such as open-road tolling or automated traffic enforcement where the traffic is flowing, this attack window can be anywhere from 0.39 to 1.57 seconds (when assuming a license plate recognition range from 7 to 20 meters [5, 6] and a speed of 30 to 120 km/h).

By coupling the short attack window (small *exposure* time) with small-perturbation-area adversarial attack generation methods, a CIAPLATES attack is more likely to go by unnoticed by observers (especially at higher traffic speeds). We also note that there is no hard triggering time requirement for a CIAPLATES attack: as long as the attacker can trigger the attack before the plate comes into view of an ALPR camera, the attack will work (although, as discussed earlier, the degree of stealthiness depends on how precise this triggering can be, to limit the exposure time).

In this section, we discuss the components that are combined to create the CIAPLATES attack, from hardware to attack generation methods.

4.1 Hardware

To facilitate our study, we adapted an open-source, FPGA-based electrophoretic display controller [68] to directly drive



Figure 3: CIAPLATES attack visualization (note that the obfuscated plate is only shown within the attack window, where it is visible to an ALPR camera, denoted by dashes).



Figure 4: Custom content displayed directly onto the EPD panel of a digital license plate via an external controller.

the E-Ink ED113TC1 [17] display panel (which is still attached to the donor electronic license plate's glass cover and frame, to maintain the exact same optical properties as the commercially available product), bypassing the electronic license plate's circuit board and allowing us to drive the EPD panel directly (in a higher refresh rate, 2 bits per pixel mode). Figure 4 shows this setup in a laboratory setting.

We performed real-world testing (more details in §A.2) that showed the EPD panel's effective full-refresh rate to be at $5Hz$. This led us to focus on slower attacks that can maintain relative stealthiness while staying within physical limitations of the hardware (recall that CIAPLATES only needs two refreshes, at worst adding 0.4 seconds to the exposure time at $5Hz$), rather than faster attacks that exploit camera frame rates (*e.g.,* intentionally introducing flickering visible to cameras but not to the human eye).

4.2 Attack generation methods

In CIAPLATES, we consider and compare multiple *adversarial plate* generation methods that can be grouped by how the perturbation is created and incorporated into the final obfuscated plate:

- **At the character level:** a perturbation is created for each individual character, resulting in a repository of perturbed characters that can later be used to assemble

adversarial, *obfuscated* license plate images. We do this in §4.2.1 and §4.2.2.

- **At the plate level:** a perturbation is created on an image of the whole license plate in a single step, creating an *obfuscated* plate with perturbation features that may span multiple characters simultaneously. We do this in §4.2.3.

We note that each method has differing benefits. For example, character-level attacks would need to be performed only once (and then can be used to create multiple obfuscated plates), allowing for careful attack crafting (*e.g.,* placing perturbed characters in spots that may be more easily overlooked). However, plate-level attacks create an obfuscated plate that more closely resembles technical glitches in the display, providing better stealthiness and plausible deniability for the attacker. We focus on *subtractive* attacks, which we have experimentally found to be more stable and results in an attack with lower perturbation area that improves stealthiness.

4.2.1 Character-level heuristic attack

Following prior work in adversarial machine learning applied to license plates [49] (with a small modification to restrict the final mask output to binary to work better with our EPD delivery mechanism) and general ideas in [60], we experimented with an attack generation pipeline as in Figure 5. First, we digitally create a ground truth dataset of license plate characters. Then, using the differential evolution algorithm [59], we attempt to create a *mask generator* system that creates perturbations on those characters while minimizing the penalty:

$$P_{CLH} = (\alpha \cdot \frac{A_{perturb}}{A_{char}}) + (\beta \cdot P_{attack}).$$

The penalty represents a multi-objective function with two terms. The first term (weighted by α) is the *perturbation penalty* which aims to keep the perturbation of the original license plate as small as possible for stealthiness reasons. The second term (weighted by β) is the *attack penalty* which aims to maximize the chances that the attack will be successful by penalizing the system when it is not. By empirical evaluation, we used $\alpha = 1$ and $\beta = 2$, following the intuition that the attack success rate is more important than minimizing perturbations.

We define the *perturbation penalty* as the area of the perturbations over the total area and calculate the *attack penalty* based on output data from Tesseract OCR (the OCR engine as used by OpenALPR) as follows:

$$P_{attack} = \begin{cases} 1.0 & \text{if } \textit{no detection} \\ 0.5 - (\frac{C_{GT} - C_{PC}}{2})) & \text{if } \textit{same character} \\ -C_{PC} & \text{if } \textit{different character} \end{cases},$$

where C_{GT} is the normalized value (*i.e.,* between 0 to 1) of OCR *confidence score* for the ground truth character (*i.e.,* non-perturbed character), and C_{PC} is similarly the normalized value of OCR confidence score for the generated perturbed character. The first (no character detection) and second (same character detection) cases represent a failed attack, hence the penalty is positive. Note that in the second case, if the attack manages to reduce the confidence score of the character recognition result, then we reduce the penalty accordingly. The third case represents a successful attack, hence a negative penalty, which is directly proportional to the confidence score of the OCR engine in recognizing a different character.

The general intuition behind these cases is that we wanted to incentivize the mask generator to create perturbations that will lead to misclassification of the character but not to result in "no detection", following earlier discussions in §3.2.

Finally, armed with a collection of perturbed characters, the attacker can then use them as characters to create their obfuscated license plates (picking and choosing which characters they would like to perturb, based on the actual characters available on their own license plate). This creates an obfuscated license plate similar to Figure 6.

4.2.2 Character-level generative model attack

We also experimented on a method to create adversarial license plates that loosely borrows from the idea of a generative adversarial network [26], as shown in Figure 7. In this method, we train two neural networks: a *mask generator* model which attempts to learn how to generate bit masks that conceals individual characters on the license plate, and a *classifier* model that learns to classify individual characters and acts as our adversary (that gets better at detecting obfuscated characters over time, such that the *mask generator* will also learn to generate better masks that are better obfuscators).

We designed our *mask generator* (generative) model (as shown in Figure 8) taking inspiration from the generator portion of the deep-convolutional GAN (DCGAN) [50] method. Our *mask generator* takes as input a concatenation of a Gaussian-sampled noise vector and a one-hot encoded character class vector, allowing the generator to learn and create different masks for different characters. We used a slightly shallower design to improve training stability and mitigate overfitting, given the limited resolution of the generated masks (64×64) and the relatively small training dataset [9, 11].

We also utilized leaky ReLU activations instead of regular ReLU utilized in DCGAN generators, to improve gradient flow and address the issue of dying neurons [37, 66] in a shallower architecture, by allowing small gradients for negative inputs, enabling the network to utilize all neurons during training (particularly advantageous in adversarial training, where the generator's stability depends on consistent gradient dynamics [50]). Additionally, the non-saturating nature of leaky ReLU helps reduce overfitting (a concern when training on smaller datasets) by maintaining gradient flow across the entire input range, enabling better generalization of models

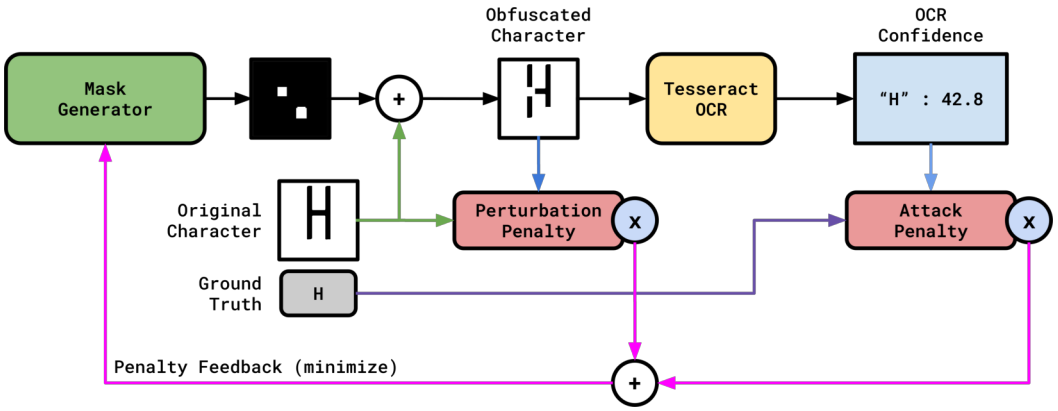


Figure 5: Character-level heuristic attack pipeline.



Figure 6: Example attack generated by the character-level heuristic method (applied on the letters *I* and *V*).

to previously unseen data [51].

We empirically found that our shallower generator design produces stronger adversarial perturbations when compared to the original DCGAN generator, effectively disrupting classifier/detector performance.

In order to validate this, we attempted inference of perturbed characters (after 2,200 epochs of generator training) using a trained, but not *fine-tuned* ResNet50 model to measure classification accuracy. We found that our generator model design successfully misled the classifier into having an accuracy as low as 2.70%, significantly lower than that of the original DCGAN generator design (10.81%).

Moving from the mask generator module to the *character classifier* module of the generative model, we utilized ResNet50 [29] for this module (as residual networks mitigate the vanishing gradient problem, helping to increase stability especially when their results are used to train a generator). We initially bootstrap the classifier by training it on sample images created from 1,313 freely available fonts[2] from Google Fonts [27] as well as DaFont [14], rendered into individual 64×64 pixel images and sorted by class (*i.e.,* character). A total of 37 character classes are considered: the numbers 0 to 9, uppercase Latin alphabets A to Z, and the *not a character* class containing random non-character glyphs not within the classes we have considered earlier (the small number of classes, as well as input size, motivated us to choose ResNet50 as our classifier, to achieve good performance while mitigating over-fitting). The classifier model is then trained using an 80/10/10 split for training, validation, and testing respectively, until it achieves an acceptable performance (accuracy ≥ 0.94 on both validation and test sets).

We set up our training cycle to use the *Adam* optimizer [33] with a learning rate of 1×10^{-5} and to fine-tune the character classifier only once per 10 epochs of training for the mask generator, to help stabilize the training process and prevent the classifier from over-fitting. The classifier is trained using categorical cross entropy (similar to the bootstrapping process, however with perturbed characters also mixed in such that it learns to better recognize perturbed characters over time). For the generator, we used a combined loss term:

$$L_{MG} = \lambda_1 \cdot L_{\text{attack}} + \lambda_2 \cdot L_{\text{invalid}} + \lambda_3 \cdot L_{\text{stealth}},$$

where we define the weights $\lambda_1, \lambda_2,$ and λ_3 as hyperparameters of the training process (fine-tuned through random search). We discuss the three terms of the loss function below.

To incorporate the idea that our mask generator should create *masks* that result in a misclassification of the character, we utilized the inversion of categorical cross-entropy for our *attack loss* (in essence, trying to maximize the categorical cross-entropy loss instead of minimizing it):

$$L_{\text{attack}} = -H(Y, \hat{Y}),$$

where $Y \in \mathbb{R}^{B \times C}$ represents the ground truth labels, $\hat{Y} \in \mathbb{R}^{B \times C}$ represents the detector's predicted softmax scores, B is the batch size, $C = 37$ is the number of classes (0-9, A-Z, and *not a character*), and $H(Y, \hat{Y})$ is the categorical cross-entropy between the ground truth labels (Y) and the predicted softmax scores ($\hat{Y}$). Specifically, the cross-entropy is calculated as follows:

$$H(Y, \hat{Y}) = -\frac{1}{B} \sum_{i=1}^{B} \sum_{j=1}^{C} Y_{i,j} \log(\hat{Y}_{i,j}),$$

[2] Fonts listed as "100% free" or licensed under the Open Font License.

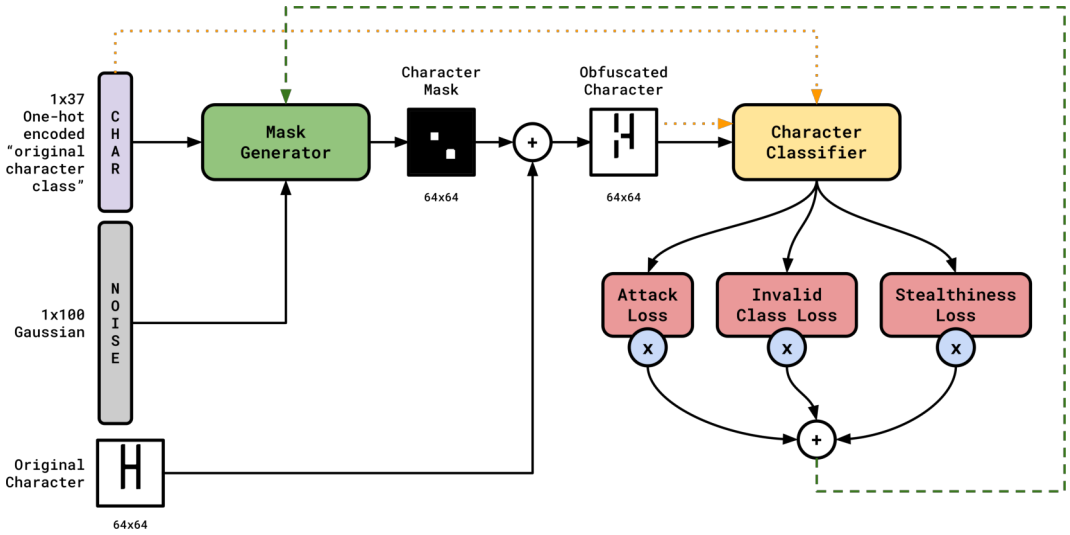


Figure 7: Character-level generative model attack generation architecture.

where $Y_{i,j}$ and $\hat{Y}_{i,j}$ represent the ground truth class j and the predicted softmax score j in batch i respectively.

In order to prevent the generator from creating masks that cause *no recognition* results, we add the *invalid class loss* which penalizes the generator if it assigns a high softmax score for the *not a character* class. Binary cross-entropy is used for this purpose to measure the difference between the target (0 score) and the predicted softmax scores for the *not a character* class, guiding the generator to minimize this undesired outcome. The *invalid class loss* is defined as follows:

$$L_{\text{invalid}} = H(\mathbf{0}, \hat{Y}_{\text{no char}}),$$

where $\mathbf{0} \in \mathbb{R}^B = (0, 0, \ldots, 0)$ is a zero vector and $\hat{Y}_{\text{no char}} \in \mathbb{R}^B$ represents the *not a character* class vector in the perturbed case.

Finally, to ensure the obfuscated image remains *stealthy*, we use the mean squared error (MSE) metric between the flattened *perturbed* character image and the flattened *original* character image, following the idea that the smaller this loss, the less the perturbation and thus the stealthier the attack:

$$L_{\text{stealth}} = \frac{1}{B} \sum_{i=1}^{B} \sum_{j=1}^{N} \left(X_{i,j} - \hat{X}_{i,j} \right)^2,$$

where B is the batch size, N is the number of pixels, and $X_{i,j}$ and $\hat{X}_{i,j}$ represent the original and perturbed pixel j in batch i respectively.

This method generates perturbations as visualized in Figure 9. While at first glance this may look like a simple character replacement, the perturbed character retains the general outline of the original character (and in many cases, parts of the original character), contributing to a lower number of pixel changes during attack, making the attack more difficult to notice especially in a very short time span (as utilized by CIAPLATES).

Similarly to the character-level heuristic method we discussed earlier, the attacker can then use this collection of perturbed characters to create obfuscated license plates, similar to an example in Figure 10.

4.2.3 Plate-level attack generation

For the plate-level attack, we expand on the ideas earlier presented in §4.2.1 and create an attack architecture as shown in Figure 11. Instead of working on single-character images as an input, here we instead use an image of the entire original license plate. To have a detector that works well on an entire license plate, we use OpenALPR (instead of Tesseract OCR) to evaluate the resulting obfuscated plate. With these changes, we also change the type of perturbation our mask generator will create (full-height or full-width lines to create a plate-spanning attack that resembles a line of dead pixels on the display for better plausible deniability). Note that while many ALPR engines (including OpenALPR) can return multiple license plate candidates, naturally, one would select among the candidate plates the one with the highest confidence to be the *final recognition result*. The attack penalty calculation is then done in the same way as P_{attack} in §4.2.1.

The general intuition behind this is to try to create perturbations that best *hides* the correct plate information, but still leave the license plate detectable and recognized as a different one (with a high confidence), again following earlier discussions in §3.2.

We note that this attack generation method does assume that the attacker is able to extract a list of license plate candidates and confidence scores from an ALPR system used in attack generation. In real-world cases where this is not possible (*e.g.,*

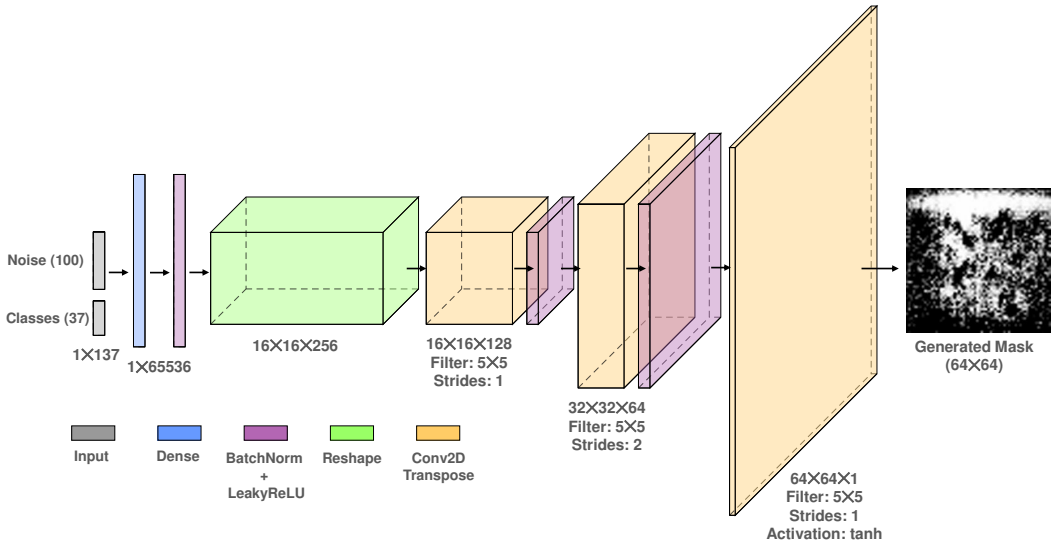


Figure 8: Architecture of the mask generator model inspired by the generator model of DCGAN [50], with a modified input layer combining noise and a one-hot encoded class vector for character-specific mask generation.

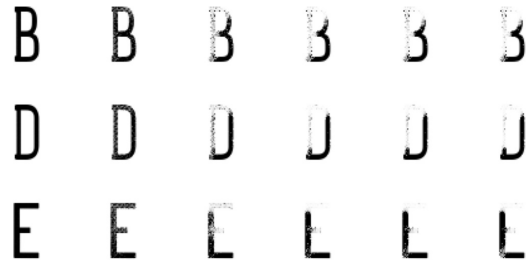


Figure 9: Character erosions from the generative method. Notice as the training goes on (from left to right), the characters get more perturbed.



Figure 10: Example attack generated by the generative method (applied on the letters *B* and *U*).

an unknown ALPR system) or only partially possible (*e.g.,* a cloud-based ALPR system with limited query allowance and rate limits), the attacker may be able to utilize a surrogate ALPR system or a black-box transfer attack (*e.g.,* [45]) to fill in this gap instead.

With the changes aforementioned, the *mask generator* can now generate a mask that can be used to create an obfuscated plate similar to Figure 12.

5 Experimental Evaluation

In order to show that a CIAPLATES attack can be performed in a physical, real-world setting, we created two physical test scenarios (with the pipeline in Figure 13): a *laboratory-based* test with controlled lighting conditions and a *real-world* outdoor test in less-than-ideal conditions.

For both tests, we used the actual digital license plate hardware (as described in §4.1) and Allied Vision Prosilica GE1910C[3] cameras with an 8 mm $f/1.4$ lens plus a circular polarizer, configured for grayscale images in *Mono8* mode.

For our test license plates, we curated a list of vanity license plate texts from [19], which we then use to create *original* license plate images (*i.e.,* benign examples with no perturbation). We only proceed with our experiments for a particular license plate text if its *benign* example can be correctly recognized with OpenALPR when physically displayed.

For plates that pass the benign recognition step, we apply our attacks earlier described in §4.2.1, §4.2.2, and §4.2.3 to create *obfuscated* plates from the original digital image (with

[3] A global shutter camera that approximates deployed ALPR hardware.

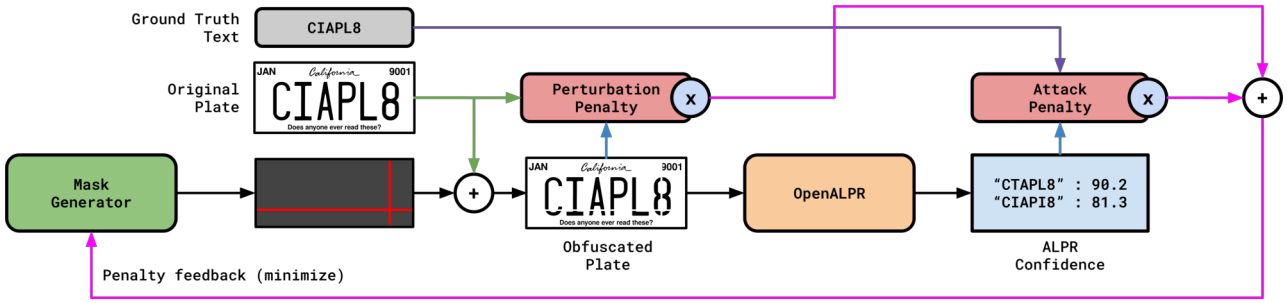


Figure 11: Plate-level heuristic attack generation architecture.



Figure 12: Example attack generated by the plate-level heuristic method (applied on the entire plate as two stripes).

up to two randomly selected characters being obfuscated for character-level methods and two perturbation features for plate-level attacks, in order to limit total perturbation). These obfuscated plates are then displayed on the digital license plate screen and recognized through ALPR again, the results of which are then compared with ground truth data.

Then, we take a subset of successful attacks from the laboratory setting and perform an extended, multi-camera transfer study, to further shed light into how the attack works across multiple ALPR systems, in different lighting conditions, and different camera setups.

Note that during our experiments, we optimized our attacks only on OpenALPR, as a surrogate system that an attacker would have easy access to. In order to evaluate transferability to *previously-unseen, black box* ALPR systems, we additionally use Plate Recognizer [46], a commercially available ALPR solution, as well as Qwen3-VL [8][4], a vision-language model (VLM) based OCR system (configured to read license plates, representing future VLM-based ALPR systems).

Finally, using the laboratory test setup, we also tested the EPD panel's response time (more details in Appendix A) to validate the prerequisite to our attack design (which involves only briefly showing the obfuscated plates for a fraction of a second, based on ALPR location triggering, to maximize stealthiness) and show that this is possible on actual hardware.

[4]As Qwen3-VL is not a full-featured ALPR solution, we only test it for text extraction and not license plate detection.

5.1 Laboratory experiments

In the laboratory-based setup as shown in Figure 14, we tested our attack generation methods with a dataset of 429 ALPR recognizable digital license plates that we have created following actual license plate layout and numbering guidelines.

We categorize experimental outcomes into three groups - *successful*, *no change to results*, and *no plate recognized*. The *successful* attack group represents an attack that causes a change in ALPR recognition results without causing a recognition failure (we discuss more about these changes in Appendix B). The other two groups - *no change to results* and *no plate recognized* represents attack failure modes, where the attack either does not cause any change to ALPR results and the correct plate is still recognized (*i.e.,* the attack failed to work entirely) or caused the ALPR system to not detect a license plate at all (resulting in loss of stealthiness as images are likely to be sent to manual review), respectively. Our findings, as reported in Table 1 show that all methods we implemented are able to generate physical, obfuscated license plates that are effective against OpenALPR with a success rate of over 82.3%, and that the character-level generative attack generation method is highly effective at transferring to Plate Recognizer, with a success rate of over 95.3%.

For *successful* attacks, we also compared the digital images of the original as well as perturbed plates to measure the perturbation area used by each attack generation method (one potential stealthiness factor) and found that all methods use less than 6.2% of the plate area for perturbations.

5.2 Attack transfer experiments

To further demonstrate CIAPLATES in more realistic conditions that are closer to actual ALPR deployments, we performed additional transfer experiments with multiple camera angles, in a more realistic outdoor setting.

As it is *likely* illegal to do so on public roads, we only perform these tests in a private parking lot.

We set up 3 test cameras in different positions as shown in Figure 15: down-looking, high pole mounted, and low pole mounted (representing example usages in open-road tolling, street surveillance, and access control respectively) follow-

Attack generation method	Average plate perturbation area %	Evaluator	Attack success %	Attack failure %		Benign unreadable %
				No change to results	No plate recognized	
Character-level heuristic	3.33	OpenALPR	84.97	11.42	3.61	0.00 (baseline)
		Plate Recognizer	20.98	74.59	0.00	4.43
		Qwen3-VL (4b)	17.48	69.00	*N/A*	13.52
Character-level generative	6.19	OpenALPR	83.10	0.00	16.90	0.00 (baseline)
		Plate Recognizer	95.34	0.23	0.00	4.43
		Qwen3-VL (4b)	86.01	0.47	*N/A*	13.52
Plate-level heuristic	0.64	OpenALPR	82.30	5.62	12.08	0.00 (baseline)
		Plate Recognizer	12.00	83.57	0.00	4.43
		Qwen3-VL (4b)	13.99	72.49	*N/A*	13.52

Table 1: Laboratory experiment results on 429 license plate samples.

Table 2: Transfer experiment results for varying camera setups.

Camera setup	Attack type	ALPR evaluator	Success %	Failure %		Number of samples tested
				No change to results	No plate recognized	
Overhead toll camera	Character-level heuristic	OpenALPR	92.00	8.00	0.00	200
		Plate Recognizer	24.00	76.00	0.00	200
		Qwen3-VL (4b)	15.20	84.80	*N/A*	125
	Character-level generative	OpenALPR	97.50	0.00	2.50	200
		Plate Recognizer	100.00	0.00	0.00	200
		Qwen3-VL (4b)	99.00	1.00	*N/A*	100
	Plate-level heuristic	OpenALPR	75.50	16.00	8.50	200
		Plate Recognizer	9.50	90.60	0.00	200
		Qwen3-VL (4b)	7.59	92.41	*N/A*	158
Pole-mounted surveillance camera	Character-level heuristic	OpenALPR	87.04	10.18	2.78	108
		Plate Recognizer	22.00	78.00	0.00	200
		Qwen3-VL (4b)	17.59	82.41	*N/A*	125
	Character-level generative	OpenALPR	92.73	0.00	7.27	110
		Plate Recognizer	100.00	0.00	0.00	200
		Qwen3-VL (4b)	100.00	0.00	*N/A*	99
	Plate-level heuristic	OpenALPR	61.87	22.88	15.25	118
		Plate Recognizer	6.50	93.50	0.00	200
		Qwen3-VL (4b)	10.18	89.82	*N/A*	167
Parking lot access control camera	Character-level heuristic	OpenALPR	80.80	11.20	8.00	125
		Plate Recognizer	19.00	81.00	0.00	200
		Qwen3-VL (4b)	16.00	84.00	*N/A*	131
	Character-level generative	OpenALPR	88.19	0.00	11.81	127
		Plate Recognizer	100.00	0.00	0.00	200
		Qwen3-VL (4b)	100.00	0.00	*N/A*	108
	Plate-level heuristic	OpenALPR	56.59	25.20	18.11	127
		Plate Recognizer	6.50	93.50	0.00	200
		Qwen3-VL (4b)	5.98	94.02	*N/A*	167

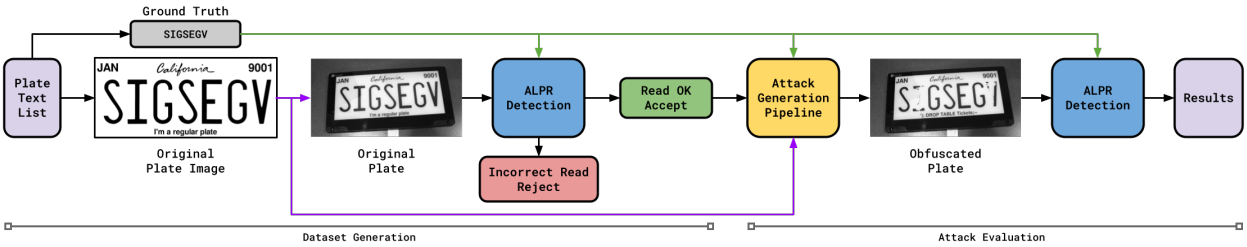


Figure 13: CIAPLATES evaluation pipeline.



Figure 14: *Test stand* (laboratory) experiment setup.

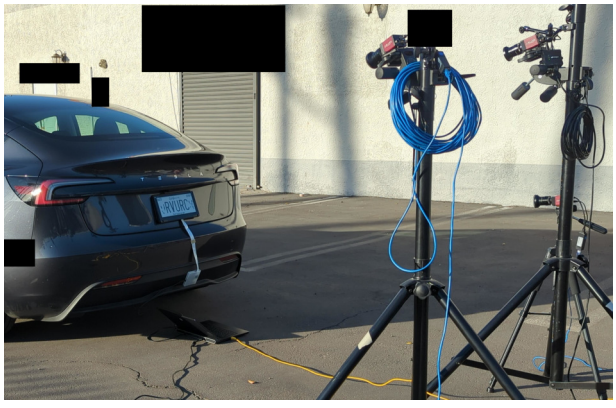


Figure 15: Real-world outdoor experiment setup.

ing best practices in [5, 6]. Then, we installed our modified digital license plate to the rear license plate holder of the vehicle, selected 100 best obfuscated plates for each attack generation method (based on highest-confidence incorrect recognitions from §5.1), captured images of each sample under multiple lighting conditions to create a combined dataset (to ensure that we account for realistic lighting conditions[5]), and then validated the attacks by running the images through OpenALPR, Plate Recognizer, and Qwen3-VL (with 4 billion parameters).

We note that due to experimental constraints, this real-world transfer evaluation is performed on a smaller, selected dataset, thus the transfer success rate should not be directly compared to the overall, end-to-end *attack success rate* from the laboratory experiment. Additionally, we reiterate that in this evaluation, both Plate Recognizer and Qwen3-VL is only used to evaluate the attacks and not at any point during attack generation, representing *black-box* target systems that the attacker did not optimize for in advance.

Since ALPR cameras typically are global shutter cameras [4, 24] with high shutter speed, motion blur from vehicle movement is generally not a problem in real-world conditions from stationary to highway speeds, yielding the same results whether the vehicle is moving or not[6].

We then gather results for each camera setup and filtered out those where the *benign* example cannot be correctly read on each combination of specific camera setup, lighting condition, and ALPR engine. Our results in Table 2 show that CIAPLATES can be effective against OpenALPR even against multiple camera angles and in real outdoor conditions and that the character-level generative model method exhibits the best performance and transferability.

We note that the character-level generative model method is remarkably successful in transferring to different ALPR engines (as shown in both the laboratory analysis in §5.1 and this section), while heuristic based methods may be prone to over-fitting to a particular ALPR system used in attack training resulting in poor performance against other black-

[5] We do not cover different weather conditions (*e.g.,* rain, snow, or fog), but such conditions, if it does not cause recognition failure, would be unlikely to shift attack trends, as the distortion effects are expected be largely uniform.

[6] Some ALPR hardware vendors, such as [52], advertise the capability to capture clear images of license plates from vehicles up to 320 km/h.



Figure 16: Examples of different perturbation area allowances through the generative model method for the character **8** (actual perturbation area of 18, 38, 46, 58, and 77% respectively).

box ALPR systems, suggesting that future research efforts could focus more on generative methods to create stronger attacks that would then feed into the development of effective countermeasures against a more sophisticated adversary.

5.3 Perturbation area requirements for the generative model attack

In order to further explore the *stealthiness* properties of generative model attacks, we further ask the question of "what is the minimum amount of perturbation needed for a specific character, in order for the attack to work?". To answer this question, we added a maximum perturbation area cap as an additional restriction to the *mask generator* model (discussed earlier in §4.2.2), resulting in perturbed characters with different perturbation levels as shown in Figure 16. We then performed further laboratory evaluations focused on single-character modifications, using *Qwen3-VL* as the evaluator. We observed an average perturbation area requirement at 39.41%, with some characters requiring much less perturbation than others (down to 20%), as shown in Figure 17.

6 Discussion

6.1 Attack practicality

While we reveal the threat of physical adversarial attacks on ALPR systems through the use of stealthy and potentially *plausibly deniable* digital license plate attacks, we also note some limitations in the practicality of a CIAPLATES attack as currently presented. First, to maximize stealthiness, the attacker would need to be able to *trigger* the attack by knowing the (approximate) locations (and therefore camera field-of-views) at a precision level that minimizes the attack exposure window. This would assume a fixed, dispersed installation of ALPR cameras (*e.g.,* open-road tolling systems). In cases where the exposure window is extended, either due to pervasive deployment of ALPR cameras, *moving-vehicle* ALPR installations (*e.g.,* [7]), or use cases where the vehicle speed is very low (*e.g.,* parking management), the stealthiness benefits of CIAPLATES may be negated.

Another challenge that remains open is the transferability of such attacks to multiple, closed-source ALPR systems in the real world that the attacker has no access to (*e.g.,* systems deployed by governmental agencies). While we have demonstrated that this is in fact possible, further work is still required to increase attack robustness with different ALPR systems (especially with systems that are more reliant on neural networks, which as we observed may be more vulnerable to certain attack generation methods than others).

Nevertheless, we believe the practical limitations discussed above does not diminish the value of this work. **Our goal is not to provide malicious actors with a turn-key method to evade law enforcement**, but rather to demonstrate that attacks such as CIAPLATES are plausible in the real world, raise awareness, and motivate stakeholders who rely on ALPR systems to develop and deploy effective countermeasures.

6.2 Countermeasures

A CIAPLATES attack is designed to perturb characters (or parts of them) on a license plate. We believe that there are multiple countermeasures that can be applied as a mitigation to our attack:

Stronger character recognition and adversarial training: in a multi-stage ALPR system, a more robust optical character recognition (OCR) system can be used. This can be achieved through the application of adversarial training techniques. Similarly, adversarial training can be used on the entire pipeline to increase the robustness of single-stage, neural network based ALPR systems.

Anomaly detection: adding another step to the ALPR pipeline, an *anomaly detection* system which spots anomalous license plates or individual characters can be implemented to flag license plates for manual inspection, raising the probability that adversarial attacks will be discovered.

As an initial proof of concept, we implemented an anomaly detection system using two state-of-the-art VLMs: GPT-5-mini [42] and Qwen3-VL [8], targeting our generative model attacks. Both VLMs exhibited reasonable malicious plate detection rates on our transfer dataset over all lighting conditions (79.5% with GPT-5-mini and 76.5% with Qwen3-VL) with less than 0.02% false positives (where a benign plate is falsely flagged as suspicious).

While VLMs are indeed compute-intensive (therefore posing a challenge in embedded ALPR systems), we believe that with enough training data, conventional machine learning models can also be trained to efficiently spot suspicious license plates, enabling multi-stage detection setups where the majority of suspicious license plates can be detected on-device, followed by VLM analysis for challenging samples.

We note that the VLM detector does not perform very well on heuristic based attacks, yielding a detection rate of only 21.5%. However, given that we observe significantly lower attack success and transfer rates with heuristics based attack on contemporary ALPR systems, we believe further adversarial training of the OCR component or neural network could be utilized to effectively defend against heuristic attacks earlier presented, at minimal extra runtime cost.

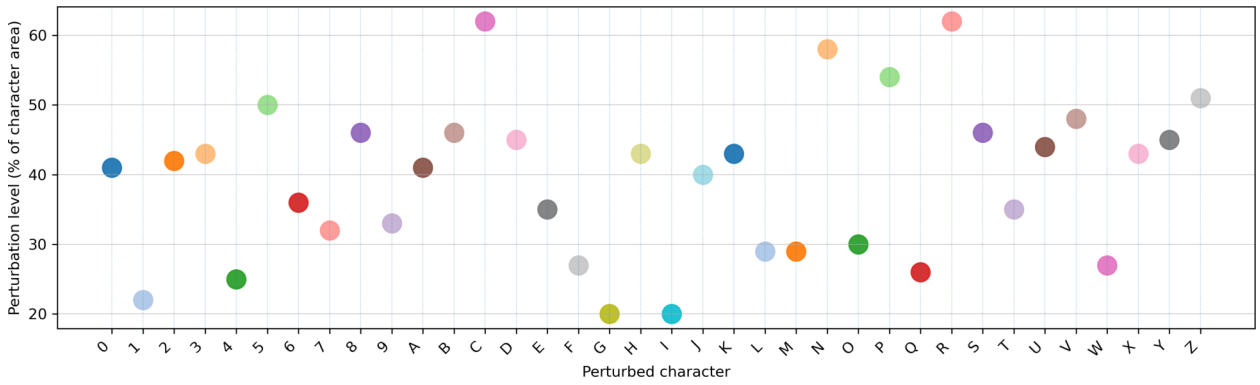


Figure 17: Minimum effective perturbation area requirement for generative model attacks.

In a simpler way, rules-based systems based on computer vision technologies such as gap detection (detecting unexplained or illegal gaps between characters, such as two consecutive spaces when not allowed by the jurisdiction) or character size rules (detecting whether the bounding box of a character conforms to an expected shape, size, or area) can also be used.

Tamper-evident hardware and secure attestation: anti-tamper physical measures, from security seals to tamper detection switches, when coupled with proper device security techniques spanning both software and hardware levels (*e.g.,* signed system software with publicly verifiable signatures, trusted execution environments, and integrity attestation mechanisms) can provide additional, easily field-verifiable indicators of whether a digital license plate has been tampered with or not [7] to increase the difficulty of physical tampering, and deterrence to would-be attackers.

While none of the methods provided above can guarantee a completely successful defense alone (for example, anomaly detection systems can be subject to a cat-and-mouse iterative cycle where when the defense gets better, the attack also gets better at avoiding those defenses), they can be layered to provide *defense-in-depth*, raising the bar for attacks such as CIAPLATES to reduce its impact in the real world.

We also observed that some defense mechanisms, such as JPEG compression [15], does not work very well against a CIAPLATES attack. As a defense that can be implemented without modifying the ALPR engine itself, we have experimented with JPEG compression (down to 10% quality) and found no significant decrease in attack success rates.

We finally note that further detailed studies of defense mechanisms and their effectiveness would be beyond the scope of this paper, as they would likely depend on very specific implementations for different ALPR engines (potentially requiring source code level modification, which is not always possible from an academic perspective, especially with commercial ALPR engines).

[7] In our experience with commercially available digital license plates, we note that they can easily be dismantled with no apparent tamper-evident or resistance features. For example, we accessed the main circuit board of our battery-powered digital license plate by simply lifting a thin plastic cover held on by light adhesives, and epoxied parts inside can easily be removed by a heat gun without damaging components.

7 Use of Generative AI

In §5.2, we used a VLM as one of the target ALPR systems for evaluation, to better represent future ALPR systems which may adopt generative AI technology as part of its back-end character recognition system.

In §6.2, we also implemented proof-of-concept countermeasures to our attack using VLMs, to show that effective countermeasures can be implemented without relying on specific parameters of CIAPLATES attacks being known beforehand.

In addition, we utilized generative AI tools to create automation and processing scripts for statistics collection. We have manually verified the generated scripts for correctness.

8 Conclusion

We have described a novel attack implementation on an ALPR system in the real world through the use of digital license plates that can lead to significant deterioration in recognition accuracy. The CIAPLATES attack exploits the capability of modern electrophoretic displays (that can be driven at a relatively high refresh rate), coupled with multiple methods of generating low-perturbation, unobtrusive physical adversarial examples, to craft a stealthy attack delivery mechanism.

We staged a proof of concept attack using commercially available, off-the-shelf hardware, and a popular, feature-complete ALPR system to show its feasibility and effectiveness over a variety of license plates and in realistic conditions (all while using less than 6.2% of the plate area for perturbations), showing that a CIAPLATES attack is plausible in the real world, highlighting the need for stakeholders to implement effective countermeasures.

Acknowledgements

We would like to thank Wenting Zhang for providing us with an early prototype of the *Glider* [68] programmable EPD driver, as well as expert help on electrophoretic displays.

This work is supported in part by the National Science Foundation under award CNS-1956393.

Ethical Considerations

Given that intentional defacement and concealment of traditional license plates to *illegally* avoid ALPR systems already exists *in the wild*, and that the deployment of digital license plates today is very limited (only in a handful of U.S. states and thus far only on a minority of vehicles), we believe that the downsides of showing that an attack such as CIAPLATES is potentially feasible at this time is outweighed by the knowledge about this potential avenue of attack.

As adversarial attacks on vision systems become more visible through mainstream media, we believe that an eventual disclosure and/or use of attacks such as CIAPLATES in the real world is not a matter of *if*, but rather *when*.

By knowing that such attacks are plausible, stakeholders that depend on the correct operation of ALPR systems can be on the lookout for similar attacks, deploy immediate workarounds (*e.g.,* stricter acceptance parameters or multi-camera imaging) on their systems, as well as consider mitigations before wide-scale deployment of digital license plates.

We re-iterate that **our goal is not to provide malicious actors with a turn-key method of evading law enforcement**. Thus, to prevent abuse of CIAPLATES, we plan to only release the attack generation source code to relevant stakeholders (*e.g.,* ALPR system developers and academic researchers) following a manual vetting process.

For our dataset (*e.g.,* images taken by the setup in Figure 14), we considered the risk of abuse of the images (*e.g.,* a malicious actor simply using the dataset images on their digital plate hardware), and believe that such risk is outweighed by the usefulness of the dataset for adversarial training. We note that plates created as part of the dataset also have tell-tale features to distinguish them from valid license plates.

As part of this work, we are performing responsible disclosure of the attack to ALPR vendors.

We also recognize that by revealing that attacks such as CIAPLATES is possible, legislators in some regions may attempt to place restrictions on the sale or usage of digital license plates, which may cause short-term harm to manufacturers of digital license plates. However, given that practical defenses are possible, we believe that by spurring development of countermeasures from every angle (from improved ALPR systems to better hardware attestation technologies on digital license plates themselves), this short-term concern is outweighed by long-term benefits (both to manufacturers of digital license plates and society at large).

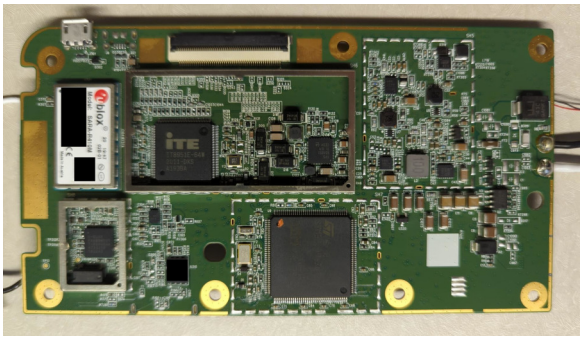


Figure 18: Main circuit board of the RPlate Pro (front).

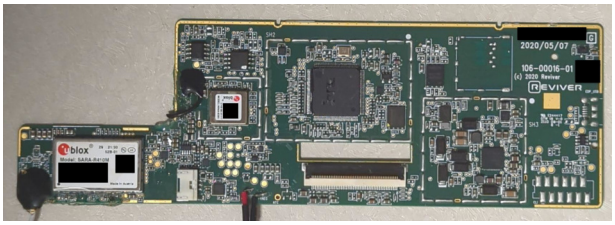


Figure 19: Main circuit board of the battery powered RPlate: a newer, more compact, single-sided PCB design.

Such considerations also extend to other applications of smart, EPD-powered public infrastructure, such as traffic signs and public information displays [18]. As physical, adversarial attacks on autonomous vehicle perception systems have been demonstrated [20,48], we believe they may also be abused in a similar way to CIAPLATES, and that the knowledge from this work will enable stakeholders of such systems to be prepared for similar attacks in the future.

Finally, based on our institution's practices and policies, our work do not meet the criteria for *human subjects research*, placing it outside of scope for IRB engagement.

A Digital License Plate Hardware

A.1 Inner workings

To shed more light into the inner workings of current, commercially available digital license plates, we have acquired two commercially available digital license plates (as of publication time): a battery-powered Reviver RPlate and a hardwired Reviver RPlate Pro for an in-depth look at the hardware (in Figures 18 and 19 respectively) and provide a detailed hardware inventory of these devices in Table 3.

First, we note that all models shared the same EPD module (E-Ink ED113TC1) and driver setup (IT8951E-64). The older *Pro* model utilized a standalone STM32L4 microcontroller[8]

[8]The firmware on flash is reportedly protected from direct readout with development tools. However, voltage glitching methods (such as [40]) has been shown to be effective on at least one Reviver plate model [28]. As our intention is to understand the key hardware details of the device (such that

	RPlate Pro (2018)	Battery RPlate (2020)
Main processor	STMicroelectronics STM32L496ZGT6	U-Blox NINA-B301
Electrophoretic display module	E-Ink ED113TC1 (2400 x 1034)	
Electrophoretic display controller	ITE IT8951E-64	
Cellular modem	U-Blox SARA-R410M	
Embedded SIM	Verizon MFF2 eSIM	
GNSS receiver	U-Blox EVA-M8M-010	N/A
Communications capabilities	LTE	LTE Bluetooth LE

Table 3: RPlate hardware overview.

without any wireless capabilities on its own, while the newer battery-powered model instead uses a microcontroller with integrated Bluetooth radio. Both models utilized a discrete cellular modem as well as an embedded SIM (with an ICCID pointing to it being on the Verizon network). Only the wired model is equipped with a GNSS receiver.

A.2 EPD response time / refresh rate

To measure the EPD panel's real-world response times, we first create software to constantly cycle the display from fully bright (white) to fully dark (black) repeatedly. Then, using our test stand (as shown in Figure 14), we took individual photos of the center of the display in grayscale format while maintaining a constant exposure time and gain setting. We then calculate the average pixel values of each frame, filter out outliers using the 20^{th} and 80^{th} percentiles (as shown in Figure 20), then extract falling and rising edge times to calculate an approximate response time. We observed an average value of 160 ms, with the highest observed value of 190 ms, which can be extrapolated to show a conservative refresh rate figure of $5Hz$ for the physical display module.

B Observed changes to ALPR results

With a CIAPLATES attack, we observed all three types of changes to ALPR recognition results: *character deletion* (where a character is removed from ALPR results, as shown in Figure 21), *character replacement* (where a character is recognized differently from the original, as shown in Figure 22),

our research can be grounded with realistic hardware expectations), we did not attempt to replicate the glitching attacks in this work.

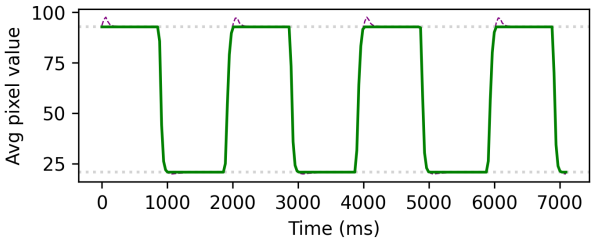


Figure 20: EPD response time experiment visualization (excerpt). Solid green line represents frame-level-averaged pixel intensity values (corrected for outliers). Dashed purple line represents raw data. Dotted gray line represents the 20^{th} and 80^{th} percentile values as calculated from raw data.



(a) Character removal (Character-level generative method)



(b) Recognition result for Figure 21a: SGSEG, 88.24%

Figure 21: Observed changes: character removal.

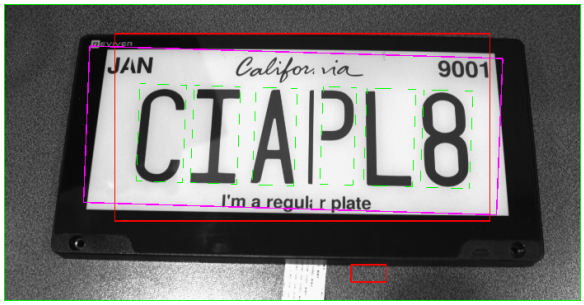
and *character addition* (where a new phantom character is inserted into the results, as shown in Figure 23).

Note that these example visualizations are created with OpenALPR, as it provides character-level bounding boxes in debug mode. These are shown for demonstration purposes only and is not used as part of our attack nor assumed to be available to the adversary.

We observed that *character deletion* and *character replacement* changes are relatively common occurrences, while *character addition* changes are rare (only appearing in a handful of cases), and may be more implementation-specific (as in our example case in Figure 23b where we conjecture that our attack on the characters induced a failure not in the character recognition stage, but rather the segmentation stage, causing



(a) Character replacement (Plate-level heuristic method)



(b) Recognition result for Figure 22a: CIA3L8, 90.36%

Figure 22: Observed changes: character replacement.



(a) Character addition (Character-level heuristic method)



(b) Recognition result for Figure 23a: IBUGE8R, 89.40%

Figure 23: Observed changes: character addition.

OpenALPR to also attempt to recognize the border of the license plate (with a similar contrast) as a character, resulting in character addition as seen in the results).

References

[1] License plates, 3th ed. Standard, American Association of Motor Vehicle Administrators, October 2023.

[2] ADOT motor vehicle division: digital license plate program. azdot.gov/mvd/services/vehicle-services/plates-and-placards/digital-license-plate-program. Accessed: 2025-01-02.

[3] Joseph Allen. Many truckers have bent license plates, but is there a deeper meaning behind it? *Distractify*.

[4] Camera modules for traffic surveillance cameras. alliedvision.com/en/its-traffic-enforcement. Accessed: 2025-08-19.

[5] Considerations when setting up cameras for license plate recognition. docs.avigilon.com/bundle/alta-video/page/Products/aware/lpr/lpr-cam-consider.htm. Accessed: 2025-01-09.

[6] Considerations for plate capture. axis.com/products/things-to-consider-for-license-plate-capture. Accessed: 2025-01-09.

[7] Axon Fleet 3 introduction - Automatic License Plate Recognition. my.axon.com/s/article/Introduction-ALPR. Accessed: 2024-12-17.

[8] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

[9] Mohammad Mahdi Bejani and Mehdi Ghatee. A systematic review on overfitting control in shallow and deep neural networks. *Artif. Intell. Rev.*, 54(8):6391–6438, December 2021.

[10] BOOX Mira e-ink monitors. shop.boox.com/collections/all/products/boox-mira-pro. Accessed: 2024-12-21.

[11] Lorenzo Brigato and Luca Iocchi. A close look at deep learning with small data. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 2490–2497, 2021.

[12] California Assembly Bill 984: vehicle identification and registration: alternative devices.

[13] Sam Curry. Web hackers vs. the auto industry: Mass assignment on Reviver allows an attacker to remotely track and overwrite the virtual license plates for all Reviver customers. samcurry.net/web-hackers-vs-the-auto-industry. Accessed: 2025-01-02.

[14] DaFont online font repository. dafont.com. Accessed: 2025-01-01.

[15] Nilaksh Das, Madhuri Shanbhogue, Shang-Tse Chen, Fred Hohman, Siwei Li, Li Chen, Michael E Kounavis, and Duen Horng Chau. Shield: Fast, practical defense and vaccination for deep learning using JPEG compression. *24nd ACM SIGKDD*, 2018.

[16] DASUNG paperlike E-Ink monitor and tablets. shop.dasung.com. Accessed: 2024-12-21.

[17] ED113TC1 11.3-inch E-Ink display module. data-modul.com/sites/default/files/products/VB3300-LCA-specification-12053273.pdf.

[18] E-Ink U.S. transportation installations. go.eink.com/us-transportation-outdoor-installations.

[19] Magen Emberton. License plate list. licenseplategame.com/list. Accessed: 2025-01-02.

[20] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *2018 IEEE/CVF CVPR*, pages 1625–1634, 2018.

[21] LPR cameras. flocksafety.com/products/license-plate-readers. Accessed: 2025-08-01.

[22] License plate reader FrogLPR. frogparking.com/Froglpr. Accessed: 2024-12-17.

[23] Parking management/enforcement. genetec.com/products/unified-security/autovu/parking-management. Accessed: 2024-12-19.

[24] Genetec SharpV ALPR cameras. genetec.com/press-center/press-releases/2021/08/genetec-introduces-new-generation-of-sharpv-alpr-cameras. Accessed: 2025-08-19.

[25] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR 2015*, 2015.

[26] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2672–2680, 2014.

[27] Google font repository. fonts.google.com/. Accessed: 2025-01-01.

[28] Andy Greenberg. Hackers can jailbreak digital license plates to make others pay their tolls and tickets. wired.com/story/digital-license-plate-jailbreak-hack. Accessed: 2024-12-17.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[30] Amanda Hernandez. 'license plate flippers' help drivers evade police, tickets and tolls. *Stateline*.

[31] Joe Hernandez. California drivers can now sport digital license plates on their cars. *NPR*.

[32] Video analytics for intelligent transportation systems and open road tolling. inextechnologies.com/highways-tolling. Accessed: 2024-12-17.

[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[34] Marcia Kramer. Port Authority, MTA teaming up to catch drivers who use fake or obstructed license plates to avoid tolls. *CBS News New York*.

[35] Hyun Kwon and Jang-Woon Baek. Adv-plate attack: Adversarially perturbed plate for license plate recognition system. *Journal of Sensors*, 2021(1):6473833, 2021.

[36] ELSAG mobile plate hunter ALPR system. leonardocompany-us.com/lpr/elsag-mobile. Accessed: 2024-12-17.

[37] Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. 2013.

[38] The Modos paper monitor. modos.tech. Accessed: 2024-12-21.

[39] Dean Naddeo. Electronic license plate, June 2002.

[40] Johannes Obermaier, Marc Schink, and Kosma Moczek. One exploit to rule them all? on the security of drop-in replacement and counterfeit microcontrollers. In *14th USENIX Workshop on Offensive Technologies*. USENIX Association, 08 2020.

[41] City of Palm Springs. ALPR technology frequently asked questions. palmspringsca.gov/government/departments/police/automated-license-plate-recognition-alpr. Accessed: 2024-12-21.

[42] OpenAI. GPT-5 mini, 2025. Accessed: 2026-01-10.

[43] OpenALPR: automatic license plate recognition library. github.com/openalpr/openalpr, 2016.

[44] OpenStreetMap object tag search: surveillance, type=ALPR. taginfo.openstreetmap.org/tags/surveillance?type=ALPR. Accessed: 2025-08-01.

[45] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *ASIA CCS '17*, page 506–519, 2017.

[46] Plate Recognizer automatic license plate recognition. platerecognizer.com/. Accessed: 2025-05-01.

[47] Plate Recognizer blog. platerecognizer.com/blog. Accessed: 2025-05-01.

[48] Steve Povolny. Model hacking ADAS to pave safer roads for autonomous vehicles. *McAfee Blog*.

[49] Yaguan Qian, Dan-feng Ma, Bin Wang, Jun Pan, Jian-Hai Chen, Jia-Min Wang, and Jing-Sheng Lei. Spot evasion attacks: Adversarial examples for license plate recognition systems with convolution neural networks. *CoRR*, abs/1911.00927, 2019.

[50] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[51] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[52] Adaptive Recognition. Vidar ALPR cameras specification sheet. adaptiverecognition.com/products/vidar-anpr-camera. Accessed: 2026-04-24.

[53] Rekor Scout ALPR and Vehicle Recognition. rekor.ai/software/scout. Accessed: 2025-01-01.

[54] Customer stories - Rekor Systems. rekor.ai/customer-stories. Accessed: 2025-01-01.

[55] Are smart phones needed to use RPlates? support.reviver.com/knowledge/are-smart-phones-needed-to-use-rplates. Accessed: 2024-12-17.

[56] RPlate digital license plate. reviver.com/rplate. Accessed: 2024-12-17.

[57] Matthew Sedacca. Unbe-leaf-able: Scofflaws dodge tolls and traffic cameras with foliage. *New York Post*.

[58] Jithmi Shashirangana, Heshan Padmasiri, Dulani Meedeniya, and Charith Perera. Automated license plate recognition: A survey on methods and techniques. *IEEE Access*, 9:11203–11225, 2021.

[59] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, Dec 1997.

[60] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, October 2019.

[61] Infinity digital lane system. transcore.com/tolling/infinity-digital-lane-system. Accessed: 2024-12-17.

[62] Texas DMV: License plates. txdmv.gov/motorists/license-plates. Accessed: 2025-01-18.

[63] Vitronic traffic enforcement technology. vitronic.com/en-us/traffic-technology/traffic-enforcement. Accessed: 2024-12-21.

[64] Conrado Vizcarra, Shadan Alhamed, Abdulelah Algosaibi, Mohammed Alnaeem, Adel Aldalbahi, Nura Aljaafari, Ahmad Sawalmeh, Mahmoud Nazzal, Abdallah Khreishah, Abdulaziz Alhumam, and Muhammad Anan. Deep learning adversarial attacks and defenses on license plate recognition system. *Cluster Computing*, 27(8):11627–11644, Nov 2024.

[65] Luke Wright. From speed cameras to AI: The future of traffic enforcement. *TechBullion*.

[66] Bing Xu. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[67] Zhenbo Xu, Wei Yang, Ajin Meng, Nanxue Lu, Huan Huang, Changchun Ying, and Liusheng Huang. Towards end-to-end license plate detection and recognition: A large dataset and baseline. In *Computer Vision – ECCV 2018*, pages 261–277, Cham, 2018.

[68] Wenting Zhang. Glider: open source E-ink monitor. gitlab.com/zephray/glider, 2024.

ARTIFACT EVALUATED
USENIX
AVAILABLE

Short: Race to Charge: Concurrency Bugs and State Integrity Violations in EV Charging Infrastructure

Jaeyeong Lee
Independent Researcher

Seonhyeong Lee
SK Shieldus

Abstract

Electric vehicle (EV) charging infrastructure must juggle hardware control, communication protocols, and cloud-based management all at once, relying on complex event-driven multithreaded software underneath. This paper examines concurrency vulnerabilities in `everest-core`, an open-source charging framework hosted under the Linux Foundation Energy umbrella. Through manual code auditing of its threading and callback architecture, combined with sanitizer-assisted reproduction experiments, we uncovered **11** concurrency vulnerabilities, all of which now have public CVE IDs. The vulnerabilities fall into three recurring patterns: inconsistent locking discipline, lack of atomicity in state transitions, and missing state validation on asynchronous events. We confirmed eight data races using TSan, ASan, and UBSan, and the remaining three state integrity violations through logic-based proof-of-concept exploits. These bugs can lead to unauthorized charging that persists after a remote stop command, authorization withdrawal bypass, and denial of service through memory corruption.

1 Introduction

EV charging stations are spreading fast across the globe. According to IEA figures, more than 1.3 million public charging points were added in 2024, bringing the global stock to over 5 million [8], and the software powering them has grown correspondingly complex.

An EVSE (EV Supply Equipment) unit handles CP signaling with the vehicle (IEC 61851, ISO 15118 [9]), cloud communication over OCPP [15], authorization workflows, and power management—all within a single multithreaded stack. Asynchronous events and callbacks coordinate the whole thing, with extensive cross-thread interaction throughout.

Research into EV charging security has branched out in many directions: protocol-level analysis [2, 3], authentication bypass in management systems [14], physical-layer attacks [19], V2G relay attacks [5], and broad threat surveys [1, 10]. Automated approaches like OCPPStorm [6] have also emerged, using protocol message fuzzing to hunt for flaws. Most of this work, however, targets protocol *specifications* or *input handling*. Empirical studies that dig into *implementation-level* concurrency bugs arising from multithreaded execution inside EVSE software remain scarce.

This gap matters because it means implementation-level risks may be underestimated. Concurrency bugs can be triggered even when the protocol itself is correctly implemented—all it takes is legitimate messages arriving at adversarial timing. The consequences range from session state corruption and authorization bypass to memory safety violations, none of which are easily caught by conventional fuzzing or protocol analysis alone.

We focus on `everest-core` [12], an open-source EV charging framework hosted by Linux Foundation Energy. `everest-core` sees real-world deployment in commercial chargers including the Pionix BelayBox, and a stack buffer overflow in its V2GTP implementation was previously reported as CVE-2024-37310 [16]. That said, a systematic empirical examination of concurrency issues across the multithreaded architecture has, to our knowledge, only been reported in limited form.

Our analysis reveals that three recurring concurrency patterns produce exploitable vulnerabilities throughout the modular architecture of `everest-core`: (1) *Inconsistent locking discipline*, where only some code paths acquire a lock before accessing a shared resource; (2) *Lack of state transition atomicity*, where a state change and its associated flags are not updated as a single unit, exposing intermediate states; (3) *Missing state validation*, where incoming asynchronous events get processed without checking whether the current state actually permits them. From these three patterns, we identified **11** concurrency vulnerabilities. All **11** now have public CVE IDs: CVE-2026-33015, CVE-2026-33014, CVE-2026-29044, CVE-2026-26074, CVE-2026-26073, CVE-2026-26072, CVE-2026-26071, CVE-2026-26070, CVE-2026-33009, CVE-2026-27814, and CVE-2026-27813. We confirmed eight data races with TSan [17]/ASan/UBSan and the three state integrity violations with logic-based PoCs.

Contributions.

- We carry out a concurrency vulnerability analysis of an open-source EV charging stack, identifying **11** vulnerabilities with **11** public CVE IDs, and verify their impact through reproduction experiments. Unlike prior work that concentrates on protocol specifications and input handling, our study targets timing-dependent flaws in the multithreaded runtime.
- We show that the discovered vulnerabilities stem from three recurring patterns and trace these back to structural characteristics of the event-driven modular architecture.
- We confirm every finding through sanitizer verification or logic PoCs, and present attack scenarios encompassing unauthorized charging continuation, authorization bypass, and denial of service.

2 Background and Threat Model

2.1 EV Charging Architecture

An EVSE communicates with two external entities. One is the *electric vehicle* (EV) via CP signaling (IEC 61851) or ISO 15118; the other is the *Charging Station Management System* (CSMS) over an OCPP WebSocket connection. Internally, four subsystems cooperate: hardware control (*BSP*), the charging state machine (*EVSE Manager*), authorization (*Auth Handler*), and cloud communication (*OCPP module*).

The `everest-core` framework (release 2025.12.1) realizes this architecture as a modular system where components talk to each other through typed interfaces and callbacks. Figure 1 illustrates the concurrent execution architecture along with the relevant attack paths.

A charging session's lifecycle is governed by the state machine inside the `Charger` class. Transitions from `Idle` through `Charging` to `Finished` run on a dedicated thread, but OCPP callbacks, MQTT handlers, authorization responses, and EV information updates each arrive on their own separate threads.

2.2 Threat Model

We consider attackers who can influence one or more legitimate EVSE input channels, but who do not have code execution inside the EVSE process. Table 1 aligns each finding with the attacker-visible channel needed to trigger it. The three main channels are: a compromised or malicious CSMS that can send OCPP commands; an EV-side adversary that can shape CP/ISO 15118 timing, plug/unplug behavior, or BCB toggles; and an exposed or tunneled MQTT/API management plane. These avenues reflect realistic deployment threats: CSMS account compromise [3] and on-site LAN access at public charging stations have both been documented.

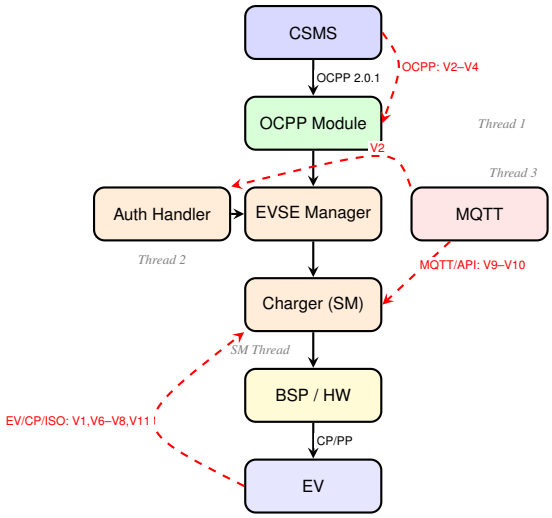


Figure 1: Concurrent execution architecture of `everest-core`. Solid arrows indicate normal data flow; red dashed arrows mark attacker-facing input channels that can shape event timing. The resulting races occur inside callback and state-machine threads.

Table 1: Attacker-facing channels required for each finding.

ID	Channel	Attacker requirement
V1	CSMS + EV/CP	RemoteStop plus EV-side BCB timing during `StoppingCharging`.
V2	CSMS + MQTT/API	OCPP stop request plus delayed Auth response via management plane; not CSMS-only.
V3	CSMS/API timing	WithdrawAuthorization before TransactionStarted is reflected in AuthHandler.
V4	CSMS/OCPP	Firmware or log status events racing with startup event buffering.
V5	Startup event	OCPP 1.6 power-meter key event during initialization.
V6,V8	EV info + meter	SoC update, meter callback, and session-end timing; can occur during normal charging.
V7	EV/ISO info	EVCCID update racing with session processing.
V9,V10	MQTT/API	Phase-switch command reaching Charger while the state-machine thread runs.
V11	EV/Auth timing	Plug-in, session, and authorization timeout paths racing on the Auth timer.

The attacker's goal is to exploit concurrency bugs through *adversarial timing* of otherwise legitimate interactions. We explicitly distinguish externally triggerable bugs from races that primarily represent reliability hazards under normal operation. In particular, V2 combines an OCPP RemoteStop with a delayed authorization response on `everest-core`'s internal Auth–EVSE command plane; we treat it as remotely triggerable when the MQTT/API plane is reachable, but not as a pure CSMS-only exploit.

3 Methodology

We conducted the analysis against `everest-core` release 2025.12.1 in three phases.

Phase 1: Identifying the Concurrency Attack Surface.

Table 2: Thread-to-shared-resource mapping. R = read, W = write, P = mutex-protected, U = unprotected. The OCPP column encompasses both OCPP callbacks and EV info / power meter callbacks.

Shared Resource	SM	OCPP	MQTT	Auth
`shared_context`	R/W P	–	W U	–
`transaction_active`	R/W P	R	–	R
`authorized`	R/W P	–	–	W U
`event_queue`	–	R/W P/U	–	–
`evse_soc_map`	–	R/W U	–	–
`evse_evcc_id`	–	R/W U	–	–
`timeout_timer`	–	–	–	W U

We analyzed four core module groups—EVSE Manager, OCPP 1.6, OCPP 2.0.1, and Auth Handler—and identified five distinct execution contexts: (1) the state machine thread (`Charger::main_thread`), (2) OCPP callback threads (WebSocket receiver), (3) the MQTT subscription handler thread, (4) authorization response threads, and (5) EV info / power meter callback threads. Context (5) fires inside the OCPP module when power meter or EV information updates come in. It shares a process with the OCPP callback thread (2) but forms a separate callback chain, making it an independent source of races. We mapped seven shared state fields accessed across these contexts; Table 2 summarizes the key thread-to-resource mappings. The OCPP column in the table covers both (2) and (5), reflecting the fact that these two contexts access the same shared resources but compete through different callback paths.

Phase 2: Systematic Code Review. For each identified shared resource, we traced every access path and checked whether it was lock-protected. Our review concentrated on three bug patterns: (1) *Inconsistent locking*: some paths acquire a mutex for a given field while others skip it entirely; (2) *Lack of state transition atomicity*: a state change and the flags it depends on get updated at separate points, leaving an observable intermediate state; (3) *Missing state validation*: an incoming event gets processed without verifying that the current state permits it.

Bug Determination Criteria. We classified a suspect site as a vulnerability when: (i) concurrent accesses to the same field exist from different execution contexts, (ii) at least one access is a write, and (iii) no shared lock serializes the two accesses, or the locking is applied inconsistently. For state integrity violations, criterion (ii) was replaced with "there exists an event sequence that permits a semantically invalid transition from the current state."

Phase 3: PoC Construction and Verification. We wrote Google Test–based PoCs for every suspect bug. For concurrency memory defects (V4–V11), we spawned concurrent threads to induce the race condition and verified the outcome with TSan (happens-before violations), ASan (heap-use-after-free), and UBSan (loading of invalid values). For state integrity violations (V1–V3), we replayed specific event sequences and asserted that the state machine reached an invalid state—these are logic-based PoCs. For V2, we additionally reproduced the sequence on real ARM hardware using a Raspberry Pi 4 deployment. This is not an arbitrary test board choice: `everest-core` is publicly packaged for Raspberry Pi via the `everest-demo` snap [18]. On the Pi, we ran the unmodified `everest-core` 2025.12.1 manager together with SIL EVSE modules; the CSMS side issued `RequestStopTransaction`, while the delayed authorization response was injected through the `everest-core` MQTT/API command plane. Across three runs with injected delays of 30 ms, 50 ms, and 70 ms, the stop request was accepted in all cases, yet no OCPP `TransactionEvent(Ended)` and no local `Transaction Finished` transition were observed. This experiment validates the state-machine failure in a hardware-backed deployment of the stack, but it should not be interpreted as a CSMS-only trigger.

4 Findings

We found **11** concurrency vulnerabilities spanning three categories. Table 3 provides a full summary. The corresponding public CVE IDs are CVE-2026-33015, CVE-2026-33014, CVE-2026-29044, CVE-2026-26074, CVE-2026-26073, CVE-2026-26072, CVE-2026-26071, CVE-2026-26070, CVE-2026-33009, CVE-2026-27814, and CVE-2026-27813. The "Access Req." column denotes the attacker-visible input channel or trigger precondition required to reach the vulnerable interleaving, thereby aligning each finding with the threat model in Section 2.2. Entries may refer either to an attacker-facing channel (e.g., CSMS, MQTT/API, EV/CP/ISO) or to the minimal triggering precondition when no dedicated external channel is required (e.g., normal operation or startup timing).

4.1 State Integrity Violations

V1 through V3 are logic-level flaws where the charging session state machine permits semantically invalid transitions under specific timing. We describe V2 in detail as a representative case.

V2: Delayed Authorization Response Defeats RemoteStop. The function `cancel_transaction()` sets `authorized = false` as part of the stop sequence. But if a delayed response to a previously issued authorization request arrives after the RemoteStop has been processed, `authorize(true)` restores `authorized = true` without checking the current state. When the EV subsequently disconnects or the contactor-open event is processed, the code may see that `authorized` is true and skip the `stop_transaction()` call, leaving the transaction hanging open. Figure 2 depicts this event sequence. The root cause maps to the third pattern (*missing state validation*): `authorize()` unconditionally sets the

Table 3: Summary of concurrency vulnerabilities discovered in `everest-core`. **Category**: A = state integrity violation, B = shared container data race, C = external-input cross-thread race. **Confirmed**: T = TSan, A = ASan, U = UBSan, L = logic PoC. **Root Cause**: P1 = inconsistent locking, P2 = lack of atomicity, P3 = missing state validation. **Access Req.** maps each finding to the corresponding attacker-facing channel or trigger precondition. V6–V8 can also occur under certain timing races during normal operation. The table intentionally excludes independently reported non-concurrency input-validation bugs.

ID	Component	Trigger Summary	Cat.	Root Cause	Access Req.	Impact	Conf.	CVE
V1	Charger	BCB toggle after RemoteStop	A	P2	CSMS+EV timing	Unauth. charge cont.	L	CVE-2026-33015
V2	Charger	Delayed AuthResp after RemoteStop	A	P3	CSMS+MQTT/API	Billing integrity viol.	L	CVE-2026-33014
V3	AuthHandler	WithdrawAuth before TransactionStarted	A	P3	CSMS/API timing	Auth. bypass	L	CVE-2026-29044
V4	OCPP 2.0.1	FW/log callback pushes without lock	B	P1	CSMS network	Crash	T	CVE-2026-26074
V5	OCPP 1.6	Power meter key callback pushes w/o lock	B	P1	Startup event	Crash	A,U	CVE-2026-26073
V6	OCPP 1.6	Concurrent R/W/reset on SoC map	B	P1	Normal operation	Exception	A	CVE-2026-26072
V7	OCPP 2.0.1	Concurrent R/W on EVCC ID string	B	P1	Normal operation	Memory corruption	A	CVE-2026-26071
V8	OCPP 2.0.1	Concurrent R/W/reset on SoC map	B	P1	Normal operation	Exception	T	CVE-2026-26070
V9	Charger	MQTT phase-switch writes without lock	C	P1	MQTT (LAN)	State corruption	T,U	CVE-2026-33009
V10	Charger	Additional field accesses in same method	C	P1	MQTT (LAN)	UB	T,A,U	CVE-2026-27814
V11	AuthHandler	Concurrent timer stop/timeout invocation	C	P1	EV/Auth timing	Crash	A	CVE-2026-27813

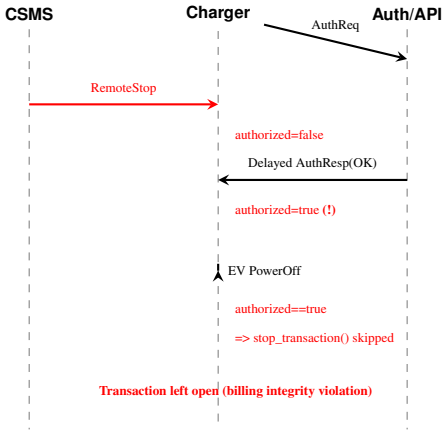


Figure 2: V2 attack sequence. A delayed authorization response, delivered through the internal Auth/MQTT command plane in our deployment experiment, reactivates the `authorized` flag so `stop_transaction()` may not fire when the EV disconnects.

flag even in the `StoppingCharging` state. We further observed the same behavior on a Raspberry Pi 4 deployment of `everest-core` 2025.12.1: even though the charge point returned `Accepted` to `RequestStopTransaction`, the transaction remained open after the delayed authorization response was injected. V1 is a case where `cancel_transaction()` does not clear `transaction_active` right away (*lack of atomicity*), allowing a BCB toggle to restart the session. V3 involves a timing window in asynchronous event delivery where a WithdrawAuthorization takes the deauthorize-only path instead of the stop-transaction path.

4.2 Shared Container Data Races

V4 through V8 involve unprotected concurrent access to `std::map`, `std::queue`, `std::optional`, and `std::string` fields from multiple callback threads.

Listing 1: V4: Locking inconsistency on the same event queue. The error handler acquires the mutex; the firmware callback does not.

```
// Error handler - correctly locked
void handle_session_event(...) {
  std::scoped_lock lock(session_event_mutex);
  event_queue[id].push(event); // Protected
}
// Firmware callback - no lock!
void handle_firmware_status(...) {
  // No mutex acquired!
  event_queue[id].push(fw_event); // Unprotected
}
```

V4: Event Queue Locking Inconsistency. The OCPP 2.0.1 module uses a queue of type `std::map<int32_t, std::queue<variant>>` to buffer events before the stack has finished initializing (`started == false`). The error event handler dutifully acquires `session_event_mutex`, but the firmware update status and log status callbacks push to the same queue without grabbing any lock at all. Listing 1 shows the inconsistency side by side. The race window is especially wide during module initialization: callbacks start arriving while the OCPP stack is still setting up its WebSocket connection, and every path buffers into the same queue yet locking is applied unevenly. TSan flagged concurrent read/write operations on the internal block pointers of `std::deque`.

V5 reproduces the same pattern in the OCPP 1.6 module, this time in the power meter public key callback; ASan/UBSan caught misaligned address access errors. V6–V8 involve completely unprotected accesses to `evse_soc_map` and `evse_evcc_id`, where a power meter callback (reading), an EV info callback (writing), and a session-end handler (resetting) all race against each other. In V7, the race on a `std::string` triggers heap-use-after-free (ASan confirmed). In V6 and V8, the optional gets invalidated between a `has_value()` check and the subsequent `value()` call, throwing `std::bad_optional_access`.

4.3 External-Input Cross-Thread Races

V9 through V11 are triggered by external inputs that reach shared state without first acquiring the state machine mutex.

V9: MQTT Phase-Switch Race. The MQTT message `switch_three_phases_while_charging` calls a method that reads and writes several fields of `shared_context` without holding `state_machine_mutex`. Meanwhile, the state machine loop accesses the exact same fields while holding the mutex. TSan reported four data races, and UBSan caught the loading of an invalid value (190) for a `bool` field. Under the C++ standard, only 0 and 1 are valid representations for `bool`. A value of 190 is one instance of an anomalous value that may be associated with unsynchronized concurrent writes. Such anomalous values can lead to unexpected control flow or outright crashes. Because the phase-switch logic determines the power delivery mode (single-phase vs. three-phase AC), corrupted state values here can propagate into erratic behavior in the higher-level control logic.

V11 is a heap-use-after-free in the Boost.Asio–based timer of `AuthHandler`. The `timeout()` and `stop()` functions get called concurrently from multiple code paths—plug-in events, authorization responses, session termination, deauthorization—and the timer callback ends up referencing a destroyed `io_context` (ASan confirmed).

5 Security Impact

The security consequences of the discovered vulnerabilities fall into three types.

Unauthorized Charging Continuation and Billing Integrity Violation. V1 lets a session restart through a BCB toggle during the `StoppingCharging` window after a RemoteStop, keeping the charger energized. V2 is different in mechanism but similar in outcome: a delayed authorization response derails the stop sequence, leaving the transaction unclosed. In our deployment experiment, the stop command was sent over OCPP and the delayed authorization response was delivered over the `everest-core` MQTT/API command plane. Both undermine the integrity of billing records.

Authorization Bypass. With V3, a session can persist even after a WithdrawAuthorization command. Picture an operator revoking a stolen RFID card's authorization—the command appears to succeed, yet charging continues uninterrupted.

Denial of Service and State Corruption. V4, V5, V7, and V11 can crash the EVSE process via heap corruption or use-after-free. V6–V8 pose reliability risks that can surface under certain timing conditions during normal charging sessions; an attacker who induces rapid plug/unplug cycles can convert them into intentional denial of service. V9–V10 corrupt the internal state of the charging state machine through MQTT message injection, and the corrupted values can propagate into erratic behavior at higher control layers.

We experimentally verified that the scenarios above are reproducible through adversarial timing of legitimate event paths, without code execution on the EVSE process. Some paths are reachable from a CSMS, while others require EV-side timing or access to the `everest-core` MQTT/API command plane. V6–V8 deserve particular note: they can manifest without any external attack under certain timing races, posing an availability risk on their own.

6 Discussion

Architectural Lessons from Recurring Patterns. Three patterns recur across all **11** findings. Inconsistent locking discipline shows up in V4–V11 (8 out of 11), lack of state transition atomicity in V1 and V2, and missing state validation in V2 and V3. These point to a structural vulnerability tendency tied to the framework's design.

Why is inconsistent locking so dominant at 8 out of 11? The answer lies in `everest-core`'s modular structure. Each module manages its own locking policy independently, so when access paths to a shared resource cross module boundaries, some paths end up without a lock. This observation aligns with the atomicity violation and order violation patterns that Lu et al. [13] documented in general-purpose software, and it underscores the need for principled design measures—such as wrapping cross-module shared state behind thread-safe accessors.

The lack of atomicity (V1, V2) and missing state validation (V2, V3) share a common root cause. The state transitions of a charging session are distributed across multiple flags—`authorized`, `transaction_active`, `current_state`—but these flags never get updated as a single atomic unit. The problem was most likely to appear where (i) state transitions and their associated flags are scattered across multiple update sites, and (ii) asynchronous callbacks do not adequately account for what state the system is actually in.

These patterns were observed in `everest-core` and may arise in similar event-driven architectures. We performed a lightweight architectural inspection of other open-source EV charging stacks, including OpenEVSE WiFi, Java-OCA-OCPP, WWCP_OCPP, and SteVe. We did not conduct full vulnerability audits or PoC validation for those projects. However, several of them exhibit the same pattern-level precondition: multiple asynchronous control paths, such as OCPP, HTTP, MQTT, scheduler logic, or WebSocket session handling, converge on shared EVSE, session, or transaction state. This suggests that the architectural conditions behind our findings are not unique to `everest-core`, although we do not claim that the same vulnerabilities are present in those projects.

Our analysis was primarily manual, but parts of the workflow are amenable to automation. Callback entry-point discovery, shared-state inventory construction, lockset comparison, and cross-channel state-update detection can be implemented

as static-analysis heuristics. The harder step is exploitability validation: deciding whether a flagged interleaving violates an EV-specific invariant such as RemoteStop irreversibility, authorization finality, or transaction accounting still requires domain knowledge and targeted replay. A natural direction for future work is therefore a domain-specific concurrency testing framework for EVSE software, combining static candidate generation with schedule fuzzing over OCPP, MQTT/API, and EV/CP/ISO event channels.

Comparing with the concurrency bug characteristics that Li et al. [11] reported for interrupt-driven embedded software, our findings show analogous patterns manifesting at the thread level rather than the interrupt level.

Why EV Charging Software Is Particularly Susceptible. Three structural characteristics of EV charging stacks make concurrency bugs more consequential than in typical networked services. First, state transitions directly govern *real-time hardware control*—contactor switching, CP signaling, and power delivery—so a race does not merely corrupt data but can sustain or interrupt physical energy flow. Second, *multiple independent protocol stacks* (OCPP, ISO 15118, MQTT) each inject events from their own threads, creating a structurally wider cross-thread interaction surface than single-protocol servers. Third, a single state machine binds *authorization, billing, and power delivery* together; any intermediate state exposed by a race therefore has immediate financial consequences (e.g., an unclosed transaction) or safety implications (e.g., unauthorized energization). In contrast, a typical web service experiencing a data race risks database inconsistency, whereas the same class of bug in an EVSE directly affects physical power output and billing integrity.

Limitations. This analysis covers a single open-source framework, `everest-core`. Commercial EVSE implementations may exhibit different concurrency patterns. We validated V2 on a Raspberry Pi 4 hardware deployment with SIL EVSE modules, but we did not test against production EVSE hardware with real contactors, PLC modems, and grid-side power electronics. Given the code-review-based nature of our methodology, additional concurrency bugs may well exist.

7 Related Work

EV charging security research has advanced along several fronts. On the protocol analysis side, Baker and Martinovic [3] studied OCPP message forgery and replay threats, while Alcaraz et al. [2] covered threats and countermeasures for charging infrastructure 4.0. OCPPStorm by Coppoletta et al. [6] applied black-box fuzzing to find message processing defects. All of these focus on message content and format rather than multithreaded execution timing. At the physical layer, Son et al. [19] worked on signal injection defense, Conti et al. [5] demonstrated relay attacks, and Becker et al. [4] conducted V2G threat analysis. Broader threat taxonomies appear in the surveys by Johnson et al. [10], Alashjaee et al. [1], and Nasr et al. [14]. Our work complements these efforts by showing that implementation-level defects rooted in multithreaded execution timing can persist even when the protocol is correctly realized.

In the concurrency analysis space, the atomicity violation and order violation patterns identified by Lu et al. [13] map directly onto our V1–V3 and V4–V10 findings. Wang et al. [20] detected races in interrupt-driven embedded systems using a combined static/dynamic approach, and Li et al. [11] empirically classified concurrency bugs in interrupt-driven software. TSan [17] and RacerX [7] are core techniques we relied on for verification and exploration.

To our knowledge, empirical studies of concurrency vulnerabilities in EV charging stacks have only been reported in limited form.

8 Conclusion

This paper presents a concurrency vulnerability analysis of the `everest-core` EV charging framework, uncovering **11** flaws with **11** public CVE IDs. The vulnerabilities cluster around three patterns—inconsistent locking discipline, lack of state transition atomicity, and missing state validation—and their consequences include unauthorized charging continuation, authorization bypass, and denial of service. The predominance of inconsistent locking at 8 out of 11 cases is particularly telling: it signals that modular architectures lacking a centralized locking policy for shared state are structurally prone to concurrency bugs. This risk factor is likely not unique to `everest-core` and may recur in other event-driven multithreaded systems of similar design.

Artifact Availability. The artifact associated with this paper is permanently archived at https://doi.org/10.5281/zenodo.19548057.

References

[1] Abdullah M. Alashjaee, Abdulhaq Mohammad, and Amer A. Alsulami. A comprehensive survey of cybersecurity threats, attacks, and effective countermeasures in electric vehicle charging infrastructure. *World Electric Vehicle Journal*, 15, 2024.

[2] Cristina Alcaraz, Javier Cumplido, and Alicia Triviño. OCPP in the spotlight: Threats and countermeasures for electric vehicle charging infrastructures 4.0. *International Journal of Information Security*, 22, 2023.

[3] Richard Baker and Ivan Martinovic. OCPP security: Exploring the attack surface of the Open Charge Point Protocol. In *Proceedings of the IEEE Symposium on Security and Privacy Workshops (SPW)*, 2023.

[4] Hanno Becker, Martin Serror, Martin Henze, and Klaus Wehrle. Threat analysis of the vehicle-to-grid charging ecosystem. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS) Workshops*, 2021.

[5] Mauro Conti, Denis Donadel, Radha Poovendran, and Federico Turrin. EVExchange: A relay attack on electric vehicle charging system. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 2022.

[6] Gaetano Coppoletta, Rigel Gjomemo, Amanjot Kaur, Nima Valizadeh, V. N. Venkatakrishnan, and Omer Rana. OCPPStorm: A comprehensive fuzzing tool for OCPP implementations. In *Proceedings of the Symposium on Vehicle Security and Privacy (VehicleSec)*, 2024.

[7] Dawson Engler and Ken Ashcraft. RacerX: Effective, static detection of race conditions and deadlocks. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[8] International Energy Agency. Global EV outlook 2025: Electric vehicle charging. https://www.iea.org/reports/global-ev-outlook-2025/electric-vehicle-charging, 2025.

[9] International Organization for Standardization. ISO 15118: Vehicle to grid communication interface, 2022.

[10] Jay Johnson, Timothy Berg, Benjamin Anderson, and Brian Wright. Review of electric vehicle charger cybersecurity vulnerabilities, potential impacts, and defenses. *Energies*, 15, 2022.

[11] Chenyang Li, Rui Chen, Bo Wang, Zhengyu Wang, Ting Yu, Yu Jiang, Bin Gu, and Mengfei Yang. An empirical study on concurrency bugs in interrupt-driven embedded software. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2023.

[12] Linux Foundation Energy. EVerest: An open source modular framework for EV charging infrastructure. https://github.com/EVerest/everest-core, 2024.

[13] Shan Lu, Soyeon Park, Eunsoo Seo, and Yuanyuan Zhou. Learning from mistakes: A comprehensive study on real world concurrency bug characteristics. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2008.

[14] Tony Nasr, Sadegh Torabi, Elias Bou-Harb, Carol Fung, and Chadi Assi. Power jacking your station: In-depth security analysis and exploitation of electric vehicle charging station management systems. *Computers & Security*, 112, 2023.

[15] Open Charge Alliance. Open Charge Point Protocol (OCPP) 2.0.1 specification. https://openchargealliance.org/, 2022.

[16] PlaxidityX. CVE-2024-37310: EVerest v2gtp implementation stack buffer overflow. https://nvd.nist.gov/vuln/detail/CVE-2024-37310, 2024.

[17] Konstantin Serebryany and Timur Iskhodzhanov. ThreadSanitizer: Data race detection in practice. In *Proceedings of the Workshop on Binary Instrumentation and Applications (WBIA)*, 2009.

[18] Snapcraft. Install EVerest - EV charging open-source stack on raspberry pi. https://snapcraft.io/install/everest-demo/raspbian, 2024.

[19] Soyeon Son, Kyungho Joo, Wonsuk Choi, and Dong Hoon Lee. Securing EV charging system against physical-layer signal injection attack. In *Proceedings of the Symposium on Vehicle Security and Privacy (VehicleSec)*, 2024.

[20] Yu Wang, Fengjuan Gao, Linzhang Wang, Tingting Yu, Jianhua Zhao, and Xuandong Li. Automatic detection, validation, and repair of race conditions in interrupt-driven embedded software. *IEEE Transactions on Software Engineering*, 48(1), 2022.

USENIX VehicleSec '26 Artifact Appendix: Race to Charge: Concurrency Bugs and State Integrity Violations in EV Charging Infrastructure

Jaeyeong Lee
Independent Researcher
lee@jaeyeong.cc

Seonhyeong Lee
SK Shieldus
leeseon4@gmail.com

A Artifact Appendix

A.1 Abstract

This artifact accompanies "Race to Charge: Concurrency Bugs and State Integrity Violations in EV Charging Infrastructure." It contains the source snapshot used in the paper, a lightweight scanner that codifies the paper's review heuristics for callback/shared-state concurrency bugs, and the Raspberry Pi 4 replay harness used for the hardware-backed validation of V2 / CVE-2026-33014. The artifact is intended for local source-code inspection and local SIL/testbed replay only; it must not be run against production charging stations or infrastructure connected to real vehicles or power electronics.

A.2 Description and Requirements

Artifact location. The frozen artifact is archived at:

https://doi.org/10.5281/zenodo.19548057

The unpacked archive contains `artifact_available/scanners/` (heuristic scanner and README), `artifact_available/poc/` (Raspberry Pi replay for V2 / CVE-2026-33014), `artifact_available/docs/` (setup notes and result summary), and `artifact_appendix/` (this appendix source).

Badge. This submission was evaluated for and awarded **Artifacts Available**. We did not apply for the Functional or Results Reproduced badges.

Hardware. The scanner can be inspected on a normal Linux workstation. The full replay requires a Raspberry Pi 4; our reference platform was a Raspberry Pi 4 Model B (Broadcom BCM2711, 4 ARM Cortex-A72 cores, 4 GB RAM) running arm64 Ubuntu 24.04. No GPU, FPGA, PLC modem, oscilloscope, production EVSE, vehicle, or power electronics are required.

Software. The replay requires Linux, Python 3.10 or newer, Python packages `paho-mqtt`, `PyYAML`, and `websockets`, a local Mosquitto or compatible MQTT broker on `127.0.0.1:1883`, and an in-place `everest-core` 2025.12.1 build with `build/dist/bin/manager`. The scanner itself requires only Python 3 and the checked-out source tree. No external dataset, trained model, or downloaded benchmark is used.

A.3 Set-up

Commands below assume the evaluator starts from the root of the unpacked artifact archive.

1. Install replay dependencies: `cd artifact_available/poc` and `python3 -m pip install -r requirements.txt`.
2. Start a local broker, for example `mosquitto -d -p 1883`.
3. Build `everest-core` if needed: `cmake -S . -B build -GNinja -DCMAKE_BUILD_TYPE=RelWithDebInfo` followed by `cmake --build build -j4`.
4. Run the scanner smoke test: `cd artifact_available/scanners` and `python3 check_concurrency_patterns.py --json /tmp/everest-candidates.json`.

The expected scanner output mentions candidate sites involving `event_queue`, `evse_soc_map`, `evse_evcc_id`, `shared_context.authorized`, `shared_context.transaction_active`, and `timer.callback`, and writes the JSON report to `/tmp/everest-candidates.json`. To check replay dependencies, run `python3 -c "import yaml, websockets"` and `python3 -c "import paho.mqtt.client"`.

A.4 Claims and Evaluation Workflow

This appendix is included for the Available badge; the following workflow summarizes the validation material shipped with the artifact.

C1. The scanner replays the paper's review heuristics on `everest-core` 2025.12.1 and rediscovers the main source-code patterns underlying the confirmed findings.

C2. The V2 / CVE-2026-33014 state-integrity failure can be replayed on a Raspberry Pi 4 deployment of the unmodified `everest-core` manager with SIL EVSE modules.

E1: Scanner replay. Run `check_concurrency_patterns.py` from `artifact_available/scanners/`. The mapping from reported candidate classes to the paper's findings is documented in `artifact_available/scanners/README.md`.

E2: Single V2 replay. After completing setup, run `f2_remote_stop_bypass_poc.py` from `artifact_available/poc/` with a 50 ms delayed authorization response. The successful run prints `RESULT VULNERABLE_REMOTE_STOP_BYPASSED`; the generated logs should show an accepted stop request without a subsequent local `Transaction Finished` or OCPP `TransactionEvent(Ended)`.

E3: Delay sweep. Repeat the replay with the 30/50/70 ms delays used for the paper revision. The expected results are summarized in `artifact_available/docs/results_summary.md`.

A.5 Reusability and Version

The scanner is intentionally heuristic rather than sound or claim-complete, so it is most useful as a regression aid for future `everest-core` versions and nearby EVSE source trees. The replay harness exposes timing and environment parameters such as `--delay-ms`, `--base`, `--run-dir`, and `--port` so that it can be integrated into other local testbeds without source edits.

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing, and badging methodology for this artifact evaluation can be found at https://secartifacts.github.io/vehiclesec2026/.

A Study on Extending the TARA Methodology for Physical AI : A Case Study of ADS

Jihun Woo
Korea Testing & Research Institute

Junhyeong Lee*
Korea Testing & Research Institute

Abstract

Automated Driving Systems (ADS) integrate complex Physical AI, rendering traditional software-centric threat modeling inadequate against emerging AI and physical-layer vulnerabilities. We propose an extended Threat Analysis and Risk Assessment (TARA) methodology, compliant with ISO/SAE 21434, that unifies STRIDE, MITRE ATLAS, and OWASP Top 10 for LLMs. Driven by a newly constructed Attack Library, our framework systematically captures multi-layered cyber-physical threats. Applied to an ADS case study on the NVIDIA Jetson platform, the method yields 38 security checklists. These outputs cover all 32 UN R155 Annex 5 Part A threat categories, identify 29% more AI-specific risks than a STRIDE-only baseline, and achieve ML 3 for data-source integration under D2TARA. Our key contributions are: an integrated Physical AI TARA Methodology, a comprehensive Attack Library synthesized from CVEs, CWEs, and Papers, and 38 actionable security checklists validated against UN R155 Annex 5.

1 Introduction

Recently, Physical AI security has been rapidly gaining attention in both academia and industry. Physical AI refers to artificial intelligence systems that directly interact with and influence the physical world through sensor-actuator loops, making autonomous decisions based on real-time perception of their environment. Unlike traditional AI systems that operate purely in digital domains, Physical AI systems are inherently cyber-physical, where algorithmic vulnerabilities can manifest as tangible safety hazards in the real world. This increased attention stems from an increasing number of cases where small manipulations of physical signals or environmental conditions lead to serious safety risks in AI-based perception and decision-making. For example, research has shown that laser-induced acoustic interference can disrupt MEMS (Micro Electro Mechanical Systems) gyroscopes and break camera stabilization. These types of physical-cyber attacks are highly effective; experiments have confirmed that the object detection performance (mAP) of vision-based vehicles and drones can drop to as low as 41%. [1]

Among various Physical AI systems, Automated Driving Systems (ADS) serve as an exemplary testbed for several reasons:

- They integrate multiple sensing modalities (cameras, LiDAR, radar) that are vulnerable to physical-layer attacks
- They operate in safety-critical environments where security failures directly impact human lives
- They are subject to regulation (e.g., UN R155) that mandate systematic security analysis throughout the vehicle lifecycle.

Conventional threat modeling techniques have evolved primarily to focus on software and IT systems. [2–4] However, these traditional approaches do not fully meet the Security-by-Design requirements and safety goals of Physical AI. [2–4] In the context of ADS, this limitation manifests across multiple dimensions. Conventional threat models such as STRIDE primarily address software vulnerabilities but overlook adversarial threats targeting AI perception and decision-making models. Moreover, they do not systematically account for physical-layer attack vectors such as sensor spoofing or environmental manipulation that can compromise the perception-planning-control loop. Additionally, emerging AI components such as LLM introduce new attack surfaces—including prompt injection and context manipulation—that fall outside the scope of traditional IT security frameworks. [5–8]

To address these challenges, this study proposes an extended threat modeling methodology based on the ISO/SAE 21434 standard TARA process. Specifically, we employ a multi-layered approach: STRIDE for traditional software and network vulnerabilities, MITRE ATLAS for adversarial threats to AI/ML models, and OWASP Top 10 for LLM Applications to capture risks in LLM-integrated components. [7–9]

*Corresponding author

By synthesizing these frameworks into a unified attack library, our approach enables systematic analysis of threats spanning IT infrastructure, AI perception models, and emerging LLM-based decision systems, while explicitly considering how physical-layer attacks can propagate through the ADS architecture.

The contributions of this study are listed as follows:

- **Extended TARA Methodology for Physical AI:** We propose a unified framework that integrates STRIDE, MITRE ATLAS, and OWASP Top 10 for LLM Applications into the ISO/SAE 21434 lifecycle to systematically address AI-specific threats such as adversarial examples and model poisoning. Our mapping selectively applies frameworks by asset functionality and is extensible to vehicle systems integrating automation, AI, or LLMs.

- **Construction of Integrated Attack Library:** We synthesize a comprehensive attack library of 89 normalized attack scenarios from CVEs, CWEs, and Papers. This structured knowledge base facilitates the identification of Physical AI threat scenarios and the analysis of cyber-physical attack paths. Unlike proprietary industry libraries, this publicly released artifact covers sensor, perception, and LLM threats and maps them to STRIDE, MITRE ATLAS, and OWASP Top 10 for LLMs.

- **Practical Validation with Security Checklists:** We derive 38 actionable security checklists through an ADS case study based on ISO/TS 5083 and NVIDIA Jetson Platform Services, of which 11 address AI-specific threats beyond STRIDE. These ADS case-study checklists cover all 32 UN R155 Annex 5 Part A threat categories and provide auditable evidence for Vehicle Type Approval and Cyber Security Management System assessments.

2 Background

2.1 UN Regulation No. 155

UN Regulation No. 155 (UN R155) is a vehicle cybersecurity regulation established by the World Forum for Harmonization of Vehicle Regulations (UN/WP.29) [10]. The regulation mandates that Original Equipment Manufacturers (OEMs) establish a Cyber Security Management System (CSMS) covering the entire vehicle lifecycle—from development to post-production phases—to obtain Vehicle Type Approval (VTA) [10, 11].

UN R155 designates ISO/SAE 21434 as the primary technical guideline for these security activities [10]. Specifically, paragraph 5.3.1 (a) stipulates "knowledge of automotive risk assessment" as a competency requirement for approval authorities and technical services, explicitly citing ISO/SAE 21434

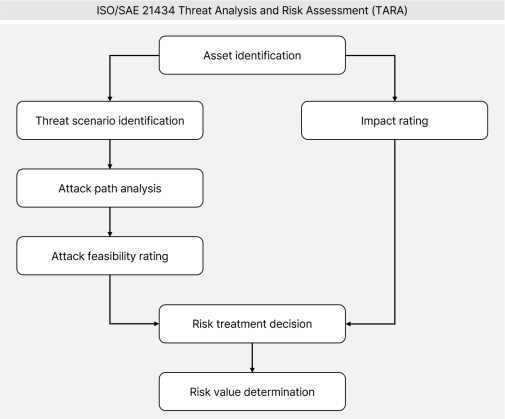


Figure 1: The TARA process on ISO/SAE 21434 [12]

in the accompanying footnote [10]. This implies that conducting Threat Analysis and Risk Assessment (TARA) and implementing security designs in compliance with ISO/SAE 21434 are critical criteria required for passing actual regulatory certification audits.

2.2 ISO/SAE 21434 TARA

ISO/SAE 21434 defines the standard for cybersecurity threat management and requirements in road vehicles [12]. The standard mandates the execution of Threat Analysis and Risk Assessment (TARA) to identify threats and evaluate associated risks. It outlines seven distinct components (or steps) for conducting the TARA process [12].

However, among these seven components, the Attack Feasibility Rating and Risk Value Determination are excluded from the scope of this study, as their evaluation criteria are highly dependent on specific organizational policies and risk appetites. Consequently, this section briefly reviews the remaining five components, which serve as the structural foundation for our extended methodology. [12].

2.3 Related Work: Research on Extending Threat Modeling

Active research is currently underway to integrate or extend existing cybersecurity threat analysis techniques into new contexts. Ghosh et al. [2] proposed a method for autonomous vehicle perception systems that integrates STPA-Sec and STRIDE based on the TARA process. This study derives causal scenarios from a safety perspective using STPA-Sec and identifies potential attack points from a security perspective. Furthermore, it links these findings to the threat analysis phase of ISO/SAE 21434 by mapping the results of STPA-Sec to the STRIDE attack classification scheme [2].

In the domain of AI-integrated systems, Jedrzejewski et al. [7, 8] pointed out that the attack surface of Large Lan-

guage Model Integrated Applications (LIA) is significantly expanded compared to traditional software. They analyzed that existing threat modeling tools are primarily manual-intensive and fail to sufficiently reflect threats specific to LLMs. To overcome this, they proposed utilizing AI-specific security knowledge bases such as OWASP Top 10 for LLM Applications and MITRE ATLAS [7, 8]. Additionally, they introduced a RAG (Retrieval-Augmented Generation) based tool that integrates existing threat models with system architecture information to continuously generate and update threat models. Their research aims to reduce the intervention of security experts in the threat modeling process, thereby saving time and costs [7, 8].

Khalil et al. [4] noted that threat modeling, which is widely used in software development projects, has not yet been systematized in the Cyber-Physical Systems (CPS) domain. This is because identifying threats in a CPS environment is complex due to the continuous interaction between cyber and physical spaces. Accordingly, their study proposed detailed procedures and step-by-step guides for applying the STRIDE technique to CPS [4]. Through a CPS case study, they demonstrated that systematic threat analysis is feasible in this field. However, this approach is limited in that it does not account for the specific threat factors of Physical AI systems that incorporate complex AI algorithms [4].

While preceding studies have laid the groundwork for extending threat modeling to specific CPS or AI applications, our research distinguishes itself by proposing a holistic framework that simultaneously addresses the triadic nature of Physical AI: IT infrastructure, AI algorithmic logic, and physical interaction. The threat scenarios are systematically derived from a unified Attack Library that synthesizes STRIDE, MITRE ATLAS, and the OWASP Top 10 for LLM Applications. It further extends prior efforts by rigorously validating the derived security requirements against UN Regulation No. 155, thereby bridging the gap between academic methodology and industrial compliance to support the actual vehicle type approval (VTA) process.

2.4 Focus Group Interview

ISO/SAE 21434 references several threat analysis frameworks applicable to threat scenario identification, including STRIDE, EVITA, TVRA, and PASTA [12]. STRIDE [13] classifies threats via data flow diagram analysis, while EVITA [14], TVRA [15], and PASTA [16] employ attack-tree- and risk-centric workflows for security-requirement derivation.

To ground our methodological design in industry practice, we conducted a focus group interview (FGI) with four automotive cybersecurity experts, each having 5–15 years of experience in automotive cybersecurity. The panel comprised two practitioners from a vehicle OEM and two engineers from a Technical Service organization accredited for UN R155 Vehicle Type Approval and CSMS conformity assessments. Two consistent themes emerged: (i) STRIDE — particularly the STRIDE-per-element variant — is the dominant framework used during threat scenario identification, and (ii) STRIDE alone is insufficient for capturing AI-specific threats, with the panel favoring incremental extension of the existing STRIDE-based workflow over replacement by an alternative end-to-end methodology.

These findings directly inform our design: STRIDE is retained as the universal baseline Section 3.3.2, and Physical AI–specific frameworks — MITRE ATLAS and OWASP Top 10 for LLM Applications — are selectively layered on top by asset archetype. This design principle also motivates the choice of a STRIDE-only reference baseline in the comparative analysis presented in Section 5.2.

3 Extended TARA Methodology

3.1 Overview of the Extended Approach

The extended TARA methodology proposed in this study is designed to reflect the multidimensional security characteristics of Physical AI systems while strictly adhering to the ISO/SAE 21434 standard process [12]. Figure 2 illustrates the unified workflow, highlighting the Attack Library Construction and multi-framework integration phases that constitute the core extensions of our methodology.

Traditional TARA approaches primarily rely on a single framework, such as STRIDE [13], to identify threats during the early lifecycle stages [17]. However, Physical AI systems—characterized by the tight coupling of IT infrastructure, AI algorithmic logic, and continuous physical interaction—present a complex attack surface that conventional software-centric models fail to capture exhaustively [2–4]. To overcome these limitations, this study introduces an integration of three distinct security frameworks.

Initially, the process begins with the construction of a comprehensive Attack Library, which systematically collects and normalizes diverse attack cases from CVEs [18], CWEs, Papers, and Technical Reports [19]. During this phase, each attack is abstracted into a structured attribute vector—comprising Attack Vector, Target, and Consequence—forming a reusable knowledge base that ensures the analytical rigor of subsequent steps.

The core of our integration strategy lies in the selective and parallel application of frameworks based on asset functionality. Unlike conventional approaches that limit framework scope, our methodology applies the STRIDE model across all identified assets to establish a consistent security baseline [17]. This ensures that fundamental security property violations—such as spoofing, tampering, and denial of service—are exhaustively addressed regardless of the asset type.

Building upon this baseline, specialized frameworks are layered according to the asset's archetype. For assets involving sensing and perception logic, the MITRE ATLAS

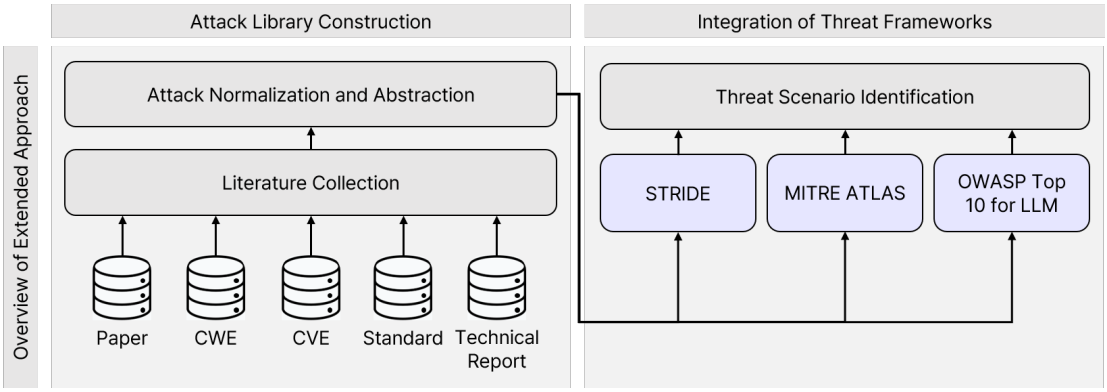


Figure 2: Extended Approach

framework [20] is integrated to identify adversarial threats, including sensor spoofing [21] and model evasion [22]. For assets characterized by reasoning or interaction via Large Language Models (LLMs), the OWASP Top 10 for LLM Applications [23] is additionally mapped to detect interface-specific risks like prompt injection [24, 25]. This modular strategy allows for a granular threat analysis that spans the entire perception-planning-control loop. Ultimately, these derived threat scenarios lead to structured Attack Path Analysis and the formulation of Security Requirements, with practical applicability validated against global regulatory standards such as UN R155 [10, 11].

3.2 Attack Library Construction

To ensure a comprehensive threat analysis for Physical AI systems, we construct a structured Attack Library by synthesizing multi-domain security knowledge. This process transforms fragmented vulnerability data into systematic attack scenarios through two phases: systematic collection and formal normalization [19].

3.2.1 Literature Collection

The collection process targets authoritative sources to capture the unique threat landscape of Physical AI, encompassing both traditional software vulnerabilities and AI-specific adversarial maneuvers. Initially, vulnerability databases such as CVE [18] and CWE are utilized to identify real-world exploitable weaknesses. To ensure practical applicability to the target architecture, the search strategy employs an approach that combines functional keywords (e.g., "Sensor Fusion", "Localization") with specific platform identifiers (e.g., "NVIDIA Jetson", "Linux Kernel") derived from the asset identification phase.

To address specialized AI threat vectors, taxonomies such as MITRE ATLAS [20] and OWASP for LLM [23] are integrated to capture techniques like data poisoning and model evasion. Furthermore, papers are analyzed to include state-of-the-art attacks validated in academic settings, such as sensor-spoofing [1, 21, 26] and perception-layer adversarial attacks [22, 27]. This systematic collection ensures that the library is grounded in both the theoretical rigor of AI security research and the practical reality of the deployed hardware and software stack.

3.2.2 Attack Normalization and Abstraction

The raw literature collected from diverse sources often contains redundant or overlapping descriptions. To transform this into a reusable and structured Attack Library, we perform normalization based on the core attributes of each threat [19]. We define an attribute vector for each record r as $Attr(r) = \langle Vector, Target, Consequence \rangle$. Within this framework, *Vector* represents the attack medium or technique (e.g., physical perturbation, signal injection), *Target* denotes the functional archetype of the victim asset (e.g., sensing interface, reasoning logic), and *Consequence* signifies the resulting system state change (e.g., integrity loss, excessive agency).

We define a normalization function $A_k = \text{Norm}(r_i, r_j)$, where two records are consolidated into a single representative attack scenario A_k if they share equivalent targets and consequences while employing technically similar vectors:

$$\begin{aligned} r_i \equiv r_j \iff & (Vector_i \approx Vector_j) \\ & \wedge (Target_i = Target_j) \\ & \wedge (Consequence_i = Consequence_j) \end{aligned} \tag{1}$$

Here, $\approx$ denotes technical similarity: the same physical or logical channel, the same DFD-level target archetype, and the same consequence category. Records are consolidated only when all three criteria are satisfied. For example, different physical artifacts that induce perception misclassification are consolidated because they share the same target and consequence.

3.3 Integration of Threat Frameworks

To systematically categorize and identify threats within the Physical AI environment, we adopt a multi-layered mapping strategy that integrates traditional and AI-specific security frameworks. This approach ensures that the analysis encompasses not only conventional software vulnerabilities but also the unique adversarial attack vectors inherent in autonomous systems [2, 28].

3.3.1 Mapping Attack Library to Frameworks

The entries in our constructed Attack Library are systematically mapped to three complementary frameworks: STRIDE [13], MITRE ATLAS [20], and the OWASP Top 10 for LLM Applications [23]. In this hierarchical mapping, every identified attack is first classified under the STRIDE model to establish its fundamental security property violation, such as Spoofing, Tampering, or Denial of Service [17].

However, as traditional models often fail to capture the nuances of AI-specific threats like data poisoning or model evasion [28,29], we further layer relevant attacks with MITRE ATLAS techniques to characterize adversarial behaviors. For assets characterized by Large Language Model (LLM) integration, we additionally apply the OWASP LLM Top 10 categories to address specialized vulnerabilities such as prompt injection and excessive agency [24,25]. This multi-framework alignment ensures that each threat is analyzed through multiple lenses, providing a comprehensive view of the physical AI risk landscape.

3.3.2 Threat Scenario Identification Strategy

Based on the integrated mapping, we derive detailed threat scenarios by applying specific frameworks to their corresponding asset archetypes defined in the DFD. To maintain analytical rigor while ensuring generality, we utilize the "STRIDE-per-element" approach across all system components [19]. This establishes a universal security baseline where even AI-specific assets are first evaluated for foundational vulnerabilities.

The identification of complex Physical AI scenarios is then refined by layering specialized frameworks onto specific functional archetypes. For *Perception Assets* responsible for environmental interpretation, we focus on Physical Environment Access techniques from ATLAS to identify scenarios such as signal injection or sensor blinding [1, 21, 26]. Simultaneously, for *Reasoning Assets* employing MLLM-based logic, we identify scenarios involving context contamination and system prompt leakage using the OWASP framework [23, 25]. This strategy enables the systematic derivation of multi-stage threat scenarios—such as a physical adversarial perturbation propagating through the perception layer to induce erroneous decision-making in the planner [22, 27]—which would be overlooked by conventional, single-framework TARA methods.

4 Case Study

To demonstrate the practical applicability and comprehensive coverage of our extended TARA methodology, we conduct a detailed case study on an Autonomous Driving System (ADS). The target system is modeled based on the logical architecture defined in ISO/TS 5083 [30] and implemented using the NVIDIA Jetson Platform Services software stack [31]. The case study proceeds through a systematic workflow aligned with the ISO/SAE 21434 lifecycle.

4.1 Asset Identification

We initiated the asset identification phase by constructing Data Flow Diagrams (DFD) using the Microsoft Threat Modeling Tool. To ensure the threat model possesses both architectural validity and implementation realism, we synthesized a reference architecture from two primary sources. First, the high-level functional logic was grounded in the ISO/TS 5083 standard. We specifically adopted the "Example architecture of the intended functionality" (Figure A.1) to define the core *Perception-Planning-Control* loop, which dictates how the ADS interprets sensor data and executes maneuvers.

However, a purely functional model lacks the granularity required to identify vulnerabilities in specific software components or underlying system layers. To bridge this gap, we instantiated the ISO model using the NVIDIA Jetson Platform Services as the concrete implementation baseline; the detailed software stack for this implementation is provided in Appendix A.2. The significance of this instantiation lies in the determinism of the operating system environment—NVIDIA Jetson Linux—which dictates the specific implementation of device drivers and hardware interfaces. By fixing the OS environment, we could define assets not as abstract logic, but as concrete entities whose security boundaries are governed by Jetson-specific kernel modules and proprietary drivers. This allows the TARA process to account for real-world constraints, such as the memory management behavior of NVIDIA proprietary GPU drivers and the specific interrupt handling of Camera Serial Interface drivers. Consequently, we identified and modeled specific Device Drivers and Shared Memory/IPC mechanisms as explicit assets. This mapping enables a systematic analysis of implementation-level vulnerabilities, such as GPU memory isolation failures (e.g., CVE-2023-4969) or UEFI Management mode side-channels (e.g., AL-V-74), which are inherent to the Jetson Linux stack. This approach ensures that the resulting threat model is a high-fidelity representation of the actual deployment stack, capturing risks that arise from the hardware-software interface.

The culmination of this modeling process is the Level 2 DFD presented in Figure A.3. This diagram provides a granular view of the internal data pathways, explicitly exposing the interactions between the AI algorithms and the system bus.

Through this decomposition, we successfully identified a total of 17 critical assets—comprising 10 processes and 7 external entities—along with 5 data stores and 30 distinct data flows. These identified elements serve as the definitive targets for the subsequent threat scenario identification phase.

4.2 Threat Scenario Identification

Following the methodology proposed in Section 3, threat scenarios were systematically identified for each asset in the ADS architecture by applying the constructed Attack Library and integrated threat frameworks.

4.2.1 Attack Library Construction Results

The literature collection process utilized keywords derived from the identified assets in the DFD. Representative keywords included "Sensor", "Sensor Fusion", "Localization", "LLM", "NVIDIA Jetson", "Jetson Driver", and "Linux Kernel", which were combined to query the five source categories described in Section 3.2.1. This targeted search resulted in the collection of 42 CWEs, 76 CVEs, 47 academic papers, 3 technical reports, and 2 standards.

Following the normalization and abstraction process described in Section 3.2.2, these 170 literature sources were analyzed to extract their attack vectors, targets, and consequences. Literature sources describing attacks with similar vectors and identical targets and consequences were consolidated into single attack scenarios. Through this process, the 170 literature sources were consolidated into 89 normalized attack scenarios, forming the unified Attack Library that serves as the foundation for threat scenario identification.

For example, AL-33 and AL-55 remain separate because their target archetypes differ, whereas sticker- and paint-based sign attacks are consolidated into AL-36 because they satisfy all three criteria.

4.2.2 Framework Mapping Results

Each attack in the constructed Attack Library was mapped to STRIDE, MITRE ATLAS, and OWASP Top 10 for LLM Applications according to the mapping strategy described in Section 3.3.1. All attacks were classified under at least one STRIDE category to capture traditional security property violations. Attacks targeting AI/ML assets were additionally mapped to MITRE ATLAS tactics and techniques, while attacks specific to LLM-integrated components were further classified under OWASP Top 10 categories.

Table 1 presents representative examples from the constructed Attack Library, illustrating how attacks are characterized by their reference sources and classified across multiple threat frameworks. The complete Attack Library is available in the public repository. [37]

4.2.3 Threat Scenario Derivation

Threat scenarios were derived by systematically applying the hierarchical strategy defined in Section 3.3.2. The DFD for the ADS architecture contains 10 processes, 7 external entities, 5 data stores, and 30 data flows, each of which was analyzed for applicable threats based on their component type and the mapped Attack Library.

STRIDE per Element Application We applied the STRIDE per Element framework to all identified assets in the DFD to systematically derive potential threat scenarios. As summarized in Table 2, each DFD component type is susceptible to a specific subset of the STRIDE threat categories based on its functional role within the system.

For each asset, attacks from the Attack Library matching the applicable STRIDE categories were selected to derive threat scenarios. For example, for the External Entity E1.1 (GPS/IMU), attacks classified as Spoofing (S) or Repudiation (R) were mapped to generate threat scenarios such as "GNSS spoofing causing contamination of position, velocity, and time data" and "Failure in post-incident actor identification due to lack of logging for GNSS/IMU anomalies."

Table 3 presents examples of threat scenarios derived through STRIDE for representative assets including external entities and OS-level processes. Through STRIDE per Element application, baseline threat scenarios covering traditional software and network vulnerabilities were systematically identified for all assets in the ADS architecture.

MITRE ATLAS Application for AI Assets For AI-related assets that perform perception, prediction, and planning functions, MITRE ATLAS was additionally applied to capture adversarial threats specific to AI/ML systems. These assets—including P2.2.3 (Interpretation & Prediction) and P2.3 (Planning)—rely on DNN-based inference and are therefore exposed to attacks such as adversarial examples, data poisoning, and model extraction that fall outside the scope of traditional STRIDE categories.

Attack Library entries classified under ATLAS tactics were mapped to these AI assets according to their relevance. For instance, attacks categorized as "Physical Environment Access" were applied to perception assets vulnerable to sensor-level manipulation, while "Poison Training Data" attacks were mapped to assets where model integrity could be compromised during the training phase.

Table 4 illustrates AI-specific threat scenarios identified for the P2.2.3 asset using MITRE ATLAS. Examples include "LiDAR point cloud injection inducing phantom object detection," "Traffic sign misclassification via physical adversarial stickers," and "DNN misclassification via FGSM/PGD adversarial examples." Through ATLAS application, comprehensive threat scenarios covering the adversarial threat landscape

Table 1: Examples : Attack Library

ID	Attack	Ref.	STRIDE	ATLAS	OWASP
AL-33	Injecting laser points to manipulate LiDAR point clouds, generating non-existent obstacles and misleading DNN-based object detectors.	[21]	S	Physical Environment Access	N/A
AL-36	Applying graffiti or stickers to a STOP sign to perform a physical adversarial attack that causes computer vision systems to misclassify the traffic sign.	[22] [32]	T	Physical Environment Access	N/A
AL-43	Executing indirect prompt injection by embedding hidden instructions in external content to manipulate the behavior of an LLM.	[24]	T	LLM Prompt Injection	LLM01 Prompt Injection
AL-51	CAN bus flooding attack — causing denial of service by monopolizing the bus using high-priority message IDs.	[33]	D	N/A	N/A
AL-52	Forged CAN message injection — remotely controlling vehicle functions via compromised telematics or gateway ECUs by injecting spoofed CAN frames.	[34] [35]	S	N/A	N/A
AL-55	Inaudible ultrasonic command injection — sending ultrasonic signals to issue hidden commands and manipulate voice assistants.	[36]	S	Physical Environment Access	N/A

Table 2: STRIDE per Elements

Threat	Data Flows	Data Stores	Processes	External Entities
Spoofing			✓	✓
Tampering	✓	✓	✓	
Repudiation			✓	✓
Information Disclosure	✓	✓	✓	
Denial of Service	✓	✓	✓	
Elevation of Privilege			✓	

of AI perception and planning modules were systematically derived.

OWASP Top 10 Application for LLM Assets For LLM-integrated assets, specifically P2.6.2 (MLLM), OWASP Top 10 for LLM Applications was employed to address vulnerabilities unique to large language models. The MLLM asset involves natural language interaction, multimodal context processing, and external knowledge retrieval via Retrieval-Augmented Generation (RAG), introducing attack surfaces not adequately covered by STRIDE or ATLAS alone.

Attack Library entries classified under OWASP Top 10 categories were mapped to the MLLM asset. Representative threat categories include Prompt Injection, which encompasses both direct and indirect manipulation of model behavior through crafted inputs; System Prompt Leakage, involving extraction of internal instructions that govern model behavior; Excessive Agency, where insufficient privilege constraints enable unauthorized actions; and Vector and Embedding Weaknesses, which cover attacks exploiting the semantic representation space of LLMs.

Table 5 demonstrates LLM-specific threat scenarios derived from OWASP Top 10 for the P2.6.2 asset. Examples include "Indirect prompt injection via ingestion of untrusted external content," "Malicious fine-tuning removing RLHF-based safety guardrails," and "Sensitive data reconstruction via embedding inversion attacks." Through OWASP Top 10 application, the unique threat landscape of LLM-integrated decision-making components was comprehensively captured.

4.2.4 Threat Scenario Identification Result

Through the hierarchical application of STRIDE, MITRE ATLAS, and OWASP Top 10 for LLM, a comprehensive set of threat scenarios was systematically identified across the ADS architecture. The methodology successfully captured threats spanning traditional IT security concerns, AI-specific adversarial attacks, and emerging LLM vulnerabilities, providing a holistic view of the Physical AI threat landscape.

The complete set of identified threat scenarios, including detailed descriptions and framework mappings, is available in the public repository. [37] These threat scenarios serve as the foundation for subsequent Impact Rating (Section 4.3) and Attack Path Analysis (Section 4.4), ultimately leading to the derivation of security requirements and actionable checklists for ADS development and deployment.

Table 3: Example : Threat Scenarios for the OS Asset based on STRIDE

Asset	No.	Threat Scenario	STRIDE	AL Ref.
P2.1. OS	T94	Corruption of kernel data and memory structures by exploiting vulnerabilities in the Linux CAN Broadcast Manager.	T	AL-62
	T105	Modification of configuration, log, or module files through Path Traversal vulnerabilities.	T	AL-29
	T106	Corruption of kernel data and memory structures by exploiting vulnerabilities in the Linux CAN Broadcast Manager.	T	AL-16

Table 4: Example : Threat Scenarios for the Planning Asset based on MITRE ATLAS

Asset	No.	Threat Scenario	ATLAS	AL Ref.
P2.3. Planning	T558	Induction of maladaptive behaviors (e.g., abrupt braking, lane deviation) in path planning algorithms via training data poisoning.	Poison Training Data	AL-40
	T559	Replication of the internal planning model and reverse engineering of vulnerabilities via exploitation of the model inference API.	Exfiltration via AI Inference API	AL-60
	T560	Induction of erroneous decision-making in the AI planning system via manipulation of external visual semantics.	Physical Environment Access	AL-36

4.3 Impact Rating

Following the identification of threat scenarios, we evaluated the potential adverse consequences on road users and ADS stakeholders. As stated in our scope, we do not perform a numerical scoring of the impact levels (e.g., Negligible to Severe), as such ratings are highly subjective and contingent upon specific organizational risk appetites and deployment contexts. Instead, we focus on the systematic derivation of damage scenarios and their qualitative categorization (Safety, Financial, Operational, and Privacy) to establish a clear causal link between AI-specific threats and vehicle-level hazards.

4.3.1 Damage Scenario Derivation

The damage scenarios were derived by analyzing how each identified threat (from STRIDE, ATLAS, and OWASP) propagates through the ADS data flow to compromise the system's primary goals. As documented in our artifacts, we identified a total of 165 damage scenarios (DS-1 to DS-165). Each scenario represents a physical or operational failure resulting from one or more threat scenarios.

For instance, the damage scenario "Incorrect path planning and lane departure" (DS-1) is not merely a software bug but a consequence of a specific chain of AI threats: GNSS spoofing (T1, T2) or authentication bypass of location data (T8). Similarly, the "Unintended braking" (DS-4) scenario is mapped to perception-layer threats such as ghost object detection via sensor spoofing (T11-T19). This mapping ensures that every identified damage has a traceable root cause in our expanded Attack Library.

4.3.2 Impact Categories

In accordance with ISO/SAE 21434, each damage scenario was categorized based on its impact domain:

- **Safety (S):** Scenarios leading to physical hazards, such as collisions due to manipulated object tracks (DS-157).
- **Financial (F):** Impacts related to liability and economic loss, often linked to the failure of non-repudiation and logging mechanisms (DS-3).
- **Operational (O):** Disruption of ADS availability, such as positioning service outages (DS-2).
- **Privacy (P):** Leakage of sensitive data, particularly relevant in MLLM-based systems where user intent or biometric data can be exposed (DS-165).

4.3.3 Impact Rating Results

Table 6 presents the representative mapping between assets, damage scenarios, and their associated threat scenarios. The analysis reveals that Physical AI threats (e.g., LiDAR/Radar spoofing) predominantly converge on Safety impacts, while LLM-specific threats (e.g., Prompt Injection) introduce a complex mix of Privacy and Operational risks, highlighting the necessity of the integrated TARA approach proposed in this paper.

The analysis revealed that while AI-specific threats (ATLAS, OWASP) often originate in the digital domain, their

Table 5: Example : Threat Scenarios for the MLLM Asset based on OWASP Top 10 for LLMs

Asset	No.	Threat Scenario	OWASP	AL Ref.
P2.6.2. MLLM	T536	Exposure of the System Prompt (internal constraints and instructions), enabling adversaries to reverse-engineer and circumvent safety policies.	System Prompt Leakage	AL-46
	T537	Ablation of RLHF-based safety guardrails via malicious Fine-tuning, aimed at eliciting restricted or harmful outputs.	Data and Model Poisoning	AL-49
	T538	Autonomous execution of high-risk commands resulting from the exploitation of Excessive Agency and insufficient scope limitation.	Excessive Agency	AL-45

propagation to the physical domain (Safety Impact) is significant in ADS, justifying the need for high-integrity protection measures in the subsequent Risk Treatment phase.

4.4 Attack Path Analysis

An attack path is defined as a series of deliberate actions undertaken to realize a mapped threat scenario [12]. In this study, attack paths were derived by establishing the previously identified threat scenarios and damage scenarios as the respective endpoints, and detailing the specific attack actions connecting them based on the Attack Library. Specifically, the construction of these attack paths comprehensively accounted for asset characteristics, adversary intent, and the technical means required for execution.

First, we identified the preconditions and entry points an adversary must secure to realize the threat scenario mapped to each asset. Next, we ensured the technical realization of the attacks by integrating specific techniques from the Attack Library—which originally grounded the derivation of these threat scenarios—as the core execution steps along the path. Finally, we completed the causal chain linking these attack actions to the ultimate damage scenarios by analyzing their functional impact on the system. Table 7 presents examples of the attack paths derived for each threat scenario through this analytical process. The comprehensive set of attack paths can be accessed via our public repository. [37]

4.5 Derivation of Security Requirements

In the standard ISO/SAE 21434 TARA process, the risk treatment phase determines the appropriate mitigation strategy for each identified threat scenario based on an organization's specific evaluation criteria and risk appetite. When the "Reducing the Risk" option is selected, technical and organizational controls must be established, which necessitates the derivation of precise security requirements. These requirements subsequently serve as the foundational elements for defining the overarching Cybersecurity Goals in the later stages of the engineering lifecycle.

However, consistent with our methodological scope—which abstracts away organization-specific risk evaluation criteria—we bypass the subjective risk treatment decision step. Instead, we proactively derive security requirements for *all* identified threat scenarios to ensure comprehensive coverage against the expanded threat landscape.

The security requirements are systematically synthesized based on the causal chain established in the previous phases: the threat scenarios, damage scenarios, attack paths, and the underlying Attack Library mapped to each asset. To optimize the mitigation strategy and prevent redundancy, threat scenarios that necessitate identical protective measures are clustered and mapped to a unified security requirement. This consolidation process not only streamlines the defense architecture but also provides the foundational artifacts necessary for constructing a comprehensive security checklist for Physical AI systems.

Table 8 presents representative examples of the derived security requirements. The complete set of requirements is accessible via our public repository. [37]

4.6 Derivation of Checklist

In this section, we consolidated the security requirements derived across all assets by performing systematic merging, normalization, and deduplication. This process ensures technical consistency and culminated in the final derivation of a comprehensive security checklist comprising 38 distinct items. Representative items from the checklist are presented in Appendix A.4; the complete 38-item checklist is available in our artifact [37].

The primary objective of this transformation is to operationalize abstract security requirements into concrete, quantifiable verification criteria, thereby eliminating ambiguity during the system implementation phase. Furthermore, this checklist is designed to serve as objective, verifiable evidence demonstrating the practical application of security controls during future CyberSecurity Management System (CSMS) audits and Vehicle Type Approval (VTA) processes.

Table 6: Representative Damage Scenarios and Impact Categories

Asset	No.	Damage Scenario	CIA	SFOP	Threats ID
P2.1. OS	DS-27	Collapse of security boundaries via privilege escalation or isolation bypass, resulting in persistent manipulation, data exfiltration, and the neutralization of safety functions, ultimately causing hazardous driving or collisions.	I/A	S	T90, T91, T92, T93, T94, T105, T106, T107, T108
	DS-30	Exposure of sensitive data resulting in privacy violations, security breaches, and regulatory non-compliance.	C	P	T97, T98, T99, T100
P2.3. Planning	DS-47	Imminent risk of collision or pedestrian injury caused by erratic maneuvers (e.g., inappropriate braking, abrupt steering, lane departure).	I/A	S	T558, T560, T561, T565, T567, T568, T571
	DS-50	Compromise of user privacy facilitating unauthorized surveillance, tracking, and targeted social engineering attacks.	C	P	T559, T563, T566, T572
P2.6.2. MLLM	DS-68	Induction of malformed vehicle control messages or invalid authorization codes via the generation of hazardous commands by the MLLM.	I	S	T576, T577, T578, T581, T582, T583, T592, T534, T535, T538, T543
	DS-70	Execution of restricted operations resulting from the circumvention of safety guardrails and misuse of privileges.	I	O	T573, T574, T584, T593, T537, T541

5 Validation

In this section, we validate the effectiveness and applicability of the proposed extended TARA methodology through four distinct dimensions: (1) Regulatory Compliance, demonstrating alignment with UN Regulation No. 155; (2) Comparative Analysis, quantifying improvements over traditional threat modeling; (3) Maturity Assessment, evaluating the data-driven robustness of the process; and (4) Real-World Validation, establishing institutional accreditation as a basis for accredited testing activities.

5.1 Legal Coverage

The derived security checklists must transcend academic theoretical proposals and serve as practical guidelines in the industrial field. To verify this applicability, we analyzed the coherence between our 38 derived checklists and the technical requirements of UN Regulation No. 155 (UN R155), the mandatory international standard for vehicle cybersecurity.

Validation Criteria: Annex 5. We selected UN R155 Annex 5 as the baseline for validation. The regulation mandates that manufacturers consider the threats listed in Annex 5 Part A and the mitigations in Part B and C during risk assessment. Therefore, coverage of Annex 5 Part A by the ADS case-study checklists indicates that the methodology can generate artifacts useful for demonstrating regulatory alignment.

Mapping and Conformity Analysis. We performed a semantic mapping between the derived checklists and the sub-level threats defined in Annex 5 Part A. The results in Table B.1 confirm that the ADS case-study checklists cover both general embedded-system threats and AI- and sensor-specific threats in Annex 5 Part A.

The 38 ADS case-study checklists collectively cover all 32 Annex 5 Part A threat categories, including emerging AI-related vulnerabilities. As illustrated in the mapping results, the derived checklists correspond to the detailed threat categories of Annex 5. For instance, checklists addressing "Sensor Spoofing" and "Adversarial Examples" directly map to the regulation's requirements for verifying sensor data integrity. This mapping indicates that the ADS case-study checklists can support compliance evidence for Vehicle Type Approval (VTA) audits and Cyber Security Management System (CSMS) assessments.

5.2 Comparative Analysis

To evaluate the improvement of the proposed methodology, we conducted a comparative analysis against the traditional STRIDE-based TARA approach using the same ADS architecture. Per Section 2.4, STRIDE-per-element is the dominant baseline in industrial ISO/SAE 21434 TARA practice, making it the most realistic comparison target for evaluating practical improvements.

Qualitative Analysis: Granularity of Threats. The primary distinction lies in the granularity and scope of threat identification. While the traditional STRIDE model is effective for identifying software and network vulnerabilities based on data flow, it exhibits structural limitations in characterizing threats related to AI inference and physical sensor data. For

Table 7: Representative Attack Paths Mapping Threat Scenarios to Damage Scenarios

Asset	Damage Scenario	Threat Scenario	Attack Path
P2.1. OS	DS-27	T105	i. Reconnaissance: The attacker analyzes the system's privilege management policies and default configurations. ii. Vulnerability Discovery: Excessive default privileges or incorrect permission assignments are identified. iii. Exploitation (Escalation): The attacker utilizes privilege escalation techniques from a low-integrity process to acquire administrative (root) privileges [AL-29]. iv. Action on Objectives: With elevated privileges, the attacker bypasses all security boundaries, neutralizes safety systems, and establishes remote control over the vehicle.
P2.3. Planning	DS-47	T560	i. The attacker modifies the physical environment with adversarial stickers or paint. ii. These visual changes alter the semantic meaning perceived by the AI [AL-36]. iii. The planner generates a trajectory that follows the manipulated semantics (e.g., driving into a wall).
P2.6.2. MLLM	DS-68	T537	i. The attacker gains access to the model's fine-tuning pipeline. ii. A malicious dataset designed to reverse safety alignment is used for fine-tuning [AL-49]. iii. The deployed model unlearns its safety constraints and executes previously restricted commands.

Table 8: Representative Security Requirements mapped to Threat Scenarios

Asset	ID	Security Requirement	Threats ID
P2.1. OS	SR-37	A hardware-rooted Secure Boot chain shall be implemented to verify the integrity of the bootloader and kernel.	T92, T103, T105, T108
P2.3. Planning	SR-62	Adversarial defense mechanisms and input sanitization shall be implemented to detect semantic perturbations and prompt injections.	T169, T172, T560, T561, T565, T567, T569
P2.6.2. MLLM	SR-89	Supply chain security controls including model signing and backdoor scanning shall be applied to all AI artifacts.	T248, T259, T260, T537, T541, T573, T574, T584, T593

example, under STRIDE, a "Poisoned Model" or an "Adversarial Example" is often broadly categorized as generic "Tampering" or "Information Disclosure," failing to distinguish the specific attack vector. In contrast, our extended methodology, by integrating MITRE ATLAS and OWASP Top 10 for LLM, successfully decomposed these vague categories into specific Physical AI threat scenarios such as *Evasion*, *Model Inversion*, and *Prompt Injection*.

Quantitative Analysis: Expanded Coverage. The quantitative results further highlight the efficacy of the proposed approach. Both methods identified standard cybersecurity threats such as network eavesdropping and unauthorized access. However, the proposed methodology exclusively identified specialized threats related to **AI & Sensor Data**, involving sensor fusion manipulation and model bias (e.g., CL-19, CL-20, CL-25 to CL-27), and **LLM Integration**, involving prompt injection and context poisoning (e.g., CL-24, CL-28, CL-29, CL-31 to CL-33). The proposed methodology exclusively identified 11 AI-specific checklist items that could not be derived through conventional STRIDE-based TARA alone—accounting for approximately 29% (11 out of 38) of the total derived checklists. These items address threat categories that are structurally absent from traditional software-centric frameworks, including sensor fusion manipulation, model bias exploitation, adversarial perturbation attacks, prompt injection, and LLM context poisoning. The fact that nearly one-third of the security controls emerged only through the integration of MITRE ATLAS and OWASP Top 10 for LLM Applications underscores a critical blind spot in conventional TARA methods when applied to Physical AI environments: a system analyzed solely with STRIDE would have left these attack surfaces unaddressed, potentially resulting in incomplete security coverage and non-compliance with emerging regulatory expectations for AI-integrated vehicles.

5.3 Maturity Assessment

This section evaluates the data integration capabilities and methodological maturity of the proposed approach by applying the Data-Driven TARA Maturity Model (D2TARA) proposed by Vielberth et al. [38]. The D2TARA model assesses the Capability Level (CL) of a TARA process based on the extent to which external data sources are standardized and integrated into the workflow. Accordingly, this assessment is limited to data-source standardization and integration, not organizational maturity.

Standardization (CL 2). We established a comprehensive Standard Pool of External Data Sources by systematically constructing an Attack Library that incorporates CVE, CWE, Papers, Technical Reports, and Standards. This collection strictly satisfies the requirements for *Standardization (CL 2)* across all data-driven TARA sections.

Integration (CL 3). The D2TARA model evaluates maturity based on data integration across five core TARA sections: *DAMAGE*, *IMPACT*, *ATTACK*, *FEASIB*, and *RISK TREAT*. [38] According to expert feedback documented in the D2TARA validation study (Section 6.2), there exists justified disagreement on whether *IMPACT* and *FEASIB* should be considered data-driven sections due to the discrepancy between ISO/SAE 21434 rating schemes and externally available metrics such as CVSS. [38] Following this expert perspective, we focus our maturity assessment on the three universally recognized data-driven sections: *DAMAGE*, *ATTACK*, and *RISK TREAT*.

Our methodology achieves CL 3 (Integration) by establishing explicit processes for utilizing external data sources in each section. The foundation of this integration is the Attack Library construction process, which systematically analyzes CVE, CWE, Papers, technical reports, and standards to extract three core attributes: Vector (attack medium and techniques), Target (affected assets), and Consequence (system state changes). Attacks sharing similar Vectors, identical Targets, and equivalent Consequences are classified into unified attack categories, which are then mapped to the taxonomies of OWASP Top 10 for LLM, STRIDE, and MITRE ATLAS.

For the **DAMAGE** section, damage scenarios are derived through a structured integration process. Threat scenarios are generated using STRIDE per element, OWASP categories for MLLM assets, and MITRE ATLAS for AI assets. Since each threat scenario is mapped to specific Attack Library entries through the framework alignment process, the resulting damage scenarios inherently incorporate the Consequence attributes from CVE impact descriptions and technical report findings. This ensures that damage identification is not based on expert judgment alone but is systematically grounded in documented external data sources.

The **ATTACK** section demonstrates clear integration through attack path construction. For each threat scenario, attack paths are analyzed by leveraging the Attack Library's Vector and Target attributes. Because threat scenarios are explicitly mapped to Attack Library entries, the attack path analysis directly incorporates attack techniques documented in CVE/CWE databases and MITRE ATLAS tactics. This mapping mechanism ensures that attack feasibility assessment is traceable to specific external data sources rather than being purely expert-driven.

For the **RISK TREAT** section, mitigation strategies are systematically derived from the attack paths through framework-based guidance. The integrated OWASP Top 10 for LLM and MITRE ATLAS frameworks provide structured mitigation recommendations that are mapped to security requirements. Since attack paths are connected to specific framework categories through the Attack Library mapping process, the risk treatment decisions are inherently informed by the mitigation strategies documented in these external sources. This integration ensures consistent and comprehensive risk treatment based on industry-recognized countermeasures.

Extended Maturity Level (ML 3). According to the D2TARA maturity level definitions, ML 3 (Extended) is achieved when CL 3 (Integration) is achieved in all data-driven TARA sections. Our methodology demonstrably achieves this through the systematic integration described above. The Attack Library serves as the integration hub, transforming raw external data sources into structured attributes (Vector, Target, Consequence) that flow through the entire TARA process. Threat scenarios are generated through framework mapping, damage scenarios inherit Consequence attributes, attack paths are constructed from Vector/Target information, and risk treatment leverages framework-embedded mitigation guidance. This end-to-end integration of external data sources into each TARA section fulfills the CL 3 requirement that "the pool of data sources is integrated into the TARA process" with explicit specification of how data is utilized. Therefore, the proposed methodology achieves **Extended Maturity Level (ML 3)** for data-source integration under the D2TARA model.

5.4 Real-World Validation: KOLAS Accreditation

Beyond the analytical validation dimensions presented above, the proposed methodology has been adopted as the technical procedure underlying accreditation by the Korea Laboratory Accreditation Scheme (KOLAS) as an internationally recognized testing laboratory for the ISO/SAE 21434 TARA process. This means that test reports produced through the framework can serve as formally recognized deliverables in real-world automotive cybersecurity testing practice. As KOLAS is a full signatory to the ILAC MRA and APAC MRA, this places reports produced through the methodology within an internationally acknowledged conformity assessment framework.

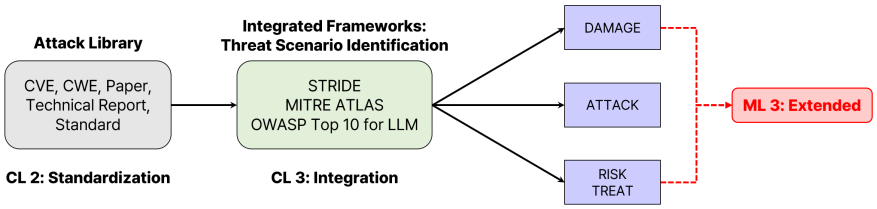


Figure 3: Proposed TARA Maturity Validation Workflow

6 Discussion

This study demonstrates that integrating STRIDE, MITRE ATLAS, and OWASP Top 10 for LLM into the ISO/SAE 21434 TARA process enables systematic and comprehensive threat identification for Physical AI systems—an area where traditional software-centric frameworks have proven structurally insufficient.

6.1 Industrial and Regulatory Significance

The proposed methodology carries direct implications for the automotive industry. As UN Regulation No. 155 mandates CSMS establishment and Vehicle Type Approval (VTA) across the entire vehicle lifecycle, manufacturers face increasing pressure to demonstrate systematic cybersecurity coverage. Our framework provides a structured, auditable process. In the ADS case study, the resulting 38 security checklists cover all 32 UN R155 Annex 5 Part A threat categories and can support regulatory audits.

6.2 Scope of Risk Assessment

A deliberate limitation of this study is the exclusion of the Attack Feasibility Rating and Risk Value Determination phases from the ISO/SAE 21434 process. These phases are highly dependent on an organization's specific risk appetite, available resources, and deployment context, making universal prescriptions impractical. While our framework comprehensively identifies and catalogs threats, practitioners adopting this methodology must define their own feasibility criteria—considering factors such as the attacker's required window of opportunity and equipment cost—to derive final risk values and prioritize mitigation efforts accordingly.

6.3 Extensibility to Broader Physical AI

Although validated on ADS, the proposed methodology exhibits high transferability to broader Physical AI domains. However, effective application to other domains requires two domain-specific adaptations: the Asset Identification phase must be tailored to the target system's physical architecture, and the Attack Library must be extended with domain-specific attack vectors. For instance, applying this framework to industrial robots would necessitate incorporating sensor spoofing techniques specific to robotic arms and manufacturing floor environments. Future work will aim to validate this transferability by applying the methodology to diverse intelligent mobility platforms beyond ADS.

7 Conclusion

This paper proposed an extended TARA methodology aligned with ISO/SAE 21434 that applies STRIDE, MITRE ATLAS, and OWASP Top 10 for LLM in a multi-layered manner to address the cyber-physical threat surface of Physical AI systems. Through an ADS case study based on the ISO/TS 5083 logical architecture and implemented on the NVIDIA Jetson platform, the framework produced 38 actionable security checklists validated against UN Regulation No. 155 Annex 5.

The contributions of this work are threefold: (i) an integrated Physical AI TARA methodology unifying three complementary security frameworks; (ii) an Attack Library synthesized from CVEs, CWEs, and academic literature, providing a structured knowledge base for cyber-physical attack path analysis; and (iii) 38 actionable security checklists demonstrating direct regulatory compliance utility for VTA and CSMS certification processes.

This work has two main limitations: the exclusion of Attack Feasibility Rating and Risk Value Determination leaves threat prioritization to adopting organizations, and the current validation is scoped to ADS on the NVIDIA Jetson platform, requiring domain-specific adaptation for broader Physical AI generalization.

Future work will pursue two complementary directions: incorporating Attack Feasibility Rating and Risk Value Determination to complete the full ISO/SAE 21434 TARA lifecycle, and applying the methodology to diverse Physical AI platforms such as autonomous robotics. AI-agent–based automation of the proposed methodology also remains a promising direction.

8 Acknowledgments

This work was supported by the Ministry of Trade, Industry and Resources (MOTIR) (No. RS-2025-13732975)

References

[1] N. Shamsi, K. Chandrasekar, Y. Long, C. Limbach, K. Rebello, and K. Fu. Wip: threat modeling laser-induced acoustic interference in computer vision-assisted vehicles. In *Symposium on Vehicles Security and Privacy (VehicleSec)*, 2024.

[2] S. Ghosh, A. Zaboli, H. Junho, and J. Kwon. An integrated approach of threat analysis for autonomous vehicles perception system. *IEEE Access*, 11, 2023.

[3] A. Chattopadhyay, K.-Y. Lam, and Y. Tavva. Autonomous vehicle: Security by design. *IEEE Transactions on Intelligent Transportation Systems*, 22(11):7015–7029, 2021.

[4] S. M. Khalil, H. Bahsi, H. O. Dola, T. Korõtko, K. McLaughlin, and V. Kotkas. Threat modeling of cyber-physical systems: A case study of a microgrid system. *Computers & Security*, 124:102950, 2023.

[5] A. Chattopadhyay and K.-Y. Lam. Security of autonomous vehicle as a cyber-physical system. In *Proceedings of the 7th International Symposium on Embedded Computing and System Design (ISED)*, pages 1–6, 2017.

[6] A. A. Ali, S. Krayem, B. Chramcov, and M. F. Kadi. Self-stabilizing fault tolerance distributed cyber physical systems. In *Proceedings of the 29th DAAAM International Symposium*, pages 1173–1180, 2018.

[7] F. V. Jedrzejewski, D. Fucci, and O. Adamov. Thremolia: Threat modeling of large language model-integrated applications. *arXiv preprint arXiv:2504.18369*, 2025.

[8] F. V. Jedrzejewski. Threat modeling of ml-intensive systems: Research proposal. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering – Software Engineering for AI*, pages 264–266, 2024.

[9] S. Dellersnyder and Toreon. Mind the Gap: STRIDE-AI - Your Clear Path to Understanding AI Vulnerabilities. https://www.toreon.com/stride-ai-your-clear-path-to-understanding-ai-vulnerabilities, 2025.

[10] UNECE WP.29. UN Regulation No. 155 – Cyber security and cyber security management system, 2021. United Nations Economic Commission for Europe.

[11] S. Lee and S. Cho. Attack scenarios based on unece automotive cybersecurity regulation (un r.155). *Transactions of KSAE*, 29(8):717–732, 2021.

[12] ISO/SAE International. Road vehicles – Cybersecurity engineering (ISO/SAE 21434:2021), 2021. Standard.

[13] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack. Uncover security design flaws using the stride approach. *MSDN Magazine*, 2006.

[14] Henniger, O., Ruddle, A., Seudié, H., Weyl, B., Wolf, M., Wollinger, T. Securing vehicular on-board it systems: the evita project. 2009.

[15] Judith E. Y. Rossebo, Scott Cadzow, and Paul Sijben. etvra, a threat, vulnerability and risk assessment method and tool for eeurope. In *The Second International Conference on Availability, Reliability and Security (ARES'07)*, pages 925–933, 2007.

[16] Moranana. M.M. UcedaVelez. T. *Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis*. 1st ed. John Wiley & Sons, 2015.

[17] Z. Abuabed, A. Alsadeh, and A. Taweel. Stride threat model-based framework for assessing the vulnerabilities of modern vehicles. *Computers & Security*, 133:103391, 2023.

[18] MITRE Corporation. CVE. https://cve.mitre.org/cgi-bin/cvename.cgi, 2025.

[19] A. Shostack. *Threat Modeling: Designing for Security*. Wiley, 2014.

[20] MITRE Corporation. MITRE ATLAS (Adversarial Threat Landscape for Artificial-Intelligence Systems). https://atlas.mitre.org/, 2021.

[21] Y. Cao, C. Xiao, B. Cyr, Y. Zhou, W. Park, S. Rampazzi, Q. A. Chen, K. Fu, and Z. M. Mao. Adversarial sensor attack on lidar-based perception in autonomous driving. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*, pages 2267–2281, 2019.

[22] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1625–1634, 2018.

[23] OWASP Gen AI Security Project. OWASP Top 10 for LLM Applications 2025. https://genai.owasp.org/llm-top-10/, 2025.

[24] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec '23)*, pages 79–90, 2023.

[25] P. Zambare, V. N. Thanikella, and Y. Liu. Securing agentic ai: Threat modeling and risk analysis for a network monitoring agentic ai system. *arXiv preprint arXiv:2508.10043*, 2025.

[26] B. Nassi, Y. Mirsky, D. Nassi, R. Ben-Netanel, O. Drokin, and Y. Elovici. Phantom of the adas: Securing advanced driver-assistance systems from split-second phantom attacks. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*, pages 293–308, 2020.

[27] H. Zhou, W. Li, Z. Kong, J. Guo, Y. Zhang, B. Yu, L. Zhang, and C. Liu. Deepbillboard: Systematic physical-world testing of autonomous driving systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20)*, pages 347–358, 2020.

[28] A. Vassilev, A. Oprea, A. Fordyce, H. Anderson, X. Davies, and M. Hamin. Adversarial machine learning: A taxonomy and terminology of attacks and mitigations (nist ai 100-2e2025). Technical report, National Institute of Standards and Technology (NIST), 2025.

[29] M. Ozdag. Adversarial attacks and defenses against deep neural networks: A survey. *Procedia Computer Science*, 140:152–161, 2018.

[30] ISO International. Road vehicles – Safety for automated driving systems – Design, verification and validation (ISO/TS 5083:2025), 2025. Technical Specification.

[31] NVIDIA Corporation. Jetson Platform Services documentation. https://docs.nvidia.com/jetson/jps/moj-overview.html, 2025.

[32] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, F. Tramèr, A. Prakash, T. Kohno, and D. Song. Physical adversarial examples for object detectors. In *Proceedings of the 12th USENIX Conference on Offensive Technologies (WOOT '18)*, 2018.

[33] S.-H. Chen and C.-H. R. Lin. Evaluation of dos attacks on vehicle can bus system. In *Proceedings of the International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP 2018)*, volume 110, pages 308–314, 2018.

[34] MITRE Corporation. CVE-2015-5611. https://cve.mitre.org/cgi-bin/cvename.cgi, 2025.

[35] MITRE Corporation. CVE-2016-9337. https://cve.mitre.org/cgi-bin/cvename.cgi, 2025.

[36] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu. Dolphinattack: Inaudible voice commands. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*, pages 103–117, 2017.

[37] J. Woo. ADS-Threat-Modeling GitHub Repository. https://github.com/ddiyoung/ADS-Threat-Modeling, 2026. GitHub repository.

[38] Manfred Vielberth, Kristina Raab, Magdalena Glas, Patrick Grümer, and Günther Pernul. Elevating tara: A maturity model for automotive threat analysis and risk assessment. In *Proceedings of the 19th International Conference on Availability, Reliability and Security*, ARES '24, New York, NY, USA, 2024. Association for Computing Machinery.

A Appendix : Case Study

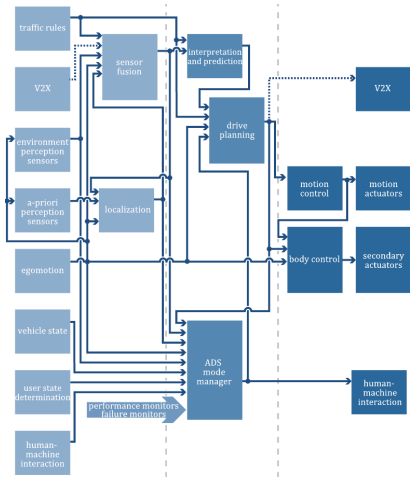


Figure A.1: Functional Architecture based on ISO/TS 5083

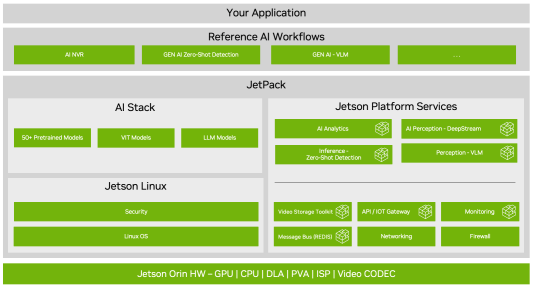


Figure A.2: NVIDIA Jetson Platform Services Software Stack

B Appendix : Validation

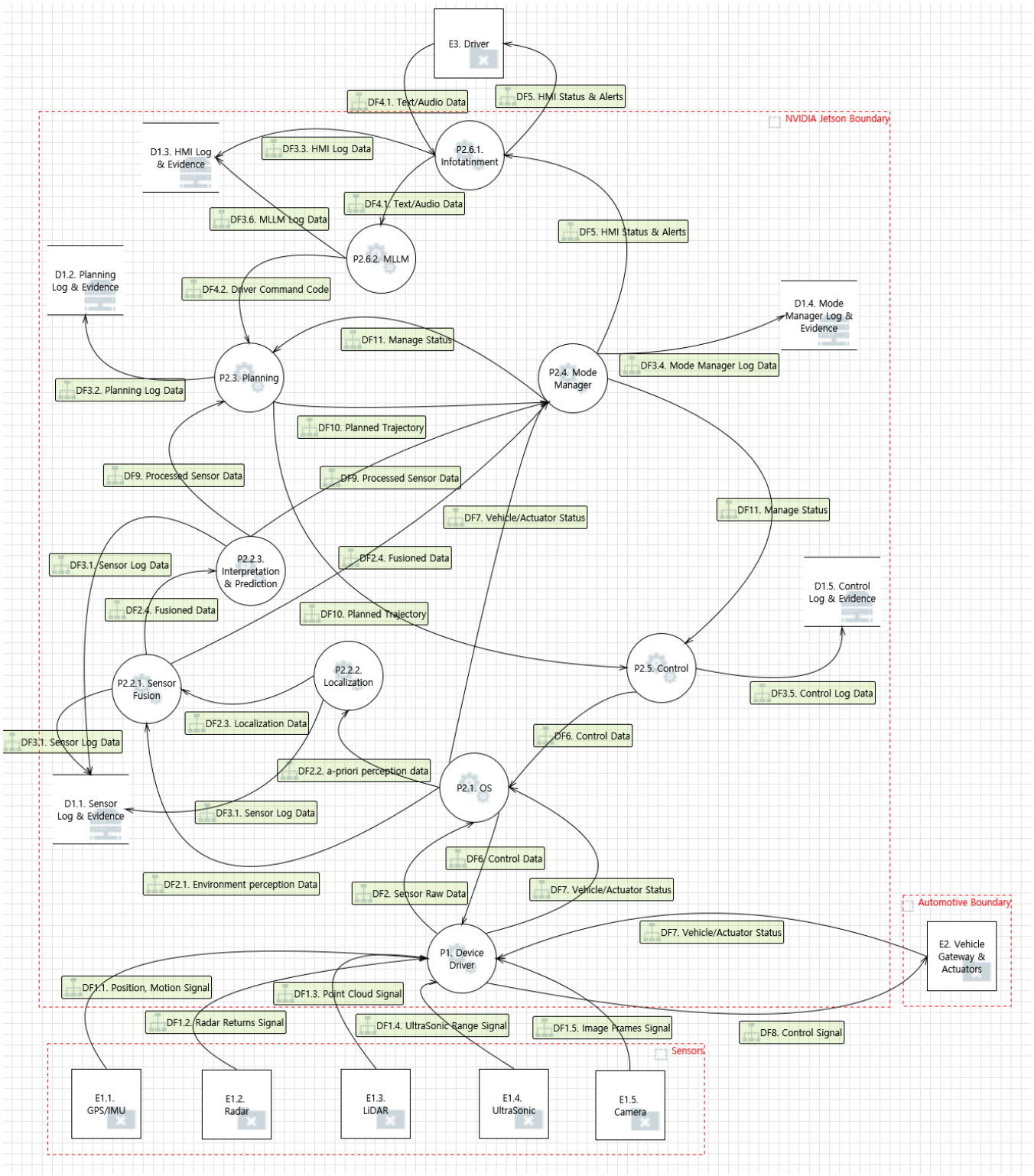


Figure A.3: Level 2 Data Flow Diagram (DFD)

Table A.4: Representative Items from the Derived Checklist

ID	Checklist Item (Security Control)	Mapped SRs
CL-01	Is data ingested from external assets subject to rigorous validation of integrity and freshness (anti-replay)? *(e.g., Implementing HMAC-SHA256 or CMAC for integrity, and verifying Timestamps or Monotonic Counters to prevent replay)*	SR-1, SR-3, SR-6, SR-7, SR-61, SR-62, ...
CL-02	Is input data accepted as trusted only when originating from registered and authenticated external assets? *(e.g., Enforcing mutual authentication via X.509 Certificates (mTLS) or validating Digital Signatures (Ed25519/ECDSA))*	SR-38, SR-43, SR-56, SR-61, SR-81, ...
CL-03	Is secure channel communication (e.g., end-to-end authentication and encryption) enforced between external assets?	SR-41, SR-53, SR-59, SR-66, SR-71, ...
CL-04	Are all events related to external assets logged at a granularity sufficient to ensure non-repudiation and enable post-incident forensic analysis?	SR-5, SR-9, SR-13, SR-17, SR-21, SR-26, ...
	...	
CL-19	Are defense functions in place to detect and mitigate spoofing or adversarial patterns at the sensor level (LiDAR, Radar, Camera, Ultrasonic, GNSS)?	SR-1, SR-6, SR-10, SR-14, SR-18, SR-27, ...
	...	
CL-24	Are Command Codes (or other AI inputs) generated by MLLMs verified for authenticity and checked for Prompt Injection and Context Contamination?	SR-62, SR-80, SR-86, ...
CL-25	Does the system possess mechanisms to detect and mitigate physical and digital adversarial examples (stickers, light sources, lasers, phantom objects, etc.)?	SR-18, SR-55, SR-62, SR-86, SR-126, ...
CL-26	Does the system possess mechanisms to detect and mitigate Transfer Attacks?	SR-44, SR-90, SR-159, SR-200
CL-27	Does the system possess mechanisms to detect and mitigate Poisoned Dataset injection?	SR-50, SR-55, SR-57, SR-62, SR-63, SR-88, SR-89
CL-28	Are inputs to the MLLM monitored to detect, verify, and block Prompt Injection, Context Contamination, and Adversarial Inputs?	SR-55, SR-62, SR-80, SR-86, SR-88, SR-194
CL-29	Does the MLLM accept input (voice, text) only from legitimate, authenticated users and processes?	SR-27, SR-79, SR-81, SR-198, SR-222
	...	
CL-32	Are sensitive data elements (user voice, text, intent, command codes, key slots) protected via encryption and strict access control mechanisms?	SR-24, SR-29, SR-38, SR-53, SR-71, SR-75, ...
CL-33	Does the system maintain operational stability without computational or memory exhaustion under conditions of excessive token input, long-context ingestion, or alert flooding?	SR-52, SR-58, SR-65, SR-76, SR-84, SR-91, ...
CL-34	Does the system transition to a defined "Safe Mode" (fail-safe state) in the event of computational or memory resource exhaustion?	SR-2, SR-70, SR-76, SR-95, SR-100, ...
CL-35	Is access to log repositories and collectors restricted exclusively to authenticated and authorized subjects? *(e.g., Implementing RBAC (Role-Based Access Control) policies and ensuring OS-level isolation via SELinux or AppArmor)*	SR-25, SR-94, SR-98, SR-103, ...
CL-36	Are attempts at unauthorized access, modification, or deletion of log repositories blocked, with such attempts being explicitly audited (logged)?	SR-93, SR-94, SR-98, SR-99, SR-103, SR-104, ...
CL-37	Is the system hardened against insecure deserialization and arbitrary code execution vulnerabilities? *(e.g., Avoiding unsafe serialization formats like Pickle, and enforcing strict schema validation for JSON/Protobuf inputs)*	SR-31, SR-33, SR-36, SR-37, SR-40, SR-45, ...
CL-38	Is the logging infrastructure resilient against manipulation or service interruption caused by the injection of malicious event records (e.g., Log Injection or Log Smashing)?	SR-80, SR-95, SR-100, SR-104, SR-105, SR-109, ...

Table B.1: Mapping to UN R 155 Annex 5 Threats

UN R155 Annex 5 Threat Category (Part A)	Mapped Checklists (CL)
1. Back-end servers used as a means to attack a vehicle or extract data	CL-01, CL-02, CL-03, CL-05
2. Services from back-end server being disrupted, affecting the operation of a vehicle	CL-08, CL-15, CL-22, CL-33, CL-34
3. Vehicle related data held on back-end servers being lost or compromised ("data breach")	CL-03, CL-06, CL-07, CL-14, CL-32, CL-35, CL-36
4. Spoofing of messages or data received by the vehicle	CL-01, CL-02, CL-05, CL-10, ...
5. Communication channels used to conduct unauthorized manipulation, deletion or other amendments to vehicle held code/data	CL-01, CL-02, CL-03, CL-09, CL-10, CL-20, ...
6. Communication channels permit untrusted/unreliable messages to be accepted or are vulnerable to session hijacking/replay attacks	CL-01, CL-02, CL-03, CL-05, CL-07, CL-10, CL-14, ...
7. Information can be readily disclosed. For example, through eavesdropping on communications or through allowing unauthorized access to sensitive files or folders	CL-03, CL-04, CL-07, CL-13, CL-14, CL-16, ...
8. Denial of service attacks via communication channels to disrupt vehicle functions	CL-08, CL-15, CL-22, CL-33, CL-34
9. An unprivileged user is able to gain privileged access to vehicle systems	CL-12, CL-17, CL-23
10. Viruses embedded in communication media are able to infect vehicle system	CL-09, CL-11, CL-31, CL-37
11. Messages received by the vehicle (for example X2V or diagnostic messages), or transmitted within it, contain malicious content	CL-01, CL-10, CL-19, CL-21, CL-24, CL-26, CL-28, CL-30
12. Misuse or compromise of update procedures	CL-03, CL-06, CL-07, CL-09, ...
13. It is possible to deny legitimate updates	CL-08, CL-09, CL-15
15. Legitimate actors are able to take actions that would unwittingly facilitate a cyber-attack	CL-05, CL-08, CL-17, CL-23
16. Manipulation of the connectivity of vehicle functions enables a cyber-attack, this can include telematics; systems that permit remote operations; and systems using short range wireless communications	CL-02, CL-03, CL-05, CL-08, CL-09
17. Hosted 3rd party software, e.g. entertainment applications, used as a means to attack vehicle systems	CL-08, CL-09, CL-12, CL-18, CL-23, CL-28, CL-31, CL-37
18. Devices connected to external interfaces e.g. USB ports, OBD port, used as a means to attack vehicle systems	CL-05, CL-08, CL-09, CL-12, CL-17, CL-23
19. Extraction of vehicle data/code	CL-06, CL-07, CL-14, CL-16, ...
20. Manipulation of vehicle data/code	CL-04, CL-13, CL-17, ...
21. Erasure of data/code	CL-04, CL-13, CL-35, CL-36, CL-38
22. Introduction of malware	CL-09, CL-11, CL-18, CL-31, CL-37
23. Introduction of new software or overwrite existing software	CL-11, CL-18, CL-20, CL-31, CL-37
24. Disruption of systems or operations	CL-08, CL-09, ...
25. Manipulation of vehicle parameters	CL-20, CL-21
26. Cryptographic technologies can be compromised or are insufficiently applied	CL-06, CL-07, CL-14, CL-16, ...
27. Parts or supplies could be compromised to permit vehicles to be attacked	CL-18, CL-27, CL-31
28. Software or hardware development permits vulnerabilities	CL-06, CL-11, CL-12, CL-15, ...
29. Network design introduces vulnerabilities	CL-08, CL-15, CL-22, CL-23, CL-33
31. Unintended transfer of data can occur	CL-05, CL-17, CL-25
32. Physical manipulation of systems can enable an attack	CL-25, CL-26, CL-27, CL-28



AutoHack: A Physically Verified Multi-Bus CAN Dataset for Intrusion Detection System Evaluation

Yuchan Song*,[1]
thddbcks618@korea.ac.kr

Sejun Ahn*,[2]
s3037444@kookmin.ac.kr

Seungjin Baek[1]
baekseungjin@korea.ac.kr

Hyeonseong Kim[2]
gj0690@kookmin.ac.kr

Huy Kang Kim†,[1]
cenda@korea.ac.kr

Sanghoon Jeon†,[2]
sh.jeon@kookmin.ac.kr

[1]Korea University [2]Kookmin University

Abstract

This paper presents the AutoHack Dataset, a high-fidelity vehicle cybersecurity dataset collected from a 2023 Hyundai vehicle in real-world driving environments. Unlike existing datasets that rely on simulations or simplified periodicity assumptions, AutoHack Dataset provides synchronized multiple CAN buses logs (C-CAN, P-CAN, and B-CAN) and incorporates sophisticated attack scenarios such as UDS-based diagnostics and timing-opaque Replay attacks. We prioritize ground-truth reliability by physically verifying attack impacts, including engine stalls and emergency call activations, to ensure accurate labeling of actual threats. Evaluation results from the AutoHack competition demonstrate that the dataset presents a significant challenge to state-of-the-art intrusion detection systems. Furthermore, we provide a comprehensive performance comparison between the competition-winning models and representative machine learning algorithms. By releasing this dataset alongside these benchmark results and optimized models, we offer a robust and reproducible framework for developing generalized vehicular security solutions. AutoHack Dataset bridges the gap between network-level detection and functional safety in modern automotive architectures.

1 Introduction

Controller Area Network (CAN) serves as the primary network for transmitting control messages between Electronic Control Units (ECUs) in vehicle systems. However, its original design did not sufficiently consider security requirements, leaving the network without inherent features such as authentication, integrity, or confidentiality. This lack of security makes CAN vulnerable to various attacks, including message spoofing, replay, and arbitrary injection [1]. Consequently, researchers have actively developed methods to detect and defend against such threats, with Intrusion Detection Systems (IDS) emerging as a critical technology for vehicle security. However, the detection performance and reliability of an IDS depend significantly on the quality and realism of the datasets used for its training and evaluation. In real vehicle environments, factors such as ECU interactions, varying driving conditions, and event-driven communications complexly alter traffic patterns. Evaluations using datasets that fail to reflect these characteristics often lead to poor performance in real-world applications. Therefore, researchers must construct high-quality datasets that encompass the diverse communication characteristics of actual vehicles to ensure the effectiveness and reproducibility of IDS studies [13].

However, existing public CAN datasets do not sufficiently reflect realistic vehicle environments. Many datasets assume that CAN messages occur periodically and use timing-based features as primary detection criteria. This assumption causes an IDS to misidentify non-periodic normal traffic, which occurs frequently in actual vehicles, as an attack [4]. Furthermore, most public datasets focus on traffic from simulations or simplified environments, failing to capture the complex ECU interactions and exceptional communication traits present in real driving. Even real-vehicle datasets often target only a single CAN bus, which prevents them from reflecting the cross-domain interactions or attack propagation that occur in multi-bus structures connected via gateways. Modern vehicles utilize multiple CAN buses—divided into functional domains such as powertrain, chassis, and body—that interoperate through a gateway. Attacks can occur simultaneously or sequentially across these buses rather than being confined to a single one. Thus, observing only one bus limits the explanation of attack paths and cross-domain dependencies. An IDS trained on single-bus data likely suffers from performance degradation or increased false positives due to partial observ-

*Equal contribution.
†Corresponding authors.

ability within a multi-bus vehicle architecture.

Existing datasets also frequently exclude interactions based on actual vehicle diagnostic protocols, such as Unified Diagnostic Services (UDS). UDS does not merely appear in attack scenarios; it constitutes event-driven traffic that arises during normal operations, including maintenance, inspection, calibration, and diagnostic sessions. These interactions exhibit non-periodic characteristics. Evaluating an IDS with datasets that omit UDS interactions leads to critical limitations, such as misidentifying frequent non-periodic communications as attacks or failing to validate diagnostic-based attacks. Consequently, performance evaluations based on existing datasets risk overestimation by biasing toward specific assumptions like periodicity, resulting in poor generalization when applied to real vehicles [18].

1.1 Dataset and the AutoHack Competition

We constructed the proposed dataset for the AutoHack competition [12, 15], which aimed to enhance vehicle security capabilities through real-vehicle attack execution and IDS development. We collected traffic from three CAN buses (C-CAN, P-CAN, and B-CAN) on a moving vehicle and reflected the complexity of simultaneous attacks on multiple buses by injecting threats into two buses concurrently. This dataset differentiates itself from existing ones by incorporating a high-difficulty traffic composition, implementing and utilizing actual UDS processes for attacks rather than simple message injections.

Figure 1 illustrates the on-site environment of the AutoHack competition. The event was conducted over three days, with ten teams participating, each composed of four members. Participants developed IDS models using the proposed dataset and were granted five official submission opportunities. A real-time leaderboard provided immediate feedback after each submission. This competition-based evaluation framework enabled external validation of the dataset's effectiveness and practical applicability.

A notable observation from the competition was that participant performance remained relatively low and dispersed. Despite using a common dataset and evaluation metric along with five official submission opportunities for iterative model improvement, the final weighted F1-scores ranged from 39% to 66%. Since this distribution was observed across multiple participating teams rather than from a single failed run, and since even the top score remained at 66%, the results suggest that the dataset was not trivially saturated under the competition setting. This observation motivated our attack-wise analysis of detection difficulty in Section 4.

1.2 Contributions of This Study

This study makes the following key contributions:



Figure 1: AutoHack competition.

- **Realistic Multi-bus Configuration and Non-Periodic Traffic Traces**: This dataset provides multiple buses' traffic data collected in real-world driving environments and reflects event-driven, non-periodic communication characteristics. These features enable a more rigorous evaluation of false positive and false negative issues that periodicity-based IDS models face in realistic settings.

- **Physically Verified High-Fidelity Labels**: We secured dataset reliability by physically verifying the impact of attack scenarios on actual vehicle maneuvers and precisely labeling attack intervals based on these observations. This approach offers a level of ground-truth fidelity that simulation-based datasets rarely provide.

- **A Challenging Benchmark for Reproducible Evaluation**: Our dataset serves as a challenging benchmark that tests the limits of state-of-the-art IDS models. By releasing it alongside the winning models from the competition, we support fair performance comparisons and reproducible evaluations in future research.

1.3 Ethical Consideration

We conducted all experiments within a safe, controlled environment and testbed on our University's internal roads, isolated from public traffic. These activities posed no risk to other vehicles or the general public. We provide the collected dataset solely for academic research purposes. Since the vulnerabilities and attack techniques we explore confirm known structural limitations of existing protocols, they present a negligible risk for actual vehicle theft or lethal exploitation. Furthermore, the entire research process adheres to relevant ethical regulations and minimizes the exposure of security vulnerabilities.

Table 1: Comprehensive comparison of open CAN IDS datasets based on six key characteristics.

#	Dataset	D.C.	A.I.	T.P.	N.T.	P.C.	L.R.	Ref.
1	CAN Intrusion (OTIDS)	Real	Real-time	Periodic-heavy	Single	CAN only	heuristic	[6]
2	Survival Analysis for Automotive IDS	Real	Real-time	Periodic-heavy	Single	CAN only	heuristic	[8]
3	Car Hacking for Intrusion Detection	Real	Offline[1]	Periodic-heavy	Single	CAN only	heuristic	[7]
4	SynCAN	Sim	Offline[1]	Periodic-heavy	Single	CAN only	heuristic	[5]
5	Automotive CAN Bus Intrusion v2	Real	Offline[1]	Periodic-heavy	Single	CAN only	heuristic	[3]
6	Can Log Infector	Real	Offline[1]	Periodic-heavy	Single	CAN only	heuristic	[2]
7	ROAD CAN Intrusion Dataset	Real	Real-time	Periodic-heavy	Single	CAN only	P.I.V.[2]	[18]
8	CAN Signal Extraction and Translation	Real	Real-time	Periodic-heavy	Single	CAN only	heuristic	[9]
9	M-CAN Intrusion Dataset	Real	Offline[1]	Periodic-heavy	Single	CAN only	heuristic	[11]
10	B-CAN Intrusion Dataset	Real	Offline[1]	Periodic-heavy	Single	CAN only	heuristic	[10]
11	can-train-and-test	Real	Real-time	Periodic-heavy	Single	CAN only	P.I.V.[2]	[14]
12	CAN-MIRGU	Real	Offline[1]	Periodic-heavy	Single	CAN only	P.I.V.[2]	[19]
13	CICIoV2024	Testbed	Real-time	Periodic-heavy	Single	CAN only	heuristic	[16]
14	**AutoHack (Ours)**	**Real**	**Real-time**	**P-H/E-D**	**Multi**	**C&U**	**P.I.V.[2]**	**This Work**

D.C.: Data Collection, **A.I.**: Attack Injection, **T.P.**: Temporal Property, **N.T.**: Network Topology, **P.C.**: Protocol Coverage, **L.R.**: Label Reliability
[1] Offline: Replay/edit/simulated injection [2] P.I.V.: Physical-impact verified P-H/E-D: Periodic-heavy/Event-driven C&U: CAN and UDS

2 Related Works

2.1 Investigation of Vehicle Cybersecurity Research Datasets

Vehicle cybersecurity datasets have evolved from early studies that identified simple attack packet injections to modern research that reflects the physical domain characteristics of actual vehicles. Early-stage datasets, such as OTIDS [6] and the Car Hacking Dataset [7], primarily focused on attacks that cause high bus loads by abnormally surging traffic, such as Denial of Service (DoS) or Fuzzing. While these datasets contributed to verifying the basic mechanisms of IDS, researchers now argue that they lack sufficient discriminative power to represent modern vehicle security threats, as contemporary detection models achieve nearly 100% detection accuracy on such simplified attack patterns [18].

Subsequently, researchers began to project communication characteristics from real driving environments into their datasets. The ROAD dataset [18] enhanced sophistication by including Masquerade attacks that influence physical behavior based on logs collected from actual vehicles. Recent studies continue this trend, such as can-train-and-test [14], which tests model generalization performance across multiple vehicle models, and CAN-MIRGU [19], which reflects the dynamic characteristics of vehicles during real-world driving.

2.2 Dataset Comparison

We categorize and evaluate the quality and utility of vehicle security datasets based on six key characteristics. Table 1 summarizes the comparative analysis of major datasets, including the AutoHack dataset proposed in this work.

2.2.1 Data Collection

We categorize data collection environments into real vehicles, laboratory-scale vehicle testbeds, and simulation environments that mimic virtual vehicles. As shown in Table 1, most datasets, including OTIDS, ROAD, and M-CAN, collect data from real vehicle environments to capture in-vehicle dynamic characteristics. While testbeds or simulation environments prove useful for verifying valid logical operations, they face limitations in fully reflecting irregularities caused by communication delays, bus loads, and exceptions that occur during actual driving. Considering these factors, we collected the AutoHack dataset in a real vehicle driving environment, providing a foundation to validate the reliability of intrusion detection models under actual driving conditions.

2.2.2 Attack Injection

We classify attack injection methods into real-time injection of attack messages and offline injection by editing pre-collected normal data. According to Table 1, several datasets, including Car Hacking, SynCAN, and Automotive v2, adopt the post-editing approach. While this method offers advantages in constructing datasets safely and easily, it fails to reflect the logical responses and interactions between in-vehicle ECUs or the dynamic system changes caused by attack injections. In contrast, the AutoHack dataset adopts the real-time injection method—similar to ROAD, can-train-and-test, and CICIoV2024—to ensure high reliability by allowing models to learn and verify variables that occur in actual threat situations.

2.2.3 Temporal Property

We distinguish the transmission period of messages based on whether they transmit at regular intervals (periodic) or irreg-

ularly based on specific events (event-driven). Most public datasets consist of periodic messages to project normal driving conditions. While useful for verifying detection models that analyze transmission interval deviations, these datasets limit the ability to reflect the complexity of non-periodic messages in actual vehicles. The AutoHack dataset integrates periodic messages with event-driven non-periodic messages arising from specific operating conditions, providing a precise metric to evaluate intrusion detection model performance.

2.2.4 Network Topology

We classify network topology as single-bus or multi-bus environments based on the structure and domain of the CAN bus where data collection occurred. Most public datasets provide data collected from a single CAN bus. While suitable for evaluating IDS models targeting individual buses, this approach limits the reference for inter-bus interactions and attack propagation through gateways across the entire vehicle network. The AutoHack dataset focuses on a multi-bus environment within an actual vehicle, reflecting the complex network architecture to enable performance evaluation for IDS models covering the entire internal CAN network.

2.2.5 Protocol Coverage

As shown in Table 1, most public datasets—including Survival Analysis, SynCAN, and CAN-MIRGU—collect data exclusively for the standard CAN protocol. However, actual vehicles utilize a mixture of various protocols for control and diagnostics. The AutoHack dataset simultaneously collects data using the UDS protocol for real-time diagnostics in addition to standard CAN messages. This provides a sophisticated benchmark for IDS models to detect threats where attackers exploit specific authorized diagnostic services to impact driving safety critically.

2.2.6 Label Reliability

Finally, differences exist in labeling precision and verification methods, which determine dataset reliability. In Table 1, most public datasets employ a post-collection labeling method based on specific patterns in CAN IDs, data fields, or injection times. While easy to implement, this method raises reliability concerns regarding label precision due to network delays or bus loads. Therefore, we performed real-time labeling during the attack injection process for the AutoHack dataset and confirmed the physical changes and impacts caused by the injections to verify the causal relationship within the vehicle system.

3 AutoHack Dataset

This section describes our data collection methodology, the physical verification of attack scenarios, the statistical composition of the dataset, and its distinctive features compared to existing datasets. To facilitate reproducibility and support the research community, the proposed AutoHack Dataset is publicly released through a permanent Zenodo archive [17], which serves as the official access channel for the dataset. The released dataset includes ① synchronized multi-bus CAN logs collected from C-CAN, P-CAN, and B-CAN interfaces, ② metadata, ③ ground-truth labels for the attack scenarios, and ④ accompanying documentation. The data are provided in CSV format for broad compatibility with various data analytics and machine-learning tools. The dataset is intended to support academic research and educational use, and released under the Creative Commons Attribution 4.0 International (CC BY 4.0) license, allowing reuse with appropriate attribution.

3.1 Data Collection Methodology

We designed this dataset to ensure data reliability and temporal precision within real-world driving environments.

Experimental Environment We conducted experiments using a 2023 Hyundai Motor Company S model and constructed an integrated testbed by disassembling vehicle components to tap into the wiring connected to the ECUs behind the Central Gateway (CGW) for the C-CAN, P-CAN, and B-CAN interfaces, as illustrated in Fig. 2. In this setup, C-CAN was persistently monitored across all vehicle states, whereas P-CAN and B-CAN data were selectively captured during driving and parking (stationary) sessions, respectively. To ensure safety, we performed all data collection and attack injections within a controlled environment on a university campus road, isolated from public traffic. We utilized a Dual Channel Kvaser hardware interface for data acquisition and managed the process using custom scripts developed with the Kvaser CANlib SDK on a Windows-based laptop acting as an Attacker PC.

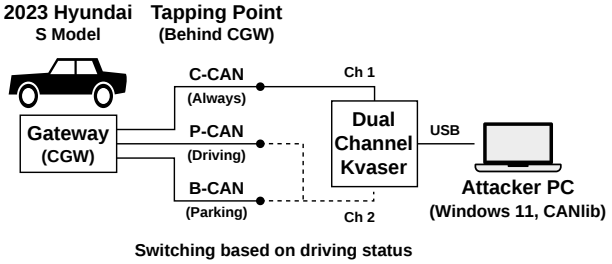


Figure 2: Experimental testbed configuration for data collection and attack injection.

Synchronization between multiple buses To achieve precise time synchronization across the functional buses, we utilized a single-device Dual Channel Kvaser interface, where both channels share a unified hardware timebase. While C-CAN was persistently monitored on the first channel, the P-CAN and B-CAN interfaces were captured on the second channel during their respective driving and stationary sessions. Since all incoming frames from both ports are timestamped by the same internal oscillator at the moment of reception, the system maintains a strictly consistent global timeline. This architecture ensures that the temporal relationship between the persistent C-CAN traffic and the alternating P-CAN and B-CAN data remains synchronized at the microsecond (μs) level, inherently eliminating software-induced time offsets or drifts between the channels.

Strict Ground-Truth Labeling To maximize labeling accuracy, we define the starting point as the moment the actual bus completes the transmission of the first attack frame, rather than the time the attack script initiates. Specifically, we apply a strict criterion by labeling reactive messages or jitter as normal data, as the attacker did not directly generate these packets. We also defined the attack termination point as the completion of the last injected packet's transmission.

3.2 Attack Scenarios & Physical Verification

We constructed scenarios that pose threats to actual vehicle control and verified them physically, going beyond simple packet injection.

3.2.1 Attack Injection Scenarios

This dataset includes the following five major attack types:

DoS: We utilize the 0x000 ID, which holds the highest priority, to flood messages at a 0.2ms interval and seize network bandwidth.

Fuzzing: We induce system malfunctions by injecting random combinations of IDs, DLCs, and payloads at $1ms$ intervals. To explore a comprehensive range of potential vulnerabilities, the attack covers the entire valid 11-bit CAN ID spectrum from $0x000$ to $0x7FF$. This broad scope intentionally includes IDs of $0x700$ and above, extending the fuzzing targets to the UDS range. By including diagnostic communication channels, the attack increases the probability of triggering unintended diagnostic states or system-level exceptions.

Spoofing: We select valid IDs directly associated with critical vehicle functions, such as the Electronic Parking Brake (EPB) and Parking Assist System (PAS), to ensure the attack remains indistinguishable from legitimate traffic. To address potential security and proprietary concerns of the vehicle manufacturer, we specifically disclose the detailed specifications of the C-CAN interface as representative cases in Table 2. The attack involves selecting a random integer between 1 and 10 and injecting that specific number of messages consecutively at $1ms$ intervals in a burst fashion to disguise them as normal signals.

Replay: We neutralize timing-analysis-based IDS by retransmitting normal data pre-collected from the C-CAN, P-CAN, and B-CAN interfaces during both driving and parking states. To ensure protocol and architectural consistency, the retransmission is strictly mapped to the corresponding bus type, such as replaying C-CAN logs specifically onto the C-CAN interface. These messages are synchronized with their original hardware timestamps to replicate legitimate traffic intervals and patterns, effectively deceiving detection systems that rely on frequency analysis.

UDS Spoofing: We execute sophisticated attacks by re-implementing legitimate diagnostic sequences extracted from an authentic Original Equipment Manufacturer (OEM) diagnostic tool, enabling the forceful control of ISO 14229-compliant sessions in core ECUs. Similar to the spoofing attacks, we exclusively disclose UDS targets associated with the C-CAN interface to address security and proprietary concerns of the vehicle manufacturer. As summarized in Table 3, these attacks focus on safety-critical components on the C-CAN bus, such as the front camera, power steering, and surround radars. By mimicking authorized diagnostic requests, we can transition these ECUs into restricted diagnostic modes or deactivate specific functions, effectively influencing the vehicle's functional safety beyond simple bus injection.

3.2.2 Physical Impact Verification

We directly verified the effectiveness of all attacks through changes in the physical state of the vehicle.

DoS: We confirmed that data continuity breaks cause warning lights and system error alarms in major ECUs.

Table 2: Detailed specifications of targeted C-CAN IDs for Spoofing attacks.

ID (Hex)	Function	Physical Response
0x366	Engine RPM	RPM gauge fluctuation
0x390	Warning Alarm	Unexpected alert trigger
0x490	EPB Error	EPB warning indicator
0x4F4	Parking Assist	PAS logic error
0x50C	Fuel Warning	Fuel warning trigger

Table 3: Detailed specifications of targeted UDS-based IDs on C-CAN for Spoofing attacks.

ID (Hex)	Targeted ECU	Physical Response
0x755	Right Radar	Diagnostic state entry
0x7B7	Left Radar	Sensor session lock
0x7C4	Front Camera	Camera mode deactivation
0x7D4	Power Steering	Steering assist limitation

Fuzzing: We captured critical defects, such as the vehicle suddenly stopping in Drive (D) mode or an automatic connection to the emergency call center without an accident.

Spoofing: We confirmed that the injection of messages specified in Table 2 leads to functional malfunctions, which are manifested as warning lights or "Check System" messages on the instrument cluster. These physical responses were directly verified through experiments, confirming that spoofing critical control IDs effectively triggers unintended system behaviors.

Replay: We verified control interference by retransmitting pre-collected normal data synchronized with original timestamps. This allowed us to arbitrarily reproduce or neutralize specific functions, such as turn signals and window control, by playing back historical driving sequences.

UDS Diagnostic Session Manipulation: We can change the ECU state or temporarily limit certain vehicle functions by forcibly starting unauthorized diagnostic sessions and performing specific service requests. For example, specific functions may deactivate or diagnostic modes may activate; this demonstrates that UDS-based attacks influence actual vehicle behavior beyond simple bus injection.

These physical verification results demonstrate that all attack scenarios, including diagnostic-based attacks, exert observable impacts on the actual vehicle state, proving that the AutoHack dataset reflects real threat environments rather than simple packet-level variations.

3.3 Dataset Composition & Statistics

The collected data follows a systematic statistical structure for objective performance evaluation of detection models.

Session-based Data Splitting To prevent bias from temporal continuity, we separated the Train Set and Test Set into independent driving sessions at a ratio of approximately 6.6:3.4. and Table 4 provide detailed configuration metrics.

Spoofing ID Separation Strategy To prevent the model from overfitting to specific IDs, we assigned spoofing IDs for the training and test sets mutually exclusively, as shown in Table 6. In particular, we assigned new IDs to the test set for B-CAN to rigorously measure generalization performance against unseen attacks.

Table 4: Detailed attack statistics by interface and session.

Bus Interface	Attack Type	Train Set	Test Set
C-CAN	Total Frames	4,026,488	1,678,489
	Fuzzing	38,191	13,147
	Replay	24,123	16,876
	Spoofing	22,088	5,384
	DoS	14,419	8,490
	UDS Spoofing*	580	420
P-CAN	Total Frames	2,655,149	883,323
	Fuzzing	39,952	10,319
	DoS	34,441	7,647
	Replay	25,797	16,746
	Spoofing	23,634	5,188
	UDS Spoofing*	887	344
B-CAN	Total Frames	240,963	149,685
	Fuzzing	57,459	29,698
	DoS	28,244	19,492
	Spoofing	12,846	11,698
	Replay	5,589	8,408
	UDS Spoofing*	580	460

*Aggregated counts of various UDS-based spoofing

3.4 Distinctive Features

The AutoHack Dataset possesses three core strengths that distinguish it from existing datasets:

- **Overcoming Limitations of Periodicity Assumption**: Most public CAN datasets assume messages occur at fixed periods, limiting detection to simple timing analysis. This dataset includes non-periodic (event-driven) communications like UDS and precisely restores the $Inter-arrival time$ of original traffic during Replay attacks. This sophisticated configuration neutralizes the detection logic of existing IDS models relying on temporal regularity and provides a challenging evaluation environment requiring more advanced detection algorithms.
- **High Complexity and Entropy Based on Real Driving Environments**: Unlike datasets collected in limited labs, we collected this dataset through actual driving on campus roads, naturally including various driving states such as acceleration, deceleration, turning, and stopping. This increases the variability of message distribution and results in high entropy, making anomaly detection

Table 5: AutoHack Dataset statistics.

Metric	Train Set	Test Set
Total Frames	6,922,600	2,711,497
Driving Duration (sec)	1,400.8182	583.1505
Unique CAN IDs	2,048	2,048
Normal Frames	6,593,770	2,557,180
Attack Frames	328,830	154,317

Table 6: Summary of target CAN IDs for Spoofing attacks by interface and dataset.

Bus Interface	Train Set (IDs)	Test Set (IDs)
C-CAN	0x366, 0x390, 0x4F4	0x490, 0x50C
P-CAN	0x171, 0x374, 0x5CF	0x080, 0x2B0
B-CAN	0x18E, 0x1BC, 0x510	0x012, 0x08C, 0x102, 0x103, 0x107, 0x1C2, 0x560

more realistic and difficult. To maximize experimental reliability, we separated the Train and Test sets into independent driving sequences at different times rather than arbitrarily splitting a single session.

- **Providing Synchronized Multi-bus data**: Unlike existing studies focused on a single CAN bus, the AutoHack dataset provides synchronized logs of C-CAN, P-CAN, and B-CAN at the hardware level. This allows for analyzing message characteristic changes through the gateway and attack propagation across multiple buses, reflecting modern vehicle architectures rarely addressed in existing datasets.

4 Benchmarking and Evaluation

Section 3 introduced the AutoHack Dataset and its design choices for realistic in-vehicle network modeling. In this section, we validate these claims through controlled benchmarking experiments and analyze whether the dataset provides a reproducible and fair evaluation environment for diverse IDS models.

4.1 Benchmark Setup

4.1.1 IDS Competition Structure: AutoHack

This study establishes the benchmark configuration based on the operational structure and evaluation framework of the AutoHack competition conducted in conjunction with the AutoHack Dataset. The competition was designed to enable undergraduate students with a strong interest in intrusion detection systems to develop and evaluate diverse detection models under a unified dataset and common evaluation metrics. This setting allowed for the observation of how the dataset is utilized in a practical research context and how performance distributions emerge across different modeling approaches.

4.1.2 Baselines for Benchmarking

The baseline configuration is not intended to highlight the superiority of any specific model. Rather, it aims to establish a reference environment to verify whether the proposed dataset supports fair and reproducible evaluations across diverse training conditions. To achieve this, we adopted widely recognized models from existing IDS studies that also emerged as top performers in the AutoHack competition. This approach allows us to examine the performance distribution of algorithms validated in both academic research and competitive settings, thereby ensuring the objectivity and comparability of our evaluation results.

Regarding evaluation metrics, F1-score was selected as the primary metric in consideration of the fact that accuracy can be overly optimistic in class-imbalanced settings. In addition, from the perspective of attack detection, recall for each attack class was reported to quantitatively reflect the risk of false negatives.

Controlled Evaluation Protocol To ensure that the benchmark results reflect genuine model generalization rather than artifacts of data leakage or memorization, we strictly adopt the controlled data configuration described in Section 3. First, the train and test sets are separated at the driving-session level to prevent temporal continuity between splits, thereby mitigating performance inflation caused by shared temporal context. This session-based split enables a more rigorous evaluation of generalization across independent driving conditions.

Second, since the feature configuration includes Arbitration_ID, we enforce mutually exclusive spoofing target IDs between the training and test sets. This prevents the model from memorizing specific attack identifiers and instead evaluates its ability to generalize to previously unseen spoofing patterns. In particular, assigning new spoofing IDs to the B-CAN test set allows us to explicitly assess cross-ID generalization capability. Under these controlled settings, the observed performance distributions can be interpreted as reflecting the structural characteristics of the dataset rather than unintended biases in data partitioning.

Baseline Models In this study, tree-based models (Random Forest and XGBoost), which demonstrated strong performance in the competition and are widely adopted in IDS research, were selected as the baseline configuration. These models are capable of effectively capturing both statistical

and temporal features, thereby providing an appropriate reference for analyzing how the structural complexity and class imbalance inherent in the dataset are reflected in performance metrics. Rather than using the baseline models as evidence of practical applicability by themselves, we use this baseline setting as a controlled and reproducible reference protocol for analyzing IDS-relevant characteristics of the AutoHack Dataset. Under this protocol, the dataset exposes several benchmark-relevant properties: heterogeneous detection difficulty across attack types, decision-boundary changes caused by non-periodic UDS traffic, and generalization challenges in a time-synchronized multi-bus setting. These observations provide empirical support that the AutoHack Dataset can serve as a realistic and challenging benchmark for IDS evaluation. To support reproducibility and transparency, we archive the IDS reference implementation on Zenodo together with the dataset-related artifact [17]. This release enables independent verification and replication of the experimental setup, feature configuration, data preprocessing, and evaluation procedures.

Feature Configuration To ensure a fair evaluation of the AutoHack Dataset, an identical feature set was applied to all baseline models. The selected features do not rely on vehicle-specific signal interpretation; instead, they consist of reproducible attributes that generally represent the structural information and temporal patterns of CAN messages. Specifically, the feature set includes 'Basic & Temporal' features and frequency-based statistical features ('Frequency & Rolling'). The detailed composition is presented in Table 7.

4.2 Benchmark Results and Key Observations

Observation 1: Variations in Detection Difficulty Across Attack Types The AutoHack Dataset is not constructed around a single level of attack difficulty. Rather, it simultaneously includes highly distinguishable attacks and stealthy attacks that are highly correlated with normal traffic. This design is significant from a benchmarking perspective, as it avoids bias toward a specific attack type and instead encompasses a broad spectrum of difficulty levels. In other words, the dataset provides a structure that enables the evaluation of IDS performance under diverse conditions by incorporating both relatively distinguishable attacks and high difficulty attacks that distributionally overlap with normal traffic.

These characteristics were consistently observed in the competition results. In particular, Replay and Spoofing attacks exhibited relatively lower performance across participating models, thereby forming a higher level of detection difficulty. The consistent degradation observed across heterogeneous model families suggests that this difficulty is not model-specific but reflects an intrinsic structural property of the dataset. Table 8 presents the classification performance (F1-score) for each attack type in the AutoHack Dataset, quantitatively demonstrating that detection difficulty varies structurally across attack categories. While relatively separable attacks achieved high F1-scores, Replay and Spoofing showed comparatively lower F1-scores, indicating greater difficulty in distinguishing them from the normal class.

The confusion matrix in Figure 3 further illustrates this distributional characteristic. Spoofing and Replay were misclassified as Normal 7,770 times (34.89%) and 8,784 times (20.90%), respectively, which are markedly higher than those of other attack types. In addition, Normal samples were misclassified as Spoofing (2,346 cases) and Replay (13,051 cases) more frequently than for other attack classes. As described in Section 3, these findings support the interpretation that Replay and Spoofing attacks exhibit structural similarity to normal traffic, since they are generated by retransmitting legitimate driving messages or by leveraging valid control IDs, thereby resulting in distributional characteristics that closely resemble normal communication.

In summary, the AutoHack Dataset is structured to encompass varying degrees of separability across attack types and to simultaneously include attacks with different levels of detection difficulty. This design enhances the suitability of the dataset as a benchmarking resource by avoiding bias toward a single difficulty level and enabling evaluation across diverse attack scenarios. The baseline results serve as a reference framework for analyzing these structural characteristics, confirming that the dataset provides an evaluation environment that is not skewed toward overly separable or uniformly simple attack patterns.

Importantly, the observed performance variations should not be interpreted merely as limitations of the evaluated models. Rather, they reflect the realistic nature of the AutoHack Dataset, in which the distributional boundary between normal and attack traffic is inherently ambiguous. By incorporating attacks that overlap with normal communication patterns, the dataset presents a detection environment that more closely resembles real-world vehicular networks. Consequently, compared to existing public datasets, it offers a more challenging and practically representative benchmark for IDS evaluation.

Observation 2: Impact of Non-Periodic (UDS) Communication on Detection Boundaries The AutoHack Dataset is designed to include not only periodic CAN communication but also non-periodic diagnostic communication based on UDS, which occurs in real vehicle environments. This design reflects the fact that in-vehicle networks are not merely composed of periodic signals, but rather exhibit a composite structure in which event-driven and diagnostic messages coexist. In practical vehicle systems, non-periodic diagnostic traffic constitutes a legitimate part of normal operation; therefore, incorporating UDS messages enables a more realistic representation of the structural diversity of benign communication patterns.

To analyze the impact of this data configuration, we com-

Table 7: Baseline feature set for multi-bus intrusion detection

Category	Feature	Description
Basic & Temporal	Arbitration_ID	CAN ID (decimal format)
	DLC	Data Length Code (bytes)
	Prev_Interval	Time interval from previous message
	ID_Prev_Interval	Inter-arrival time (same CAN ID)
	Data_Prev_Interval	Inter-arrival time (same CAN ID and payload)
Frequency & Rolling	ID_Frequency	ID occurrence count (10 second time window)
	Data_Frequency	ID + payload occurrence count (10 second time window)
	Frequency_diff	ID frequency difference

Table 8: Per-attack performance of baseline IDS.

Attack Type	Precision	Recall	F1-score	AUC
Normal	0.9940	0.9928	0.9934	0.9952
DoS	1.0000	1.0000	1.0000	1.0000
Spoofing	0.8328	0.7449	0.7864	0.9908
Replay	0.6498	0.7128	0.6799	0.9854
Fuzzing	0.9706	0.9943	0.9823	0.9999
UDS_Spoofing	0.9498	0.9126	0.9308	0.9992

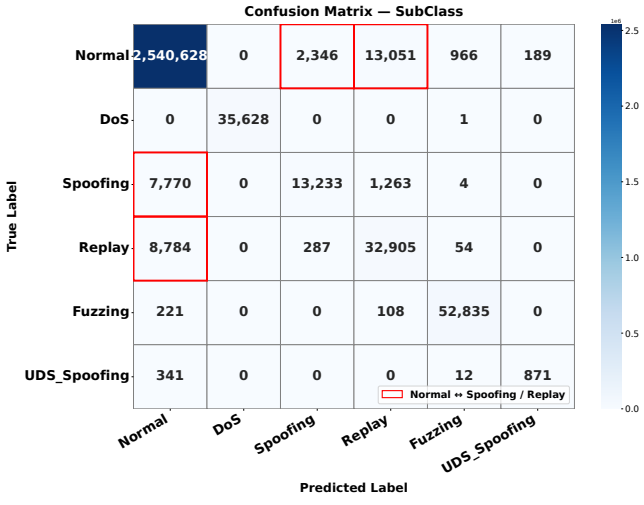


Figure 3: Confusion matrix of the Random Forest model.

pared two experimental settings: one including UDS messages and the other excluding them. The comparison was conducted under identical preprocessing procedures, feature configurations, model settings, and evaluation protocols, with the inclusion of UDS traffic as the only varying factor. As shown in Table 9, removing UDS results in a macro F1-score of 0.8634, suggesting that aggregate performance metrics exhibit no substantial difference. However, a closer examination of the confusion matrix in Figure 4 reveals meaningful distributional changes that are not captured by the overall metric.

In particular, when UDS messages are excluded, the number of Normal samples misclassified as Fuzzing increases from 966 to 5,296—representing more than a fivefold rise. This change indicates a significant restructuring of the decision boundary between specific classes rather than a simple fluctuation in overall accuracy.

This phenomenon is closely related to the structural characteristics of the Fuzzing attack defined in Section 3. As described therein, the Fuzzing attack generates random CAN IDs within the range 0x000 to 0x7FF, randomly selects DLC values between 0 and 8, and injects arbitrary payloads at 1 ms intervals to induce unexpected system malfunctions. Such traffic lacks periodicity and exhibits high variability and randomness, forming a statistically irregular distribution.

UDS traffic, while benign, also exhibits non-periodic behavior, as it follows a request–response pattern and is triggered under specific conditions or events. From a temporal and

Table 9: Per-attack performance of baseline IDS (Excluding UDS).

Attack Type	Precision	Recall	F1-score	AUC
Normal	0.9933	0.9916	0.9925	0.9948
DoS	0.9999	1.0000	1.0000	1.0000
Spoofing	0.8003	0.6187	0.6979	0.9898
Replay	0.6465	0.7122	0.6777	0.9846
Fuzzing	0.9074	0.9946	0.9490	0.9999
Macro Avg	0.8695	0.8634	0.8634	0.9938
Weighted Avg	0.9848	0.9844	0.9844	0.9938

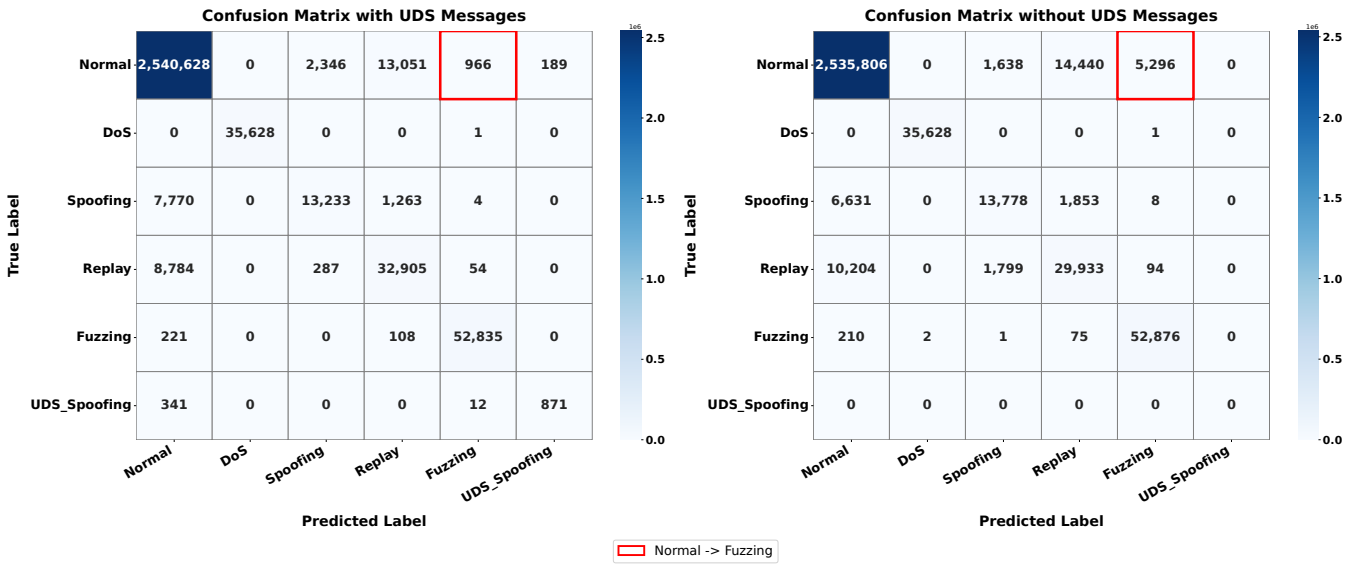


Figure 4: Confusion matrices of the Random Forest with and without non-periodic (UDS) messages.

statistical perspective, UDS shares partial similarities with Fuzzing in terms of non-periodicity and variability. When UDS is included, the model learns that irregular communication patterns do not necessarily indicate malicious activity, enabling it to form a more refined decision boundary between stochastic attack traffic and legitimate diagnostic communication. In contrast, when UDS is removed, the Normal class becomes relatively simplified and centered around periodic patterns. Consequently, the model may develop a stronger association between non-periodic behavior and malicious activity, increasing the likelihood that boundary samples are misclassified as Fuzzing.

Although removing UDS inevitably alters the total number of samples and class distribution to some extent, misclassification changes for other attack types (*e.g.*, DoS, Spoofing, and Replay) remain relatively limited. In contrast, the Normal-to-Fuzzing misclassification exhibits a pronounced increase. This suggests that the observed performance variation is better interpreted not as a simple consequence of reduced data volume, but rather as a structural weakening of the distributional boundary between Normal and Fuzzing due to the exclusion of non-periodic benign traffic.

In summary, the inclusion of UDS traffic should be understood not merely in terms of numerical performance differences, but as a factor that enhances the structural richness and realism of the dataset. By incorporating non-periodic diagnostic communication that occurs in real vehicle environments, the AutoHack Dataset more faithfully represents the complex distributional characteristics of in-vehicle networks and provides a more realistic and structurally challenging benchmark for IDS evaluation.

Observation 3: Empirical Characteristics of the Time-Synchronized Multi-Bus Setting The AutoHack Dataset differs from conventional single-bus datasets in that it simultaneously captures C-CAN, P-CAN, and B-CAN traffic under time synchronization. This configuration preserves the coexistence of multiple functional domains within the same vehicle and enables integrated analysis of bus-specific and cross-bus traffic characteristics that cannot be directly examined in single-bus datasets.

In Observation 3, we employ an expanded 38-feature configuration. This full configuration extends the feature set in Table 7 by adding 30 additional descriptors, including payload byte-level features, payload statistical features, per-ID rolling-window statistics, and payload entropy. We leverage this expanded representation to investigate how a wider range of features is reflected in the multi-bus setting. The complete feature list and extraction definitions for the expanded configuration are provided in the `feature_configuration.txt` document included in the Zenodo benchmark archive [17].

We first compare two training regimes using the same 38-feature representation. In the *single-bus* regime, a model is trained and evaluated on data from the same bus. In the *multi-bus* regime, a single model is trained on the union of all three buses and evaluated on the corresponding full test set; the predictions are then partitioned by the `Interface` field to report per-bus performance. As shown in Table 10, the multi-bus setting yields lower per-bus macro F1-scores than the single-bus setting across all three buses for both Random Forest and XGBoost. We additionally compared feature importance between the single-bus and multi-bus settings under the same 38-feature configuration, and the top-ranked features remained broadly similar across both regimes. Specifically, ID-aware

temporal and frequency features such as `ID_Prev_Interval` and `ID_Frequency` frequently appeared in the highest ranks. This suggests that the observed F1-score reduction is difficult to attribute to a significant shift in feature importance alone.The corresponding feature-importance summary is provided in Appendix **??**.

Since feature importance alone does not fully explain the observed reduction in F1-scores, we next examine an ID-level structural property specific to the integrated multi-bus setting. Although ID-aware temporal and frequency features remain highly ranked in both regimes, this very consistency suggests that ID context continues to play a pivotal role in classification. Rather than searching for shifts in feature importance, we investigate how IDs are organized across buses, prompting a closer look at structural ID-level properties. Specifically, we focus on overlapping CAN IDs for two reasons. First, given that ID-aware features remain dominant in both regimes, ID context is a natural candidate for further analysis. Second, unlike message-rate or payload-statistical features—which can be computed within an individual bus—overlapping CAN IDs are only observable when multiple buses are analyzed jointly. We therefore utilize overlapping CAN IDs as a structural variable to examine how the multi-bus setting influences error composition, rather than as the sole explanation for performance degradation.

One distinguishing characteristic of the time-synchronized multi-bus design of the AutoHack Dataset is that the same CAN identifier can be observed across different buses. We refer to such identifiers as *overlapping CAN IDs*. After excluding Fuzzing attacks, which span the entire ID space by construction, 40 overlap IDs are identified: seven between P-CAN and B-CAN, thirty-three between P-CAN and C-CAN, and four between B-CAN and C-CAN. A small subset of these IDs also exhibits label inconsistency across buses. Rather than treating overlapping IDs as a single determining factor, we present them as a structural characteristic that can arise in integrated multi-bus data and that is difficult to observe directly in conventional single-bus datasets.

To further examine how this structural characteristic appears in the multi-bus setting, we compare the proportion of overlap-ID messages in the full test set with that in the subset misclassified by the multi-bus model. The resulting pattern differs by class. As shown in Table 11, for Random Forest, the overlap-ID ratio increases for the Normal class (46.12% $\rightarrow$ 53.14%), remains nearly unchanged for Spoofing (25.47% $\rightarrow$ 25.75%), and decreases for Replay (41.20% $\rightarrow$ 38.52%). These results show that overlap-ID messages do not appear with a single common tendency across classes in the integrated multi-bus setting. In particular, the opposite directions observed for Normal and Replay suggest that the overlap-ID structure is reflected differently in the error composition of each class. In this sense, while domain shift may still contribute to the overall performance change, the observed class-wise variation indicates that the multi-bus setting cannot be understood solely in terms of a single uniform effect.

DoS, Fuzzing, and UDS were excluded from this comparison because their specific traffic characteristics render the overlap-ID ratio less informative. Specifically, DoS attacks yield an insufficient number of misclassified samples for analysis, Fuzzing attacks inherently span the entire ID space, and UDS involves non-periodic diagnostic traffic with no overlapping IDs in this dataset.

In summary, these results show that the time-synchronized multi-bus design of the AutoHack Dataset enables analysis of structural properties, such as overlapping CAN IDs, that are difficult to observe in conventional single-bus datasets. This is also meaningful in light of future vehicle architectures: as E/E systems evolve from functionally separated P-, B-, and C-CAN domains toward zonal architectures, bus organization may be restructured not only by function but also by physical location and centralized control. In such settings, interactions across previously distinct domains may become more tightly coupled, making multi-domain analysis increasingly relevant for IDS design and evaluation.

4.3 Dataset Fidelity and Benchmark Validity

The quantitative analyses and benchmark results presented in this study demonstrate that the AutoHack Dataset possesses structural characteristics and a level of realism that distinguish it from existing vehicular network datasets. As observed in Observations 1–3, the dataset incorporates varying levels of attack difficulty, the influence of non-periodic communication, and interactions arising from a multi-bus integrated systems.

In particular, unlike existing public datasets that primarily rely on periodic message patterns within a single bus, the Au-

Table 10: Macro F1-score of single-bus versus multi-bus training.

Model	Bus	Single-bus	Multi-bus	Δ
RF	B-CAN	0.8411	0.6840	−0.1572
	C-CAN	0.8368	0.7299	−0.1069
	P-CAN	0.8880	0.7775	−0.1104
XGBoost	B-CAN	0.7879	0.7339	−0.0540
	C-CAN	0.8613	0.7505	−0.1109
	P-CAN	0.8236	0.7791	−0.0445

Table 11: Comparison of overlap-ID ratios by class.

Class	r_{test}	r_{mis}	Direction
Normal	46.12%	53.14%	$\uparrow$
Spoofing	25.47%	25.75%	$\approx$
Replay	41.20%	38.52%	$\downarrow$

Table 12: Cyber-physical impact taxonomy: mapping attack types to physical domains and functional safety relevance.

Attack Type	Target Bus	Affected Domain	Physical Impact	Severity	Safety Relevance
DoS	C-CAN	Multi-domain	ECU warning lights, system error alarms	Medium	Degraded situational awareness
Fuzzing	P-CAN, B-CAN	Powertrain, Body	Engine stall in D-mode, unintended emergency call	Critical	Direct driving safety hazard
Spoofing	C-CAN	Chassis, Body	EPB warning, PAS logic error, false fuel warning	Medium–High	Misleading driver information
Replay	C-CAN, P-CAN, B-CAN	Control, Body	Arbitrary turn signal / window activation	Medium	Unintended actuator control
UDS Spoofing	C-CAN	ADAS, Steering	Camera deactivation, steering assist limitation	Critical	ADAS function loss during driving

toHack Dataset includes a complex communication structure across multiple buses as well as non-periodic messages such as UDS. These characteristics more faithfully reflect communication patterns observed in real vehicular environments and introduce a degree of practical complexity that has not been sufficiently addressed in prior datasets.

Furthermore, the benchmark results reveal that identical models exhibit performance variations depending on attack type and bus configuration. This indicates that distributional overlap, class imbalance, and inter-bus interactions within the dataset directly influence IDS performance. Such phenomena should not be interpreted merely as differences in model capability, but rather as consequences of the dataset's inherent structural properties.

In conclusion, the AutoHack Dataset provides a more challenging and realistic evaluation environment compared to existing datasets and can serve as a benchmark for more rigorously assessing the generalization capability of IDS models. These findings underscore the importance of incorporating realistic vehicular network conditions into future automotive security research and evaluation frameworks.

5 Discussion

5.1 Bridging Cyber Attacks and Physical Malfunctions

Beyond evaluating IDS performance, the proposed dataset introduces a new research direction by enabling the analysis of how in-vehicle network attacks affect the physical behavior of the vehicle. Automotive cybersecurity is increasingly recognized not merely as a network-level anomaly detection problem, but as a cyber-physical challenge in which the impact of cyber attacks on vehicle dynamics and operational safety must be jointly considered.

Various anomalous phenomena observed during the experiments illustrate this characteristic. When attacks were injected, abnormal changes in vehicle states were identified across multiple functional domains, including driving assistance and control systems, powertrain components, body control modules, and convenience features. These observations indicate that the attacks extend beyond communication-layer anomalies and are directly associated with the vehicle's control logic. In particular, certain attacks were found to influence safety-related functions, suggesting that automotive security concerns transcend traditional information security and are closely intertwined with functional safety.

To systematically formalize these observations, Table 12 presents a cyber-physical impact taxonomy that maps each attack type to its targeted bus, affected functional domain, physical impact, and relevance to functional safety. As illustrated, attacks such as Fuzzing on the P-CAN and B-CAN directly induce critical safety hazards, including engine stalls in D-mode and unintended emergency calls. Furthermore, UDS Spoofing on the C-CAN compromises the ADAS and steering domains, resulting in severe consequences like camera deactivation and steering assist limitation during driving. Even attacks classified with medium severity, such as DoS or Replay, cause degraded situational awareness and unintended actuator control. This taxonomy explicitly demonstrates that network-level intrusions in modern vehicles inevitably cascade into physical safety risks, providing a structured framework for assessing the functional safety implications of automotive cyber threats.

Accordingly, future vehicle security research should move beyond network-level detection and adopt an integrated perspective that accounts for cyber-physical interactions. Moreover, quantitatively establishing the causal relationships underlying these cyber-physical interactions remains an important open research challenge. Since the AutoHack Dataset includes both real attack injections and the corresponding physical effects observed in an actual vehicle, it provides a foundation for analyzing cyber-physical correlations that have been difficult to investigate using existing datasets.

5.2 Limitations and Future Directions

Although the proposed dataset was collected from a real vehicle platform, it is currently limited to a single vehicle model and a finite set of driving conditions. Therefore, the benchmark results obtained from the current AutoHack Dataset should not be interpreted as direct evidence of vehicle-independent generalization. CAN identifiers, payload semantics, gateway routing policies, diagnostic implementations, and message timing characteristics may vary across vehicle manufacturers, vehicle models, and E/E architectures. In addition, the included attack scenarios focus on representative attack types and do not comprehensively cover all possible attack vectors.

Nevertheless, this limitation concerns the current coverage of the dataset rather than the extensibility of the proposed dataset-construction methodology. Prior CAN IDS datasets show that comparable CAN-level attributes, attack families, and evaluation protocols can be applied across different vehicle platforms, even though the resulting identifiers, payload distributions, and functional semantics remain vehicle-specific. In this sense, AutoHack should be viewed as an initial high-fidelity multi-bus benchmark and a transferable data-construction protocol, rather than as a final vehicle-independent benchmark. The core components of our methodology—hardware-level synchronized multi-bus logging, real-time attack injection, physically verified labeling, UDS-aware diagnostic traffic collection, session-level train/test separation, and cross-ID spoofing evaluation—can be replicated on additional vehicle platforms.

Future work will extend the dataset by applying the same collection and evaluation protocol to multiple vehicle models, manufacturers, and driving environments. Such an extension would enable systematic cross-vehicle, cross-bus, and cross-manufacturer evaluation of IDS models, thereby providing a stronger basis for assessing the scalability and practical applicability of multi-bus vehicular IDS benchmarks in real deployment scenarios.

6 Conclusion

In this paper, we introduced AutoHack Dataset, an automotive cybersecurity dataset collected from a real vehicle platform. AutoHack provides a higher degree of realism compared to existing datasets by incorporating synchronized communication data from multiple CAN buses (C-CAN, P-CAN, and B-CAN), various attack scenarios, and validated physical impact information derived from actual vehicle behavior.

We conclude that IDS performance is strongly influenced by attack type, communication characteristics, and bus configuration, with aperiodic traffic and multiple buses settings revealing clear limitations of conventional periodicity- or single-bus–based approaches. These findings were further supported by external evaluation conducted as part of the AutoHack competition, demonstrating the dataset's benchmark validity and practical relevance for IDS research. Overall, our results provide empirically grounded evidence that automotive cybersecurity must be considered in close integration with functional safety, as network-level attacks can directly impact vehicle behavior.

Future work should focus on expanding the dataset to include diverse vehicle platforms and driving environments, as well as developing IDS architectures capable of effectively modeling the interactions and aperiodic communication patterns in multiple buses. AutoHack Dataset is expected to serve as a realistic and challenging benchmark to advance research in automotive cybersecurity.

References

[1] Mehmet Bozdal, Mohammad Samie, and Ian Jennions. A survey on can bus protocol: Attacks, challenges, and potential solutions. In *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*, pages 201–205. IEEE, 2018.

[2] Irina Chiscop, András Gazdag, Joost Bosman, and Gergely Biczók. Detecting message modification attacks on the CAN bus with temporal convolutional networks. Proceedings of the 7th International Conference on Vehicle Technology and Intelligent Transport Systems, 2021.

[3] Guillaume Dupont, Alexios Lekidis, Jerry den Hartog, and Sandro Etalle. Automotive controller area network (CAN) bus intrusion dataset v2. 4TU.ResearchData, 2019. https://doi.org/10.4121/uuid:b74b4928-c377-4585-9432-2004dfa20a5d.

[4] Lorenzo Guerra, Linhan Xu, Paolo Bellavista, Thomas Chapuis, Guillaume Duc, Pavlo Mozharovskyi, and Van-Tam Nguyen. Ai-driven intrusion detection systems (ids) on the road dataset: A comparative analysis for automotive controller area network (can). In *Proceedings of the 2024 Cyber Security in CarS Workshop*, pages 39–49, 2023.

[5] M. Hanselmann, T. Strauss, K. Dormann, and H. Ulmer. Canet: An unsupervised intrusion detection system for high dimensional can bus data. *IEEE Access*, 8:58194–58205, 2020.

[6] HCRL. CAN Intrusion Dataset (OTIDS). http://ocslab.hksecurity.net/Dataset/CAN-intrusion-dataset, 2017.

[7] HCRL. Car Hacking for Intrusion Detection. http://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset, 2018.

[8] HCRL. Survival Analysis for Automotive IDS. http://ocslab.hksecurity.net/Datasets/survival-ids, 2018.

[9] HCRL. CAN Signal Extraction and Translation Dataset. https://ocslab.hksecurity.net/Datasets/can-signal-extraction-and-translation-dataset, 2021.

[10] HCRL. B-CAN Intrusion Dataset. https://ocslab.hksecurity.net/Datasets/b-can-intrusion-dataset, 2022.

[11] HCRL. M-CAN Intrusion Dataset. https://ocslab.hksecurity.net/Datasets/m-can-intrusion-dataset, 2022.

[12] Kookmin University. Kookmin University Successfully Hosts 'CO-SHOW AutoHack 2025'. https://english.kookmin.ac.kr/kookmin/newsplus/616, 2025. Kookmin University News Plus. Accessed: 2026-05-04.

[13] Brooke Lampe and Weizhi Meng. Intrusion detection in the automotive domain: A comprehensive review. *IEEE Communications Surveys & Tutorials*, 25(4):2356–2426, 2023.

[14] Brooke Lampe and Weizhi Meng. can-train-and-test: A curated can dataset for automotive intrusion detection. *Computers Security*, 140:103777, 2024.

[15] Mobility Cybersecurity Lab. AutoHack2025. https://mose.kookmin.ac.kr/mose/AutoHack2025.do, 2025. Kookmin University Mobility Cybersecurity Lab. Accessed: 2026-04-30.

[16] Euclides Carlos Pinto Neto, Hamideh Taslimasa, Sajjad Dadkhah, Shahrear Iqbal, Pulei Xiong, Taufiq Rahman, and Ali A Ghorbani. Ciciov2024: Advancing realistic ids approaches against dos and spoofing attack in iov can bus. *Internet of Things*, 26:101209, 2024.

[17] Yuchan Song, Sejun Ahn, Seungjin Baek, Hyeonseong Kim, Huy Kang Kim, and Sanghoon Jeon. AutoHack Dataset and Benchmark Code: A Physically Verified Multi-Bus CAN Dataset for Intrusion Detection System Evaluation. Zenodo. https://doi.org/10.5281/zenodo.20321379, 2026. Includes the AutoHack dataset archive and the benchmark implementation code.

[18] Miki E. Verma, Robert A. Bridges, Michael D. Iannacone, Samuel C. Hollifield, Pablo Moriano, Steven C. Hespeler, Bill Kay, and Frank L. Combs. A comprehensive guide to can ids data and introduction of the road dataset, 2024.

[19] Armin Wasicek et al. CAN-MIRGU: A Dataset for Moving Vehicle Intrusion Detection. In *Proceedings of the 31st Network and Distributed System Security (NDSS) Symposium*, 2024.

A Artifact Appendix

A.1 Description & Requirements

A.1.1 How to access

The **AutoHack artifact** is permanently archived on Zenodo (DOI: https://doi.org/10.5281/zenodo.20321379), containing both **Component 1** (*AutoHack Dataset*) and **Component 2** (*AutoHack Benchmark Code*). After downloading the archive, evaluators should consult the `README.md` file in the root directory for environment setup, recommended execution order, and reproduction steps.

A.1.2 Hardware dependencies

The artifact supports two evaluation tiers. The **Minimal** configuration for Component 1 dataset verification requires 2+ CPU cores, 8 GB RAM, and 2 GB storage. The **Full** configuration for Components 1+2 benchmark reproduction requires 8+ CPU cores, 32 GB RAM, and 30 GB SSD storage. The full benchmark was tested on a 24-core CPU with 32 GB RAM and SSD storage using Docker flags `--cpus=20` and `--memory=28g`; evaluators may lower `--cpus` to match their available cores. All evaluations are CPU-bound, and no GPU, FPGA, proprietary hardware, hardware counters, or remote SSH access is required.

A.2 Set-up

A.2.1 Installation

To set up the artifact, download the two archives from the Zenodo record 10.5281/zenodo.20321379: `Autohack2025_Dataset.zip` (Component 1) and `AutoHack-Benchmark.zip` (Component 2). Extract both archives in the working directory. For local execution, place the extracted `Autohack2025_Dataset/` directory under `AutoHack-Benchmark/`. Detailed Linux and Windows commands are provided in `README.md`.

A.2.2 Basic Test

To verify the dataset integrity and ensure all statistical metrics match the paper, run the following command from the root directory:

```
python check_dataset.py
```

The script is pre-configured to load `autohack_train_both_interface.csv` and `autohack_test_both_interface.csv` from the `Interface` directory. It performs a comprehensive validation of schema consistency, null values, temporal integrity, and DLC ranges. The test is successful if the output reproduces the statistical tables and concludes with the `[PASSED]` status.

Expected Output. Upon successful execution, the terminal should display the following logs. Please confirm that the frame counts for each bus match the statistics reported in Table 4 and Table 5 of the main paper:

```
Loading Train Set from: Interface\train\autohack_train_both_interface.csv
[Train Set] Verifying data integrity...
Integrity Check PASSED for Train Set.

Loading Test Set from: Interface\test\autohack_test_both_interface.csv
[Test Set] Verifying data integrity...
Integrity Check PASSED for Test Set.

============================================================
Table 4: Detailed attack statistics by interface and session.
============================================================
[C-CAN]
Attack Type     |       Train Set |        Test Set
-----------------------------------------------------
Total Frames    |       4,026,488 |       1,678,489
Fuzzing         |          38,191 |          13,147
... (omitted for brevity) ...

============================================================
FINAL STATUS: [PASSED] Dataset is valid and Ready-to-Use.
============================================================
```

A.3 Evaluation workflow

A.3.1 Major Claims

Component 1 — Dataset Claims:

The AutoHack dataset is designed to provide a realistic and challenging environment for vehicular IDS evaluation, supported by the following claims:

(C1): **Hardware-level Synchronized Multi-bus Realism.** AutoHack provides microsecond-level synchronized logs from three separate CAN interfaces (C/P/B-CAN) and incorporates realistic non-periodic UDS interactions, distinguishing it from existing periodicity-reliant datasets.

(C2): **Physically Verified Reliability and Benchmark Challenge.** The ground-truth labels are secured through direct physical verification on a real vehicle, providing a challenging benchmark where sophisticated attacks exhibit high distributional similarity to legitimate traffic.

Component 2 — Benchmark Claims:

(C3): The dataset serves as a realistic and challenging IDS benchmark with three structural properties:

- **(C3-i)** Replay and Spoofing are harder to detect than DoS, Fuzzing, and UDS_Spoofing because they overlap more with normal traffic (RF F1: 0.6799 and 0.7864 vs. ≥ 0.93 for the others).
- **(C3-ii)** Removing non-periodic UDS traffic mainly shifts the Normal–Fuzzing boundary: Normal→Fuzzing errors increase from 966 to 5,296, while Macro/Weighted F1 remain about 0.86/0.98.
- **(C3-iii)** Joint multi-bus training lowers per-bus macro-F1 on all three buses for both Random Forest and XGBoost, while key features remain similar. Overlap-ID ratios

shift by class: Normal rises (46.12% → 53.14%), Spoofing stays stable (25.47% ≈ 25.75%), and Replay falls (41.20% → 38.52%).

A.3.2 Experiments

The following experiments reproduce the three IDS observations in Section 4.2 of the main paper. All commands are run under `AutoHack-Benchmark/`; detailed OS-specific commands are provided in `README.md`.

(D1) Observation 1 — Per-Attack Detection Difficulty
[10 human-min + ∼8 compute-min + 15GB disk]. Supports **C3(i)** and reproduces Table 8 and Figure 3. Run `python preprocess.py` and `python observation1.py`. The experiment passes if all three conditions hold: (a) Normal, DoS, Fuzzing, and `UDS_Spoofing` achieve F1 $\geq$ 0.93; (b) Spoofing and Replay are the two lowest-F1 classes and both fall below 0.80; (c) the three largest off-diagonal cells are Replay→Normal, Spoofing→Normal, and Normal→Replay, each exceeding 5,000.

(D2) Observation 2 — Non-Periodic UDS Impact
[10 human-min + ∼5 compute-min + 15GB disk]. Supports **C3(ii)** and reproduces Table 9 and Figure 4. Run `python observation2.py` using the 8-feature files from (D1). The experiment passes if (a) the Normal→Fuzzing cell increases by at least 2× compared with (D1), and (b) all other off-diagonal cells (Normal→Spoofing, Normal→Replay, Spoofing→Normal, Replay→Normal) change by less than 1.3× in either direction.

(D3) Observation 3 — Multi-Bus Characteristics
[20 human-min + ∼100 compute-min + 30GB disk]. Supports **C3(iii)** and reproduces Tables 10–11. Run `python preprocess38f.py` and `python observation3.py`. The experiment passes if all three conditions hold: (a) all RF/XGBoost single-bus→multi-bus Macro-F1 deltas are negative across B-CAN, C-CAN, and P-CAN; (b) `ID_Prev_Interval` and `ID_Frequency` remain among the top features in both single- and multi-bus settings; (c) RF overlap-ID ratios show diverging class trends: Normal increases (∼46% → ∼53%), Spoofing stays stable (∼25% ≈ ∼26%), and Replay decreases (∼41% → ∼39%).

Docker execution. D1–D3 can also be reproduced with the provided Docker image using the resource flags in Section A.1.2. Use `--cpus=8` and `--memory=24g` for the minimum full configuration, or `--cpus=20` and `--memory=28g` for the tested configuration. The same pass criteria above apply.

Determinism. All benchmark scripts fix `random_state=42`; minor numerical variation may occur across machines due to floating-point and thread-scheduling non-determinism in scikit-learn / XGBoost, but the qualitative trends supporting C3 are stable.

A.4 Notes on Reusability

To facilitate broader research in automotive security, this artifact is designed with modularity and extensibility in mind.

Reuse Scenarios for Component 1 (AutoHack Dataset).

- **Cross-Domain Analysis:** The synchronized C-, P-, and B-CAN logs support multi-domain IDS research, including cross-bus correlation and attack propagation analysis.
- **Non-Periodic Traffic Analysis:** UDS-based event-driven messages enable evaluation of anomaly detection methods in realistic non-periodic vehicle traffic.
- **Feature Engineering Benchmarking:** The raw CSV logs allow researchers to design and evaluate alternative feature sets on the same ground truth.

Reuse Scenarios for Component 2 (IDS Baselines).

- **Model Comparison:** New machine learning or deep learning models can be integrated and compared directly against the provided Random Forest and XGBoost baselines.
- **Multi-Bus Research:** The benchmark supports studies on cross-bus transfer learning, domain shift, zonal-architecture IDS design, and overlap-ID analysis.
- **Custom Feature Engineering and Education:** The raw five-column log format and reference preprocessing pipelines (`preprocess/`) support both custom feature design and educational use in reproducible IDS experiments.

License. The AutoHack Dataset is licensed under the **Creative Commons Attribution 4.0 International (CC BY 4.0)** license. The source code for evaluation scripts and IDS baselines is released under the **MIT License**.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.

ARTIFACT EVALUATED
USENIX
AVAILABLE

Bridging Two TARAs: Integrating MITRE CTSA/CRRA Prioritization into ISO/SAE 21434 for Automotive Security

Minhyuk Park
School of Cybersecurity
Korea University
mhpark99@korea.ac.kr

Yejun Kim
School of Cybersecurity
Korea University
v3locy@korea.ac.kr

Seungjoo Kim
School of Cybersecurity
Korea University
skim71@korea.ac.kr

Abstract

Threat Analysis and Risk Assessment (TARA) in ISO/SAE 21434 primarily focuses on threat identification, which makes it difficult to quantitatively prioritize countermeasures and can lead to substantial variation in results when performed manually. To address this limitation, this paper proposes AutoTARA, a framework that extends the conventional TARA methodology and automates its core steps. AutoTARA integrates the MITRE Threat Assessment and Remediation Analysis (MITRE TARA) methodology to address the lack of countermeasure prioritization and the consistency issues of conventional TARA. It also uses Meta Attack Language (MAL) to formally model vehicle systems and leverages high-level attack paths derived from simulation as reasoning guidelines for an LLM, thereby translating abstract attack paths into realistic attack scenarios. In addition, it maps VehicleLang attack steps to the MITRE ATT&CK ICS Matrix to ensure traceability to standardized countermeasures. In an evaluation using five real-world vehicle attack cases, the proposed pipeline improved the proportion of attack approaches that progressed to subsequent attack steps from 65.9% to 80.5% and increased the number of chain-progressing attack approaches by a factor of 2.8.

1 Introduction

Modern vehicles contain more than 100 Electronic Control Units (ECUs) interconnected through heterogeneous networks such as CAN, LIN, FlexRay, and Ethernet, while external connectivity through V2X communication, telematics, and OTA updates has expanded rapidly [8, 15, 19]. This expansion of connectivity has greatly increased the vehicle attack surface, making severe security threats increasingly realistic [7, 17, 18].

To respond to this changing threat landscape, the UNECE R155 regulation [25] and the ISO/SAE 21434 standard [10] were established, requiring vehicle manufacturers (OEMs) and suppliers (tiers) to establish and implement systematic cybersecurity engineering processes. In particular, ISO/SAE 21434 defines TARA as a core activity in the Concept Phase, through which a sequence of tasks is performed, from asset identification, threat scenario derivation, and attack path analysis to attack feasibility assessment, risk value determination, and cybersecurity goal formulation. However, the current TARA process has the following fundamental limitations.

Inefficiency and inconsistency of manual execution. Performing the entire TARA process manually not only consumes substantial time and human resources, but also causes the quality of results to vary significantly depending on the analyst's experience and capabilities. Even for the same system, different analysts may identify different threat scenarios and attack paths, which is a major factor reducing the reproducibility and reliability of security assessments.

Absence of criteria for countermeasure prioritization. In conventional TARA, individual attack steps mainly serve as inputs for estimating the feasibility of an overall attack path, rather than as units of analysis for assessing technique-specific susceptibility or assigning direct risk values. As a result, TARA is effective for assessing high-level threat severity and deriving security goals, but it is limited in establishing concrete technical countermeasures for detailed attack steps and in prioritizing them based on cost-effectiveness.

Limitations of attack path derivation. Recently, various studies have attempted to automate TARA. However, rule-based or datalog-based approaches [11] rely on predefined attack libraries and have difficulty deriving diverse threat scenarios. In contrast, approaches that use LLMs alone to derive attack paths suffer from hallucination and non-deterministic behavior, making it difficult to guarantee reproducibility and consistency in safety-critical domains such as vehicles, and may result in invalid attack paths.

Addressing these issues requires an integrated methodology that can simultaneously achieve standards conformance and efficient countermeasure selection. It also requires a mechanism that consistently combines formal attack path derivation with LLM-based scenario concretization, as well as a countermeasure derivation framework that preserves traceability.

In this paper, we propose a methodology and supporting tool, AutoTARA, that addresses these three challenges simultaneously. AutoTARA is a web-based prototype tool that integrates Cyber Threat Susceptibility Analysis (CTSA) and Cyber Risk Remediation Analysis (CRRA) from the MITRE methodology [24] into the conventional TARA process for the automotive domain. The main contributions of this paper are as follows:

- **Extension of the TARA methodology and automation of core steps (§4).** We propose a methodology that integrates MITRE CTSA and CRRA into the current TARA process for the automotive domain and implement it as a web-based tool, AutoTARA. This enables countermeasure prioritization based on quantitative metrics.
- **Derivation of standardized countermeasures and traceability (§4).** We systematically map the abstract attack steps defined in VehicleLang [13], a vehicle-domain-specific language based on MAL [12], to the MITRE ATT&CK ICS Matrix [23]. This enables automatic derivation of standardized countermeasure lists for each attack step and establishes traceability from attack steps to countermeasures.
- **Hybrid attack scenario concretization combining MAL and LLM (§7.2).** We propose a method that uses abstract attack paths derived from MAL-based attack simulation as structured input context for an LLM to systematically concretize attack scenarios. The proposed method reflects dependencies between attack steps and linkages between attack techniques in the LLM's scenario generation process, thereby supporting more consistent generation of attack paths.

The rest of the paper is organized as follows. Section 2 reviews the background of ISO/SAE 21434 TARA, MITRE CTSA/CRRA, and MAL. Section 3 surveys prior studies on TARA automation and methodology extension. Section 4 presents the proposed AutoTARA methodology, and Section 5 describes its implementation. Section 6 introduces the case study, and Section 7 reports the evaluation results. Section 8 discusses implications, limitations, and ethical considerations, and Section 9 concludes the paper.

2 Background

2.1 ISO/SAE 21434 TARA

ISO/SAE 21434 is an international standard that defines cybersecurity engineering activities for the electrical/electronic (E/E) systems of road vehicles. The standard presents a process for systematically identifying, assessing, and treating cybersecurity risks throughout the vehicle development lifecycle, and TARA is used as a core activity in the Concept Phase to derive cybersecurity goals and requirements. The purpose of TARA is to identify the assets that need protection and the potential damage scenarios based on the functions, boundaries, external interfaces, and operational environment of the item under analysis, and to determine risk by analyzing through which threats and attack paths such damage could occur.

TARA based on ISO/SAE 21434 is generally performed as follows. First, **item definition** is used to define the functions, components, communication interfaces, trust boundaries, and external dependencies of the system under analysis. Next, the assets and their security properties are identified, and possible **damage scenarios** that may arise if those assets are compromised are derived. Damage scenarios provide the basis for evaluating **impact** from multiple perspectives, including vehicle safety, driver and passenger privacy, vehicle operation, and financial loss. Next, **threat scenarios** and **attack paths** that could cause each damage scenario are derived. An attack path is represented as a sequence of attack steps performed by an attacker from an entry point to the target asset, and **attack feasibility** is assessed for each path. Attack feasibility is calculated based on factors such as elapsed time, attacker expertise, knowledge of the target item or component, windows of opportunity, and required equipment. Finally, **risk value** is determined by combining impact and attack feasibility, and **cybersecurity goals** and subsequent **cybersecurity requirements** are derived for unacceptable risks.

ISO/SAE 21434 TARA has the advantage of providing a standardized procedure specialized for the automotive domain that supports regulatory conformance and presents a consistent process from item definition to cybersecurity goal derivation. However, because this procedure focuses on risk assessment at the threat-scenario level, it is limited in quantifying fine-grained risk at the level of individual attack techniques (TTPs) or automatically deriving countermeasures for specific attacks. In particular, it lacks a mechanism for prioritizing countermeasures based on cost-effectiveness under limited resources.

2.2 MITRE TARA

Despite sharing the same acronym as ISO/SAE 21434 TARA, MITRE TARA [24] serves a different purpose: it is a methodology for assessing how exposed cyber assets are to an adversary's Tactics, Techniques, and Procedures (TTPs) or attack vectors, and for identifying and prioritizing effective countermeasures. This methodology consists of two main stages: CTSA and CRRA.

CTSA quantitatively evaluates how susceptible a target system or asset is to a specific TTP or attack vector. To do so, it first defines the assessment scope, target assets, threat actors, operating environment, and lifecycle phase, and models the attack surface of the target system. It then identifies candidate attack vectors based on attack catalogs and removes attack

vectors that are unlikely to be feasible given the system architecture or operating conditions. For the remaining attack vectors, it applies a predefined scoring model to calculate susceptibility and summarizes the results in the form of a susceptibility matrix or threat matrix. This matrix provides a basis for determining which assets are relatively more susceptible to which attack techniques and which attack techniques should be mitigated first.

CRRA identifies and prioritizes countermeasures based on the results of CTSA. It first selects high-susceptibility or security-critical attack vectors as remediation targets and derives applicable countermeasure candidates using mapping information between attack vectors and countermeasures. It then removes impractical countermeasures by considering system constraints, applicability, operating environment, and cost, and evaluates the effectiveness or utility and implementation cost of the remaining countermeasures. Finally, CRRA selects the set of countermeasures that can provide the greatest risk reduction under limited budget and resources, and provides traceability showing which attack vector, asset, mission, or function each countermeasure protects.

MITRE TARA has the strength of quantitatively assessing threats through CTSA TTP scoring and prioritizing countermeasures through CRRA utility/cost (U/C) analysis. It also provides traceability from attack vectors to standardized countermeasures. However, MITRE TARA was designed for IT and general cyber assets, and therefore does not directly address automotive-domain-specific E/E architectures, in-vehicle networks, or ECU-level threats, nor is it directly aligned with automotive cybersecurity regulatory processes such as ISO/SAE 21434. ISO/SAE 21434 TARA provides automotive-domain standards conformance and systematic risk identification, whereas MITRE CTSA/CRRA provides quantitative threat assessment, countermeasure prioritization, and traceability. The two methodologies are therefore complementary. In this paper, we integrate the strengths of both methodologies and present an automotive-specific TARA framework that maintains standards conformance while enabling more quantitative threat assessment and U/C-based prioritization of countermeasures.

2.3 Meta Attack Language

MAL [1, 12] is a meta-language for defining domain-specific security knowledge in the form of a domain-specific language (DSL) and automatically generating attack graphs based on that knowledge. Conventional attack-graph-based analysis has the advantage of explicitly representing dependencies between system components and attack steps, but it requires attack graphs to be manually written for each new domain or system. To address this problem, MAL allows domain-level definitions of asset types, relationships between assets, attack steps, defense mechanisms, precedence relations among attack steps, and probabilistic attributes such as time-to-compromise. Analysts use a DSL defined in MAL to create an architecture model of the target system, and the MAL simulator automatically generates possible attack paths by combining that model with the attack logic encoded in the DSL.

VehicleLang [13] is a vehicle-domain-specific DSL proposed to model the IT and E/E infrastructure of modern vehicles based on MAL. VehicleLang includes various asset types needed for automotive cybersecurity analysis, such as ECU, Firmware, CAN Network, and UDS Service, as well as the relationships among them. It also defines the attack steps and attack propagation relations that can occur for each asset based on automotive security literature and real attack cases, enabling the modeling of attack paths that extend through in-vehicle networks, firmware, sensors, and actuators.

3 Related Works

3.1 Literature Search

This section presents related work in the form of a structured literature overview. The literature review period was set from 2021, when the ISO/SAE 21434 standard was established, to 2026, and the search was conducted on Google Scholar using the queries listed in Listing 1 as of February 3, 2026. To ensure the quality of the reviewed studies, we prioritized papers indexed in authoritative security venues and academic databases such as USENIX, IEEE Xplore, ACM Digital Library, and NDSS, and also included preprints on arXiv to reflect recent research trends.

```
"Threat Analysis and Risk Assessment" AND Auto, "Threat
Analysis and Risk Assessment" AND ("LLM" OR "Large
Language Model")
```

Listing 1: Search Query

The inclusion criteria were as follows: (I1) studies that directly address TARA, (I2) studies that automate or extend part or all of the TARA procedure, and (I3) studies that propose a methodology or tool for at least one of the following: attack paths, threat scenarios, risk assessment, or mitigation derivation. The exclusion criteria were as follows: (E1) general security risk assessment unrelated to vehicles, (E2) simple vulnerability case reports that do not propose a TARA methodology, (E3) studies that deal only with vulnerabilities in external mobile apps or web services and do not clearly describe in-vehicle attack paths, and (E4) duplicate papers or papers that can be replaced by an extended version.

After removing duplicates, we collected 215 papers as the initial candidate set. We screened 59 papers based on titles and abstracts and then conducted full-text review to select nine papers on TARA automation and five papers on methodology improvement or extension for final comparison. Manual screening was necessary because search terms such

as "TARA," "automotive security," and "risk assessment" are used in broad senses, making it difficult to determine through keyword matching alone which stages of TARA were actually automated, whether mitigation prioritization was provided, and whether attack paths were formally generated.

TARA Methodology Improvement. Various studies have proposed extensions and improvements to address the limitations of the TARA methodology. Kawanishi et al. [14] proposed an asset container and CWSS-based evaluation method to compensate for the limitation of CVSS in insufficiently reflecting attacks involving direct physical access to vehicles. Zhang et al. [28] proposed CSEF, which extends conventional TARA to security testing. Benyahya et al. [5] presented TARA 2.0, which reflects privacy requirements for SAE L4/L5 autonomous vehicles. Bellini et al. [4] conducted an ADVISE Meta-ontology-based assessment for off-road and defense vehicles. Saulaiman et al. [22] proposed an approach that maps STIX to TARA to support threat intelligence sharing. However, these studies commonly focus on risk identification and assessment and do not address cost-effectiveness analysis or prioritization of countermeasures.

TARA Process Automation. To improve the efficiency and consistency of TARA execution, various automation approaches have been proposed. SecureAuto [3] is a web-based tool that combines STRIDE and the concept of lateral movement to automate part of threat identification and attack path analysis. Brankovic et al. [6] proposed a security design automation approach that combines ARAM and security pattern engineering to support risk treatment and verification stages. Rao and Roy [21] proposed a hybrid TARA engine that combines asset-based analysis and attacker-based analysis. These approaches benefit from rule-based and model-based automation, but they remain limited in handling attack combinations that were not predefined or in estimating the cost-effectiveness of mitigations.

CarVal [11] automates automotive attack surface and attack path analysis using a MulVal-based inference engine. ThreatSurf [27] automates graph-based attack surface evaluation and attack feasibility rating. THREATGET [9] supports TARA attack path analysis by generating attack trees based on preconditions and postconditions. These studies provide traceability and reproducibility through formal or rule-based path generation, but their capability to derive and prioritize standardized mitigations from attack paths is limited.

More recently, studies have begun applying AI and LLMs to TARA. Ahmad et al. [2] automatically assessed impact and feasibility using a STRIDE-based synthetic dataset and KNN/Naïve Bayes/ACO. Obradov et al. [20] proposed a multi-agent LLM framework composed of a Threat Scenario Generator, Writer Agent, and Reviewer Agent. DefenseWeaver [26] automates function-level TARA using the OpenXSAM++ model and a multi-agent framework. However, while LLM-centric approaches are strong at scenario generation, they may lack formal constraints and validation regarding whether the generated paths are actually feasible in real vehicle architectures. Our comparative evaluation is summarized in Section 7.4.

4 Methodology

4.1 AutoTARA Overview

This section explains the methodology designed to support the proposed TARA process and automate its core steps. The technical design of AutoTARA can be viewed in four main aspects, summarized as follows.

First, to quantify risk and countermeasure analysis, we integrate CTSA TTP scoring and CRRA cost-versus-utility analysis from the MITRE methodology into the automotive domain. This complements the limitation of conventional ISO/SAE 21434 TARA, which mainly remains at the level of risk identification, and enables quantitative threat assessment together with U/C-based prioritization of countermeasures.

Second, to replace manual attack path derivation, we adopt MAL- and VehicleLang-based simulation. Even without analysts manually inferring transition relations between assets and attack steps, a formally defined DSL and simulator can automatically generate reproducible attack paths for the same system model.

Third, to complement the concrete attack context that is difficult to express with MAL alone, we incorporate an LLM into the scenario-concretization stage. The attack paths derived by MAL remain at the level of abstract sequences and are therefore limited in expressing concrete attack techniques at the module level. To address this limitation, we examined the design choice of the order in which LLM and MAL should be integrated, and the comparison results are summarized in Table 1.

Table 1: Comparison of LLM–MAL integration orders.

Criterion	**LLM→MAL**	**MAL∧LLM**	**MAL→LLM**
Traceability	Low	Med.	**High**
Reproducibility	Low	Low	**High**
LLM freedom efficiency	Med.	High	**Med.–High**

Med. denotes partial satisfaction.

In the LLM→MAL approach, the LLM generates attack scenarios first, and these scenarios are then aligned with the MAL model afterward. This approach has the advantage of allowing the LLM to generate attack context relatively freely. However, if a generated scenario includes assets, relationships, or attack steps that do not exist in the MAL model, those elements must be projected back onto the MAL model. During this process, parts of the path may be omitted or abstracted

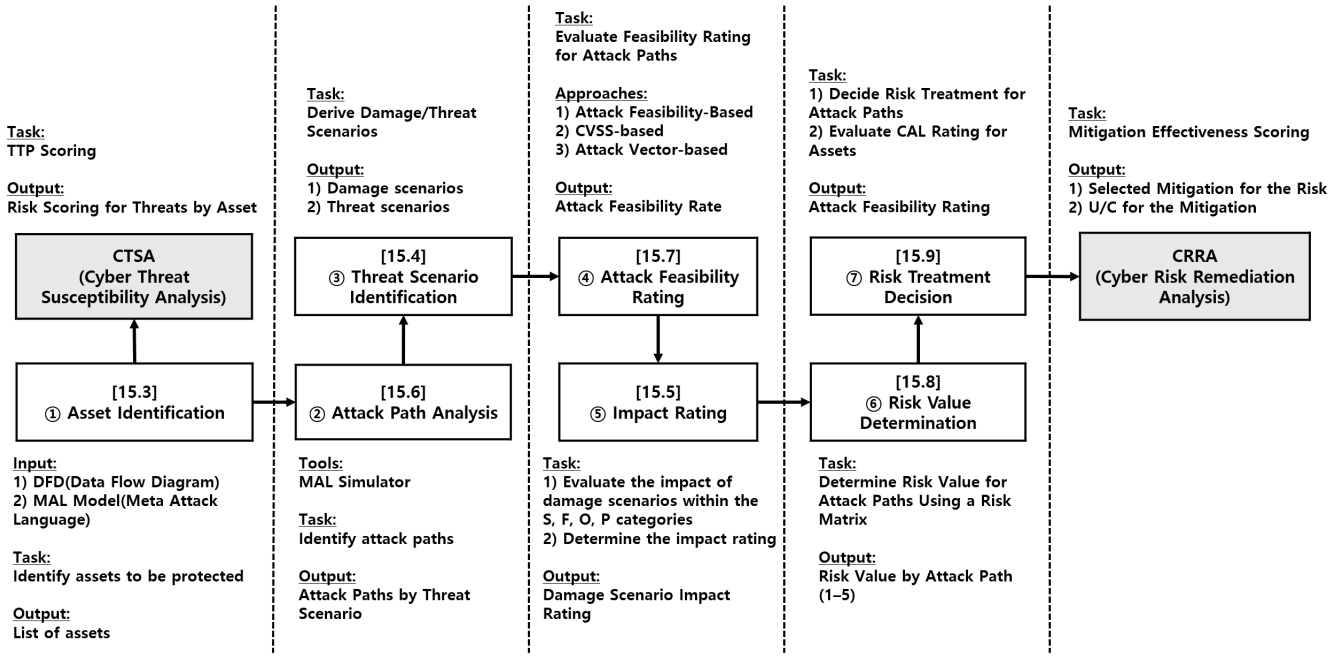


Figure 1: Overview of the proposed AutoTARA methodology

away, making it difficult to clearly trace which MAL attack step the final scenario is based on.

The MAL∧LLM approach derives MAL simulation results and LLM-generated results in parallel and then integrates them. This approach can provide the greatest degree of freedom for LLM-based expression, but because the two outputs have different levels of abstraction and semantic structures, semantic alignment issues arise during the integration process. Resolving these issues requires additional mapping rules or manual validation, and it is difficult to demonstrate that the integrated scenario fully preserves the original formal path.

By contrast, the MAL→LLM approach adopted in this study first fixes the formal attack path derived by the MAL Simulator as a seed and then constrains the LLM to concretize the natural-language attack scenario while preserving the order of the path and each path node. In this approach, the LLM does not generate the attack path itself from scratch; rather, it enriches the already derived formal path with concrete attack approaches, prerequisites, relevant vulnerabilities, or analogous cases. As a result, each natural-language scenario can be directly linked to the original MAL attack step, thereby maintaining traceability. For these reasons, this study adopts the MAL→LLM approach, which complementarily combines the formal reproducibility of MAL with the scenario-concretization capability of the LLM.

Fourth, to unify the analyst's workflow, we provide a GUI-based web tool. This GUI supports automatic Data Flow Diagram (DFD) conversion of DSL models, per-asset TTP scoring input, visual review of simulated attack paths, validation of LLM-generated scenarios, and comparison and selection of U/C-ranked countermeasure results within a single interface.

Figure 1 presents a workflow that maps these design choices onto the seven-step TARA framework, and AutoTARA aims to automate steps ❷–❹, which are central to TARA and typically require the most time.

4.2 Methodology Details

This section describes each step in sequence according to the workflow shown in Figure 1.

❶ **Asset Identification.** We first identify the assets that may lead to damage scenarios and model the system architecture using tools such as a DFD. At this stage, MAL is used to design a DSL, enabling us to enumerate assets and build a DSL for high-level vehicle architecture by predefining the attack steps that can occur at each asset and the logical attack paths that lead to subsequent attacks. Taking the system file modeled in the DSL as input, the user can then calculate a risk score for each attack step by quantitatively evaluating 12 CTSA TTP scoring factors, including Proximity, Locality, and Recovery Time. This allows prioritization of attack steps that may occur within a given asset.

❷ **Attack Path Analysis.** From the attacker's perspective, we specify the asset corresponding to the initial attack vector and the final target asset, and then simulate possible attack paths based on the system modeled in MAL. Using the MAL Simulator [16], we can derive attack paths under specific simulation settings or derive all attack paths and shortest paths while taking AND/OR conditions into account.

In general attack graph analysis, exploring all combinations of assets and attack steps can easily lead to path explosion. To address this, AutoTARA selectively derives only candidate paths that are reachable between an analyst-specified entry point and target asset. Specifically, it traces the path from the Entry to the Target and the reverse path from the Target back to the Entry, and then derives as final candidate paths only those that overlap in both directions. If no reachable path exists even after this process, the MAL Simulator reports it as an unreachable path. The abstract attack paths derived in this way are then used as input to the LLM-based path-generation stage to produce realistic natural-language attack scenarios. A verification stage for generated attack paths subsequently evaluates, through Chain-of-Thought (CoT) reasoning, whether the derived scenarios are realistic and whether the concretized attack methods are valid ways of instantiating the MAL attack path.

❸-❹ **Threat Scenario Identification and Impact Rating.** In conventional TARA, damage scenarios and threat scenarios are derived first, and attack paths are then identified on that basis. AutoTARA, however, takes the reverse approach in order to increase the level of automation: it derives damage and threat scenarios backward from the attack paths obtained through simulation. Based on the derived scenarios, the LLM-based path-generation stage estimates the impact categories Safety, Financial, Operational, and Privacy (S/F/O/P), as well as the impact ratings severe, major, moderate, and negligible, according to the assessment criteria provided by ISO/SAE 21434, and outputs them together with the rationale.

❺ **Attack Feasibility Rating.** The feasibility rating for each attack path is assessed using five factors: Elapsed Time, Specialist Expertise, Knowledge of the Item or Component, Windows of Opportunity, and Equipment. For the threats used in an attack path derived through simulation, Specialist Expertise, Equipment, and Elapsed Time are automatically calculated using the Required Skills and Required Resources items from the TTP Scoring model used in the CTSA stage, together with the average Time-to-Compromise (TTC) obtained from the simulation results. The remaining factors, Knowledge of the Item or Component and Window of Opportunity, can be accompanied by initial assessment guidance from the LLM, but because the final ratings may vary depending on the attacker type and the characteristics of the evaluating organization, the system is designed so that the user provides the final assessment. The final Attack Feasibility score for a derived attack path $\mathcal{P}$ is calculated using (1).

$$AF_{score}(\mathcal{P}) = \frac{1}{n}\sum_{i=1}^{n}(\tau_i^{local} + SE_i + Eq_i) + WoO(\mathcal{P}) + KoIC(\mathcal{P}) \tag{1}$$

In (1), n denotes the total number of attack steps included in the path. For each attack step i, τ_i^{local} is the TTC value calculated by MAL simulation and discretized according to the Elapsed Time criterion in ISO/SAE 21434; SE_i is the value obtained by mapping the Required Skills item of the TTP Scoring model to Specialist Expertise; and Eq_i is the value obtained by mapping Required Resources to Equipment. These three factors can be automatically derived by averaging across all steps in the attack path. By contrast, $WoO(\mathcal{P})$ and $KoIC(\mathcal{P})$ denote Window of Opportunity and Knowledge of the Item or Component, respectively, and are user-provided judgments for the overall attack path. This calculation separates quantitative factors that can be automated from qualitative factors that require expert judgment, thereby improving both the efficiency and the accuracy of the assessment.

❻-❼ **Risk Value & Risk Treatment.** This is the stage in which the user evaluates the final TARA outputs. Based on the results from steps ❹ and ❺, the final risk value is determined, and the attack path with the highest value is identified as the item to be considered with highest priority. The treatment option for the threat is then decided, and for assets whose risk is to be reduced, the optimal mitigation to protect that asset must be selected. The key here is to present related mitigation candidates based on the mapping between Techniques and Mitigations in the Attack Library and to help the user intuitively choose the most cost-effective mitigation by scoring the Utility/Cost (U/C) ratio of the selected mitigations using the CRRA methodology.

5 Implementation

This section describes the design details of AutoTARA. AutoTARA follows a three-tier architecture consisting of a presentation layer (front–end), an application layer (back–end), and a data layer (database and external services). Figure 2 shows the overall system architecture. The front–end, built with Vue.js, provides an intuitive interface through which analysts can model vehicle architectures, define assets, and review generated TARA outputs. The back–end, implemented with Node.js and Express, orchestrates the overall TARA workflow and is responsible for communication with external services, including the MAL Simulator and the LLM API.

Threat Identification. Assets are extracted from files modeled by the user using a DSL such as VehicleLang. Existing models written in a MAL-based DSL can also be imported, and imported DSL models are converted into a visual DFD form for ease of analysis. At this stage, TTP scoring is performed for the threats (attack steps) defined for each asset to identify the threats that require the highest-priority response. In addition, the constructed Attack Library allows analysts to intuitively check which MITRE ATT&CK Technique a specific attack step is mapped to.

Attack Path Generation. When the user selects an Entry Point where the attacker starts and a Target asset to be compromised, the MAL Simulator performs attack simulation through an API call. This produces attack paths in the form of an attack graph that can reach the target asset. The resulting attack paths are then provided to the LLM-based path-

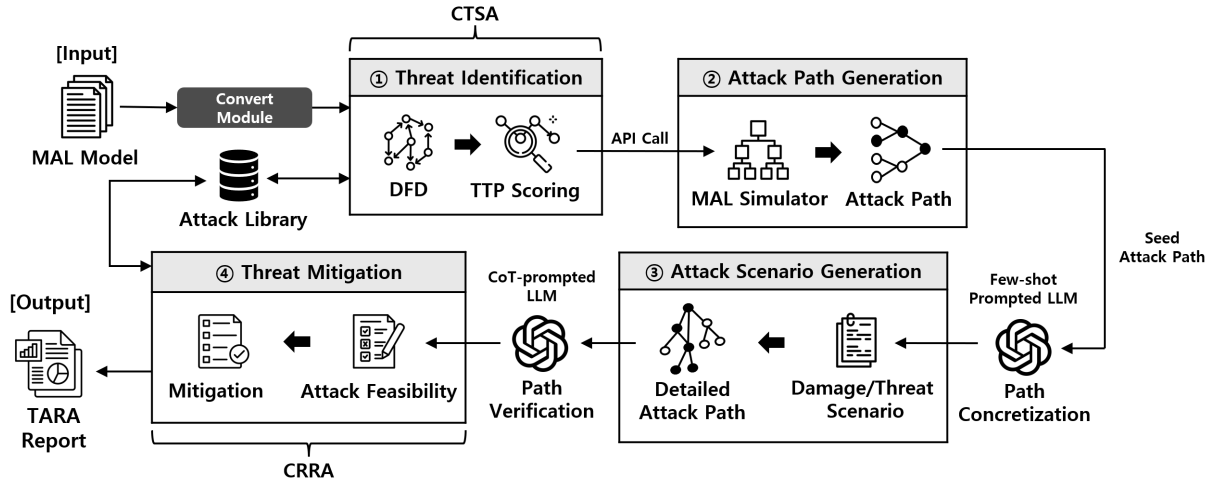


Figure 2: Architecture of AutoTARA

generation stage as input seeds that guide the generation of concrete attack scenarios.

Attack Scenario Generation. The selected attack paths are provided to a few-shot-prompted stage for LLM-based path generation. The LLM generates detailed scenarios that conform to the TARA format. Specifically, based on the CIA properties (Confidentiality, Integrity, Availability) that may be affected for the target asset, it derives damage scenarios, threat scenarios, and the attack scenarios that realize them in an integrated manner. The derived scenarios are cross-validated in a verification stage for generated attack paths to assess realism and technical plausibility.

Prompt Engineering. To use LLMs in a safety-critical environment such as vehicles, it is necessary to ensure not only the realism of the generated results but also traceability to the original attack path, reproducibility, and verifiability. Accordingly, AutoTARA does not allow the LLM to generate attack paths freely; instead, the prompt is designed so that the attack path derived by the MAL Simulator is fixed as an input guideline and then concretized into natural language in a form consistent with TARA outputs.

Scenario Generation: The output is constrained not to alter the order of the original attack path or the Path Node values, and to produce, for each path step, a concrete attack approach, a related CVE or real attack case, a mapping type, and a confidence score. If there is no direct evidence, the model is instructed not to invent CVEs or cases, but instead to distinguish concept-based or analogous-case-based approaches. The output format is also constrained by a JSON schema so that subsequent modules can parse it consistently.

Scenario Verification: The LLM verifies the plausibility of occurrence in a real automotive environment, whether any MAL-based attack steps are missing, the technical validity, and the appropriateness of evidence mapping. The verification prompt assigns a realism score from 1 to 5 to each attack approach and identifies whether path preservation, path drift, and preconditions are appropriate between the original MAL path and the generated JSON. Through this process, AutoTARA provides LLM-generated attack scenarios as a candidate list of TARA outputs, helping analysts prioritize review of attack paths with higher feasibility.

Threat Mitigation. Finally, based on the attack steps included in the verified attack scenarios, the system queries mapped countermeasures in the Attack Library and provides a list of applicable mitigations. These defense strategies are ranked by their Utility/Cost (U/C) ratio so that users can select the most efficient countermeasures under limited resources.

6 Case Study

6.1 Tesla Model S: Remote Hacking

This section describes, step by step, how the full AutoTARA workflow is applied to a real automotive cyberattack case. As a concrete example, we use the Tesla Model S remote hacking case [18] to explain the entire process from system modeling and attack path derivation to scenario generation and threat mitigation derivation.

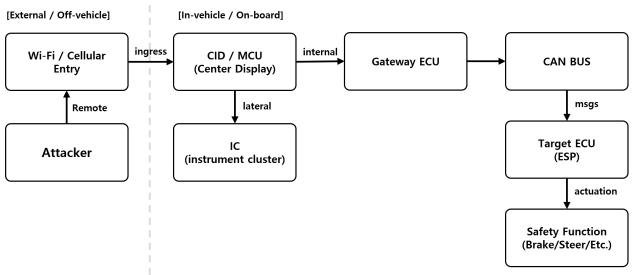


Figure 3: Tesla Model S Attack Process

This attack case involved remotely penetrating the internal vehicle network (CAN) of a Tesla vehicle without physical access and ultimately controlling the brakes while the vehicle was in motion. Figure 3 illustrates the entire process by which an attacker enters the infotainment system through an external network (Wi-Fi/Cellular), moves through the internal gateway, and gains control of the safety-critical CAN bus. The attack proceeded in three stages.

Stage 1: Compromise of the infotainment system. The attacker used a remote code execution (RCE) vulnerability in the web browser embedded in the Tesla CID (Center Information Display) as the initial entry point. The attacker then exploited a kernel vulnerability to obtain root privileges and disabled AppArmor to gain full control of the CID system.

Stage 2: Compromise of the gateway. Using the acquired root privileges, the attacker targeted a weakness in the gateway firmware update mechanism. At the time, Tesla's gateway firmware lacked an integrity verification procedure based on code signing, and the attacker took control of the gateway by transmitting and overwriting manipulated malicious firmware through the Ethernet network.

Stage 3: CAN bus message injection. Through the gateway on which malicious firmware had been installed, the attacker freely injected UDS and control messages into BDY-CAN, the vehicle's safety-critical internal bus, and thereby seized physical control functions of the moving vehicle (e.g., brake actuation).

6.2 Modeling and Simulation

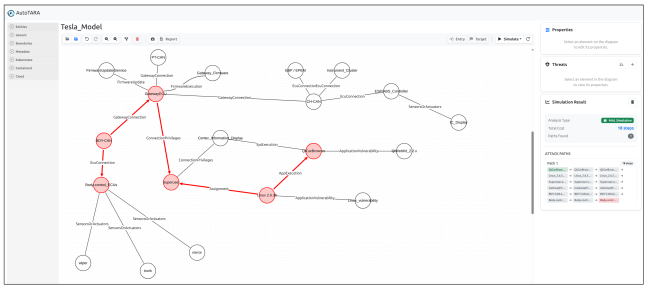


Figure 4: System Modeling with AutoTARA

Based on the Tesla case, we modeled the vehicle architecture using the VehicleLang DSL in AutoTARA. As shown in Figure 4, the system model automatically converted into DFD form intuitively shows the data flows from external network interfaces to internal ECUs, and the key assets constituting the system under analysis were identified as follows.

- **HeadUnit (CID):** the vehicle infotainment system, including a web browser and wireless connectivity
- **GatewayECU:** the central gateway ECU connecting the Ethernet domain and the in-vehicle CAN domain
- **BDY-CAN:** the in-vehicle CAN network connected to body control ECUs downstream of the gateway, through which UDS and control messages are delivered
- **Body control ECUs:** the set of control ECUs responsible for body functions such as braking, doors, and lighting, which are ultimately affected by the attack

```
Entry: QtCarBrowser:localConnect
Goal: Body Control ECUs:fullAccess
-------------------------------------------------------------
Step 1: [Application] QtCarBrowser:localConnect ->
attemptUseVulnerability -> successfulUseVulnerability ->
useVulnerability
Step 2: [SoftwareVulnerability] QtWebKit
2.2.x:attemptAbuse -> abuse -> attemptExploit ->
exploitTrivially -> exploit -> impact -> modify
Step 3: [Application] QtCarBrowser:modify -> fullAccess
Step 4: [OperatingSystem] Linux 2.6.36:localConnect ->
attemptUseVulnerability -> successfulUseVulnerability ->
useVulnerability
Step 5: [SoftwareVulnerability] Linux vulnerability:
attemptAbuse -> abuse -> attemptExploit ->
exploitTrivially -> exploit -> impact -> modify
Step 6: [OperatingSystem] Linux 2.6.36:modify -> fullAccess
Step 7: [VehicularIdentity] Superuser:attemptAssume ->
successfulAssume -> assume
Step 8: [GatewayECU] GatewayECU:authenticate ->
authenticatedAccess -> fullAccess
Step 9: [CANNetwork] BDY-CAN:adversaryInTheMiddle ->
accessNetworkLayer -> accessUninspected
Step 10: [ECU] Body control ECUs:connect ->
authenticatedAccess -> fullAccess
```

Listing 2: Simulation Result

We set the attacker's initial entry point to `localConnect` of QtCarBrowser embedded in the CID and the final goal to full access to the ECU, and then performed MAL-based simulation. The simulation result in Listing 2 shows a path by which the attacker exploits a vulnerability through QtCarBrowser, then leverages vulnerabilities in an outdated operating system, acquires access to the GatewayECU, and ultimately reaches the Body Control ECU.

The abstract attack path derived in this way is passed to the LLM-based path-generation stage as input context (Seed Attack Path) and is first concretized into a textual scenario such as Listing 3. It is then cross-validated in a verification stage for generated attack paths to assess realism and technical plausibility, and the results are presented through the AutoTARA dashboard. Finally, each attack step in the finalized attack path is linked to a mitigation list through the mapping table between the MITRE ATT&CK ICS Techniques and the entries in AutoTARA's Attack Library. A concrete example of the mapping results is provided in Appendix A.

7 Evaluation

In this section, we answer the following research questions to evaluate the effectiveness of the hybrid TARA automation approach proposed by AutoTARA using publicly disclosed real-world vehicle hacking cases.

```
(1) Set up a malicious Wi-Fi hotspot with a captive portal
that the vehicle automatically connects to, forcing the
QtCarBrowser to render a malicious page.
(2) Trigger a type confusion vulnerability in WebKit's
handling of JavaScript objects to gain read/write
primitives in memory.
(3) Execute a Return-Oriented Programming (ROP) chain to
allocate executable memory and run a reverse shell payload.
(4) Interact with vulnerable kernel interfaces or
character devices (e.g., `/dev/` nodes) exposed to the
unprivileged user.
(5) Exploit a known Local Privilege Escalation (LPE)
vulnerability in the 2.6.36 kernel, such as the RDS
protocol vulnerability or similar memory corruption.
(6) Use the kernel exploit to execute
`commit_creds(prepare_kernel_cred(0))` in kernel space,
elevating the current process to root.
(7) Extract cryptographic keys, certificates, or hardcoded
passwords stored in root-only directories (e.g., `/root/`
or `/etc/keys/`).
(8) Use extracted cryptographic keys from the root
filesystem to authenticate a diagnostic session with the
Gateway ECU.
(9) Use a diagnostic passthrough feature (e.g., J2534 or
similar proprietary mode) on the Gateway to inject raw CAN
frames onto BDY-CAN.
(10) Send spoofed CAN control messages (e.g., door unlock,
trunk open) that the Body-control ECUs accept without
authentication.
```

Listing 3: Example of a Concretized Scenario

- **RQ1:** Is providing MAL-based attack paths as input context to the LLM more effective for generating chain-progressing attack approaches than providing only the entry and target? (Section 7.2)
- **RQ2:** How do experts with experience in TARA and automotive penetration testing assess AutoTARA in terms of standards conformance, methodological validity, automation and reproducibility, scenario realism, and practical applicability? (Section 7.3)
- **RQ3:** Compared with prior work on TARA automation and extension, what differentiates AutoTARA in terms of standards conformance, formal path derivation, scenario expansion, and prioritization of threats and countermeasures? (Section 7.4)

7.1 Evaluation Setup

The case selection criteria were as follows: (1) the case should be a direct attack that exploits vulnerabilities in the vehicle itself, and (2) the attack path should be documented in enough detail at the white-paper level to enable modeling and validation through MAL. Accordingly, indirect attack cases that exploited vulnerabilities in external vehicle-management apps or web services were excluded.

According to these criteria, we selected representative attack cases for three vehicle models, as shown in Table 2, and conducted evaluation on a total of five scenarios involving BMW [7], Jeep [17], and the Tesla case from Section 6 [18]. Table 3 shows the system environment used in the experiment.

Table 2: Evaluation model.

Model	Year	Scenario ID	Entry	Target
Jeep	2015	S01	D-Bus	PCM
		S02	D-Bus	Uconnect Radio
Tesla	2017	S03	QtCarBrowser	Body Control ECUs
BMW	2019	S04	OBD-II	K-CAN
		S05	HU-Intel QNX	Central Gateway

Table 3: Experimental environment and LLM configuration used for scenario concretization.

Component	Specification / Description
CPU	Intel Core i7-1065G7
RAM	8GB
OS	Windows 11 x64
LLM Model	Gemini 3 Flash API
Parameters (Generation)	Temperature = 0.4; top-p = 0.95
Parameters (Verification)	Temperature = 0.1; top-p = 0.95

To evaluate the scenario concretization capability of AutoTARA's LLM-based path-generation stage, the path-generation prompt was configured to derive at least two concrete attack approaches for each attack step identified by MAL. As a result, an average of 3.5 concrete attack scenarios with a confidence score of Medium or higher were produced per MAL attack path. In terms of execution time, MAL-based attack path generation took about 0.25 seconds, scenario concretization using the LLM-based path-generation stage took about 25 seconds, and verification in the verification stage for generated attack paths took about 40 seconds. This shows that dozens of concretized attack scenarios and verification results can be generated within minutes, which, as also reflected in the survey results in Figure 5, is practical from the perspective of supporting TARA execution and documentation.

7.2 Qualitative Analysis

This section analyzes how providing MAL-based attack paths as input seeds to the LLM affects attack scenario concretization results. To ensure a fair comparison, we kept the system prompt, per-vehicle system model, LLM model and temperature settings, and output JSON schema identical, varying only the input condition. w/MAL denotes the setting in which the attack path derived by the MAL Simulator was provided as the LLM's input context, whereas w/o-MAL denotes the setting in which only the same entry asset and target asset were provided and the LLM was allowed to concretize the attack path on its own. We generated outputs five times for each scenario, and analyzed a total of 280 attack approaches derived from five scenarios.

The unit of analysis was divided into two categories from the perspective of attack chaining. *asset-only* refers to at-

tacks that terminate within the current asset and do not clearly establish the conditions needed to transition to the next asset, such as privilege acquisition, network access, routing, bridge or proxy setup, firmware modification, or message-injection conditions. Examples include simple DoS, information theft, UI/audio manipulation, and local persistence. *chain-progressing* refers to attacks that enable transition from the current asset to the next asset or satisfy prerequisites for subsequent attack steps, including privilege escalation, acquisition of diagnostic access, SPI/CAN raw access, gateway reconfiguration, transparent proxy setup, and network access enabled through firmware modification. Attacks occurring at the final target asset are valid in terms of goal achievement, but because they do not progress to subsequent assets, they were conservatively classified as *asset-only* in this analysis.

Table 4: Chain-oriented analysis of generated attack approaches with and without MAL guidance.

Scen.	Mode	Steps	Total	Asset only	Chain prog.	Prog. ratio
S01	w/MAL	11	55	9	46	83.6%
S01	w/o-MAL	5	25	9	16	64.0%
S02	w/MAL	5	25	8	17	68.0%
S02	w/o-MAL	2	10	3	7	70.0%
S03	w/MAL	10	50	9	41	82.0%
S03	w/o-MAL	4	20	5	15	75.0%
S04	w/MAL	6	30	5	25	83.3%
S04	w/o-MAL	3	15	6	9	60.0%
S05	w/MAL	7	35	7	28	80.0%
S05	w/o-MAL	3	15	6	9	60.0%
Overall	w/MAL	39	195	38	157	80.5%
Overall	w/o-MAL	17	85	29	56	65.9%

Table 4 summarizes the classification results. Overall, w/MAL showed a higher proportion of chain-progressing approaches than w/o-MAL, with the overall ratio increasing from 65.9% to 80.5%. In other words, providing MAL-based guidance increased the absolute number of *chain-progressing* candidates by a factor of 2.8 and improved the ratio by 14.6 percentage points. At the same time, the *asset-only* ratio decreased from 34.1% in w/o-MAL to 19.5% in w/MAL. This indicates that although MAL seeds do not completely eliminate free-form LLM generation, they guide the generated attack approaches to align more precisely with intermediate transition steps in the attack chain.

Looking at each scenario, w/MAL outperformed w/o-MAL in both the proportion and absolute number of *chain-progressing* approaches for S01, S03, S04, and S05. In particular, for S04 and S05, the *chain-progressing* ratio of w/MAL was 83.3% and 80.0%, respectively, whereas w/o-MAL remained at 60.0% for both scenarios. By contrast, in S02, w/o-MAL showed a slightly higher *chain-progressing* ratio than w/MAL, 70.0% versus 68.0%. However, w/o-MAL compressed the entire path into two steps and generated only 10 attack approaches, seven of which were classified as *chain-progressing*. In contrast, w/MAL generated 25 attack approaches across five steps, 17 of which were classified as *chain-progressing*. This shows that when judging the effect of MAL guidance, it is necessary to consider not only the simple success ratio but also the granularity of the overall attack path and the number of valid attack paths generated.

These results support the contribution of the hybrid scenario concretization proposed in this paper. Even when using an LLM alone, it can generate some *chain-progressing* attacks, but the resulting paths tend to consist of relatively simplified steps. When a MAL-based seed path is provided, by contrast, the LLM preserves the intermediate paths and attack steps from the entry asset to the target asset more precisely, while generating richer concrete attack approaches for transitions between steps. In other words, the essential effect of MAL guidance is not simply to increase the number of generated attacks, but to derive traceable *chain-progressing* candidates that can be used as TARA outputs while preserving the structure of the formally defined attack path. However, this should not be interpreted as meaning that MAL guidance completely eliminates hallucination or guarantees consistent ratio improvements in every scenario; rather, it should be interpreted as initial evidence that combining formal paths with LLMs can strengthen attack-chain traceability and path preservation.

7.3 Expert Survey

We conducted a survey with five experts who perform TARA assessment and automotive penetration testing to evaluate the practical usefulness of the proposed methodology and the attack scenarios concretized by the LLM based on MAL. The composition of the expert review team is summarized in Table 5.

Table 5: Expert review team.

Expert ID	Work Exp[1]	Task Position[2]
Expert A	1	TARA
Expert B	1	TARA
Expert C	4	Pentester
Expert D	5	Pentester
Expert E	5	TARA, Pentester

[1] Years of work experience.
[2] TARA: Threat Analysis and Risk Assessment; Pentester: vehicle penetration tester who performs testing based on TARA documents.

Responses were collected for a total of five questions, each rated on a five-point scale from 1 (Poor) to 5 (Excellent). The survey included the architecture of AutoTARA and examples

of attack paths concretized by AutoTARA from MAL attack paths. The questions were as follows.

1) Standards Conformance: The proposed methodology faithfully reflects the major procedures of ISO/SAE 21434 TARA.

2) Methodological Validity: The integration of CTSA/CRRA from the MITRE methodology can effectively address the lack of countermeasure prioritization in conventional TARA.

3) Automation and Reproducibility: MAL-based attack path derivation and the LLM-based approach can improve consistency and reproducibility compared with manual analysis.

4) Scenario Realism: Expanding and verifying attack scenarios with an LLM based on MAL-derived attack paths is useful for concretizing real automotive security threats.

5) Practical Applicability: The proposed methodology has a practical level of usability for real automotive security assessment settings.

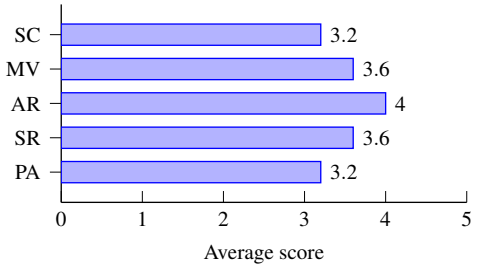


Figure 5: Survey result of expert evaluation across five dimensions (SC: Standards Conformance, MV: Methodological Validity, AR: Automation & Reproducibility, SR: Scenario Realism, PA: Practical Applicability).

Figure 5 shows the average scores for the five evaluation items. The overall average was approximately 3.52, and all items were above the neutral point (3), but the differences across items showed a consistent pattern that reveals both the strengths and the limitations of this study.

In addition to the survey, we conducted follow-up interviews about actual TARA practice with an expert who had more than five years of experience in both TARA and penetration testing in order to identify practical limitations in the field. The interview indicated that while the output examples provided in ISO/SAE 21434 TARA documents remain at a high level, actual OEMs require more fine-grained analysis results at the module level. It was also noted that even for the same architecture, the attack paths derived by organizations performing TARA can differ each time in the absence of clear criteria, and that considerable time is required to coordinate opinions on the feasibility of attack paths derived from TARA results from a penetration-testing perspective.

Interpreting the survey results in light of the interviews, the highest-scoring item, Automation and Reproducibility (AR, 4.0), can be seen as the most direct indicator of the practical strength of the proposed methodology. This suggests that MAL-based formal attack path derivation standardizes, to a certain extent, the process of identifying attack paths, which can otherwise vary greatly depending on analysts' experience and judgment, and that combining this with LLM-based scenario concretization can produce TARA outputs of more consistent quality even across repeated executions. In particular, considering the practical issue noted in the interviews that even for the same architecture, derived attack paths and their interpretations differ across organizations or analysts, the formal path seeds and structured scenario generation provided by AutoTARA can make a meaningful contribution by reducing variation between analysts and more systematically organizing candidate attack scenarios that require further review.

Methodological Validity (MV, 3.6) and Scenario Realism (SR, 3.6) received moderate scores. This indicates that the direction of integrating CTSA/CRRA and LLM-based scenario expansion was viewed positively, but that there is still room for improvement in terms of mapping consistency and the reliability of LLM outputs. In the case of SR in particular, residual concerns about LLM hallucination and the difference in abstraction level between VehicleLang and MITRE ATT&CK ICS mappings likely affected the evaluation.

The lowest-scoring items were Standards Conformance (SC, 3.2) and Practical Applicability (PA, 3.2), which were tied. The lower SC score is consistent with the interview feedback that the output examples presented by the ISO/SAE 21434 standard itself remain at a high level and are difficult to use to satisfy OEM demands for module-level granularity. In other words, while this study faithfully reflected the seven-step procedure of the standard, the scores suggest that it did not overcome the abstraction limitations inherent in the standard itself. For PA, the score appears to reflect the combined effects of the limited VehicleLang asset types, the evaluation scope restricted to three vehicle models, and the implementation maturity of the prototype stage.

Overall, the survey results show that the key value of the proposed methodology—automation and reproducibility—was recognized, while the abstraction level of the standard itself and the maturity of the tool remain tasks for future improvement.

7.4 Comparison with Related Works

This section compares the characteristics and differentiating features of AutoTARA relative to prior studies. Table 6 shows the results of comparing AutoTARA with the 14 prior studies reviewed in Section 3 according to five evaluation criteria. The five criteria used in the comparison are as follows.

- **Standard Coverage**: To what extent does the method or tool cover the seven-step TARA framework (from asset identification to risk treatment decision)?

Table 6: Comparison with related works.

Method	Standard Coverage	Prioritization (Threat/Mitigation)	Formal Path	Scenario Expansion	Automation Level
Kawanishi et al. [14]	△	◯ / ×	×	Manual	Low
Zhang et al. [28]	△	◯ / ×	×	None	Low
Benyahya et al. [5]	◯	◯ / △	×	None	Medium
Bellini et al. [4]	△	◯ / △	◯	None	Medium
Saulaiman et al. [22]	◯	× / △	×	None	Low
SecureAuto [3]	△	◯ / ×	×	None	Medium
Brankovic et al. [6]	△	× / △	×	None	Medium
Rao and Roy [21]	△	◯ / ×	×	Formula-based	Medium
CarVal [11]	△	◯ / ×	◯	Datalog / Rule-based	Medium
ThreatSurf [27]	△	◯ / △	◯	None	Medium
THREATGET [9]	◯	◯ / △	◯	Rule-based	Medium
Ahmad et al. [2]	△	◯ / ×	×	None	Medium
Obradov et al. [20]	△	△ / △	×	LLM	Medium
DefenseWeaver [26]	△	◯ / △	△	LLM + Topology	High
AutoTARA (Ours)	◯	**◯ / ◯**	◯	**LLM Hybrid**	**High**

Note. ◯: Full support; △: Partial support; ×: Not supported.

- **Prioritization (Threat/Mitigation)**: Does it produce quantitative scores to prioritize identified threats and countermeasures, respectively?
- **Formal Path**: Does it derive reproducible attack paths through explicit models and rules such as MAL, MulVal, or attack graphs?
- **Scenario Expansion**: Does it have the flexibility to expand beyond predefined rules into new threat scenarios?
- **Automation Level**: To what extent does it reduce human intervention in generating TARA outputs?

The comparison shows that prior work has strengths in particular aspects of TARA, but no prior study satisfies all five criteria simultaneously. First, only three studies—Benyahya et al. [5], Saulaiman et al. [22], and THREATGET [9]—fully cover all seven steps of ISO/SAE 21434, whereas the others provide only partial coverage focused on specific stages. Quantitative threat assessment is supported by many studies, but among the reviewed studies, we did not identify prior work that quantitatively prioritizes countermeasures based on cost-effectiveness (U/C). Formal path derivation is supported by Bellini et al. [4], CarVal [11], ThreatSurf [27], and THREATGET [9], but none of these provides a mechanism to systematically map derived attack paths to a standardized threat framework (e.g., MITRE ATT&CK) and trace them through to countermeasures. In terms of scenario expansion, DefenseWeaver [26] achieves a high level of automation using LLMs but lacks formal validation, while Obradov et al. [20] attempts a hybrid approach but relies only on cross-validation between LLMs without formal path derivation.

By contrast, the proposed AutoTARA addresses these limitations in a complementary manner. By integrating CTSA and CRRA from the MITRE methodology into the seven-step framework of ISO/SAE 21434, AutoTARA achieves both standards conformance and quantitative prioritization of countermeasures. It also combines a hybrid approach—using the formal attack path derived by the VehicleLang-based MAL Simulator as input context for the LLM—with a CoT-based verification stage for generated attack paths, enabling the derivation of concretized threat scenarios.

8 Discussion

The goal of AutoTARA is not to replace automotive security engineers with a fully autonomous TARA system, but to provide an *analyst-in-the-loop* tool that assists repetitive and documentation-centered tasks in the ISO/SAE 21434 process. Accordingly, this framework is better understood not as an independent procedure that replaces existing automotive cybersecurity development processes, but as a pre-analysis and documentation support layer used with engineer review.

Under this design, the LLM is not an independent decision-maker; rather, it plays a limited role in translating the formal attack paths derived by the MAL Simulator into natural-language scenarios that are easier for humans to review. In other words, the LLM generates drafts of concrete descriptions for threat scenarios and attack paths and proposes, in candidate form, the preconditions needed to carry out the attack, but decisions about which scenarios to adopt, whether to accept residual risk, and how to conduct real validation must be made by security analysts and the organization's development process. In addition, because AutoTARA concretizes attack paths within the attack space represented by VehicleLang and the input MAL model, it neither covers all unknown threats nor guarantees complete threat-space cov-

erage through the LLM alone. The strength of this approach lies in improving analysis efficiency by combining natural-language scenario concretization and countermeasure explanation with the traceability and reproducibility provided by rule-based and model-based methods; it is therefore more appropriate to view it as a hybrid approach that complements rather than replaces traditional formal analysis.

MAL-based constraints and the verification stage can reduce the generation of unrealistic or non-chainable attack steps compared with *unconstrained generation*, but they do not eliminate hallucination inherent to LLMs. If the input system model or formal seed path is inaccurate, later results can be distorted as well. Therefore, the evaluation in this paper should be interpreted not as demonstrating a solution to hallucination, but as initial evidence that formal constraints and follow-up verification may improve scenario quality. Moreover, the generalizability of AutoTARA depends less on the target vehicle itself than on the fidelity of the DSL and system model used to represent it. The Jeep, Tesla, and BMW cases are only initial signals of possible applicability across different vehicles; they do not guarantee general performance across all vehicle architectures, communication protocols, or unknown attack types. Future work should extend the evaluation to more diverse OEMs and E/E architectures and systematically verify how DSL coverage affects generalization performance.

In this study, we presented countermeasures by mapping the ICS Matrix of MITRE ATT&CK to VehicleLang attack steps, but this mapping is not always an exact one-to-one correspondence. VehicleLang attack steps are abstract concepts specialized for the automotive domain, whereas ATT&CK ICS techniques and mitigations are defined from a more generalized industrial control systems perspective, and differences in semantic scope and level of abstraction may therefore exist. Some mappings should thus be understood not as exact matches but as approximate interpretations that take the attack context and intent into account, and practical application requires optimization for each OEM environment. Nevertheless, this mapping is meaningful in that it presents a practical methodology for systematically identifying and tracing standardized countermeasure candidates from individual attack steps.

8.1 AI Ethics

Because AutoTARA is a security analysis tool intended for safety-critical environments, the ethical and practical boundaries of using LLMs must be clearly defined. The proposed framework is not a system that automatically executes attacks or makes immediately deployable decisions in operational environments, but an auxiliary analysis tool for research purposes. Therefore, the scenarios and countermeasures derived in this paper should not be used in production environments without verification, and responsible expert review should precede actual vehicle security decision-making.

8.2 Limitations

This study has the following limitations. First, because AutoTARA depends on the input DSL specification and MAL model, it is difficult to directly analyze asset types, architectures, communication structures, and attack steps that are not defined in VehicleLang. When attack paths are derived using VehicleLang, the range of assets is limited; even within a single asset (e.g., Infotainment), there may be various chips and modules, but the limited asset types make detailed modeling at the internal chip level difficult. Second, because the LLM is focused on concretizing formally derived attack paths, it does not guarantee the discovery of threats that are not captured in the system modeling stage or attack patterns outside the DSL. Third, the current evaluation is limited to five scenarios for which attack paths can be reconstructed from publicly available white-paper-level descriptions, and further validation is needed for more complex commercial platforms or recent software-defined vehicle environments.

These limitations also indicate directions for future work. One direction is to strengthen the domain model by extending VehicleLang asset types and attack steps so that new E/E architectures and compound attack scenarios can be covered. Another is to dynamically update the DSL itself by continuously reflecting CVEs, attack reports, and threat intelligence, thereby enabling faster adaptation to changing attack patterns. It is also necessary to more explicitly define the scope of application and failure conditions of AutoTARA through long-term evaluation across more diverse vehicles, supply-chain configurations, and communication protocols.

9 Conclusion

This paper proposed AutoTARA to address three fundamental limitations of the TARA process in automotive cybersecurity simultaneously: the inefficiency and lack of consistency of manual execution, the absence of criteria for countermeasure prioritization, and the reliability problem of attack path derivation.

AutoTARA supports quantitative threat assessment as well as U/C-based quantitative prioritization of countermeasures by complying with the seven-step ISO/SAE 21434 TARA framework while integrating CTSA and CRRA from the MITRE methodology into the automotive domain. It also establishes end-to-end traceability from individual attack steps to standardized countermeasures by systematically mapping abstract VehicleLang attack steps to the MITRE ATT&CK ICS Matrix. In addition, by using the formal attack paths derived by the MAL Simulator as input seeds to the LLM, the proposed hybrid approach mitigates the non-determinism that arises when using an LLM alone and preserves both scenario concreteness and path consistency.

In the evaluation using five real vehicle attack cases, the proposed pipeline improved the proportion of attack approaches that progressed to subsequent attack steps from 65.9% to 80.5% and increased the absolute number of chain-progressing attack approaches by a factor of 2.8. A survey of security experts also showed that all items were rated above the neutral point (3), with Automation and Reproducibility receiving the highest average score (4.0) and the overall average being approximately 3.52. The combination of formal path derivation and LLM-based scenario concretization, which is the core contribution of this study, was reflected most clearly in the highest-scoring evaluation dimension, whereas the relatively lower scores for standards conformance and practical applicability suggest the need for future domain-model expansion and OEM-specific customization. This suggests the potential of the proposed methodology as a practical support tool while also indicating that domain-model expansion and OEM-specific customization remain follow-up tasks.

In conclusion, AutoTARA presents a reproducible and scalable threat analysis framework that complementarily integrates the standards conformance of ISO/SAE 21434, the quantitative threat-assessment and countermeasure-prioritization capabilities of the MITRE methodology, and LLM-based scenario concretization grounded in MAL's formal attack paths, thereby supporting the automation, consistency, and cost-efficiency of TARA.

Acknowledgments

This work was supported by Korea Agency for Infrastructure Technology Advancement (KAIA) Grant funded by the Ministry of Land, Infrastructure and Transport under Grant RS-2021-KA162334

Availability

AutoTARA is publicly available on GitHub at https://github.com/HackProof/AutoTARA.

References

[1] Meta Attack Language: MAL. https://mal-lang.org/.

[2] Usman Ahmad, Mu Han, and Shahid Mahmood. AI-enhanced threat analysis and risk assessment for connected and autonomous vehicles. *Software Quality Journal*, 33(26):1–19, 2025.

[3] Sai Sharanya Anand, Sandeep Vijayaraghavan, and Shinu Abhi. Enhancing the connected vehicle security using the secureauto tool. In *2023 International Conference on Computational Intelligence for Information, Security and Communication Applications (CIISCA)*, pages 278–284, Bengaluru, India, 2023. IEEE.

[4] Umberto Bellini, Marzieh Kordi, Bruno Crispo, Paolo Lollini, Andrea Bondavalli, and Davide Compar. A model-based approach for cybersecurity assessment of off-road and defense vehicles. In *2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 160–167, Trento, Italy, 2025.

[5] Meriem Benyahya, Anastasija Collen, Teri Lenard, and Niels Alexander Nijdam. TARA 2.0 for connected and automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 26(8):11864–11878, Aug 2025.

[6] Boris Brankovic, Christoph Binder, Marco Ebster, Katharina Polanec, and Christian Neureiter. Towards an automated security-by-design approach in automotive system-of-systems architectures. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, pages 122–125, March 2023.

[7] Zhiqiang Cai, Aohui Wang, Wenkai Zhang, Michael Gruffke, and Hendrick Schweppe. 0-days & mitigations: Roadways to exploit and secure connected bmw cars. *Black Hat USA*, 2019(39):6, 2019.

[8] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX security symposium (USENIX Security 11)*, 2011.

[9] Sebastian Chlup, Korbinian Christl, Christoph Schmittner, Abdelkader Magdy Shaaban, Stefan Schauer, and Martin Latzenhofer. THREATGET: Towards automated attack tree analysis for automotive cybersecurity. *Information*, 14(1):14, 2023.

[10] International Organization for Standardization and SAE International. ISO/SAE 21434:2021 Road vehicles – Cybersecurity engineering, August 2021.

[11] Pengfei Jing, Zhiqiang Cai, Yingjie Cao, Le Yu, Yuefeng Du, Wenkai Zhang, Chenxiong Qian, Xiapu Luo, Sen Nie, and Shi Wu. Revisiting automotive attack surfaces: a practitioners' perspective. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 2348–2365, 2024.

[12] Pontus Johnson, Robert Lagerström, and Mathias Ekstedt. A meta language for threat modeling and attack simulations. In *Proceedings of the 13th international conference on availability, reliability and security*, pages 1–8, 2018.

[13] Sotirios Katsikeas, Pontus Johnsson, Simon Hacks, and Robert Lagerström. Vehiclelang: A probabilistic modeling and simulation language for modern vehicle it infrastructures. *Computers & Security*, 117:102705, 2022.

[14] Yasuyuki Kawanishi, Hideaki Nishihara, Hirotaka Yoshida, Hideki Yamamoto, and Hiroyuki Inoue. A study on threat analysis and risk assessment based on the ''asset container'' method and CWSS. *IEEE Access*, 11:18148–18156, 2023.

[15] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *2010 IEEE symposium on security and privacy*, pages 447–462. IEEE, 2010.

[16] mal-lang. MAL Simulator. https://github.com/mal-lang/mal-simulator, 2026. GitHub repository. Accessed: 2026-04-29.

[17] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015(S 91):1–91, 2015.

[18] Sen Nie, Ling Liu, and Yuefeng Du. Free-fall: Hacking tesla from wireless to can bus. *Briefing, Black Hat USA*, 25(1):16, 2017.

[19] Nasser Nowdehi, Aljoscha Lautenbach, and Tomas Olovsson. In-vehicle can message authentication: An evaluation based on industrial criteria. In *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*, pages 1–7. IEEE, 2017.

[20] Anja Obradov, Bogdan Pavković, Dimitrije Stojanović, and Nenad Četić. Applying multi-agent LLMs to threat analysis and risk assessment in automotive security. In *2025 33rd Telecommunications Forum (TELFOR)*, pages 1–4. IEEE, 2025.

[21] Shrinidhi Rao and Subir Kumar Roy. A novel scalable hybrid threat analysis and risk assessment engine for an automotive subsystems. In *2024 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, pages 1–6, Gwalior, India, 2024.

[22] Mera Nizam-Edden Saulaiman, Miklos Kozlovszky, and Akos Csilling. Exploring the intersection between STIX framework and TARA for sharing cyber threat intelligence for automotive systems. In *2025 IEEE 29th International Conference on Intelligent Engineering Systems (INES)*, pages 205–212, Palermo, Italy, June 2025.

[23] The MITRE Corporation. MITRE ATT&CK® Matrix for ICS. https://attack.mitre.org/matrices/ics/.

[24] The MITRE Corporation. Threat Assessment and Remediation Analysis (TARA). https://www.mitre.org/news-insights/publication/threat-assessment-and-remediation-analysis-tara, August 2020.

[25] United Nations Economic Commission for Europe. UN Regulation No. 155: Cyber Security and Cyber Security Management System.

[26] Yuqiao Yang, Yongzhao Zhang, Wenhao Liu, Jun Li, Pengtao Shi, DingYu Zhong, Jie Yang, Ting Chen, Sheng Cao, Yuntao Ren, Yongyue Wu, and Xiaosong Zhang. Automating function-level TARA for automotive full-lifecycle security. *arXiv preprint arXiv:2504.18083*, April 2025.

[27] Daniel Zelle, Christian Plappert, Roland Rieke, Dirk Scheuermann, and Christoph Krauß. Threatsurf: A method for automated threat surface assessment in automotive cybersecurity engineering. *Microprocessors and Microsystems*, 90:104461, May 2022.

[28] Haichun Zhang, Yuqian Pan, Zhaojun Lu, Jie Wang, and Zhenglin Liu. A cyber security evaluation framework for in-vehicle electrical control units. *IEEE Access*, 9:149690–149706, 2021.

A MITRE Mapping Table of Case Study

VehicleLang			MITRE ATT&CK ICS Matrix		
Asset	**Object Asset**	**Attack Step**	**Tactics**	**Technique**	**Mitigation**
GatewayECU	GatewayECU	authenticate	[TA0109] Lateral Movement	[T0859] Valid Accounts	M0801, M0936, M0915, M0913, M0947, M0937, M0932, M0927, M0926, M0918
GatewayECU	GatewayECU	authenticatedAccess	[TA0109] Lateral Movement	[T0859] Valid Accounts	M0801, M0936, M0915, M0913, M0947, M0937, M0932, M0927, M0926, M0918
GatewayECU	GatewayECU	fullAccess	[TA0111] Privilege Escalation	[T0890] Exploitation for Privilege Escalation	M0948, M0950, M0919, M0951
CANNetwork	BDY-CAN	adversaryInTheMiddle	[TA0100] Collection	[T0830] Adversary-in-the-Middle	M0947, M0802, M0942, M0931, M0930, M0810, M0813, M0814
CANNetwork	BDY-CAN	accessNetworkLayer	[TA0108] Initial Access	[T0822] External Remote Services	M0936, M0942, M0935, M0932, M0930, M0927, M0918
CANNetwork	BDY-CAN	accessUninspected	[TA0108] Initial Access	[T0886] Remote Services	M0801, M0800, M0937, M0804, M0807, M0930, M0927, M0813, M0918
ECU	Body control ECUs	connect	[TA0108] Initial Access	[T0886] Remote Services	M0801, M0800, M0937, M0804, M0807, M0930, M0927, M0813, M0918
ECU	Body control ECUs	authenticatedAccess	[TA0109] Lateral Movement	[T0859] Valid Accounts	M0801, M0936, M0915, M0913, M0947, M0937, M0932, M0927, M0926, M0918
ECU	Body control ECUs	fullAccess	[TA0109] Lateral Movement	[T0859] Valid Accounts	M0801, M0936, M0915, M0913, M0947, M0937, M0932, M0927, M0926, M0918

USENIX VehicleSec '26 Artifact Appendix: Bridging Two TARAs: Integrating MITRE CTSA/CRRA Prioritization into ISO/SAE 21434 for Automotive Security

Minhyuk Park
School of Cybersecurity
Korea University

Yejun Kim
School of Cybersecurity
Korea University

Seungjoo Kim
School of Cybersecurity
Korea University

A Artifact Appendix

A.1 Abstract

The Threat Analysis and Risk Assessment for vehicles (Vehicle-TARA) required by the ISO/SAE 21434 standard primarily focuses on threat identification, resulting in a lack of criteria for the quantitative prioritization of countermeasures. Furthermore, when the entire process is performed manually, the consistency of the results can vary significantly depending on the analyst's expertise. AutoTARA addresses the limitations of existing Vehicle-TARA methodologies by integrating compliance with the ISO/SAE 21434 standard, the quantitative prioritization capabilities of MITRE Threat Assessment and Remediation Analysis (MITRE-TARA), and the formal attack-path derivation capabilities of Meta Attack Language (MAL). This integration enables the automation of Vehicle-TARA, improves consistency, and facilitates the development of cost-effective response strategies, thereby providing a reproducible and scalable threat analysis framework for industrial environments.

A.2 Description & Requirements

To set up the artifact, you'll need Python, Node.js, and PostgreSQL, but you can easily build it using Docker Compose. For optimal performance, we recommend at least 8 GB of RAM and a network connection.

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

The artifact is available via both Zenodo and GitHub.

- https://github.com/HackProof/AutoTARA
- https://doi.org/10.5281/zenodo.20222966

A.2.3 Badges

Artifacts Available

A.2.4 Hardware dependencies

None.

A.2.5 Software dependencies

You can build the project files using Docker Compose. If you wish to build them manually, you will need Python 3.12, Node 24.14.1, and Postgres 17.

A.2.6 Benchmarks

None.

A.3 Set-up

AutoTARA is distributed as a Docker Compose application with four services:

- PostgreSQL (`db`)
- FastAPI-based MAL simulator (`simulator.server`)
- Node.js backend (`tara.server`)
- Vue frontend (`tara.vue`).

The recommended evaluation path is to run the artifact from the project root with Docker Compose, which avoids separate host-side installation of Node.js, npm, Python, or PostgreSQL.

In the current configuration, the services are exposed on ports:

- 8080 (Frontend)
- 3000 (Backend)
- 5432 (PostgreSQL)
- 8000 (Simulator)

The LLM-based `/api/v1/tara/analyze` endpoint additionally requires a valid LLM API key in `tara.server/.env`, but the basic functionality test below does not depend on any external LLM service.

A.3.1 Installation

From the project root, build and start all services with:

```
docker compose up -d --build
```

A.3.2 Basic Test

This basic functionality test verifies that the frontend, backend, database, and MAL simulator are correctly connected and operating together. The test does not require evaluation of the full system, but it confirms that the main analysis pipeline can be executed successfully from model import to attack-path generation and optional scenario expansion. For a detailed explanation with screenshots, please visit https://github.com/HackProof/AutoTARA/wiki.

Before running the test, ensure that the Docker containers are up and healthy. The required inputs are one MAL model file and one MAL language file. For testing, the sample inputs provided in the repository can be used: `Tesla-Diagram-Model.json` as the MAL model and `vehicleLang-langgraph.json` as the MAL language file. If you also want to test attack-scenario expansion is also to be tested, a valid LLM API key must be configured in `tara.server/.env`.

To perform the test, first open the frontend at `http://127.0.0.1:8080`. Import the sample MAL model and MAL language files through the **Import MAL Model** function. After the files are loaded, execute **Convert to DFD**, select the desired DFD type, and then run **Apply & Generate DFD**. Successful execution at this stage is indicated by the generation of a DFD in the frontend workspace, which confirms that the frontend, backend, and data-conversion pipeline are functioning correctly.

Next, select an asset in the generated model and configure it as the Entry Point or Target Point together with the corresponding Entry or Target Attack Step. Then execute **Simulate > Run malsim**. Successful execution is indicated by the appearance of a computed attack path in the interface. This confirms that the MAL simulator service is reachable and that simulation results are correctly returned to the AutoTARA frontend.

As an optional final check, open the dashboard at `http://127.0.0.1:8080/dashboard` and locate the generated simulation session. Execute **Expand Attack Path** for that session. If the LLM API key is configured correctly, the system should return an expanded textual attack scenario derived from the attack path. This confirms that the external LLM-based expansion component is functioning as expected. If no API key is configured, the expected behavior is that the DFD generation and attack-path simulation still succeed, while the attack-path expansion feature remains unavailable.

A.4 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.



WIP: A Benchmark for Security Evaluation of Adversarial Manoeuvre Attacks in Autonomous Driving

Adibeh Emamy
CybPass Limited

Marcello Maugeri
CybPass Limited

PingChen Lin
CybPass Limited

Aryan Pasikhani
The University of Sheffield

Abstract

This paper presents a simulation-only benchmark for security evaluation of autonomous driving agents under adversarial manoeuvre attacks. The benchmark is implemented in *MetaDrive* and supports training attacker policies, evaluating victim policies, and comparing attack outcomes under controlled driving settings. The attacker is black-box with respect to the victim and operates only through physically valid driving actions; sensor spoofing, communication tampering, model-internal access, and code tampering are excluded.

The benchmark records training logs, per-episode outcomes, aggregate safety metrics, trained policies, and evaluation logs for reproducible analysis. Preliminary results show that the benign driving performance of victim policies varies substantially across policy types and driving settings. In addition, highway experiments show that a trained PPO attacker increases the victim mission-failure rate by 10 percentage points. The benchmark provides an open-source platform for training attackers, testing victim policies, and studying adversarial manoeuvre attacks under controlled simulation conditions.

1 Introduction

Physical-world attacks against Autonomous Driving (AD) perception have shown that external perturbations can propagate from sensing errors to unsafe driving outcomes, including attacks on camera-based lane centring [12], multi-sensor fusion [3], and LiDAR-based object detection [2]. Adversarial manoeuvre attacks expose a related security problem: another road user may induce harmful outcomes through physically plausible driving behaviour rather than sensor or perception-channel manipulation [15]. This problem matters because an attacker without privileged software access may still affect the victim through ordinary vehicle control alone.

This paper addresses that gap through a benchmark for security evaluation of robustness to adversarial manoeuvre attacks. The benchmark evaluates two-agent interaction in *MetaDrive*, where victim and attacker vehicles act through shared simulator dynamics. The attacker is black-box, physical-interaction-only, and acts through standard low-level vehicle control. Accordingly, the evaluation measures degradation in victim safety outcomes. The benchmark defines the threat model, simulator interface, controlled factors, outputs, and metrics needed to compare adversarial manoeuvre evaluations across victim policies, driving settings, attacker spawn presets, and attacker designs.

Existing manoeuvre-level attacks in *CARLA* provide the closest attack-oriented reference point [15]. Safety-critical scenario generation in *MetaDrive* provides a related simulator reference point for producing difficult traffic interactions and robustness-oriented training conditions [20]. In parallel, Reinforcement Learning (RL) surveys for AD security emphasise persistent fragmentation in threat modelling, attack/defence categorisation, and reporting methodology [5]. The present benchmark addresses this methodological need through an explicit threat model, a reproducible evaluation protocol, and structured reporting.

The benchmark is motivated by four research questions:

RQ1. To what extent can a physical-interaction-only attacker degrade victim safety outcomes?

RQ2. How does robustness to adversarial manoeuvre attacks vary across DQN, DDQN, PPO, and SAC victims?

RQ3. How does victim robustness to adversarial manoeuvre attacks vary across attacker types, including learned RL and scripted heuristics?

RQ4. To what extent can learned attackers increase attack effectiveness while remaining within realistic behavioural constraints?

As a work in progress, this paper makes the following contributions toward answering these research questions:

- A simulation-only benchmark for security evaluation of robustness to adversarial manoeuvre attacks under a black-box, physical-interaction-only attacker model, with explicit capabilities and constraints.
- A coupled Partially Observable Markov Decision Process (POMDP) formulation for victim and attacker agents over

shared dynamics, with explicit definitions of state, observation, action, reward, and objectives.
- A reproducible benchmark specification and evaluation protocol for training attackers, evaluating victim policies, and recording per-episode outcomes, aggregate safety metrics, trained policies, and evaluation logs.
- Preliminary results showing that benign driving performance varies across victim policies and driving settings, and that a trained PPO attacker increases the victim mission-failure rate by 10 percentage points in the highway setting.

2 Related Work and Positioning

ACERO [15] introduced adversarial driving manoeuvres as a security threat against AD software and evaluated them in *CARLA*. The present benchmark builds on this threat model, but shifts the focus from attack discovery against concrete AD stacks to repeatable evaluation under a constrained black-box attacker model.

Within the same simulator family as the present benchmark, *CAT* [20] generates safety-critical traffic scenarios in *MetaDrive* and uses them to improve policy robustness through adversarial training. *SEAL* [16] and *SAGE* [10] extend this direction by studying generation quality, attacker realism, and controllable trade-offs between adversariality and realism. These works inform the design space, but their primary object remains scenario generation or adversarial training [14] rather than benchmark reporting for a fixed adversarial manoeuvre threat model.

Scenario datasets and generation tools provide reusable inputs for AD simulation testing: *SCTrans* [4] constructs a public scenario dataset with safety-critical traffic interactions, while *ChatScene* [19] generates safety-critical scenarios from structured knowledge and language-guided specifications. *SafeBench* [18] and *MetAdv* [8] move closer to benchmark infrastructure by supporting comparable safety or adversarial testing pipelines. The present benchmark instead fixes the attacker capability model, excluded attack channels, victim outcomes, controlled factors, and reporting outputs for black-box adversarial manoeuvre evaluation.

3 Problem Formulation

The adversarial manoeuvre evaluation problem is formulated as a two-agent interaction between a victim vehicle and an attacker vehicle in the same road scene. A frozen victim policy and an attacker policy act through standard vehicle controls, such as steering, throttle, and braking. At each step, both agents receive observations, select actions, and the road scene evolves according to both actions.

Let $\mathcal{S}$ denote the state space of the road scene, and let $s_t \in \mathcal{S}$ be the state at time t. The victim and attacker receive observations $o_t^v \in O_v$ and $o_t^a \in O_a$, and produce actions $a_t^v \in \mathcal{A}_v$ and $a_t^a \in \mathcal{A}_a$. The victim policy is denoted by $\pi_v(a_t^v \mid o_t^v)$ and is frozen during adversarial evaluation, while the attacker policy is denoted by $\pi_a(a_t^a \mid o_t^a)$.

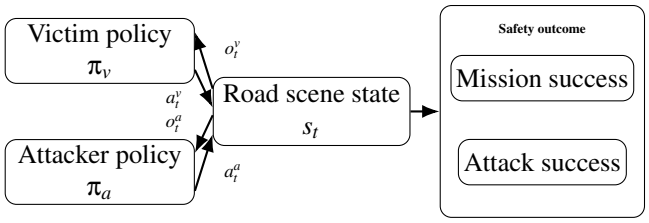


Figure 1: Overview of the attacker-victim interaction.

The shared state-transition kernel is denoted by P, following standard POMDP notation [1, 7]. The two-agent interaction induces transitions of the form

$$P(s_{t+1} \mid s_t, a_t^v, a_t^a). \tag{1}$$

The attacker-victim interaction is coupled because both agents act on the same road scene, i.e., the victim and attacker actions jointly determine the next-state transition.

To formalise partial observability explicitly, the victim and attacker are modelled as coupled Partially Observable Markov Decision Process (POMDP) agents over shared dynamics:

$$\mathcal{M}_v = (\mathcal{S}, \mathcal{A}_v, O_v, P, \Omega_v, r_v, \gamma), \quad \mathcal{M}_a = (\mathcal{S}, \mathcal{A}_a, O_a, P, \Omega_a, r_a, \gamma). \tag{2}$$

Here, Ω_v and Ω_a are observation models, r_v and r_a are the victim and attacker reward functions, and γ is the discount factor.

3.1 Threat Model

The threat model considers one victim vehicle and one attacker vehicle in the same road scene. The single-attacker assumption keeps attribution of safety outcomes explicit.

The attacker is black-box with respect to the victim: it does not access victim model weights, gradients, reward internals, training data, or privileged identifiers. The attacker observes only information available through its own driving observation and acts only through physically valid vehicle controls, such as steering, throttle, and braking. Sensor spoofing, communication tampering, map tampering, model-internal access, and code tampering are excluded. The attacker is constrained by road topology and vehicle dynamics.

3.2 Attacker Objective

The formulation separates the victim mission from the attacker objective. Victim mission success corresponds to reaching the destination without going off-road or colliding. At the same time, for a fixed victim policy, the attacker objective is to maximise the probability that the victim goes off-road or

collides without direct attacker contact:

$$\max_{\pi_a} \Pr_{\pi_v,\pi_a,P}(S=1). \tag{3}$$

Notably, victim collisions with the attacker are excluded from attack success because they involve direct attacker contact rather than indirect victim degradation. Let C_t denote a victim collision that does not involve direct attacker-victim contact. The per-step violation indicator used for attack success is therefore

$$F_t = C_t \vee \mathbb{1}[\text{victim off-road}]. \tag{4}$$

At episode level, the attacker success indicator is defined as

$$S = \mathbb{1}[\exists t : F_t = 1]. \tag{5}$$

Over N episodes, the attack success rate (ASR) is defined as

$$\text{ASR} = \frac{1}{N}\sum_{i=1}^{N} S_i. \tag{6}$$

Figure 1 illustrates this formulation.

TTC is not itself a mission violation, but a constant-velocity estimate of how soon the victim and attacker would collide if their current relative motion continued. It is used as a reward signal during attacker training and as an evaluation signal when reporting how close an interaction came to collision. Let p_t^v and p_t^a denote victim and attacker positions, and let v_t^v and v_t^a denote their velocity vectors. Define the relative position and velocity as $\Delta p_t = p_t^v - p_t^a$ and $\Delta v_t = v_t^v - v_t^a$. For non-zero separation, let c_t denote the projected closing rate, obtained by projecting the relative velocity onto the line joining the victim and attacker:

$$c_t = -\frac{\Delta v_t \cdot \Delta p_t}{\|\Delta p_t\| + \varepsilon}. \tag{7}$$

If c_t is non-positive, the projected distance is not decreasing and no finite collision time is predicted, so TTC is set to ∞. Otherwise,

$$\text{TTC}_t = \frac{\|\Delta p_t\|}{c_t}. \tag{8}$$

The attacker reward used for training follows this objective with terminal victim outcomes, TTC, victim reactions, and attacker failure penalties, as detailed in Section 4.

4 Benchmark Design

This section specifies how the benchmark instantiates the problem formulation as a reproducible experimental instrument. Building on *MetaDrive* enables procedural road generation, configurable traffic density, repeatable scenario seeds, multi-agent vehicle interaction, and direct access to simulator event flags such as arrival, off-road, and collision events. In addition, the benchmark inherits the scope of the selected *MetaDrive* environments, vehicle dynamics, observation interfaces, and event definitions. Figure 2 provides an overview of the benchmark design.

Scenario	Environment class	Road geometry characteristics
Highway	`MultiAgentMetaDrive`	Straight multi-lane segment with low interaction complexity.
Intersection	`MultiAgentIntersectionEnv`	Four-way conflict zone with crossing flows.
Roundabout	`MultiAgentRoundaboutEnv`	Curved merge/yield dynamics with multi-entry interception opportunities.
Bottleneck	`MultiAgentBottleneckEnv`	Lane-drop forced merge and congestion coupling.

Table 1: Scenario set and road geometry characteristics.

Scenarios and interfaces The benchmark is implemented in *MetaDrive* and supports four road geometries: highway, intersection, roundabout, and bottleneck. Table 1 summarises the scenario, the environment class, and the road geometry characteristics.

Victim policies receive the observation exposed by the selected *MetaDrive* multi-agent environment class. In the highway setting, this observation is provided by `MultiAgentMetaDrive`. It contains ego-state features describing the controlled vehicle and LIDAR features describing nearby road context and obstacles. The attacker receives the same raw *MetaDrive* observation augmented with six victim-relative features: relative distance, relative bearing, victim speed, attacker speed, relative longitudinal displacement, and relative lateral displacement. These features expose the victim-attacker relation needed for manoeuvre learning while preserving the black-box threat model.

Control is defined by

$$a_t = (\delta_t, u_t) \in [-1,1]^2, \tag{9}$$

where δ_t is steering and u_t is throttle/brake. For value-based victims, a discrete wrapper provides

$$\mathcal{A}_{disc} = \{\delta_i\}_{i=1}^{5} \times \{u_j\}_{j=1}^{5}. \tag{10}$$

Controlled experiment factors The benchmark fixes core environment controls to isolate attacker effects from uncontrolled simulator variation. Base controls include two agents, no respawn, a 1000-step horizon, deterministic scenario seed pools, and configurable traffic density. Spawn presets define attacker placement as an explicit controlled factor. *near*, *med*, and *far* place the attacker approximately 10, 20, and 40 metres behind the victim, respectively. *random* samples uniformly between the near and far offsets.

Reward term	Weight
Indirect victim collision	100.0
Victim off-road event	100.0
Victim mission success	-100.0
Attacker crash	-100.0
Attacker off-road event	-100.0
Following distance	0.3
Closing distance	2.0
Low TTC	2.0
Victim lateral deviation	3.0
Victim deceleration	0.5
Attacker velocity change	-0.5
Per-step cost	-0.01

Table 2: Attacker reward weights used in the proposed configuration. Top: terminal outcome terms. Bottom: Step-level terms.

Algorithm	Key settings
DQN	MLP 259-256-256-25, lr 1×10^{-4}, replay 100k, warm-up 10k, batch 64, $\gamma = 0.99$, target update 1k.
DDQN	DQN profile with Double-Q target selection/ evaluation split.
PPO	Actor/critic MLP 259-256-256, lr 3×10^{-4}, rollout 2048, batch 64, epochs 10, $\gamma = 0.99$, GAE $\lambda = 0.95$, clip 0.2.
SAC	Policy MLP 259-256-256, twin Q MLP (259+2)-256-256, lr 3×10^{-4}, replay 100k, warm-up 10k, batch 256, $\gamma = 0.99$, $\tau = 0.005$.

Table 3: Core victim hyperparameter profile. Here, lr denotes learning rate, GAE denotes generalised advantage estimation, γ is the discount factor, and τ is the SAC soft-update coefficient.

Adversarial wrapper A dedicated `AdversarialEnv` wrapper maps two-agent simulator dynamics into a single-agent attacker-learning interface while stepping the frozen victim internally at each timestep. At each step, the wrapper obtains the frozen victim action, coerces both victim and attacker actions into the *MetaDrive* action space, applies the joint action to the simulator, and returns the attacker observation and reward. During this transition, the wrapper records victim collision, victim off-road, victim arrival, direct attacker-victim collision, TTC, and reward terms under a fixed episode-termination rule.

Attacker reward The attacker reward is computed at each simulator step, and the reported episode reward is the sum of the per-step rewards. The reward combines terminal and step-level signals to encourage victim mission violations without relying on direct attacker-victim collisions or attacker failures. Terminal outcome terms reward indirect victim collisions and victim off-road events, while victim arrival, attacker crashes, and attacker off-road events are treated as attacker failures or penalties. Distance-based credit assignment reduces outcome reward when the attacker is too far from the victim to plausibly contribute to the failure. Step-level shaping terms reward low TTC, closing behaviour, following distance, victim braking, and victim lane deviation. The bottom block of Table 2 also includes a penalty for abrupt attacker motion, measured as the norm of the velocity change between consecutive steps, and a small per-step cost.

Reproducibility controls Each scenario is generated from a controlled seed pool, and each evaluation condition maps episodes deterministically into that pool. This makes the generated road instance, attacker placement procedure, victim policy, attacker policy, and driving setting repeatable for the same condition. The evaluation records per-episode outcomes, aggregate metrics, and run metadata to support auditing, reproduction, and comparison of benchmark runs.

5 Experimental Setup

Figure 3 summarises the experimental pipeline used in the current evaluation. First, victim policies are trained without an attacker to learn the driving task before adversarial interaction is introduced. Second, the trained victim policy is kept fixed while an attacker is trained against it. Finally, evaluation runs cover the selected combinations of policies, attacker spawn presets, traffic densities, and seeds defined by the evaluation matrix.

Victim and attacker training Agents are defined through a policy interface compatible with *Stable-Baselines3* [11]. In the current evaluation, victim policies are trained with DQN [9], DDQN [17], PPO [13], and SAC [6]. Their core hyperparameters are reported in Table 3. Similarly, RL attackers are trained against fixed victim policies through the same policy interface, whereas scripted attackers use fixed heuristics and require no training. The no-op attacker baseline keeps the attacker vehicle present but applies zero steering and zero throttle or braking, providing a passive baseline for attacker comparisons.

The action space follows the benchmark interfaces defined in Section 4. DQN and DDQN use the discrete 25-action wrapper, whereas PPO and SAC use continuous control. In our experiments, both victim training and PPO attacker training use 500,000 simulator steps before evaluation. Each completed evaluation condition is then run for 100 episodes.

Evaluation matrix The highway attacker evaluation fixes the road geometry to highway and the victim to the PPO pol-

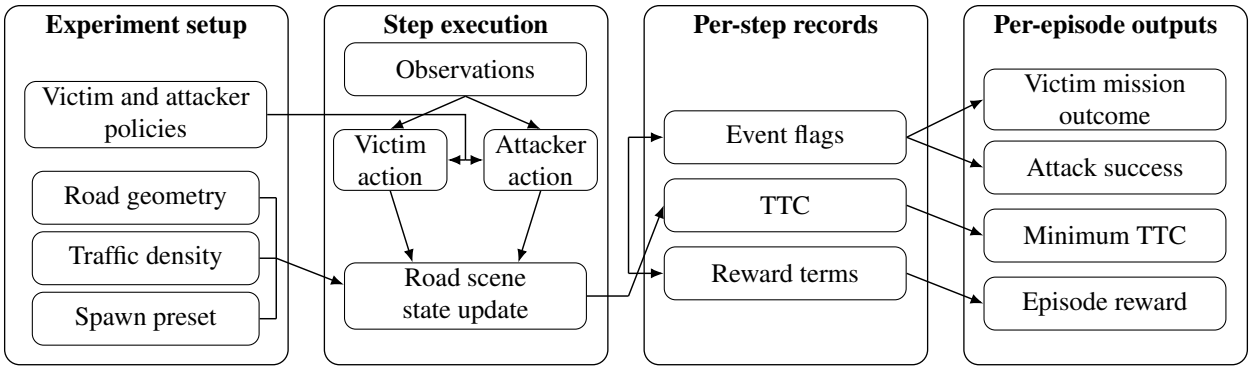


Figure 2: Benchmark design from experiment setup to per-episode outputs.

icy trained on highway. The PPO highway victim used in this comparison is trained with traffic density 0.1. The highway attacker evaluation then considers two traffic settings for the attacker comparison. The primary controlled setting removes traffic and compares the no-op attacker baseline with the PPO attacker trained under the *med* attacker spawn preset, to directly measure attacker influence. A second setting repeats the same comparison with traffic density 0.1 to test whether the attacker effect persists when traffic is present. Both settings use start seed 42 for *MetaDrive* procedural road generation and the same deterministic highway scenario pool for both attacker configurations.

Roundabout is used to study how victim learning changes under a more complex road geometry. The reported roundabout evaluation is limited to driving competence before an attacker is introduced. It varies the victim algorithm across DQN, DDQN, PPO, and SAC. Victim driving competence is reported over the final 100 training episodes of each victim training run.

Evaluation metrics The evaluation metrics follow the outcomes defined in Section 3. Victim driving competence is reported through mission success, off-road rate, and crash rate. The primary comparison is between mission success and attack success: mission success requires the victim to reach its destination without a safety violation, whereas attack success requires a victim off-road event or a victim collision that does not involve direct contact with the attacker. Specifically, ASR is the fraction of episodes in which this attack-success event occurs. To characterise successful attacks beyond their frequency, the evaluation also considers Time-to-failure (TTF) and TTC. TTF measures how many steps pass before an attack succeeds, and is therefore computed only over successful-attack episodes. TTC measures the time to a possible attacker-victim collision under the current relative motion, and is used to interpret close interactions rather than to define attack success. Attacker crash and reward metrics are reported alongside victim outcomes to distinguish victim degradation from attacker failure. To evaluate attacker realism, the evaluation also uses attacker crash rate and attacker velocity change as indicators of whether the attack remains plausible and, indirectly, stealthy.

6 Results

This section reports the conducted experiments used to evaluate victim driving competence, attacker learning, and highway attack effectiveness. The roundabout attacker evaluation remains work in progress; the current roundabout evidence is therefore limited to victim driving competence.

6.1 Experiment 1: Victim Driving Competence in Highway and Roundabout

This experiment evaluates victim driving competence before adversarial evaluation. Table 4 reports mission success, off-road rate, and crash rate over the last 100 training episodes for each completed victim run. On highway, the PPO victim reaches 86.0% success, with 15.0% off-road rate and no crashes, providing a suitable baseline for the attacker evaluation. At the same time, PPO falls to 36.0% success in roundabout, with 56.0% off-road rate and 15.0% crash rate. Notably, DQN and DDQN do not obtain competence in the roundabout road geometry, with 0.0% success rate and 100.0% off-road rate. Finally, SAC provides the strongest roundabout victim, with 57.0% success, 29.0% off-road rate, and 13.0% crash rate. Table 4 summarises these findings. The corresponding victim training success curves are shown in Appendix A.

`Takeaway.` Preliminary results indicate that victim driving competence depends on the interaction between road geometry and policy-learning algorithm.

6.2 Experiment 2: Attacker Learning Dynamics

This experiment checks whether the PPO attacker learns to induce victim failures in the reported highway setting. The

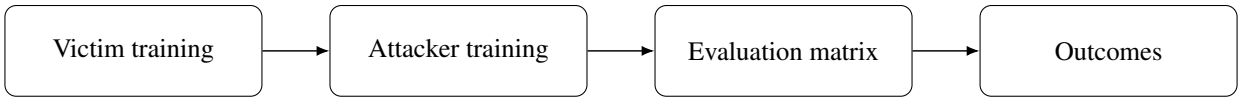


Figure 3: Experimental pipeline from victim training to outcome reporting.

Victim run	Success	Off-road	Crash
PPO highway	86.0%	15.0%	0.0%
DQN roundabout	0.0%	100.0%	0.0%
DDQN roundabout	0.0%	100.0%	0.0%
PPO roundabout	36.0%	56.0%	15.0%
SAC roundabout	57.0%	29.0%	13.0%

Table 4: Victim driving competence over the last 100 training episodes.

attacker is trained against the fixed PPO highway victim under the *med* attacker spawn preset for 500,000 simulator steps. Without traffic, victim off-road failures occur in 49.0% of the final 100 training episodes. The same final window has mean episode reward 25.8 and mean episode length 104.2 steps. With traffic density 0.1, victim off-road failures occur in 51.0% of the final 100 training episodes, while victim crashes remain at 0.0%. The corresponding final window has mean episode reward 82.3 and mean episode length 187.6 steps. These outcomes indicate that the PPO attacker remains able to operate in traffic and induce victim off-road failures. The corresponding attacker training curves are shown in Appendix A.

`Takeaway.` The PPO attacker learns to induce victim off-road failures in the highway setting and remains effective with traffic density 0.1.

6.3 Experiment 3: Highway Attack Effectiveness

This experiment quantifies safety degradation caused by a learned attacker in the highway environment. Table 5 reports outcomes for a PPO victim under the no-op attacker baseline and PPO attacker over 100 recorded episodes per configuration.

Without traffic, the PPO attacker increases ASR from 0.49 to 0.59. This increase is entirely due to victim off-road failures, as victim crashes remain at 0.00 in both attacker configurations. With traffic density 0.1, the same comparison remains positive but weaker, with ASR increasing from 0.42 to 0.46. Nevertheless, the PPO attacker still increases ASR, but the effect is smaller than in the no-traffic setting. This is consistent with a more complex interaction in which traffic adds constraints and occasional attacker failures. At the same time, the traffic setting introduces a small rate of victim crashes involving traffic, which is obviously absent without traffic.

Metric	No-op attacker	PPO attacker
ASR_{crash}	0.00	0.00
$ASR_{offroad}$	0.49	0.59
ASR	0.49	0.59
Mean TTF (steps)	40.51	40.14
Mean episode length	100.49	89.26
ASR_{crash}	0.00	0.01
$ASR_{offroad}$	0.42	0.45
ASR	0.42	0.46
Mean TTF (steps)	85.57	99.57
Mean episode length	203.09	207.30

Table 5: Highway evaluation outcomes for the PPO victim under the *med* attacker spawn preset. Top: without traffic. Bottom: traffic density 0.1.

The corresponding ASR summary and TTF distributions are shown in Appendix A.

`Takeaway.` The PPO attacker increases ASR in both highway traffic settings, while traffic reduces the effect size and introduces a small rate of victim crashes involving traffic.

6.4 Experiment 4: Trade-offs between Effectiveness and Realism

For an adversarial manoeuvre to be security-relevant, it must increase attack effectiveness without relying on frequent attacker crashes or visibly irregular driving. Therefore, this experiment evaluates how attack effectiveness is balanced with attacker realism through crash rate, mean reward, and velocity change for the same highway configurations considered in Experiment 3. Table 6 reports these metrics for the no-op attacker baseline and PPO attacker.

Without traffic, the PPO attacker increases mean reward from 32.02 to 38.88 and does not increase crash rate. However, velocity change rises from 0.08 to 3.55, indicating more abrupt attacker motion than the no-op attacker baseline. With traffic density 0.1, the PPO attacker remains effective, but traffic constrains the manoeuvre. As a consequence, mean reward is lower than the no-op attacker baseline, which is consistent with traffic acting as an obstacle to the attacker. Nevertheless, crash rate remains low at 0.02, and velocity change is lower than in the no-traffic PPO attacker setting. These results indicate that the PPO attacker remains effective under traffic while partially preserving realism and stealthiness.

Metric	No-op attacker	PPO attacker
Crash rate	0.00	0.00
Mean reward	32.02	38.88
Velocity change	0.08	3.55
Crash rate	0.00	0.02
Mean reward	95.21	68.87
Velocity change	0.08	1.44

Table 6: Attacker-side realism indicators for the PPO victim under the *med* attacker spawn preset. Top: without traffic. Bottom: traffic density 0.1.

Takeaway. Traffic reduces PPO attacker effectiveness, but the learned policy still maintains a favourable trade-off between attack effectiveness and realism.

7 Discussion

The benchmark provides a controlled setting for studying how LIDAR-based driving policies respond to adversarial manoeuvres performed through ordinary vehicle control.

The results suggest that such policies can be susceptible to physically valid attacker behaviour, but they also show that attack evaluation is easy to misinterpret without a structured protocol. In particular, victim competence, attacker effectiveness, and attacker realism must be separated before an attack outcome can be interpreted. This separation is important because an observed failure can otherwise reflect weak victim driving, an unrealistic attacker, or a genuine adversarial effect.

The roundabout results illustrate this point. Before adversarial evaluation is meaningful in that road geometry, the victim policy must first demonstrate sufficient driving competence. The current results indicate that this competence depends on the policy-learning algorithm, which motivates further training, diagnosis of the DQN and DDQN failures, and evaluation of additional victim policies.

The current benchmark models a single attacker vehicle and uses a LIDAR-based observation interface. The same protocol can be extended along the controlled factors already exposed by the benchmark, including road geometry, traffic density, attacker spawn preset, and procedural seed. Future work should extend the training budget for both victims and attackers, diagnose the DQN and DDQN roundabout failures, and evaluate further victim–attacker algorithm combinations. The evaluation matrix should also be expanded to cover additional attacker spawn presets, traffic densities, and road geometries.

8 Ethical Boundary

This work is intended for defensive security evaluation. All experiments are simulation-only and are used to assess robustness to adversarial manoeuvre attacks under a constrained, physical-interaction-only attacker model. The paper does not provide real-world deployment procedures for adversarial behaviour, and the benchmark is intended for controlled simulation and robustness assessment only.

Because the paper studies safety-critical failure induction in autonomous-driving settings, ethical and legal risks were considered during the design of the evaluation protocol. To mitigate these risks, the study excludes attacks outside physical vehicle manoeuvres (e.g., sensor spoofing, communication tampering, and code tampering) and reports results only in simulation.

9 Conclusion

This paper presents a reproducible benchmark for adversarial manoeuvre attacks in Autonomous Driving. It instantiates the attacker-victim interaction in *MetaDrive*, fixes the attacker capability and success criteria, and reports victim outcomes together with attacker-side realism indicators.

The current evaluation shows that a PPO attacker can increase highway ASR against a PPO victim while remaining within ordinary vehicle control. It also shows that this result must be interpreted together with attacker crash rate, velocity change, and the driving competence of the victim policy.

The victim results further show that driving competence depends on the road geometry and policy-learning algorithm. This makes victim validation a necessary part of adversarial manoeuvre evaluation, especially before extending the attack study to roundabout settings.

The benchmark is intentionally scoped to the conducted experiments. Its main contribution is to make adversarial manoeuvre evaluation repeatable, explicit, and extensible across road geometries, traffic settings, attacker spawn presets, and policy combinations.

These results support further systematic testing of adversarial manoeuvre attacks against Autonomous Driving policies.

References

[1] Karl Johan Åström. Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 1965.

[2] Yulong Cao, S. Hrushikesh Bhupathiraju, Pirouz Naghavi, Takeshi Sugawara, Z. Morley Mao, and Sara Rampazzi. You can't see me: Physical removal attacks on LiDAR-based autonomous vehicles driving frameworks. In *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[3] Yulong Cao, Ningfei Wang, Chaowei Xiao, Dawei Yang, Jin Fang, Ruigang Yang, Qi Alfred Chen, Mingyan Liu, and Bo Li. Invisible for both camera and lidar: Security of multi-sensor fusion based perception in autonomous driving under physical-world attacks. In *2021 IEEE Symposium on Security and Privacy (SP)*, 2021.

[4] Jiarun Dai, Bufan Gao, Mingyuan Luo, Zongan Huang, Zhongrui Li, Yuan Zhang, and Min Yang. Sctrans: Constructing a large public scenario dataset for simulation testing of autonomous driving systems. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024.

[5] Ambra Demontis, Srishti Gupta, Maura Pintor, Luca Demetrio, Kathrin Grosse, Hsiao-Ying Lin, Chengfang Fang, Battista Biggio, and Fabio Roli. Security of deep reinforcement learning for autonomous driving: A survey, 2025.

[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[7] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 1998.

[8] Aishan Liu, Jiakai Wang, Tianyuan Zhang, Hainan Li, Jiangfan Liu, Siyuan Liang, Yilong Ren, Xianglong Liu, and Dacheng Tao. Metadv: A unified and interactive adversarial testing platform for autonomous driving. In *Proceedings of the 33rd ACM International Conference on Multimedia*, 2025.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[10] Tong Nie, Yuewen Mei, Yihong Tang, Junlin He, Jie Sun, Haotian Shi, Wei Ma, and Jian Sun. Steerable adversarial scenario generation through test-time preference alignment. In *The Fourteenth International Conference on Learning Representations*, 2026.

[11] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 2021.

[12] Takami Sato, Junjie Shen, Ningfei Wang, Yunhan Jia, Xue Lin, and Qi Alfred Chen. Dirty road can attack: Security of deep learning based automated lane centering under Physical-World attack. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[14] Aizaz Sharif and Dusica Marijan. Adversarial deep reinforcement learning for improving the robustness of multi-agent autonomous driving policies. In *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*, 2022.

[15] Ruoyu Song, Muslum Ozgur Ozmen, Hyungsub Kim, Raymond Muller, Z. Berkay Celik, and Antonio Bianchi. Discovering adversarial driving maneuvers against autonomous vehicles. In *Proceedings of the 32nd USENIX Security Symposium*, 2023.

[16] Benjamin Stoler, Ingrid Navarro, Jonathan Francis, and Jean Oh. Seal: Towards safe autonomous driving via skill-enabled adversary learning for closed-loop scenario generation. *IEEE Robotics and Automation Letters*, 2025.

[17] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

[18] Chejian Xu, Wenhao Ding, Weijie Lyu, ZUXIN LIU, Shuai Wang, Yihan He, Hanjiang Hu, DING ZHAO, and Bo Li. Safebench: A benchmarking platform for safety evaluation of autonomous vehicles. In *Advances in Neural Information Processing Systems*, 2022.

[19] Jiawei Zhang, Chejian Xu, and Bo Li. Chatscene: Knowledge-enabled safety-critical scenario generation for autonomous vehicles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[20] Linrui Zhang, Zhenghao Peng, Quanyi Li, and Bolei Zhou. CAT: Closed-loop adversarial training for safe end-to-end driving. In *Conference on Robot Learning (CoRL)*, 2023. MetaDrive-based closed-loop adversarial scenario generation.

A Additional Plots

This appendix reports supporting plots for the conducted experiments.

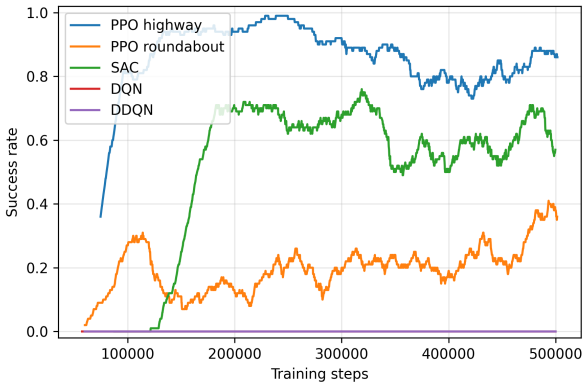


Figure 4: Victim training success rate over training steps. PPO is shown for both highway and roundabout; SAC, DQN, and DDQN are shown for roundabout.

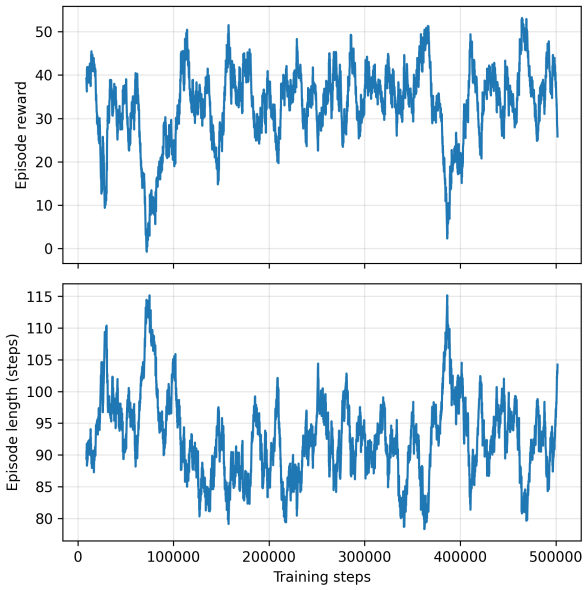


Figure 5: PPO attacker training curves in the highway setting under the *med* attacker spawn preset.

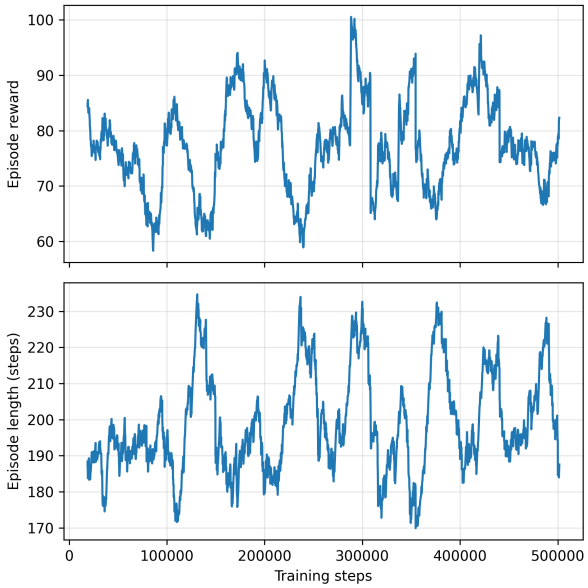


Figure 6: PPO attacker training curves in the highway setting with traffic density 0.1 under the *med* attacker spawn preset.

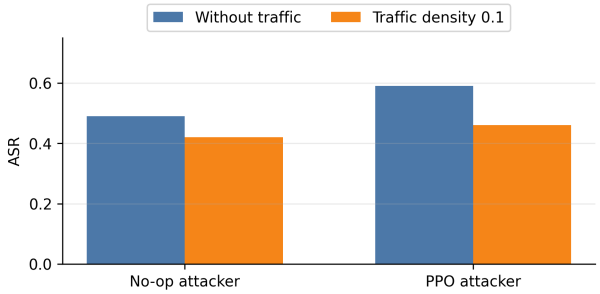


Figure 7: ASR summary for the highway evaluation under the *med* attacker spawn preset.

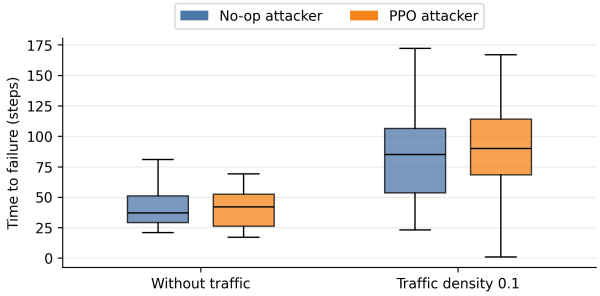


Figure 8: TTF distributions for the highway evaluation under the *med* attacker spawn preset.

B Artifact Appendix

B.1 Abstract

This artifact contains the source code, configuration files, pre-trained models, and pre-computed outputs needed to reproduce the experimental results reported in the paper.

B.2 Description & Requirements

The artifact consists of approximately 3,000 lines of Python source code organised under `src/`, experiment configuration files under `configs/`, and pre-trained victim and attacker policy checkpoints under `outputs/paper/models/`. A single CLI command (`python main.py --config configs/paper.jsonc`) executes the complete pipeline: victim training, attacker training, and evaluation with metric computation. Post-processing utilities under `scripts/` generate result tables and figures from the evaluation outputs.

B.2.1 Security, privacy, and ethical concerns

The artifact runs simulation-only experiments in the MetaDrive driving simulator. No network connections are required during execution, no real vehicles or physical systems are involved, and no destructive operations are performed on the host machine. The trained attacker policies are reinforcement-learning models for a simulated environment and are not deployable on real vehicles.

B.2.2 How to access

The artifact, including the source code, configuration files, and pre-trained models, is archived on Zenodo at https://doi.org/10.5281/zenodo.20530026.

B.2.3 Badges

The artifact has been submitted for the Available, Functional, and Results Reproduced badges.

B.2.4 Hardware dependencies

- **CPU**: Modern x86-64 or ARM64 processor (8+ cores recommended).
- **Memory**: 8 GB RAM minimum, 16 GB recommended.
- **Disk**: 20 GB free space for the virtual environment, trained models, and evaluation outputs.
- **GPU**: Not required.
- **Display**: Not required (rendering is disabled by default).

B.2.5 Software dependencies

- Python 3.11.
- Pinned dependencies in `requirements.txt`.
- **Ubuntu only:** OpenGL runtime package `libgl1`.

Tested on Ubuntu 22.04 (x86-64) and macOS 26 (ARM64). On macOS, `--device cpu` is the recommended configuration.

B.2.6 Benchmarks

None.

B.3 Set-up

B.3.1 Installation

1. Download the artifact archive and extract it.
2. Create a virtual environment and install dependencies:

```
sudo apt install -y libgl1  # Ubuntu only
python3.11 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

B.3.2 Basic Test

Run the smoke-test configuration, which trains a short PPO victim and evaluates it with a no-op attacker:

```
python main.py \
  --config configs/test.jsonc \
  --experiment-name test --device cpu
```

Expected wall time is approximately 2 minutes. The command should create `outputs/test/experiment.json` and metric files under `outputs/test/eval/`.

B.4 Evaluation workflow

B.4.1 Major Claims

(C1): Victim robustness is algorithm-dependent: on the roundabout scenario, SAC achieves 57% success (mean reward 191.5), PPO 36% (177.5), while DQN and DDQN achieve 0% success with 100% off-road rate. On the highway, PPO achieves 86% success (mean reward 238.9). See Experiment 1 in Section 6.1, Table 4, and Figure 4.

(C2): The learned PPO attacker converges to positive mean reward (≈25.8) on the highway environment with a fixed spawn geometry over 500,000 training steps. See Experiment 2 in Section 6.2 and Figure 5.

(C3): In the highway evaluation with the *med* attacker spawn preset, a learned PPO attacker raises ASR from 0.49 to 0.59 without traffic, and from 0.42 to 0.46 with traffic density 0.1. See Experiment 3 in Section 6.3, Table 5, and Figures 7–8.

(C4): The highway evaluation reports attacker crash rate, mean reward, and velocity change to interpret the trade-off between attack effectiveness and attacker realism. With traffic density 0.1, the PPO attacker shows lower velocity change and low crash rate, indicating less abrupt

manoeuvres in traffic while remaining effective. See Experiment 4 in Section 6.4 and Table 6.

B.4.2 Experiments

All paper experiments are generated by one pipeline run, followed by dedicated scripts under `scripts/` for E1–E4.

(E0): Pipeline execution (for all experiments) [10 human-min + 12–18 compute-hr + 20 GB disk]: Runs victim training, attacker training, and evaluation in a single command, producing all outputs required by E1–E4.
Preparation: Complete the Set-up section and ensure 20 GB free disk.
Execution:
```
source .venv/bin/activate
python main.py \
  --config configs/paper.jsonc \
  --experiment-name paper \
  --num-jobs 1 --device cpu
```
With `-num-jobs 12`, wall time is approximately 8–12 hours on a 16-core machine.
Results: Outputs appear under `outputs/paper/`, including models and evaluation results.

(E1): Victim learning quality [5 human-min + 1 compute-min] (validates C1):
Preparation: Verify that victim training directories exist under `models/victims/`, one per algorithm and scenario (e.g., `ppo_highway`, `sac_roundabout`), each containing a `training_episodes.csv` file.
Execution:
```
python scripts/make_victim_training_table.py
```
Results: The script prints success rate, off-road rate, crash rate, and mean reward per victim algorithm over the last 100 training episodes. The expected values match Section 6.1: on the roundabout, SAC success ≈57%, PPO ≈36%, DQN/DDQN ≈0% with 100% off-road; on the highway, PPO success ≈86%.

(E2): Attacker learning dynamics [5 human-min + 1 compute-min] (validates C2, Figure 5):
Preparation: Verify that attacker training directories exist under `models/attackers/`, named by attacker algorithm, victim algorithm, scenario, and spawn geometry (e.g., `ppo_vs_ppo_highway_med`), each containing a `training_episodes.csv` file. Victim training directories from E1 must also be present.
Execution:
```
python scripts/\
plot_victim_training_curves.py
python scripts/\
plot_attacker_training_curves.py
```
Results: The first script produces Figure 4 (victim training success curve) and the second produces Figures 5–6 (attacker reward and episode length curves), both saved under `outputs/plots/`. The highway attacker curve should show convergence to positive mean reward (≈25.8 over the last 100 training episodes).

(E3): Highway attack effectiveness [5 human-min + 1 compute-min] (validates C3, Table 5, Figures 7–8):
Preparation: Verify that evaluation directories exist under `eval/`, named by scenario, victim, attacker, spawn geometry, and seed, each containing `metrics.json` and `episodes.csv`.
Execution:
```
python scripts/make_eval_table.py
python scripts/plot_attack_success_rate.py
python scripts/\
plot_time_to_failure_distribution.py
```
Results: The first script prints Table 5, where ASR increases from 0.49 to 0.59 without traffic and from 0.42 to 0.46 with traffic density 0.1. The plotting scripts produce Figure 7, which compares ASR for the no-op and PPO attackers, and Figure 8, which shows the TTF distributions for the same two traffic settings.

(E4): Effectiveness–realism trade-off [5 human-min + 1 compute-min] (validates C4, Table 6):
Preparation: The same evaluation directories from E3 are used. Verify that the evaluation outputs contain the metrics used for attacker crash rate, mean reward, and velocity change.
Execution:
```
python scripts/make_realism_table.py
```
Results: The script prints Table 6 (attacker crash rate, mean reward, and velocity change). The expected Table 6 output without traffic is crash rate 0.00 for both attackers, mean reward 32.02 and 38.88, and velocity change 0.08 and 3.55. With traffic density 0.1, the expected output is crash rate 0.00 and 0.02, mean reward 95.21 and 68.87, and velocity change 0.08 and 1.44.

B.5 Notes on Reusability

The benchmark is designed around three main extension points:

- **Scenarios:** Add a JSON file to `configs/scenarios/` specifying the MetaDrive environment class and configuration, then reference the scenario name in `configs/paper.jsonc`.
- **Victim/attacker algorithms:** Add policy classes that inherit from Stable-Baselines3 algorithms and register them in the agent registry. Discrete and continuous action-space mapping is handled automatically.
- **Reward design:** Add or modify presets in `src/reward/config.py`. The reward engine evaluates a configurable weighted sum of 16 components.

B.6 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.

Perspective-Shift Attacks Against Optical Perception Sensors: A Novel Attack Vector on LiDAR and Camera

Marco Calipari*[1], Michael Kühr*[1], Dominik Kulmer[2], Maximilian Luedecke[1], Mohammad Hamad[1], Sebastian Steinhorst[1]
[1]*TUM School of Computation, Information and Technology, Technical University of Munich*
[2]*TUM School of Engineering and Design, Technical University of Munich*

Abstract

Autonomous vehicles rely heavily on optical perception sensors, such as LiDARs and cameras, to understand their environment. Both sensors have been the subject of study in the automotive security community, with attacks ranging from physical hijacking, by introducing adversarial stimuli within the sensing range, denoted as the field of view, to crafting specialized adversarial objects designed to fool the underlying machine learning model in tasks such as object detection. This paper introduces a novel class of physical attacks, namely Perspective Shift Attacks, that aim to manipulate the sensors' field of view to induce dangerous driving maneuvers in life-critical scenarios. We show that using Direction Turning Film, an optical material that bends light by a predefined angle, can adversarially alter the field of view of optical perception sensors. Unlike prior attacks that inject adversarial signals or modify the environment, our approach stealthily alters the perceived sensing region without significantly degrading detection confidence performance. We present a digital simulation framework and validate the attack through controlled laboratory experiments and real-world deployments on a real autonomous driving research vehicle. Results show that even moderate field-of-view shifts of 20° significantly impact critical tasks, including LiDAR-based odometry and camera-based lane detection, potentially triggering dangerous maneuvers, such as unintentionally invading the opposite lane, while having only a minor effect on the confidence of object detection algorithms, despite altering object positions. Finally, we propose detection mechanisms that leverage physical-layer artifacts and semantic consistency checks to mitigate perspective-shift attacks. Our findings reveal perspective-shift attacks as an unexplored and practical attack surface, underscoring the need for robust sensor security in autonomous vehicles.

*These authors contributed equally to this work.

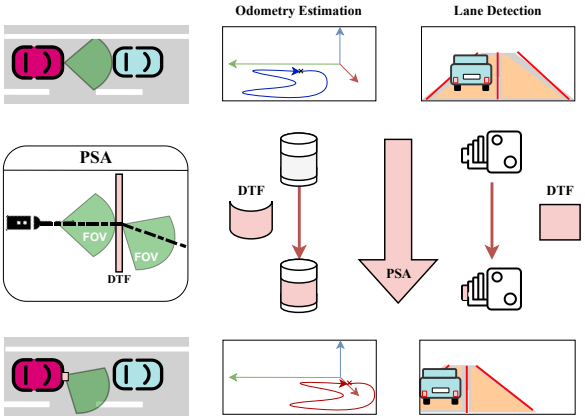


Figure 1: Perspective-Shift Attacks induce a shift in the FoV of Optical Perception Sensors without causing a noticeable degradation in the performance of tasks such as object detection. As a result, the AV cannot detect the attack, which may lead to unsafe behavior in safety-critical scenarios due to incorrect decision-making. For instance, the vehicle fails to accurately estimate its trajectory or street lanes.

1 Introduction

Autonomous Vehicles (AVs) have become a reality [40, 54]. A key enabler of this progress is the perception sensor suite, which allows an AV to gather information from its environment and forward it to subsequent processing modules. Among perception sensors, Light Detection and Rangings (LiDARs) and cameras have received considerable attention in the context of autonomous vehicle security [1], leading to the identification of multiple attack categories targeting these sensors [29, 51].

Within these categories, physical attacks have been proposed with the aim of disrupting system behavior. Most existing attacks focus on impairing the functionality of the AV ei-

ther by directly targeting the sensors themselves, e.g., through blinding attacks [44], or by targeting the underlying Machine Learning (ML) models via the introduction of artifacts into sensor data [13, 58]. Such artifacts are typically introduced using adversarial objects, such as patches or stickers, placed in the environment [13, 24, 35]. Consequently, an alternative and largely unexplored direction is to affect AV functionality by deliberately altering the sensor's perceived sensing region, namely its Field Of View (FoV).

The FoV is an inherent characteristic of every Optical Perception Sensor (OPS), and no prior physical attack has explicitly targeted shifting it. By deliberately altering the sensing region, it is possible to introduce a perspective shift that changes the area in which perception algorithms operate, without degrading their performance. This contrasts with existing physical attacks, in which the FoV remains unchanged, and the attack aims to perturb sensor measurements within it. By contrast, simply shifting the sensor's perspective can elicit unintended behaviors. We refer to this new attack vector as a *Perspective-Shift Attack (PSA)*. To achieve this PSA, we investigate optical films as a class of materials capable of modifying the angle of incoming light [36], shaping light into specific shapes [38], or increasing perceived brightness [37].In contrast to attacks based on translucent patches [64] or stickers [13], PSAs do not aim to directly induce misdetections or misclassifications. Instead, the goal is to shift the sensor's optical perspective in a way that covertly manipulates the spatial perception of the environment, thereby misleading other driving functions and leading to hazardous, potentially safety-critical decisions.

To instantiate our optical film-based PSA, we employ Direction Turning Film (DTF) [36], which bends incoming light by a predefined angle. Depending on the material properties, this light-bending effect can be achieved with varying deflection magnitudes [4]. As this manipulation affects all subsequent processing modules in the autonomous driving pipeline, we evaluate its impact on multiple representative algorithms, including odometry estimation, lane detection, and object detection. Using digital simulations, we first assess the effect of adversarial FoV changes on both LiDAR and camera sensors. We then conduct extensive real-world evaluations in laboratory environments and on real vehicles, targeting individual sensors and demonstrating a multi-modal attack. Based on these evaluations, we further derive detection mechanisms to protect LiDARs and cameras against PSAs.

In summary, we make the following contributions:

- **Identify PSAs as an attack vector against optical perception sensors**: We present a class of physical attacks that deliberately alter the perceived FoV of OPSs, and refer to them as PSAs. In contrast to prior attacks, we do not aim to attack the sensors [44] or underlying ML models by placing adversarial patches [13, 55] or introducing additional objects [27]. Instead, our goal is not to degrade perception performance, but to cause a controlled shift in the perceived FoV. We demonstrate this effect using DTF, an optical film that deflects light by a predefined angle.

- **Evaluate the effect of PSAs**: We present a digital simulation to investigate the effect of an altered FoV on the sensor data. Specifically, we show the effect for lane detection and odometry estimation algorithms. We provide our simulation and measurement data as an open-source code[1].

- **Real-world evaluation of our DTF-based PSA**: To validate the attack in the real world, we evaluate the impact of a shift in perspective in real vehicles, individually for LiDAR and camera, as well as from a multimodal setting, where we show the effect of coherently applied PSA.

- **Detect PSAs**: We propose two methods to detect FoV changes, one for LiDARs and one for cameras. Our methods analyze the sensed data and do not require any change to the hardware setup.

2 Background

This section provides background on existing physical adversarial attacks, the DTF material, the FoV of OPSs, and odometry estimation and lane-detection algorithms.

2.1 Physical Attacks on Autonomous Vehicles

Due to the high number of environmental sensors used in AVs, they are vulnerable not only to digital attacks known from connected vehicles but also to physical attacks that tamper with sensor input [43], leading to unexpected behavior and potentially safety-critical maneuvers. Previous studies have shown that especially perception sensors, such as cameras and LiDAR sensors, are vulnerable to such physical attacks [11, 16, 18]. As shown in Table 1, the majority of well-known physical attacks aims to disrupt the performance of a specific algorithm within the AV stack, targeting perception algorithms to hide, alter, or create objects in the object detection algorithms [13, 19, 34, 35, 55, 62, 63]. Highly specialized attacks, such as [8, 42], target subsequent stages of the multi-stage pipeline of AVs [56], focusing on algorithms such as Simultaneous Location And Mapping (SLAM). The high degree of specialization required for these attacks to be successful categorizes them as non-function-agnostic attacks, because the attack-crafting mechanism is tailored to the internals of the targeted algorithm. Moreover, they targeted a single sensor, either camera or LiDAR, and they have not transferred the specific attack to multi-sensor scenarios. On

[1] https://github.com/tum-esi/PSA

Table 1: Comparison of existing physical attacks against OPSs on AVs with our proposed DTF-based.

Attack	Multi-Sensor	Stealth	Phys. Access to Victim	Function-Agnostic	Selectable Victim
Eykholt *et al.* [13]	✗	✗	✗	✗	✗
Zhu *et al.* [62]	✗	✓	✗	✗	✓
Lovisotto *et al.* [35]	✗	✗	✗	✗	✓
Liu *et al.* [34]	✗	✓	✗	✗	✓
Zhu *et al.* [63]	✗	✗	✗	✗	✓
Cao *et al.* [5]	✗	✓	✗	✗	✓
Kobayashi *et al.* [27]	✗	✗	✗	✗	✗
Chen *et al.* [8]	✗	✗	✗	✗	✗
Nagata *et al.* [42]	✗	✗	✗	✗	✗
Zolfi *et al.* [64]	✗	✗	✓	✗	✓
Li *et al.* [31]	✗	✓	✓	✗	✓
Zhou *et al.* [61]	✗	✓	✓	✗	✓
Ours	✓	✓	✓	✓	✓

the other hand, [31, 61, 64] require physical access to the victim to deploy the attack, because they expect to place either a translucent patch or a lens in front of the victim's sensor. In terms of human stealthiness, namely those attacks that are not visible to human eyes, many of the works we researched lack this property [6, 8, 13, 27, 35, 63, 64]. Finally, the possibility of having a selectable victim allows for attacking a specific vehicle rather than affecting all passing vehicles. This is different from works such as [6, 8, 13, 27], where the attack medium is deployed in the environment and potentially affects all the passing AVs.

As shown in Table 1, our PSA distinguishes itself from the existing approaches by being effective in a multi-sensor setup with LiDARs and cameras, stealthy to humans, and function-agnostic against selectable victims with only one-time physical access.

2.2 Direction Turning Film

DTF is a film used mainly for illumination applications, specifically for wall-washing or for avionic displays, where the viewing angle needs to be adjusted to ensure visibility [36]. These applications exploit the material's bending property, i.e., collimated light passing through an optic at a specific angle. Such an effect is possible due to the material's design, which features asymmetric microprisms embedded directly within a polycarbonate sheet, allowing it to bend the outgoing light. The behaviour of the material can be characterized via Snell's law of refraction [25], which describes the interaction of electromagnetic waves at the interface of two materials with different refractive indices.

2.3 Optical Perception Sensors in Autonomous Vehicles

OPSs are a class of sensors used in AVs to perceive the environment in which the vehicle operates. This category includes sensors such as LiDARs and cameras. Because these sensors rely on optical elements, their perception of the environment ε is inherently constrained by their optics and physical design. We define the environment ε as the physical scene observable by an OPS at a given time and location. As a result, each OPS observes only a limited portion of the surrounding area in the horizontal and vertical directions, referred to as the FoV, adapted from the SAE-J1050 standard [9]. The FoV of an OPS is then defined as the tuple of these angular intervals as shown in Equation 1.

$$\mathrm{FoV}_{\mathrm{OPS}} := (H_{\mathrm{FoV}}, V_{\mathrm{FoV}}) \tag{1}$$

The horizontal (H_{FoV}) and vertical FoV (V_{FoV}), as shown in Equation 2, are constrained by the technical specification of the respective OPS.

$$H_{\mathrm{FoV}} = [\theta_{\min}, \theta_{\max}], \quad V_{\mathrm{FoV}} = [\varphi_{\min}, \varphi_{\max}] \tag{2}$$

While ε is unbounded in principle, the sensor's optical and physical constraints restrict observation to a finite subset $\varepsilon^* \subset \varepsilon$. It is defined in Equation 3, where $\theta(p)$ and $\varphi(p)$ denote the azimuth θ and elevation φ of a point p expressed in the sensor reference frame.

$$\varepsilon^* = \{p \in \varepsilon \mid \theta(p) \in H_{\mathrm{FoV}} \wedge \varphi(p) \in V_{\mathrm{FoV}}\} \tag{3}$$

Finally, the output x of an OPS operating in environment ε is modeled in Equation 4, where $\mathcal{S}$ denotes the sensing function that maps the observable portion of the environment ε^* to the sensor's measurement space.

$$x = \mathcal{S}(\varepsilon^*) \tag{4}$$

2.4 ICP-Based LiDAR Odometry

Iterative Closest Point (ICP) based odometry allows AVs to achieve precise pose estimation via processing consecutive LiDAR point clouds, without relying directly on feature extraction [60]. Being agnostic to features allows ICP based methods to perform better in environments where features are sparser or less present, relying additionally on the geometry of the LiDAR scans. The core idea behind ICP based odometry is to iteratively compute a transformation matrix that minimizes the point-to-point distance among successive scans. A representative example of ICP based odometry is Keep It Small and Simple - Iterative Closest Point (KISS-ICP) [52], which presents a pipeline that works across different domains and LiDAR models with a single set of parameters, making it the reference for odometry estimation in the scientific literature.

2.5 Lane Detection

Lane detection is an important perception task in AVs, and its results are typically used for subsequent tasks like correct

lane keeping [59]. In order to function correctly, an AV needs to derive its position within the lane [41]. By changing the FoV, this estimation can be disrupted. Many lane detection ML models, such as HybridNets [53], provide both detected lanes and the drivable area. The result will be a segmentation map that classifies the pixel positions in an image as either a lane or a street.

3 Threat Model

In the following sections, we will introduce the threat model used throughout the remainder of this paper and the attacker's knowledge and capabilities.

3.1 Attack Goal and Characteristics

The goal of our PSA is to change the FoV of an OPS, specifically targeting LiDAR sensors and cameras. In contrast to existing attacks [13, 19, 34, 35, 55, 62, 63], PSAs do not aim to hide, alter, or create objects in the sensor data by changing the environment ε but rather change the FoV of the targeted sensor, thus leading to wrongly derived positions of objects. This is possible by the introduction of an angular shift Δ_{PSA} which is positive when the shift is applied to the left in the horizontal plane or upward in the vertical plane, and negative for the opposite directions.

Practically, this translates into the FoV by adapting Equation 2 into Equation 5 for a horizontal shift, and into Equation 6 for a vertical shift. It is important to note that only one of these shifts can be applied at a time.

$$H_{\text{FoV,PSA}} = [\theta_{\text{min}} + \Delta_{\text{PSA}}, \theta_{\text{max}} + \Delta_{\text{PSA}}] \tag{5}$$

$$V_{\text{FoV,PSA}} = [\varphi_{\text{min}} + \Delta_{\text{PSA}}, \varphi_{\text{max}} + \Delta_{\text{PSA}}] \tag{6}$$

This alters the subset of perceived points ε^* described in Equation 3, and consequently the sensor output x from Equation 4.

The primary characteristic of our attack is that the shift Δ_{PSA} is not induced by mechanical manipulation of the sensor, such as physically rotating it, but rather by light deflection through an optical film, specifically DTF. Consequently, the applied DTF causes the sensor to perceive the environment as if it were being rotated, forcing the AV to compensate for the rotation, and causing safety-critical situations.

Our DTF-based PSA can be characterized as follows:

- **Stealthiness:** An important characteristic of our attack is the stealthiness of the transparent patch against humans. This is achieved by using transparent DTF, which involves adding only a transparent, inconspicuous film to the OPS, with a minimum size required to fully cover the respective sensor.
- **Limited Physical Access:** Once the DTF is placed on the OPS of the target vehicle, no further physical access is needed.
- **Controlled Target:** While adversarial patches on remote objects, such as traffic signs [13], might affect multiple or even unintended assets, our attack is directly deployed on the intended target, allowing the attacker to specifically control a selected target.
- **Minor Adversarial Impact on Perception:** PSAs change the FoV of an OPS with ideally no impact on the detection or classification performance of the perception models used in the respective data processing. While introducing a film on the sensor will create some optical artifacts or noise, this is only a side effect with ~~very~~ limited impact on the perception modules.
- **Multi-Modal Attack:** Our attack is effective both against LiDAR sensors and cameras, making it effective for systems that use sensor fusion.
- **Independent of Illumination:** Many attacks are known to work only in specific illumination conditions [3, 10, 15]. By contrast, our attack is effective independently of the environmental illumination.

3.2 Attacker Knowledge and Capabilities

We assume an attack with **no prior knowledge of the perception modules** used in an AV ("black-box attack"). This includes details of the ML model weights, architecture, and the stages within the perception module, and distinguishes our attack from similar attacks that apply patches to cameras [31].

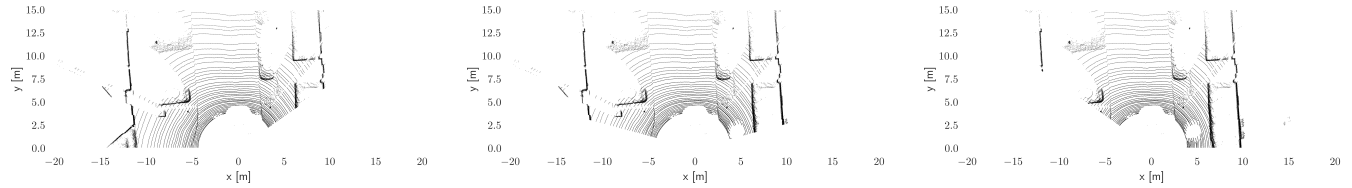
As highlighted in our attack characteristics, the attacker requires **one-time physical access to the target vehicle** to place the DTF on the respective OPS. This is similar to other attacks that apply patches on the cameras [17, 31, 50, 64].

An attacker needs to **have minimum system knowledge** of the targeted AV, specifically the positions of the OPSs. Such knowledge can be obtained through visual inspection of the target vehicle or public sources of reference vehicles. We emphasize that only the sensor positions are required, without any further technical details.

Lastly, for our DTF-based PSA, an attacker needs to have **enough DTF to cover the respective sensor area**, which is typically limited to a relatively small area. Depending on the desired bending angle, specific DTF material is required. For our attack deployment, we will use a material which introduces an offset of 20° [36].

4 Digital Attack Simulation

We create a digital simulation of our DTF-based PSA that operates on pre-captured real data to simulate the effect of the FoV shift on specific tasks for the targeted OPSs.



(a) FoV left shift (b) Benign FoV (c) FoV right shift



(d) FoV left shift (e) Benign FoV (f) FoV right shift

Figure 2: Representative frames from the digital simulation with a simulated 20° PSA. The top row shows LiDAR scans from the KITTI Odometry benchmark, while the bottom row presents the camera views from a real-world driving scenario.

4.1 Digital Simulation

The perspective shift of out DTF-based PSA depends on the material properties and cannot be changed by the attacker after deployment. The magnitude and direction of the shift, however, can be chosen so that an attacker can deploy the DTF to shift the perceived sensor data up, down, left, or right, by applying a different film. In our simulation, we use a perspective shift of 20°, as this is physically realizable with commercially available DTF [36]. However, the effect of other shifts can be simulated with our tool.

To simulate the FoV shift at the LiDAR, we digitally modify the KITTI odometry dataset [14], which allows for the dynamic adjustment of the FoV in conjunction with the description of OPS provided in Equation 4. As a result, it is possible to create custom sequences from the KITTI odometry to be fed to any algorithm, as shown in Figures 2a-2c.

For the camera, our simulation uses a dual-fisheye representation of spherical images, as is often captured by omnidirectional cameras. Based on a freely selectable center point, our simulation creates a curvilinear projection of a specified FoV. This projection mimics the FoV of a camera in an environment ε, represented by the spherical image. Figures 2d-2f depict the benign image in the center and both simulated directions of the FoV. Although not depicted, the upward and downward shifts can also be simulated.

4.2 Digital Evaluation

For both sensors, we evaluate the impact of the attack on left- and right-shifted sensor data, as these shifts are more interesting to attackers than up- and downward shifts, since they can lead to misbehavior, such as incorrect steering commands. As representative AV algorithms, we use a KISS-ICP-based odometry estimation [52] and evaluate its relative displacement for the LiDAR, and HybridNets [53] for lane detection and its displacement in cameras.

4.2.1 LiDAR

As previously described, we evaluate the PSA-induced relative displacement, using the KITTI odometry dataset [14]. Figure 3 shows the comparison of the effect of a shifted FoV. In our digital evaluation, we consider a Horizontal FoV of 150° oriented in the vehicle's driving direction in the KITTI odometry dataset, in line with the specifics of commercially available LiDARs, and the vertical FoV remains untouched. As per the displacement, we consider 20° in both the right and left directions.

It is worth noticing that the displacement is not computed with respect to the KITTI Ground Truth (GT), but relative to a reference run executed without any shift for the specific FoV under analysis. This approach enables the evaluation of displacement while avoiding the bias introduced by cropping the original point cloud. The evaluation is conducted on all KITTI odometry sequences that provide a Global Navigation Satellite System (GNSS) GT to compensate for possible noise introduced by the algorithm. The results reported in Figure 3 show the average displacement for each sequence. The KISS-ICP employs a robust optimization strategy to mitigate potential errors in pose estimation. Specifically, the registration process enforces temporal consistency by incorporating prior pose estimates, particularly the two preceding poses. The results indicate that sequences with larger average dis-

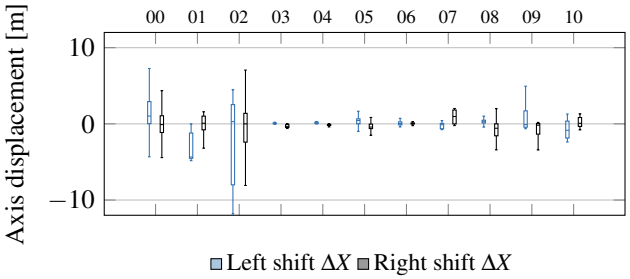


Figure 3: Result of a run of the LiDAR digital simulation. We reported the measured displacement, in meters, of the ΔX coordinate of the estimated pose relative to the benchmark reference towards the KITTI driving direction.

placements generally correspond to turning maneuvers (e.g., sequences 00, 01, and 02). In contrast, in sequences where the vehicle follows a predominantly straight trajectory (e.g., 03 and 04), the average displacement remains negligible. This behavior can be attributed to the fact that, when the FoV is sufficiently wide and the applied shift is moderate, the extracted features remain largely consistent, resulting in limited degradation of the pose estimate. Conversely, during turning maneuvers, the shift has a more pronounced impact on the registration process, leading to significantly corrupted pose estimates.

4.2.2 Camera

We use the street scenery from Figures 2d to 2f, captured with a dual-fisheye camera as a ground-truth for our digital simulation, and perform a HybridNets-based lane detection [53]. Figure 4b shows the benign lane detection, while Figures 4a and 4c depict the simulated left- and right shift by 20°, introduced by the PSA. The individual lines are color-separated to differentiate them. While the detected lanes are qualitatively similar, they have different positions that can result in incorrect steering commands, since functionality like lane departure warnings and other lane-related tasks do not compensate for a changing FoV but assume a fixed mounted sensor for their control algorithms [2, 41]. The left shift of Figure 4a results in a right shift of the vanishing point, and the right shift in Figure 4c leads to a left shift of the vanishing point. In both cases, the vanishing point is not at the optical center anymore, as in the benign case of Figure 4b. As our simulation considers the camera-specific FoV, an attacker can evaluate the necessary direction of the PSA to perform a successful attack, such as removing neighboring lines.

5 Experiments

We evaluate our DTF-based PSA against LiDAR and camera sensors, both in static and real-world tests.

5.1 Overview

In our initial overview, we will define our setup, the attack success metrics, and the targeted algorithms.

5.1.1 Hardware Setup

For the real-world deployment of our PSA, we use a DTF that introduces an offset of 20° [36]. We apply the DTF to create both left and right shifts as they are especially interesting in the use case of autonomous driving. We test different LiDAR and camera sensors to show the system-agnostic effects of the attack. The exact sensing hardware used is specified within each evaluation use case. Lastly, we evaluate both LiDAR and camera individually, as well as in a common setup within an autonomous driving research vehicle [26].

5.1.2 Attack Success & Targeted Algorithms

For the LiDAR-based system, we measure the attack impact using the *relative displacement of the odometry estimation*, expressed in meters, of the estimated pose along the axis orthogonal to the vehicle's driving direction (i.e., the lateral axis). LiDAR odometry is essential for AVs, providing accurate localization by estimating the vehicle's position through scan alignment, and serving as a foundation for trajectory estimation [32] and SLAM [20]. To evaluate the effect of our attack, we focus on the KISS-ICP algorithm [52], a state-of-the-art ICP-based method ideally suited for laser scan-based odometry. The relative displacement is directly motivated by the effect of PSAs on the LiDAR sensor. Each point of a LiDAR's point cloud p_i can be expressed in spherical coordinates $(r_i, \theta_i, \varphi_k)$, where r_i is the distance of the point in space, θ_i is the azimuth angle (horizontal angle), and φ_k is the elevation, set by the disposition of the k-th laser array (vertical angle). When applied to the sensor's optics, PSAs introduces a horizontal shift Δ_{PSA} in the sensor's horizontal FoV, which translates in Cartesian coordinates as a rotation of the forward-lateral plane around the vertical axis. Empirical evaluation showed that the coordinate most affected by the attack is the lateral axis, and for this reason, we deem the attacks successful for the LiDAR if the average displacement is greater than 1m, which corresponds to scenarios where a dangerous maneuver might happen, such as entering the opposite lane.

For the camera, we demonstrate the PSA effect on lane detection by investigating the *vanishing-point displacement*. Lane detection is an important functionality for AVs and advanced driver assistance systems. Because they can capture color images, cameras are typically used for lane detection tasks [21,59]. We will use HybridNets [53] as a representative lane detection algorithm. For straight lanes and a centrally mounted front-facing camera, the vanishing point is located at the optical center of the image; however, it shifts with a PSA. We quantify this effect by calculating the horizontal



(a) Left shift (b) Benign (c) Right shift

Figure 4: Detected, color-separated lanes for the digitally simulated PSA. The central image shows the benign FoV. While the attacked images still contain all lanes, their coordinates differ significantly.

offset of the vanishing point from the optical center in pixels. As the pinhole camera model is accurate enough for most cameras [48], we can also derive the bending angle of our DTF-based PSA, as demonstrated in our static physical tests.

In parallel, we utilize object detection algorithms to showcase that PSAs have only minor adverse impact on perceived object appearance. For the cameras, we select the generic Faster R-CNN [45] object detector as well as a specifically trained version of YOLOv10 [28] to detect objects. For the object detection with the LiDAR, we use PointPillars [30], and SECOND [57] pretrained on the KITTI dataset [14]. To demonstrate that detection of smaller objects remains possible, we use YOLOv10, trained on the Mapillary traffic sign dataset [12, 28], for camera-based traffic sign detection. Since the attacked data has an offset in the FoV, they will *not* contain the same objects or the same number of objects as the benign data. Typical metrics, such as mean average precision, will therefore not provide valid results. As a solution, we use the *average confidence of detected objects* of the same type in all captured scenes for both scenarios to investigate the impact on the perception.

5.2 Static Physical Tests

To assess the effectiveness of PSAs against OPSs in a real environment, we conducted static physical tests with a DTF. The goal of the tests is to observe and quantify the optical shift caused by the DTF when applied to the sensor's optics. To do so, for both LiDAR and camera, we use a reference object posed precisely in front of the sensor, aligned with the optical center axis. The first set of measurements is carried out without any perspective shift, whereas the second set of measurements is carried out with DTF-induced perspective shift. The experiment is evaluated based on the shift perceived by the OPS of the reference objects. This test enables us to confirm our assumptions about the effectiveness of the attack in a controlled scenario, with minimal external noise sources.

5.2.1 LiDAR

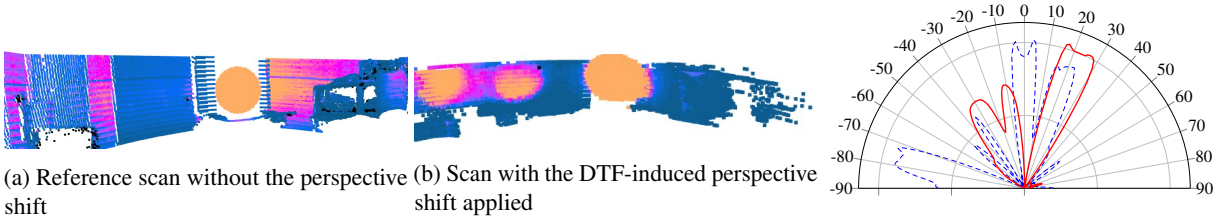
For the LiDAR, we use an object that is highly distinguishable within the surrounding environment. As traffic signs are retroreflective, resulting in a high intensity value perceived by the LiDAR, we use them to evaluate the FoV shift by measuring the change in azimuth of the highest reflectivity peak [39]. This, combined with a plain wooden background, is the identification of the traffic sign in the perceived data.

Figure 5c shows the result of the test using an Ouster OS1 64 [22], where the reference scene is depicted in blue, and the adversarial case in red. Within the reference polar plot, it is evident that the highest reflectivity values are centered at 0°, indicating the location where the traffic sign is placed. If we now observe the same peak in the attack line, we can see that it is shifted by approximately half of the first circumference sector, confirming a shift of 20°, which is coherent with the description of the DTF we employed for the test. It is worth noting that some of the optical aberrations in the adversarial point cloud are due to the rounded surfaces of the sensor's optics. As we will investigate in Subsection 5.3.1, they have a negligible effect on the final perception algorithms. Moreover, since a left shift of the FoV yields qualitatively the same effect, only mirrored in the opposite direction, Figure 5 shows only the right-shifted data for clarity.

5.2.2 Camera

We project a reference image with vertical lines at 10° intervals in a static indoor setup, capturing images with a Sony ILCE-6400 [46]. Figure 6 depicts these images, with the red line at the center marking the 0° position.

With the knowledge of camera parameters, namely the physical image sensor width w in mm, the focal length f in mm, and the image width x_I in pixels, we can calculate the horizontal angle θ of a specific pixel $p_{x,y}$ as an offset from the center. Since the pinhole camera model is sufficiently accurate for our application [48], and w, f, and x_I are static parameters for a captured image, θ only depends on the horizontal pixel

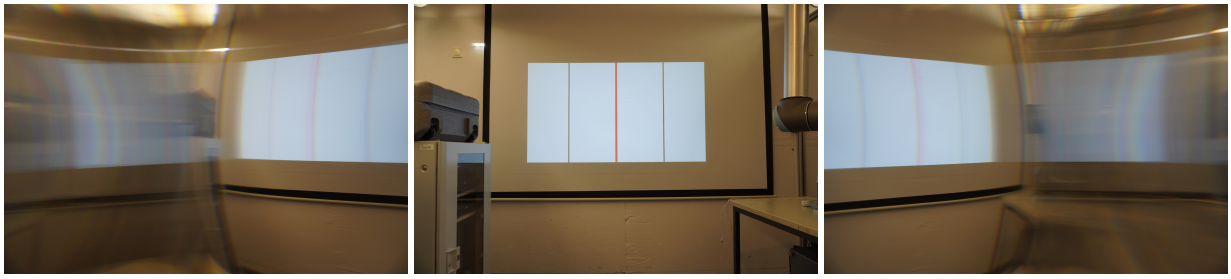


(a) Reference scan without the perspective shift

(b) Scan with the DTF-induced perspective shift applied

(c) Average *normalized* reflectivity per azimuth angle. Benign: ---; Attack: —

Figure 5: Figure 5c shows the reflectivity peak under attack (Figure 5b, red) at $\approx 20°$, compared to the benign case (Figure 5a, dotted blue). This confirms that real LiDAR systems are affected by the PSA.



(a) DTF-induced left shift

(b) Benign

(c) DTF-induced right shift

Figure 6: Reference images with vertical lines at each 10°. The central image shows the benign FoV, while the outer images show the FoV with the PSA. The central red reference line has a relative offset of $0.02°$ in the benign image, and $-19.42°$ and $+19.40°$ in the respective adversarial images.

position p_x, as shown in Equation 7:

$$\theta(p_x) = \frac{\arctan\left(\frac{w}{2f}\right)(2p_x - x_I)}{x_I} \tag{7}$$

Applying this to the red reference line in the images of Figure 6 results in $\theta(p_{x,\text{benign}}) = 0.02°$ for the benign case of Figure 6b. The left shift in Figure 6a results in $\theta(p_{x,\text{left}}) = 19.40°$, and the right shift of Figure 6c results in $\theta(p_{x,\text{right}}) = -19.42°$. These measurements confirm the practical applicability of the DTF for PSAs in a static scenario. We will investigate the effect of possible artifacts from the DTF with object detection models in our following real-world tests.

5.3 Real-World Tests

We categorize our real-world tests into three categories to evaluate PSAs against both LiDAR and a camera, individually and in a multimodal setup. Data are collected using an autonomous driving research vehicle [26] operating under natural urban traffic conditions. Experiments are conducted at variable vehicle speeds ranging from 10 to 50km/h, consistent with local urban traffic regulations. Throughout all trials, the DTF remains securely attached to the sensor, demonstrating the physical plausibility and robustness of the proposed attack under realistic deployment conditions.

5.3.1 LiDAR

We evaluate the real-world impact of the proposed PSA by deploying the DTF directly on the optical aperture of an Innovusion Falcon Gen2 LiDAR [49] sensor, as shown in Figure 7a, which features a horizontal FoV of 120°. To quantify the effect of the induced FoV displacement, we compare the trajectory estimated via KISS-ICP against the GT trajectory obtained from high-precision GNSS pose measurements. The DTF is mounted directly in front of the LiDAR optics and secured using tape, ensuring both mechanical stability and visual stealthiness. This mounting approach enables a robust attachment under both low- and high-speed driving conditions, without compromising the sensor's operational integrity.

As illustrated in Figure 8, the trajectory estimated by KISS-ICP exhibits a clear and systematic deviation from the GNSS ground truth, confirming the effectiveness of the induced perturbation. Moreover, the observed distortion in the recorded point clouds is consistent with both our prior controlled experiments and the initial digital simulations, manifesting as a measurable displacement in the estimated pose coordinates attributable to the applied DTF. These results provide strong evidence that the proposed PSAs are effective in transferring from simulation to real-world driving scenarios, underscoring their practical feasibility and potential real-world impact.

For the LiDAR-based object detection, we use Pointpil-



(a) DTF deployed on a real vehicle LiDAR



(b) DTF deployed on a front-facing camera

Figure 7: Examples of the DTF deployed on LiDAR and and camera of real vehicles

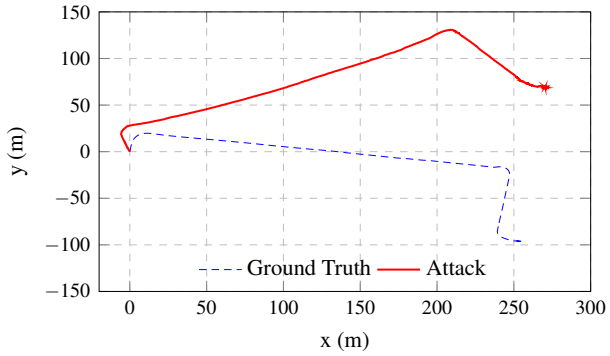


Figure 8: Plot of the estimated odometry using the LiDAR with the DTF deployed against the GNSS system present on the autonomous vehicle. The plot highlights that a 20° offset over an FoV of 120° highly impacts the estimated poses, deeming our attack on the LiDAR a success, with a displacement of the poses in the orthogonal driving direction (y axis in the innovusion coordinate system) of 100m.

lars [30], and SECOND [57] on both benign and attacked point cloud sequences. We set the detection confidence threshold to 0.6 for both models and object classes supported by KITTI-trained models. Table 2 shows the impact of the PSA on the average detection confidence of these classes. Except for bikes, there is no systematic change in detection confidence visible for either of the evaluated object detection models, as they typically drive on the side of the road, which is not covered by the shifted FoV.

5.3.2 Camera

For the camera, we place a small piece of the DTF in front of a Sony IMX500 with a 12MP sensor [47], as depicted in Figure 7b, and capture videos at 30 frames per second (FPS). The DTF is placed to cause a right shift on the captured images. Additionally, we capture reference videos with a mobile phone camera without a perspective shift. With this

Table 2: Impact of the DTF-induced PSA on the average detection confidence for the LiDAR, using Pointpillars, and SECOND. Further limitations are discussed in Section 7.

Object	Pointpillars		SECOND	
	Benign	PSA	Benign	PSA
Car	68.39%	68.90%	70.60%	72.34%
Pedestrian	68.40%	60.50%	67.12%	63.93%
Bike	69.39%	0%	68.20%	63.92%

setup, we capture approximately 18 minutes of recordings that we use for our evaluation.

Figure 9 shows an overview of representative example images of this test drive. While the lanes and the drivable area are correctly segmented in their respective FoV, HybridNets [53] provides wrong coordinates in the attacked cases of Figure 9b. Especially, the direct neighboring lanes of the ego-vehicle in Figure 9a lead to the vanishing point in the optical center of the image. By contrast, the neighboring lanes of the right-shifted images from Figure 9b do not lead to the optical center of the image, but a vanishing point outside of the image on the left border. This observation is similar to the digitally simulated images of Figure 4. We emphasize that other models will lead to a similar result, since the PSAs do not attack the segmentation model but the FoV of the camera, as shown in Equations 5 and 6. Especially the incorrect segmentation of lanes and free space can lead to severe consequences in a fully autonomous vehicle, since cameras are typically the sole source of information for these tasks [21, 59].

To investigate the effect on object detection, we deploy Faster R-CNN [45], and an automotive-specific version of YOLOv10 [28] on both benign and attacked images. Similar to lane detection, these ML models are selected purely for being representatives of different model architectures, but the effect of our PSA is independent of the deployed model. While the detection threshold for Faster R-CNN is set ot 0.8, it is 0.5 for YOLOv10. We investigate the following automotive-relevant object types for our evaluation: "Car," "Bicycle," "Person," "Motorcycle," "Bus," "Truck," and "Traffic Light." Table 3 shows the impact of the DTF-induced PSA on the average detection confidence of these classes. There is no systematic change in the detection confidence visible for either of the evaluated object detection models. While individual confidences, such as those for motorcycles, differ, they are the result of a low sample size in our recorded videos. Other object types that appeared more often in the test drive (e.g., cars) show no significant difference in confidence between the benign case and the attacked cases, but are incorrectly detected in position.

In addition to object detection, we evaluate the impact of PSAs on traffic sign detection, a task typically performed by cameras [21]. As previously described, we use YOLOv10,



(a) Representative benign images of the real-world tests, including detected lanes



(b) Representative right-shifted images of the real-world tests, including detected lanes

Figure 9: Overview of representative example images from our real-world tests. The images were captured in a real vehicle, and the lanes are detected through HybridNets [53]. While the lanes (green) and drivable area (orange) in the respective FoV are detected, their coordinates and direction differ. This can lead to misbehavior for AVs, especially for lane-keeping tasks.

Table 3: Average detection confidence for the camera, using Faster R-CNN, YOLOv10. While Faster R-CNN shows no significant difference, except for motorcycles, YOLOv10 shows a confidence decrease at objects that appeared less often. Further limitations are discussed in Section 7.

Object	Faster R-CNN		YOLOv10	
	Benign	PSA	Benign	PSA
Car	95.33%	95.10%	79.69%	78.30%
Bicycle	92.24%	94.50%	69.21%	83.49%
Person	93.66%	91.47%	78.34%	72.57%
Motorcycle	96.26%	85.81%	75.33%	69.78%
Bus	88.59%	91.81%	75.25%	68.74%
Truck	92.19%	88.40%	74.54%	74.41%
Traffic Light	94.69%	88.69%	69.63%	65.41%

specifically trained on the Mapillary dataset [12, 28]. As this model detects a wide range of traffic signs, we will focus on the most frequently detected traffic signs from the benign reference camera. While the adverse FoV change of our PSA may not be as relevant for this task, it is still important to avoid any negative impact on detection confidence, similar to object detection. As the results in Table 4 show, the detection confidence shows a slight reduction in the adverse case, but they are still detected with sufficient confidence. Especially traffic signs with unique shapes (e.g., "Yield") preserve their detection confidence, while less distinctive traffic signs (e.g., a 30km/h speed limit) show lower confidence.

Table 4: Impact of the DTF-induced PSA on the average camera-based traffic sign detection confidence for YOLOv10.

Sign	YOLOv10	
	Benign	PSA
Yield	83.94%	83.67%
Keep Right	83.94%	–*
Priority Road	80.78%	73.29%
Pedestrians Crossing	86.27%	69.82%
Roundabout	89.11%	77.21%
30km/h speed limit	87.34%	72.28%
No Entry	80.11%	88.28%

* Traffic Signs outside FoV

5.3.3 Multimodal

Besides the individual evaluation of the effects of PSAs on the LiDAR and the camera, we conduct real-world tests using both the previously mentioned front-facing Falcon Gen2 LiDAR [49] and the car's front-facing stereo camera, a Carnegie Robotics MultiSense S30 camera [7], simultaneously, mounted on top of a autonomous driving research vehicle, as shown in Figure 10

The deployment of our PSA on the LiDAR is the same as explained in Section 5.3.1. For the camera, we deploy the DTF in front of one of the image sensors of the stereo camera. This allows us to capture the benign reference image independently. For both sensors, we deploy our PSA to create

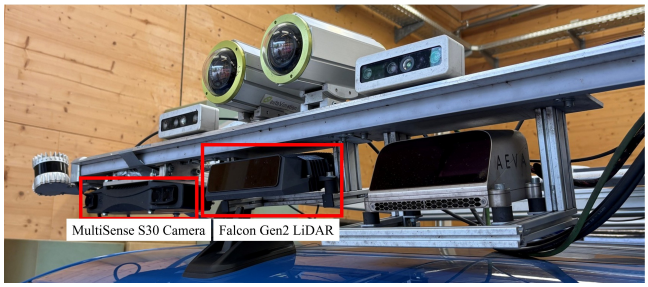
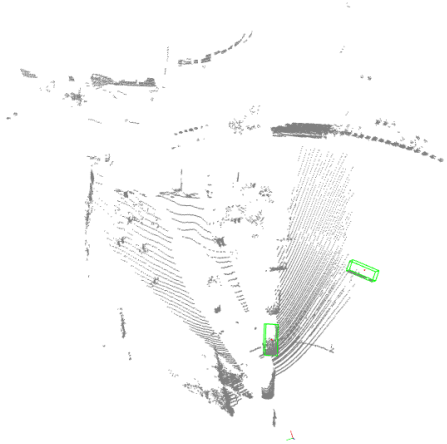


Figure 10: The sensor setup used for our multi-modal evaluation consists of a Falcon Gen2 LiDAR and a MultiSense S30 camera mounted on an autonomous driving research vehicle.

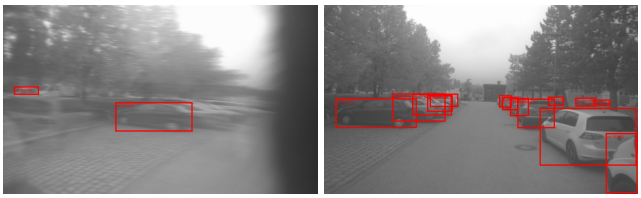
a left shift in the FoV. The goal of this analysis is the coherent deployment of PSAs in a multimodal sensor setup.

We evaluated the coherence of the attack in the multimodal setting by selecting a representative scene with many objects to be detected. This allows for visual inspection to observe the shift and its consequences on object detection.

As it is possible to see from Figure 11, both the LiDAR, in Figure 11a, and the camera, in Figure 11b, are under the effect of the left shift caused by the coherent PSA deployment. When comparing the results with our GT, the AV fails to detect objects that are present on the right side of the road. This confirms the success of our attack in the multimodal case, with detected objects from both sensors **exhibiting a coherent offset in their locations** and emphasizes the effectiveness of DTF-based PSAs against multi-sensor setups.



(a) LiDAR object detection with deployed DTF using Second



(b) Camera object detection with Faster-RCNN for both benign image (right) and deployed DTF (left)

Figure 11: Visual comparison of object detection for both camera and LiDAR. If PSAs are deployed coherently, the resulting detections are also coherent.

6 Attack Detection

The result of the multimodal analysis showed that the detection of PSAs must rely on OPSs themselves, and their captured data. Geometric consistency checks are ineffective because the transformation can be applied from the moment the AV starts, leaving no prior baseline for comparison. Instead, defenses must focus on physical artifacts introduced by the attack that subtly affect sensor behavior.

6.1 LiDAR

Under the threat model considered in this work, the adversary can shift the LiDAR FoV using a DTF. To classify whether a frame is under PSA manipulation, we propose a signal-level defense that exploits physical-layer artifacts observable in sensor data. An analysis of our LiDAR sensor reveals that when a DTF is applied to the LiDAR optics, the reflectivity signal undergoes a pronounced smoothing effect. Specifically, spatial regions that typically exhibit structured reflectivity distributions become significantly flattened. This observation provides a distinctive physical fingerprint of the attack: a salient high-frequency component. We further confirm this behavior in the frequency domain. The Fast Fourier Transform (FFT) of the reflectivity signal shows a substantial suppression of high-frequency components in the presence of the DTF, consistent with the behavior of an optical low-pass filter. Based on this observation, we can set a cutoff frequency, in our case 0.1 (normalized spatial frequency), for the specific LiDAR model and monitor the magnitudes of spectral components exceeding this threshold. This cutoff is selected empirically; in our experiments, spectral magnitudes below this frequency remain largely comparable across benign and attacked signals, offering limited discriminatory power. However, for a different LiDAR, it might be necessary to evaluate which cutoff frequency works best, and analyze the false-positive ratio to avoid misdetection.

To quantitatively evaluate the proposed defense, we compute the *High-Frequency Energy Ratio* (HFR), defined as

$$\mathrm{HFR} = \frac{|\mathcal{R}_{\mathrm{cutoff}}|^2}{|\mathcal{R}_{\mathrm{total}}|^2}, \tag{8}$$

where $\mathcal{R}_{\mathrm{cutoff}}$ denotes the FFT coefficients whose spatial frequencies exceed the chosen cutoff, and $\mathcal{R}_{\mathrm{total}}$ denotes the full reflectivity spectrum coefficients.

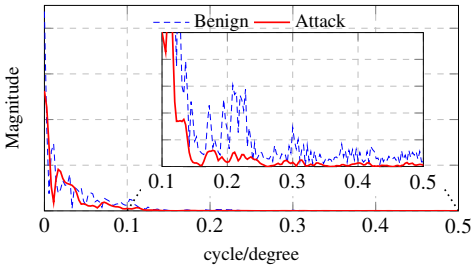


Figure 12: Fourier transform of the reflectivity sequences from two static laboratory scenes. The red plot corresponds to the case with the DTF applied, while the blue plot represents the baseline without the DTF. The low-frequency components are comparable across both cases and do not provide a strong discriminative feature for detecting the attack. In contrast, the high-frequency components exhibit a noticeable difference, indicating that high-frequency energy can serve as a strong feature for attack detection.

For benign scenes, we empirically estimate a threshold τ_{HFR}. At runtime, we apply the following decision rule: if the measured HFR falls below τ_{HFR}, the system flags the measurement as being under attack, indicating the presence of an adversarial optical filter.

6.2 Camera

Typically, automotive cameras capture parts of the ego vehicle within their FoV. Examples include the engine hood for front-facing cameras, as well as parts of the vehicle body and wheels for surround cameras. As these parts of images typically include unwanted reflections or distortions, they are usually removed [33]. We applied similar steps in our digital simulation: While the full-scale images in Figures 2d to 2f show large portions of the engine hood, the inference results in Figure 4 were cropped to avoid misdetections by unwanted reflections.

By contrast, such static ego-vehicle parts can serve as a basis for PSA detection. As the engine hood or other parts of the vehicle body remain stationary with respect to the camera position, their positions within captured images must always be similar. Semantic segmentation can be used to identify static portions of the ego vehicle. As PSAs affect the entire image, these stationary vehicle body parts will have a differently perceived position. Our proposed PSA detection for cameras, therefore, requires the full image from the sensor and must be applied before potential cropping.

To demonstrate the working principle of our proposed detection, we perform semantic segmentation on both the benign and attacked images in Figure 9. We use OneFormer as a state-of-the-art semantic segmentation model [23]. As shown in Figure 13, the shape of the detected ego vehicle engine hood (dark blue) differs significantly. For the benign images in Figure 13a, it shows the expected central part of the engine hood with slight curves, while the adversarially shifted images in Figure 13b have a clearly distinguishable edge, resulting from the now visible front right edge of the ego vehicle. To quantify this, we perform a quadratic fitting on the pixel values of the detected edge between the engine hood and other pixels, highlighted as a red line in Figure 13. There are two observations from the quadratic fit:

1. The adversarial images show the edge of the ego vehicle, **resulting in a lower quadratic coefficient**, as the quadratic fitting results in a downward-opening parabola.
2. Due to the changed FoV, the initial slope on the left side of the image is higher for adversarial images, compared to the benign ones, **resulting in a higher linear coefficient**.

These two parameters are static for benign images captured with a front-facing camera mounted statically and depend only on the exact vehicle model; for example, vehicles with a larger engine hood have different parameters. During vehicle development, manufacturers can integrate reference parameters into the camera for later use in identifying adversarial changes in the FoV. While our example is based on a front-facing camera and the engine hood, similar static vehicle parts can also be used for other camera placement options.

7 Limitations

We specifically investigated the adversarial effect on lane detection and odometry estimation, as well as a possible decrease in confidence for object and traffic sign detection. While humans can perceive artifacts in the captured data from both LiDAR and camera, our results in Tables 2 and 3 show that ML models show no systematic reduction in their confidence. While these artifacts are not a systematic drawback of PSAs, as our simulation shows, they are an effect of our specific DTF-based PSA, resulting from an optical film on OPSs.

Another limiting factor of DTF-based PSAs is the physical access that an attacker requires to place the film on the OPS. While the one-time physical access to the target vehicle is a limitation, it allows control of the target, not affecting unintended assets. Existing attacks require similar physical access in order to be deployed [17, 31, 50, 64].

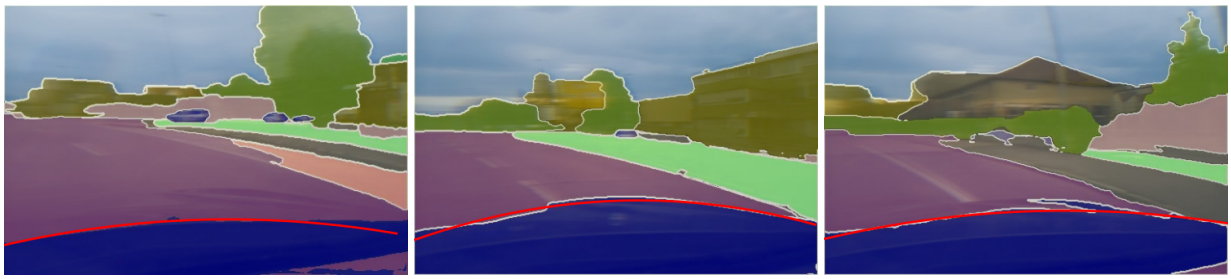
For our evaluation, we considered a shift of 20° as per the commercial availability of the material. However, PSAs might be deployed with a different shift, and our digital simulation allows to examine different shifts and their effects on various algorithms.

8 Ethical Considerations

For the evaluation of our PSA, we use an autonomous driving research vehicle [26], operated in manual mode by a trained



(a) Semantic Segmentation and quadratic approximation results overlayed on benign reference images



(b) Semantic Segmentation and quadratic approximation results overlayed on right-shifted images

Figure 13: Results of OneFormer semantic segmentation on the images of Figure 9. The red line shows a quadratic approximation of the engine hood. While the engine hood of the ego-vehicle, highlighted in dark blue, shows a slightly curved engine hood without strong edges in the benign case, it clearly shows the right edge of the ego-vehicle for the attacked images.

driver who maintained continuous control. This precaution ensured that human supervision was always present, thereby minimizing any potential risks to public safety. All data collection activities were conducted in low-traffic areas, in strict compliance with local traffic regulations, including applicable speed limits. The experiments were designed to avoid any hazardous driving behavior and did not interfere with other road users or public infrastructure. As our DTF-based PSA is manufacturer-agnostic, we did not perform responsible disclosure for specific sensor manufacturers.

9 Conclusion

In our work, we introduced PSAs as a novel class of attacks against LiDAR sensors and cameras used in AVs. Unlike prior physical adversarial attacks that aim to decrease the performance of perception algorithms, our approach operates by physically altering the sensing region itself via a DTF-induced perspective shift, thereby shifting the effective FoV with only minor adverse impact on the sensor operation. By providing a digital simulation of the effects for both LiDAR and camera, we enable the evaluation of an altered FoV on different algorithms.

For our evaluation, we specifically investigated the adversarial effect on lane detection and odometry estimation. Additionally, we show that PSAs result in no systematic change in confidence for object and traffic sign detection, however, their perceived location changes. Our evaluation is threefold: (i) *Digital Evaluation:* We evaluate the effect of a changed FoV using the KITTI odometry dataset for the LiDAR and spherical images for the camera; (ii) *Static Physical Tests:* By performing physical tests in a controlled, static environment, we investigate the effect of the DTF to confirm its feasibility for a PSA, and (iii) *Real-World Tests:* We conduct extensive experiments with real vehicles to analyze the effect of the DTF-induced PSA in dynamic real-world scenarios, both for LiDAR and camera individually, as well as in a coherently applied multi-modal setup. The experimental results corroborate our central hypothesis: inducing systematic FoV shifts is sufficient to significantly degrade critical AV perception functionalities, even without injecting adversarial signals or modifying the environment.

By providing individual detection mechanisms for the LiDAR and camera, we aim to enhance the security of AVs. Our detection mechanisms work by solely receiving the sensor data and comparing it with a statistically derived threshold without the need for additional sensing hardware. While the detection mechanism for LiDAR is based on an intensity analysis of the received point clouds, our proposed mechanism relies on static reference points that are typically captured in vehicles, such as the engine hood.

Together, these results establish PSAs as previously unexplored, practically viable attacks for AVs, with significant implications for sensor security and perception robustness.

Acknowledgments

This work is supported by the European Union-funded projects CyberSecDome (Agreement No.: 101120779) and PANDORA (Agreement No.: 101135775).

References

[1] Bakary Badjie, José Cecílio, and Antonio Casimiro. Adversarial Attacks and Countermeasures on Image Classification-based Deep Learning Models in Autonomous Driving Systems: A Systematic Review. *ACM Computing Surveys*, 57(1):1–52, January 2025.

[2] Rodrigo F. Berriel, Edilson De Aguiar, Alberto F. De Souza, and Thiago Oliveira-Santos. Ego-Lane Analysis System (ELAS): Dataset and algorithms. *Image and Vision Computing*, 68:64–75, December 2017.

[3] Sri Hrushikesh Bhupathiraju, Takeshi Sugawara, Takami Sato, Qi Alfred Chen, Michael Clifford, and Sara Rampazzi. On the Vulnerability of Traffic Light Recognition Systems to Laser Illumination Attacks. In *ISOC Symposium on Vehicle Security and Privacy (VehicleSec)*, San Diego, CA, USA, 2024. ISOC.

[4] Brightview Technologies. Angle Bending Diffusers, 2018.

[5] Yulong Cao, S. Hrushikesh Bhupathiraju, Pirouz Naghavi, Takeshi Sugawara, Z. Morley Mao, and Sara Rampazzi. You can't see me: Physical removal attacks on LiDAR-based autonomous vehicles driving frameworks. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 2993–3010, Anaheim, CA, August 2023. USENIX Association.

[6] Yulong Cao, Chaowei Xiao, Dawei Yang, Jing Fang, Ruigang Yang, Mingyan Liu, and Bo Li. Adversarial objects against lidar-based autonomous driving systems. *arXiv preprint arXiv:1907.05418*, 2019.

[7] Carnegie Robotics LLC. Multisense S30 - Product Summary, 2024.

[8] Baodong Chen, Wei Wang, Pascal Sikorski, and Ting Zhu. Adversary is on the Road: Attacks on Visual SLAM using Unnoticeable Adversarial Patch. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 6345–6362, Philadelphia, PA, August 2024. USENIX Association.

[9] Driver Vision Standards Committee. J1050_200902: Describing and measuring the driver's field of view, 2009. SAE Standard.

[10] Ranjie Duan, Xiaofeng Mao, A. K. Qin, Yuefeng Chen, Shaokai Ye, Yuan He, and Yun Yang. Adversarial Laser Beam: Effective Physical-World Attack to DNNs in a Blink. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16057–16066, Nashville, TN, USA, June 2021. IEEE.

[11] Zeinab El-Rewini, Karthikeyan Sadatsharan, Niroop Sugunaraj, Daisy Flora Selvaraj, Siby Jose Plathottam, and Prakash Ranganathan. Cybersecurity Attacks in Vehicular Sensors. *IEEE Sensors Journal*, 20(22):13752–13767, November 2020.

[12] Christian Ertler, Jerneja Mislej, Tobias Ollmann, Lorenzo Porzi, Gerhard Neuhold, and Yubin Kuang. The Mapillary Traffic Sign Dataset for Detection and Classification on a Global Scale. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, volume 12368, pages 68–84. Springer International Publishing, Cham, 2020. Series Title: Lecture Notes in Computer Science.

[13] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust Physical-World Attacks on Deep Learning Visual Classification. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1625–1634, Salt Lake City, UT, USA, June 2018. IEEE.

[14] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[15] Abhiram Gnanasambandam, Alex M. Sherman, and Stanley H. Chan. Optical Adversarial Attack. In *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 92–101, Montreal, BC, Canada, October 2021. IEEE.

[16] Amira Guesmi, Muhammad Abdullah Hanif, Bassem Ouni, and Muhammad Shafique. Physical Adversarial Attacks for Camera-Based Smart Systems: Current Trends, Categorization, Applications, Research Challenges, and Future Outlook. *IEEE Access*, 11:109617–109668, 2023.

[17] Amira Guesmi, Muhammad Abdullah Hanif, and Muhammad Shafique. AdvRain: Adversarial Raindrops to Attack Camera-Based Smart Vision Systems. *Information*, 14(12):634, November 2023.

[18] Amira Guesmi and Muhammad Shafique. Navigating threats: A survey of physical adversarial attacks on lidar perception systems in autonomous vehicles. *arXiv preprint arXiv:2409.20426*, 2024.

[19] Zhongyuan Hau, Kenneth T Co, Soteris Demetriou, and Emil C Lupu. Object removal attacks on lidar-based 3d object detectors. *arXiv preprint arXiv:2102.03722*, 2021.

[20] Leyao Huang. Review on lidar-based slam techniques. In *2021 International conference on signal processing and machine learning (CONF-SPML)*, pages 163–168. IEEE, 2021.

[21] Javier Ibanez-Guzman and You Li. LiDAR and cameras in autonomous driving. *Nature Reviews Electrical Engineering*, May 2025.

[22] Ouster Inc. Os1: High-res mid-range lidar sensor for automation & security. `https://ouster.com/products/hardware/os1-lidar-sensor`.

[23] Jitesh Jain, Jiachen Li, MangTik Chiu, Ali Hassani, Nikita Orlov, and Humphrey Shi. OneFormer: One Transformer to Rule Universal Image Segmentation. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2989–2998, Vancouver, BC, Canada, June 2023. IEEE.

[24] Zizhi Jin, Xiaoyu Ji, Yushi Cheng, Bo Yang, Chen Yan, and Wenyuan Xu. Pla-lidar: Physical laser attacks against lidar-based 3d object detection in autonomous vehicle. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1822–1839. IEEE, 2023.

[25] William B. Joyce and Alice Joyce. Descartes, newton, and snell's law. *J. Opt. Soc. Am.*, 66(1):1–8, Jan 1976.

[26] Phillip Karle, Tobias Betz, Marcin Bosk, Felix Fent, Nils Gehrke, Maximilian Geisslinger, Luis Gressenbuch, Philipp Hafemann, Sebastian Huber, Maximilian Hübner, Sebastian Huch, Gemb Kaljavesi, Tobias Kerbl, Dominik Kulmer, Tobias Mascetta, Sebastian Maierhofer, Florian Pfab, Filip Rezabek, Esteban Rivera, Simon Sagmeister, Leander Seidlitz, Florian Sauerbeck, Ilir Tahiraj, Rainer Trauth, Nico Uhlemann, Gerald Würsching, Baha Zarrouki, Matthias Althoff, Johannes Betz, Klaus Bengler, Georg Carle, Frank Diermeyer, Jörg Ott, and Markus Lienkamp. EDGAR: An Autonomous Driving Research Platform – From Feature Development to Real-World Application, January 2024. arXiv:2309.15492 [cs].

[27] Ryunosuke Kobayashi, Kazuki Nomoto, Yuna Tanaka, Go Tsuruoka, and Tatsuya Mori. Invisible but detected: Physical adversarial shadow attack and defense on lidar object detection. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 7369–7386, 2025.

[28] Michael Kühr, Maximilian Mittmann, Mohammad Hamad, and Sebastian Steinhorst. CATI - An Open-Source Framework to Evaluate Attacks on Cameras for Autonomous Vehicles. In *28th Euromicro Conference Series on Digital System Design (DSD) 2025*, Salerno, Italy, 2025.

[29] Michael Kühr, Mohammad Hamad, Pedram Mohajer-Ansari, Mert D. Pesé, and Sebastian Steinhorst. Sok: Security of the image processing pipeline for camera-based sensing in autonomous vehicles, 2026.

[30] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12697–12705, 2019.

[31] Juncheng Li, Frank R. Schmidt, and J. Zico Kolter. Adversarial camera stickers: A physical camera-based attack on deep learning systems. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, volume 97 of *Proceedings of Machine Learning Research*, pages 3896–3904, Long Beach, California, USA, June 2019. PMLR.

[32] Shuaixin Li, Li Wang, Jiuren Li, Bin Tian, Long Chen, and Guangyun Li. 3d lidar/imu calibration based on continuous-time trajectory estimation in structured environments. *IEEE Access*, 9:138803–138816, 2021.

[33] Flora Lian, Dinh Quang Huynh, Hector Penades, J. Stephany Berrio Perez, Mao Shan, and Stewart Worrall. Data Augmentation Strategies for Robust Lane Marking Detection, November 2025. arXiv:2511.18668 [cs].

[34] Taifeng Liu, Yang Liu, Zhuo Ma, Tong Yang, Xinjing Liu, Teng Li, and Jianfeng Ma. L-HAWK: A Controllable Physical Adversarial Patch Against a Long-Distance Target. In *Proceedings 2025 Network and Distributed System Security Symposium*, San Diego, CA, USA, 2025. Internet Society.

[35] Giulio Lovisotto, Henry Turner, Ivo Sluganovic, Martin Strohmeier, and Ivan Martinovic. SLAP: Improving Physical Adversarial Examples with Short-Lived Adversarial Perturbations. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1865–1882. USENIX Association, August 2021.

[36] Luminit Inc. Direction Turning Film | Luminit Prismatic Light Control, October 2025.

[37] Luminit Inc. Display Brightness Film | Luminit Prismatic Optics, 2025.

[38] Luminit Inc. Light Shaping Diffusers® | Uniform Light Control & Optics Solutions, 2025.

[39] Paul Mcmanamon. *LiDAR Technologies and Systems*. July 2019.

[40] Mercedes-Benz Group AG. Mercedes-Benz world's first automotive company to certify SAE Level 3 system for U.S. market | Mercedes-Benz Group > Innovations > Product innovation > Autonomous driving, January 2023.

[41] Nicoleta Minoiu Enache, Mariana Netto, Saïd Mammar, and Benoit Lusetti. Driver steering assistance for lane departure avoidance. *Control Engineering Practice*, 17(6):642–651, June 2009.

[42] Rokuto Nagata, Kenji Koide, Yuki Hayakawa, Ryo Suzuki, Kazuma Ikeda, Ozora Sako, Qi Alfred Chen, Takami Sato, and Kentaro Yoshioka. Slamspoof: Practical lidar spoofing attacks on localization systems guided by scan matching vulnerability analysis, 2025.

[43] Simon Parkinson, Paul Ward, Kyle Wilson, and Jonathan Miller. Cyber Threats Facing Autonomous and Connected Vehicles: Future Challenges. *IEEE Transactions on Intelligent Transportation Systems*, 18(11):2898–2915, November 2017.

[44] Jonathan Petit, Bas Stottelaar, Michael Feiri, and Frank Kargl. Remote attacks on automated vehicles sensors: Experiments on camera and lidar. *Black Hat Europe*, 11(2015):995, 2015.

[45] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[46] Sony Electronics Inc. ILCE-6400 Specifications | Sony USA, January 2025.

[47] Sony Group Corporation. IMX500 Key Specifications | Developer World, March 2025.

[48] Richard Szeliski. Image Formation. In *Computer Vision*, pages 27–83. Springer International Publishing, Cham, 2022. Series Title: Texts in Computer Science.

[49] Wendy Tam. Innovusion falcon lidar. Blog post on TechInsights, Nov 18 2025. Accessed 2025-12-05.

[50] Kalibinuer Tiliwalidi, Chengyin Hu, Weiwen Shi, Guangxi Lu, and Hao Wu. Adversarial translucent patch: a robust physical attack technique against object detectors. *Pattern Analysis and Applications*, 28(4):157, December 2025.

[51] James Tu, Mengye Ren, Sivabalan Manivasagam, Ming Liang, Bin Yang, Richard Du, Frank Cheng, and Raquel Urtasun. Physically realizable adversarial examples for LiDAR object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13716–13725, 2020.

[52] Ignacio Vizzo, Tiziano Guadagnino, Benedikt Mersch, Louis Wiesmann, Jens Behley, and Cyrill Stachniss. Kiss-icp: In defense of point-to-point icp–simple, accurate, and robust registration if done the right way. *IEEE Robotics and Automation Letters*, 8(2):1029–1036, 2023.

[53] Dat Vu, Bao Ngo, and Hung Phan. HybridNets: End-to-End Perception Network, March 2022. arXiv:2203.09035 [cs].

[54] Waymo LLC. Self-Driving Car Technology for a Reliable Ride - Waymo Driver, July 2024.

[55] Junqi Wu, Wen Yao, Zijian Zhu, Donghua Wang, Tingsong Jiang, and Chao Ma. Invisibility stickers against lidar: Adversarial attacks on point cloud intensity for lidar-based detection.

[56] Yuan Xu, Xingshuo Han, Gelei Deng, Jiwei Li, Yang Liu, and Tianwei Zhang. Sok: Rethinking sensor spoofing attacks against robotic vehicles from a systematic view. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 1082–1100. IEEE, 2023.

[57] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10), 2018.

[58] Kaichen Yang, Tzungyu Tsai, Honggang Yu, Max Panoff, Tsung-Yi Ho, and Yier Jin. Robust roadside physical adversarial attack against deep learning in lidar perception modules. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 349–362, 2021.

[59] Noor Jannah Zakaria, Mohd Ibrahim Shapiai, Rasli Abd Ghani, Mohd Najib Mohd Yassin, Mohd Zamri Ibrahim, and Nurbaiti Wahid. Lane Detection in Autonomous Vehicles: A Systematic Review. *IEEE Access*, 11:3729–3765, 2023.

[60] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and systems*, volume 2, pages 1–9. Berkeley, CA, 2014.

[61] Ce Zhou, Qiben Yan, Daniel Kent, Guangjing Wang, Ziqi Zhang, and Hayder Radha. Optical Lens Attack on Deep Learning Based Monocular Depth Estimation. In *Security and Privacy in Communication Networks*, volume 627, pages 214–239. Springer Nature Switzerland, Cham, 2026.

[62] Wenjun Zhu, Xiaoyu Ji, Yushi Cheng, Shibo Zhang, and Wenyuan Xu. TPatch: A Triggered Physical Adversarial Patch. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 661–678, Anaheim, CA, August 2023. USENIX Association.

[63] Yi Zhu, Chenglin Miao, Tianhang Zheng, Foad Hajiaghajani, Lu Su, and Chunming Qiao. Can we use arbitrary objects to attack lidar perception in autonomous driving? In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1945–1960, 2021.

[64] Alon Zolfi, Moshe Kravchik, Yuval Elovici, and Asaf Shabtai. The translucent patch: A physical and universal attack on object detectors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15232–15241, June 2021.

Seeing is Deceiving: Systematic Vulnerability Analysis of LiDAR-Based Autonomous Driving to Mirror-Induced Perception Failures

Selma Yahia* Ildi Alla* Girija B. Mohan† Daniel Rau† Mridula Singh‡ Valeria Loscri§

*Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg
†University of Applied Sciences Saarland (htw saar), 66117 Saarbrücken, Germany
‡CISPA Helmholtz Center for Information Security
§Inria Lille – Nord Europe

Abstract

Autonomous vehicles rely heavily on LiDAR for accurate 3D perception; however, this sensing modality remains vulnerable to several physical-world failure modes. This paper presents a systematic vulnerability analysis of LiDAR-based autonomous driving stacks against mirror-induced perception failures. We identify, formalize and evaluate two distinct failure modes arising from planar mirrors in the environment: *Object Addition*, where multi-path reflections create phantom obstacles in free space, and *Object Removal*, where beam deflection renders real obstacles invisible to the sensor. Our analysis makes three contributions. First, we conduct an end-to-end vulnerability assessment on a full-stack autonomous vehicle running Autoware, tracing how mirror-induced artifacts propagate from raw perception through planning and control, ultimately triggering emergency braking and cascade collisions. Second, we derive physics-informed empirical models that characterize artifact location, point count, and appearance probability as functions of distance, angle, and mirror area. Third, we integrate these models into a CARLA-based simulation framework for real-time artifact injection, addressing a key limitation of current simulators that do not model specular reflections. Our findings reveal that even small mirrors ($0.18\,m^2$, costing under \$60) can cause high-confidence false detections (74% "Vehicle" classification) and trigger unsafe driving behaviors. We discuss potential countermeasures and highlight that robust defenses against physically legitimate but geometrically deceptive sensor inputs remain an open challenge.

1 Introduction

The deployment of autonomous driving systems on public roads depends on the ability of their perception pipelines to construct an accurate, trustworthy model of the physical world. Among the sensor modalities employed, Light Detection and Ranging (LiDAR) plays a central role, providing high-resolution 3D point clouds that form the basis for obstacle detection, localization, and path planning [14, 30]. The integrity of LiDAR data is therefore not merely a performance concern but a safety-critical requirement. Errors in the raw sensor data can propagate through the entire autonomy stack, from perception through planning to control, with potentially catastrophic consequences.

A substantial body of research has investigated LiDAR vulnerabilities through *active* spoofing attacks, which use external laser emitters to inject forged pulses into the sensor [10, 18, 19, 21, 26], or through *passive* adversarial objects, such as 3D-printed shapes or specially crafted road patterns designed to exploit weaknesses in learned perception models [1, 24, 27]. Active attacks require specialized hardware and precise synchronization with the victim sensor. Passive adversarial objects typically target specific neural network architectures rather than the fundamental sensing modality.

However, a more fundamental class of vulnerability has received remarkably little attention: the susceptibility of LiDAR to deception through ordinary specular reflection. The robotics community has long recognized that mirrors and reflective surfaces create problematic artifacts in LiDAR data [3, 28], and recent work has begun to explore reflective materials for targeted point removal [11, 12]. Yet the broader implications of this physical phenomenon, particularly the ability of simple mirrors to both *create* phantom objects and *erase* real ones, have not been systematically analyzed in the context of full-stack autonomous driving systems.

This vulnerability is particularly concerning for several reasons. First, unlike active spoofing, mirror-based artifacts are generated from the vehicle's own laser pulses, making the returned signals *physically legitimate* in terms of wavelength and temporal structure. Defenses designed to reject externally injected pulses are therefore ineffective against such physically consistent reflections. Second, the artifacts are structured and spatially coherent, following the deterministic laws of geometric optics rather than appearing as random noise. Third, mirror placement requires no electronics, no precise synchronization, and no knowledge of the victim system's internal parameters. Only general knowledge of road geometry and

Table 1: Comparison of mirror-based LiDAR vulnerability analysis with prior work on LiDAR deception.

Work	Mirror-Based	Passive	Object Addition	Object Removal	Geometric Modeling	Simulation	Real-World Validation	AV Stack Evaluation	Defense Proposal
Ours	●	●	●	●	●	CARLA	Outdoor (Vehicle)	Autoware	●
[24]	○	●	●	○	○	○	Indoor (Basic)	○	○
[1]	○	●	●	○	○	○	Outdoor	○	○
[27]	○	●	●	○	●	LGSVL	Outdoor	○	●
[11]	●	●	○	●	○	AWSIM	○	○	●
[12]	●	●	○	●	○	AWSIM	Outdoor (Basic)	○	●
[31]	○	●	○	○	●	SVL	Outdoor	○	●
[2]	○	○	●	○	○	Baidu Apollo	Indoor (Basic)	○	●
[22]	○	○	●	○	●	Dataset	Indoor (Basic)	○	●
[32]	○	●	○	●	●	Dataset	Outdoor (Drones)	○	●

typical sensor mounting configurations is required.

Table 1 provides a comparative overview of prior work. To our knowledge, prior research has not provided a systematic, end-to-end analysis of mirror-induced LiDAR deception in autonomous driving systems. We identify four specific gaps: **(G1)** Prior studies evaluated reflective artifacts only at the point-cloud or perception-model level [2, 22, 31], without tracing their propagation through a complete autonomy pipeline to quantify system-level safety implications. **(G2)** No empirical or analytical models exist to describe how artifact properties depend on controllable physical parameters such as angle, distance, and area, preventing systematic prediction or reproduction [3, 28]. **(G3)** Current LiDAR simulators [4] rely on simplified single-bounce ray-casting that does not model specular multi-path reflections. Consequently, mirror-induced artifacts cannot be realistically reproduced in simulation, limiting controlled and repeatable evaluation. **(G4)** Recent mirror-related work [11, 12] focused narrowly on point removal, without exploring the dual capability of mirrors to also *inject* structured phantom objects.

This Work. We present the first end-to-end vulnerability analysis of LiDAR-based autonomous driving systems against mirror-induced perception failures. Our approach follows a structured methodology consisting of real-world vulnerability discovery, physics-informed modeling, simulation-based scalable evaluation, and full-stack impact validation. Specifically, we make the following contributions:

- **End-to-End Vulnerability Assessment (G1):** We evaluate the impact of mirror-induced artifacts on a fully integrated autonomous vehicle running the Autoware stack with a commercial LiDAR. We trace how artifacts propagate through perception, from point cloud to occupancy grid to object detection, and into planning and control. We demonstrate that these artifacts trigger false emergency braking, high-confidence phantom vehicle classifications of up to 74%, and cascade collision scenarios.
- **Physics-Informed Empirical Models (G2):** We derive and validate four analytical models that characterize mirror-induced LiDAR artifacts, including radial distance, lateral offset, point count, and appearance probability, as functions of distance, mirror angle, and area. These models are grounded in geometric optics and calibrated through controlled outdoor experiments.
- **Simulation Framework for Specular Artifacts (G3):** We implement a real-time, model-driven artifact injection pipeline in CARLA that generates physically realistic mirror artifacts during simulation. This framework bridges the sim-to-real gap for specular reflections and enables safe, repeatable testing of diverse configurations.
- **Comprehensive Threat Characterization (G4):** We formalize two complementary failure modes, Object Addition, defined as phantom obstacle injection, and Object Removal, defined as real obstacle concealment. We demonstrate these failure modes through a combination of real-world experiments and simulation, providing a comprehensive characterization of the vulnerability surface, as illustrated in the end-to-end pipeline shown in Figure 1.

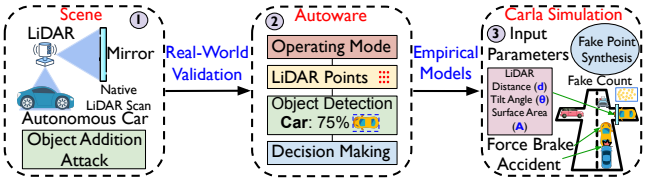


Figure 1: End-to-end evaluation pipeline. (①) Real-world experiments characterize mirror-induced artifacts and produce empirical models. (②) Key outcomes are validated on an Autoware-equipped vehicle, confirming perception failures and unsafe driving behaviors. (③) Empirical models drive real-time artifact injection in CARLA for scalable evaluation.

Responsible Disclosure. All experiments were conducted in a controlled environment with no risk to public safety. We disclose this vulnerability to support the development of reflection-aware perception and validation systems. Demon-



Figure 2: Types of reflection in LiDAR sensing. (a) Diffuse: incident rays scatter broadly on rough surfaces. (b) Specular: a smooth mirror reflects coherently with $\theta_r = \theta_i$.

stration videos and supplementary materials are available at: https://lidar-attack.github.io/lidar-research/.

2 Background

2.1 LiDAR Operation and Reflection Physics

LiDAR sensors emit pulses of near-infrared laser light and compute the distance to a surface using the time-of-flight (ToF) principle: $d_l = \frac{c \cdot \Delta t}{2}$, where c is the speed of light and Δt is the measured round-trip time. Contemporary systems scan the environment using rotating or solid-state arrays with fixed angular resolution, producing a 3D point cloud $P = \{(x_i, y_i, z_i, r_i)\}_{i=1}^{N}$, where (x_i, y_i, z_i) is the 3D location and r_i the return intensity. Point cloud fidelity is therefore tightly coupled to how beams interact with physical surfaces.

Two surface interaction modes determine LiDAR returns (Figure 2). *Diffuse reflection* occurs on rough surfaces, scattering return energy broadly and allowing detection from multiple angles. *Specular reflection* occurs on smooth, mirror-like surfaces (glass, wet roads, glossy vehicle bodies) and follows the law of reflection:

$$\vec{v}_{\text{ref}} = \vec{v}_{\text{in}} - 2\,(\vec{v}_{\text{in}} \cdot \vec{n})\,\vec{n}, \tag{1}$$

where $\vec{v}_{\text{in}}$ is the incoming ray and $\vec{n}$ is the surface normal. This specular behavior is the source of the measurement artifacts central to this work. We note that real mirrors absorb a small fraction of incident energy at each reflection (typically 3–8% per bounce for silver-backed glass). For the two-hop paths in our scenarios, the cumulative energy loss remains below the sensor's detection threshold in most configurations, as validated by our experimental data. At greater path lengths or with additional bounces, absorption could reduce artifact intensity below detectable levels, a factor captured by the distance-dependent attenuation in our empirical models.

2.2 Mirror-Induced LiDAR Measurement Artifacts

When a LiDAR beam strikes a specular surface, its path is redirected according to the law of reflection. This can corrupt the resulting point cloud through two primary mechanisms:
Data Omission (Figure 3a) occurs when a mirror deflects a LiDAR beam away from the sensor, preventing a return signal.

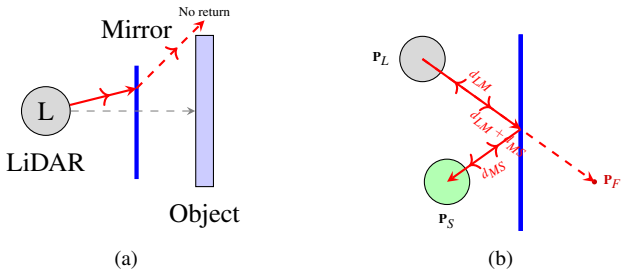


Figure 3: Primary LiDAR measurement artifacts from specular reflections. (a) Data omission: beam reflected away, no return. (b) Data fabrication: two-hop path creates a virtual point at $\mathbf{P}_F$.

A beam that would normally strike a real object ($\mathbf{P}_O$) instead reflects along a path that never returns, causing the LiDAR to record empty space. The object is effectively removed from the point cloud.
Data Fabrication (Figure 3b) occurs when a LiDAR beam is redirected by a mirror, strikes a secondary surface ($\mathbf{P}_S$), and returns to the sensor via the same mirror. The LiDAR assumes a direct line of sight and misinterprets the total travel time:

$$d_{\text{measured}} = \frac{d_{\text{total}}}{2} = d_{LM} + d_{MS}, \tag{2}$$

where d_{LM} is the LiDAR-to-mirror distance and d_{MS} is the mirror-to-secondary-surface distance. The system projects this total distance along the original emission direction, placing the reflected object at a false location $\mathbf{P}_F$, corresponding to the geometric mirror image of the actual surface. This creates phantom obstacles in empty space.

3 Threat Model

Urban driving environments contain numerous specular surfaces, including glass storefronts, building facades, bus shelters, and traffic convex mirrors, that can redirect LiDAR pulses along unintended paths. While such reflections occur naturally and may cause transient artifacts, the underlying vulnerability can also be *deliberately exploited* by an adversary who controls the placement or orientation of a reflective surface. This section formalizes the threat across a spectrum from incidental to adversarial.

3.1 Adversary Goals

The adversary seeks to introduce physically plausible but semantically deceptive artifacts into the AV's LiDAR point cloud, leading to unsafe or disruptive driving behavior. We define two complementary failure modes:
• **Object Addition (OA):** Create a *false positive* by fabricating a phantom obstacle in the AV's path, aiming to trigger an inappropriate defensive response such as emergency braking in flowing traffic or an unnecessary evasive maneuver.

• **Object Removal (OR):** Create a *false negative* by making a real obstacle invisible to the LiDAR sensor, causing the AV to fail to detect a genuine hazard.

Both failure modes produce structured, plausible point clouds rather than obvious noise, making them difficult to distinguish from legitimate sensor data at the perception level.

3.2 Adversary Capabilities and Constraints

The adversary operates as a *passive, external* agent in the physical domain with the following profile: (1) **Black-box access:** no internal access to the AV's software, sensor data, or control systems; the manipulation is non-invasive, relying only on observable physical reactions. (2) **Passive mechanism:** no electronic emitters are used; unlike active spoofing [20, 21], the adversary redirects the victim's own LiDAR pulses, producing physically legitimate but geometrically deceptive returns. (3) **General environmental knowledge:** the adversary knows the road layout and typical LiDAR mounting positions but not the specific vehicle's calibration parameters. (4) **Physical mirror placement:** one or more portable planar mirrors are placed before the AV, potentially disguised as infrastructure elements (utility boxes, construction signs, roadside art).

Adversary Controllability. The adversary's ability to manipulate phantom artifacts is constrained by the underlying two-hop reflection geometry (Section 2), yet remains sufficiently structured to enable practical attacks. According to the law of reflection (Equation 1), the adversary can act on three physical variables: mirror placement, mirror orientation (tilt angle θ), and reflective area A. Specifically, *phantom location* is governed by the combination of mirror placement and orientation. Mirror placement sets the LiDAR-to-mirror distance d, which in turn scales the lateral displacement $d\tan(2\theta)$ and the apparent depth along the two-hop path (Equation 2). Consequently, by positioning the mirror at a chosen standoff distance from the victim's trajectory and selecting an appropriate tilt, the adversary can place the phantom within the victim's planned path (e.g., at an intersection turning corridor). *Phantom density*, quantified as the number of reflected points per frame, is primarily controlled by the mirror area A. Larger surfaces intercept a greater fraction of the divergent LiDAR beam and produce denser clusters.

Phantom shape is not directly chosen by the adversary but is determined by the mirror image of whatever secondary surface the deflected beam strikes. The secondary surface is therefore environmental rather than adversary-controlled. In typical urban deployments, walls, facades, parked vehicles, and the ground plane provide abundant candidates at predictable distances from intersection corners. Even in the absence of such infrastructure, the ego vehicle itself can serve as the secondary surface, yielding a naturally vehicle-shaped phantom that might be misclassified by the perception stack. Consequently, the feasibility of the attack does not hinge on fine-grained environmental control by the adversary.

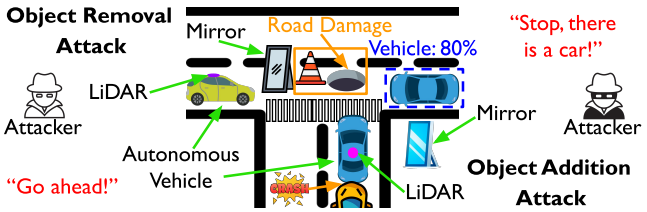


Figure 4: Illustrative scenarios. **OR**: A mirror redirects LiDAR beams away from a traffic cone, creating the illusion of a clear path. **OA**: A mirror reflects pulses to create a phantom obstacle, triggering a false positive.

3.3 Threat Spectrum: Incidental to Adversarial

Mirror-induced LiDAR artifacts can arise across a spectrum of intent, from incidental environmental reflections to deliberate adversarial placement of reflective surfaces.

• **Non-adversarial (environmental):** Glass building facades, store windows, wet road surfaces, and metallic infrastructure can produce unintended specular returns.

• **Semi-adversarial (repurposed infrastructure):** Convex mirrors are routinely deployed at blind curves and narrow intersections for human driver visibility. An adversary could subtly re-angle such a mirror/install an additional one that appears functionally legitimate. Because convex mirrors are *expected* in these environments, no modifications are required.

• **Fully adversarial (dedicated placement):** Purpose-built reflector deployment, intended to induce dense and spatially coherent artifacts, mounted on roadside infrastructure or disguised as a traffic convex mirror at an intersection corner.

3.4 Representative Scenarios

We ground the threat model with two deployment scenarios, illustrated in Figure 4:

• **Scenario 1: Intersection Blocking via Object Addition.** A mirror is positioned at an urban intersection corner, oriented toward the approaching AV's expected trajectory. This mirror causes the LiDAR sensor to register phantom points in the AV's turning path, where no real obstacle exists. The perception system interprets these as a legitimate obstruction, triggering unnecessary braking or evasive maneuvers.

• **Scenario 2: Obstacle Concealment via Object Removal.** A mirror is placed at the roadside to intercept LiDAR beams that would otherwise detect a real obstacle (e.g., a construction barrier). Instead of returning from the obstacle, the sensor receives returns consistent with a clear road surface. The AV consequently infers a traversable path through an actually obstructed region.

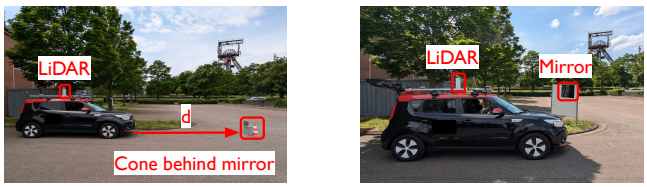


Figure 5: Experimental setups. **(a)** Object Removal: a mirror obscures a traffic cone from the LiDAR's line of sight. **(b)** Object Addition: a mirror array redirects LiDAR pulses toward a nearby structure, creating phantom returns in the vehicle's path.

4 Real-World Experimental Campaign

This section describes the controlled outdoor experiments we conducted to characterize the physical behavior of mirror-induced LiDAR artifacts. The experiments serve two purposes: (1) providing empirical evidence of the vulnerability on a real sensor platform, and (2) generating the dataset from which our empirical models (Section 6) are derived.

4.1 Platform and Setup

All experiments used a test vehicle with a roof-mounted *Ouster OS1-128* LiDAR sensor at approximately 2.2 m height, operating at 10 Hz with 128 vertical channels and full 360° horizontal coverage [16]. Data was recorded using ROS 2, logging raw LiDAR packets for precise offline analysis.

We used optically flat, silver-backed glass mirrors mounted on adjustable stands with angular protractors for tilt control. Two mirror configurations were employed: single panels of approximately 60 cm×40 cm for OR tests, and modular arrays of 30 cm×30 cm tiles (2, 4, or 6 tiles, yielding 0.18, 0.36, and 0.6 m^2 of reflective area) for OA tests. Total cost of all mirror hardware was approximately $60.

Testing was conducted in a closed campus parking lot and adjacent roadway under clear, dry daytime conditions. The vehicle was driven at speeds up to 8 km/h to replicate low-speed approach scenarios (e.g., intersection turns, parking), enabling capture of the full temporal evolution of artifacts across multiple LiDAR frames as the geometry between vehicle and mirror changed.

4.2 Object Removal Trials

In the OR setup (Figure 5a), a traffic cone (50 cm tall, 29 cm base) was placed 4 m from the vehicle. A single mirror (60×40 cm) was positioned between the LiDAR and the cone, covering the cone's vertical extent as seen by the sensor. We tested mirror tilt angles of 0°, 15°, 30°, and 45° relative to the LiDAR's forward axis. Each configuration was recorded for 10 seconds (∼100 LiDAR frames), with the vehicle stationary to isolate the mirror's effect.

Observation. Across all tested configurations, the cone was completely absent from the point cloud when the mirror was present (Figure 6a and Figure 6b). This demonstrates that a mid-sized obstacle can be rendered fully invisible through passive beam redirection. Critically, this failure mode is fundamentally different from simple occlusion by an opaque object or from beam redirection into empty space: the mirror deflects pulses toward the ground plane, where they produce valid-looking ground returns that are spatially continuous with the surrounding floor (Figure 6c). Rather than leaving a visible data void, the obstructed region is populated by geometrically consistent returns, causing the sensor to perceive a continuous traversable surface where the cone is located. This distinction is what makes OR particularly insidious. Void-based anomaly detectors that reason about missing returns or shadow regions [8] can catch opaque occlusion, but cannot catch this substitute-surface mechanism where the voided region is replaced by geometrically consistent ground returns.

4.3 Object Addition Trials

In the OA setup (Figure 5b), a modular mirror array was mounted on a vertical board at an intersection edge. We tested three tilt angles (30°, 45°, 60°) and three reflective areas (0.18, 0.36, 0.6 m^2), yielding a 3×3 parameter grid. The vehicle approached from ∼20 m at 8 km/h, executing an intersection turn while continuously recording LiDAR data.

Observation. Virtual points behind the mirror were consistently observed across all trials (Figure 7). Two key findings emerged. First, **mirror area controls artifact density**: increasing from 0.18 m^2 to 0.6 m^2 transformed the artifact from a sparse cluster into a dense, cohesive structure resembling a solid object. Second, **mirror angle controls artifact timing and location**: shallow angles (30°) induced abrupt, close-range phantoms that emerged at short LiDAR-to-mirror distances, whereas steep angles (60°) produced more spatially persistent artifacts detectable earlier in the vehicle's approach and at larger perceived depths, as shown in Figure 8 (extended analysis is provided in Appendix B.1). The 45° case exhibited intermediate behavior with noticeable frame-to-frame variability in artifact persistence and spatial consistency. These observations reveal a key trade-off: *angle* determines artifact *placement*, while *area* determines artifact *density* and *perceptual plausibility*.

5 Full-Stack Vulnerability Assessment on Autoware

This section presents the core vulnerability assessment: tracing how mirror-induced artifacts propagate through a fully integrated autonomous driving system and trigger safety-critical failures. Building on the experimental evidence from the previous section, we demonstrate the end-to-end impact on perception, planning, and control using an Autoware-equipped

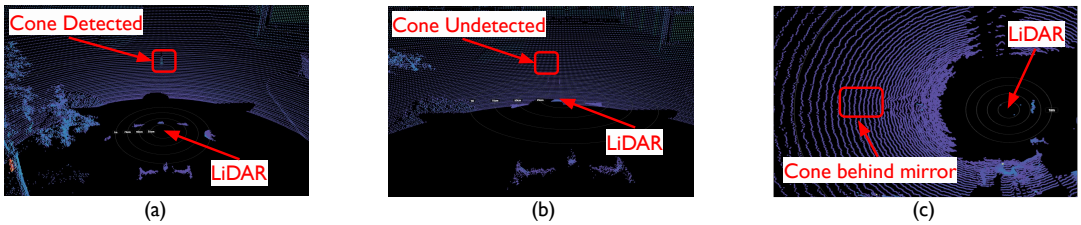


Figure 6: Object Removal at 0° mirror tilt. **(a)** Baseline: cone visible. **(b)** Side view with mirror: cone completely removed from perception. **(c)** Top view with mirror: the region where the cone was located is filled with ground returns that are spatially consistent with the surrounding floor, rather than showing a data void.

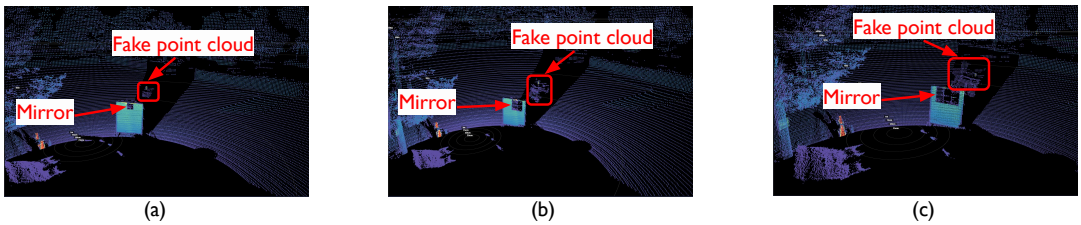


Figure 7: OA point clouds with varying mirror area. **(a)** 2 tiles: sparse cluster (circled). **(b)** 4 tiles: denser vertical surface. **(c)** 6 tiles: high-density cluster approximating a vehicle profile.

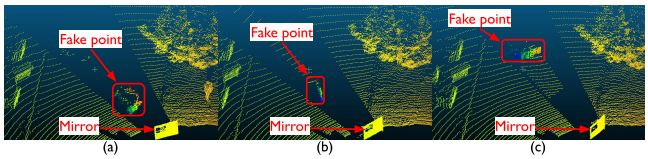


Figure 8: Impact of mirror tilt angle on phantom location: **(a)** 30°; **(b)** 45°; **(c)** 60°.

test vehicle, showing that phantom points at the sensor level cascade into emergency braking, unnecessary stops, and disrupted trajectory planning.

5.1 Methodology

The validation used the same Kia Soul EV with Ouster OS1-128 LiDAR described in Section 4, running the Autoware.Universe stack with CenterPoint [29] as the primary 3D object detector. CenterPoint was selected for its strong performance on standard benchmarks and its deployment in real-world AV systems. Experiments were conducted in two modes:

- **Manual Mode:** A safety driver maintained precise trajectories while logging perception outputs, enabling clean analysis of how the perception stack interprets adversarial inputs.
- **Autonomous Mode:** The full autonomy stack (perception, planning, control) was engaged, allowing observation of the system's emergent behaviors in response to artifacts.

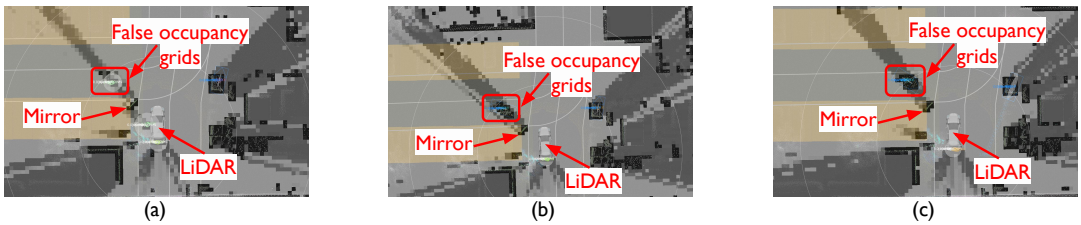
Each scenario was tested across the full parameter grid (three mirror areas × three tilt angles). Synchronized logs captured occupancy grids, detected object lists, and vehicle control signals.

5.2 Perception-Level Failure Analysis

① **Object Addition: Phantom Detection and Misclassification.** Under OA conditions, mirror-reflected returns corrupt the perception pipeline at multiple levels. In the occupancy grid (Figure 9), two mirror tiles mark several cells as "occupied." With four and six tiles, the occupied region grows to match a vehicle-sized footprint, persisting across multiple frames as the mapping module accumulates occupancy evidence from consecutive LiDAR scans.

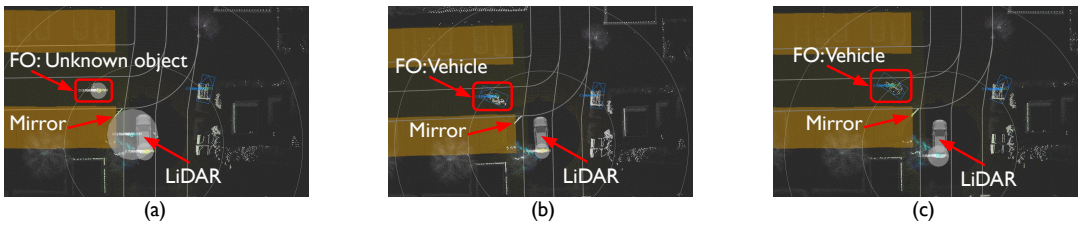
At the semantic level, the corruption escalates from spatial to categorical (Figure 10). With two tiles, the sparse cluster is classified as "UNKNOWN", but still treated as an obstacle by downstream modules. With four tiles, the denser artifact triggers a "CAR" classification at 65% confidence. With six tiles, confidence rises to 74%. The detector progressively hallucinates a plausible, high-confidence vehicle where none exists, revealing a fundamental breakdown in perception integrity: the physically legitimate but geometrically deceptive signals are indistinguishable from real objects within the detector's learned feature space.

② **Object Removal: Silent Obstacle Erasure.** The OR scenario introduces a particularly insidious failure: the mirror redirects LiDAR beams away from the cone and toward the ground plane, where they produce geometrically consistent returns. These reflected signals are processed as structurally valid measurements, and Autoware's ground segmentation interprets them as part of a continuous, traversable surface. As shown in Figure 11, the baseline correctly marks the cone as occupied (black cell), while the mirror condition shows the same region as free space despite the physical presence of both the cone and the mirror. This data replacement makes the OR vulnerability particularly dangerous. It can pass through



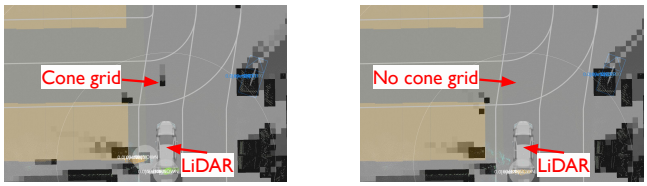
(a) (b) (c)

Figure 9: Occupancy grid corruption under Object Addition. **(a)** 2 tiles: thin line of occupied cells behind mirror. **(b)** 4 tiles: dense cluster triggers "CAR" detection. **(c)** 6 tiles: occupied area matches a real vehicle footprint.



(a) (b) (c)

Figure 10: CenterPoint detection outcomes under Object Addition. **(a)** 2 tiles: "UNKNOWN" detection (white circle). **(b)** 4 tiles: "Vehicle" at 65% confidence (blue box). **(c)** 6 tiles: "Vehicle" at 74% confidence.



(a) (b)

Figure 11: Occupancy grid under Object Removal. **(a)** Baseline: cone correctly marked occupied. **(b)** With mirror: same region falsely marked free.

anomaly detection systems that search for sensor voids or dropouts, and it compromises safety architectures that treat LiDAR as ground-truth for obstacle detection.

5.3 System-Level Behavioral Failures

Having characterized the perception-level impact, we now examine how OA artifacts propagate to planning and control in **autonomous mode**. We focus on OA using the 6-tile configuration (maximum artifact density) across three mirror angles, with the vehicle following a route toward the mirror setup at 8 km/h.

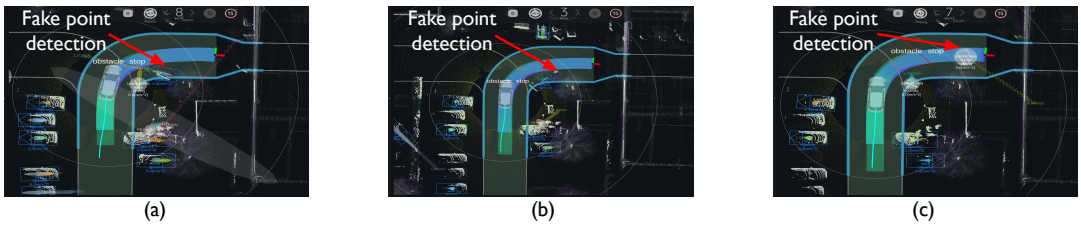
Observed Failures. In every trial, the planner interpreted the phantom as a real threat and engaged its *obstacle stop* procedure. However, the nature of the failure varied significantly with mirror angle, as shown in Figure 12, producing a spectrum of dangerous behaviors:

• **Late-Onset, High-Jerk Braking (30°):** At a shallow 30° angle, the reflective geometry only produced a detectable phantom when the vehicle was within a few meters of the mirrors. As shown in Figure 12a, the vehicle has almost completed the turn's apex when the phantom object (blue box) appears abruptly at extremely close range. This late-onset detection created a critically low Time-to-Collision (TTC), forcing the motion planner to issue an immediate emergency *obstacle stop* command. The vehicle halted with an extremely high jerk profile, a maneuver that, while logically correct from the system's perspective, would be highly dangerous in a real traffic environment and would likely cause a rear-end collision by a following vehicle.

• **Mid-Turn Stop with Decision Instability (45°):** Increasing the angle to 45° widens the window for reflection. Consequently, the phantom object is detected earlier in the vehicle's approach, as it begins to enter the curve but before it reaches the apex. The RViz view shows (see Figure 12b) the vehicle braking while still on the initial part of the turn. The planner has slightly more time to react compared to the 30° case, but the outcome is still a full stop in the middle of a dynamic maneuver. This scenario can be particularly hazardous if it leads to detection flickering, causing decision instability (oscillating between braking and proceeding) and creating an unpredictable hazard for other road users.

• **Pre-Turn Path Obstruction (60°):** At 60°, the mirror presents a large, effective reflective surface to the approaching vehicle. This results in the earliest possible detection, occurring well before the vehicle even enters the turn. The image clearly shows the phantom object appearing as a full obstruction far down the planned trajectory (the light white circle). The system, identifying a high-confidence obstacle directly in its path, immediately terminates the plan and commands a full stop. While an early stop might seem safer, it is still a hazardous phantom braking event. The vehicle stops for no apparent reason long before an intersection, disrupting traffic flow and creating a dangerously unpredictable situation for human drivers who cannot see the phantom obstacle.

Broader Safety Implications. This analysis demonstrates



(a) (b) (c)

Figure 12: Autoware's response to phantom detections with the 6-tile mirror at different angles. **(a)** 30°: late-onset phantom triggers immediate emergency stop. **(b)** 45°: earlier detection causes mid-turn braking. **(c)** 60°: early detection triggers pre-turn full stop.

that the vulnerability triggers a spectrum of dangerous *errors of commission*: the vehicle performs unnecessary, hazardous actions ranging from sudden high-g emergency stops to inexplicable halts in live traffic. The severity and timing are directly tunable by the mirror angle parameter.

Although tests were conducted at 8 km/h for safety, autonomous vehicles typically approach intersections at 10–20 km/h, where reduced reaction time and increased momentum amplify the risk of cascade collisions. Crucially, in every case, the Autoware stack performed *exactly as designed*: executing a logical, defensive maneuver based on its sensory input. The vulnerability lies not in the planning software but at the physical-sensor interface, where the vehicle's perception of reality was corrupted by physically legitimate but geometrically deceptive signals. This finding underscores that robust AV safety must extend beyond algorithmic integrity to include defenses against physical-layer sensor deception.

Note on OR System-Level Evaluation. For the OR scenario, we did not evaluate full autonomous driving through the removed obstacle, as doing so would require the vehicle to drive into a physical hazard. Instead, we verified the system-level implication at the perception level: Autoware's occupancy grid and planner consistently treated the mirror-occluded region as free space (Figure 11), confirming that the vehicle would proceed into the obstructed zone if operating autonomously. A full collision demonstration for OR would require a simulation-only evaluation with a destructible obstacle, which we leave to future work.

6 Physics-Informed Empirical Modeling

The full-stack assessment revealed that even a small number of phantom points can cascade into safety-critical failures. This section derives analytical models that characterize mirror-induced artifacts as functions of controllable physical parameters, formalizing the geometric and statistical mechanisms that govern artifact appearance, depth, and intensity. These models serve two purposes: (1) they provide a quantitative understanding of the vulnerability's underlying physics, and (2) they enable realistic artifact injection in simulation.

6.1 Feature Extraction Pipeline

From the experimental data (Section 4), we extracted mirror-induced artifact points through a frame-differencing procedure. For each trial, we aligned the mirror-present scan with a baseline scan captured at the same vehicle position and under identical conditions but without the mirror. Points appearing exclusively in the mirror-present scan were labeled as mirror-induced artifacts. We applied fine-grained spatial registration using the Iterative Closest Point (ICP) algorithm to compensate for minor pose deviations and vehicle motion between runs. The resulting artifact clusters were segmented and annotated with the physical parameters (d, θ, A): LiDAR-to-mirror distance, mirror tilt angle, and mirror surface area.

From each cluster, we extracted four quantitative features: (1) **Radial distance** (R_{art}): the magnitude of the centroid vector, representing the artifact's apparent distance from the sensor; (2) **Lateral offset** (X_{art}): the displacement of the centroid from the LiDAR's forward axis; (3) **Point count** (N_{art}): the number of points per frame, serving as a proxy for artifact salience; (4) **Appearance probability** (P_{app}): the fraction of frames in which an artifact cluster (with $N > 0$) appears under a given configuration.

6.2 Model Formulation

This section models the dependence of artifact characteristics on mirror distance d, tilt angle θ, and surface area A. We propose four semi-empirical formulations derived from reflection geometry and LiDAR time-of-flight behavior and validate them against the experimental observations in Section 4. All fitted parameters and their physical interpretations are provided in Appendix A.

❶ **Artifact Location Model.** The artifact's perceived position is determined by the two-hop path: the LiDAR pulse strikes the mirror, reflects toward a secondary surface (e.g., wall or ground), bounces back to the mirror, and returns to the sensor. Because the sensor interprets the total round-trip time as a straight-line range, it registers a phantom point at a location with no real object. We derive lateral and radial coordinates separately.

Lateral Offset. A planar mirror rotated by a yaw angle θ relative to the sensor's optical axis deflects the beam laterally

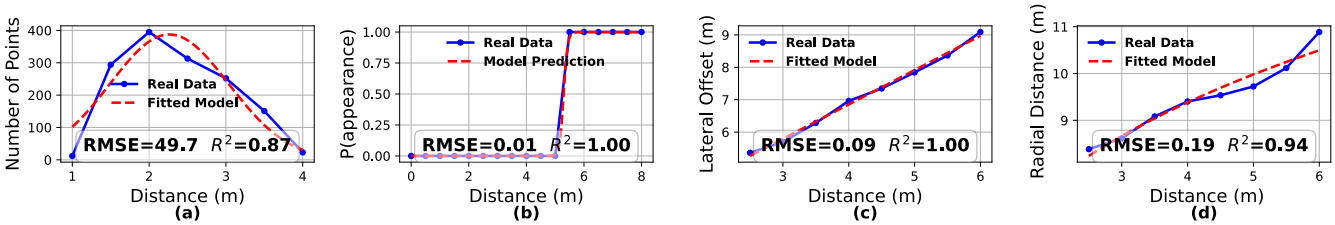


Figure 13: Model validation against experimental data for representative configurations. **(a)** Point count model captures the bell-shaped intensity profile (30°, 0.36 m^2). **(b)** Appearance probability model delineates the detection window (60°, 0.6 m^2). **(c, d)** Lateral offset and radial distance models track artifact location (45°, 0.36 m^2).

by 2θ (law of reflection). This angular deflection produces a lateral displacement that grows with propagation distance. Since the post-reflection path scales approximately with the sensor-to-mirror distance d during vehicle approach, the first-order displacement is:

$$X_{\text{geom}} = d\tan(2\theta). \tag{3}$$

In practice, the sensor and mirror are mounted at different heights, and the beam strikes the mirror as a finite spot rather than an infinitesimal point. The height difference slightly alters the effective deflection geometry (multiplicative correction c_X), while the finite beam footprint on the tilted surface introduces a systematic lateral bias (additive shift δ_X):

$$X_{\text{art}}(d,\theta) = c_X\, d\tan(2\theta) + \delta_X. \tag{4}$$

Fitted values ($c_X \approx 0.98$, $\delta_X \approx -0.05$ m) confirm close agreement with the ideal geometry of Equation 3.

Radial Distance. The LiDAR computes range as half the total round-trip time multiplied by the speed of light. For the two-hop path, this distance is necessarily longer than the direct round-trip to the mirror. With a perfectly collimated beam, the perceived range would grow linearly with d. However, real beams diverge: the beam diameter on the mirror grows as $d\cdot\phi$ (where ϕ is the divergence angle), so for a mirror of width w the captured fraction is:

$$f_{\text{capture}}(d) \approx \min\left(1,\ \frac{w}{d\,\phi}\right). \tag{5}$$

As d increases, the beam overfills the mirror and only the central portion is reflected. Central rays travel shorter geometric paths than peripheral ones, so clipping the edges biases the LiDAR's timing estimate (which operates on the peak or leading edge of the return pulse) toward shorter values. This produces sub-linear growth of perceived range:

$$R_{\text{base}}(d) = c_R\, d^{n_d}, \quad n_d < 1. \tag{6}$$

Mirror tilt θ also modifies the total path length by changing how far the deflected beam travels to reach the secondary surface. Expanding this angular dependence to second order gives the full model:

$$R_{\text{art}}(d,\theta) = c_R\, d^{n_d}\,(1 + a_1\theta + a_2\theta^2), \tag{7}$$

where fitting yields $n_d \approx 0.88$, confirming the expected sub-linear regime. Mirror area A has negligible effect on artifact location, consistent with geometric optics: image position depends on angles and distances, not mirror size.

❷ **Artifact Point Count Model.** The LiDAR records a point only when received power exceeds a threshold P_{min}. For a single-hop return at range R:

$$P_r = \frac{P_t\,\rho\,A_r}{R^4}\,\eta, \tag{8}$$

where P_t is transmitted power, ρ target reflectivity, A_r receiver aperture, and η system efficiency. In the two-hop case, the effective range, cross-section, and reflectivity become coupled functions of (d,θ,A), precluding a closed-form solution. We identify three dominant effects and model each independently.

Mirror area A. A larger mirror intercepts more of the divergent beam. For a Gaussian beam profile, captured power scales super-linearly when the mirror is smaller than the footprint (the beam center carries disproportionately more energy), motivating a power-law term A^{β} with $\beta > 1$.

Mirror tilt θ. Tilting reduces the projected area seen by the beam (foreshortening). In our two-hop geometry this loss applies on both incoming/outgoing reflections, and angular misalignment further reduces return coupling, producing a steeper-than-cosine falloff modeled as $\cos^{\gamma}(\theta)$ with $\gamma > 1$.

Distance d. The dependence is non-monotonic. At short range, the deflected beam overshoots nearby surfaces and the return angle exceeds the receiver's field of view. At long range, R^{-4} attenuation dominates. Between these regimes, a peak occurs at an optimal distance $\mu(\theta)$. We model this bell-shaped envelope with a Gaussian:

$$g(d,\theta) = \exp\left(-\frac{(d-\mu(\theta))^2}{2\sigma^2}\right), \tag{9}$$

where $\mu(\theta)$ shifts the peak with angle and σ controls the width. Combining the three effects:

$$N_{\text{art}}(d,\theta,A) = c_0 A^{\beta}\cos^{\gamma}(\theta)\exp\left(-\frac{(d-\mu(\theta))^2}{2\sigma^2}\right), \tag{10}$$

with fitted $\beta \approx 1.25$, $\gamma \approx 3.5$.

❸ **Appearance Probability Model.** An artifact is detectable only when (i) the deflected beam actually hits a secondary surface rather than escaping into open space, and (ii) the return signal exceeds $P_{\min}$. At short range, the steep 2θ deflection sends the beam past nearby surfaces, making the two-hop path physically impossible. At long range, the extended multi-leg path causes $1/R^4$ decay to push the signal below the noise floor. These opposing constraints create a finite detection window $[d_{\min}, d_{\max}]$.

The transitions at both boundaries are gradual (the fraction of valid rays and the SNR both degrade smoothly), so we model each edge with a logistic sigmoid and compose them into a bandpass window:

$$P_{\text{app}}(d,\theta,A) = \frac{1}{1+e^{-k(d-d_{\min})}} \cdot \frac{1}{1+e^{k(d-d_{\max})}}, \qquad (11)$$

where k controls transition steepness. The boundaries shift with mirror configuration: steeper θ deflects the beam more aggressively, requiring a greater distance for the geometry to work, pushing $d_{\min}$ outward; larger A captures more energy and extends the detectable range, increasing $d_{\max}$. We model both as linear functions:

$$d_{\min}(\theta,A) = b_0^{\min} + b_1\theta + b_2 A, \qquad (12)$$

$$d_{\max}(\theta,A) = b_0^{\max} + c_1\theta + c_2 A. \qquad (13)$$

6.3 Model Fitting and Validation Summary

Figure 13 presents representative validation results. The lateral offset model (c) achieves a near-perfect fit ($R^2 \approx 0.98$ globally), confirming that the first-order geometric prediction $d \cdot \tan(2\theta)$ accurately describes the sideways displacement. The radial distance model (d) achieves $R^2 \approx 0.92$, with the sub-linear exponent $n_d = 0.88$ consistently capturing the beam-spillover effect across all configurations. The appearance probability model (b) achieves $R^2 \geq 0.87$ across all tested setups, and $R^2 = 1.0$ for several, accurately reproducing the sharp on/off detection windows observed in the data. The point count model (a) achieves a global $R^2 \approx 0.85$; we note that certain configurations (e.g., 30°, $0.36\,\text{m}^2$) exhibit higher absolute residuals (RMSE $\sim$49.7) driven by the high peak counts at these geometrically favorable setups, but the relative error remains bounded and the model correctly captures the bell-shaped trend in all cases. The complete per-configuration validation across all setups is provided in Appendix A.1, and all fitted parameter values with their physical interpretations are tabulated in Appendix A.

7 Simulation-Based Evaluation in CARLA

Here we integrate the previously derived artifact models into CARLA [4] by injecting synthetic LiDAR points whose position and structure follow the predicted artifact behavior. CARLA, like most AV simulators, relies on ray casting that assumes purely diffuse reflections, so mirror-like materials produce no measurement artifacts despite being a known source of errors in real deployments. Consequently, perception stacks tested only in simulation never encounter reflection-induced failures. By generating artifacts according to our empirical formulations, we extend the simulator with realistic specular reflection effects and enable safe, repeatable evaluation of AV robustness under physically consistent conditions.

7.1 Injection Pipeline

The pipeline operates in real-time within CARLA's simulation loop and is formalized as a four-stage Algorithm 1:

Stage 1: State extraction. The pipeline begins by calculating the precise geometric relationship between the ego vehicle and the adversarial mirror. Using CARLA's APIs, we access the world-space transforms of both the LiDAR sensor and the mirror object. These are used to compute the three critical input parameters for our models: the instantaneous LiDAR-to-mirror distance (d), mirror tilt angle (θ), and mirror surface area (A). This step effectively translates the dynamic 3D world of the simulator into the specific parametric inputs our empirical models require.

Stage 2: Probabilistic trigger. The extracted state vector is passed into a stochastic gate to determine whether an artifact should be injected. The inherent randomness in real-world sensor physics, where signals fluctuate near detection thresholds, is captured by our empirically derived appearance probability P_{app}. For a given state (d,θ,A), the model outputs the likelihood of artifact presence. A uniformly distributed random number $r \in [0,1]$ is sampled, and the artifact is generated only if $r < P_{\text{app}}(d,\theta,A)$. The probabilistic mechanism reflects the intermittent flicker of marginal detections observed in practice.

Stage 3: Artifact synthesis. Once triggered, the pipeline synthesizes a phantom object based entirely on the validated empirical models. All characteristics are derived from the same state vector (d,θ,A). The point count model N_{art} determines the number of LiDAR points forming the artifact. Simultaneously, the location models X_{art} and R_{art} estimate the 3D centroid C_{art}. A cluster of N points is then sampled from a multivariate Gaussian distribution $\mathcal{N}(C_{\text{art}}, \Sigma_{\text{fixed}})$, where Σ_{fixed} is a small, constant covariance matrix. This process ensures a compact, spatially coherent artifact that resembles a physical object in the scene.

Stage 4: Point cloud injection. The final stage performs the insertion of the synthesized artifact cluster, $\mathcal{F}$, into the LiDAR data stream. The generated 3D points are merged with the native LiDAR scan, $\mathcal{L}_{\text{native}}$, using a set union: $\mathcal{L}_{\text{modified}} = \mathcal{L}_{\text{native}} \cup \mathcal{F}$. This modification is applied directly at the sensor-data level, prior to any downstream perception or control processing. As a result, the attack remains entirely transparent to the autonomy stack, which interprets the modified point

Algorithm 1 Model-Driven Artifact Injection Pipeline

Require: Current mirror configuration (d, θ, A)
Require: Native LiDAR scan $\mathcal{L}_{\text{native}}$
Require: Empirical models: $P_{\text{app}}, N_{\text{art}}, X_{\text{art}}, R_{\text{art}}$

$p_{app} \leftarrow P_{\text{app}}(d, \theta, A)$
if r() $< p_{app}$ **then**
  $N \leftarrow \text{int}(N_{\text{art}}(d, \theta, A))$
  $R \leftarrow R_{\text{art}}(d, \theta)$
  $X \leftarrow X_{\text{art}}(d, \theta)$
  $C_{\text{art}} \leftarrow \text{ConvertTo3D}(R, X)$ ▷ Convert to 3D point
  $\mathcal{F} \leftarrow \emptyset$ ▷ Initialize empty artifact cluster
  for $i = 1$ to N **do**
    Sample point $\mathbf{p}_i \sim \mathcal{N}(C_{\text{art}}, \Sigma_{\text{fixed}})$
    Add $\mathbf{p}_i$ to $\mathcal{F}$
  end for
  $\mathcal{L}_{\text{modified}} \leftarrow \mathcal{L}_{\text{native}} \cup \mathcal{F}$
else
  $\mathcal{L}_{\text{modified}} \leftarrow \mathcal{L}_{\text{native}}$
end if
return Modified LiDAR scan $\mathcal{L}_{\text{modified}}$

cloud as authentic sensor input, enabling the observation of unfiltered system reactions.

7.2 Scenario-Based AV Response Analysis

The system-level impact of our attack was evaluated in a critical urban driving scenario designed in CARLA. This scenario enables observation of the complete event chain, from initial perception failure to a physical collision, thereby providing a clear measure of the threat's severity.

Simulation Setup. Our primary scenario involves two vehicles approaching a T-intersection. The lead vehicle, which we term the "Ego AV," is equipped with our full attack injection pipeline and a standard autonomous driving agent (`BehaviorAgent` in CARLA). A second "Follower Vehicle" trails the Ego AV, governed by a basic car-following logic that maintains a safe distance based on its own perception. An adversarial mirror is placed at the corner of the T-intersection, positioned to trigger model-based artifact injection that simulates a phantom object in the Ego AV's path.

Qualitative Outcome. As the Ego AV approaches the mirror, the pipeline continuously evaluates the configuration state (d, θ, A) and determines whether artifact generation should occur. Once the appearance probability exceeds the stochastic threshold, a dense cluster of phantom points is injected into the Ego AV's LiDAR stream at a physically consistent location and depth. The perception module interprets this cluster as a nearby obstacle occupying the driving corridor, which propagates to tracking and planning as a potential collision risk. The safety layer therefore overrides the nominal trajectory and initiates emergency braking. The follower vehicle reacts only after observing the abrupt deceleration of the Ego AV and cannot brake in time, resulting in a rear-end collision (Figure 14).

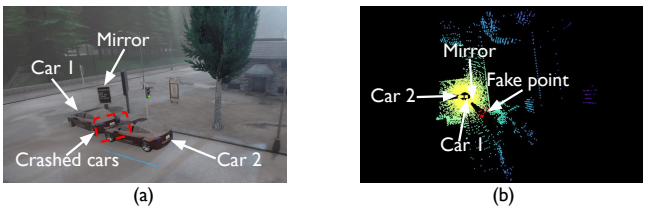


Figure 14: Cascade collision in CARLA. The Ego AV (Car 1) emergency-brakes for a phantom obstacle; the Follower (Car 2) rear-ends it. **(a)** Collision moment. **(b)** Ego AV's LiDAR view showing injected phantom points.

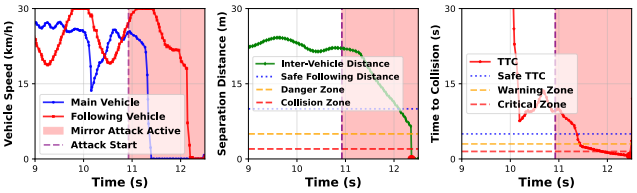


Figure 15: System-level dynamics following artifact injection at $t \approx 11.0$ s. Ego AV emergency brakes; Follower Vehicle fails to react, leading to collision (separation and TTC reach zero).

Quantitative Analysis. Figure 15 presents time-aligned vehicle dynamics during a representative scenario. At $t \approx 11.0$ s, artifact injection begins. The Ego AV (blue) decelerates from >25 km/h to zero within $\sim$1 s. The Follower (red) continues at speed before reacting with a delay, causing the separation distance to collapse to zero. The Time-to-Collision (TTC) drops below the critical 3-second threshold and ultimately reaches zero, confirming a collision. This causal chain, perception deception $\rightarrow$ emergency braking $\rightarrow$ rear-end collision, demonstrates how a passive optical manipulation at the sensor layer can cascade into a catastrophic system-level failure.

Point Count and Attack Success. Table 2 summarizes the relationship between mirror configuration, phantom point count, perception output, and system-level behavior across both the real-world Autoware experiments (Section 5) and the closed-loop CARLA evaluation presented above. The three mirror areas span roughly an order of magnitude in artifact density (from $\approx$25 to $\approx$200 points per frame), yet all configurations produce the same system-level outcome: emergency braking followed by rear-end collision in simulation, and planner-level obstacle response in the real-world perception traces. This reveals a two-tier threshold structure. Dense clusters ($\gtrsim$50 points) drive high-confidence semantic misclassification as "Vehicle" (65% and 74% confidence), whereas sparse clusters ($\approx$25 points) are labeled "UNKNOWN" but still populate occupancy cells and trigger defensive planner behavior. The operationally relevant threshold for attack success is therefore the occupancy-grid threshold, not the semantic-classification threshold, and it is met even by the sparsest configurations we

Area (tiles)	Pts/Frame	Perception	Outcome
0.18 m^2 (2)	≈25	UNKNOWN	Brake + Crash
0.36 m^2 (4)	≈55	CAR (65%)	Brake + Crash
0.60 m^2 (6)	≈200	CAR (74%)	Brake + Crash

Table 2: Point count and attack outcomes across mirror configurations, real-world perception (Section 5), and CARLA closed-loop behavior.

tested. This indicates that the attack does not require finely tuned geometry or large mirrors to produce unsafe system-level behavior.

Speed sensitivity. Since varying vehicle speed in our real-world test environment was not feasible for safety reasons, we used CARLA to evaluate the vulnerability at realistic urban approach speeds of 25, 30, and 35 km/h. The injected phantom triggered emergency braking and a rear-end collision at all three speeds, with collision severity increasing at higher velocities due to greater closing momentum and shorter available braking margins.

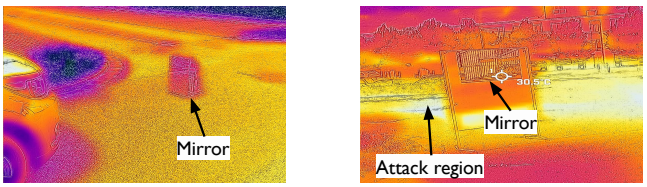
8 Discussion and Countermeasures

Our findings show that mirror-induced artifacts exploit fundamental assumptions in LiDAR perception, creating serious safety risks for autonomous systems. This section examines potential defense strategies and discusses their limitations.

8.1 Defense Strategies

① **Thermal Camera Augmentation.** The fundamental principle of this defense is to leverage a different sensing modality that is not susceptible to the same physical deception. Thermal (long-wave infrared) cameras detect emitted heat rather than reflected light. In OA, the phantom object is a LiDAR-specific illusion that lacks physical substance and therefore has no corresponding heat source. A thermal camera observing the phantom's perceived location would see only the ambient background temperature, directly falsifying the LiDAR detection. In OR, the mirror itself becomes the detectable anomaly. Its surface typically presents a thermal signature distinct from the surrounding road, creating a clear thermal discontinuity that reveals the presence of the adversarial element. In both cases, the thermal modality provides critical information to either invalidate a phantom or detect the tool of deception.

Implementation and Evaluation. We integrated a FLIR One thermal camera into our test vehicle and developed a detection module to identify anomalous thermal regions. The camera proved effective in both scenarios (see Figure 17). In the OR case (**a**), the mirror panel is clearly delineated as a cool (purple) rectangular surface against the warmer (yellow) asphalt, creating a distinct thermal boundary that allows for direct detection of the adversarial element. In the OA scenario (**b**), the thermal image shows that the region where LiDAR detected



(a) (b)

Figure 16: Thermal imagery of mirror attack scenarios. **(a)** OR: The mirror is visible as a cool (purple) surface against the warmer (yellow) road. **(b)** OA: The phantom object region shows no thermal anomaly, matching the ground temperature and confirming it is not a real object.

a phantom vehicle is thermally uniform with its surroundings. The absence of any localized heat source different from the ground plane confirms the object's non-physical nature.

Limitations. Despite its promise, thermal imaging is not a panacea. Its efficacy can be reduced by environmental factors, such as high ambient temperatures that minimize the thermal contrast between objects (thermal crossover). The resolution of thermal cameras is typically lower than that of RGB cameras, making it difficult to detect small or distant mirrors. Furthermore, practical deployment involves challenges such as cost, calibration, and maintenance, as lenses can be obscured by rain or dirt.

② **Multi-Sensor Fusion.** Another potential defense strategy involves leveraging the inherent redundancy of an autonomous vehicle's multi-modal sensor suite. Standard sensor fusion algorithms typically operate on the principle of data reinforcement, assuming that different sensors will provide consistent, corroborating evidence about the state of the world. Mirror-based attacks are designed to violate this assumption by creating a fundamental *disagreement* between modalities. An OA creates a phantom visible only to LiDAR, while an OR can make a real object invisible only to LiDAR. A defense can therefore be built around a "fusion consistency monitor" designed to detect and flag these specific inter-sensor conflicts as anomalies.

Conceptual Implementation and Mitigation. This defense would be implemented as a logic layer that continuously cross-verifies detections between LiDAR and other sensors, such as RGB cameras or radar. Aligned with proposals such as that of [15], the system would monitor for high-confidence inconsistencies. For instance, in an OA scenario, a LiDAR-detected object that lacks a corresponding visual object in the camera feed would be flagged as anomalous. Conversely, in an OR scenario, a visually-detected obstacle that corresponds to a region of navigable free space in the LiDAR data would trigger a similar alert. Upon detecting such a conflict, the system could respond by significantly lowering the confidence score of the suspect perception, or by triggering a safe-state behavior, such as reducing speed, until the conflict is resolved by subsequent sensor readings.

Limitations and Challenges. The primary challenge of this approach lies in robustly disambiguating malicious inconsistencies from benign, modality-specific sensor failures. A naively implemented consistency check could generate high-risk false alarms. For example, in low-light conditions, a camera will naturally fail to detect a real obstacle that is clearly visible to LiDAR; a simple check would incorrectly discard this valid detection as a phantom. Similarly, benign physical phenomena, such as a retro-reflective road sign being highly salient to a camera but producing sparse returns for LiDAR, can also create disagreement. Therefore, developing a sophisticated consistency model that accounts for varying environmental conditions and the unique operational characteristics of each sensor without introducing an unacceptable rate of false positives is a non-trivial research challenge. Other advanced concepts, such as using redundant LiDARs at different physical locations to break the attack's geometric constraints, also warrant future exploration.

③ **Intensity-Based Anomaly Detection.** A recent line of work leverages LiDAR return intensity to detect physical adversarial attacks, observing that injected points often exhibit abnormal intensity relative to genuine returns [17]. This defense is effective against active laser spoofing, where received power is set by the attacker's emitter and does not match the expected intensity–range relationship. Mirror artifacts, however, are the vehicle's own pulses following a legitimate two-hop path: the received power follows Equation 8 applied to the total path length $d_{LM} + d_{MS}$, which is also the range the sensor reports. Intensity and range are therefore physically consistent by construction. The sensor observes exactly the return it would expect from a real object at that range with the reflectivity of the secondary surface, leaving no radiometric anomaly to detect.

Empirical Evaluation. To test this quantitatively, we computed the distribution of calibrated Ouster reflectivity values (0 to 255 scale) for mirror-induced artifact points and legitimate returns from the same scans, across three representative configurations spanning our parameter grid: (45°, 0.36 m^2), (30°, 0.60 m^2), and (60°, 0.18 m^2). For each configuration, we compared ∼1 to 3 k artifact points against ∼3 M legitimate returns drawn scene-wide, excluding only the artifact region, since a per-point detector operates across the full scan. Table 3 reports the key detection metrics.

The detector operating in the Park direction, flagging high-reflectivity points as adversarial, performs worse than random, with AUC values of 0.12 to 0.17. This reflects a physical reality: the two-hop path attenuates the return through mirror reflectivity (∼0.95), the diffuse albedo of the secondary surface (∼0.01 to 0.10), and additional geometric spreading loss. Mirror artifacts are therefore systematically *dimmer* than typical direct returns, the opposite of active-spoofing injections. At a deployable 5% false-positive rate, the Park-style detector catches only 0.7 to 4.4% of artifact points. We also evaluated the inverted detector, flagging low-reflectivity points. While this captures the directional asymmetry of the data, it still fails at operationally relevant thresholds: catching 95% of artifacts requires flagging 52 to 86% of legitimate returns, because dark asphalt, shadowed regions, and low-albedo road surfaces populate the same low-reflectivity tail as mirror artifacts. Neither threshold direction yields a deployable operating point, confirming that intensity-based detection cannot address this class of vulnerability.

Config	μ_{art}	μ_{base}	AUC	TPR @ 5% FPR
(45°, 0.36 m^2)	3.1	12.9	0.12	0.7%
(30°, 0.60 m^2)	4.1	12.8	0.13	1.6%
(60°, 0.18 m^2)	10.4	12.7	0.17	4.4%

Table 3: Intensity-based detection performance against mirror artifacts. AUC values well below 0.5 indicate that the Park-direction detector (flag high-reflectivity outliers) is anti-correlated with the attack; artifacts are dimmer than legitimate returns. At a deployable 5% false-positive budget, the detector catches essentially none of the attack.

8.2 Broader Implications

8.2.1 Generality Across LiDAR Platforms

We replicated the Object Addition experiment on four additional LiDAR sensors to verify that the attack is not specific to our test platform. The sensors span three vendors and a 4× range in vertical resolution: Ouster OS1-64, Ouster OS0-64 (wide vertical FOV variant), Robosense RS-Helios 5515 (32 channels), and Hesai XT32M2X (32 channels). Each sensor was deployed in a static outdoor configuration at a representative mirror tilt of 45°, matching the geometry used in our main experiments. We did not integrate these sensors into the autonomous vehicle stack, as physical integration and per-sensor calibration are non-trivial. The goal is to confirm reproducibility of the underlying physical phenomenon rather than to repeat the full perception-and-planning analysis. Figure 17 compares the baseline scans without the mirror against the corresponding mirror-present scans for each sensor. Across all four platforms, the mirror induces a spatially coherent phantom cluster behind the mirror surface, consistent with the two-hop reflection mechanism described in Section 2. These results indicate that the phenomenon is not an artifact of a single sensor model, but rather a cross-platform vulnerability that can manifest across LiDAR configurations representative of those used in autonomous driving systems.

8.2.2 Geometric Trust as a Security Assumption

Our results show that the vulnerability does not stem from noise, learning errors, or injected signals, but from a deeper architectural assumption: LiDAR perception treats measured range as geometrically correct by construction. The autonomy stack verifies signal reception but never verifies whether

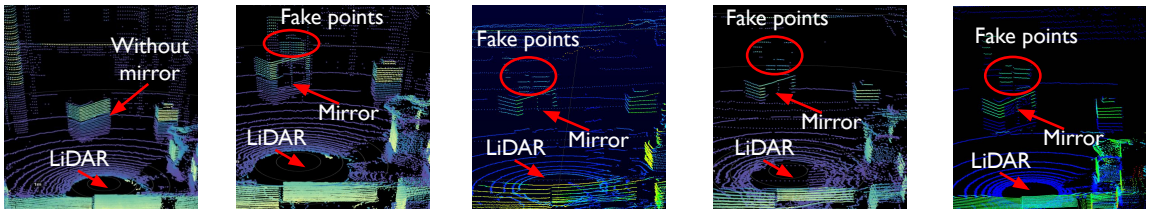


Figure 17: Object Addition across four LiDAR platforms. **(a)** Baseline (no mirror). **(b)** Ouster OS1-64. **(c)** Robosense RS-Helios 5515. **(d)** Ouster OS0-64. **(e)** Hesai XT32M2X.

the light propagation path is physically plausible. Specular reflections preserve valid timing and intensity statistics while altering spatial origin, producing observations that are internally consistent yet geometrically incorrect.

This distinguishes mirror-induced artifacts from traditional spoofing in a way that has direct defensive implications. Active attacks introduce foreign signals and can in principle be detected through authentication or waveform analysis. Mirror artifacts, by contrast, are the sensor's own photons returning through a physically valid but geometrically unexpected path. The deception is entirely in the propagation domain, not the signal domain. Consequently, no signal-level defense can address this class of vulnerability. Effective countermeasures must instead reason about *path plausibility*: whether the spatial configuration of observed returns is consistent with single-bounce propagation. This represents a verification layer that no current production LiDAR system or perception stack provides.

8.2.3 Implications for AV Validation

Current AV validation practices rely heavily on simulation and curated datasets that implicitly assume direct line-of-sight propagation. Perception systems may therefore satisfy standard safety metrics while never encountering geometrically inconsistent measurements. The absence of such scenarios represents a systematic blind spot in testing methodology rather than an implementation flaw.

By introducing physically calibrated multi-path artifacts into simulation, our framework exposes failure modes that remain invisible under conventional evaluation. More broadly, this suggests that validating autonomous systems requires explicitly modeling sensing physics, including multi-path propagation and specular surface interactions, not only sensor accuracy or algorithmic performance.

9 Limitations and Future Work

Sensor Generality. Our experiments used a spinning ToF LiDAR (Ouster OS1-128). While the underlying specular reflection vulnerability applies to any ToF system, Frequency Modulated Continuous Wave (FMCW) LiDAR and sensors with full-waveform analysis capabilities may exhibit different susceptibility profiles. Validation across sensor technologies is an important next step.

Environmental Conditions. All tests were conducted in clear, dry weather. Rain, fog, and humidity can alter mirror reflectivity and beam propagation, potentially affecting artifact characteristics in ways our current models do not capture. Systematic investigation of weather effects is needed.

Vehicle Speed. Our real-world experiments were limited to $\leq$8 km/h for safety. At higher urban speeds (30–50 km/h), the reduced number of frames within the appearance window could affect tracking-module behavior. However, our models predict that the appearance window spans several meters of distance even at 60°, and safety-critical planners are designed to react to even brief high-confidence detections. Controlled high-speed testing remains a priority.

Defense Integration. The countermeasures discussed in Section 8 were evaluated at a conceptual or prototype level. Full integration into an operational AV stack with rigorous false-positive analysis is needed before they can be considered practical defenses.

10 Related Work

Active LiDAR Spoofing. Most known LiDAR spoofing attacks use external laser emitters to inject fake pulses, creating phantom objects at arbitrary distances [2, 5, 7, 10, 18, 19]. These attacks are effective but require specialized hardware, line-of-sight access, and often precise synchronization with the victim sensor's timing. Our work demonstrates that comparable perception failures can be induced passively, without any electronic emissions.

Passive Physical-World Attacks. Passive approaches have used 3D-printed adversarial objects [1, 24] or crafted road patterns [27] to exploit learned perception models. These target model-specific vulnerabilities rather than the fundamental sensing modality. The "Shadow Hack" work [11, 12] demonstrated point removal using reflective mats but did not explore object addition or derive predictive models. Other work examined adversarial materials on road signs [31] or used drones for occlusion [32]. Our contribution differs in providing a comprehensive analysis of both addition and removal, grounded in geometric optics modeling, with end-to-end AV

stack evaluation.

LiDAR Artifacts and Reflective Surfaces. The robotics community has long recognized challenges from reflective surfaces in LiDAR mapping [3, 23, 28], and recent work has proposed SLAM algorithms robust to such artifacts [9, 13, 25]. However, these studies treat reflections as incidental noise rather than adversarial threats, and none provide the empirical models necessary for systematic evaluation.

Defenses and Sensor Fusion. Defenses against LiDAR spoofing include statistical filtering [8, 22], signal-level authentication, and multi-sensor fusion [1, 6, 15]. These approaches assume that spoofed signals differ from legitimate returns in detectable ways. Mirror-based artifacts challenge this assumption: they are generated from the vehicle's own pulses and produce physically legitimate returns, making them fundamentally harder to distinguish from benign reflections.

11 Conclusion

This work presents the first systematic vulnerability analysis of LiDAR-based autonomous driving systems against mirror-induced perception failures. Through controlled real-world experiments, we demonstrate that inexpensive planar mirrors can corrupt LiDAR point clouds to create phantom obstacles or erase real ones. We trace these artifacts through the Autoware autonomy pipeline, showing they trigger high-confidence false detections (up to 74% "Vehicle" confidence), corrupt occupancy maps, and cause safety-critical failures including emergency braking and cascade collisions.

Our physics-informed empirical models capture the geometric and radiometric properties of mirror-induced artifacts with high fidelity, enabling their integration into a CARLA-based simulation framework that addresses a gap in current simulators' inability to model specular reflection phenomena. Both the real-world validation and simulation framework provide a comprehensive methodology and dataset to support the development of reflection-aware perception systems.

The vulnerability we characterize is particularly challenging to address because mirror-based artifacts are generated from the vehicle's own signals, making them physically indistinguishable from legitimate returns at the signal level. Our analysis of potential countermeasures, including thermal imaging and multi-sensor fusion, reveals that each faces significant practical limitations. Developing robust defenses against physically legitimate but geometrically deceptive sensor inputs remains a critical open challenge for autonomous vehicle safety.

Acknowledgments

This work was supported in part by the DePhaSe-CPS and CORTESE projects, and by the CYPHER-AV project funded by the German Federal Ministry of Transport. The authors also acknowledge the FASTer project (EFRE-HS-0000774) and the htw saar ITS Research Group, and thank Aymen Bouferroum and Jishitha Kondaveti for their support.

References

[1] Yulong Cao, Ningfei Wang, Chaowei Xiao, Dawei Yang, Jin Fang, Ruigang Yang, Qi Alfred Chen, Mingyan Liu, and Bo Li. Invisible for both camera and lidar: Security of multi-sensor fusion based perception in autonomous driving under physical-world attacks. In *2021 IEEE symposium on security and privacy (SP)*, pages 176–194. IEEE, 2021.

[2] Yulong Cao, Chaowei Xiao, Benjamin Cyr, Yimeng Zhou, Won Park, Sara Rampazzi, Qi Alfred Chen, Kevin Fu, and Z Morley Mao. Adversarial sensor attack on lidar-based perception in autonomous driving. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 2267–2281, 2019.

[3] Deeptha Damodaran, Saeed Mozaffari, Shahpour Alirezaee, and Mohammed Jalal Ahamed. Experimental analysis of the behavior of mirror-like objects in lidar-based robot navigation. *Applied Sciences*, 13(5):2908, 2023.

[4] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[5] Masashi Fukunaga and Takeshi Sugawara. Random spoofing attack against lidarbased scan matching slam. *VehicleSec2024*, 2024.

[6] R Spencer Hallyburton, Yupei Liu, Yulong Cao, Z Morley Mao, and Miroslav Pajic. Security analysis of {Camera-LiDAR} fusion against {Black-Box} attacks on autonomous vehicles. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1903–1920, 2022.

[7] Zhongyuan Hau, Kenneth T Co, Soteris Demetriou, and Emil C Lupu. Object removal attacks on lidar-based 3d object detectors. *arXiv preprint arXiv:2102.03722*, 2021.

[8] Zhongyuan Hau, Soteris Demetriou, Luis Muñoz-González, and Emil C Lupu. Shadow-catcher: Looking into shadows to detect ghost objects in autonomous vehicle 3d sensing. In *Computer Security–ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part I 26*, pages 691–711. Springer, 2021.

[9] Connor Henley, Siddharth Somasundaram, Joseph Hollmann, and Ramesh Raskar. Detection and mapping of specular surfaces using multibounce lidar returns. *Optics Express*, 31(4):6370–6388, 2023.

[10] Zizhi Jin, Xiaoyu Ji, Yushi Cheng, Bo Yang, Chen Yan, and Wenyuan Xu. Pla-lidar: Physical laser attacks against lidar-based 3d object detection in autonomous vehicle. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1822–1839. IEEE, 2023.

[11] Ryunosuke Kobayashi, Kazuki Nomoto, Yuna Tanaka, Go Tsuruoka, and Tatsuya Mori. Wip: Shadow hack: Adversarial shadow attack against lidar object detection. In *Proc. Symp. Veh. Security Privacy (VehicleSec)*, pages 1–7, 2024.

[12] Ryunosuke Kobayashi, Kazuki Nomoto, Yuna Tanaka, Go Tsuruoka, and Tatsuya Mori. Invisible but detected: physical adversarial shadow attack and defense on lidar object detection. In *Proceedings of the 34th USENIX Conference on Security Symposium*, SEC '25, USA, 2025. USENIX Association.

[13] Rainer Koch, Stefan May, and Andreas Nüchter. Detection and purging of specular reflective and transparent object influences in 3d range measurements. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 42:377–384, 2017.

[14] Pui Yee Leong and Nur Syazreen Ahmad. Lidar-based obstacle avoidance with autonomous vehicles: A comprehensive review. *IEEE Access*, 2024.

[15] Jinshan Liu and Jung-Min Park. "seeing is not always believing": detecting perception error attacks against autonomous vehicles. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2209–2223, 2021.

[16] Ouster, Inc. OS1 Mid-Range Lidar Sensor. https://ouster.com/products/hardware/os1-lidar-sensor. Accessed: 2026-02-23.

[17] Yeji Park, Hyunsu Cho, Dong Hoon Lee, and Wonsuk Choi. Leveraging intensity as a new feature to detect physical adversarial attacks against lidars. In *2024 Annual Computer Security Applications Conference (ACSAC)*, pages 1002–1014. IEEE, 2024.

[18] Jonathan Petit, Bas Stottelaar, Michael Feiri, and Frank Kargl. Remote attacks on automated vehicles sensors: Experiments on camera and lidar. *Black Hat Europe*, 11(2015):995, 2015.

[19] Takami Sato, Yuki Hayakawa, Ryo Suzuki, Yohsuke Shiiki, Kentaro Yoshioka, and Qi Alfred Chen. LiDAR spoofing meets the new-gen: Capability improvements, broken assumptions, and new attack strategies. *arXiv preprint arXiv:2303.10555*, 2023.

[20] Takami Sato, Ryo Suzuki, Yuki Hayakawa, Kazuma Ikeda, Ozora Sako, Rokuto Nagata, Ryo Yoshida, Qi Alfred Chen, and Kentaro Yoshioka. On the realism of lidar spoofing attacks against autonomous driving vehicle at high speed and long distance. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2025.

[21] Hocheol Shin, Dohyun Kim, Yujin Kwon, and Yongdae Kim. Illusion and dazzle: Adversarial optical channel exploits against lidars for automotive applications. In *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 445–467. Springer, 2017.

[22] Jiachen Sun, Yulong Cao, Qi Alfred Chen, and Z Morley Mao. Towards robust {LiDAR-based} perception in autonomous driving: General black-box adversarial sensor attack and countermeasures. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 877–894, 2020.

[23] Haileleol Tibebu, Jamie Roche, Varuna De Silva, and Ahmet Kondoz. Lidar-based glass detection for improved occupancy grid mapping. *Sensors*, 21(7):2263, 2021.

[24] James Tu, Mengye Ren, Sivabalan Manivasagam, Ming Liang, Bin Yang, Richard Du, Frank Cheng, and Raquel Urtasun. Physically realizable adversarial examples for lidar object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13716–13725, 2020.

[25] Miguel A Vega-Torres, Alexander Braun, and André Borrmann. Slam2ref: advancing long-term mapping with 3d lidar and reference map integration for precise 6-dof trajectory estimation and map extension. *Construction Robotics*, 8(2):13, 2024.

[26] Jian Wang, Fan Li, Xuchong Zhang, and Hongbin Sun. Adversarial obstacle generation against lidar-based 3d object detection. *IEEE Transactions on Multimedia*, 2023.

[27] Kaichen Yang, Tzungyu Tsai, Honggang Yu, Max Panoff, Tsung-Yi Ho, and Yier Jin. Robust roadside physical adversarial attack against deep learning in lidar perception modules. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 349–362, 2021.

[28] Shih-Wei Yang and Chih-Chung Wang. On Solving Mirror Reflection in LIDAR Sensing. *IEEE/ASME Transactions on Mechatronics*, 16(2):255–265, 2011.

[29] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Centerpoint: Center-based 3d object detection and tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11784–11793, 2021.

[30] Yongjun Zhang, Pengcheng Shi, and Jiayuan Li. Lidar-based place recognition for autonomous driving: A survey. *ACM Computing Surveys*, 57(4):1–36, 2024.

[31] Shenchen Zhu, Yue Zhao, Kai Chen, Bo Wang, Hualong Ma, and Cheng'an Wei. {AE-Morpher}: Improve physical robustness of adversarial objects against {LiDAR-based} detectors via object reconstruction. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 7339–7356, 2024.

[32] Yi Zhu, Chenglin Miao, Tianhang Zheng, Foad Hajiaghajani, Lu Su, and Chunming Qiao. Can we use arbitrary objects to attack lidar perception in autonomous driving? In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1945–1960, 2021.

A Empirical Model Validation and Parameter Analysis

This appendix summarizes the empirical model parameters used in this study. Table 4 reports both the fitted values and their compact physical interpretation for the four empirical models. The coefficients were obtained through global nonlinear least-squares fitting across all relevant experimental configurations and were used to generate the "Fitted Model" curves in our results.

A.1 Validation Across Experimental Configurations

❶ **Artifact Point Count (N_{art}).** The Gaussian-like model captures the rise-and-fall behavior of phantom point counts across configurations, with R^2 values between 0.70 and 1.00 (Figure 18). Point count increases as the mirror subtends a larger solid angle, peaks at favorable alignment, and then decreases as the reflection geometry degrades. Mirror area is the main driver of artifact magnitude, while larger tilt angles reduce point density through foreshortening and less efficient reflection paths.

❷ **Appearance Probability (P_{app}).** The double-sigmoid model fits the observed appearance windows with high accuracy, with R^2 values above 0.87 and often close to 1.00 (Figure 19). The results confirm that artifacts appear only within a finite distance interval bounded by $d_{\min}$ and $d_{\max}$.

Table 4: Empirical model parameters.

Model	Param.	Value	Meaning
Point count N_{art} $R^2\approx0.85$	c_0	2500.0	Max. point-count scale
	β	1.25	Mirror-area sensitivity
	γ	3.5	Tilt sensitivity
	μ	3.0 m	Peak-response distance
	σ	1.5 m	Effective range width
Appearance P_{app}	k	15.0	Window sharpness
	$b_0^{\min}$	-2.858	Lower-bound offset
	b_1, c_1	0.1389	Angle-boundary shift
	b_2	-0.3003	Lower-bound area effect
	$b_0^{\max}$	-0.946	Upper-bound offset
	c_2	1.5390	Upper-bound area effect
Lateral X_{art} $R^2\approx0.98$	c_X	0.98	Geometry scale
	δ_X	-0.05 m	Setup bias
Radial R_{art} $R^2\approx0.98$	c_R	1.02	Distance scale
	n_d	0.88	Growth exponent
	a_1	0.15	Linear angle correction
	a_2	0.08	Quadratic angle correction

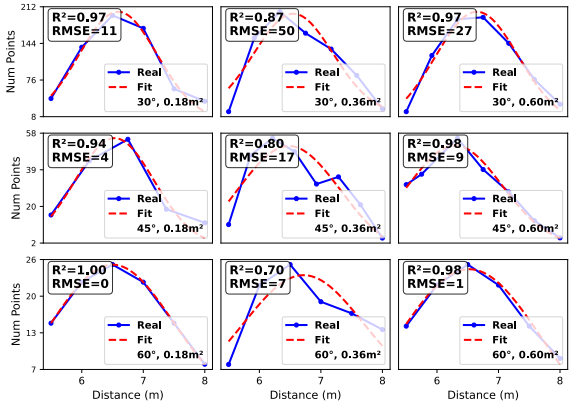


Figure 18: Artifact point count: experimental data ("Real", blue) vs. model predictions ("Fit", red dashed) across all 9 configurations ($\theta \times A$).

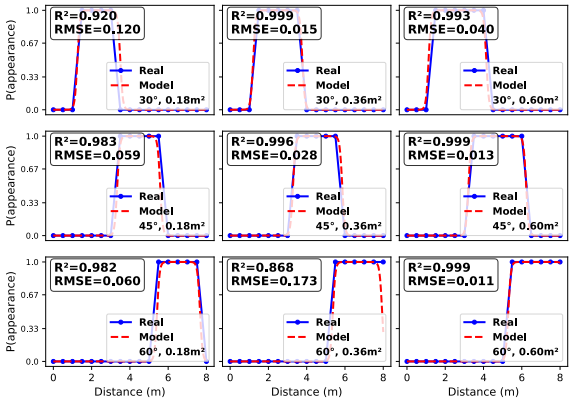


Figure 19: Appearance probability: experimental data ("Real") vs. model predictions ("Model") across all 9 configurations.

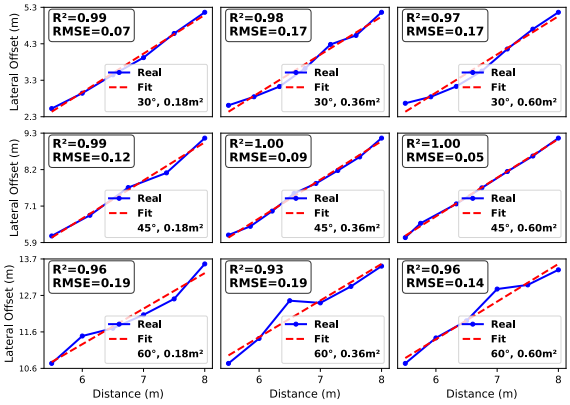


Figure 20: Lateral offset: experimental data ("Real") vs. model predictions ("Fit") across all 9 configurations.

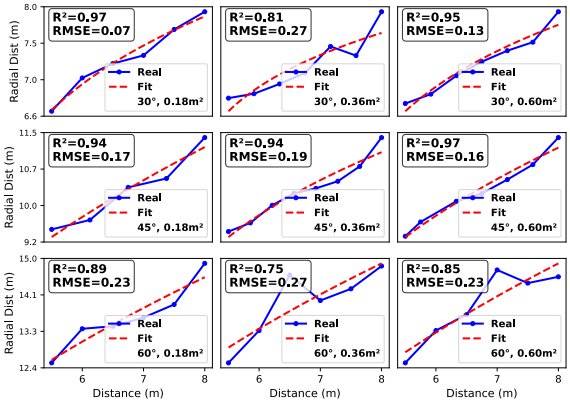


Figure 21: Radial distance: experimental data ("Real") vs. model predictions ("Fit") across all 9 configurations.

Larger mirror areas widen this interval, whereas tilt primarily shifts it along the distance axis.

❸ **Lateral Offset (X_{art}).** The linear model achieves near-perfect agreement, with $R^2 \geq 0.93$ across configurations (Figure 20). The offset scales linearly with distance through the $\tan(2\theta)$ term. Thus, mirror tilt governs phantom placement, while mirror area mainly affects cluster density rather than location.

❹ **Radial Distance (R_{art}).** The power-law model provides strong fits, with R^2 values mostly above 0.85 (Figure 21). The perceived radial distance grows nearly linearly with the true distance, while the d^{n_d} term captures sub-linear effects due to beam spillover. Tilt changes the reflected path length, whereas mirror area has little influence on perceived distance.

B Supplementary Point Cloud Visualizations

This appendix presents supplementary visualizations of the raw LiDAR point clouds generated under different OA and OR configurations. These provide direct visual evidence of how physical parameters influence raw sensor data.

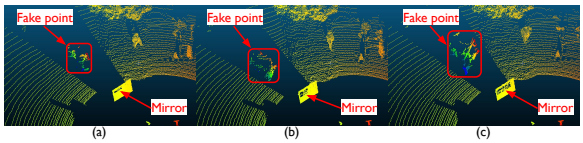


Figure 22: Impact of mirror surface area on phantom point cloud: **(a)** Two tiles (0.18 m^2); **(b)** Four tiles (0.36 m^2); **(c)** Six tiles (0.60 m^2).

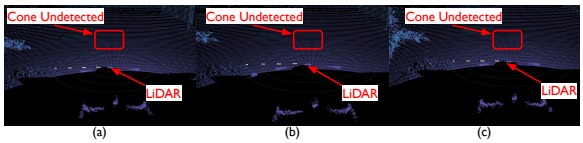


Figure 23: LiDAR point clouds under OR at various mirror angles. **(a)** 15°: cone fully invisible. **(b)** 30°: cone fully invisible. **(c)** 45°: cone fully invisible. The consistency across angles demonstrates a wide operational envelope.

B.1 OA: Phantom Point Cloud Characteristics

① **Impact of Mirror Surface Area.** Figure 22 shows that larger mirror areas produce denser and more coherent phantom clusters. With 2 tiles (0.18 m^2), the cluster survives filtering and is labeled "UNKNOWN." With 4 tiles (0.36 m^2), it gains sufficient volumetric structure to be classified as "CAR" with 65% confidence, increasing to 74% with 6 tiles (0.60 m^2). Thus, reflective area directly controls deception severity, from a low-level anomaly to confident semantic misclassification.

② **Impact of Mirror Tilt Angle.** Mirror tilt controls both the perceived location and temporal stability of the phantom. At 30°, the restrictive geometry yields a late-onset phantom visible only at close range, while 60° creates a persistent barrier that appears earlier and remains stable over a longer trajectory. The 45° case shows stronger frame-to-frame flicker, indicating that tilt can tune not only phantom position but also temporal consistency.

B.2 OR: Robustness Across Angles

Figure 23 shows that OR remains effective across mirror angles of 15°, 30°, and 45°. In all cases, the traffic cone disappears from the LiDAR scan while the deflected beams generate valid-looking ground returns, making the obstacle region appear navigable. This consistency indicates that OR does not require precise alignment and can operate over a broad range of practical configurations.



Short: Investigating Stealthy Latency Attacks Against Camera-Based Object Detection Systems

Lichen Xia, Xing Gao, Weisong Shi
Department of Computer and Information Sciences
University of Delaware, Newark, DE 19716, USA
{lxia, xgao, weisong}@udel.edu

Abstract

Object detection (OD) systems are fundamental to safety-critical applications such as autonomous driving and indoor mobile robots, yet remain vulnerable to latency-based attacks that degrade real-time performance. Existing attacks suffer from computational inefficiency and poor stealthiness, as phantom object generation disrupts original detections, making attacks easily identifiable. We propose Groundswell, a novel latency attack employing Regional Perturbation Balance (RPB) to inject adversarial phantom objects while preserving original detections. By leveraging spatial object distribution patterns, RPB generates masks that guide phantom placement into regions with minimal interference, enabling efficient optimization via Projected Gradient Descent. Evaluations across multiple OD architectures demonstrate that Groundswell reduces resource utilization by 50% and training time by 80% over existing methods, while successfully inducing latency delays that exceed real-time thresholds for safety-critical systems.

1 Introduction

Deep learning-based object detection (OD) systems, including the YOLO series [11], have achieved outstanding detection precision and real-time efficiency, making them essential in safety-critical domains such as autonomous vehicles (AVs) where real-time perception is crucial. Modern OD architectures including Faster R-CNN [21], SSD [16], and YOLO [9] rely on Non-Maximum Suppression (NMS) as a post-processing step to remove redundant predictions.

Adversarial attacks deceive machine learning systems through carefully crafted input perturbations, and their security implications have been extensively studied [4, 23, 27]. However, these traditional attacks primarily target model integrity by causing misclassifications or missed detections. Recent work has proposed latency-based attacks against OD system availability [17, 19, 24, 28], focusing on increasing computational execution time rather than compromising model integrity. Current methods [3, 15, 19, 22] amplify detected object quantities through phantom object generation, exploiting the positive correlation between NMS processing time and detection count. Such attacks pose considerable dangers to time-critical applications, particularly Level 4 and Level 5 autonomous driving systems that require maintaining perception output generation under 30 milliseconds [5].

Unfortunately, existing latency attacks suffer from critical limitations hindering practical deployment. First, phantom objects can disrupt original detections, making the attack easily noticeable. Second, approaches that attempt to preserve original detections introduce additional loss functions requiring pairwise computation between detected objects, incurring significant computational overhead.

We introduce Groundswell, a latency attack employing a novel framework called Regional Perturbation Balance (RPB) to preserve original detection performance while maximizing latency impact. Exploiting the observation that objects tend to concentrate in specific spatial regions across sensor inputs, Groundswell crafts a universal adversarial perturbation (UAP) via Projected Gradient Descent (PGD). RPB learns these spatial distribution patterns to generate a mask identifying suitable regions for phantom object insertion, maximizing NMS computational overhead while minimizing interference with legitimate detections.

We evaluate Groundswell across varying configurations and diverse OD architectures. The results demonstrate a strong balance between effectiveness and stealthiness, with 50% less resource utilization during training and 80% faster optimization than existing work, making it significantly more practical for real-world deployment.

Our main contributions are outlined as follows:

- We propose a novel latency attack with RPB, which directs phantom object placement into image regions with minimal impact on original detections, achieving a favorable trade-off between attack effectiveness and stealthiness while improving optimization efficiency.
- We perform extensive experiments across diverse config-

urations and target object detection architectures, demonstrating our methodology substantially improves optimization efficiency compared to existing approaches.

- We establish the effectiveness of our approach across various OD architectures, highlighting its potential as a significant threat.

2 Background

2.1 Perception Pipeline

A typical perception pipeline consists of object detection and Non-Maximum Suppression (NMS). Object detection extends beyond image classification by simultaneously localizing and recognizing multiple objects, with deep learning approaches falling into two categories: two-stage detectors (e.g., Faster R-CNN [21], Mask R-CNN [7]) that separate region proposal and classification, and one-stage detectors (e.g., SSD [16], YOLO [9]) that unify both tasks in a single network. The pipeline operates in three stages: (1) preprocessing, where sensor inputs are resized, normalized, and augmented; (2) inference, where the CNN generates bounding boxes with class probabilities; and (3) post-processing, where NMS eliminates redundant overlapping boxes, retaining only the highest-confidence detection per object.

2.2 Non-maximum Suppression

NMS is a critical post-processing component in modern OD systems. A recent survey shows that 11 out of 12 OD systems employ NMS for refining their results [14]. Since detection models generate numerous candidate bounding boxes, each containing position, dimensions, objectness confidence, and class probabilities, NMS consolidates overlapping predictions corresponding to the same physical object.

The execution of the most common NMS algorithm (i.e., vanilla NMS [10]) can be decomposed into three distinct stages. During the initial stage, NMS processes the candidate set C, which is the output of the second phase of OD models, and eliminates entries with confidence scores falling below the threshold τ_{conf}:

$$C_1 = \{c_{obj} > \tau_{conf} | c \in C\} \tag{1}$$

where $c \in C$ is a single candidate included in the candidate set C, and the objectness score c_{obj} represents the model's confidence in the presence of an object.

Subsequently, candidates with insufficient class confidence are also eliminated:

$$C_2 = \{c_{obj} \cdot \max\{c_{cls,i}\}_{i=0}^{N} > \tau_{conf} | c \in C_1\} \tag{2}$$

where the class confidence score $c_{cls,i}$ represents the classification certainty for each class i from 0 to N. The class with the highest class confidence score is assigned as the class label for the candidate c.

In the second phase, filtered candidates C_2 are sorted by confidence score c_{obj}, then offset by class to group detections into separate coordinate spaces. This prevents cross-class detections from being incorrectly merged.

Finally, starting from the highest-confidence detection, NMS iteratively computes the Intersection over Union (IoU) between each box and all remaining candidates. Boxes of the same class exceeding a specified IoU threshold (e.g., $\tau_{IoU} = 0.5$) are suppressed as redundant, retaining only the highest-confidence detection per object. This continues until no redundant detections remain. In the worst case where no candidates are removed in phase one, the time complexity of the NMS reaches $O(|C|^2)$.

The strong dependency of NMS's execution time on the number of detections makes it vulnerable for latency-based attacks against object detection systems.

3 Latency attacks and limitations

Sponge examples [24] first revealed that vision systems could be manipulated to consume excessive computational resources. Subsequent latency attacks on OD systems targeted preprocessing [28] and postprocessing [15, 17] stages. For NMS-targeted attacks, Daedalus [26] introduced phantom objects to increase execution latency, but produced non-transferable patches, limiting practicality. Shapira et al. [22] proposed universal adversarial patches that flood NMS with fake detections while minimizing overlap with original detections, but at high computational cost during optimization. Chen et al. [3] improved efficiency by inserting synthetic objects in sparse image regions, though still requiring per-image patch generation. Muller et al. [19] extended digital attacks to physical settings via zone stitching, but without stealthiness mechanisms. Our work addresses these limitations with a framework that efficiently optimizes attack effectiveness and stealthiness for practical real-world deployment.

4 Threat Model

Following prior work [3, 19, 22], we assume a white-box attacker with full knowledge of the target OD model. When the victim runs a publicly documented stack (e.g., Autoware with YOLOv5), the attacker can simply download the same public weights without any privileged access. The proliferation of model-sharing platforms further reinforces this assumption: Hugging Face alone hosts over two million public models, making it straightforward for any attacker to obtain the exact weights of widely deployed perception models [8]. As shown in Figure 1, the optimized UAP can be delivered physically by affixing it as a sticker on the camera lens [4] or by projecting it onto the attack surface in front of the victim vehicle using a

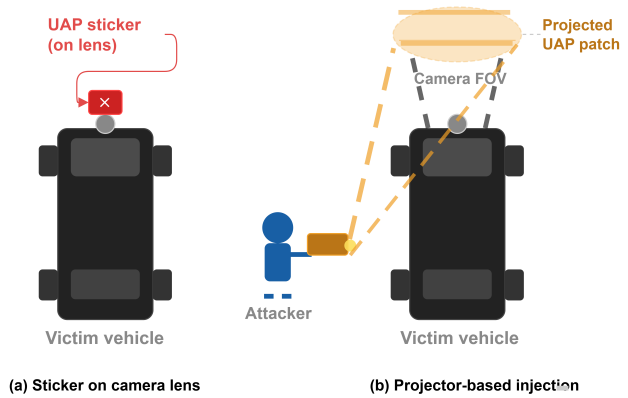


Figure 1: To effect perturbations, an attacker can either (a) affix a UAP sticker directly over the onboard camera lens [4], or (b) project the perturbation onto the surface in front of the victim vehicle using a portable projector [19].

portable projector [19]. Both methods inject the perturbation into raw frames before any in-vehicle processing occurs, requiring neither software access nor network interaction. The sticker approach is passive and persistent. The attacker only needs a single physical access to the vehicle and is not required to be present during the attack. The projector approach instead requires the attacker to maintain proximity to the victim, but allows real-time adjustment of the perturbation as driving conditions change.

The primary objective of the attacker is to identify a UAP δ that can be applied to input sequences (X) in real-time in order to maximize the total processing time (T_{OD}) of the target OD model. Additionally, to maintain attack stealthiness, the attacker aims to preserve the detection outputs from the original clean images. This dual objective can be formulated as the following optimization problem:

$$\max_{\|\delta\|_p \leq \varepsilon} \sum_{x \in X} T_{\text{OD}}(x+\delta) + \lambda_{stealth}(D(f(x+\delta), f(x)) \quad (3)$$

where the perturbation δ is bounded by an ε-ball under a given norm p. Larger ε values produce more visible perturbations, compromising attack stealthiness. Here, f represents the target OD model, while D measures the preservation of original detections. The parameter $\lambda_{stealth}$ controls the trade-off between maximizing latency and maintaining stealthiness of the attack.

5 Groundswell Attack

We propose the Groundswell attack that aims to maximize the attack effectiveness while preserving stealthiness by identifying image regions suitable for adversarial object insertion with minimal impact on original detections. The proposed RPB framework learns spatial distribution patterns of target sensors to generate masks that preserve original predictions. By combining with PGD [18], our method can generate an optimized UAP to enhance attack effectiveness while maintaining stealthiness.

5.1 Maximize the latency

As analyzed in Section 2.2, the NMS's third phase has the highest time complexity of $O(n^2)$, where n is the number of processed candidates. Thus, to induce latency, we attempt to generate a large number of phantom objects that require processing by the third phase of the NMS. However, many candidates may be discarded during the first phase of the NMS before reaching the third stage. To avoid being filtered out by the NMS's first phase, the injected objects must maintain confidence scores higher than the configured threshold τ_{conf} of the targeted OD models. Let C denote the candidate set output by the target OD model with unperturbed images, and C' denote the candidate set when inputs contain the UAP δ. For an individual proposal $c' \in C'$, the loss ℓ is defined as:

$$\ell(c', k) = max(0, \tau_{conf} - c'_{obj} \cdot c'_{cls,k}) \quad (4)$$

where k is the target class to which the adversary requires the proposal c' belongs. ℓ aims to minimize the gap between the confidence threshold τ_{conf} and the product of c'_{obj} and $c'_{cls,k}$. It encourages the proposal c' to be detected as a valid object with label k. If c' already qualifies as a valid object, the loss function applies no modification.

The overall loss function for maximizing the number of objects across an input image is formulated as:

$$\ell_{max\ obj} = \frac{1}{|C'|} \sum_{c' \in C'} \ell(c', k) \quad (5)$$

Bounding Box Area Loss. Minimizing the size of phantom object candidates can free up space for insertions, potentially enabling more qualified candidates to reach NMS processing. [22]. Thus, we further include a bounding box area loss:

$$\ell_{box\ area} = \frac{1}{|C'|} \sum_{c' \in C'} c'_w \cdot c'_h \quad (6)$$

where c'_w and c'_h are the width and height of the candidate c'. To craft the UAP, we minimize a loss function combining two components into a single equation:

$$\ell = \ell_{max\ obj} + \lambda \ell_{box\ area} \quad (7)$$

where λ is the weight of bounding box area loss $\ell_{box\ area}$.

5.2 Regional Perturbation Balance (RPB)

The secondary goal is attack stealthiness: the perturbed image x' should preserve as many original detections from clean

image x as possible. We find that images from specific sensors can exhibit certain patterns. For instance, for images collected by AV cameras, objects are more likely to accumulate in the central area of the images [2, 6]. The upper areas tend to be sky, while the bottom areas are typically roads. Consequently, perturbations applied to the image periphery have less impact on stealthiness than those in the central region, which we validate in Section 6.2.

Based on this observation, we propose RPB to efficiently balance attack effectiveness and stealthiness. An image is divided into three regions (upper, middle, bottom) by boundaries b_{upper} and b_{lower}, with perturbations applied only to the upper and bottom regions, where original detections (e.g., vehicles) are sparse.

The boundaries are dynamically adjusted via Algorithm 1 based on T_{ratio}, the target percentage of original objects the adversary wishes to preserve. At each optimization round, the ratio $r_{detection}$ of objects in the middle region to total detections in the clean image is computed (*line 5*). If $r_{detection} > T_{ratio}$, the middle region shrinks to allow more perturbation (*line 6-8*); if $r_{detection} < T_{ratio}$, it expands to preserve more original detections (*line 9-11*). The perturbation mask is then updated accordingly with the new boundaries (*line 14-16*).

Algorithm 1 Adaptive Boundary Update for Adversarial Patches

1: **procedure** SINGLE ROUND UPDATE
2: $T_{ratio} \leftarrow$ Target detection ratio
3: $C \leftarrow$ Set of candidates produced by the clean image
4: $s \leftarrow$ boundary adjustment step size
5: $r_{detection} \leftarrow detection_ratio(C, b_{upper}, b_{lower})$
6: **if** $r_{detection} > T_{ratio}$ **then** ▷ Too many detections in middle region
7: $b_{upper} \leftarrow b_{upper} + s$
8: $b_{lower} \leftarrow b_{lower} - s$
9: **else** ▷ Too few detections in middle region
10: $b_{upper} \leftarrow b_{upper} - s$
11: $b_{lower} \leftarrow b_{lower} + s$
12: **end if**
13: $*, height, width \leftarrow image.shape$
14: $mask \leftarrow$ zeros_matrix$(image.shape)$
15: $mask[0 : b_{upper}, *] \leftarrow 1$ ▷ Upper region for perturbation
16: $mask[b_{lower} : height, *] \leftarrow 1$ ▷ Bottom region for perturbation
17: **return** $mask, b_{upper}, b_{lower}$
18: **end procedure**

6 Evaluation

To evaluate Groundswell attack, we conduct comprehensive experiments across multiple YOLO detector versions (YOLOv5 [12], YOLOv8 [25], YOLOv11 [11]) pretrained on MS-COCO [13]. Our evaluation enables direct comparison with existing work (Phantom Sponges [22], Overload [3], DETSTORM [19]) while assessing transferability to advanced architectures.

6.1 Experimental setup

Dataset and Models We utilize the Berkeley Deep Drive (BDD) dataset [29], comprising 100,000 autonomous driving images with diverse environmental conditions, including varied weather (clear, rainy), settings (city streets, residential areas), and lighting conditions (daytime, nighttime). YOLOv5 serves as our primary model for baseline comparisons, while YOLOv8 and YOLOv11 assess attack performance on more advanced architectures. All models process images at 640×640 resolution using small-sized variants (YOLOv5s, YOLOv8s, YOLOv11s) for computational efficiency.

Attack Parameters. Perturbations are bounded using L_2 norm with $\varepsilon = 70$ across all evaluated attacks. Detection thresholds τ_{conf} and τ_{IoU} are set to 0.25 and 0.45, respectively, matching YOLO detectors' default settings. The target class is fixed to 2 with $\lambda = 10$ in equation 7, following the best practices from Phantom Sponges [22].

Implementation Detail. UAP training employs PGD with a learning rate of 0.0005, conducted over 100 epochs using a batch size of 8. All experiments run on NVIDIA RTX 3090 hardware.

Metrics. Our attack targets end-to-end latency in object detection models while maintaining stealthiness, which we evaluate using three metrics consistent with previous works [3, 19, 22]. These include:

1. $M' = |C_2|$: representing the number of candidates fed into the third phase of NMS. The more candidates NMS processes, the longer it should take.
2. Time: inference timing measurements comprising total end-to-end time (T_{total}[ms]) and NMS execution time (T_{NMS}[ms]). This metric measures the latency introduced to the NMS and the overall OD model, providing a direct evaluation of the attack's effectiveness.
3. Recall [%]: measuring the ratio of original objects still detectable in perturbed images relative to clean images. A higher recall indicates a higher attack stealthiness.

6.2 Verifying the observation supporting RPB.

To validate the observation supporting RPB, we analyze per-image object distributions in the BDD dataset. Images are divided into three horizontal regions, with the middle region gradually expanded from 10% to 90% of the image area, and the ratio of middle-region objects to total objects recorded across 2,000 randomly sampled test images. As shown in Figure 2, when the middle region occupies just 30% of the image,

it already contains over 71% of all objects. This confirms that objects in AV camera inputs tend to concentrate in specific spatial regions, supporting the core assumption of RPB.

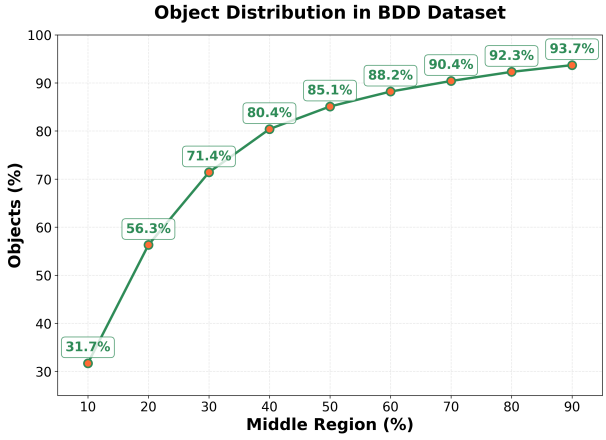


Figure 2: Object distributions of the images in the BDD dataset.

6.3 Attack Evaluation

Attack Performance. Table 1 presents the experimental results when our attack is applied to YOLOv5s, YOLOv8s, and YOLOv11s (marked by the grey background). We also evaluate various configurations (T_{ratio}) of the RBP. On YOLOv5s, compared to clean images (images without perturbation), our attack results in a more than 300-fold increase in the number of candidates fed into the third phase of NMS (M') with the most aggressive settings ($T_{ratio} = 0$). The overall execution time of YOLOv5s is increased by approximately 250% accordingly. However, the recall drops from 100% to 18%, making the attack easy to identify. When the T_{ratio} of RBP is set to 0.4, the recall doubles with slight sacrifices in M' and the introduced latency. When configured with a conservative setting ($T_{ratio} = 0.8$), our attack manages to push the recall above 64% while still retaining most of the introduced phantom objects.

When applied to YOLOv8s and YOLOv11s, our attack with the RBP framework also demonstrates good performance, dramatically increasing the stealthiness of the attack (recall) without significantly compromising its effectiveness (M' and T_{total}).

RBP Evaluation. Figure 3 shows how the RBP influences attack effectiveness and stealthiness with various configurations. As T_{ratio} (from 0.0 to 0.8) increases, the size of the middle region (i.e., the image area that cannot be perturbed) grows gradually. We denote the proportion of the middle region to the entire image area as R_{middle}. Consequently, there is less area available for adding phantom objects, leading to a decrease in M' as well as T_{total}. Since fewer phantom objects are added to the images, more original detections are preserved from the clean images.

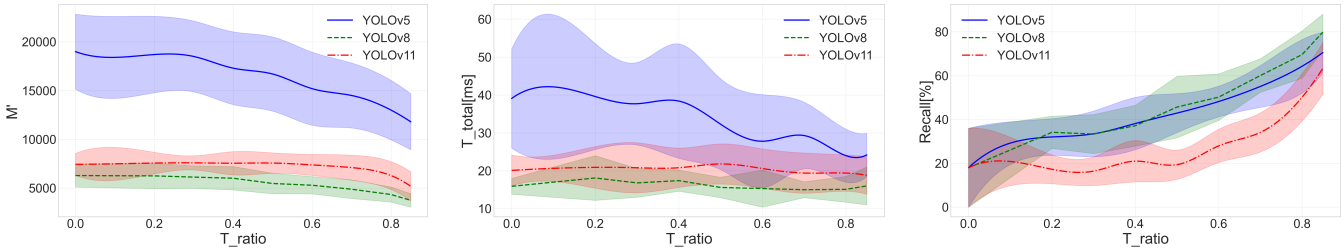
Table 1: Experimental results of the Groundswell attack compared to previous methods on YOLOv5s, YOLOv8s, and YOLOv11s. Gswell(T_{ratio}) refers to Groundswell attack with the T_{ratio} configuration shown in brackets. PSs($\ell_{max\ IoU}$) refers to the Phantom Sponge attack with the $\ell_{max\ IoU}$ setting shown in brackets.

YOLOv5s				
Attack	M'	T_{total}	T_{NMS}	Recall
Clean	101	15.2	1.7	100
Gswell(0)	19000	41.4	30.7	18.1
Gswell(0.4)	17300	38.4	23.8	38.2
Gswell(0.8)	13000	24.0	11.6	64.3
PSs(0)	18900	39.8	26.2	18
PSs(0.2)	15500	29.3	16.9	45.6
PSs(0.3)	13300	24.9	12.8	62.9
Overload	15400	31.0	19.1	4.5
DETSTORM	19800	41.1	28.5	2.2

YOLOv8s				
Attack	M'	T_{total}	T_{NMS}	Recall
Clean	52	12.8	1.5	100
Gswell(0.0)	6400	18.2	5.2	21.2
Gswell(0.4)	6100	17.7	4.9	37.2
Gswell(0.8)	4400	14.2	3.7	69.6
PSs(0)	6400	18.8	5.7	20.8
PSs(0.2)	5400	14.2	4.5	50.1
PSs(0.3)	4900	14.5	4.0	59.2
Overload	7200	18.3	6.7	4.5
DETSTORM	5700	16.8	4.5	4.0

YOLOv11s				
Attack	M'	T_{total}	T_{NMS}	Recall
Clean	50	15.3	1.6	100
Gswell(0.0)	7500	20.9	6.4	8.7
Gswell(0.4)	7500	20.8	6.0	21.0
Gswell(0.8)	6200	18.7	4.8	50.1
PSs(0)	7600	20.3	6.8	8.4
PSs(0.2)	6200	18.5	4.9	46.3
PSs(0.3)	5700	16.8	3.0	53.4
Overload	7600	20.1	6.5	2.1
DETSTORM	7800	20.8	6.9	0.9

Figure 4 shows how R_{middle}, guided by RBP, evolves during the training process. The R_{middle} of RPB with $T_{ratio} = 0.2$ converges to 0.1 within 25 iterations. Similarly, the R_{middle} of RPB with $T_{ratio} = 0.8$ converges to 0.4 with some fluctuation. The evolution of the RPB boundaries follows the observation in Figure 2: when the middle region occupies 10% of the whole image ($R_{middle} = 0.1$), it contains more than 20% of the original objects ($T_{ratio} = 0.2$). When the middle region



(a) M' vs T_{ratio} (b) T_{total} vs T_{ratio} (c) Recall vs T_{ratio}

Figure 3: Latency attack with different RPB configurations.

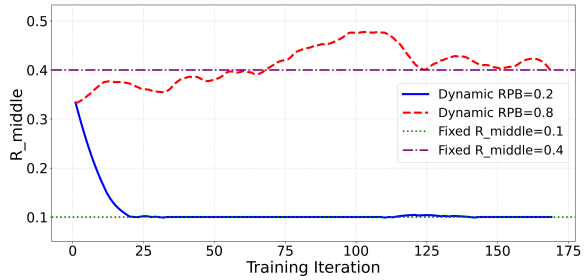


Figure 4: Evolution of the R_{middle} guided by RPB during the attack optimization.

occupies 40% of the whole image ($R_{middle} = 0.4$), more than 80% of the original objects are located within it ($T_{ratio} = 0.8$).

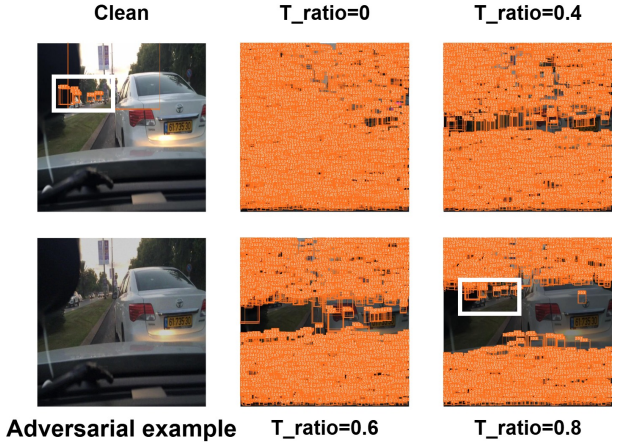


Figure 5: Visualization of Groundswell attack with different configurations of RPB when applied to a clean image.

Attack Visualization. Figure 5 visualizes how different configurations of RPB affect the attack. As shown, the crafted adversarial example, generated by applying the trained UAP and mask to a clean image, is virtually indistinguishable from the original image to the human eye. We further adjust target stealthiness level T_{ratio}. As T_{ratio} increases from 0 to 0.8, the area uncovered by the perturbation becomes larger, and progressively more original detections are revealed. When T_{ratio} is set to 0.8, most of the original detections (marked by white boxes) are preserved even when our attack is deployed.

Evaluation on different GPUs. We evaluate the Groundswell attack on different GPUs. The RPB framework is configured with T_{ratio} values of 0.2, 0.5, and 0.8, respectively. The evaluation results are presented in Table 2. As shown, on both the RTX 3080 and RTX 4080, our Groundswell attack introduces significant computational overhead to YOLOv5s. The results demonstrate that our method is effective across various GPU platforms.

6.4 Comparison to SOTA Approaches.

We compare our attack to three existing latency attacks (Phantom Sponges [22], Overload [3], and DETSTORM [19]) on YOLOv5s, YOLOv8s, and YOLOv11s.

The Phantom Sponges attack includes a third loss function $\ell_{max\ IoU}$ with an associated weight $\lambda_{max\ IoU}$ for adjusting attack stealthiness. Different configurations of the Phantom Sponges regarding $\ell_{max\ IoU}$ are presented for comprehensive comparison. Overload was initially designed for a different scenario. It generates distinct perturbations for individual images; we adapted its open-source code [20] to generate UAPs across multiple images and fine-tuned the training process for comprehensive comparison.

As shown in Table 1, Overload exhibits the worst performance in both attack effectiveness and stealthiness since it was not designed for training UAPs. DETSTORM has similar performance to the most aggressive configuration of our approach since it does not take attack stealthiness into consideration during attack optimization. Compared to Phantom Sponges, the biggest advantage of our approach lies in the efficiency of attack optimization. Table 3 compares our Groundswell attack with the RBP framework and the Phantom Sponges attack with $\ell_{max\ IoU}$ in terms of CPU utilization and execution time per epoch during optimization. Our approach reduces CPU utilization by half and optimization time by 80%. The loss function $\ell_{max\ IoU}$ employed by Phantom Sponges tracks the IoU between each pair of detected objects, which becomes costly when the number of detected objects is maximized.

Table 2: Performance evaluation of our Groundswell attack on different hardware with various configurations of RPB. T_{clean} and T_{attack} denote the end-to-end execution times of YOLOv5s for clean and adversarial images, respectively. Increase[%] shows the percentage increase in execution time from adversarial vs. clean images.

		$T_{ratio} = 0.2$			$T_{ratio} = 0.5$			$T_{ratio} = 0.8$		
GPU	T_{clean}	T_{attack}	Increase[%]	Recall	T_{attack}	Increase[%]	Recall	T_{attack}	Increase[%]	Recall
RTX3080	10.5	46.4	341.9	32	42.6	305.7	43	29.3	179.0	64
RTX4080	8.6	39.6	360.5	32	35.5	312.8	43	19.2	123.3	64

Table 3: UAP generation cost comparison between our Groundswell attack (Gswell) and the Phantom Sponges attack (PSs)

Method	CPU(%)	Time(s)
Gswell	48.0	∼50
PSs	96.0	∼250

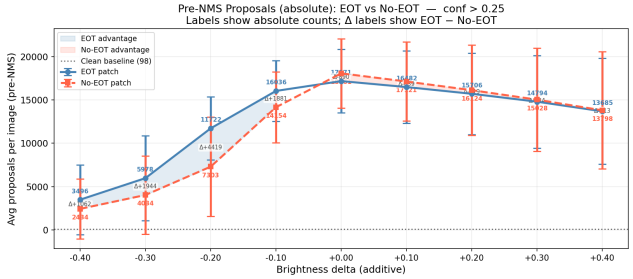


Figure 6: Pre-NMS proposal counts (M') per image at $\tau_{\text{conf}} = 0.25$ for the EOT and No-EOT patches across additive brightness deltas δ. Shaded region marks EOT advantage; Δ labels show EOT − No-EOT at each level; error bars denote standard deviation.

6.5 Robustness Under Environmental Factors

Real-world deployment of adversarial patches is subject to uncontrolled environmental variation (e.g., changing illumination, weather, and sensor noise). Expectation Over Transformation (EOT) [1] addresses this by averaging gradients over a distribution of transformations during training, producing a perturbation that is adversarially effective in expectation across the target distribution rather than at a single operating point. To demonstrate the impact of EOT on Groundswell, we use brightness variation as a representative environmental factor and evaluate how both patches hold up as ambient illumination shifts away from the neutral training condition.

Setup. We train two YOLOv5s patches under identical conditions (Section 6.1, $T_{\text{ratio}} = 0.3, \varepsilon = 70$, 100 epochs): a standard Groundswell patch trained at neutral brightness (No-EOT), and an EOT-augmented patch whose PGD gradient at each step is averaged over n=8 brightness samples drawn uniformly from $\delta \in (-0.3, +0.3)$ (EOT). Both patches are evaluated on the BDD100K validation split under a fixed sweep $\delta \in \{-0.40, \ldots, +0.40\}$, measuring M' at $\tau_{\text{conf}} = 0.25$.

Results. Figure 6 reveals three distinct conditions. In dark conditions ($\delta < -0.10$), EOT substantially outperforms the baseline. At $\delta = -0.20$, the EOT patch generates 9555 proposals per image versus 6132 for No-EOT (+55.8%). This confirms that EOT builds robustness to brightness suppression that a neutrally trained patch cannot acquire. At ($\delta \approx 0.00$), No-EOT holds a negligible edge (15505 vs. 14412, −7.0%). The baseline concentrates its entire gradient budget at this operating point, so the marginal advantage is expected and small. Above $\delta = +0.10$, both patches converge to near-identical counts as pixels saturate at the $[0, 1]$ boundary.

7 Discussion

While Groundswell demonstrates strong performance, two limitations warrant discussion. First, RPB relies on spatial object distribution patterns learned from BDD100K, so cameras mounted at significantly different heights or pitch angles could misalign the perturbation mask with actual sparse regions. Since mounting positions are fixed post-installation and follow consistent conventions across vehicle classes, a practical mitigation is to maintain a small offline dictionary of UAPs per mounting configuration. Second, while BDD100K's diverse weather and lighting conditions provide implicit environmental robustness, dedicated analysis remains future work; EOT [1] can be incorporated to optimize perturbations over distributions of noise, brightness, and contrast shifts. We conduct a preliminary evaluation, and the results confirm that EOT introduces extra robustness to the optimized UAP under various brightness conditions.

8 Conclusion

In this paper, we present Groundswell, a novel latency-based attack against deep learning-powered object detection systems that addresses critical limitations of existing attack methodologies. Our experimental results demonstrate that the Groundswell attack successfully reduces computational resource requirements by 50% and decreases optimization

time by 80%, while maintaining comparable or superior attack effectiveness. These efficiency gains make our approach particularly practical for real-world deployment scenarios.

Acknowledgments

We would like to extend our appreciation to the shepherd and anonymous reviewers for their invaluable input on our study. This work was partially supported by the National Science Foundation (NSF) grants CNS-2317830 and CNS-2338837.

References

[1] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In International conference on machine learning, pages 284–293. PMLR, 2018.

[2] Manuel Carranza-García, Pedro Lara-Benítez, Jorge García-Gutiérrez, and José C Riquelme. Enhancing object detection for autonomous driving by optimizing anchor generation and addressing class imbalance. Neurocomputing, 449:229–244, 2021.

[3] Erh-Chung Chen, Pin-Yu Chen, I Chung, Che-Rung Lee, et al. Overload: Latency attacks on object detection for edge devices. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 24716–24725, 2024.

[4] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1625–1634, 2018.

[5] GSA Forum. The challenges to achieve level 4 & level 5 autonomous driving, 2023.

[6] Abhishek Gupta, Kandasamy Illanko, and Xavier Fernando. Object detection for connected and autonomous vehicles using cnn with attention mechanism. In 2022 IEEE 95th Vehicular Technology Conference:(VTC2022-Spring), pages 1–6. IEEE, 2022.

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In Proceedings of the IEEE international conference on computer vision, pages 2961–2969, 2017.

[8] Hugging Face. State of open source on Hugging Face: Spring 2026. https://huggingface.co/blog/huggingface/state-of-os-hf-spring-2026, March 2026. Accessed: 2026-05-01.

[9] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics: Yolo. https://github.com/ultralytics/ultralytics, 2023. Accessed: April 10, 2025.

[10] Glenn Jocher et al. Ultralytics yolo documentation. https://docs.ultralytics.com/, 2023. Accessed: May 13, 2025.

[11] Glenn Jocher, Jing Qiu, and Ayush Chaurasia. Ultralytics YOLO, January 2023.

[12] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, ChristopherSTAN, Liu Changyu, Laughing, tkianai, Adam Hogan, lorenzomammana, yxNONG, AlexWang1900, Laurentiu Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Francisco Ingham, Frederik, Guilhen, Hatovix, Jake Poznanski, Jiacong Fang, Lijun Yu, changyu98, Mingyu Wang, Naman Gupta, Osama Akhtar, PetrDvoracek, and Prashant Rai. ultralytics/yolov5: v3.1 - bug fixes and performance improvements, October 2020.

[13] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In Computer vision–ECCV 2014: 13th European conference, zurich, Switzerland, September 6-12, 2014, proceedings, part v 13, pages 740–755. Springer, 2014.

[14] Chongwei Liu, Haojie Li, Shuchang Wang, Ming Zhu, Dong Wang, Xin Fan, and Zhihui Wang. A dataset and benchmark of underwater object detection for robot picking. In 2021 IEEE international conference on multimedia & expo workshops (ICMEW), pages 1–6. IEEE, 2021.

[15] Han Liu, Yuhao Wu, Zhiyuan Yu, Yevgeniy Vorobeychik, and Ning Zhang. Slowlidar: Increasing the latency of lidar-based detection using adversarial examples. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 5146–5155, 2023.

[16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14, pages 21–37. Springer, 2016.

[17] Chen Ma, Ningfei Wang, Qi Alfred Chen, and Chao Shen. Slowtrack: Increasing the latency of camera-based perception in autonomous driving using adversarial examples. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 38, pages 4062–4070, 2024.

[18] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083, 2017.

[19] Raymond Muller, Ruoyu Song, Chenyi Wang, Yuxia Zhan, Jean-Phillipe Monteuuis, Yanmao Man, Ming Li, Ryan Gerdes, Jonathan Petit, and Z Berkay Celik. Investigating physical latency attacks against camera-based perception. In 2025 IEEE Symposium on Security and Privacy (SP), pages 4588–4605. IEEE, 2025.

[20] Maria-Irina Nicolae, Mathieu Sinn, Minh-Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. Adversarial robustness toolbox (art) - python library for machine learning security. https://github.com/Trusted-AI/adversarial-robustness-toolbox, 2018.

[21] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28, 2015.

[22] Avishag Shapira, Alon Zolfi, Luca Demetrio, Battista Biggio, and Asaf Shabtai. Phantom sponges: Exploiting non-maximum suppression to attack deep object detectors. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pages 4571–4580, 2023.

[23] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. A general framework for adversarial examples with objectives. ACM Transactions on Privacy and Security (TOPS), 22(3):1–30, 2019.

[24] Ilia Shumailov, Yiren Zhao, Daniel Bates, Nicolas Papernot, Robert Mullins, and Ross Anderson. Sponge examples: Energy-latency attacks on neural networks. In 2021 IEEE European symposium on security and privacy (EuroS&P), pages 212–231. IEEE, 2021.

[25] Rejin Varghese and M Sambath. Yolov8: A novel object detection algorithm with enhanced performance and robustness. In 2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS), pages 1–6. IEEE, 2024.

[26] Derui Wang, Chaoran Li, Sheng Wen, Qing-Long Han, Surya Nepal, Xiangyu Zhang, and Yang Xiang. Daedalus: Breaking nonmaximum suppression in object detection via adversarial examples. IEEE Transactions on Cybernetics, 52(8):7427–7440, 2021.

[27] Kaidi Xu, Gaoyuan Zhang, Sijia Liu, Quanfu Fan, Mengshu Sun, Hongge Chen, Pin-Yu Chen, Yanzhi Wang, and Xue Lin. Adversarial t-shirt! evading person detectors in a physical world. In Computer vision–ECCV 2020: 16th European conference, glasgow, UK, August 23–28, 2020, proceedings, part v 16, pages 665–681. Springer, 2020.

[28] Oryan Yehezkel, Alon Zolfi, Amit Baras, Yuval Elovici, and Asaf Shabtai. Desparsify: Adversarial attack against token sparsification mechanisms. Advances in Neural Information Processing Systems, 37:127536–127560, 2024.

[29] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 2636–2645, 2020.

Artifact Appendix

Investigating Stealthy Latency Attacks Against Camera-Based Object Detection Systems

.1 Abstract

This artifact contains the implementation of Groundswell, a novel latency attack against deep learning-based object detection (OD) systems using the Regional Perturbation Balance (RPB) framework. The attack generates a Universal Adversarial Perturbation (UAP) via Projected Gradient Descent (PGD) that maximizes the number of phantom objects passing through Non-Maximum Suppression (NMS), while spatially constraining perturbations to image regions with fewer original detections to preserve stealthiness. The artifact includes: the Groundswell attack implementation (`uap_RPB.py`), local YOLO model copies (`local_yolos/`), dataset utilities (`datasets/`), and a main entry point (`run_attack.py`).

The artifact supports the following major claims:

- Groundswell reduces CPU resource utilization by ∼50% and optimization time by ∼80% compared to Phantom Sponges during UAP training.
- Groundswell increases the number of NMS candidates (M') by over 300× on YOLOv5s, increasing end-to-end latency by ∼250%.
- The RPB framework provides a configurable stealthiness-effectiveness trade-off via T_{ratio}, achieving over 64% recall at T_{ratio}=0.8 on YOLOv5s.
- Results transfer across YOLOv5s, YOLOv8s, and YOLOv11s architectures.

The artifact includes: the Groundswell attack implementation (`uap_RPB.py`), local YOLO model copies (`local_yolos/`), dataset utilities (`datasets/`), and a main entry point (`run_attack.py`). Output artifacts include the trained UAP patch, per-epoch loss logs, validation images, and saved patches.

.2 Description & Requirements

.2.1 Security, Privacy, and Ethical Concerns

The attack operates entirely in the digital domain. No physical hardware, real vehicles, or live camera systems are required. All attacked models (YOLOv5s, YOLOv8s, YOLOv11s) are publicly available pretrained models on MS-COCO. No undisclosed vulnerabilities or zero-day exploits are involved. The BDD100K dataset is a publicly available autonomous driving dataset. Generated UAP patches are research artifacts not optimized for physical-world deployment.

.2.2 How to Access

Zenodo DOI: https://doi.org/10.5281/zenodo.20397573

.2.3 Badges

- **Artifacts Available** — publicly hosted on Zenodo with permanent DOI.
- **Artifacts Functional** — installs and runs to produce UAP patches and per-epoch metrics.
- **Results Reproduced** — reproduces Tables 1, 2, and 3.

.2.4 Hardware Dependencies

- **GPU:** NVIDIA GPU with CUDA 11.3 support. Paper uses RTX 3090; RTX 3080 and RTX 4080 also validated (Table 2).
- **GPU Memory:** Minimum 10 GB VRAM (`batch_size=8` on 640×640 images).
- **RAM:** Minimum 16 GB recommended.
- **Storage:** ∼20 GB free (YOLO weights + BDD100K subset + outputs).

.2.5 Software Dependencies

OS: Ubuntu 18.04 LTS. Python 3.8 exactly. CUDA 11.3. PyTorch 1.10.1+cu113, TorchVision 0.11.2+cu113 (installed separately). TensorFlow 2.13.1. Other: `numpy==1.24.3`, `opencv-python-headless==4.10.0.84`, `scikit-learn==1.3.2`, `scikit-image==0.19.3`, `scipy==1.10.1`, `pandas==2.0.3`, `matplotlib==3.7.5`, `albumentations==1.0.3`, `Pillow==8.4.0`, `seaborn==0.13.2`, `tqdm==4.67.1`, `PyYAML==6.0.2`.

.2.6 Benchmarks

Dataset: Berkeley Deep Drive (BDD100K), available at https://www.kaggle.com/datasets/solesensei/solesensei_bdd100k. Use validation set (∼190 images suffice; full 10K set yields more stable statistics). Format: images in `./val/`, YOLO-format labels in `./det_20/`.
Models: YOLOv5s, YOLOv8s, YOLOv11s pretrained on MS-COCO — weights bundled in `local_yolos/*/weights/`.

.3 Set-up

.3.1 Installation

1. Verify Python 3.8: `python3.8 -version`
2. Create virtual environment: `cd RPBAttack/ && python3.8 -m venv artifact_env && source artifact_env/bin/activate`
3. Upgrade pip: `python3 -m pip install -upgrade pip setuptools wheel`
4. Install dependencies: `python3 -m pip install -r requirements.txt`
5. Install PyTorch: `python3 -m pip install torch==1.10.1+cu113`

`torchvision==0.11.2+cu113 -extra-index-url https://download.pytorch.org/whl/cu113 -no-deps`

6. Prepare BDD100K: place images in `RPBAttack/val/` and labels in `RPBAttack/det_20/`. Run `python3 convert.py` to convert to YOLO format. Sample images/labels are included for testing.

.3.2 Basic Test

Run the following to verify all imports and CUDA availability:

```
python3 -c "import torch; import cv2; import numpy; print('All imports OK')"
python3 -c "import torch; print('CUDA:', torch.cuda.is_available())"
```

.4 Evaluation Workflow

.4.1 Major Claims

(C1): Groundswell (T_{ratio}=0) increases NMS candidates (M') from 101 to ∼19,000 and total latency from 15.2 ms to ∼41.4 ms (∼250% increase) on YOLOv5s. Recall drops to ∼18% (Table 1).

(C2): Groundswell (T_{ratio}=0.8) maintains ∼64% recall while increasing M' to ∼13,000 and T_{total} to ∼24 ms on YOLOv5s (Table 1).

(C3): Groundswell uses ∼48% CPU at ∼50 s/epoch vs. Phantom Sponges at ∼96% CPU and ∼250 s/epoch (Table 3).

.4.2 Experiments

(E1): *Main Attack Training on YOLOv5s* [30 human-minutes + ∼1.5 compute-hours + 5 GB disk]: trains Groundswell UAP on YOLOv5s to reproduce YOLOv5s rows of Tables 1 and 3 for three T_{ratio} configurations (Claims C1, C2, C3).
Preparation: Edit `run_attack.py`: set `attack_type = "RPB"`, `models_vers = [5]`, `epoch = 100`, `epsilon = 70`, `iter_eps = 0.0008`, `lambda_1 = 1`, `lambda_2 = 10`, `batch_size = 8`, and set `BDD_IMG_DIR`/`BDD_LAB_DIR` to the dataset paths. Set `threshold_ratio = 0.0` for the first run.
Execution: Run `python3 run_attack.py`. Repeat with `threshold_ratio = 0.4` and `0.8` for all YOLOv5s rows. Monitor CPU usage in a separate terminal via `top`. To reproduce the Phantom Sponges baseline (Table 3), set `attack_type = "PS"`.
Results: Compare printed `average_proposal_nums` (M'), `T_total`, `T_NMS`, and `Recall` against Table 1. Output files are written to `experiments/<patch_name>/`: `final_results/final_patch.png`, `saved_patches/` (best checkpoint), `applied_patch/` (validation images with UAP), and `losses/` (pickled loss lists).

(E2): *Transferability to YOLOv8s and YOLOv11s* [20 human-minutes + ∼3 compute-hours]: reproduces YOLOv8s and YOLOv11s rows of Table 1 (Claim C1).
Preparation: Same configuration as E1.
Execution: For YOLOv8s set `models_vers = [8]` and run `python3 run_attack.py`. For YOLOv11s set `models_vers = [11]`. YOLOv8/11 outputs are auto-converted to YOLOv5 format via `convert_yolov11_to_yolov5_format()` before NMS.
Results: Compare printed metrics against the corresponding rows of Table 1. Numeric variance of ±5–10% across GPU hardware is expected due to floating-point non-determinism.

.5 Notes on Reusability

The artifact generalizes beyond the paper experiments.
Custom datasets: replace `./val/` and `./det_20/` with any dataset in YOLO format.
Custom YOLO targets: `models_vers` accepts any combination of `[5, 8, 11]`; multi-model training selects a random model each batch.
Attack aggressiveness: `epsilon` controls the L_2 perturbation budget (default 70); `iter_eps` controls PGD step size (default 0.0008).
Stealthiness: `threshold_ratio` (T_{ratio}, range 0.0–0.8) is the key parameter: 0.0 = maximum latency, 0.8 = maximum stealthiness. Docker deployment is strongly recommended for cross-platform reproducibility.
Known limitation: CPU-only execution is not supported; CUDA-capable GPU is mandatory.

.6 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.



Beyond Classical PKI: A Quantum-Safe Authentication Framework for Automotive OTA Updates*

Wei-Lun Lin
National Sun Yat-sen University

Ting-Yu Chen
National Sun Yat-sen University

Arijit Karati
National Sun Yat-sen University
arijit.karati@mail.cse.nsysu.edu.tw

Abstract

The automotive industry is transitioning to software-defined architectures, making Over-the-Air (OTA) updates essential for managing the long-term lifecycles of modern vehicles, which often exceed a decade. Although Public Key Cryptography (PKC) currently secures these updates, it remains vulnerable to emerging quantum threats. While Post-Quantum Cryptography (PQC) is a feasible solution, its improper adoption in the automotive sector may lead to larger key storage requirements and higher computational costs, thereby hindering the security of resource-constrained devices. To address long-term deployment concerns and comply with vehicle software update requirements, we propose a quantum-safe authentication protocol to deliver software OTA updates securely over the vehicle-to-cloud (V2C) link. Our protocol employs Leighton-Micali Signatures (LMS) for Original Equipment Manufacturer (OEM) server identification and the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) to establish session keys for OTA updates. To mitigate server-side bottlenecks in bursty-request scenarios, we introduce a one-time ticket-gating approach that filters out invalid requests, preventing flooding or DoS attacks against the OEM server. Security analysis demonstrates that our protocol preserves authenticity, confidentiality, integrity, and anti-rollback properties while resisting critical threats, including replay and quantum attacks. Comprehensive performance results show that it achieves more security traits with adequate functional overhead compared to existing works in the context of automotive systems.

1 Introduction

In recent years, the automotive industry has experienced a paradigm shift, with traditional mechanical vehicles evolving into advanced mobile computing platforms. This adaptation has enabled a range of cutting-edge capabilities, including Advanced Driver-Assistance Systems (ADAS), immersive infotainment, and energy management systems for electric vehicles (EVs). The complexity of automotive software correlates with an increasing demand for regular maintenance and feature enhancements. As a result, the industry is shifting from conventional, in-person service models to efficient wireless software delivery. To meet these evolving requirements, Over-the-Air (OTA) update technology has become the standard for many contemporary automotive systems. OTA updates simplify remote software deployment, thereby significantly reducing operational costs for manufacturers and alleviating the necessity for consumers to make repeated visits to service centers [14]. This transition to wireless transmission across wide-area networks necessitates a strong reliance on effective security frameworks to preserve the integrity of vehicle firmware and the safety of its occupants.

Modern EVs rely on wireless connectivity (e.g., cellular/5G or Wi-Fi) to receive updates via public networks. To ensure software safety, the Public Key Cryptography (PKC), such as Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC) for data confidentiality, and Digital Signature Algorithm (DSA) for software integrity. However, attackers may exploit network connectivity by eavesdropping on traffic, modifying transmitted data, and replaying observed messages if the security mechanism is insufficiently robust [10, 28]. For instance, works [1, 13, 35] show that remote attackers may exploit In-vehicle Infotainment (IVI) systems, telematics, or other wireless interfaces, and inject crafted messages to affect in-vehicle communication, such as the Controller Area Network (CAN). This flaw may allow attackers to leak sensitive data, including vehicle location and driving record, to attackers.

1.1 OTA Workflow and Challenges

Figure 1 shows an overview of OTA workflow, where **(a)** the OEM Server prepares an update package at the backend, uploads it to a cloud server, and notifies target vehicles; **(b)**

*This work was supported in part by the National Science and Technology Council (NSTC) under Grants NSTC 114-2221-E-110-042-MY3 and 114-2634-F-110-001-MBK.

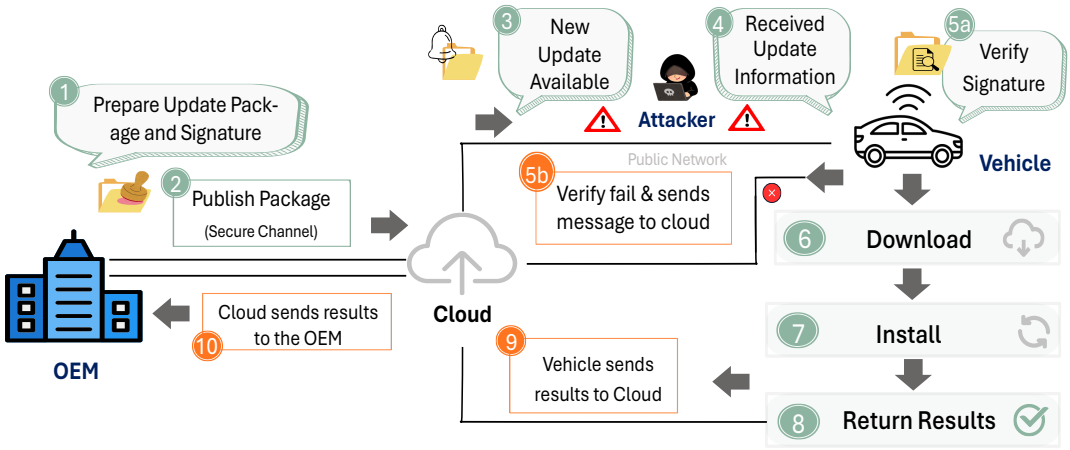


Figure 1: Overview of Over-the-Air (OTA) communication threats

after receiving the notification, vehicles verify the package and download it before installation, typically by checking a cryptographic hash and validating a digital signature; **(c)** after the install is complete, the vehicle will return the results to the OEM Server. For safety, vehicles should only install *trusted* software. Any software that fails verification is not accepted, so security in public network transmission is a priority.

1.1.1 Regulatory Industry and Technical Context

Contemporary vehicle-to-cloud (V2C)-based OTA updates rely on PKC-based security, with ECC used for key management and digital signatures (e.g., ECDSA) to ensure software integrity and accountability. Payload encryption secures the payload during transmission, but it is not sufficient. Vehicles require verifiable proof that the notification is from authentic OEM servers. The OTA updates involve a complex process for interconnected ECUs, including IVIs and safety-critical systems, such as ADAS. Halder et al. [14] demonstrated that this multi-node architecture is highly vulnerable, and a vulnerable ECU may lead to critical system failures. Thus, a security protocol must hold: (a) *Authentication and Integrity* to ensure the software is from a genuine OEM server; (b) *confidentiality* to protect intellectual property and sensitive firmware; and (c) *Freshness* to prevent replay attacks. In the absence of these traits, malicious nodes can replay old packages, leading to resource flooding or Denial-of-Service (DoS) attacks that jeopardize the vehicle network and the OEM infrastructure.

OTA updates are a necessary long-term maintenance in the automotive industry. On the regulation side, the UN-ECE/WP.29 UN R156 [29] deals with software updates and the Software Update Management System (SUMS), while the UN R155 [9] deals with vehicle cybersecurity and the Cybersecurity Management System (CSMS). At a more engineering level, ISO 24089 defines requirements and recommendations for the software update engineering, and the objectives of UN R156 are similar. Similar to UN R155, ISO/SAE 21434 [9] specifies cybersecurity engineering requirements across the vehicle lifecycle. ISO 26262 updates remain relevant to safety-related E/E functions and are up to date. Benyahya et al. [5] have shown that In-vehicle R156, ISO 24089, and Out-of-vehicle ISO 21177 for V2X are often part of industry regulations. The increasing adoption of EVs [36] and the growing complexity of IVI software stacks [16, 32] result in numerous update requests that can strain OEM infrastructure, leading to compute pressure and potential DoS vulnerabilities during bursty release cycles. The fundamental framework for this ecosystem is Uptane, proposed by Kuppusamy et al. [17]. *Uptane* is the first comprehensive update framework designed for compromise resilience, ensuring that vehicles install authorized, fresh, and compatible images even if parts of the infrastructure, such as a repository or an in-vehicle ECU, are compromised. The Uptane v2.1.0[1] provides a comprehensive security model centered on three pillars:

i. *Trusted Software Identification:* Utilizing signed metadata and Image/Director repositories to mitigate rollback, freeze, and mix-and-match attacks.

ii. *In-Vehicle ECU Management:* Leveraging secure time and Primary/Secondary verification models to ensure safe internal distribution.

iii. *Operational Interoperability:* Providing Protocols, Operations, Usage, and Formats (POUFs) to support standardized implementation across different OEMs.

Uptane accommodates confidentiality[2] either as end-to-end or through classical transport layers like TLS. It has exhibited adaptability to emerging threats, including experimental support for PQC (e.g., SLH-DSA, a.k.a. SPHINCS+), since 2022.

[1] https://uptane.org/docs/2.1.0/standard/uptane-standard
[2] https://uptane.org/docs/latest/standard/uptane-standard#431-read-updates

While Uptane provides a software update framework for G1 (Authenticated announcement), G2 (Integrity, Confidentiality), and G3 (Anti-rollback), it remains transport-agnostic. Consequently, it does not provide the protocol-level optimizations, specifically regarding session setup and handshake overhead, required to mitigate the substantial performance and latency penalties inherent in PQC-based V2C communication.

We develop a secure protocol to safeguard the wireless V2C delivery (Uptane's Type (i) deployment). While we adopt Uptane's security goals G1-G3, we specifically focus on G4 (Server-side DoS resilience) and G5 (Low-latency session setup) against quantum adversaries. Our protocol implements a ticket-gating cryptographic mechanism to mitigate resource exhaustion during the heavy PQC handshake. Besides, it minimizes the number of online exchanges to prevent session loss in high-mobility conditions with irregular connectivity.

1.1.2 Security Goals

Considering specific vehicle and OTA issues [14, 17], as well as the aforementioned practical constraints that motivate our security goals. Particularly, we aim for (i) authenticated release announcements, (ii) software package integrity and confidentiality, (iii) replay and anti-rollback resistance, (iv) backend robustness to withstand burst requests, and (v) low-latency session setup. We summarize our key security goals below.

- **G1 (Authenticated announcement).** V accepts a release announcement only if it originates from the legitimate OEM backend S and has not been modified.
- **G2 (Software package integrity and confidentiality).** The software package ensures integrity to prevent tampering and confidentiality to protect intellectual property.
- **G3 (Anti-replay and Anti-rollback).** Fresh sessions reject replayed messages; updates are only accepted if their version number exceeds the locally stored maximum.
- **G4 (Server-side DoS resilience).** The server executes an expensive session setup only after an authorization check succeeds, enabling early rejection during a flood.
- **G5 (Low latency session setup).** Since a vehicle is a moving device, avoiding packet loss during transmission requires mitigating the online authentication.

1.2 Our Contributions

In addition to the challenges of external software validation, we propose a lightweight quantum-resistant authentication for secure OTA updates. The design prioritizes OEM authenticity verification while maintaining a 1-RTT online authentication to mitigate network latency and server strain during high-concurrency scenarios. For long-term security, the protocol incorporates post-quantum cryptographic primitives.

Following NIST guidance and recommendation, we adopt LMS for signed announcements [8] Moreover, ML-KEM is used to establish an ephemeral shared secret [23]. To further reduce server-side computing pressure under request flooding, We add a one-time ticket-gating mechanism based on a Merkle root R_{ID} and a hash-based one-time signature (OTS). The server checks the ticket (Merkle proof, OTS verification, and spent flag). After key establishment, symmetric cryptography, such as ASCON-AEAD [33], is used to efficiently protect large OTA payloads. This structure keeps heavy payload protection in the data plane and keeps the authentication process simple. The proposed protocol has the following features:

- **Quantum-safe and server authentication:** The vehicle only accepts the announcement if it verifies success under pk^{sig}. This reduces future quantum risks while preserving authenticated OEM-origin announcements.
- **Software package integrity and confidentiality** By integrating the NIST algorithm, we employ ASCON for AEAD to make the receiver verify TAG^{AEAD} to ensure the software package is not tampered with, and to provide confidentiality during transmission.
- **Mitigate OEM server computing pressure:** The ticket mechanism is designed to handle invalid vehicle requests and mitigate server pressure during short bursts of requests. Most out-of-list floods are filtered.
- **Feasibility of Deployment and 1-RTT:** The online exchange uses lightweight primitives, including HKDF, HMAC, hashing, ticket checks, and ASCON-AEAD to efficiently protect large OTA payloads. Then, we achieve key confirmation in 1-RTT, which is excellent for practical deployment.

Our protocol achieves deployment feasibility and operational efficiency through a multi-stage verification process. Once the server publishes a signed update announcement, the vehicle verifies the version, software authenticity, and announcement integrity. Next, the vehicle selects an unused OTS secret key (sk_x) and executes a lightweight authentication to derive a shared secret and confirm the key. By checking the OTS ticket first, the server can reject invalid requests early, saving computational resources. This online exchange completes within 1-RTT, and payload transmission uses the lightweight ASCON AEAD cipher for maximum efficiency.

2 Related Work

Table 1 groups existing OTA methods into three scopes: Type 1 – *OEM/Repository to Vehicle*, prioritizing secure public transmission; Type 2–*OEM to ECU*, addressing gateway distrust and update management; and Type 3–*End-to-End framework*, covering the entire pipeline from repository to ECU.

Table 1: Related Secure OTA Update Mechanisms comparison

Work	Year	Type	Method	Advantages	Drawbacks
Mbakoyiannis et al. [22]	2019	Type 3	Role-separated	∗ Low single trust ∗ Replay resistance	∗ Deployment complexity ∗ Time dependency
Manna et al. [20]	2021	Type 1	Attribute-Based Encryption on Embedded	∗ Measured overhead ∗ Practical integration	∗ Key revocation maintenance ∗ Key distribution overhead
Yeasmin et al. [37]	2021	Type 1	MFA + blockchain-based OTA framework	∗ Decentralization ∗ Distributed System	∗ Consensus latency ∗ Operational complexity
Ghosal et al. [12]	2022	Type 1	Ciphertext-Policy ABE	∗ Cloud secrecy ∗ Scalable scheduling	∗ Policy/ABE management ∗ ABE overhead
Plappert et al. [26]	2023	Type 2	ECU attestation	∗ Integrity assurance	∗ Hardware requirement ∗ Provisioning cost
Bazzi et al. [4]	2023	Type 2	Merkle-tree	∗ Fine-grained updates ∗ Efficient verification	∗ Tree state maintenance
Shoker et al. [31]	2023	Type 3	Update-station	∗ Lower cellular bandwidth ∗ Mitigate server bottleneck	∗ New operator boundary ∗ Station storage needed
Wang et al. [34]	2024	Type 1	PKI hybrid encryption + AES encryption	∗ Mature primitives ∗ Integrity verification	∗ Classical PKI reliance ∗ Verification failure risk
Kovacevic et al. [19]	2024	Type 1	Decentralized Identifier Framework	∗ Decentralization ∗ Explicit revocation	∗ Integration complexity ∗ Ecosystem maturity
Kim et al. [18]	2025	Type 3	Multi-Factor Authentication	∗ Gateway independence ∗ Deployment practicality	∗ Credential overhead
Chung et al. [7]	2025	Type 3	Zonal hierarchical + PQC-hybrid TLS	∗ PQC-ready zonal OTA architecture	∗ Relies on hardware ∗ Infrastructure requirement
Lu et al. [21]	2025	Type 3	One-machine-one-certificate OTA	∗ Integrated security ∗ Lightweight overhead	∗ Gateway trust assumption ∗ Certificate management

Type 1: Backend-to-vehicle OTA security. **Type 2**: ECU-level authentication or verification. **Type 3**: Hierarchical OTA distribution via gateway, TCU, primary ECU, or zonal gateway.

Manna et al. [20] evaluate attribute-based encryption (ABE) on an automotive-grade embedded platform, showing the practical cost of using ABE to protect OTA update confidentiality even when updates are stored and delivered via an untrusted cloud. Ghosal et al. [12] propose STRIDE, which uses CP-ABE to enforce fine-grained access control for OTA packages. Yeasmin et al. [37] designed a multi-factor authenticated OTA framework in which the blockchain ledger provides decentralized validation/auditability of update records. [12] propose STRIDE, which uses CP-ABE to enforce fine-grained access control for OTA packages and further designs a scheduling algorithm to improve update throughput under dynamic cellular traffic while maintaining end-to-end security, including confidentiality and freshness. Wang et al. [34] select an RSA signature with AES to protect OTA payload transmission. Kovacevic et al. [19] proposed a decentralized identifier (DID)-based OTA framework that uses verifiable credentials and a distributed ledger, emphasizing decentralization with explicit revocation and traceable, accountable update authorization.

Plappert et al. [26] use TPM 2.0-based ECU attestation with DICE support to provide a hardware root of trust; the backend can verify ECU integrity before accepting and installing OTA updates. Bazzi et al. [4] propose MT-SOTA. This Merkle-tree-based scheme enables fine-grained updates for independent software clusters, improving update efficiency while keeping integrity/compatibility checks lightweight, and it is compatible with hash-based signatures for quantum-resilient settings.

Shoker et al. [31] propose ScalOTA, which introduces update stations integrated with EV charging stations to mitigate the cellular repository bottleneck and reduce download latency, while formalizing an end-to-end chain of trust across OEMs/suppliers/stations/ECUs. Chung et al. [7] design an automotive OTA system with zonal gateways and PQC-hybrid TLS, focusing on a zonal hierarchical architecture and a PQC-ready secure channel for future deployment. Mbakoyiannis et al. [22] devised a role-separated secure firmware workflow (e.g., update manager, firmware server, key server, timestamp server), where role separation helps reduce single points of trust and improves replay resistance. Kim et al. [18] propose an in-vehicle OTA protocol based on multi-factor authentication, in which vehicles directly establish mutual authentication with the server without relying on a gateway, thereby improving the practicality of deployment in some architectures.

Quantum Threat Discussion: Vehicles often remain in service for over a decade; OTA security should account for long-term cryptographic risks. Shor's algorithm shows that a sufficiently capable quantum computer can solve integer factoring efficiently, which erodes the security foundation of RSA-type public-key cryptography and weakens long-term confidence in widely deployed PKC trust anchors used for

authentication and key establishment in OTA systems [6, 24]. Regarding the migration requirement, Fernandez et al. [11] discuss quantum-resistant cryptosystems for IoT and explain why performance and resource overhead must be considered alongside security. Shim et al. [30] compare PQ signature families and emphasize long-term security in domains with long product lifetimes; they also suggest that the transition may take years and should be planned in phases. In the real world, most vehicles last 10 or more years. Thus, the OTA must consider long-term deployment; it is better to plan PQC migration early and keep a practical design.

PQC was proposed to remain secure even against quantum adversaries [25, 30]. Motivated by this, the National Institute of Standards and Technology (NIST) has released PQC standards and recommendations, including LMS for firmware signing [8], ML-KEM for post-quantum key establishment [23], and ASCON-AEAD for large-scale data protection [33]. As we know, OTA packages may contain both firmware and higher-layer software components, and post-quantum key establishment, such as ML-KEM, is also relevant in OTA settings [23]. However, a recent study [30] shows that PQC often introduces higher cost, and many schemes use larger keys. These overheads increase bandwidth usage, and more computing time will delay the update. For resource-limited telematics units, this is not a small issue. In an OTA process, stronger cryptography can directly lead to longer update times and a worse user experience. Therefore, balancing security and efficiency is necessary, not optional.

3 Preliminaries

This section lists symbols in Table 2 and the cryptographic primitives and assumptions used throughout the paper. We formalize LMS as a stateful signature primitive, model ML-KEM as a KEM abstraction, and state the underlying hardness assumptions referenced in the protocol and security analysis.

3.1 Leighton–Micali Signature

Let pk^{sig} and sk^{sig} be the public and secret keys, and σ denotes a Leighton–Micali Signature (LMS). LMS signature σ includes a one-time leaf index i and the corresponding Merkle authentication path π_i used by LMS.Verify.

Definition 1 (Leighton–Micali Signature (LMS))**.** The LMS consists of signing and verification algorithms for $b \in \{0,1\}$:

$$\sigma \leftarrow \mathsf{LMS.Sign}(sk^{sig}, m), \quad b \leftarrow \mathsf{LMS.Verify}(pk^{sig}, m, \sigma)$$

Statefulness. LMS is stateful: each one-time leaf index must be used at most once. The signer maintains a persistent state idx and updates it after each signature:

$$\mathsf{idx} \leftarrow \mathsf{idx} + 1 \tag{1}$$

Security notion. Assuming correct state management (no OTS key reuse), we treat LMS as EUF-CMA secure.

Table 2: Summary of Notations

Symbol	Description
S, V, ID	OEM Server & Vehicle & Vehicle ID
N_S, N_V	Fresh nonces (server, vehicle)
$T_{1\sim8}, M_{1\sim8}$	Timestamp & Transmission Message
Ver_S, Ver_V	Version & local maximum version
$H(\cdot), \|$	Hash function & concatenation
Dst	authentication transcript hash
ek^{eph}, dk^{eph}	Server ephemeral ML-KEM key pair
cti, ss	index i ML-KEM ciphertext & shared secret
hs	Handshake secret for online authentication
sk^{sig}, pk^{sig}	LMS signing key pair
σ	LMS signature on announcement
MD_S	Signed metadata
HKDF	HMAC-based Key Derivation Function
Len	Application-key length
K_S^{kc}, K_R	Key-confirmation keys
TAG_S, TAG_R	Key-confirmation tags & Recovery tags
K_s, K_v	Session Key
ASCON	Lightweight AEAD (ASCON-AEAD)
$Pack, Res, \Delta T$	OTA payload & Vehicle install results, & $T_{1\sim8}$ tolerant
ct^{Pack}, ct^{Res}	Ciphertext (ct) of payload & ct of install results
TAG^{AEAD}	ASCON tag
OTS	One-Time Signature for ticket proof
R_{ID}	Merkle root of ticket public keys
pk_x, sk_x	Ticket one-time public/secret key pair (index)
$path_x$	Merkle authentication path for pk_x
Sig_x	OTS signature for ticket (proof of sk_x)

3.2 Module-Lattice-Based Key-Encapsulation Mechanism and Hardness Assumptions

For the Key-Encapsulation Mechanism (KEM) component, the server's ephemeral key pair is denoted by $(KEMpk_S^e, KEMsk_S^e)$; cti and ss denote the encapsulation ciphertext and the derived shared secret, respectively. $H(\cdot)$ denotes a cryptographic hash function. For the MLWE assumption, q, n, and k denote the modulus, ring dimension, and module rank, and $\mathsf{KEM}.*$ denotes the KEM algorithms.

Definition 2 (Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM))**.** It has the following algorithms:

$$\begin{aligned}(pk^e, sk^e) &\leftarrow \mathsf{KEM.KeyGen}(),\\ (cti, ss) &\leftarrow \mathsf{KEM.Encaps}(pk^e),\\ ss &\leftarrow \mathsf{KEM.Decaps}(sk^e, cti).\end{aligned}$$

For honestly generated inputs, if $(pk^e, sk^e) \leftarrow \mathsf{KEM.KeyGen}()$ and $(cti, ss) \leftarrow \mathsf{KEM.Encaps}(pk^e)$, then $\mathsf{KEM.Decaps}(sk^e, cti) = ss$.

We consider ML-KEM as an IND-CCA2-secure KEM.

Definition 3 (Decision-MLWE)**.** Let q be a modulus and let $R_q = \mathbb{Z}_q[x]/(x^n+1)$. Sample $A \xleftarrow{\$} R_q^{k\times k}$, $s \xleftarrow{\$} R_q^k$, and $e \xleftarrow{\$} \chi^k$

from a noise distribution χ. Let $b := As + e \in R_q^k$. The decision-MLWE assumption states that (A, b) is computationally indistinguishable from (A, u) where $u \xleftarrow{\$} R_q^k$.

Definition 4 (Hash one-wayness and second-preimage resistance)**.** Let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a cryptographic hash. We assume no PPT adversary can find (i) a preimage for a random hash output, or (ii) $x' \neq x$ such that $H(x') = H(x)$, with non-negligible probability.

The ML-KEM is IND-CCA2 secure under the MLWE assumption (see Definition 3). Besides, the LMS is EUF-CMA secure under the hash assumptions (see Definition 4) and correct state management.

3.3 System Model and Adversary Model

In this section, we will discuss the system model, protected assets, trust/provisioning assumptions, and the adversary model.

3.3.1 System Model

The OTA authentication framework and system model are illustrated in Fig. 2. It consists of the following entities.

- **Vehicle OTA Client (V):** A vehicle-side client (e.g., a TCU) receives the release announcement, verifies OEM authenticity, executes a 1-RTT online exchange, and installs the update only after cryptographic verification.
- **Cloud/CDN (delivery layer, optional):** The Cloud/CDN functions as a transparent delivery layer, facilitating the storage and distribution (caching/relaying) of OTA payloads. Security is maintained through end-to-end cryptographic trust between the Source (S) and V_i, ensuring that the delivery infrastructure remains outside the trust boundary.

OTA Workflow and Two-Plane Structure. As depicted in Fig. 1, the workflow has two planes. *Control plane: S* broadcasts an LMS-signed announcement carrying minimal metadata $MD_S = \{Ver_S, ek^{eph}\}$; after accepting the announcement, V runs a 1-RTT exchange to obtain a fresh session secret with explicit key confirmation. *Data plane: S* encrypts the OTA payload under an AEAD key derived from the session secret, and V installs only after verification.

3.3.2 Adversary Model

We adopt a Dolev-Yao-style network attacker A operating over the public channel. The adversary can eavesdrop on, modify, inject, replay, delay, drop, reorder, and interleave messages across concurrent sessions. The attacker may also create bursty request patterns to increase backend workload.

Our DoS discussion focuses on *compute-exhaustion* against the OEM backend, where the goal is to force the server to check whether the vehicle is valid or not. Accordingly, our design places ticket validation before server-side to enable early rejection. Volumetric network DDoS attacks, wireless jamming, and physical compromise of ECUs are outside our scope. These can be handled by deployment controls (e.g., rate limiting, filtering, redundancy) orthogonal to our protocol.

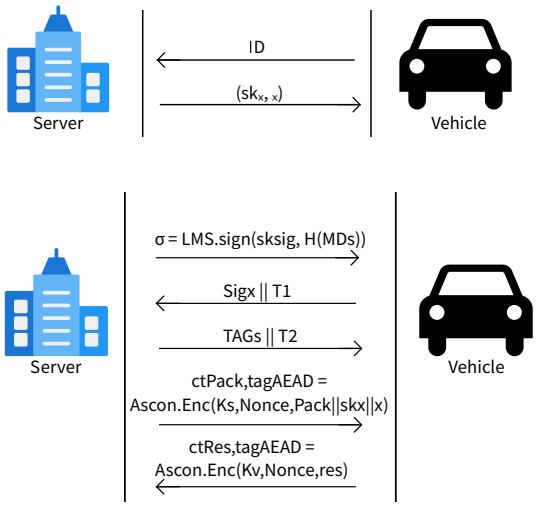


Figure 2: Registration and authentication process

3.3.3 OTA Resource-Pressure Setting

In practical OTA deployments, releases are often published within short time windows, and many vehicles connect concurrently after receiving notifications. We therefore consider a bursty workload in which $n \in \{100, 500, 1000\}$ vehicles request updates during the same period. This setting stresses both the public network path and the OEM backend, where delays and congestion can occur simultaneously. Our primary objective under this setting is to keep the online authentication lightweight (1-RTT with key confirmation), and to reduce server-side, expensive work under heavy load by validating tickets before running the full authentication phase.

4 Proposed Protocol

As shown in Figs. 3-5, our design separates the control plane and the data plane for OTA authentication and OTA payload transmission. In the control plane, we integrate the LMS post-quantum signature algorithm to sign the release announcement MD_S; these signatures are performed offline; ML-KEM to establish an ephemeral shared secret.

Moreover, HKDF for key derivation, HMAC for explicit key confirmation; in the data plane, we select NIST AEAD ASCON to protect the OTA payload, and the entire design is divided into four phases. For the network factor such as packet loss problem happened in phase 4, we strictly consider this

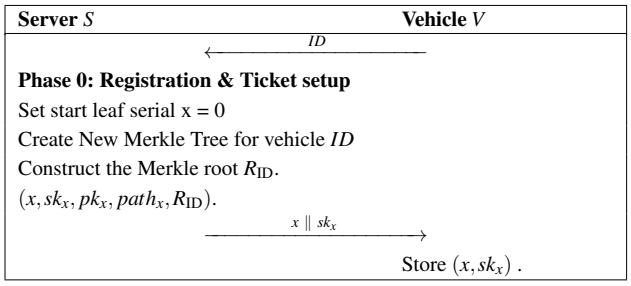


Figure 3: Registration phase over a secure link

kind of situation, we design the ticket recovery mechanism for the specific V, the concept is V select valid hs and execute HMAC to generate TAG_R then sending his $H(ID)$ to S that S can identify V is legal or not, and implement the recovery process. All long-term secret keys, including the master secret key, are stored in TPM 2.0, which provides hardware-backed secure key storage.

Registration and Ticket Setup: In phase 0, V reveals its personal information, as defined by the vehicle ID ID it sends to S. Then S create a unique Merkle tree that only for specific vehicle ID, then transmission the ticket mechanism, as we define the V as a signing party, and OEM Server is a verifying party, hence if the V cannot not demonstrate Sig_x under sk_x then we define that it is not valid V, and will reject OTA update in advanced in the authentication phase, as we define phase 3.

4.1 Offline Update Announcement

In Phase 1 (Offline signature and update announcement), the S generates an ephemeral ML-KEM key pair $(dk^{eph}, ek^{eph}) = \mathsf{ML\text{-}KEM.Gen}()$, the OEM server S prepares a signed metadata object $MD_S = \{Ver_S, ek^{eph}\}$, where Ver_S is the target software version. The key point is that the V must be able to determine whether this release package is genuine before incurring any costs for the online authentication.

For the signing phase, S uses the stateful LMS signature. Each LMS leaf key is one-time; hence, S maintains a signing state (e.g., a leaf index) and advances it with each release. If frequent releases are expected, S can provision a larger LMS tree or rotate to a fresh LMS instance once the current tree approaches exhaustion. This behavior keeps the protocol message flow unchanged and remains a deployment requirement. To preserve the one-time property under reboot and prevent state rollback or key exposure, S protects sk^{sig} and its state in a secure environment (e.g., TPM-backed storage). Then S is computed and signed offline:

$$\sigma = \mathsf{LMS.Sign}(sk^{sig}, H(MD_S))$$

The OEM server sends the update announcement.

$$S \rightarrow V: \; M_1 = (\sigma \parallel MD_S)$$

All of these steps in the OEM server are performed offline.

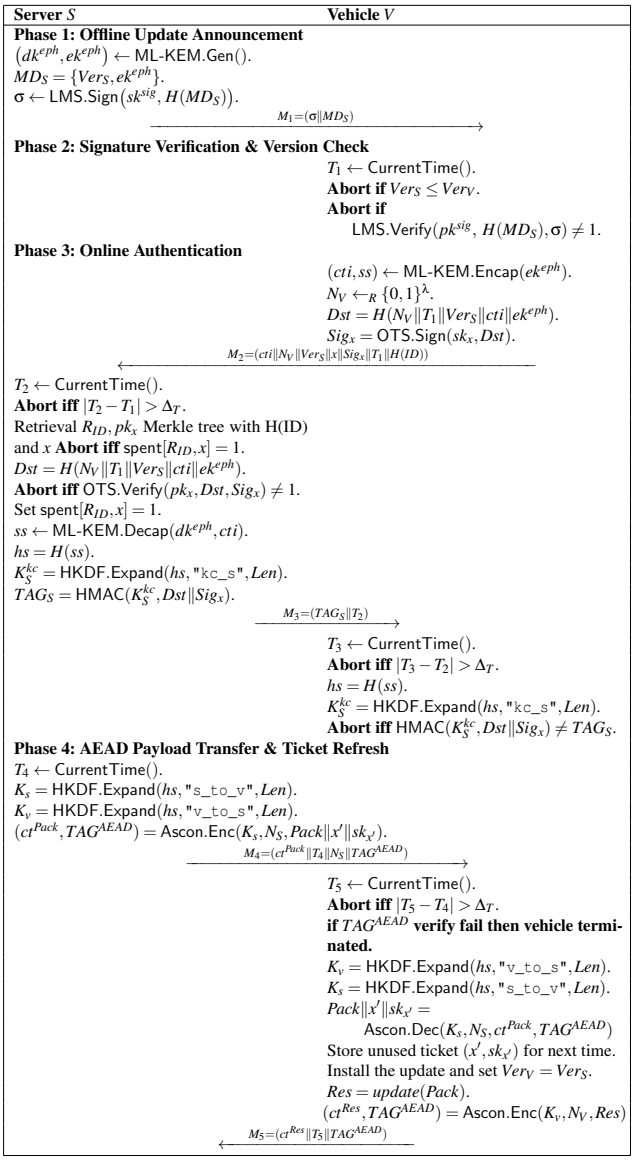


Figure 4: Quantum-safe ticket-gated server authentication

4.2 Signature Verification & Version Check

In Phases 2 (Version check and signature verification), V first generates T_1 and checks the version to avoid anti-rollback, then selects pk^{sig} as a trusted public key to verify σ under pk^{sig}. If verification fails, V rejects the update immediately.

4.3 Online Authentication

In phase 3 (Online Authentication), after accepting the signed announcement, V gets $(cti, ss) = \mathsf{ML\text{-}KEM.Encap}(ek^{eph})$ first, and creates a fresh nonce N_V. The ciphertext cti is sent to S, and only the owner of dk^{eph} can decapsulate cti and get the same shared secret ss under IND-CCA2 security. To ensure both parties agree on which session they are in, we define

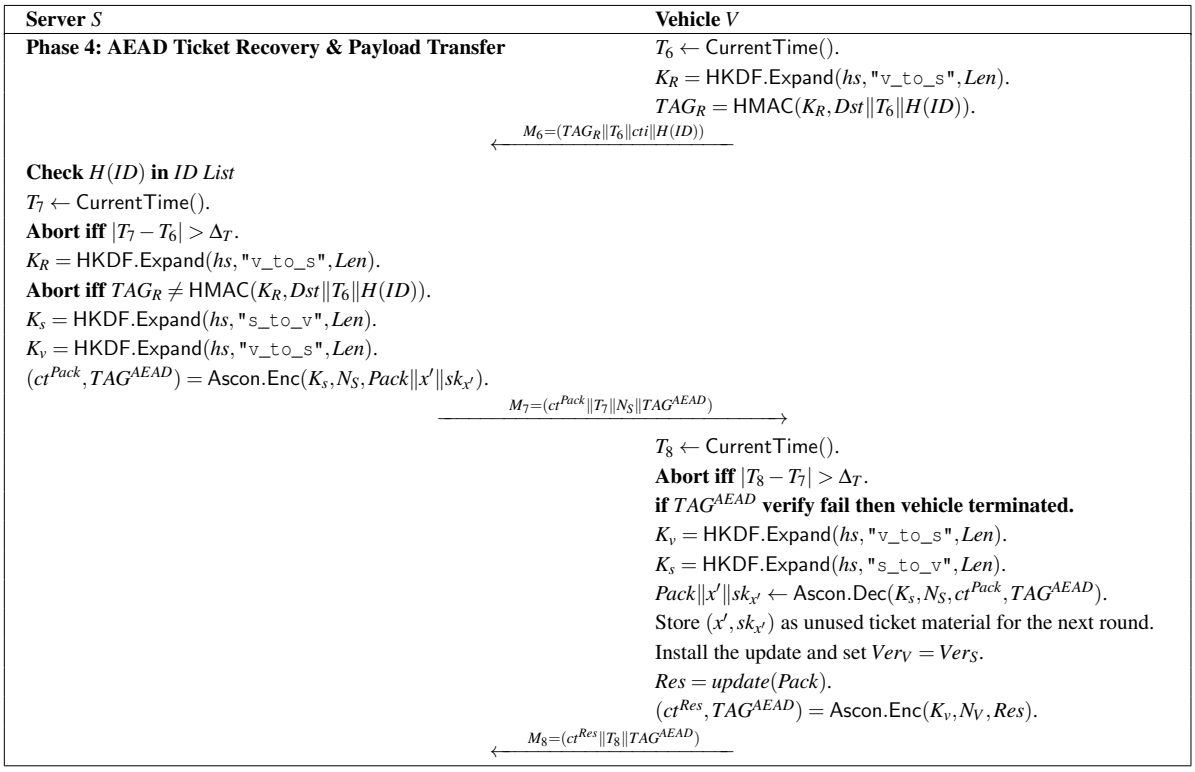


Figure 5: Ticket recovery mechanism and AEAD payload transfer

a transcript hash. For ticket proof, the V computes a one-time signature: $Sig_x = \mathsf{OTS.Sign}(sk_x, Dst)$. It sends ($V \rightarrow S$): $M_2 = (cti \| N_V \| Ver_S \| x \| Sig_x \| T_1 \| H(ID))$.

After receiving M_2, S generates T_2 and checks whether $|T_2 - T_1| \leq \Delta T$; otherwise, it aborts immediately. Before doing the whole authentication operation, the S checks the ticket in a lightweight way: (1) checks whether the specific leaf is used or not, then it reconstructs the transcript:

- Merkle proof check for (R_{ID}, x, pk_x) under R_{ID},
- one-time use check $\mathsf{spent}[R_{ID}, x] = 0$,
- S computes:

$$Dst = H(N_V \| T_1 \| Ver_S \| cti \| ek^{eph})$$

$$\mathsf{OTS.Verify}(pk_x, Dst, Sig_x) = 1$$

If any check fails, the S aborts silently. If it passes, S sets $\mathsf{spent}[R_{ID}, x] = 1$. $ss = \mathsf{ML\text{-}KEM.Decap}(dk^{eph}, cti)$, and derives $hs = H(ss)$. The S derives its confirmation keys

$$K_S^{kc} = \mathsf{HKDF\text{-}Expand}(hs, \texttt{"kc_s"}, Len)$$

If verification succeeds, the S computes $TAG_S = \mathsf{HMAC}(K_S^{kc}, Dst \| Sig_x)$, and sends:

$$S \rightarrow V: \; M_3 = (TAG_S \| T_2).$$

Finally, the V creates T_3, performs timestamp verification, and verifies TAG_S using its locally derived key material; if verification fails, it will be terminated immediately. After this, both parties share (ss, hs) and have a common transcript Dst that binds Phases 1-3. Ticket gating also improves robustness against flooding, since invalid clients are dropped before the authentication operation completes.

4.4 Payload Transfer and Ticket Refresh

In phase 4, we use the established authentication secret hs to generate the K_s and K_v to protect the data plane. Both parties derive directional application keys:

$$K_s = \mathsf{HKDF.Expand}(hs, \texttt{"s_to_v"}, Len),$$

$$K_v = \mathsf{HKDF.Expand}(hs, \texttt{"v_to_s"}, Len),$$

Next, S delivers the update over ASCON-AEAD.

$$\mathsf{Ascon.Enc}\big(K_s, N_S, \; Pack \| x' \| sk'_{x'}\big).$$

To amortize the session and avoid an extra round, S bundles the OTA payload with the next-round ticket material under the same protected channel:

$$S \rightarrow V : M_4 = (ct^{Pack} \| T_4 \| N_S \| TAG^{AEAD})$$

After decryption, V returns an install status message

$$V \rightarrow S: \; M_5 = (ct^{Res} \| T_5 \| TAG^{AEAD})$$

AEAD Ticket Recovery and Payload Transfer: For the ticket recovery (Fig. 5), V sends $M_6 = (TAG_R \| T_6 \| ct_i \| H(ID))$ to S, where TAG_R is a HMAC generated using hs. S checks H(ID) in ID list and verifies the timestamp. Then, it builds TAG_R to execute the HMAC verification to ensure that TAG_R is generated by $H(ID)$. Next, it executes the ASCON AEAD scheme with ticket-update mechanism $sk_{x'}, x'$ to V with $M_7 = (ct^{Pack} \| T_7 \| N_S \| TAG^{AEAD})$. Upon receive, V verifies timestamp and TAG^{AEAD}, then builds (K_s, K_v) to run $ASCON.Dec$ under K_s and installs it. Next, it performs $ASCON.Enc$ under K_v and returns $M_8 = (ct^{Res} \| T_8 \| TAG^{AEAD})$.

5 Security Analysis

This section presents formal security models and proofs for mutual authentication and session key agreement. Let Π_V^i and Π_S^j denote the i-th instance of vehicle V and the j-th instance of server S, respectively. The adversary $\mathcal{A}$ is a probabilistic polynomial-time (PPT) adversary that fully controls the public channel. Therefore, $\mathcal{A}$ may eavesdrop, replay, delay, block, modify, and inject public-channel messages. We give proofs using the stated security models, in which the simulator $\mathcal{S}$ interacts with the polynomial-time opponent $\mathcal{A}$.

1) Security Model for Mutual Authentication: $\mathcal{A}$ outputs a target vehicle V^* to $\mathcal{S}$ before the game starts.

Setup Phase: $\mathcal{S}$ initializes the public parameters, generates the LMS signing key pair (pk^{sig}, sk^{sig}), and prepares the signed announcement metadata $MD_S = (Ver_S, ek^{eph})$, generates the server ephemeral ML-KEM key pair (ek^{eph}, dk^{eph}), and maintains the server-side ticket state $\mathsf{spent}[R_{ID}, x]$ together with the per-vehicle ticket records indexed by $H(ID)$.

Training: $\mathcal{A}$ can query the following oracles to $\mathcal{S}$:

- **Registration:** On input a vehicle identity ID, $\mathcal{S}$ simulates the secure provisioning phase and bootstrap ticket issuance for that vehicle.
- **Authentication:** On input a registered vehicle V, $\mathcal{S}$ returns the public messages exchanged during an honest execution of the proposed protocol for that vehicle.

Challenge: $\mathcal{A}$ sends challenges to $\mathcal{S}$ by impersonating the legal server S toward the target vehicle V^* (Theorem 1) or by impersonating a legal V_i toward the honest S (Theorem 2).

Guess: $\mathcal{S}$ uses the final forgery generated by $\mathcal{A}$ to contradict the corresponding primitive or the required hidden ticket-binding in the proposed protocol.

Theorem 1 (Server Impersonation)**.** *If the LMS is EUF-CMA-safe, then our protocol is secure against server impersonation.*

Proof. Assume that $\mathcal{A}$ is a PPT attacker with a non-negligible advantage against the proposed protocol. Then one can construct a simulator $\mathcal{S}$ that breaks the EUF-CMA security of LMS with non-negligible probability.

1) Setup Phase: $\mathcal{S}$ receives the LMS public verification key pk^{sig} from the LMS challenger. Then, $\mathcal{S}$ sets pk^{sig} as the system public key, prepares the remaining public parameters of the proposed protocol, and sends them to $\mathcal{A}$.

2) Training: $\mathcal{A}$ may query the registration and authentication oracles. $\mathcal{S}$ answers all registration queries consistently. For each honest announcement, $\mathcal{S}$ queries the LMS signing oracle on $H(MD_S)$ and returns the valid announcement

$$M_1 = (\sigma \| MD_S).$$

Hence, $\mathcal{A}$ may observe valid server-originated announcements, but the signing key sk^{sig} is never revealed.

3) Challenge: To make the target vehicle V^* accept a forged server-originated announcement, $\mathcal{A}$ must output

$$M_1^* = (\sigma^* \| MD_S^*)$$

such that $\mathsf{LMS.Verify}\big(pk^{sig}, H(MD_S^*), \sigma^*\big) = 1$. If M_1^* is accepted by the target vehicle and $H(MD_S^*)$ was not queried by the signing oracle before the signing, then $\mathcal{A}$ has produced a valid LMS forgery.

4) Guess: $\mathcal{S}$ outputs $\big(H(MD_S^*), \sigma^*\big)$ to the LMS challenger. Hence, if $\mathcal{A}$ succeeds with non-negligible probability, then $\mathcal{S}$ breaks the EUF-CMA security of LMS with non-negligible probability. Since the LMS is EUF-CMA secure, the proposed protocol is secure against server impersonation. ■

Theorem 2 (Vehicle Impersonation)**.** *If the challenged unused ticket remains hidden, the OTS scheme is EUF-CMA secure, and the Merkle authentication structure remains binding under the adopted hash function. The proposed protocol is secure against vehicle impersonation.*

Proof. Assume that $\mathcal{A}$ is a PPT attacker with a non-negligible advantage against the proposed protocol. Then one shows that the ticket-gating binding of the proposed protocol is violated with non-negligible probability.

1) Setup Phase: $\mathcal{S}$ prepares the public parameters, the server-side ticket records, the per-vehicle Merkle root R_{ID}, and the state table $\mathsf{spent}[R_{ID}, x]$ of the proposed protocol. For the target vehicle V^*, the corresponding challenged unused ticket secret sk_x^* remains hidden.

2) Training: $\mathcal{A}$ may issue the stated registration and authentication queries and may observe valid public transcripts generated by honest executions. Therefore, $\mathcal{A}$ may obtain the public values contained in protocol transcripts, including

$$M_1,\ ct_i,\ N_V,\ Ver_S,\ x,\ T_1,\ H(ID),\ Dst.$$

However, the challenged unused ticket sk_x^* is not revealed.

3) Challenge: $\mathcal{A}$ sends a forged request to the honest server:

$$M_2^* = (ct_i^* \| N_V^* \| Ver_S^* \| x^* \| Sig_{x^*} \| T_1^* \| H(ID^*)).$$

On getting M_2^*, the server checks the freshness window and locates the corresponding ticket tree by $H(ID^*)$. It then finds

$$Dst^* = H(N_V{}^* \,\|\, T_1{}^* \,\|\, Ver_S{}^* \,\|\, ct_i{}^* \,\|\, ek^{eph}),$$

and retrieves the stored ticket record corresponding to $(H(ID^*), x^*)$, and finally verifies $\mathsf{spent}[R_{ID^*}, x^*] = 0$, and

$$\mathsf{Merkle.Verify}(R_{ID^*}, x^*, leaf_{x^*}, path_{x^*}) = 1$$

$$\mathsf{OTS.Verify}(pk_{x^*}, Dst^*, Sig_{x^*}) = 1.$$

Thus, if $\mathcal{A}$ makes the server accept a forged request, then at least one of the following must occur:

1. $\mathcal{A}$ produces a valid OTS forgery on Dst^* under pk_{x^*};
2. $\mathcal{A}$ constructs a false Merkle authentication that still verifies under R_{ID^*};
3. $\mathcal{A}$ reuses an already accepted ticket.

4) Guess: The first event contradicts the EUF-CMA security of the OTS scheme. The second event contradicts the binding property of the Merkle authentication structure under the adopted hash function. The server-side state transition prevents the third event

$$\mathsf{spent}[R_{ID}, x] : 0 \to 1 \tag{2}$$

after one successful acceptance. Therefore, if $\mathcal{A}$ nevertheless makes the server accept a forged request, $\mathcal{A}$ must violate either OTS unforgeability or the Merkle/hash binding with non-negligible probability. Given these assumptions, the proposed protocol is secure against vehicle impersonation. ■

Theorem 3 (Session Key Indistinguishability)**.** *If ML-KEM is IND-CCA2 secure, $H(\cdot)$ behaves as a secure key-compression function, and HKDF behaves as a secure pseudorandom key-derivation function, then no PPT adversary can distinguish the derived session keys from random values with non-negligible advantage.*

Proof. Assume that $\mathcal{A}$ is a PPT attacker with a non-negligible advantage against the proposed protocol. Then one can construct a simulator $\mathcal{S}$ that breaks the security of the underlying key-establishment chain with non-negligible probability.

1) Setup Phase: $\mathcal{S}$ receives the challenge ML-KEM public key and embeds it as the server ephemeral public key ek^{eph} in the public parameters of our protocol. Then, $\mathcal{S}$ prepares the remaining public parameters and sends them to $\mathcal{A}$.

2) Training: $\mathcal{A}$ may issue the stated registration and authentication queries and may observe valid public transcripts generated by honest executions. However, the challenged decapsulation key dk^{eph}, the challenged shared secret ss^*, and the corresponding derived key material remains hidden.

3) Challenge: $\mathcal{S}$ runs an honest target execution and returns the public challenge transcript. Let

$$(ct_i^*, ss^*) = \mathsf{ML\text{-}KEM.Encap}(ek^{eph}),$$

and let $hs^* = H(ss^*)$,

$$K_S^{kc*} = \mathsf{HKDF.Expand}(hs^*, \texttt{"kc_s"}, Len),$$

$$K_v^* = \mathsf{HKDF.Expand}(hs^*, \texttt{"v_to_s"}, Len),$$

$$K_s^* = \mathsf{HKDF.Expand}(hs^*, \texttt{"s_to_v"}, Len).$$

If $\mathcal{A}$ distinguishes the real challenge-session key material from random with non-negligible probability, then at least one of the following must hold:

1. $\mathcal{A}$ distinguishes the challenged ML-KEM shared secret from random;
2. $\mathcal{A}$ distinguishes $hs^* = H(ss^*)$ from a random value;
3. $\mathcal{A}$ distinguishes the HKDF-expanded keys from independent random strings.

4) Guess: $\mathcal{S}$ uses the final guess made by $\mathcal{A}$ to contradict the IND-CCA2 of ML-KEM or the PRF-style security of HKDF. Hence, the session keys derived in our protocol are computationally indistinguishable from random values. ■

5.1 Informal Security Analysis

Recall that the authentication transcript digest is $Dst = H(N_V \,\|\, T_1 \,\|\, Ver_S \,\|\, ct_i \,\|\, ek^{eph})$.

Proposition 1 (Authentication)**.** *Our protocol holds mutual authentication between the OEM server S and the vehicle V.*

Proof. First, V authenticates S by checking $\mathsf{LMS.Verify}(pk^{sig}, H(MD_S), \sigma) = 1$. Hence, a fake server cannot forge a valid announcement without breaking the LMS. Next, after the online exchange, V accepts only if $TAG_S = \mathsf{HMAC}(K_S^{kc}, Dst \,\|\, Sig_x)$, where K_S^{kc} is derived from the shared secret ss. Thus, V confirms that the sender knows the correct session-dependent secret and that the response is bound to the same ticket-authenticated session context. On the other hand, S authenticates V only if $\mathsf{spent}[R_{ID}, x] = 0$, $\mathsf{Merkle.Verify}(R_{ID}, x, leaf_x, path_x) = 1$, and $\mathsf{OTS.Verify}(pk_x, Dst, Sig_x) = 1$. Since only a legitimate vehicle possesses the unused one-time secret key sk_x, a forged requester cannot produce a valid Sig_x on Dst without breaking the OTS scheme. Therefore, both sides authenticate each other, thereby achieving mutual authentication. ■

Proposition 2 (Confidentiality)**.** *Our protocol preserves the confidentiality of the OTA payload over the public channel.*

Proof. After the vehicle receives the signed announcement, it computes $(ct_i, ss) = \mathsf{ML\text{-}KEM.Encap}(ek^{eph})$, while the server computes $ss = \mathsf{ML\text{-}KEM.Decap}(dk^{eph}, ct_i)$. Then both sides derive $hs = H(ss)$,

$$K_s = \mathsf{HKDF.Expand}(hs, \texttt{"s_to_v"}, Len),$$

$$K_v = \mathsf{HKDF.Expand}(hs, \texttt{"v_to_s"}, Len).$$

The OTA payload is protected by

$$(ct^{Pack}, TAG^{AEAD}) = \mathsf{Ascon.Enc}(K_s, N_S, Pack \| x' \| sk_{x'}).$$

Hence, without recovering the hidden shared secret ss, $\mathcal{A}$ cannot derive the AEAD key and cannot reveal the OTA package contents, including the next-round ticket material $(x', sk_{x'})$. Therefore, payload confidentiality is preserved. ■

Proposition 3 (Integrity)**.** *The proposed protocol preserves the integrity of the announcement, the online authentication messages, and the OTA payload.*

Proof. LMS protects the announcement's integrity:

$$\sigma = \mathsf{LMS.Sign}(sk^{sig}, H(MD_S)).$$

Thus, any modification to MD_S causes

$$\mathsf{LMS.Verify}(pk^{sig}, H(MD_S), \sigma) = 0.$$

The ticket proof and key-confirmation tag protect the integrity of online authentication. In particular, $\mathsf{OTS.Verify}(pk_x, Dst, Sig_x) = 1$ binds $(N_V, T_1, Ver_S, ct_i, ek^{eph})$ to the vehicle ticket, and

$$TAG_S = \mathsf{HMAC}(K_S^{kc}, Dst \| Sig_x)$$

binds the server response to the same authenticated session context. Hence, tampering with the online messages changes Dst or invalidates the tag/signature verification.

Finally, the payload integrity is protected by ASCON-AEAD. If an adversary modifies the ciphertext, nonce, or tag, the receiver-side AEAD verification fails. ■

Proposition 4 (Freshness and Replay Resistance)**.** *Our protocol resists replay attacks and preserves message freshness.*

Proof. The protocol binds fresh values into the transcript

$$Dst = H(N_V \,\|\, T_1 \,\|\, Ver_S \,\|\, ct_i \,\|\, ek^{eph}).$$

Since N_V is freshly generated by the vehicle, ct_i is freshly generated by ML-KEM encapsulation, and T_1 is checked under the freshness window at the server, different sessions produce different transcript digests. In addition, the server enforces

$$\mathsf{spent}[R_{ID}, x] = 0$$

before accepting a ticket. After one successful acceptance, the state changes to $\mathsf{spent}[R_{ID}, x] = 1$. Therefore, replaying old online messages or reusing an old ticket cannot make the server accept them again. Hence, the proposed protocol preserves freshness and resists replay attacks. ■

Proposition 5 (Anti-Rollback)**.** *The proposed protocol resists rollback attacks.*

Proof. The advertised version value Ver_S is included in the signed announcement. Hence, an attacker cannot replace the version field without invalidating the LMS signature on $H(MD_S)$. Moreover, upon receiving the announcement, the vehicle compares the advertised version with its locally stored maximum-accepted version, Ver_V. The vehicle proceeds only if $Ver_S > Ver_V$. If the received version is at least as old as the vehicle aborts. After a successful installation, the vehicle updates $Ver_V \leftarrow Ver_S$. Thus, replaying an old but previously valid OTA package still cannot force the vehicle to downgrade. Hence, the protocol is resistant to rollback attacks. ■

Proposition 6 (Quantum Resistance)**.** *The proposed protocol provides quantum-resistant protection, assuming the security of the primitives it uses.*

Proof. The protocol does not rely on RSA, ECC, or Diffie–Hellman in its core authentication and key-establishment path. Instead, it uses LMS for authenticated announcement and ML-KEM for shared-secret establishment. Thus, the protocol avoids the classical public-key assumptions that are directly threatened by Shor's algorithm, and achieves quantum-resistant OTA protection. ■

Proposition 7 (Forward Secrecy)**.** *Assume that each release uses a fresh ephemeral ML-KEM key pair (ek^{eph}, dk^{eph}) and that the old dk^{eph} is erased after use. Then the proposed protocol provides forward secrecy against later compromise of the long-term authentication key.*

Proof. The long-term key sk^{sig} is used only to sign $H(MD_S)$, while traffic keys are derived from the ephemeral shared secret

$$ss = \mathsf{ML\text{-}KEM.Decap}(dk^{eph}, ct_i).$$

The ticket secret sk_x is used only for one-time vehicle authentication on Dst and is not used as input to the traffic-key derivation. Hence, later compromise of long-term authentication material alone does not reveal prior session keys.

If each release uses a fresh (ek^{eph}, dk^{eph}) and the old dk^{eph} is erased after the release window, then compromise of the long-term authentication key in the future does not reveal previously established traffic keys K_S^{kc}, K_s, K_v. Therefore, the protocol provides forward secrecy against the later compromise of the long-term authentication key. ■

6 Empirical Performance Analysis

Experimental Setting: Our experimental testbed emulates the vehicle-side TCU using a Raspberry Pi 3 Model B+, which comprises a 1.4 GHz Broadcom BCM2837B0 SoC and 1 GB of LPDDR2 RAM. The OEM server includes a GTX 1660 GPU, an i5-10400F processor, and 16 GB of DDR4-2400

Table 3: Online Computation Cost Comparison of Related OTA Works

Protocol	Method	Computational costs (one session)
Work [34]	PKI hybrid encryption and signature-based verification	$T_H + T_{\text{RSA.Dec}} + T_{\text{AES.Dec}} + T_{\text{RSA.Verify}}$
Work [18]	PBKDF2-based ECU-level OTA authentication	$T_{\text{PBKDF2}} + T_{\text{HKDF}} + T_{\text{HMAC}} + T_{\text{Sig.Sign}} + 2T_{\text{Sig.Verify}} + 3T_H$
Work [12]	STRIDE: CP-ABE-based secure OTA	$T_{\text{CP-ABE.Dec}} + T_{\text{HashVerify}}$
Work [20]	CP-ABE-protected OTA update with RSA signature	$T_H + T_{\text{RSA.Verify}} + T_{\text{CP-ABE.Dec}}$
Ours	LMS authentication and ticket-gating mechanism	$T_{\text{LMS.Verify}} + T_{\text{KEM.Encap}} + T_{\text{OTS.Sign}} + T_{\text{MerkleVerify}} + T_{\text{OTS.Verify}} + T_{\text{KEM.Decap}} + 4T_{\text{HKDF}} + 2T_{\text{HMAC}} + 6T_H$

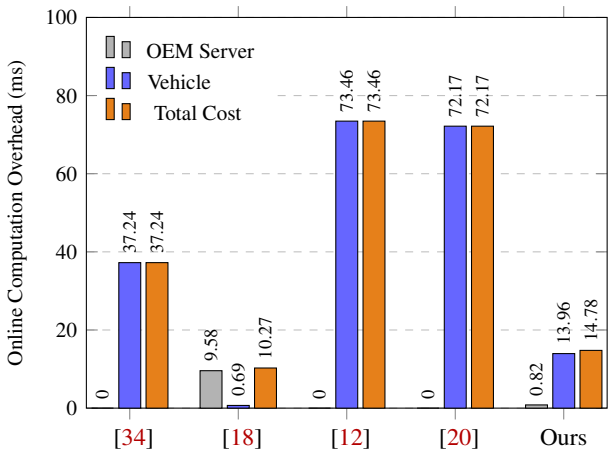


Figure 6: Comparison with related security protocols

memory. All systems utilize 32 GB Class 10 microSD cards for persistent storage and firmware image buffering.

We first compare the per-session computational cost of the proposed protocol with that of related work. We then estimate the OTA download time at fleet scale with concurrency. Finally, we measure the protocol runtime and report its relative delay as a percentage of download time.

6.1 Simulation via Raspberry Pi 3 Model B+

Comparison with Related Security Protocols: Table 3 provides a theoretical estimation of per-session computation costs of related works. All data is shown in Fig. 6 is a simulation from Raspberry Pi 3B+ and PC, comparing the per-session computational cost of our protocol with related protocols. The selected baseline paper includes mechanisms to verify and protect OTA software updates. In-vehicle secure OTA design based on multi-factor authentication, Public Key Infrastructure (PKI) hybrid encryption, STRIDE baseline, and Ciphertext-Policy Attribute-Based Encryption (CP-ABE) for software update. Our protocol achieves competitive per-session online cost while additionally supporting post-quantum mutual authentication and ticket-gated early request filtering to discard invalid requests and avoid DoS or flooding issues, simulating an average computational online overhead 0.82 ms and 13.96 ms on the server and vehicle sides.

Table 4: Comparison of Security Key Sizes (bytes)

Scheme	Private Key	Public Key
Ed25519	32	32
ECDSA P-256	32	32
Falcon	1281	897
ML-DSA (Dilithium)	2528	1312
LMS (RFC 8554)	64	60

Storage Cost: As shown in Table 4, the table compares PKC and post-quantum signatures, including Ed25519, ECDSA, Falcon, ML-DSA, and LMS. This study [3] demonstrates that most currently post-quantum signature algorithms are highly demanding in terms of storage costs for public and secret keys; however, LMS has a very small key size for both public and private keys, which is why we select it.

Security Comparison & Protocol Advantage Discussion: The selected baselines, as shown in Table 5, span different types of OTA-related work: four security-oriented schemes, three efficiency-oriented schemes, and the Uptane framework. The comparison shows that although security-oriented works generally provide better protection than efficiency-oriented ones, the Uptane framework [17], which serves as the high-level orchestrator for authentication, integrity, freshness, and anti-rollback, is also transport-agnostic. Besides, it explicitly relies on transport-layer security (TLS) or encrypted payloads to satisfy data confidentiality. As a result, it only provides a partial solution to quantum resistance (supporting signatures such as SLH-DSA since 2022). It does not provide any built-in support for Forward Secrecy, Server-Side DoS Resilience, and Low-Latency Handshake Setup. Our design addresses these gaps by separating the authentication phase from the transmission of the OTA payload. This guarantees that initial verification takes only a minimal amount of time in the event of an authentication failure. In contrast, current works, such as [34], combine RSA-based authentication with AES-protected payload transmission. If authentication fails, substantial overhead may already have been incurred due to superfluous decryption operations. Through the optimization of the V2C delivery phase (Uptane's Type (i) deployment), our protocol accomplishes additional G4 (Server-side DoS resilience) through a ticket-gating mechanism and G5 (Low-

Table 5: Security Features Comparison of Related Works

Security Property	Work [34]	Work [18]	Work [12]	Work [20]	Work [15]	Work [2]	Work [27]	Work [17]	**Ours**
Mutual Authentication	×	✓	×	×	×	×	×	✓	✓
Confidentiality	✓	✓	✓	✓	×	×	×	✓	✓
Integrity	✓	✓	✓	✓	×	✓	×	✓	✓
Freshness (Anti-Replay)	△	✓	✓	×	×	×	×	✓	✓
Anti-Rollback	×	✓	×	✓	×	×	×	✓	✓
Forward Secrecy	×	×	×	×	×	×	×	×	✓
Quantum-Resistance	×	×	×	×	×	×	×	△	✓
Server-Side DoS Resilience	×	×	×	×	×	×	×	×	✓
Low-Latency Handshake Setup	✓	✓	✓	✓	×	×	×	×	✓

(✓: considered, △: partially considered, ×: not addressed)

Table 6: Evaluation of OTA Payload Download Time (sec) and Throughput (Mbps)

Payload	**10**	**20**	**50**	**70**	**100**	**500**	**1000**
1MB	1.37 (7.31)	2.60 (7.70)	5.69 (8.80)	7.90 (8.86)	13.48 (7.42)	56.24 (8.89)	114.81 (8.71)
10MB	7.51 (13.32)	14.07 (14.22)	34.47 (14.51)	48.07 (14.56)	91.77 (10.90)	352.66 (14.18)	699.60 (14.29)
50MB	31.32 (15.97)	61.29 (16.32)	155.61 (16.07)	213.48 (16.39)	350.27 (14.27)	1550.94 (16.12)	3055.39 (16.36)
100MB	62.82 (15.92)	123.10 (16.25)	304.29 (16.43)	447.71 (15.64)	627.96 (15.92)	3028.60 (16.51)	6066.83 (16.48)
200MB	165.02 (12.12)	360.71 (11.09)	764.49 (13.08)	892.66 (15.68)	1216.56 (16.44)	6134.59 (16.30)	12027.40 (16.63)
300MB	183.11 (16.38)	358.15 (16.75)	892.17 (16.81)	1252.84 (16.76)	1797.80 (16.69)	9068.84 (16.54)	18719.52 (16.03)

Table 7: Evaluation of Proposed Protocol Consumption (ms)

Vehicles	**10**	**20**	**50**	**70**	**100**	**500**	**1000**
Ours	148	296	739	1035	1478	7390	14780

latency session setup) by reducing online exchanges, even in high-mobility environments with intermittent connectivity.

OTA Download-Time Model: As shown in Table 6, we simulate concurrent OTA payload downloads with vehicles $v \in \{10, 20, 50, 70, 100, 500, 1000\}$, where the payload size per vehicle is $s \in \{1, 10, 50, 100, 200, 300\}$ MB. The download makespan is mainly considered by the aggregate throughput TP (MB/s). Thus, the time in Table 6 can be estimated as:

$$t_{\text{OTA}} \approx \frac{v \cdot s}{TP} \tag{3}$$

Protocol Runtime Measurement: As shown in Table 7, we measure the time overhead of the proposed protocol alone, excluding the OTA payload transfer. This isolates the pure security cost and enables us to report how much of the end-to-end OTA time is spent on the protocol.

Protocol Delay Percentage: Table 8, derived from Tables 6-7, shows protocol delay of OTA download time:

$$P(\%) = 100 \cdot \frac{t_{\text{proto}}}{t_{\text{OTA}}} \tag{4}$$

Table 8: Evaluation of the Proposed Protocol Overhead as a Percentage (%) of Payload Download Time

Payload	**10**	**20**	**50**	**70**	**100**	**500**	**1000**
1MB	10.80	11.38	12.99	13.10	10.96	13.14	12.87
10MB	1.97	2.10	2.14	2.15	1.61	2.10	2.11
50MB	0.47	0.48	0.47	0.48	0.42	0.48	0.48
100MB	0.24	0.24	0.24	0.23	0.24	0.24	0.24
200MB	0.09	0.08	0.10	0.12	0.12	0.12	0.12
300MB	0.08	0.08	0.08	0.08	0.08	0.08	0.08

where t_{proto} is from Table 7 and t_{OTA} is from Table 6. For example, with 100 vehicles and a 100 MB payload, $t_{\text{proto}} = 1.47$ sec and $t_{\text{OTA}} = 627.96$ sec, so $P = 0.24\%$. Once the payload exceeds 100 MB, the percentage drops to a negligible level.

7 Conclusion and Future Work

In this paper, we present a quantum-safe server authentication protocol for vehicular OTA updates, using LMS for announcement authenticity and an IND-CCA2-secure KEM to establish session keys with low RTT. To reduce OEM server pressure under bursty requests, we add a one-time ticket-gating mechanism that allows the server to filter unauthorized requests before running the expensive operation. We implement the protocol on a Raspberry Pi 3 Model B+ testbed and measure both protocol execution time and OTA payload download time. Results show that the protocol overhead becomes neg-

ligible as the payload grows, dropping to about 0.10–0.12% for 200–300 MB updates in our settings, which suggests the design is practical for real deployments where updates are typically hundreds of MB. As future work, we plan to evaluate in more realistic ECU and network conditions and build a more comprehensive deployment-level simulation.

Acknowledgments

This work was supported in part by the National Science and Technology Council (NSTC) under Grants NSTC 114-2221-E-110-042-MY3 and 114-2634-F-110-001-MBK, and assisted by the Cryptology and Network Security Laboratory at National Sun Yat-sen University (CANSEC-Lab@NSYSU). The authors would also like to thank the anonymous reviewers and our paper shepherd for their valuable feedback and guidance during the revision process.

Artifact Availability

https://doi.org/10.5281/zenodo.19682385

References

[1] Rafael Abreu, Frederico Branco, Manuel JCS Reis, and Carlos Serôdio. Cybersecurity in connected and autonomous vehicles: A systematic review of automotive security. *IEEE Access*, 2025.

[2] Nicholas Ayres, Lipika Deka, and Daniel Paluszczyszyn. Continuous automotive software updates through container image layers. *Electronics*, 10(6):739, 2021.

[3] Gustavo Banegas, Koen Zandberg, Emmanuel Baccelli, Adrian Herrmann, and Benjamin Smith. Quantum-resistant software update security on low-power networked embedded devices. In *International Conference on Applied Cryptography and Network Security (ACNS)*, pages 872–891. Springer, 2022.

[4] Abir Bazzi, Adnan Shaout, and Di Ma. MT-SOTA: A merkle-tree-based approach for secure software updates over the air in automotive systems. *Applied Sciences*, 13(16), 2023.

[5] Meriem Benyahya, Anastasija Collen, and Niels Alexander Nijdam. Analyses on standards and regulations for connected and automated vehicles: Identifying the certifications roadmap. *Transportation Engineering*, 14:100205, 2023.

[6] Vaishali Bhatia and KR Ramkumar. An efficient quantum computing technique for cracking RSA using Shor's algorithm. In *2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA)*, pages 89–94, 2020.

[7] Jiseong Chung, Sungkwon Park, and Songnam Hong. Design of an automotive OTA system with zonal gateways and PQC-hybrid TLS. In *2025 IEEE/IEIE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pages 1–6, 2025.

[8] David A. Cooper, Daniel C. Apon, Quynh H. Dang, Michael S. Davidson, Morris J. Dworkin, and Carl A. Miller. NIST Special Publication 800-208: Recommendation for stateful hash-based signature schemes, 2020.

[9] Gianpiero Costantino, Marco De Vincenzi, and Ilaria Matteucci. A comparative analysis of UNECE WP. 29 R155 and ISO/SAE 21434. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 340–347, 2022.

[10] Popy Das, Md. Rashid Al Asif, Sohely Jahan, Kawsar Ahmed, Francis M. Bui, and Rahamatullah Khondoker. STRIDE-based cybersecurity threat modeling, risk assessment and treatment of an in-vehicle infotainment system. *Vehicles*, 6(3):1140–1163, 2024.

[11] Tiago M Fernández-Caramés. From pre-quantum to post-quantum iot security: A survey on quantum-resistant cryptosystems for the internet of things. *IEEE Internet of Things Journal*, 7(7):6457–6480, 2019.

[12] Anirban Ghosal, Suman Halder, and Mauro Conti. Secure over-the-air software update for connected vehicles. *Computer Networks*, 218:109394, 2022.

[13] Tian Guan, Yi Han, Nan Kang, Ningye Tang, Xu Chen, and Shu Wang. An overview of vehicular cybersecurity for intelligent connected vehicles. *Sustainability*, 14(9):5211, 2022.

[14] Subir Halder, Amrita Ghosal, and Mauro Conti. Secure over-the-air software updates in connected vehicles: A survey. *Computer Networks*, 178:107343, 2020.

[15] Roland Herberth, Sidney Körper, Tim Stiesch, Frank Gauterin, and Oliver Bringmann. Automated scheduling for optimal parallelization to reduce the duration of vehicle software updates. *IEEE Transactions on Vehicular Technology*, 68(3):2921–2933, 2019.

[16] Seonghoon Jeong, Minsoo Ryu, Hyunjae Kang, and Huy Kang Kim. Infotainment system matters: Understanding the impact and implications of in-vehicle infotainment system hacking with automotive grade linux. In *13th ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 201–212, 2023.

[17] Trishank Karthik, Akan Brown, Sebastien Awwad, Damon McCoy, Russ Bielawski, Cameron Mott, Sam Lauzon, André Weimerskirch, and Justin Cappos. Uptane: Securing software updates for automobiles. In *Intl. Conf. on Embedded Security in Car*, volume 11, 2016.

[18] Hyeonseong Kim and Sanghoon Jeon. Multi-factor authentication for in-vehicle secure OTA protocol. *IEEE Access*, 13:208626–208641, 2025.

[19] Ana Kovacevic and Nenad Gligoric. Enhancing security of automotive ota firmware updates via decentralized identifiers and distributed ledger technology. *Electronics*, 13(23):4640, 2024.

[20] Michele La Manna, Luigi Treccozzi, Pericle Perazzo, Sergio Saponara, and Gianluca Dini. Performance evaluation of attribute-based encryption in automotive embedded platform for secure software over-the-air update. *Sensors*, 21(2), 2021.

[21] Ruiqi Lu, Guoqi Xie, Lida Huang, Xinzhong Liu, Wei Xu, Jianmei Lei, and Junqiang Jiang. LigSecOTA: Lightweight over-the-air (OTA) software updates with integrated security. *IEEE Transactions on Dependable and Secure Computing*, pages 1–15, 2025.

[22] Dimitris Mbakoyiannis, Othon Tomoutzoglou, and George Kornaros. Secure over-the-air firmware updating for automotive electronic control units. In *34th ACM/SIGAPP Symposium on Applied Computing (SAC)*, pages 174–181, 2019.

[23] NIST. Module-lattice-based key-encapsulation mechanism standard. FIPS 203, 2024. Available: https://doi.org/10.6028/NIST.FIPS.203.

[24] Ugochukwu Orji and CH Ugwuishiwu. An overview of quantum cryptography and shor's algorithm. *International Journal of Advanced Trends in Computer Science and Engineering*, 2020.

[25] Sebastian Paul, Patrik Scheible, and Friedrich Wiemer. Towards post-quantum security for cyber-physical systems: Integrating PQC into industrial M2M communication1. *Journal of Computer Security*, 30(4):623–653, 2022.

[26] Christian Plappert and Andreas Fuchs. Secure and lightweight ecu attestations for resilient over-the-air updates in connected vehicles. In *39th Annual Computer Security Applications Conference (ACSAC)*, pages 283–297, 2023.

[27] Muhammad Hamza Sarwar, Munam Ali Shah, Muhammad Umair, and Syad Hassnain Faraz. Network of ecus software update in future vehicles. In *2019 IEEE 25th International Conference on Automation and Computing (ICAC)*, pages 1–5, 2019.

[28] Roshan Sedar, Charalampos Kalalas, Francisco Vázquez-Gallego, Luis Alonso, and Jesus Alonso-Zarate. A comprehensive survey of V2X cybersecurity mechanisms and future research paths. *IEEE Open Journal of the Communications Society*, 4:325–391, 2023.

[29] Jaewan Seo, Jiwon Kwak, and Seungjoo Kim. Development for high-assurance software update management system complying with UN R156. *IEEE Access*, 12:135811–135830, 2024.

[30] Kyung-Ah Shim. A survey on post-quantum public-key signature schemes for secure vehicular communications. *IEEE Transactions on Intelligent Transportation Systems*, 23(9):14025–14042, 2021.

[31] Ali Shoker, Fernando Alves, and Paulo Esteves-Verissimo. Scaiota: Scalable secure over-the-air software updates for vehicles. In *2023 IEEE 42nd International Symposium on Reliable Distributed Systems (SRDS)*, pages 151–161, 2023.

[32] Junko Takahashi, Makoto Iwamura, and Masashi Tanaka. Security threat analysis of automotive infotainment systems. In *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, pages 1–7, 2020.

[33] Meltem Sönmez Turan, Kerry McKay, Donghoon Chang, Jinkeon Kang, and John Kelsey. Ascon-based lightweight cryptography standards for constrained devices. In *NIST Special Publication (SP) NIST SP 800–232 ipd*. National Institute of Standards and Technology, 2024.

[34] Jingxuan Wang, Rui Chen, Haiou Wen, Yaxiang Chen, Qianping Hou, and Wendong Duan. A secure and efficient vehicle ota software update system: Design and validation. In *2024 IEEE 6th International Conference on Electronic Engineering and Informatics (EEI)*, pages 1176–1180, 2024.

[35] Jun Yeon Won, Wenzhuo Wang, Keith Redmill, and Zhiqiang Lin. CarPlay at risk: Unveiling security threats of third-party infotainment adapters. In *3rd USENIX Symposium on Vehicle Security and Privacy (VehicleSec 25)*, pages 143–159, 2025.

[36] Minfeng Wu and Wen Chen. Forecast of electric vehicle sales in the world and china based on PCA-GRNN. *Sustainability*, 14(4):2206, 2022.

[37] Sadia Yeasmin and Anwar Haque. A multi-factor authenticated blockchain-based ota update framework for connected autonomous vehicles. In *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*, pages 1–6, 2021.



USENIX VehicleSec '26 Artifact Appendix: Beyond Classical PKI: A Quantum-Safe Authentication Framework for Automotive OTA Updates

Wei-Lun Lin
National Sun Yat-sen University

Ting-Yu Chen
National Sun Yat-sen University

Arijit Karati
National Sun Yat-sen University
arijit.karati@mail.cse.nsysu.edu.tw

A Artifact Appendix

The artifact considers two PowerShell sessions on a single device: shell 1 as a server and shell 2 as a vehicle. The entire experimental process is divided into three segments: (1) **Set-up**, open two PowerShell *with Administrator and all shells into WSL* in the PC, (2) **Install dependencies one time on any shell**, and (3) **Replicate the entire experimental process** and simulate the recovery mechanism. We design a drop Ascon, and when executing drop Ascon, one can see a packet-loss message, execute the recovery mechanism, and receive the payload and the next ticket $\langle sk_x, x \rangle$.

A.1 Abstract

We provide hardware and software dependencies, setup and installation, a basic test description, and demonstrate that our artifacts match our paper claims. We also provide instructions on what these artifacts need in README.txt.

A.2 Description & Requirements

Description & Requirements. We define shell 1 (wsl) as the OEM server and shell 2 (wsl) as the vehicle side. Both the OEM server and the vehicle side operate on a single host machine. The project is located at `C:\VS_ticket` and accessed in WSL as `/mnt/c/VS_ticket`.

A.2.1 Security, Privacy, and Ethical Concerns

This artifact should be executed only in a private local network. Evaluators should not expose the ports used to the public network or run the artifact on a public real IP. This prototype should not be connected to any production system, real vehicle, or other real-world operational environment.

A.2.2 How to Access

The evaluated artifact is available as an archived Zenodo record with a stable DOI. Our artifact's DOI is: https://doi.org/10.5281/zenodo.19682385. This URL includes the artifacts zip file and the demo video zip.

A.2.3 Badges

Artifacts Available, Artifacts Functional, Results Reproduced.

A.2.4 Hardware Dependencies

We using single PC to simulate our artifacts, a Windows 10 64-bit with an Intel core i5-10400f CPU, and 16 GB RAM.

A.2.5 Software Dependencies

(1) For the wsl setting, open PowerShell in Windows with administrator, run **wsl –install**, and reboot the system (2), Feel free to choose any shell (shell 1, shell 2) to install software library only once. The required third-party libraries are `Ascon-c`, `hash-sigs`, `liboqs`, and `liboqs-install`. The artifact is compiled once in the wsl environment.

A.2.6 Benchmarks

None.

A.3 Set-up

Set-up. Shell 1 as an OEM server and shell 2 as a vehicle.

A.3.1 Installation

Extracting the `VS_ticket.zip` archive into `C:\`, will automatically create the `C:\VS_ticket` directory. Then, install the required system packages and dependencies.

A.3.2 Basic Test

The basic test includes registration, authentication, Ascon payload phase.

Note: Instruction *–vehicle-id 123000*, if you registered vehicle ID 123000, you should use the same ID in Step 2 (online

authentication), Step 3 (payload transmission) on both side. Optional (Recovery Mechanism) also provided. All *ip* in the artifacts are setting same: 127.0.0.1.

Step 1: Registration. On the server side (PC/wsl), run:

```
./build/registration_server \
  --state-prefix local_eval \
  --host 127.0.0.1 \
  --port 6000 \
  --versions 60 \
  --once
```

On the vehicle side (shell 2), run:

```
./build/registration_vehicle \
  --state-prefix local_eval \
  --server-ip 127.0.0.1 \
  --server-port 6000 \
  --vehicle-id 123000
```

Expect: You will type the height of the Merkle tree on Shell 1 and see the Registration ID on both sides.

Step 2: Online Authentication, Version 1. On the server side (shell 1), run:

```
./build/control_server \
  --state-prefix local_eval \
  --host 127.0.0.1 \
  --port 6000 \
  --version 1 \
  --delta-seconds 300 \
  --once
```

On the vehicle side (shell 2), run:

```
./build/control_vehicle \
  --state-prefix local_eval \
  --server-ip 127.0.0.1 \
  --server-port 6000 \
  --version 1 \
  --delta-seconds 300 \
  --vehicle-id 123000
```

Expect: You will see the online authentication phase.

Step 3: Payload Transmission, Version 1. On the server side (shell 1), execute:

```
./build/payload_server \
  --state-prefix local_eval \
  --host 127.0.0.1 \
  --port 6000 \
  --version 1 \
  --payload-bytes 1048576 \
  --once
```

On the vehicle side (shell 2), execute:

```
./build/payload_vehicle \
  --state-prefix local_eval \
  --server-ip 127.0.0.1 \
  --server-port 6000 \
  --version 1 \
  --vehicle-id 123000
```

Expect: You will see the Ascon encryption transmission.

A.4 Evaluation workflow

It demonstrates various phases: (1) *Registration* (2) *Online authentication* (3) *Payload transmission & ticket refresh* and when packet loss happen in (3) are as follow: (2) *Online authentication* (3) *Drop Ascon payload & ticket refresh* (4) *Recovery Ascon payload & ticket refresh*.

A.4.1 Major claims

C1: The artifact achieves registration phase, authentication phase, payload transmission. (E1).

C2: It simulates the recovery mechanism successfully. (E2).

C3: The multiple times simulation. (E3).

A.4.2 Experiments

(E1) [Proposed protocol] *[20 human minutes + one PC]–multiple times OTA Workflow, Version 1:*
⋆ **Preparation:** Open two powershell with wsl.
⋆ **Execution:** Step 1: Registration, Step 2: Online authentication, and Step 3: Payload dispatch & ticket refresh.
⋆ **Results:** It shows that the server takes nearly 0.83 ms, and the vehicle takes 3.23 ms in a 20-time multiple test.

(E2) [Recovery mechanism] *[10 human minutes + one PC]–Dropped Payload and Recovery Workflow, Version 2:* We show the process in a demo video accessed from Zenodo.
⋆ **Preparation:** Open two powershell with wsl.
⋆ **Execution:** The workflow is online authentication, drops payloads, and invokes the recovery mechanism.
⋆ **Results:** We demonstrate that even if we deliberately drop the payload, the recovery mechanism still finishes the payload transmission with the next-ticket sk_x, x.

(E3) [Repeat simulation of proposed protocol] [5 human-minutes + one PC]: Multiple-run evaluation.
⋆ **Preparation:** Open two powershell with wsl.
⋆ **Execution:** Full steps are as follows: Step 1: Registration. In this step, both Shell 1 and Shell 2 need to input the *same vehicle ID and IP* in the instruction.
⋆ **Results:** You will expect to see multiple results on the screen and finally get the averaged data.

A.5 Notes on Reusability

A reset operation is provided to clean the registration list. Running the commands below on Shell 1 and Shell 2 removes the registration-related state and restores a clean environment.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing, and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.

Short: VIN2VICTIM: From Vehicle Identification to Targeted Tracking via Data Broker Exploitation

Hannaneh B. Pasandi
University of California, Berkeley
h.pasandi@berkeley.edu

Mohammad Sepahi
Rivian, Inc. and Old Dominion University
msepa001@odu.edu

Abstract

Connected vehicles relay GPS telemetry to manufacturers, who routinely sell it to data brokers. We present VIN2VICTIM, an attack pipeline that converts a single Vehicle Identification Number into targeted physical tracking through four composable stages: owner deanonymization, trajectory re-identification, home and workplace inference, and departure prediction. Three structural properties make vehicles qualitatively worse than phones for privacy. VINs are immutable lifelong correlators, consent is coarse and vehicle-scoped, and VINs bridge the digital-to-physical gap. We evaluate the pipeline on three public GPS datasets (T-Drive, 18 taxis; GeoLife, 11 users; Porto, 50 taxis) spanning 2008 to 2014 and test its resilience to broker-grade temporal coarsening. Two spatio-temporal points re-identify 96 to 99% of taxi traces and 82% of personal GPS traces (N=11 to 50). Home clusters are produced for 92 to 100% of users; for the 11 GeoLife users with multi-modal traces, 10 of 11 clusters fall inside an OpenStreetMap residential polygon. The attack survives 60-minute sampling intervals despite 91% data reduction, and geo-indistinguishability at ε=0.1 still preserves 74 to 78% unicity.

1 Introduction

Connected vehicles continuously relay GPS telemetry to OEM cloud platforms, which in turn sell this data to third-party brokers. Recent enforcement actions confirm that this data flow is both pervasive and poorly governed. On January 14, 2026, the Federal Trade Commission finalized a 20-year order against General Motors and OnStar for collecting drivers' precise geolocation every three seconds and selling it to data brokers including LexisNexis and Verisk, following allegations first announced in January 2025 [10, 11]. Separately, the Texas Attorney General filed suit against Allstate and its subsidiary Arity in January 2025 for operating what the complaint calls "the world's largest driving behavior database," harvesting data from 45 million Americans through SDKs embedded in apps such as Life360 and GasBuddy [17]. In December 2024, a Volkswagen subsidiary leaked the precise GPS coordinates of 800,000 electric vehicles across Europe [6]. These are not isolated failures. Thompson and Warzel's investigation for The New York Times demonstrated that a single commercial phone-location dataset covering 12 million devices could track individuals to their homes, workplaces, and private meetings [27]; the vehicle ecosystem replicates and amplifies this same threat through structurally weaker privacy controls. Upstream Security's 2025 automotive cybersecurity report finds that data and privacy incidents now account for 60% of all automotive cybersecurity events [29]. These cases together establish the legal premise that vehicle location data flowing to brokers is harmful. What they do not establish is what an adversary can actually do with that data once obtained. We close that gap.

The instinct is to lump this problem with phone tracking. That comparison obscures something important. Phones have privacy mitigations that vehicles structurally lack.[1] Apple and Google spent a decade building advertising ID resets, app-level permissions, and approximate-location toggles. Vehicles have none of these. A Vehicle Identification Number cannot be reset. It is stamped into the frame, encoded in every telematics packet, and cross-referenceable against NHTSA recall databases and public license-plate lookups. Where a phone exposes one person at a time, a family minivan exposes every occupant for its 12-year average lifespan.

The automotive security community has responded vigorously to control-plane threats. CAN bus injection [3], ECU firmware exploitation, and sensor spoofing have all received attention. The data plane is different. The adversary who buys broker data needs no physical proximity, no zero-day exploit, and no per-vehicle effort. One subscription covers every VIN in the feed. Location privacy research proves that mobility traces are re-identifiable [4], but operates on generic trajectories without vehicle-specific structural amplifiers. Defense

[1] Apple introduced App Tracking Transparency in iOS 14.5 (April 2021), requiring per-app opt-in for cross-app tracking. Google followed with Privacy Sandbox in Android 13. No vehicle OEM offers equivalent controls.

work such as PRICAR [20] addresses privacy-preserving data sharing but assumes cooperative OEMs, not adversarial brokers. The missing piece is a concrete, end-to-end analysis of what a data-purchase adversary can actually achieve against the vehicle ecosystem.

We present VIN2VICTIM, a composable attack pipeline that bridges this gap. The key insight is that three structural amplifiers, which we characterize in Section 2, create a qualitatively different threat surface for vehicles compared to phones. Our pipeline exploits these amplifiers across four stages: VIN-based identity resolution (VINRESOLVE), trajectory re-identification (TRAJLINK), home and workplace inference (LOCPIN), and departure prediction (SCHEDPREDICT). A fifth module, BROKERTEST, stress-tests the pipeline at broker-realistic temporal resolutions.

We scope our claims carefully. We demonstrate feasibility on small public datasets, not prevalence across the U.S. fleet.[2] We have not purchased broker data for ethical and legal reasons. Our validation relies on public GPS datasets and conservative analytical extrapolation. With those caveats, here is what breaks: two spatio-temporal points suffice to re-identify 99.1% of taxi trajectories (T-Drive), 96.0% of Porto taxis, and 81.8% of personal GPS traces (GeoLife). Home clusters are produced for 92 to 100% of users across all three datasets. The attack survives 60-minute broker sampling. And geo-indistinguishability [1] at ε=0.1, a strong formal defense, still preserves 74 to 78% unicity. No single defense we tested defeats the full pipeline, which creates a defender's dilemma.

We make four contributions:

- We identify three structural amplifiers (immutable VINs, coarse consent, physical anchoring) that make vehicle privacy qualitatively worse than phone privacy (Section 2).
- We design and implement a composable four-stage attack pipeline from broker data to physical-world outcomes (Sections 3 4).
- We evaluate the pipeline on three public GPS datasets spanning 2008 to 2014 and demonstrate that broker-realistic temporal coarsening does not defeat it (Section 4).
- We propose three structural mitigations, each targeting one amplifier, and articulate the open problems each raises (Section 5).

The code, datasets, and replication scripts are available at https://github.com/hanabhp/VIN2VICTIM.

We do not claim algorithmic novelty in the individual pipeline stages; trajectory unicity [4], DBSCAN clustering [9], and home inference from GPS [13] are established techniques. Our contribution is the composition: chaining these techniques through vehicle-specific structural amplifiers and testing them at broker-realistic resolutions produces a qualitatively different threat than what the generic mobility-privacy literature describes.

[2]The U.S. vehicle fleet comprises approximately 285 million registered vehicles as of 2024. Our evaluation covers 79 users across three datasets; the quantitative results are proof-of-concept, not fleet-scale validation.

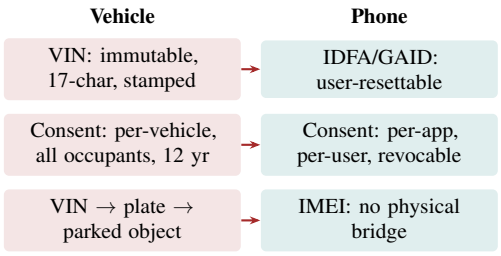


Figure 1: Three structural amplifiers separate vehicle data brokers from phone data brokers.

2 Why Vehicles Are Not Phones

This section characterizes the structural properties that distinguish the vehicle data broker threat from the well-studied phone tracking problem. We identify three amplifiers; Figure 1 places each alongside the corresponding phone-world mitigation.

A1: Immutable identifiers. A smartphone advertising ID (IDFA on iOS, GAID on Android) can be reset by the user at any time. A VIN cannot be reset. It is physically stamped into the chassis, encoded in title records, insurance policies, and every telematics packet transmitted to the OEM. We decompose the 17-character VIN per 49 CFR Part 565 and find approximately 44 effective bits of identity entropy.[3] Even with the last six digits (serial number) redacted, ≈24 prefix bits remain. Against the U.S. fleet of ≈285M registered vehicles, this yields a mean anonymity set of only $285\text{M}/2^{24} \approx 17$ vehicles. The VIN is a lifelong correlator: it links broker data purchased today to historical records from a decade ago.

A2: Coarse consent. Phone apps request location permission individually, per user, with granularity options (precise versus approximate). Vehicle telematics consent is granted once, at purchase or lease, for the vehicle rather than the driver. Every occupant is exposed. The consent persists for the vehicle's average 12-year lifespan and typically survives resale. A family minivan with four regular occupants exposes four people under one consent decision made by a single buyer.

A3: Digital-to-physical bridge. A phone's IMEI resolves to a subscriber but not directly to a visible, stealable physical object parked at a known address. A VIN resolves through public databases (NHTSA recalls, state DMV records, Carfax reports) to an owner name, and through broker GPS data to a home address. The adversary can then walk to the vehicle. This bridge from digital identifier to physical target is

[3]VIN entropy derivation. World Manufacturer Identifier (positions 1 to 3, ≈600 active WMIs, ≈9 bits); Vehicle Descriptor Section (positions 4 to 8, constrained by WMI to ≈130 body-engine-transmission combinations per manufacturer, ≈7 bits); check digit (position 9, deterministic, 0 bits); model year (position 10, ≈30 years on road, ≈5 bits); assembly plant (position 11, ≈8 per manufacturer, ≈3 bits); sequential serial number (positions 12 to 17, 10^6 values, ≈20 bits). Total: $9+7+0+5+3+20 = 44$ bits. This is an upper bound; empirical entropy is lower because manufacturers do not use all combinations, making the attack *easier*.

Table 1: Adversary cost model per amplifier.

Amplifier	Vehicle	Phone
Identifier	VIN: immutable, 17-char	IDFA/GAID: resettable
Lifespan	12 years avg.	App-level, revocable
Consent	Per-vehicle, all occupants	Per-app, per-user
Physical link	VIN → plate → object	IMEI → subscriber only
Adversary cost	Single subscription, all VINs	Per-device targeting

what transforms data privacy violations into physical safety threats.[4] Table 1 summarizes the three amplifiers and their practical consequences for an adversary. The cost asymmetry is the key observation: a single broker subscription gives the adversary simultaneous access to all four pipeline stages for every VIN in the feed, with no per-vehicle effort required. Upstream Security [29] documents that the vehicle data broker market involves dozens of intermediaries between OEMs and downstream consumers.

3 Attack Pipeline

This section describes the VIN2VICTIM pipeline architecture and threat model. The pipeline is composable: each stage's output feeds the next, and the adversary needs only commercially available data. Figure 2 shows the four stages and the cross-resolution resilience module, alongside a worked example threading a synthetic VIN through every stage.

Threat model. The adversary is a non-state actor who purchases vehicle GPS telemetry from a commercial data broker. The feed contains pseudonymized VINs, timestamps, and coordinates at intervals of 1 to 60 minutes. The adversary's goal is physical surveillance of a target individual. They know the target's full VIN (obtainable from public records, dealer lots, or windshield inspection). We assume the broker feed indexes each trajectory by a stable pseudonymous vehicle identifier derived from the VIN; this is consistent with the FTC's findings in the GM/OnStar order, which documented that OnStar transmitted VIN-linked data to LexisNexis and Verisk [11]. Before its 2023 bankruptcy, Wejo publicly advertised vehicle-level GPS traces segmented by trip; Otonomo similarly offered coordinate-level feeds to enterprise subscribers [29]. The precise field schemas and pricing tiers vary by broker and are not publicly standardized.

3.1 Per-stage specification

Each stage takes a typed input, applies a deterministic operation with named parameters, and produces a typed output. We list these explicitly so readers can map the empirical results in Section 4 back to a single, unambiguous procedure.

[4]The FTC's finalized order against GM explicitly acknowledges this physical dimension, noting that geolocation data "reveals sensitive information about consumers' physical movements" and "can be used to locate and stalk individuals" [10, 11].

S1: VINRESOLVE. *Input:* a 17-character VIN. *Operation:* decompose per 49 CFR Part 565 into world manufacturer identifier (positions 1 to 3), vehicle descriptor section (4 to 8), check digit (9), model year (10), assembly plant (11), and sequential serial (12 to 17); compute identity entropy from the active WMI registry and per-manufacturer attribute combinatorics. *Output:* a ($\approx$44)-bit identity anchor for the full VIN, falling to $\approx$24 bits when the serial is redacted. This anchor seeds the remaining pipeline.

S2: TRAJLINK. *Input:* broker GPS feed indexed by pseudonymous vehicle identifier. *Operation:* apply the de Montjoye, Hidalgo, Verleysen, and Blondel unicity framework [4]; given p randomly sampled spatio-temporal points from the target trajectory, count how many vehicles in the feed share all p points within resolution cells of Δs km (spatial) and Δt minutes (temporal). *Parameters:* Δs=0.5 km and Δt=60 min by default; 200 Monte Carlo trials. *Output:* a unicity score and, when the score is 1.0, a re-identified trajectory.

S3: LOCPIN. *Input:* re-identified trajectory. *Operation:* detect stay-points (locations where the vehicle remains within a 200 m radius for $\geq$10 minutes); cluster them with DBSCAN [9]. *Parameters:* ε=500 m, `min_samples`=3. *Output:* the largest nighttime cluster (20:00 to 06:00) is labeled "home"; the largest daytime cluster (09:00 to 17:00) is "work." On taxi data the "home" cluster corresponds to the depot or driver's parked location rather than a residence; we discuss this in Section 5.

S4: SCHEDPREDICT. *Input:* home cluster and timestamps of departures from it. *Operation:* bin departure times into 30-minute windows; select the modal bin. *Parameters:* 30-minute window; $\pm$60-minute prediction tolerance. *Output:* a predicted departure window, plus a predictability score equal to the fraction of actual departures falling within tolerance of the predicted time.

What we do not claim. The pipeline breaks when the target vehicle lacks a connected telematics unit (pre-2018 models without cellular modems). It weakens when broker data is temporally coarser than 60-minute intervals. We have not purchased actual broker data; our validation relies on public GPS datasets. The S1$\rightarrow$S2 link (resolving a VIN to a specific trajectory in a broker feed) is analytically modeled rather than empirically tested, as it requires broker access.

4 Evaluation

This section evaluates VIN2VICTIM on three public GPS datasets and tests its resilience to temporal coarsening and formal privacy defenses. Our goal is to establish feasibility, not prevalence.

Setup, metrics, and units. Three datasets are used throughout: T-Drive (Beijing taxis, N=18 after completeness filtering, 24,104 GPS points over February 2 to 8, 2008) [33]; GeoLife (Beijing personal traces, N=11, 2,037,581 points

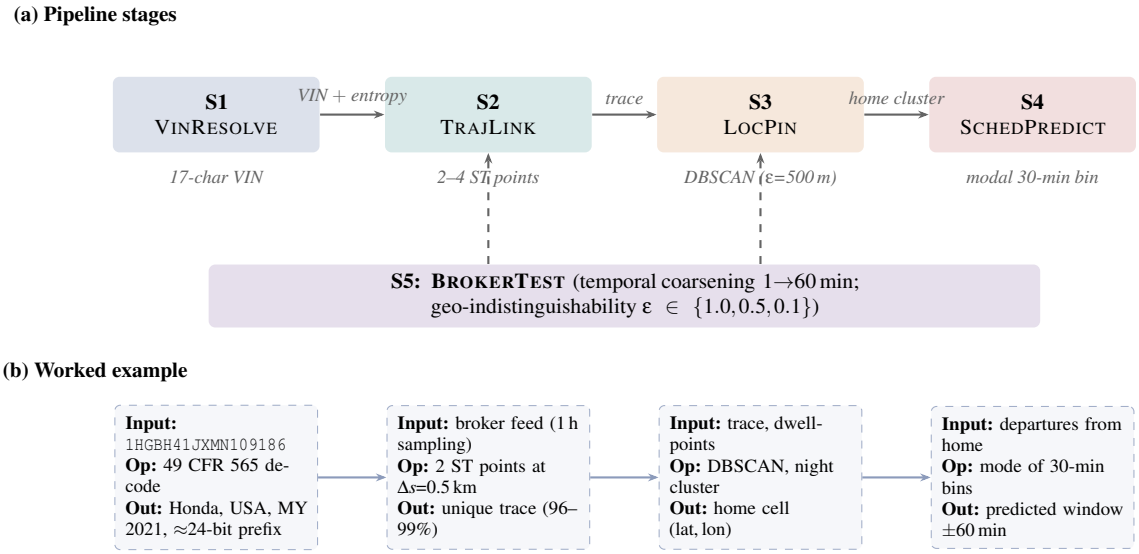


Figure 2: Pipeline architecture (a) and worked example threading a synthetic Honda VIN (b).

spanning 2007 to 2012) [35]; and Porto (taxis, N=50 uniformly sampled from 442, 895,325 points across 14 contiguous days from the 2013 to 2014 corpus) [15]. Three metrics are reported. *Unicity* (dimensionless, fraction in $[0,1]$, reported as percentage) is the share of users uniquely identified by p randomly sampled spatio-temporal points at resolution Δs km $\times$ Δt min, averaged over 200 Monte Carlo trials. *Home/work cluster yield* (percentage) is the fraction of users for which LOCPIN produces a non-empty cluster of the corresponding type; for GeoLife, we additionally report *residential-polygon overlap*, the fraction of inferred home cells that fall inside an OpenStreetMap residential land-use polygon. *Predictability* (percentage) is the fraction of actual departures falling within $\pm$60 min of the modal 30-minute bin. All confidence intervals are 95% Wilson binomial intervals. Default parameters are Δs=0.5 km, Δt=60 min, DBSCAN ε=500 m, `min_samples`=3.

Trajectory re-identification. Figure 3 shows unicity as a function of the number of sampled spatio-temporal points p. On T-Drive, two points suffice to uniquely identify 99.1% of taxis (95% CI: $[98.4, 99.8]$). On Porto, two points yield 96.0% ($[94.1, 97.9]$) despite a 2.8$\times$ larger fleet. On GeoLife, two points yield 81.8% ($[78.2, 85.4]$), rising to 100% at p=4.

Scaling extrapolation. At N=18 or N=50, high unicity is expected even under weak spatial structure. To project scaling, we apply the birthday-bound framework: the probability that a given user is unique across all p sampled points in a grid of G spatio-temporal cells is approximately $\left(1-G^{-p}\right)^{N-1}$. With Δs=0.5 km and Δt=60 min over Beijing's road network ($G \approx 4{,}800$ cell-hours per week), the model predicts unicity of 94.2% at p=4 for N=10,000 and 82.1% at N=100,000. These estimates assume spatial independence, which real road networks violate (vehicles are funneled through corridors),

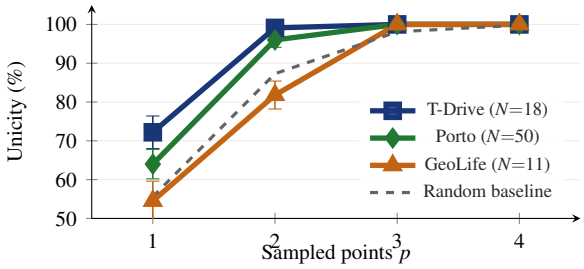


Figure 3: Unicity versus sampled spatio-temporal points p across three datasets and a structureless random baseline.

so they are conservative lower bounds. The key takeaway is that unicity degrades logarithmically with population size, not linearly.

A critical caveat: at small N, near-perfect unicity at p=2 approaches a mathematical tautology. The random baseline in Figure 3 (dashed line) addresses this concern. We generate synthetic trajectories by uniformly sampling from the same spatial grid and temporal range, removing all driving structure while preserving population size and resolution. Even this structureless baseline achieves 87.3% unicity at p=2 for N=18, confirming that population size inflates the result. The gap between actual T-Drive data and the random baseline (99.1% versus 87.3%) isolates the structural contribution of real driving patterns: road-network funneling and commute regularity push unicity 12 percentage points above what small N alone would produce. Porto is informative here because its larger N=50 lowers the baseline, yet real driving patterns still push unicity to 96.0%. What matters is not the absolute unicity but a comparative observation: vehicles reach the same unicity threshold with fewer points than phones.

De Montjoye, Hidalgo, Verleysen, and Blondel report that four points re-identify 95% of 1.5 million phone users [4]. Zang and Bolot [34] confirm at scale (25 million cell-phone users) that anonymization of location data fails: the top-2 locations uniquely identify the majority of users at cell-sector granularity. We observe equivalent unicity at p=2 on our vehicle data, even after accounting for the population-size effect. This comparison is qualitative and depends on resolution and road-network constraint structure, not a direct statistical equivalence across population sizes. Road-network constraint and commute regularity explain this: vehicles are funneled through a finite set of corridors, and daily home-work-home patterns are more rigid than phone-carrying pedestrian mobility. Gambs, Killijian, and Núñez del Prado Cortez [12] demonstrate that mobility Markov chain models can de-anonymize geolocated datasets even after downsampling, further confirming that sparse vehicle traces carry rich identifying structure.

Home and workplace inference. LocPin produces a candidate home cluster for 17 of 18 T-Drive taxis (94.4%; Wilson 95% CI: [74.2, 99.0]), 46 of 50 Porto taxis (92.0%; [81.2, 96.8]), and all 11 of 11 GeoLife users (100%; [74.1, 100]). Ground-truth verification is dataset-dependent. For GeoLife, which contains personal multi-modal traces, 10 of 11 inferred home cells (90.9%) fall inside an OpenStreetMap residential land-use polygon, our proxy for residential ground truth. For T-Drive and Porto, no per-user residential ground truth exists: these are taxi datasets, and the cluster output corresponds to a depot or parked-overnight location rather than a private residence; we report cluster yield only and discuss this in Section 5. Work inference reaches 55.6% on T-Drive ([33.7, 75.4]), 8.0% on Porto ([3.0, 18.8]; expected, since taxis have no fixed workplace), and 100% on GeoLife ([74.1, 100]). These results are consistent with Krumm's foundational finding [13] that home-like locations can be inferred from GPS tracks with simple clustering; what our results add is that the technique works on broker-grade data with 60-minute gaps and degrades gracefully rather than failing.

Broker-resolution resilience. Figure 4 shows the key finding: temporal coarsening does not defeat the pipeline. At 60-minute intervals (91% data reduction), T-Drive unicity remains 97.2% at p=2. Porto unicity holds at 93.2%. Home cluster yield drops only 9 to 10 percentage points across both datasets. The simplest mitigation a broker could deploy, namely reducing sampling frequency, is ineffective because two points per hour still capture the spatial distinctiveness of driving patterns.

Defense evaluation. We test the pipeline against geo-indistinguishability [1], a formal ε-differential privacy mechanism for location data. Shokri, Theodorakopoulos, Le Boudec, and Hubaux [22] provide the Bayesian game-theoretic framework for quantifying location privacy; our evaluation instantiates their attacker model with vehicle-specific priors. We add planar Laplace noise calibrated to $\varepsilon \in \{1.0, 0.5, 0.1\}$ (higher ε means less noise). At ε=1.0, unicity drops to 94.3% (T-Drive) and 91.6% (Porto). At ε=0.1 (strong privacy), T-Drive unicity still remains at 78.4%, Porto at 74.2%, and home cluster yield degrades to 62 to 67%. Spatial cloaking at 500 m reduces home cluster precision but renders location-based services unusable for navigation. This creates a defender's dilemma: no defense we tested simultaneously preserves utility and defeats the full pipeline.

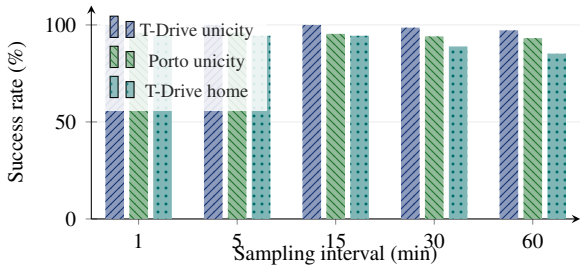


Figure 4: Pipeline performance under broker-realistic temporal coarsening.

Table 2: Headline results, with broker coarsening and geo-indistinguishability ablations.

Stage	Metric	T-Drive	Porto	GeoLife
TrajLink	Unicity p=2	99.1%	96.0%	81.8%
TrajLink	Unicity p=4	100%	100%	100%
LocPin	Home cluster yield	94.4%	92.0%	100%
LocPin	Work cluster yield	55.6%	8.0%	100%
SchedPredict	Predictability	66.9%	19.3%	32.0%
Under 60-min broker sampling:				
TrajLink	Unicity p=2	97.2%	93.2%	78.4%
LocPin	Home cluster yield	85.2%	84.0%	90.9%
Under geo-indist. ε=0.1*:*				
TrajLink	Unicity p=2	78.4%	74.2%	63.6%
LocPin	Home cluster yield	66.7%	62.0%	72.7%

Summary of findings. Table 2 consolidates our evaluation results across all three datasets and pipeline stages. Three observations stand out. First, TrajLink achieves near-ceiling unicity on T-Drive and Porto, with Porto's larger N=50 producing only a modest 3-point drop relative to T-Drive's N=18, suggesting the attack scales. Second, LocPin is the most robust stage: home cluster yield stays above 84% even at 60-minute broker sampling across all datasets. Third, SchedPredict is the weakest stage: predictability reaches 66.9% on T-Drive ([43.5, 84.0]) but only 19.3% on Porto ([10.3, 33.0]) and 32.0% on GeoLife ([12.3, 60.3]). This is expected for taxi drivers, whose schedules are irregular by nature. SchedPredict is best understood as an optional escalation step that applies mainly to commuter-like private vehicles; even without it, TrajLink and LocPin already enable the core stalking threat (identifying who parks where, every night). The composed pipeline degrades gracefully rather than failing catastrophically, which means partial defenses reduce but do not eliminate the threat.

Sensitivity to spatial resolution and clustering parameters. The default Δs=0.5 km is informed by published broker feed resolutions and by Beijing's typical road-block scale. Halving Δs to 0.25 km tightens cells and pushes T-Drive unicity at p=2 from 99.1% to nearly 100%, while doubling it to 1 km lowers unicity to roughly 92% by merging adjacent road segments into the same cell; this confirms the result is not balanced on a knife-edge of resolution. The DBSCAN parameters ε=500 m and `min_samples`=3 control how stay-points coalesce into clusters. Tightening ε below 200 m fragments single residences into multiple clusters and lowers home-cluster yield; loosening it above 1 km merges adjacent venues (home, neighborhood store, gas station) and reduces the precision of the inferred location. Increasing `min_samples` above 5 suppresses small clusters and disproportionately hurts GeoLife (where some users have fewer total stay-points), while decreasing it below 2 admits noise. Across the parameter neighborhood we explored, the qualitative picture is stable: TRAJLINK remains in the 90s for taxi datasets and the 80s for GeoLife, and LOCPIN home cluster yield stays above 80% wherever clusters are defined sensibly. We treat a quantitative sensitivity sweep as future work; the qualitative robustness is what the threat model needs.

5 Discussion and Limitations

This section proposes three mitigations, each targeting one structural amplifier, and states the limitations of our evaluation.

M1: Ephemeral pseudonyms (targets A1). Broker interfaces should replace VINs with rotating pseudonyms that expire after a configurable window. Rotation periods shorter than one week would break cross-session trajectory linking. The open problem: pseudonym rotation must be coordinated with legitimate uses (insurance, recalls) that require persistent vehicle identity.

M2: Dwell-aware suppression (targets A3). Suppress GPS pings when the vehicle has been stationary for more than τ minutes at the same location. This directly defeats LOCPIN by removing the dwell events that anchor home and work inference. The tradeoff: fleet operators and usage-based insurers rely on stationary data for billing and parking analytics. PRICAR [20] would defeat S3 by retaining raw GPS locally, but it does not address the threat when the OEM itself is the data source transmitting to brokers, as documented in the FTC/GM order [11].

M3: Per-occupant consent (targets A2). Replace the current vehicle-level consent with individual opt-in for each driver or passenger. This mirrors the phone model where each user independently controls their location sharing. Implementation requires in-vehicle identity verification, which raises usability and cost barriers.

Defenses we did not evaluate. Our defense evaluation tested geo-indistinguishability and temporal coarsening because both are deployable broker-side without breaking interoperability. Two further defense classes are widely cited in the location-privacy literature and warrant explicit acknowledgement. *k-anonymity* cloaks each released coordinate inside a region containing at least $k-1$ other vehicles; it weakens TRAJLINK at high k but increases query latency and degrades utility for navigation. *Dummy trajectory injection* mixes synthetic traces into the feed to lower the marginal probability that a sampled point belongs to the target; it imposes storage and bandwidth costs on the broker and can be filtered by an adversary who can detect statistical regularities of the dummy generator. Neither defense is broker-deployable without changes to consumer applications, and a quantitative evaluation requires modeling broker query workloads we do not have. We flag both as priorities for the full-paper extension of this work.

Limitations. Our evaluation has four limitations we do not attempt to conceal. First, the dataset scale (N=79 total, with the largest single dataset at N=50) is modest; our quantitative results are proof-of-concept, not fleet-scale validation. We present analytical scaling arguments based on the birthday-bound framework, but these assume spatial independence that real road networks violate. Second, all three datasets are from non-U.S. cities (Beijing, Porto); U.S. driving patterns would differ. Third, we have not used real broker data; the S1→S2 link is analytically modeled and requires that the broker feed indexes trajectories by a stable pseudonymous vehicle identifier. Fourth, two of three datasets (T-Drive, Porto) consist of taxi traces, which lack residential ground truth: the cluster LOCPIN produces is a depot or parked-overnight location, not a private residence. The GeoLife result is the closest proxy for private-vehicle behavior; on those 11 multimodal traces, 10 of 11 inferred home cells fall inside an OpenStreetMap residential polygon. Stronger ground truth, ideally a U.S. household-level vehicle GPS dataset with reported residential addresses, is not publicly available; obtaining it without compromising occupant privacy is itself an open problem.

State-level privacy laws (CCPA, VCDPA) and the proposed ADPPA would constrain but not eliminate broker data sales; van Kempen, Bagayatkar, Frolikov, Georgiou, and Tsudik [30] measure CCPA compliance across all 543 California-registered data brokers and report that more than 40% fail to respond to consumer requests at all, suggesting that even the strongest U.S. regime does not bind in practice. Our mitigations address the technical layer that persists regardless of regulatory regime.

6 Related Work

This section positions VIN2VICTIM relative to three research threads.

Automotive control-plane security. Checkoway, McCoy, Kantor, Anderson, Shacham, Savage, Koscher, Czeskis, Roes-

ner, and Kohno [3] published the first comprehensive automotive attack surface analysis, demonstrating remote code execution via telematics units. Subsequent work has targeted CAN bus injection, LiDAR spoofing [2], and GPS spoofing. Enev, Takakuwa, Koscher, and Kohno [8] showed that in-vehicle CAN bus sensor data can fingerprint individual drivers, demonstrating that vehicles leak identity even without GPS. Troncoso, Danezis, Kosta, Balasch, and Preneel [28] designed PriPAYD, a privacy-preserving pay-as-you-drive insurance architecture that performs premium calculation locally, directly relevant to our M2 mitigation. Recent work studies the connectivity layer that broker pipelines depend on: Tavasoli, Sarrafzadeh, Khaleghi, Zakaria, Pasandi, and Karimoddini [26] survey data communication challenges for connected and automated vehicles in rural areas, where intermittent V2X coverage and sparse 5G deployment shift where telemetry actually originates. Tavasoli, Sarrafzadeh, Karimoddini, Khaleghi, and Pasandi [25] build an edge-first hybrid clustering and transformer intrusion-detection system for the same rural-CAV setting; on real CAN datasets it reaches 100% recall and 98.9% accuracy, and shifts the detection burden to the vehicle rather than the OEM cloud. Both works defend the channel between vehicle and OEM. VIN2VICTIM targets a layer further downstream that neither addresses: the broker market that purchases the data once it has reached the OEM. The on-device machine-learning trend that motivates these IDS designs is itself surveyed by Pasandi, Pasandi, Parastar, Moradbeikie, and Nadeem [18], whose comprehensive treatment of next-generation on-device ML maps the deployment substrate that future broker-side or vehicle-side mitigations would have to live on. Existing control-plane attacks require per-vehicle effort or physical proximity. VIN2VICTIM requires neither.

Location privacy. Krumm [13] was the first to quantify home-address inference from GPS tracks, testing four algorithms on 172 volunteers and achieving 5% identity recovery via reverse geocoding. De Montjoye, Hidalgo, Verleysen, and Blondel [4] establish unicity bounds on phone traces; the same group later proved that four credit-card transactions uniquely identify 90% of shoppers [5]. Song, Qu, Blumm, and Barabási [24] derive information-theoretic limits on mobility predictability. Gambs, Killijian, and Núñez del Prado Cortez [12] formalize mobility Markov chain models for de-anonymization attacks and show resilience to downsampling. Dhondt, Le Pochat, Dimova, Joosen, and Volckaert [7] compose API scraping, trilateration, and profile correlation into an identity-inference pipeline targeting location-based dating apps at USENIX Security '24; their methodology of chaining known techniques to expose a composed threat surface is the closest procedural precedent to our work, though in a different domain. The 2018 Strava Global Heatmap incident [23], in which aggregated fitness-tracker GPS data revealed the locations and patrol routes of military bases in conflict zones, demonstrated that even anonymized and aggregated location data can expose sensitive facilities at scale. Wang, Gao, Li, Wang, Jin, and Sun [31] evaluate nine de-anonymization algorithms on 2.16M cellular trajectories and show that spatio-temporal mismatches between data sources are the primary barrier. Yadav, Yu, Xie, Huang, and Qiu [32] introduce VehiTrack, a context-aware attack that defeats geo-indistinguishability for vehicles by exploiting road-network and traffic constraints, reducing average inference error by 61% over the context-free baseline; this is the closest direct precedent to our BROKERTEST result that ε=0.1 geo-indist still leaves 74 to 78% unicity. These operate on generic location data without vehicle-specific structural amplifiers.

Vehicle privacy measurement. Pesé [19] examines third-party app permissions on Android Automotive OS and finds that 78% of 55 apps fail to disclose dangerous permissions in their privacy policies. Mozilla's "Privacy Not Included" project [16] rates cars as the worst product category for privacy. Mallojula, Li, Du, and Luo [14] find that 70% of 125 OBD-II companion apps have vulnerabilities enabling private information leakage. Satt, Hu, Zielinski, and Huang [21] approach the same threat surface from the victim side, using a low-cost spectrum analyzer to detect hidden cellular GPS trackers placed on vehicles by stalkers; their attack surface and ours are complementary, since both exploit that vehicles are tied to a stable physical object that an adversary can reach. These works catalog data flows but do not model adversarial exploitation from broker data into a multi-stage attack pipeline.

No prior work composes a broker-to-physical attack chain using vehicle-specific structural amplifiers with an implemented pipeline on real GPS data.

7 Conclusion

We presented VIN2VICTIM, a composable attack pipeline demonstrating that the structural conditions for broker-mediated vehicle surveillance exist today. Three amplifiers, namely immutable VINs, coarse consent, and physical anchoring, make vehicles qualitatively worse than phones for location privacy. Our evaluation across three datasets and two continents shows that temporal coarsening and formal privacy defenses do not defeat the composed pipeline. We urge the community to prioritize the data-plane threat alongside control-plane security. We flag one near-term risk: the pipeline stages we describe are automatable by current large language models acting as OSINT agents. This lowers the skill barrier from "data analyst" to "anyone with API access" and warrants attention from the vehicular security community before such tools become widely deployed. As enforcement actions move from individual cases (FTC v. GM, Texas v. Allstate) toward broader rule-making, the technical primitives we document define what mitigation must actually defeat.

Ethical Considerations

This paper is a measurement and synthesis contribution, not a vulnerability disclosure. The data flow we analyze is already public in active FTC actions against GM and OnStar [10, 11] and the Texas AG complaint against Allstate and Arity [17], and the underlying re-identification techniques are over a decade old [4, 24, 34]. There is no specific vendor whose product is uniquely vulnerable, no zero-day, and no defender who would gain meaningful lead time from a private notification, so coordinated disclosure as ordinarily understood does not apply. We took standard scoping precautions regardless: only public GPS datasets with no PII, no real broker feed purchased or accessed, synthetic VINs in worked examples (Figure 2), and a code release at https://github.com/hanabhp/VIN2VICTIM that contains no broker data or attack-enabling artifacts. The precedent for publishing attack details to motivate defenses is well established in automotive security [3].

Ethics attestation. We attest that the research team considered the ethics and the AI Use Policy of this research, that the authors believe the research was done ethically.

Use of Generative AI

Generative AI tools (Claude, Anthropic) were used for prose editing across the paper and for initial Python scaffolding of the DBSCAN clustering and unicity modules. All experimental design, analytical arguments, result interpretation, and conclusions were produced by the authors.

Policy attestation. We attest that the research team has read the ethics considerations in the VehicleSec Symposium Call for Papers, and adheres to the three key criteria described in the Use of Generative AI Policy.

References

[1] Miguel E. Andrés, Nicolás E. Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Geo-indistinguishability: Differential privacy for location-based systems. In *Proc. 20th ACM Conf. Computer and Communications Security (CCS)*, pages 901–914. ACM, 2013. DOI: https://doi.org/10.1145/2508859.2516735.

[2] Yulong Cao, Chaowei Xiao, Benjamin Cyr, Yimeng Zhou, Won Park, Sara Rampazzi, Qi Alfred Chen, Kevin Fu, and Z. Morley Mao. Adversarial sensor attack on LiDAR-based perception in autonomous driving. In *Proc. ACM Conf. Computer and Communications Security (CCS)*, pages 2267–2281. ACM, 2019. DOI: https://doi.org/10.1145/3319535.3339815.

[3] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *Proc. 20th USENIX Security Symposium*, pages 77–92. USENIX Association, 2011.

[4] Yves-Alexandre de Montjoye, César A. Hidalgo, Michel Verleysen, and Vincent D. Blondel. Unique in the crowd: The privacy bounds of human mobility. *Scientific Reports*, 3:1376, 2013. DOI: https://doi.org/10.1038/srep01376.

[5] Yves-Alexandre de Montjoye, Laura Radaelli, Vivek Kumar Singh, and Alex "Sandy" Pentland. Unique in the shopping mall: On the reidentifiability of credit card metadata. *Science*, 347(6221):536–539, 2015. DOI: https://doi.org/10.1126/science.1256297.

[6] Der Spiegel. Volkswagen subsidiary leaked location data of 800,000 electric vehicles. Der Spiegel / Chaos Computer Club, December 2024. Reported Dec. 27, 2024. Cariad misconfiguration exposed GPS traces on Amazon cloud storage. See also https://www.bleepingcomputer.com/news/security/customer-data-from-800-000-electric-cars-and-owners-exposed-online/.

[7] Karel Dhondt, Victor Le Pochat, Yana Dimova, Wouter Joosen, and Stijn Volckaert. Swipe left for identity theft: An analysis of user data privacy risks on location-based dating apps. In *Proc. 33rd USENIX Security Symposium*, pages 5053–5070. USENIX Association, 2024.

[8] Miro Enev, Alex Takakuwa, Karl Koscher, and Tadayoshi Kohno. Automobile driver fingerprinting. *Proceedings on Privacy Enhancing Technologies*, 2016(1):34–51, 2016. DOI: https://doi.org/10.1515/popets-2015-0029.

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining (KDD)*, pages 226–231. AAAI Press, 1996.

[10] Federal Trade Commission. FTC takes action against General Motors for sharing drivers' precise location and driving behavior data without consent. Press Release, Jan. 2025, 2025. https://www.ftc.gov/news-events/news/press-releases/2025/01/ftc-takes-action-against-general-motors-sharing-drivers-precise-location-driving-behavior-data.

[11] Federal Trade Commission. FTC finalizes order with General Motors and OnStar addressing allegations of sharing drivers' data without consent. Press Release, January 2026. https://www.ftc.gov/news-events/news/press-releases/2026/01/ftc-finalizes-order-settling-allegations-gm-onstar-collected-sold-geolocation-data-without-consumers.

[12] Sébastien Gambs, Marc-Olivier Killijian, and Miguel Núñez del Prado Cortez. De-anonymization attack on geolocated data. *Journal of Computer and System Sciences*, 80(8):1597–1614, 2014. DOI: https://doi.org/10.1016/j.jcss.2014.04.024.

[13] John Krumm. Inference attacks on location tracks. In *Proc. 5th Int. Conf. Pervasive Computing (Pervasive)*, volume 4480 of *LNCS*, pages 127–143. Springer, 2007. DOI: https://doi.org/10.1007/978-3-540-72037-9_8.

[14] Prashanthi Mallojula, Fengjun Li, Xiaojiang Du, and Bo Luo. Companion apps or backdoors? On the security of automotive companion apps. In *Proc. 29th European Symposium on Research in Computer Security (ESORICS), Part III*, volume 14984 of *LNCS*, pages 24–44. Springer, 2024. DOI: https://doi.org/10.1007/978-3-031-70896-1_2.

[15] Luís Moreira-Matias, João Gama, Michel Ferreira, João Mendes-Moreira, and Luis Damas. Predicting taxi-passenger demand using streaming data. *IEEE Trans. Intelligent Transportation Systems*, 14(3):1393–1402, 2013. DOI: https://doi.org/10.1109/TITS.2013.2262505.

[16] Mozilla Foundation. Privacy not included: Cars. https://foundation.mozilla.org/privacynotincluded/categories/cars/, 2023.

[17] Office of the Texas Attorney General. Attorney General Paxton sues Allstate and Arity for illegally collecting, using, and selling over 45 million Americans' driving data. Press Release, January 2025. https://www.texasattorneygeneral.gov/news/releases/attorney-general-ken-paxton-sues-allstate-and-arity-unlawfully-collecting-using-and-selling-over-45.

[18] Hannaneh B. Pasandi, Faezeh B. Pasandi, Farideh Parastar, Azin Moradbeikie, and Tamer Nadeem. Echoing the future: On-device machine learning in next-generation networks: A comprehensive survey. Manuscript, available on ResearchGate, 2025.

[19] Mert D. Pesé. A first look at Android Automotive privacy. *SAE Technical Paper 2023-01-0037*, 2023. DOI: https://doi.org/10.4271/2023-01-0037.

[20] Mert D. Pesé, Jay W. Schauer, Murali Mohan, Cassandra Joseph, Kang G. Shin, and John Moore. PRICAR: Privacy framework for vehicular data sharing with third parties. In *Proc. IEEE Secure Development Conf. (SecDev)*, pages 184–195. IEEE, 2023. DOI: https://doi.org/10.1109/SecDev56634.2023.00032.

[21] Moshe Chaim Satt, Donghan Hu, Patrick Zielinski, and Danny Yuxing Huang. You can drive but you cannot hide: Detection of hidden cellular GPS vehicle trackers. In *Proc. 3rd USENIX Symposium on Vehicle Security and Privacy (VehicleSec)*, Seattle, WA, 2025. USENIX Association. https://www.usenix.org/conference/vehiclesec25/presentation/satt.

[22] Reza Shokri, George Theodorakopoulos, Jean-Yves Le Boudec, and Jean-Pierre Hubaux. Quantifying location privacy. In *Proc. 32nd IEEE Symp. Security and Privacy (S&P)*, pages 247–262. IEEE, 2011. DOI: https://doi.org/10.1109/SP.2011.18.

[23] Liz Sly. U.S. soldiers are revealing sensitive and dangerous information by jogging. The Washington Post, January 2018. Jan. 28, 2018. Aggregated fitness-tracker GPS data from Strava's Global Heatmap inadvertently revealed locations of military installations in conflict zones.

[24] Chaoming Song, Zehui Qu, Nicholas Blumm, and Albert-László Barabási. Limits of predictability in human mobility. *Science*, 327(5968):1018–1021, 2010. DOI: https://doi.org/10.1126/science.1177170.

[25] Mahsa Tavasoli, Abdolhossein Sarrafzadeh, Ali Karimoddini, Milad Khaleghi, and Hannaneh B. Pasandi. Hybrid clustering-transformer IDS for rural CAVs. In *Proc. CoNEXT'25 Student Workshop (CoNEXT-SW)*, pages 7–8, Hong Kong, 2025. ACM. DOI: https://doi.org/10.1145/3769700.3771699.

[26] Mahsa Tavasoli, Abdolhossein Sarrafzadeh, Milad Khaleghi, Mohammad Zakaria, Hannaneh Barahouei Pasandi, and Ali Karimoddini. Data communication challenges of connected and automated vehicles in rural areas. *IEEE Access*, 13:30245–30268, 2025. DOI: https://doi.org/10.1109/ACCESS.2025.3539598.

[27] Stuart A. Thompson and Charlie Warzel. Twelve million phones, one dataset, zero privacy. The New York Times, December 2019. Dec. 19, 2019. Investigation into a commercial phone-location dataset covering 12 million U.S. devices over several months.

[28] Carmela Troncoso, George Danezis, Eleni Kosta, Josep Balasch, and Bart Preneel. PriPAYD: Privacy-friendly pay-as-you-drive insurance. *IEEE Trans. Dependable and Secure Computing*, 8(5):742–755, 2011. DOI: https://doi.org/10.1109/TDSC.2010.71.

[29] Upstream Security. 2025 global automotive cybersecurity report. Technical report, Upstream Security Ltd., 2025. https://upstream.auto/ty-2025-gacr-executive-summary/.

[30] Elina van Kempen, Isita Bagayatkar, Pavel Frolikov, Chloe Georgiou, and Gene Tsudik. Consumer beware! exploring data brokers' CCPA compliance. *Proc. Priv. Enhancing Technol. (PoPETs)*, 2025(4):166–182, 2025. DOI: https://doi.org/10.56553/popets-2025-0125.

[31] Huandong Wang, Chen Gao, Yong Li, Gang Wang, Depeng Jin, and Jingbo Sun. De-anonymization of mobility trajectories: Dissecting the gaps between theory and practice. In *Proc. 25th Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2018. DOI: https://doi.org/10.14722/ndss.2018.23211.

[32] Sourabh Yadav, Chenyang Yu, Xinpeng Xie, Yan Huang, and Chenxi Qiu. Protecting vehicle location privacy with contextually-driven synthetic location generation. In *Proc. 32nd ACM Int. Conf. Advances in Geographic Information Systems (SIGSPATIAL)*, pages 29–41, Atlanta, GA, 2024. ACM. DOI: https://doi.org/10.1145/3678717.3691211.

[33] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. T-Drive: Driving directions based on taxi trajectories. In *Proc. 18th ACM SIGSPATIAL Int. Conf. Advances in Geographic Information Systems (GIS)*, pages 99–108. ACM, 2010. DOI: https://doi.org/10.1145/1869790.1869807.

[34] Hui Zang and Jean Bolot. Anonymization of location data does not work: A large-scale measurement study. In *Proc. 17th Annual Int. Conf. Mobile Computing and Networking (MobiCom)*, pages 145–156. ACM, 2011. DOI: https://doi.org/10.1145/2030613.2030630.

[35] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. Mining interesting locations and travel sequences from GPS trajectories. In *Proc. 18th Int. Conf. World Wide Web (WWW)*, pages 791–800. ACM, 2009. DOI: https://doi.org/10.1145/1526709.1526816.



Short: Is Your ECU Really Yours?

Abbas Ali and Hafiz Malik
University of Michigan-Dearborn
{mrabbas, hafiz}@umich.edu

Abstract

Vehicular networks rely on embedded devices (Electronic Control Units (ECUs), CAN bus nodes, and sensor modules) that communicate via digital ON/OFF switching signals. A compromised or counterfeit device that passes cryptographic checks poses a critical threat to vehicle safety. We present a hardware fingerprinting framework that authenticates embedded devices by analyzing the intrinsic analog characteristics of their digital switching behavior. The key insight is that every digital transition (a CAN bus bit transition, an ECU command pulse, a GPIO toggle) forces hardware through abrupt state changes where manufacturing variations in output drivers, clock sources, and passive components become observable as device-specific imperfections. We extract 211 features from captured waveforms and propose a hybrid classification strategy combining a cycle-based Random Forest (Approach A) and an OVA stacked ensemble (Approach B), achieving 99.5% real-time accuracy under controlled mounting conditions (93.6% without) and 100% device type accuracy across 8 devices spanning 4 hardware types, including same-model clone pairs. We demonstrate this framework on embedded development platforms as a proof-of-concept step toward automotive ECU authentication, laying the groundwork for physical-layer authentication in CAN bus and ECU security contexts, subject to validation on automotive hardware.

1 Introduction

Modern vehicles contain over 100 Electronic Control Units (ECUs) communicating via CAN bus, LIN, Ethernet, and other protocols, all of which are fundamentally built on digital ON/OFF switching signals [1]. The security of these networks depends critically on the authenticity of participating devices. An attacker who substitutes a legitimate ECU with a rogue device, or injects a counterfeit sensor into the vehicle network, can manipulate braking, steering, or powertrain functions with potentially catastrophic consequences [2, 15]. The automotive cybersecurity threat landscape has grown substantially [16], and regulatory frameworks such as ISO/SAE 21434 [17] and UNECE WP.29 R155 [18] now mandate systematic security risk management for vehicle systems.

Current vehicle security relies heavily on cryptographic authentication (e.g., SecOC in AUTOSAR) [3]. While cryptography verifies message integrity, it cannot detect a *hardware-level substitution attack*: an adversary who physically replaces a legitimate ECU with an identical-model clone programmed with modified firmware and transfers valid credentials to the replacement. Once credentials are compromised or cloned, software authentication provides no defense [19].

Hardware fingerprinting offers a complementary defense by exploiting manufacturing-induced physical variations that are present even in identical-model devices. Unlike cryptographic keys, these variations cannot be transferred or copied; they are intrinsic to the physical hardware [4].

1.1 Our Contribution

We propose fingerprinting embedded devices through their *digital switching behavior*: the same ON/OFF signaling that underpins CAN bus, ECU commands, SPI, I^2C, and UART. The fundamental observation is that when a device switches a digital output from LOW to HIGH (or vice versa), the *exact shape* of that transition is determined by its hardware, including parasitic capacitances, output driver transistor characteristics, PCB trace impedance, and clock source jitter. These sub-microsecond analog details are device-specific and reproducible across time.

Why vehicular systems are an ideal target:

- CAN bus nodes repeatedly transmit ON/OFF bit patterns at known rates, providing exactly the repetitive, structured stimulus our fingerprinting approach requires
- ECU commands are discrete ON/OFF switching events observable on digital I/O lines
- Vehicle networks are physically accessible (diagnostic OBD-II port, exposed wiring)

- The threat model is concrete: counterfeit ECUs and compromised sensor nodes are real adversarial scenarios in automotive supply chains [16, 20]

Our framework does not require firmware modification or network access; a probe on any digital line during normal vehicle operation is sufficient. We validate on 8 embedded devices across 4 hardware types, including same-model clone pairs, demonstrating that the approach works against the hardest adversarial scenario: identical-model device substitution. The devices used in this study are embedded development platforms rather than production automotive ECUs. This is a deliberate scoping decision: our goal is to establish the feasibility and methodology of physical-layer fingerprinting at the analog signal level before extending to the automotive deployment context, which introduces additional environmental and network complexity addressed in Section 6.3. Code, data, and pre-trained models are publicly available on Zenodo as part of the artifact submission accompanying this paper.

2 Related Work

2.1 Vehicle Network Security

CAN bus security has been studied extensively since Koscher et al. [1] demonstrated remote vehicle compromise and Hoppe et al. [15] analyzed systematic CAN attack vectors. Intrusion detection systems (IDS) for CAN bus focus on message timing anomalies [13], entropy-based detection [21], and payload-level analysis [14]. VoltageIDS [14] notably exploits voltage-level characteristics on the CAN bus for anomaly detection (closest in spirit to our work), but targets traffic anomalies rather than authenticating the specific physical device producing the traffic. Avatefipour et al. [22] propose physical-layer ECU fingerprinting via clock behavior, but require network-level packet capture. None of these approaches can answer: "Is this the *same physical ECU* that was originally installed?"

2.2 Physical Unclonable Functions (PUFs)

PUFs exploit silicon-level manufacturing variations to generate device-specific challenge-response pairs [7]. Arbiter PUFs [23] and SRAM PUFs [24] have been studied as lightweight authentication primitives for IoT and automotive ECUs. While theoretically strong, PUFs require dedicated circuit modifications during manufacturing, making them impractical for authenticating existing deployed ECUs. Moreover, machine learning modeling attacks [25] have compromised PUF security with prediction accuracy exceeding 90%, and aging effects degrade reliability over time [6].

2.3 Hardware Fingerprinting Approaches

RF Fingerprinting [8] leverages hardware imperfections in wireless transmissions but is inherently limited to wireless-capable devices. Brik et al. [26] demonstrate radiometric signatures for 802.11 device identification, while deep learning approaches [27] achieve >95% accuracy in controlled environments. In-vehicle CAN bus nodes are wired, so RF fingerprinting simply does not apply.

MCU-Token [6] (NDSS 2024) achieves 97.78% TPR by combining SRAM initial states, dual-clock phase features, and flash read times across 60 MCU devices. However, it requires firmware modification to expose internal memory states, which is not feasible for sealed production ECUs.

Clock Skew Analysis [9, 28] achieves 100% accuracy on same-model Arduino boards but requires 70+ network packets and active network connectivity, making it unsuitable for offline or manufacturing-time authentication.

Behavioral Fingerprinting [12] identifies 25 Raspberry Pi boards at 91.9% accuracy via behavioral characteristics, but tests different device *models* rather than same-model clones. Network traffic-based fingerprinting [29] classifies IoT device *types* but cannot distinguish units of the same model.

Analog Signal Fingerprinting: Gerdes et al. [10] identify Ethernet devices via PHY-layer analog characteristics using a matched filter approach. Danev et al. [30] fingerprint RFID devices via physical-layer characteristics. Mohammed et al. [11] demonstrated signal-based fingerprinting on function generator channels with 100% accuracy using 38 features, but did not address real microcontroller hardware, same-model clone detection, or vehicular security contexts. Shaout et al. [36] extract electrical noise from PWM signals of same-model Arduino Uno boards and train a CNN for device identification, achieving 94% accuracy on a 2-device closed set — validating the feasibility of noise-based fingerprinting, though limited in device population and without cross-session or multi-type evaluation.

ML Classifiers for Fingerprinting: Random Forest [31] provides robust ensemble classification for high-dimensional feature spaces, while gradient boosting [32] and stacked ensemble architectures [33] further improve accuracy on heterogeneous device populations.

2.4 Summary and Gap

Table 1 compares existing approaches with our work across key dimensions relevant to vehicular deployment.

No existing work simultaneously achieves same-model clone detection, non-invasive operation (no firmware modification), offline capability, and applicability to wired embedded devices: the exact requirements for practical ECU authentication in deployed vehicular systems.

3 Threat Model

Assets: Legitimate ECUs, CAN bus nodes, and sensor modules in a vehicle network whose identities must be authenticated.

Table 1: Comparison of Hardware Fingerprinting Approaches for Embedded Device Authentication

Work	Acc.	#Dev	Clone	No FW Mod	Offline	Wired	Signal / Method
MCU-Token [6]	97.8%	60	△	✗	✓	✓	SRAM state + dual-clock + flash time
Clock Skew [9]	100%	–	✓	✓	✗	✓	Network packet timing (70+ pkts req.)
RF Fingerprint [8]	95%+	–	✗	✓	✗	✗	RF emission (wireless only)
Behav. FP [12]	91.9%	25	△	✗	✗	✓	Behavioral / network traffic
Gerdes et al. [10]	99%	–	✗	✓	✗	✓	Analog PHY matched filter
Mohammed et al. [11]	100%	–	✗	✓	✓	✓	Analog signal (function gen. only)
Shaout et al. [36]	94%	2	✓	✓	✓	✓	PWM electrical noise (CNN, 2-device)
This work	**99.5%**⋆	**8**	✓	✓	✓	✓	Digital switching transient (4 types)

✓ Yes ✗ No △ Partial. Clone = same-model unit detection. No FW Mod = no firmware modification required. Offline = no live network needed.

⋆99.5% with controlled mounting (non-conductive standoffs); 93.6% without. – = not reported or not applicable.

Adversary Capabilities: We assume an adversary who can:

- Physically replace a legitimate ECU with an identical-model device (hardware substitution attack)
- Obtain valid cryptographic credentials from the legitimate device or the OEM
- Program the replacement device with arbitrary firmware
- Optionally modify firmware to mimic the original device's software behavior

Adversary Limitations: The adversary *cannot* replicate the exact manufacturing-induced analog characteristics of the original device's output driver circuits, clock sources, and passive components. These physical properties are determined by silicon-level process variations at nanometer scale and cannot be cloned by reflashing or credential transfer.

Defense Goal: Detect device substitution by comparing the analog switching waveform of a device under test against an enrolled reference fingerprint, providing authentication that is independent of and complementary to cryptographic checks.

Enrollment Assumption: The system operator has physical access to legitimate devices during an enrollment phase (e.g., at the manufacturing plant or during vehicle commissioning) to capture reference fingerprints.

Out of Scope: This work does not address the threat of genuine hardware running modified or malicious firmware. Physical-layer fingerprinting captures hardware manufacturing characteristics below the firmware abstraction boundary and cannot directly detect firmware tampering. This threat is best addressed by complementary software attestation mechanisms (secure boot, code signing, TPM-based remote attestation). Our approach and software attestation form orthogonal and complementary layers of a defense-in-depth architecture: physical-layer fingerprinting answers *"Is this the correct physical device?"* while software attestation answers *"Is this device running the correct software?"*. A complete automotive device authentication system should combine both layers.

Ethical Considerations: This research involves non-invasive passive measurement of analog signals on embedded devices owned and operated by the authors in a controlled laboratory setting. No human subjects, personal data, or production vehicle systems were involved. The security benefits (detecting counterfeit ECUs, hardware substitution attacks) substantially outweigh any risks; the technique requires physical probe access, limiting its potential for misuse compared to remote attack vectors.

4 Methodology

Fig. 1 provides an overview of the complete system pipeline: signal acquisition, feature extraction, hybrid classification, and authentication decision.

4.1 Signal Acquisition

We capture analog waveforms using a Red Pitaya STEMlab 125-14 [5], selected for three specific reasons: (1) **Input range:** its selectable ± 1 V / ± 20 V range accommodates the full 3.3–5 V logic swing of all target microcontrollers, whereas prior ADC boards (e.g., AD9467 with ± 1.25 V input) require external attenuation that degrades signal fidelity; (2) **Resolution and rate:** 14-bit ADC at 125 MS/s provides millivolt-scale sensitivity and 125 samples per 1 MHz cycle, sufficient to resolve sub-microsecond transition edges where manufacturing variations are most visible; (3) **Automation:** the board runs Linux with SCPI-over-Ethernet, enabling scripted, unattended data collection across all 8 devices. In a vehicular deployment, the same hardware would probe a CAN bus data line, GPIO output, or ECU digital control signal. Each acquisition captures 16,384 samples ($\sim$131 complete cycles), providing rich statistical characterization of a device's switching behavior.

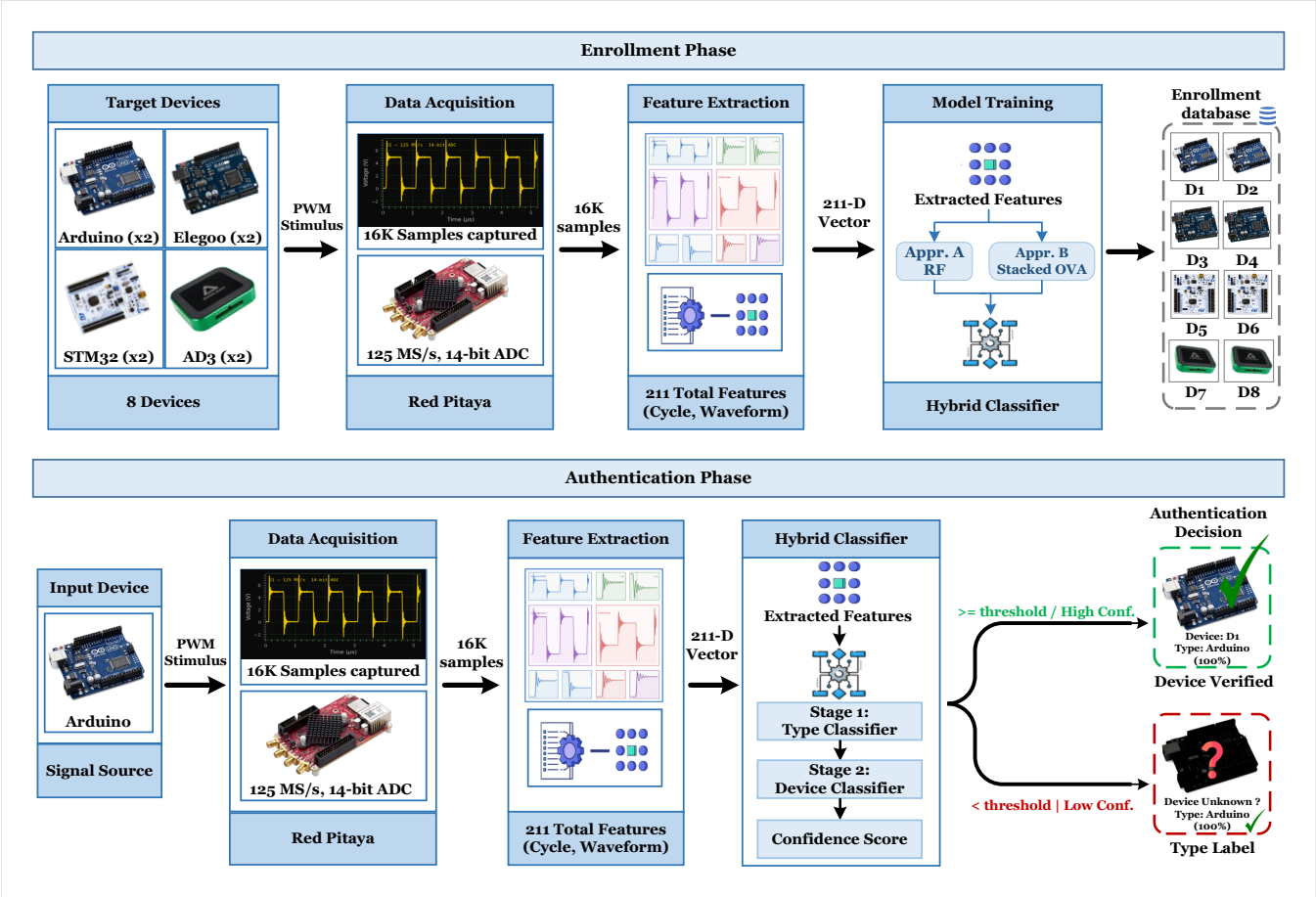


Figure 1: System architecture overview. **Enrollment Phase (top):** Target devices generate a 1 MHz PWM stimulus; the Red Pitaya STEMlab 125-14 captures 16K samples at 125 MS/s (14-bit ADC); 211 features (cycle-based and waveform) are extracted and used to train the hybrid classifier (Approach A: Random Forest; Approach B: Stacked OVA ensemble); fingerprints are stored in the enrollment database. **Authentication Phase (bottom):** An input device undergoes the same acquisition and feature extraction pipeline; the hybrid classifier applies Stage 1 type routing followed by Stage 2 device classification; high-confidence predictions ($\geq$threshold) yield a specific device identity (D1), while low-confidence outputs fall back to a guaranteed type label (graceful degradation).

Fig. 2 shows the experimental setup: the Red Pitaya ADC probes the device under test (DUT) digital output line while the DUT generates a 1 MHz ON/OFF switching signal. No firmware modification is required; the probe attaches non-invasively to the signal line.

4.2 Feature Extraction (211 Features)

Raw waveforms are processed through two complementary extractors:

Cycle-Based Features (94): The repetitive structure of ON/OFF switching (CAN bits, PWM cycles, GPIO toggles) provides natural cycle boundaries. Per-cycle measurements across five categories (timing (12), shape (18), transitions (24), stability (16), aggregate statistics (24)) capture fine-grained device behavior. Timing features (rise time $t_r = t_{90\%} - t_{10\%}$, duty cycle $D = t_{high}/T$) reflect output driver characteristics and clock precision. Shape features (overshoot, ringing amplitude) encode parasitic RLC behavior specific to each device's physical layout.

Enhanced Features (117): Holistic signal characterization via time-domain statistics (15), frequency-domain (FFT magnitudes, spectral centroid, 42 features), statistical moments (24), edge analysis (18), and discrete wavelet transform (Daubechies-4, 18 features) [34]. These capture global properties complementary to per-cycle measurements.

Fig. 3 shows representative switching waveforms from all 8 devices. The top row illustrates that different device types exhibit visually distinct switching signatures, supporting 100% type-level classification. The bottom row reveals that even within same-model pairs, subtle differences in slew rate,

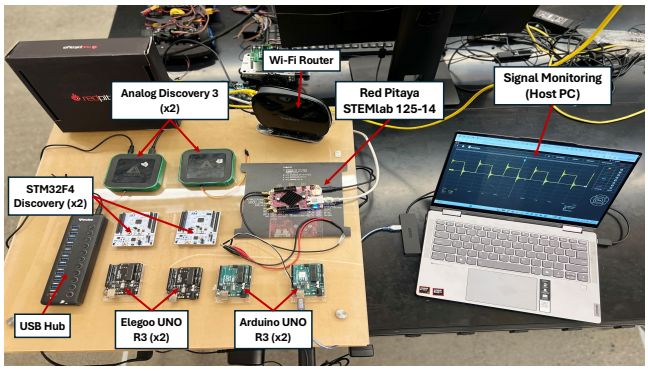


Figure 2: Experimental setup: Red Pitaya STEMlab 125-14 probing the digital output of a device under test (DUT). The same setup applies to CAN bus line monitoring in a vehicular context.

overshoot amplitude, and ringing behavior are consistently observable at the rising edge, forming the basis for same-model clone detection.

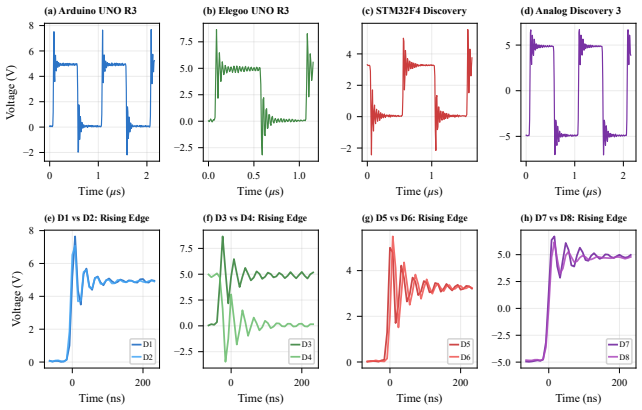


Figure 3: ON/OFF switching waveforms for all 8 devices. **Top (a–d):** Full cycles per type show distinct inter-type signatures. **Bottom (e–h):** Rising edge zoom reveals subtle intra-pair differences exploited for clone detection.

Feature importance analysis confirms that transition timing features dominate: rise time, fall time, and their statistical moments account for 6 of the top 10 discriminative features. This validates the core hypothesis: abrupt ON/OFF transitions expose intrinsic hardware behavior most directly.

4.3 Hybrid Classification

Approach A (Cycle-Based Random Forest): 94 cycle-based features with z-score normalization, 100 decision trees [31]. *Why Random Forest?* It handles correlated features naturally (our 94 features include mean, std, skewness, and kurtosis of the same underlying measurements), provides built-in feature importance for interpretability, is robust to overfitting on moderate-size datasets via bootstrap aggregation, and generalizes well across sessions without hyperparameter tuning. It is optimized for Arduino/Elegoo (ATmega328P) devices, which exhibit stable, low-variance cycle-to-cycle timing where per-cycle statistics are the most discriminative signal.

Approach B (OVA Stacked Ensemble): All 211 features with min-max normalization. *Why OVA Gradient Boosting + meta-learner?* For STM32 and AD3 devices, the 8-class boundary is complex and heterogeneous; a single Random Forest struggles because the globally optimal splits are pulled toward the easy pairs (AD3) and underfit the hard ones (STM32). OVA decomposition trains one specialist binary classifier per device, allowing each to focus on the feature subspace most discriminative for that specific device vs. all others [32]. Gradient Boosting is chosen over Random Forest for the binary classifiers because it iteratively corrects residual errors, which is critical when inter-device differences are small. A Logistic Regression meta-learner then combines the 8 OVA probability scores [33], learning to resolve conflicts (e.g., when both D5 and D6 specialists report high confidence) and recalibrate overconfident base classifiers.

Hybrid Routing: A dedicated Random Forest type classifier, trained on all 211 features and evaluated separately from Approach B, achieves 100% accuracy because different hardware types produce clearly separable feature distributions. We exploit this by first predicting device type, then routing to the optimal approach:

- Arduino / Elegoo (ATmega328P-based) → Approach A
- STM32 / Analog Discovery 3 → Approach B

When device-level confidence falls below a threshold (70% for A, 90% for B), the system reports type-level identification, which is always 100% accurate, providing graceful degradation suitable for safety-critical vehicle applications.

Algorithm 1 formalizes the complete hybrid classification procedure.

5 Experimental Evaluation

5.1 Setup

We evaluate on 8 devices across 4 hardware types (Table 2), specifically selecting same-model pairs to assess clone detection, which is the hardest case in the hardware substitution threat model. Data collection spans 3 sessions on different days (6,000 samples/device total). Cross-session evaluation (train on S1+S2, test on S3) assesses temporal stability across different days, while real-time evaluation tests live deployment with immediate classification.

5.2 Data Collection Protocol

Data was collected across 3 sessions on different days to evaluate cross-session generalization (Table 3). Each session

Algorithm 1 Hybrid Classification with Graceful Degradation

Require: Signal $\mathbf{x}$, models M_T (type classifier), M_A (Approach A), M_B (Approach B)
Ensure: Device ID or type label, with confidence score

$\mathbf{f}_c \leftarrow$ ExtractCycleFeatures($\mathbf{x}$) {94 features}
$\mathbf{f}_e \leftarrow$ ExtractEnhancedFeatures($\mathbf{x}$) {117 features}
$\mathbf{f} \leftarrow [\mathbf{f}_c, \mathbf{f}_e]$ {211 combined}
// Stage 1: Type Routing (100% accurate)
$type \leftarrow M_T$.Predict($\mathbf{f}$) {Dedicated RF type classifier}
// Stage 2: Device-Level Classification
if $type \in$ {Arduino, Elegoo} **then**
    $dev, conf \leftarrow M_A$.Predict($\mathbf{f}_c$); $\tau \leftarrow 0.70$
else
    $dev, conf \leftarrow M_B$.Predict($\mathbf{f}$); $\tau \leftarrow 0.90$
end if
// Graceful Degradation
if $conf \geq \tau$ **then**
    return dev, $conf$
else
    return $type$, $conf$ {Type always correct}
end if

Table 2: Target Device Specifications (Same-Model Pairs)

ID	Device	MCU/Core	Clock
D1	Arduino UNO R3 #1	ATmega328P	16 MHz
D2	Arduino UNO R3 #2	ATmega328P	16 MHz
D3	Elegoo UNO R3 #1	ATmega328P	16 MHz
D4	Elegoo UNO R3 #2	ATmega328P	16 MHz
D5	STM32 Nucleo-L476RG #1	STM32L476RG	80 MHz
D6	STM32 Nucleo-L476RG #2	STM32L476RG	80 MHz
D7	Analog Discovery 3 #1	FPGA-based	100 MHz
D8	Analog Discovery 3 #2	FPGA-based	100 MHz

comprises two runs (morning and evening) to capture intra-day environmental variations. Devices undergo a 5-minute warmup before each run; samples are captured at 1-second intervals.

Cross-session evaluation trains on Sessions 1+2 and tests on Session 3, which was collected on a different day under potentially different environmental conditions. Real-time evaluation deploys trained models for live signal capture and immediate classification (100 samples/device).

5.3 Results

We note that our evaluation addresses *closed-set identification*, which assigns an observation to one of N known enrolled devices. The operational deployment scenario ultimately requires *open-set verification*, which determines whether an observation matches a single enrolled reference. Closed-set identification accuracy is a necessary condition for open-set verification and is the standard evaluation framework in hardware fingerprinting research [6, 10]. Extending our framework to open-set verification via confidence thresholding against enrolled reference templates is identified as future work.

Table 3: Data Collection Summary

Session	Day	Runs	/Device	Usage
Session 1	1	2	2,000	Training
Session 2	2	2	2,000	Training
Session 3	3	2	2,000	Testing
Total	—	6	**6,000**	48,000 total

Table 4 summarizes performance across all evaluation conditions.

Table 4: Classification Results Summary

Condition	Appr. A	Appr. B	Hybrid
Cross-Session Acc.	86.58%	90.05%	**87.64%**
Real-Time Acc.	88.6%	79.7%	**99.5%**
Type Acc. (both)	**100%**	**100%**	**100%**
Avg. Confidence (hybrid)	N/A	N/A	85.6%

Cross-Session Performance: Approach B achieves 90.05% overall, with per-device accuracy ranging from 66.75% (D4, Elegoo clone) to 100% (D7/D8, Analog Discovery 3). The AD3 pair achieves perfect separation due to precision instrumentation-grade circuitry. The Elegoo clone pair is hardest (67–88%) because clone manufacturers optimize for cost compatibility, reducing inter-unit manufacturing variation.

Clone Pair Analysis (D3/D4): During initial real-time testing, D4 showed degraded accuracy (66.6%) while all other same-manufacturer pairs, including D1/D2 (both Arduino UNO R3), D5/D6 (both STM32 Nucleo), and D7/D8 (both Analog Discovery 3), all achieved 99–100%. This rules out the same-manufacturer relationship as the root cause. Upon investigation, we identified the actual cause as physical: parasitic capacitance coupling between D4's PCB ground plane and the conductive lab table surface was shifting its signal Vpp by approximately 0.15–0.25 V, sufficient to push D4's feature vectors across the decision boundary. D4 was more susceptible due to its board position and contact area with the table surface in our particular setup. D3 and D4 do share the same ATmega328P chip, identical PCB design, and process parameters, which results in tightly overlapping feature distributions. Specifically, the top discriminating features (half_std_diff, undershoot, ringing_amplitude) overlap in approximately 20–30% of their range for D3/D4, compared to below 5% for cross-manufacturer pairs (Arduino vs. Elegoo). However, this structural overlap alone did not prevent accurate classification once the environmental coupling was eliminated. Table 5 shows the impact of electrical isolation via non-conductive

plastic standoffs on all affected devices, measured in a dedicated isolation study conducted to isolate and quantify the parasitic capacitance effect. Note that the "Original" column reflects values from this dedicated isolation study; the overall 93.6% initial result in Table 4 is from a separate earlier evaluation under different mounting conditions.

Table 5: Effect of Electrical Isolation (3D-Printed Standoffs) on Real-Time Accuracy.

Device	Type	Original	With Standoffs
D1 — Arduino UNO #1	Arduino	99.1%	99.0%
D2 — Arduino UNO #2	Arduino	99.5%	100.0%
D3 — Elegoo UNO #1	Elegoo	87.8%	100.0%
D4 — Elegoo UNO #2	Elegoo	66.6%	98.0%
D5 — STM32 #1	STM32	82.5%	100.0%
D6 — STM32 #2	STM32	88.1%	99.0%
D7 — AD3 #1	AD3	100.0%	100.0%
D8 — AD3 #2	AD3	100.0%	100.0%

This finding establishes **electrical isolation as a required measurement protocol** for reliable device fingerprinting and has direct implications for the adversarial model: an attacker who deliberately introduces ground coupling (e.g., by placing a conductive surface beneath the target ECU) could cause D4 to be systematically misclassified as D3, masking a device swap. Our hybrid framework mitigates this through its tiered fallback: even when instance-level confidence is low for the D3/D4 pair, the Stage 1 type classifier correctly identifies the device as an "Elegoo UNO" with 100% confidence, providing meaningful security assurance even when instance-level separation is ambiguous.

Real-Time Deployment: Approach B suffers significant degradation for certain devices (D2: 0% real-time vs. 99.95% cross-session) due to distribution shift from environmental factors during live capture. Approach A is more robust for Arduino/Elegoo but underperforms for STM32/AD3. The hybrid strategy resolves this complementarity. In initial evaluation (without controlled mounting), the hybrid achieves **93.6% real-time accuracy**. After applying the electrical isolation protocol described above (non-conductive 3D-printed standoffs), the hybrid accuracy improves to **99.5%**, with five of eight devices reaching 100%; Table 6 presents the per-type breakdown for the initial evaluation.

Type Classification: 100% accuracy in both conditions. In the vehicle security threat model, this enables detection of cross-type substitution (e.g., a counterfeit Arduino substituted for an STM32 ECU) with certainty, while within-type clone detection achieves 67–100% depending on device pair.

Table 6 summarizes hybrid strategy real-time performance per device type, including routing decision and average confidence.

Table 6 shows that STM32 and AD3 pairs achieve 100% real-time identification with high confidence (98.6–99.8%),

Table 6: Hybrid Strategy Real-Time Performance by Type (Initial Evaluation, Without Controlled Mounting)

Type (Pair)	Route	Acc. (%)	Conf. (%)
Arduino (D1/D2)	A	90.0	81.5
Elegoo (D3/D4)	A	84.5	62.5
STM32 (D5/D6)	B	100.0	98.6
AD3 (D7/D8)	B	100.0	99.8
Hybrid Overall		**93.6**	**85.6**

while ATmega328P-based pairs (Arduino/Elegoo) reach 84.5–90% at lower confidence (62.5–81.5%) due to their shared oscillator heritage reducing inter-unit variation. Notably, Approach B fails completely on D2 (0% real-time vs. 99.95% cross-session): during live capture, D2's feature distribution shifted outside its training boundary, causing systematic misclassification as D1 (its same-model pair); routing Arduino/Elegoo to Approach A, which is more robust to such shifts, prevents this failure.

Fig. 4 shows the confusion matrix for the hybrid classifier under controlled mounting conditions (standoffs); five of eight devices reach 100%, with minor residual errors on D1 (99%), D4 (98%), and D6 (99%). Cross-type misclassification does not occur under either evaluation condition.

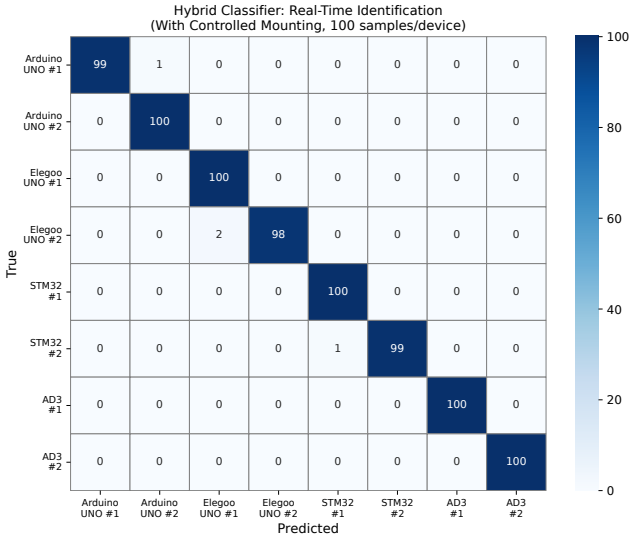


Figure 4: Confusion matrix for the hybrid classifier under real-time evaluation with controlled mounting conditions (non-conductive 3D-printed standoffs, 100 samples/device). Diagonal entries show per-device accuracy; five of eight devices reach 100%, with minor residual errors on D1 (99%), D4 (98%), and D6 (99%).

6 Discussion

6.1 Physical Transferability to CAN Bus Signals

Our current validation uses PWM as a representative ON/OFF switching stimulus. CAN bus NRZ encoding produces dominant/recessive voltage transitions on the CAN_H/CAN_L differential pair, which are fundamentally ON/OFF switching events at the driver output. Each CAN frame contains hundreds of bit transitions, providing ample signal for fingerprint extraction without modifying the vehicle network or ECU firmware.

Shared physics: Every digital state transition, whether a GPIO toggle, a PWM pulse edge, or a CAN bit transition, forces the output driver MOSFET through the same underlying switching transient, determined by transistor threshold voltage, on-resistance, and parasitic capacitances of PCB traces, bond wires, and package parasitics. These are manufacturing-process artifacts unique to each device. CAN transceivers (e.g., MCP2551, TJA1050) contain analogous output driver circuitry, and prior work has confirmed that CAN bit transitions carry device-specific analog characteristics [38]. Our 211-feature extraction pipeline targets exactly these transient characteristics.

CAN bit transitions share this same output driver physics, but differ from our periodic ON/OFF switching signal in ways that warrant direct validation: CAN transitions are data-dependent (bit stuffing determines when they occur rather than a fixed clock), the differential bus topology introduces loading and reflection effects from multiple connected transceivers, and CANH/CANL differential signaling partially rejects common-mode noise while altering the measurable waveform shape. These are engineering considerations for a CAN deployment, not fundamental barriers to fingerprinting; the device-specific transient characteristics originate at the driver level and are present regardless of the higher-level protocol. Validating discriminability under bus loading and data-dependent transition patterns on real CAN hardware is the primary direction for future work.

6.2 Comparison with Existing Vehicle Security

SecOC (Secure Onboard Communication) provides message authentication via MACs but cannot detect hardware substitution if keys are compromised [3, 19]. Emerging V2X standards (IEEE 802.11p, C-V2X) [35] also rely on PKI-based authentication that cannot detect physical device substitution. Our approach is *complementary* to these cryptographic methods: operating at the physical layer, below the communication stack, it requires no changes to ECU software or vehicle network protocol and provides physical-layer device verification that cryptographic methods alone cannot achieve. We acknowledge that the re-enrollment policy for legitimate device replacement is a critical system design challenge: re-enrollment authorization must combine cryptographic credentials with physical presence requirements (e.g., at an authorized service facility) to prevent an adversary with cloned credentials from authorizing re-enrollment of a swapped device. Under such a policy, our approach and cryptographic authentication together raise the bar for hardware substitution attacks substantially beyond what either provides alone.

6.3 Limitations and Future Work

Limitations. Physical probe access is required, acceptable for manufacturing-time enrollment and maintenance but not continuous monitoring without installed sensing hardware. Within-pair accuracy for the Elegoo clone pair (D3: 100%, D4: 98% under controlled mounting) reflects reduced inter-unit variation in cost-optimized clones; the adversarial implications of this narrow decision boundary are discussed in Section 5.3.

Bench vs. automotive environment: Our experiments were conducted in a controlled bench environment with regulated power supplies. Real automotive deployments introduce multi-ECU bus loading, shared 12 V battery supply variation from concurrent high-current loads (HVAC, lighting), and long wiring harnesses (up to 10 m or more). Notably, longer transmission lines are expected to *improve* fingerprint discriminability rather than degrade it: the wire's parasitic inductance and capacitance act as a dispersive channel that stretches the sub-microsecond switching transient in time, making device-specific edge characteristics more observable at the measurement point. The most challenging geometry for our approach is therefore close-proximity measurement, which is what we test. Crosstalk from adjacent wires, multi-transceiver bus loading, and termination impedance effects nonetheless require systematic evaluation, which we plan to conduct on our Danlaw vehicle test bench.

Environmental and temporal variation: Our evaluation spans three sessions across different days at stable lab temperature (≈20–25°C). Automotive operating conditions span −40°C to +85°C, with additional variation from humidity, vibration, and component aging over years of operation. Temperature-induced changes in transistor threshold voltages and passive component values are expected to shift feature distributions; our parasitic capacitance finding (a 0.25 V Vpp shift causing misclassification) suggests these effects may be significant. Temperature-controlled longitudinal testing is a critical direction for future work.

Population size: Our evaluation uses 8 devices across 4 hardware types (2 per type), sufficient to demonstrate framework feasibility and the hardest adversarial case (same-model clone detection), but not sufficient to characterize large-population performance or false rejection rates across hundreds of units of the same model. Large-population validation is required before deployment claims can be made.

Accuracy for safety-critical use: The current 99.5% overall accuracy under controlled mounting (93.6% in initial evaluation without standoffs) is promising, but not sufficient for hard security decisions in production automotive systems, where measurement conditions cannot be assumed controlled. Our framework is designed for a tiered advisory output, in which high-confidence predictions trigger hard security decisions while low-confidence predictions produce type-level advisory alerts, which is appropriate for the current state of the technology. Achieving automotive-grade reliability will require large-population validation, environmental robustness testing, and extended longitudinal evaluation.

Future Directions. We plan to: (i) validate on live CAN bus traffic from production ECUs on our Danlaw test bench, where CAN bit transitions share the same output driver physics as our PWM stimulus; (ii) explore higher ADC sampling rates (500 MS/s–1 GS/s) to resolve sub-nanosecond transition details and widen the inter-device feature margin for same-model clone pairs; (iii) deploy on embedded hardware using Zynq SoC HW/SW co-design [37], partitioning feature extraction to the programmable logic (PL) and ML inference to the ARM processing system (PS) for on-board continuous authentication; (iv) study adversarial robustness against deliberate hardware modifications designed to spoof a target device's fingerprint; and (v) extend the framework to LIN bus, 100BASE-T1 automotive Ethernet, and FlexRay protocols with open-set enrollment support for dynamically added devices.

7 Conclusion

Counterfeit and substituted ECUs in automotive supply chains pose a hardware-level threat that cryptographic authentication alone cannot address. Once valid credentials are transferred to a rogue device, software-based authentication has no way to detect the swap. Physical-layer fingerprinting addresses this gap directly by exploiting manufacturing variations that are intrinsic to the silicon itself and cannot be copied through reflashing or credential transfer.

We presented a non-invasive hardware fingerprinting framework with three core contributions: (1) a 211-feature extraction pipeline combining per-cycle timing and shape measurements with time-domain statistics, FFT-based spectral features, and discrete wavelet transform coefficients, validated to capture device-specific imperfections in digital switching transients; (2) a hybrid two-stage classifier that first routes by device type with 100% accuracy and then applies the best-fit model per type, specifically a cycle-based Random Forest for ATmega328P devices and an OVA stacked ensemble for STM32 and FPGA-based devices; and (3) a graceful degradation mechanism that falls back to guaranteed type-level identification when device-level confidence is insufficient, providing meaningful security assurance even under challenging conditions.

Evaluated on 8 embedded devices across 4 hardware types, including same-model clone pairs representing the hardest adversarial case, the framework achieves **99.5% real-time accuracy** under controlled mounting conditions (**93.6%** without) and **100% device-type accuracy** in all conditions, without firmware modification or network access. Cross-session evaluation confirms temporal stability across different days. A key finding is that parasitic capacitance coupling from conductive surfaces shifts feature distributions sufficiently to cause misclassification, which establishes electrical isolation as a required measurement protocol and motivates investigation of environmental robustness as the primary challenge for automotive-grade deployment. This work establishes physical-layer switching analysis as a validated, complementary authentication primitive that operates below the firmware abstraction boundary, providing a foundation for future integration with cryptographic mechanisms in defense-in-depth automotive security architectures.

Acknowledgment

This research was carried out in the Intelligent Systems and Security Forensics (ISSF) Laboratory at the University of Michigan-Dearborn. The authors gratefully acknowledge the support of the National Science Foundation (NSF) under awards #2414729, #2214830, and #2035770. The authors also wish to thank the anonymous reviewers for their thoughtful and constructive comments, which significantly improved the quality of this work.

References

[1] K. Koscher et al., "Experimental Security Analysis of a Modern Automobile," in *Proc. IEEE S&P*, 2010.

[2] C. Miller and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle," *Black Hat USA*, 2015.

[3] AUTOSAR, "Specification of Secure Onboard Communication (SecOC)," AUTOSAR Classic Platform, Release R21-11, 2021.

[4] A. Ferdous et al., "Device Fingerprinting for Cyber-Physical Systems: A Survey," *ACM Computing Surveys*, vol. 55, no. 12, 2023.

[5] Red Pitaya, "STEMlab 125-14 Documentation," [Online]. Available: https://redpitaya.readthedocs.io/

[6] J. Zhang et al., "From Hardware Fingerprint to Access Token: Enhancing the Authentication on IoT Devices," in *Proc. NDSS*, 2024.

[7] A. Mostafa et al., "Physical Unclonable Functions (PUF) for IoT Devices," *ACM Computing Surveys*, 2023.

[8] Y. Wang et al., "Radio Frequency Fingerprint Identification for Internet of Things: A Survey," *Security and Safety*, vol. 3, 2024.

[9] Y. Cao et al., "CADFA: A Clock Skew-Based Active Device Fingerprint Authentication Scheme for Class-1 IoT Devices," *IEEE IoT Journal*, 2024.

[10] R. M. Gerdes et al., "Device Identification via Analog Signal Fingerprinting: A Matched Filter Approach," in *Proc. NDSS*, 2012.

[11] A. W. Mohammed et al., "A Lightweight IDS Framework Using FPGA-Based Hardware Fingerprinting on Zynq SoC," in *Proc. IEEE CSR*, 2025.

[12] P. Sanchez et al., "A methodology to identify identical single-board computers based on hardware behavior fingerprinting," *Computers & Security*, vol. 123, 2022.

[13] M. Marchetti and D. Stabili, "Anomaly Detection of CAN Bus Messages through Analysis of ID Sequences," in *Proc. IEEE IV*, 2017.

[14] W. Choi et al., "Voltageids: Low-Level Communication Characteristics for Automotive Intrusion Detection System," *IEEE Trans. Information Forensics and Security*, vol. 15, 2020.

[15] T. Hoppe, S. Kiltz, and J. Dittmann, "Security Threats to Automotive CAN Networks—Practical Examples and Selected Short-Term Countermeasures," *Reliability Engineering & System Safety*, vol. 96, no. 1, pp. 11–25, 2011.

[16] Upstream Security, "Global Automotive Cybersecurity Report," [Online]. Available: https://upstream.auto/research/global-automotive-cybersecurity-report/, 2023.

[17] ISO/SAE, "ISO/SAE 21434: Road Vehicles—Cybersecurity Engineering," International Standard, 2021.

[18] UNECE, "UN Regulation No. 155: Cybersecurity and Cybersecurity Management Systems," WP.29, 2021.

[19] J. Petit and S. E. Shladover, "Potential Cyberattacks on Automated Vehicles," *IEEE Trans. Intelligent Transportation Systems*, vol. 16, no. 2, pp. 546–556, 2015.

[20] SEMI, "Counterfeit Electronic Parts Avoidance," SEMI Standard G87-0312, 2022.

[21] A. Taylor, N. Japkowicz, and S. Leblanc, "Frequency-Based Anomaly Detection for the Automotive CAN Bus," in *Proc. IEEE WCICSS*, 2015.

[22] O. Avatefipour et al., "Physical-Fingerprinting of Electronic Control Unit (ECU) Based on Inter-Packet Arrival Time," *IEEE Access*, vol. 7, 2019.

[23] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon Physical Random Functions," in *Proc. ACM CCS*, 2002.

[24] D. E. Holcomb, W. P. Burleson, and K. Fu, "Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers," *IEEE Trans. Computers*, vol. 58, no. 9, pp. 1198–1210, 2009.

[25] U. Rührmair et al., "Modeling Attacks on Physical Unclonable Functions," in *Proc. ACM CCS*, 2010.

[26] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless Device Identification with Radiometric Signatures," in *Proc. ACM MobiCom*, 2008.

[27] K. Merchant, S. Revay, G. Stantchev, and B. Nousain, "Deep Learning for RF Device Fingerprinting in Cognitive Communication Networks," *IEEE J. Select. Areas Commun.*, vol. 36, no. 10, 2018.

[28] T. Kohno, A. Broido, and K. C. Claffy, "Remote Physical Device Fingerprinting," *IEEE Trans. Dependable and Secure Computing*, vol. 2, no. 2, pp. 93–108, 2005.

[29] M. Miettinen et al., "IoT Sentinel: Automated Device-Type Identification for Security Enforcement in IoT," in *Proc. IEEE ICDCS*, 2017.

[30] B. Danev and S. Čapkun, "Transient-Based Identification of Wireless Sensor Nodes," in *Proc. ACM/IEEE IPSN*, 2009.

[31] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[32] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[33] A. Sivanathan et al., "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics," *IEEE Trans. Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2019.

[34] S. Mallat, *A Wavelet Tour of Signal Processing*, 3rd ed. Academic Press, 2008.

[35] L. Liang, H. Ye, G. Yu, and G. Y. Li, "Vehicular Communications: A Physical Layer Perspective," *IEEE Trans. Vehicular Technology*, vol. 66, no. 12, pp. 10647–10659, 2017.

[36] A. Shaout, A. Hasan, and M. Shravan, "Hardware Embedded Fingerprinting Based on Electric Noise," in *Proc. 25th Int'l Arab Conf. on Information Technology (ACIT)*, IEEE, 2024. DOI: 10.1109/ACIT62805.2024.10877015

[37] A. Ali et al., "Speed vs. Efficiency: A Framework for High-Frequency Trading Algorithms on FPGA Using Zynq SoC Platform," *Alexandria Engineering Journal*, vol. 93, pp. 276–290, 2024. https://doi.org/10.1016/j.aej.2024.03.064

[38] P.-S. Murvay and B. Groza, "Source Identification Using Signal Characteristics in Controller Area Networks," *IEEE Signal Processing Letters*, vol. 21, no. 4, pp. 395–399, 2014.



Artifact Appendix: "Short: Is Your ECU Really Yours?"

Abbas Ali *Hafiz Malik*
Information Systems, Security and Forensics Lab
University of Michigan-Dearborn, Dearborn, USA
{mrabbas, hafiz}@umich.edu

A Artifact Appendix

A.1 Abstract

This artifact supports the cross-session hardware fingerprinting results in Table 4 of the paper. The system distinguishes 8 embedded devices across 4 hardware types (Arduino UNO R3, Elegoo UNO R3, STM32 Nucleo, Analog Discovery 3) by analyzing the analog characteristics of a periodic GPIO switching signal captured at 125 MS/s with a Red Pitaya STEMlab 125-14 Pro Gen 2. The artifact includes the full raw waveform dataset (≈2.6 GB), pre-extracted feature CSVs, pre-trained models, and Jupyter notebooks for evaluation, training, and real-time classification. Running the evaluation notebook alone is sufficient to reproduce all three Table 4 accuracy results without any hardware.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None. The dataset contains only GPIO switching waveforms from off-the-shelf embedded development boards. No personally identifiable information is included. The notebooks and models pose no risk to evaluator machines.

A.2.2 How to access

The artifact is publicly archived on Zenodo as source code, notebooks, pre-extracted datasets, and pre-trained models. We provide the concept DOI (redirects to the latest version) to support artifact updates during the evaluation discussion period:
DOI: `10.5281/zenodo.19666939`
The archive extracts to a single `VehicleSec26_Artifact/` directory containing all data, models, and notebooks.

A.2.3 Hardware dependencies

Experiments E1–E3 (offline): Any standard laptop or desktop with at least 8 GB RAM. No GPU required. Full dataset requires ≈2.6 GB disk; running E1 from pre-extracted CSVs requires <50 MB.
Experiment E4 (real-time, optional): A Red Pitaya STEMlab 125-14 Pro Gen 2 connected to the target device over the local network (SSH/SCPI). Without this hardware, E1–E3 are sufficient to reproduce all paper results.

A.2.4 Software dependencies

OS: Any major OS (Windows, Linux, macOS) with Python 3.12.
Packages: Jupyter Notebook and the following (all via `pip install -r requirements.txt`): `numpy`≥`1.21`, `pandas`≥`1.3`, `scikit-learn`≥`1.0`, `scipy`≥`1.7`, `matplotlib`≥`3.4`, `seaborn`≥`0.11`, `joblib`≥`1.0`, `tqdm`≥`4.60`.

A.2.5 Benchmarks

48,000 waveform captures: 3 sessions × 8 devices × 2 runs × 1,000 captures. Each capture: 16,384 samples at 125 MS/s. Training set: Sessions 1–2 (32,000 samples). Test set: Session 3 (16,000 samples).

Label	Device
D1/D2	Arduino UNO R3 #1/#2
D3/D4	Elegoo UNO R3 #1/#2
D5/D6	STM32 Nucleo #1/#2
D7/D8	Analog Discovery 3 #1/#2

A.3 Set-up

A.3.1 Installation

1. Download and extract the artifact from Zenodo.
2. Run: `pip install -r requirements.txt`

3. Launch: `jupyter notebook`

A.3.2 Basic Test

Open `04_Evaluation/evaluate_results.ipynb` and run only the first two cells. These load the pre-extracted CSVs and pre-trained models from `05_Pretrained_Models/`. Successful completion without errors confirms all dependencies are installed correctly.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): The cycle-based Random Forest achieves **86.58%** cross-session accuracy (train Sessions 1–2, test Session 3), as reported in Table 4. Supported by E1.

(C2): The OVA stacked ensemble achieves **90.05%** cross-session accuracy, as reported in Table 4. Supported by E1.

(C3): The hybrid routing approach achieves **87.64%** cross-session accuracy, as reported in Table 4. Supported by E1.

A.4.2 Experiments

(E1): *Reproduce Table 4* [5 human-minutes + 5–10 compute-minutes + <50 MB disk]: Verifies claims C1, C2, and C3 using pre-extracted features and pre-trained models.
Preparation: No setup beyond the basic installation.
Execution: Open `04_Evaluation/evaluate_results.ipynb`. Run all cells (Kernel → Restart & Run All).
Results: The final cell prints accuracy for all three approaches. Expected: C1=86.58%, C2=90.05%, C3=87.64%. Minor variation (±0.05%) is acceptable across scikit-learn versions.

(E2): *Retrain from pre-extracted features (optional)* [10 human-minutes + ≈25 compute-minutes (Approach A) or ≈10 compute-hours (Approach B, full pipeline)]: Approach B full retraining comprises three stages: Stage 1 (8 OVA GradientBoosting classifiers, ≈96 min), Stage 2 (8-device × 5-fold meta-feature construction, ≈500 min), and Stage 3 (meta-learner selection, ≈1 min). Use pre-trained models from `05_Pretrained_Models/` to reproduce all Table 4 claims via E1 in under 10 minutes without retraining. **Note:** Training notebooks overwrite models in `05_Pretrained_Models/`. Run E1 first and optionally back up the folder before proceeding.
Execution: Run `03_Training/train_ApproachA_cycle_rf.ipynb` and/or `train_ApproachB_ova_stacked.ipynb`.
Results: Re-run E1 to confirm retrained models reproduce the same accuracy as C1–C3.

(E3): *Re-extract features from raw waveforms (optional)* [10 human-minutes + several compute-hours + 2.6 GB disk]: Requires downloading `Raw_Waveforms/` from Zenodo.
Execution: Open `02_Feature_Extraction/extract_features.ipynb`. Set `DATASET_DIR` to `01_Dataset/Raw_Waveforms/` and run all cells.
Results: Re-run E2 then E1 to confirm end-to-end reproducibility.

(E4): *Real-time live classification (optional, requires Red Pitaya hardware)* [variable]: Open `06_Realtime/realtime_evaluation_all_devices.ipynb`, set the Red Pitaya IP and device label, and run all cells for live capture and classification using the same pre-trained models as E1.

A.5 Notes on Reusability

The feature extraction pipeline works with any periodic GPIO switching signal at a compatible sample rate. Adding a new device requires training one additional OVA binary classifier and retraining the Logistic Regression meta-learner, without modifying existing classifiers.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.

ARTIFACT EVALUATED USENIX AVAILABLE
ARTIFACT EVALUATED USENIX FUNCTIONAL

RAIL: Risk-Aware Human-in-the-Loop Framework with Adaptive Intrusion Response for Autonomous Vehicles

Dawood Wasif[1] *Terrence J. Moore*[2] *Seunghyun Yoon*[3] *Hyuk Lim*[3]
Dan Dongseong Kim[4] *Frederica F. Nelson*[2] *Jin-Hee Cho*[1]

[1]*Virginia Tech*
[2]*US DEVCOM Army Research Laboratory*
[3]*Korea Institute of Energy Technology (KENTECH)*
[4]*The University of Queensland*

Abstract

Autonomous vehicles must remain safe and effective when encountering rare long-tailed scenarios or cyber–physical intrusions during driving. We propose a risk-aware human-in-the-loop framework, RAIL, that closes the detect–respond–learn loop within a unified runtime architecture. RAIL introduces an interpretable Intrusion Risk Index (IRI) that fuses heterogeneous runtime cues, including curvature actuation integrity, time-to-collision proximity, and observation-shift consistency, through a weighted Noisy-OR to enable probabilistic, source-decomposed risk quantification at control rate. When risk rises, a contextual bandit selects cue-specific shields and a graded authority blending mechanism interpolates between nominal and safeguarded actions, preserving efficiency under low risk while enabling proactive containment under threat. RAIL further integrates dual reward shaping and risk-prioritized replay so that takeovers and near misses guide learning. Across MetaDrive and CARLA, RAIL matches expert-level success while reducing safety violations and operator interventions relative to RL, safe RL, offline and imitation baselines. Under Controller Area Network (CAN) injection and LiDAR spoofing, RAIL reduces attack success rates by up to 48% and lowers disengagement under attack by more than 50%, showing that explicit multi-source risk modeling yields strong robustness gains in safety-critical autonomy.

1 Introduction

From Pilot Systems to Open-World Autonomy. Autonomous vehicles (AVs) are transitioning from controlled pilots to open-world deployment, where reliability must extend beyond nominal driving to adversarial and safety-critical conditions. Advances in large-scale simulation, high-capacity function approximation, and model-based and model-free control have produced competent lane keeping, merging, and urban navigation, often approaching human-level performance under benign settings [30]. However, deployment exposes a systems-level gap: detection, control, and learning remain loosely coupled. Real-world operation requires more than task competence. An AV must *detect* anomalous conditions, whether rare environmental events or deliberate cyber–physical intrusions, *translate* those detections into calibrated control adaptations at actuation rate, and *integrate* the outcome into learning so similar failures become less likely. No existing subsystem guarantees this closed-loop detect–respond–learn capability under adversarial conditions. Consider, for example, an autonomous vehicle in urban traffic when a compromised electronic control unit injects a bounded steering bias onto the Controller Area Network bus. The trajectory drifts within a few control cycles, yet the learned policy receives no structured signal attributing the anomaly to its source, and the human safety operator has not yet perceived the deviation. Scenarios of this form expose the gap between cyber-physical detection and calibrated control adaptation at the actuation rate that motivates this work.

Limits of Implicit Safety. A dominant class of approaches encodes safety *implicitly* within the learning objective: RL may violate constraints during exploration, Safe RL relies on carefully tuned scalar penalties that degrade under sparse cost signals [2, 40], and imitation or offline RL inherit distributional bias [6, 18, 24]. These methods treat safety as an external scalar constraint rather than an internal state variable, so the policy lacks an explicit representation of *what* failed or *which channel* is compromised. Safety boundaries are learned indirectly, offering neither decomposed attribution nor graded response at test time.

Limits of Purely Reactive Human Oversight. Human-in-the-loop (HITL) frameworks incorporate demonstrations, interventions, or shared control to improve safety [19, 23, 25, 36], with recent methods such as HACO [25] and HAIM-DRL [19] achieving strong nominal performance. However, when human correction is the dominant containment mechanism, the policy cannot act preemptively before the operator perceives the threat, and takeover is encoded as a binary override rather than a structured signal distinguishing perception, planning, and actuation hazards [23, 26]. The limitation lies in purely reactive use of the human, not in human oversight itself: calibrated automated mitigation executing before takeover is

required, with the operator retained for exception handling.

The Architectural Gap: From Implicit Penalties to Explicit Risk. The limits of scalar safety costs and reactive human intervention expose a structural gap in autonomy stacks. Existing methods treat safety as an outcome to penalize after violation rather than as a calibrated runtime variable that governs control. Implicit approaches infer safety boundaries from rewards or takeovers but do not expose a decomposable representation of risk. As a result, the system cannot explain *why* intervention occurred, *which subsystem* triggered concern, or provide an auditable trail aligned with emerging cybersecurity requirements such as UN R155 and ISO/SAE 21434 [20, 29]. Explicit risk modeling elevates risk to a first-class variable: a calibrated, multi-source estimate computed at control rate and directly coupled to shield selection and graded authority blending. Because risk is decomposed by source, mitigation becomes targeted rather than binary, and learning receives structured feedback. This shift to explicit, decomposed risk estimation is essential for adversarially robust and certifiable autonomous systems, as demonstrated by RAIL.

RAIL: A Runtime Risk-Aware Control and Learning Architecture. We introduce Risk-Aware Human-in-the-Loop with Adaptive Intrusion Response (**RAIL**), a runtime architecture that operationalizes explicit risk modeling and closes the detect–respond–learn loop within a single system. RAIL elevates risk to a first-class control variable through a probabilistic Intrusion Risk Index (IRI) that fuses three runtime cues, a curvature-consistency cue for actuation integrity, a time-to-collision cue for planning hazards, and an observation-shift cue for perception anomalies, via weighted Noisy-OR aggregation, converting anomaly detection into calibrated, source-aware control adaptation.

When risk is low, the nominal policy executes without interference. As risk increases, a lightweight contextual bandit selects a cue-specific shield, and the executor blends nominal and safeguarded actions with authority that grows monotonically with risk. This enables proactive, graded containment rather than binary takeover. Human override remains available, but intervention shifts from primary containment to rare exception handling. The learning loop integrates penalties for safety events with risk-aware shaping and prioritized replay, ensuring that takeovers and near misses provide structured feedback rather than ad hoc correction. RAIL tightly couples detection, mitigation, and learning to deliver interpretable, composable, and adversarially robust autonomy.

Figure 1 illustrates the overall runtime architecture. At each control step, the AV agent proposes an action a_t based on observations o_t. Before execution, this action is evaluated by a runtime safety layer that performs risk assessment and, when necessary, mitigates elevated risk by adjusting the action. The executed control u_t is then applied to the environment. The environment returns updated observations, while a human operator may provide situational cues, manual override, or corrective feedback. These signals are incorporated into

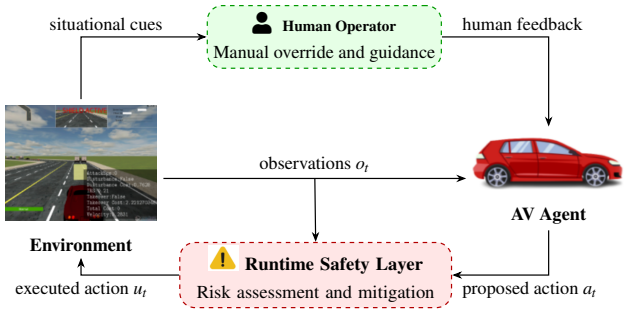


Figure 1: Risk-Aware Runtime Architecture for Human-in-the-Loop Autonomous Driving.

both the policy update and the runtime risk arbitration mechanism, forming a closed detect–assess–mitigate–learn loop that tightly couples anomaly detection, control adaptation, and experience-driven improvement.

Key Contributions. Our contributions are fourfold. First, we present **RAIL as a runtime risk-aware control and learning architecture** that closes the detect–respond–learn loop within a single system. Unlike prior HITL approaches that treat detection, mitigation, and learning as loosely coupled components [19, 23, 25, 26], RAIL tightly integrates probabilistic risk assessment, graded cue-specific shielding, human escalation, and experience replay at a control rate. This integration enables proactive containment of cyber-physical threats before human intervention becomes necessary.

Second, we introduce the **Intrusion Risk Index (IRI)**, an interpretable and multi-source risk quantification mechanism. IRI fuses complementary runtime cues, including curvature uncertainty, time-to-collision proximity, and LiDAR observation-shift, through a weighted Noisy-OR formulation. In contrast to implicit scalar safety costs, IRI decomposes risk by source, supports contextual shield selection, and provides an auditable trail at each control step.

Third, we develop a **safety-aware learning framework** that leverages structured operator feedback. By combining explicit penalties for violations and takeovers with risk-based shaping and prioritized replay, RAIL focuses learning on high-risk transitions while preserving nominal behavior, accelerating safety alignment in an off-policy actor–critic backbone.

Finally, we provide **comprehensive empirical validation** under both nominal and adversarial conditions. Across MetaDrive and CARLA, and under Controller Area Network (CAN) injection and LiDAR spoofing attacks, RAIL demonstrates improved safety, robustness, and reduced operator burden relative to RL, safe RL, imitation, offline RL, and prior HITL baselines, showing that explicit risk modeling yields calibrated control rather than binary fail-safe behavior.

2 Related Work

In this section, we position RAIL within prior work on learning-based control, runtime safety mechanisms, human-

in-the-loop autonomy, and automotive intrusion detection, highlighting architectural limitations that motivate explicit runtime risk modeling.

2.1 Learning-Based and HITL Autonomy

Learning-based control for autonomous driving is largely built on deep reinforcement learning methods that provide strong baselines for continuous decision making. Algorithms such as Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO) enable data-efficient optimization through entropy regularization and clipped surrogate objectives [16, 37]. Safe RL extends these methods with cost constraints, including constrained policy optimization (CPO) [2] and Lagrangian variants that tune safety multipliers [15, 40]. Although these approaches reduce violations relative to unconstrained RL, safety remains encoded as scalar penalties in the objective. Risk is not represented as an explicit runtime variable, and mitigation is learned indirectly rather than executed through calibrated control adaptation at the actuation rate.

Complementary lines of work seek to reduce unsafe exploration through imitation and offline learning. Behavioral cloning [6] and adversarial approaches such as Generative Adversarial Imitation Learning (GAIL) [18] bootstrap policies from expert demonstrations, while interactive variants including Human-Gated DAgger (HG-DAgger) [23] and intervention-weighted regression [26] incorporate corrective labels during execution. Offline reinforcement learning methods such as Conservative Q-Learning (CQL) [24] rely on static datasets to limit exploration risk. Although these techniques improve data efficiency and reduce training-time hazards, safety knowledge remains embedded in data distributions rather than operationalized as structured runtime risk. As a result, they provide neither decomposed risk attribution nor graded authority modulation under adversarial perturbations.

More recently, human-in-the-loop frameworks explicitly integrate operator feedback into policy optimization. Human–AI Copilot Optimization (HACO) [25] penalizes divergence from inferred human preferences, and Human-as-AI-Mentor DRL (HAIM-DRL) [19] incorporates mentoring costs during exploration. These methods reduce intervention frequency and achieve strong nominal performance. However, intervention remains fundamentally reactive: takeover events are treated as scalar penalties rather than structured, multi-source indicators of risk. The policy cannot communicate which subsystem triggered concern, nor can it calibrate mitigation strength according to risk severity. In adversarial settings, this limits preemptive containment and forensic traceability.

In contrast, RAIL introduces **explicit, decomposed runtime risk modeling** that directly governs shield selection and graded control blending. By elevating risk to a first-class control variable, RAIL bridges the gap among detection, mitigation, and learning under cyber-physical threats.

Table 1: Comparison of runtime safety filtering architectures.

Capability	**CBF** [5]	**HJ** [12]	**Simplex** [38]	**Shield** [4]	**RAIL (Ours)**
Model-based guarantee	✔	✔	✔	▲	▲
Multi-source cue fusion	✘	✘	✘	✘	✔
Online mitigation adaption	✘	✘	✘	✘	✔
Operator awareness	✘	✘	✘	✘	✔
Graded authority blending	✘	✘	✘	▲	✔
Risk decomposition	✘	✘	✘	▲	✔
Model-free compatibility	✘	✘	▲	▲	✔

✔ = supported, ▲ = partially supported, ✘ = not supported. CBF = Control Barrier Function; HJ = Hamilton–Jacobi reachability; Shield = reactive shield synthesis.

2.2 Risk-Sensitive Control and Runtime Safety

A substantial body of work studies risk-sensitive reinforcement learning as a means to reduce catastrophic outcomes. These approaches model the tails of return distributions rather than optimizing expected reward alone. Techniques based on coherent risk measures and distributional critics, such as Implicit Quantile Networks (IQN) and Quantile Regression Deep Q-Networks (QR-DQN), replace the standard objective with Conditional Value-at-Risk (CVaR) or entropic risk formulations [14, 43]. Subsequent advances combine model uncertainty with distributional value estimation to improve robustness in safety-critical robotics [34], and extend proximal policy gradients with safety or risk constraints [44]. While these methods attenuate high-loss outcomes during training, risk is defined over abstract reward distributions rather than grounded in explicit, multi-source runtime signals. They leave open how to operationalize heterogeneous cues such as perception anomalies or vehicle-to-everything (V2X) inconsistencies, and how to translate elevated concern into interpretable, graded control adaptation at actuation rate [21, 33].

Complementary lines of research enforce safety through runtime filtering rather than reward shaping. Control Barrier Functions (CBFs) guarantee forward invariance of a safe set by solving a quadratic program at each control step under control-affine dynamics assumptions [5, 10]. Hamilton–Jacobi reachability analysis computes backward reachable tubes to certify safety over finite horizons, accommodating bounded disturbances but requiring offline grid-based value computation [12]. The Simplex architecture [38] switches control to a verified baseline when a monitor is violated, and shield synthesis [4] constructs reactive strategies from temporal-logic specifications. These frameworks provide formal guarantees when accurate dynamics models and constraint specifications are available. However, they typically assume single-constraint formulations, provide binary intervention decisions, and do not adapt mitigation strategies online from operator feedback. Nor do they natively fuse heterogeneous cyber-physical cues into a calibrated risk estimate. RAIL is instead positioned as a **runtime risk orchestration**

layer that can incorporate formally verified controllers as shield arms while additionally providing graded authority blending, multi-source cue fusion, and structured feedback to learning, thereby extending coverage to model-free settings where formal approaches alone are insufficient.

In Table 1, *model-free compatibility* denotes operation without a hand-specified dynamics model or invariant set, *operator awareness* denotes incorporation of human-intervention signals into shield selection or learning updates, and *risk decomposition* denotes attribution of overall risk to an identifiable subsystem source. Existing filtering approaches rely on model-based guarantees with fixed constraints or provide binary containment, and none natively fuse heterogeneous cyber–physical cues, adapt mitigation from operator feedback, or support graded authority blending. RAIL addresses these limitations while remaining compatible with formal controllers that serve as shield components.

2.3 Autonomous Vehicle Intrusion Defense

Automotive cybersecurity research has long established that in-vehicle networks and electronic control units (ECUs) are vulnerable to remote compromise, motivating systematic intrusion detection mechanisms for connected and autonomous vehicles [8, 27]. Subsequent work introduced specification-based monitoring and learning-driven anomaly detection on the Controller Area Network (CAN) bus, leveraging signal timing, message frequency, and fingerprinting techniques to identify malicious deviations [9, 11, 28]. More recent approaches incorporate reinforcement learning and distributional modeling to characterize deviations under uncertainty [22, 42]. In parallel, runtime monitors extend beyond bus-level anomalies to cross-check vehicle dynamics, actuator consistency, and state-estimation residuals, detecting control-relevant inconsistencies during operation [41].

Sensor-layer vulnerabilities further expand the attack surface. LiDAR spoofing, adversarial camera perturbations, and GPS manipulation have been demonstrated in both laboratory and field settings, showing that perception pipelines can be deceived without compromising internal software [3, 7, 35, 39]. Collectively, these studies underscore that autonomous vehicle security spans heterogeneous attack channels across perception, planning, and actuation.

More recent efforts move beyond detection toward runtime response. REACT [17] formulates intrusion response as an optimization problem, selecting countermeasures via linear programming under cost–benefit trade-offs. AVMON [1] combines invariant learning with residual prediction to enable anomaly detection and partial recovery during execution. Despite this progress, detection and control remain architecturally separated in most systems: anomaly scores are generated upstream, while mitigation actions are pre-defined or weakly integrated with human supervisory intent. Human intervention typically functions as an external override rather than as a structured component of containment.

Within the HITL autonomy literature, risk is commonly encoded implicitly as scalar penalties. In HACO [25], the human preference signal is modeled through a proxy advantage $\hat{A}^h$ that penalizes divergence from an inferred value function; the resulting cost is aggregated without attribution to specific risk sources. Similarly, HAIM-DRL [19] integrates takeover events as scalar negative rewards that shape exploration but do not expose the subsystem responsible for intervention. In both cases, human guidance influences learning, yet the representation of risk remains undifferentiated and opaque.

These limitations motivate a more tightly integrated formulation in which detection, containment, and learning are co-designed rather than sequentially layered.

RAIL models **risk as an explicit, decomposable control variable** instead of an implicit scalar penalty. The Intrusion Risk Index (IRI) aggregates cue-level contributions and identifies the dominant source via the inline selection rule $i^\star = \arg\max_i w_i r_i$, where each r_i corresponds to a structured cue aligned with a specific kill-chain layer such as perception, planning, or actuation. A Noisy-OR aggregation provides a probabilistic interpretation as the likelihood that at least one cue signals danger. This structure enables targeted shield selection rather than uniform penalty injection, allowing mitigation to be context-sensitive and subsystem-aware.

This design establishes **detection–mitigation coupling within a unified control loop**. The Intrusion Risk Index (IRI) serves as both detection output and control input, eliminating the interface gap between anomaly assessment and policy execution. A contextual bandit selects mitigation actions conditioned on the cue vector and operator feedback, enabling the response policy to evolve with operational experience. Explicit cue attribution supports auditability through logged risk decomposition, composability by adding new cue terms and corresponding shields without redesigning the architecture, and regulatory traceability through interpretable cause-effect mappings. In contrast to scalar-cost human-in-the-loop approaches, RAIL operationalizes risk as a structured signal that directly informs detection, containment, and learning.

3 Proposed Approach: RAIL

System overview. Figure 2 shows the RAIL pipeline, which turns runtime risk into graded, interpretable control. At each control step, the policy π_θ proposes a nominal action a_t from state s_t. Three cues covering actuation (r_{CURV}), planning (r_{TTC}), and perception (r_{OOD}) are fused via weighted Noisy-OR into $\mathrm{IRI}_t \in [0,1]$. If $\mathrm{IRI}_t \le \tau$, a_t executes unmodified; otherwise a contextual bandit selects a cue-specific shield $\mathcal{S}_{k_t}$ producing a safeguarded action $\tilde{a}_t$, and the executor outputs $u_t = (1-\alpha_t)a_t + \alpha_t \tilde{a}_t$ with α_t monotone in IRI. A human takeover, when issued, overrides u_t. Each annotated transition is stored in a prioritized replay buffer that drives off-policy SAC updates.

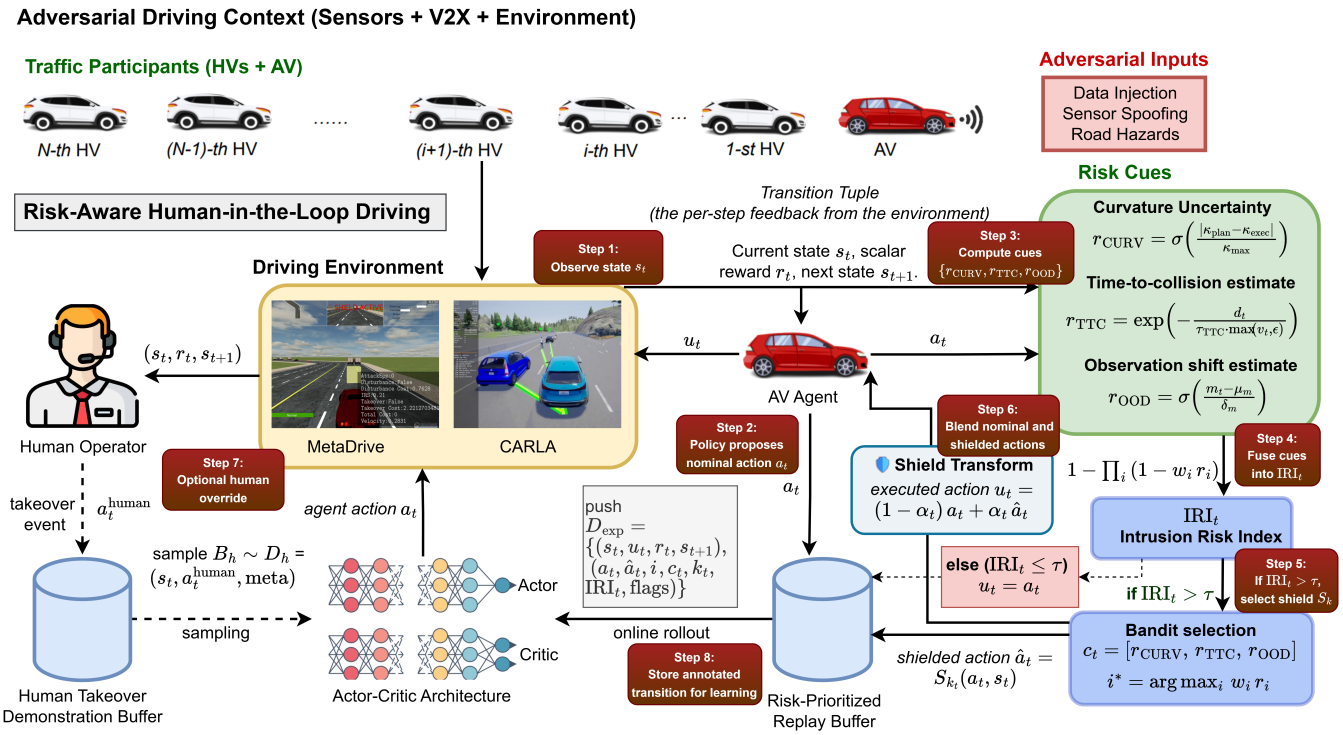


Figure 2: **RAIL architecture.** Steps 1–6 illustrate the runtime control path. The AV agent observes s_t, proposes a nominal action a_t, computes three risk cues $(r_{\text{CURV}}, r_{\text{TTC}}, r_{\text{OOD}})$, and fuses them via Noisy-OR into the Intrusion Risk Index IRI_t. When $\text{IRI}_t \le \tau$, the system follows the low-risk pass-through path and executes $u_t = a_t$. When $\text{IRI}_t > \tau$, a contextual bandit selects a cue-specific shield S_{k_t} to produce $\hat{a}_t = S_{k_t}(a_t, s_t)$, and the executor applies the blended control $u_t = (1-\alpha_t)a_t + \alpha_t \hat{a}_t$. Step 7 shows optional human override, and Step 8 shows that each annotated transition is logged to a risk-prioritized replay buffer for off-policy SAC updates.

Threat model (summary). RAIL targets two adversary classes representative of documented automotive attacks: actuation-channel injection modeling CAN-level ECU compromise [8, 27], and LiDAR spoofing modeling optical relay attacks [7, 39]. We assume black-box adversaries that operate in bounded bursts. The three IRI cues are chosen to cover the actuation, planning, and perception layers exercised by these attacks; full attack parameterizations and a discussion of stronger adversaries appear in Section 4.2.

3.1 Risk-Aware Control Architecture

We consider an ego vehicle operating in continuous state and action spaces. At time t, the vehicle state s_t captures ego kinematics, local map features, and perception outputs, while a stochastic policy $\pi_\theta(a_t \mid s_t)$ generates a continuous control action a_t consisting of steering and longitudinal commands. The environment evolves under the applied control u_t according to the transition model $p(s_{t+1} \mid s_t, u_t, \xi_t)$, where ξ_t represents exogenous disturbances.

RAIL augments this closed-loop interaction with an Intrusion Risk Index $\text{IRI}_t \in [0,1]$ computed from heterogeneous runtime cues spanning cyber and physical layers. When risk remains low, the nominal policy action is executed directly ($u_t = a_t$). As risk increases, a cue-specific shield produces a safeguarded proposal $\tilde{a}_t$, and an execution module blends $\tilde{a}_t$ with a_t according to risk-dependent authority. A human override mechanism remains available but transitions from primary containment to exception handling as automated mitigation becomes more reliable.

Shield selection and gain adaptation are governed by a lightweight contextual bandit conditioned on the current cue vector and delayed performance signals. This design enables mitigation policies to evolve online while preserving computational efficiency. Learning proceeds off-policy using the Soft Actor-Critic (SAC) algorithm [16]. The replay buffer stores the primary transition tuple (s_t, u_t, r_t, s_{t+1}), augmented with structured annotations $(a_t, \tilde{a}_t, \text{IRI}_t, c_t, i^\star, k_t, \text{flags})$. High-risk and human-corrected transitions are prioritized, ensuring that parameter updates concentrate on safety-critical operating regimes rather than nominal behavior. Table 2 summarizes the principal symbols used throughout the paper.

Table 2: Summary of symbols and definitions used in RAIL.

Symbol	Description
s_t	Ego state (kinematics, map, perception)
a_t	Nominal policy action [steer, accel]
u_t	Executed (blended) action
$\tilde{a}_t$	Shield output $\mathcal{S}_{k_t}(a_t, s_t)$
π_θ	Stochastic policy parameterized by θ
ξ_t	Environmental disturbances (including attacks)
IRI_t	Intrusion Risk Index $\in [0,1]$
r_i, w_i	Cue i risk value and fixed weight
$i^\star$	Dominant cue $\arg\max_i w_i r_i$
α_t	Shield authority $\in [0,1]$
c_t	Cue context vector $[r_{\mathrm{CURV}}, r_{\mathrm{TTC}}, r_{\mathrm{OOD}}]$
k_t	Selected shield arm
τ	IRI activation threshold
H_t	Human takeover flag $\in \{0,1\}$
R_t	Shaped reward signal

3.2 Intrusion Risk Index (IRI)

We define the Intrusion Risk Index (IRI) as a unified, interpretable risk metric that fuses heterogeneous cyber–physical cues into a single scalar value for real-time decision support.

Noisy-OR fusion. [32] The IRI aggregates multiple cyber-physical risk cues into a single interpretable value. At each time step t:

$$\mathrm{IRI}_t = 1 - \prod_{i=1}^{M} \big(1 - w_i\, r_i(s_t, a_t)\big), \tag{1}$$

where M is the number of monitored cues, $r_i(s_t,a_t) \in [0,1]$ is the normalized risk contribution of cue i, and $w_i \in [0,1]$ are fixed importance weights with $\sum_i w_i = 1$. The dominant cue $i^\star = \arg\max_i w_i r_i(s_t,a_t)$ is recorded for interpretability. The multiplicative Noisy-OR models the probability that at least one cue indicates danger: a single high-confidence anomaly is sufficient to elevate IRI_t, while multiple weak but consistent signals compound non-linearly.

Three cues suffice for the present study because they span the actuation, planning, and perception layers of the AV kill chain and are computable at control rate from signals already available in our simulator stack. The IRI is modular: additional cues such as GPS/GNSS spoofing indicators, radar–camera disagreement, and V2X consistency terms can be integrated by defining a normalized $r_i \in [0,1]$ and a matching shield arm, without retraining the base policy.

These cues are selected to satisfy four design principles [13, 31]: (i) **Observability**: each cue is computable from signals available at control rate without requiring additional sensing hardware. (ii) **Normalizability**: each raw measurement admits a monotone mapping to $[0,1]$, ensuring that Noisy-OR fusion remains well-defined.(iii) **Complementarity**: under common driving scenarios, the cues activate on distinct failure modes to maximize the information captured by the IRI, as verified empirically in Section 5.5. (iv) **Kill-chain coverage**: the cue set spans actuation, planning, and perception layers.

Table 3: Cue-to-threat mapping across the AV kill chain.

Cue	Kill-chain	Attacks detected	Signal
r_{CURV}	Actuation	CAN injection, actuator fault	Yaw vs. plan
r_{TTC}	Planning	Phantom obstacle, brake disable	Range + vel.
r_{OOD}	Perception	LiDAR spoof / mask / blind	Beam stats

Curvature uncertainty cue. This cue measures control integrity by comparing planned and executed curvature:

$$r_{\mathrm{CURV}}(s_t,a_t) = \sigma\Big(\tfrac{|\kappa_{\mathrm{plan}}(s_t) - \kappa_{\mathrm{exec}}(s_t,a_t)|}{\kappa_{\max}}\Big), \tag{2}$$

where κ_{plan} is the planned path curvature from route geometry, κ_{exec} is the realized curvature from yaw rate over speed with a numerical floor, $\kappa_{\max}$ normalizes by feasible limits, and σ is a logistic squashing. Risk grows as the executed path diverges from the intended trajectory, flagging actuation faults or injected steering.

Time-to-collision cue. This cue converts available reaction time into risk:

$$r_{\mathrm{TTC}}(s_t,a_t) = \exp\Big(-\tfrac{d_t}{\tau_{\mathrm{TTC}}\cdot\max(v_t,\varepsilon)}\Big), \tag{3}$$

where d_t is the path-aligned distance to the most threatening object, v_t is the closing speed, $\varepsilon > 0$ avoids division by zero, and τ_{TTC} sets the time scale. The score approaches one when contact is imminent and decays as available time increases.

Observation shift cue. This cue flags sensor-level distribution shift in LiDAR returns:

$$r_{\mathrm{OOD}}(s_t) = \sigma\left(\frac{\sqrt{(z_t-\mu)^\top(\Sigma+\varepsilon I)^{-1}(z_t-\mu)} - \mu_m}{\delta_m}\right), \tag{4}$$

where $z_t \in \mathbb{R}^{72}$ is the LiDAR range vector, (μ,Σ) are the clean-data mean and covariance with Tikhonov regularization εI, and (μ_m,δ_m) center and scale the Mahalanobis distance. Risk increases as the current observation departs from the nominal sensor model, detecting spoofing, masking, or blinding.

3.3 Risk-Conditioned Shielding and Action Blending

This section describes how risk modulates control authority and how cue-dependent shields are selected and adapted.

Let a_t denote the nominal policy action and $\tilde{a}_t = \mathcal{S}_{k_t}(a_t,s_t)$ the output of shield k_t selected at time t. The executor computes the *executed control input* u_t, which is applied to the vehicle actuators, by blending the nominal and shielded actions using an authority coefficient $\alpha_t \in [0,1]$:

$$u_t = (1-\alpha_t)\,a_t + \alpha_t\,\tilde{a}_t, \tag{5}$$

where α_t is a monotonically non-decreasing function of IRI_t, smoothed via an exponential moving average and rate-limited across control cycles to prevent abrupt authority shifts.

This blending rule satisfies the properties below, consistent with Table 1.

1. **Passivity.** If $\mathrm{IRI}_t < \tau$, then $\alpha_t = 0$ and $u_t = a_t$, meaning the safety layer remains inactive and the nominal policy operates without modification.
2. **Monotonicity.** The authority coefficient α_t is monotonically non-decreasing in IRI_t, ensuring that increased risk cannot reduce the level of shielding applied.
3. **Bounded deviation.** Since $\|u_t - a_t\| = \alpha_t \|\tilde{a}_t - a_t\| \le \|\tilde{a}_t - a_t\|$, the executed action deviates from the nominal action by at most the shield correction magnitude, ensuring that blending never amplifies the intervention.

Shield selection is formulated as a contextual bandit over a small library of interpretable transformations. The context vector is $c_t = [r_{\mathrm{CURV}}, r_{\mathrm{TTC}}, r_{\mathrm{OOD}}]$. Each arm k is assigned a linear score $z_{k,t} = \theta_k^\top c_t$, yielding the selection probability

$$p_t(k) = \frac{\exp(\theta_k^\top c_t)}{\sum_j \exp(\theta_j^\top c_t)}. \tag{6}$$

After executing control input u_t (see Eq. (5)), delayed scalar feedback is computed as an indicator of success (one if no takeover occurs within horizon Δ) minus a small effort penalty proportional to $\|\tilde{a}_t - a_t\|$. The parameter vector θ_{k_t} is then updated in the direction of c_t scaled by this feedback, while other arms remain unchanged. This design preserves locality, since only the contextually relevant shield is adapted; auditability, since the active shield, α_t, and scores $\{z_{k,t}\}$ are logged at each step; and convergence, as temperature annealing drives concentration on the empirically best shield for each cue configuration.

3.4 Risk-Aware Reward Design

This subsection describes the reward formulation used to train RAIL, which integrates discrete safety penalties with continuous risk-aware shaping.

RAIL's training reward combines explicit penalties for safety-critical events with continuous shaping terms that discourage elevated risk before failures occur:

$$\begin{aligned} R_t = r_{\mathrm{task}} &- \lambda_1 \mathbb{1}[\text{coll.}] - \lambda_2 \mathbb{1}[\text{off-road}] - \lambda_3 H_t \\ &- \lambda_4 \mathrm{IRI}_t - \lambda_5 \|\tilde{a}_t - a_t\| \mathbb{1}[\mathrm{IRI}_t > \tau], \end{aligned} \tag{7}$$

where r_{task} rewards driving progress, lane centering, and speed tracking. The explicit component penalizes collisions (λ_1), off-road departures (λ_2), and human takeovers (λ_3). The shaping term penalizes elevated intrusion risk via $\lambda_4 \mathrm{IRI}_t$, encouraging low-risk behavior even without discrete failures. An additional penalty on shield effort, $\lambda_5 \|\tilde{a}_t - a_t\|$ when $\mathrm{IRI}_t > \tau$, discourages excessive reliance on shielding. We set $\lambda_1 = 1.0$, $\lambda_2 = 1.0$, $\lambda_3 = 0.5$, $\lambda_4 = 0.1$, and $\lambda_5 = 0.05$.

3.5 Risk-Prioritized Replay and Training

Unlike standard prioritized replay, which scales the sampling probability by the TD error alone, *risk-prioritized replay* additionally weights transitions by the runtime IRI and by whether a human takeover occurred, so that adaptation concentrates on cyber–physical hazards rather than generic high-error states. This subsection describes how RAIL integrates this mechanism with off-policy SAC learning.

Each transition stores the core tuple (s_t, u_t, R_t, s_{t+1}) together with structured annotations $(a_t, \tilde{a}_t, \mathrm{IRI}_t, c_t, i^\star, k_t, H_t)$. The sampling priority is defined as

$$p_t = |\delta_t|^\eta + \beta \mathrm{IRI}_t + \gamma H_t + \varepsilon_p, \tag{8}$$

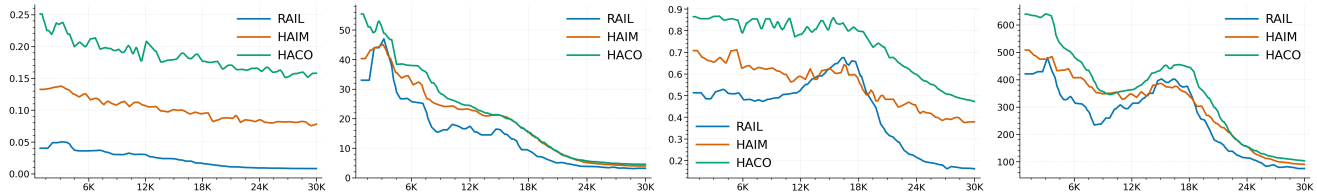
where δ_t is the temporal-difference (TD) error, η controls TD scaling, β weights the contribution of risk magnitude, γ emphasizes human takeovers, and $\varepsilon_p > 0$ prevents sample starvation. We use $\eta = 0.6$, $\beta = 0.3$, $\gamma = 1.0$, and $\varepsilon_p = 10^{-6}$. These choices follow standard prioritized-replay practice: $\eta < 1$ preserves partial uniform coverage, β is set so that risk-based priority contributes on the same order as typical early-training TD magnitudes, $\gamma \ge 1$ guarantees that takeover transitions dominate the priority budget, and ε_p prevents sample starvation. Moderate perturbations of these values did not materially change training dynamics. Transitions are sampled proportionally to p_t, and importance-sampling weights are applied to debias gradient updates.

Algorithm 1 summarizes the integrated runtime and training loop. Training alternates short online rollouts that refresh contextual bandit statistics and collect new experience with off-policy Soft Actor-Critic (SAC) updates from the prioritized buffer. Rollouts run with risk-aware blending and contextual shield selection enabled. Low-risk transitions are retained with small priority to preserve nominal coverage. When $\mathrm{IRI}_t > \tau$, the log records k_t and α_t for bandit credit assignment. Human takeover transitions receive the highest priority, followed by high-risk near-miss events, accelerating safety alignment and reducing future interventions.

4 Experimental Setup

4.1 Evaluation Environments

We evaluate in METADRIVE on procedurally generated road graphs, including straight segments, curves, T/X intersections, ramps, roundabouts, and forks, populated by heterogeneous stochastic traffic. Background vehicles follow Intelligent Driver Model (IDM) longitudinal dynamics with MOBIL lane changing, while static cones, barriers, and occasional slow vehicles induce long-tail interactions. Episodes terminate upon reaching the goal, collision, off-road departure, or timeout (1,000 steps). For cross-simulator generalization, we additionally test in CARLA using a top-down semantic raster



(a) Disturbance rate (b) Disturbance count (c) Takeover rate (d) Takeover count

Figure 3: Training curves on MetaDrive comparing RAIL, HAIM [19], and HACO [25]. RAIL reduces disturbance and operator takeovers fastest and reaches the lowest steady state across all four metrics.

Algorithm 1: RAIL Runtime and Learning Loop

Init: actor π_θ, critics $Q_{\psi_{1,2}}$, targets, replay $\mathcal{D}$; cue weights $\{w_i\}$; bandit params $\{\theta_k\}$; threshold τ; horizon Δ
for each environment step t **do**
  Sample $a_t \sim \pi_\theta(\cdot \mid s_t)$
  Compute $c_t = [r_{\mathrm{CURV}}, r_{\mathrm{TTC}}, r_{\mathrm{OOD}}]$; $\mathrm{IRI}_t \leftarrow 1 - \prod_i (1 - w_i r_i)$
  if $\mathrm{IRI}_t > \tau$ **then** ▷ Shield activation
    $k_t \sim \mathrm{Softmax}(\{\theta_k^\top c_t\})$; $\tilde{a}_t \leftarrow \mathcal{S}_{k_t}(s_t, a_t)$
    $u_t \leftarrow (1-\alpha_t)\, a_t + \alpha_t\, \tilde{a}_t$ ▷ Eq. (5)
  else $u_t \leftarrow a_t$
  end if
  if human takeover **then** $u_t \leftarrow a_t^{\mathrm{human}}$; $H_t = 1$
  else $H_t = 0$
  end if
  Execute u_t; observe s_{t+1}; compute R_t ▷ Eq. (7)
  Push to $\mathcal{D}$ with priority p_t ▷ Eq. (8)
  Sample minibatch from $\mathcal{D}$; update $Q_{\psi_{1,2}}$, π_θ, temperature
  if $\mathrm{IRI}_t > \tau$ and no takeover in next Δ steps **then**
    $\theta_{k_t} \leftarrow \theta_{k_t} + \eta_b\, y_t\, c_t$ ▷ Bandit update
  end if
end for

input with identical control heads, confirming that RAIL's performance gains transfer across observation modalities.

4.2 Threat Model

We evaluate RAIL under representative cyber and physical attack scenarios targeting both control and perception channels. In both cases, the adversary operates under a **black-box** threat model: it has no knowledge of RAIL's internal policy parameters, IRI state, or shield selection, and cannot observe or adapt to the defense in real time. The attacker controls injection timing and magnitude but not the victim's software stack. Attacks are **non-persistent**, occurring in fixed-duration bursts with inter-burst recovery periods, consistent with transient exploitation scenarios documented in the automotive security literature [8, 27].

Stronger adversaries lie outside the current scope. A white-box attacker with knowledge of IRI weights, τ, and shield parameters could in principle craft low-magnitude perturbations that stay below per-cue activation bounds, or exploit shield-selection dynamics by alternating across cue sources. A persistent adversary could maintain steady-state bias across recovery windows, eroding the assumption that inter-burst intervals restore nominal behavior. The present evaluation therefore characterizes RAIL under a widely reported exploitation regime consistent with documented automotive attacks, rather than the worst case. Adaptive thresholding, online cue-weight adaptation, and adversarial training against white-box and persistent attackers are identified as future work in Section 7.

Cyber attack: Action-channel injection. An adversary injects bounded perturbations into steering or longitudinal acceleration after the policy but before the safety layer, modeled as $u_t = \mathrm{sat}(a_t + \eta_t)$. Piecewise-constant biases of up to 50% full scale affect either control channel for 5 s every 30 s, with random start time, sign, and channel selection. This emulates electronic control unit (ECU) or Controller Area Network (CAN) faults that introduce unintended steering or throttle/brake bias. The 50% magnitude reflects the upper range of demonstrated CAN frame injection amplitudes [27], while the 5 s burst and 30 s interval model intermittent exploit windows observed in ECU fuzzing campaigns [8]. These injections are not inherently stealthy: large biases produce observable trajectory deviations that RAIL's curvature cue can detect, though smaller or ramping injections could evade magnitude-based detection and are left to future work.

Physical attack: LiDAR spoofing. A phantom obstacle is created by overwriting a contiguous azimuth sector in the 72-beam LiDAR with short-range returns corresponding to a rigid object 4 m ahead, with ± 0.5 m lateral jitter. Attack bursts last 5 s and recur every 30 s. This setup follows prior demonstrations of relay-based phantom injection and optical-channel LiDAR exploits, where spoofed returns at 3–5 m range were sufficient to trigger emergency braking in commercial systems [7, 39]. The spoofed sector is detectable by RAIL's observation-shift cue as an out-of-distribution perturbation; however, spatially distributed or gradually intensifying spoofs that better approximate natural obstacles may reduce detectability and represent an important direction for future evaluation. An attack is considered successful if a collision or off-road event occurs within 3 s of a burst.

4.3 Baseline Methods

We benchmark RAIL against 12 representative baselines grouped into five categories.

(1) RL with reward shaping. These methods add dense safety penalties to the reward but do not enforce constraints.

1. **Soft Actor-Critic with Reward Shaping (SAC-RS)** [16]. SAC with entropy regularization and dense penalties on collisions, off-road events, and excessive actuation.
2. **Proximal Policy Optimization with Reward Shaping (PPO-RS)** [37]. PPO with a clipped surrogate objective and identical shaping terms to stabilize safety updates.

(2) Safe RL with explicit cost constraints. These approaches separate reward and safety cost signals and enforce constraints during optimization.

3. **Soft Actor-Critic with Lagrangian Constraints (SAC-Lag)** [15]. SAC with a learned dual variable enforcing episodic safety budgets via Lagrangian relaxation.
4. **Proximal Policy Optimization with Lagrangian Constraints (PPO-Lag)** [40]. PPO with responsive dual updates to achieve faster constraint satisfaction.
5. **Constrained Policy Optimization (CPO)** [2]. Trust-region updates projected onto a linearized cost-feasible set using a safety critic.

(3) Offline RL and imitation learning. These methods learn from fixed datasets or expert demonstrations rather than fully online exploration.

6. **Conservative Q-Learning (CQL)** [24]. Offline RL with conservative Q-regularization to penalize out-of-distribution actions.
7. **Behavior Cloning (BC)** [6]. Supervised regression on expert state-action pairs, prone to compounding error.
8. **Generative Adversarial Imitation Learning (GAIL)** [18]. Adversarial occupancy matching to replicate expert visitation distributions.

(4) Conventional human-in-the-loop methods. These approaches incorporate human corrections through data aggregation or weighted regression.

9. **Human-Gated Dataset Aggregation (HG-DAgger)** [23]. Selective human corrections aggregated into the training buffer to address distributional shift.
10. **Intervention-Weighted Regression (IWR)** [26]. Higher regression weights assigned to human takeover transitions.

(5) Modern human-in-the-loop methods. These methods integrate intervention signals into reinforcement learning.

11. **Human-AI Copilot Optimization (HACO)** [25]. Proxy advantage penalizing divergence between agent and human actions during online correction.
12. **Human-AI Mentor Deep Reinforcement Learning (HAIM-DRL)** [19]. Binary takeover-based mentoring cost shaping updates to reduce intervention frequency.

All baselines use the same observation encoder, action bounds, control rate (10 Hz), random seeds, and training budget as RAIL, with hyperparameters from the original works.

4.4 Metrics

We evaluate performance, safety, efficiency, and robustness using episode-aggregated metrics. Let N denote the number of evaluation episodes, T_i the length of episode i, and $\mathcal{A} \subseteq \{1,\dots,N\}$ the subset subjected to adversarial attack.

(1) Test Success Rate (TSR).

$$\mathrm{TSR} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}[\mathrm{goal}_i = 1 \wedge \mathrm{coll}_i = 0 \wedge \mathrm{offroad}_i = 0]\,. \quad (9)$$

TSR measures the fraction of episodes that reach the goal without safety violations; higher values indicate reliable task completion.

(2) Test Return (TR).

$$\mathrm{TR} = \frac{1}{N}\sum_{i=1}^{N} G_i, \quad G_i = \sum_{t=0}^{T_i} r_t^{(i)}, \quad (10)$$

where $r_t^{(i)}$ follows (7). TR reflects cumulative task performance and driving quality.

(3) Test Safety Violations (TSV).

$$\mathrm{TSV} = \frac{1}{N}\sum_{i=1}^{N}\left(n_{\mathrm{coll}}^{(i)} + n_{\mathrm{offroad}}^{(i)}\right). \quad (11)$$

TSV averages collision and off-road counts per episode; lower values indicate safer behavior.

(4) Disturbance Rate (DR).

$$\mathrm{DR} = \frac{1}{\sum_{i=1}^{N} T_i}\sum_{i=1}^{N}\sum_{t=0}^{T_i} \mathbb{1}\left[|a_t^{\mathrm{acc},(i)}| > a_{\max}\right]. \quad (12)$$

DR quantifies excessive longitudinal acceleration, capturing ride comfort and traffic smoothness.

(5) Training Safety Violations ($\mathrm{TSV}_{\mathrm{train}}$).

$$\mathrm{TSV}_{\mathrm{train}} = \sum_{k=1}^{K}\left(n_{\mathrm{coll}}^{(k)} + n_{\mathrm{offroad}}^{(k)}\right). \quad (13)$$

This aggregates all safety events during training, reflecting exploration cost.

(6) Training Data Usage (TDU).

$$\mathrm{TDU} = \sum_{k=1}^{K} T_k. \quad (14)$$

TDU measures total interaction steps and thus sample efficiency.

(7) Disengagement Rate under Attack (DRA).

$$\mathrm{DRA} = \frac{1}{|\mathcal{A}|} \sum_{i \in \mathcal{A}} \mathbb{1}[\mathrm{takeover}_i = 1]. \tag{15}$$

DRA captures reliance on human takeover under adversarial conditions; lower is better.

(8) Attack Success Rate (ASR).

$$\mathrm{ASR} = \frac{1}{|\mathcal{A}|} \sum_{i \in \mathcal{A}} \mathbb{1}[\mathrm{coll}_i = 1 \vee \mathrm{offroad}_i = 1]. \tag{16}$$

ASR measures the fraction of attacked episodes resulting in safety failure and serves as the primary robustness metric.

4.5 Training and Implementation Details

All methods use Soft Actor-Critic (SAC) with shared hyperparameters selected via grid search: actor, critic, and entropy learning rates of 1×10^{-4}, batch size 1024, horizon 1,000, learning starts after 100 steps, control at 10 Hz, action clipping, and first-order rate limiting. Observations include ego kinematics, a 72-beam LiDAR with 60 m range, side detectors, lane features, and RAIL diagnostics. Actions $a_t = [a_t^{\mathrm{steer}}, a_t^{\mathrm{acc}}] \in [-1,1]^2$ map to ± 0.35 rad steering and $[-4.0, 2.5]\,\mathrm{m\,s^{-2}}$ acceleration. The Intrusion Risk Index uses weights $(0.3, 0.4, 0.3)$ with threshold $\tau = 0.3$ (sensitivity analysis in Section 5.5). The reward follows Eq. (7). All methods and results use five random seeds; the best checkpoint on the validation split is retained for evaluation. All experiments run on a single NVIDIA RTX 3090 GPU (24 GB VRAM); RAIL trains in approximately 2.5 hours for 30K steps.

Scripted human surrogate. Large-scale evaluation with real operators is infeasible, so we use a deterministic scripted surrogate in place of a human co-pilot, consistent with prior HITL evaluations [19, 25]. The surrogate monitors the same cue vector c_t as the policy and issues a takeover when IRI_t exceeds a higher threshold $\tau_H = 0.7$, well above the shield activation threshold $\tau = 0.3$, or when a collision or off-road cue persists for more than five consecutive control steps while shielding is active. On activation, it applies an IDM-style braking command with lane-centering steering until the triggering condition clears. The surrogate therefore acts as a safety backstop that fires only after proactive shielding has already engaged, not as a primary controller. Section 5.4 isolates the respective contributions of the shield and the surrogate.

5 Experimental Results and Analysis

This section presents a systematic evaluation of RAIL across nominal, adversarial, and ablation settings. We organize the empirical study around five guiding research questions that examine performance, robustness, component contributions, sensitivity, and practical overhead.

We structure the analysis around the following questions:

Q1: *How does RAIL perform relative to reinforcement learning, safe reinforcement learning, offline reinforcement learning, imitation learning, and human-in-the-loop baselines in terms of safety, task performance, and traffic smoothness?*

Q2: *To what extent does RAIL maintain robustness under representative cyber-physical intrusion scenarios?*

Q3: *Which architectural components of RAIL contribute most significantly to overall performance and safety gains?*

Q4: *How sensitive is RAIL to key design parameters, including the Intrusion Risk Index threshold, cue weights, and cue interactions?*

Q5: *What are the runtime overhead and human intervention requirements associated with RAIL?*

5.1 Comparative Performance

We first address Q1 by comparing RAIL against reinforcement learning, safe reinforcement learning, offline learning, imitation learning, and human-in-the-loop baselines under nominal conditions.

Reward-shaped RL vs. Safe RL. Table 4 shows that reward-shaped RL attains high return but weak safety. SAC-RS achieves 384.56 return but incurs 0.87 safety violations and 0.015 disturbance, while PPO-RS drops to 305.41 return with 4.12 violations. Safe RL reduces violations at the cost of performance: SAC-Lag lowers violations to 0.78 but achieves only 0.71 success rate, and CPO collapses to 0.21 success. These results suggest that implicit cost signals, whether reward penalties or Lagrangian multipliers, lack failure-mode decomposition, limiting targeted mitigation.

Offline and Imitation Learning. CQL, BC, and GAIL underperform across all metrics (SR ≤ 0.13), indicating that fixed datasets and supervised objectives fail to capture the diversity of procedurally generated MetaDrive scenarios.

Human-in-the-Loop Methods vs. RAIL. Figure 3 shows that RAIL reduces disturbance and takeovers more rapidly than HACO and HAIM-DRL during training. At convergence (Table 4), RAIL achieves 360.65 return, matches the top success rate (0.85), and records the lowest test safety violations (0.75) and training violations (29.07) using only 30K samples. Compared to HAIM-DRL, RAIL improves return by 1.8% while maintaining comparable disturbance and success. RAIL is the only method that simultaneously matches expert-level success, minimizes training safety violations, and preserves sample efficiency. Similar trends appear in CARLA (Table 5), where RAIL achieves 1609.70 return and 0.41 success rate with 8K steps, confirming cross-simulator generalization.

5.2 Robustness to Cyber-Physical Intrusions

We next address Q2 by evaluating RAIL under actuation-channel injection and perception spoofing attacks.

Table 4: Performance comparison on MetaDrive. Values are mean ± std over five seeds; best results in **bold**.

Category	Method	Total Training Safety Violation	Training Data Usage	Test Return	Test Safety Violation	Disturbance Rate	Test Success Rate
Expert	Human	–	–	$388.16_{\pm 45.00}$	$0.03_{\pm 0.00}$	0	1.00
RL	SAC-RS [16]	$2.78K_{\pm 0.97K}$	1M	$\mathbf{384.56}_{\pm \mathbf{37.5}}$	$0.87_{\pm 1.47}$	$0.015_{\pm 0.005}$	$0.83_{\pm 0.32}$
	PPO-RS [37]	$27.51K_{\pm 3.86K}$	1M	$305.41_{\pm 14.23}$	$4.12_{\pm 1.24}$	$0.021_{\pm 0.014}$	$0.67_{\pm 0.12}$
Safe RL	SAC-Lag [15]	$1.98K_{\pm 0.75K}$	1M	$352.46_{\pm 108.78}$	$0.78_{\pm 0.58}$	$0.019_{\pm 0.007}$	$0.71_{\pm 0.79}$
	PPO-Lag [40]	$15.46K_{\pm 5.13K}$	1M	$298.98_{\pm 50.99}$	$3.28_{\pm 0.38}$	$0.025_{\pm 0.016}$	$0.52_{\pm 0.27}$
	CPO [2]	$4.36K_{\pm 2.22K}$	1M	$194.06_{\pm 108.86}$	$1.71_{\pm 1.02}$	–	$0.21_{\pm 0.29}$
Offline RL	CQL [24]	–	49K	$116.45_{\pm 34.94}$	$3.68_{\pm 7.61}$	$0.007_{\pm 0.006}$	$0.13_{\pm 0.09}$
IL	BC [6]	–	49K	$36.13_{\pm 10.06}$	$1.05_{\pm 0.54}$	$0.012_{\pm 0.017}$	$0.01_{\pm 0.02}$
	GAIL [18]	$3.68K_{\pm 3.17K}$	49K	$108.36_{\pm 16.08}$	$4.18_{\pm 1.25}$	$0.001_{\pm 0.0009}$	$0.03_{\pm 0.01}$
Conventional HITL	HG-Dagger [23]	35.58	50K	106.21	2.63	0.108	0.04
	IWR [26]	69.74	50K	298.87	3.61	0.122	0.61
Human-AI Copilot	HACO [25]	$30.05_{\pm 10.89}$	30K	$350.01_{\pm 9.72}$	$0.78_{\pm 0.85}$	$0.038_{\pm 0.0083}$	$0.83_{\pm 0.07}$
Human as AI Mentor	HAIM-DRL [19]	$29.84_{\pm 10.25}$	30K	$354.34_{\pm 11.08}$	$0.76_{\pm 0.28}$	$\mathbf{0.0023}_{\pm \mathbf{0.00072}}$	$\mathbf{0.85}_{\pm \mathbf{0.03}}$
Risk-Aware HITL	**RAIL (Ours)**	$\mathbf{29.07}_{\pm \mathbf{11.36}}$	**30K**	$360.65_{\pm 23.06}$	$\mathbf{0.75}_{\pm \mathbf{0.03}}$	$0.0027_{\pm 0.00052}$	$\mathbf{0.85}_{\pm \mathbf{0.07}}$

Table 5: CARLA generalization with top-down semantic raster input.

Method	**Training Data**	**TSV↓**	**TR↑**	**TSR↑**
PPO [37]	500K	80.84	1591.00	0.35
HACO [25]	8K	12.14	1578.43	0.35
HAIM [19]	8K	**11.25**	1590.85	0.38
RAIL (Ours)	**8K**	12.38	**1609.70**	**0.41**

Table 6: Robustness on MetaDrive under cyber and physical intrusions.

Method	**CAN Injection**	**LiDAR Spoof**
	TSR↑ / DRA↓ / ASR↓	*TSR↑ / DRA↓ / ASR↓*
PPO [37]	0.23 / 0.85 / 0.62	0.49 / 0.21 / 0.23
HACO [25]	0.58 / 0.90 / 0.75	0.74 / 0.24 / 0.29
HAIM [19]	0.62 / 0.74 / 0.65	0.77 / 0.17 / 0.20
RAIL (Ours)	**0.68 / 0.37 / 0.34**	**0.80 / 0.03 / 0.11**

Robustness Results. Table 6 shows that RAIL is the most robust method under both intrusion types. Under Controller Area Network injection, RAIL achieves TSR 0.68 compared to PPO (0.23), HACO (0.58), and HAIM-DRL (0.62), while reducing DRA to 0.37 (HACO: 0.90, HAIM-DRL: 0.74) and ASR to 0.34 (HAIM-DRL: 0.65). Under LiDAR spoofing, RAIL attains TSR 0.80 with near-zero DRA (0.03) and ASR (0.11). The concurrent improvement in TSR and reduction in DRA and ASR indicates active containment rather than conservative avoidance: cue-driven shielding mitigates both actuation corruption and phantom obstacles without frequent human takeover or performance collapse.

Table 7: RAIL ablations on MetaDrive.

Variant	**SR↑**	**SV↓**	**DR↓**	**AIRI↓**	**AS↑**
w/o curvature cue	0.78	0.80	0.0140	0.17	22.7
w/o TTC cue	0.67	1.02	0.0220	**0.12**	22.5
w/o OOD cue	0.83	0.77	0.0049	0.15	**23.3**
w/o bandit (fixed)	0.79	1.10	0.0180	0.16	19.7
w/o weights (uniform)	0.81	0.79	0.0056	0.16	20.4
w/o shield penalty	0.74	**0.72**	0.0044	**0.12**	16.6
RAIL (full)	**0.85**	0.75	**0.0027**	0.15	22.8

SR = success rate; SV = safety violations; DR = disturbance rate; AIRI = avg. IRI per episode; AS = avg. speed (km/h).

Attack Model Grounding. The CAN injection model, implemented as bounded additive bias on actuator channels with randomized timing, reflects real-world automotive exploits in which compromised electronic control units inject arbitrary frames onto the bus [8, 27]. The LiDAR spoofing setup follows documented relay-based phantom injection and optical-channel attacks [7, 39]. Both intrusions are scheduled periodically rather than adaptively; evaluating adversaries that respond to RAIL's shielding strategy remains future work.

5.3 Component Ablation Analysis

We next address Q3 by isolating the contribution of each RAIL component under nominal conditions.

Table 7 highlights distinct roles across cues and modules.

Table 8: IRI threshold (τ) sensitivity on MetaDrive.

τ	SR↑	SV↓	DR↓	AS↑	Shield %
0.1	0.79	0.61	0.0016	18.9	41.7
0.2	0.84	0.69	0.0021	21.0	24.8
0.3	**0.86**	**0.74**	**0.0028**	**22.5**	**13.6**
0.4	0.82	0.95	0.0076	23.3	7.4
0.5	0.75	1.21	0.0142	23.8	4.6

Shield % = fraction of control steps with $\text{IRI}_t > \tau$.

Removing the **time-to-collision (TTC)** cue causes the largest degradation, reducing SR to 0.67 and increasing SV to 1.02 and DR to 0.022, as braking shields fail to activate in time. Removing the **curvature** cue lowers SR to 0.78 and raises DR to 0.014 due to unchecked actuation faults. Dropping the **out-of-distribution (OOD)** cue preserves SR (0.83) but increases DR (0.0049) and AS (23.3), reflecting reduced caution under sensor anomalies. Replacing the adaptive **contextual bandit** with a fixed shield increases SV to 1.10 and reduces AS to 19.7, demonstrating that cue-aware arbitration is necessary to match shield type to risk context. Finally, removing the **shield-effort penalty** (λ_5 in (7)) causes persistent over-shielding, resulting in a 27% reduction in average speed (16.6 vs. 22.8 km/h) and the lowest SR (0.74). This confirms that the penalty term is essential to balance safety intervention and task efficiency.

Figure 4 shows qualitative examples where IRI spikes trigger dominant-cue selection and shielded control. For each cue type, RAIL responds with a targeted intervention that averts the incident while maintaining driving progress.

5.4 Shield-Only vs. Surrogate-Only Ablation

A natural concern with any human-in-the-loop design is whether reported robustness is driven by the proactive safety layer or by the fallback operator. We therefore decompose RAIL into its two intervention mechanisms and evaluate four configurations under nominal driving and both attack regimes from Section 4.2: *RAIL (full)* with both active, *Shield only* with the scripted surrogate disabled, *Surrogate only* with cue-conditioned shielding disabled, and *Neither*, which reduces RAIL to its SAC backbone. All four variants share the same policy weights, observations, and control rate.

Table 9 shows a consistent ordering across all three settings: *Shield only* sits much closer to *RAIL (full)* than *Surrogate only* does. Under nominal driving, *Shield only* recovers most of the full-system performance (TSR 0.82 vs. 0.85), while *Surrogate only* lags at 0.76 and the base policy sits at 0.71, consistent with SAC-RS in Table 4. The separation is starkest under CAN injection, where *Shield only* cuts ASR nearly in half relative to *Surrogate only* (0.39 vs. 0.53); by the time a takeover fires, the injected steering bias has already produced a safety-relevant deviation that the curvature cue would have counteracted within a few control cycles. Under LiDAR spoofing the same ordering holds with a smaller gap (ASR 0.15 vs. 0.24), because phantom-obstacle spoofs trigger slower trajectory responses that give a reactive surrogate more time to correct. In all three cases, the residual improvement from *Shield only* to *RAIL (full)* reflects the surrogate acting as a backstop for the small number of episodes where shielding alone is insufficient, supporting the central claim that the robustness gains in Section 5.2 come primarily from cue-conditioned shielding rather than from operator intervention.

5.5 Intrusion Risk Index Sensitivity

We address Q4 by analyzing sensitivity to the threshold, cue weights, and inter-cue correlations.

Threshold Sensitivity. Table 8 varies τ across five values. At $\tau = 0.1$, shielding activates on 42% of steps, reducing average speed (18.9 km/h) and SR (0.79) due to over-braking. At $\tau = 0.5$, activation drops to 5%, and SV rises to 1.21 as threats go unmitigated. The default $\tau = 0.3$ balances safety (SV=0.74) and performance (22.5 km/h), achieving the highest SR (0.86).

Weight Sensitivity. Table 10 compares four configurations to the default $(0.3, 0.4, 0.3)$. TTC-heavy weights $(0.2, 0.6, 0.2)$ lower SV to 0.70 but increase DR (0.0058) due to excessive braking. CURV-heavy weights $(0.5, 0.25, 0.25)$ raise SV to 0.93 because TTC is under-weighted. Uniform weights $(0.33, 0.33, 0.33)$ degrade modestly across metrics. The default achieves the highest SR (0.86) and lowest DR (0.0026), supporting greater emphasis on TTC.

Cue Correlation. Figure 5 reports pairwise Spearman correlations across test steps. All off-diagonal values are below 0.35; the highest is between r_{CURV} and r_{TTC} ($\rho = 0.34$), reflecting coupled trajectory deviation and reduced headway. Correlations involving r_{OOD} remain low ($\rho \leq 0.17$), indicating independence from control cues. These results support the conditional-independence assumption underlying Noisy-OR fusion and confirm cue complementarity.

5.6 Runtime Efficiency and Operator Burden

We finally address Q5 by quantifying computational overhead and human workload during deployment.

Runtime Overhead. Table 11 reports a micro-profile over 1,000 control steps. The full risk pipeline adds 0.215 ± 0.043 ms per step, or 0.22% of a 100 ms control cycle at 10 Hz. The dominant cost arises from the observation-shift cue (r_{OOD}) at 0.168 ± 0.037 ms due to a lightweight Mahalanobis distance on the 72-beam LiDAR vector. Noisy-OR fusion, bandit selection, and $\mathbb{R}^2$ blending each operate in the microsecond range. Overall, RAIL delivers interpretable risk estimation and adaptive shielding with negligible runtime impact, preserving real-time responsiveness.



(a) Curvature (b) TTC (c) OOD

Figure 4: **Qualitative IRI responses under dominant cues.** Each column shows two representative episodes in which curvature, TTC, or OOD dominates IRI_t. In each case, RAIL activates a cue-specific shield to avert the incident while maintaining progress.

Table 9: Shield-only vs. surrogate-only ablation under nominal and adversarial conditions on MetaDrive. Best result per metric in **bold**. Higher TSR is better; lower TSV, DRA, and ASR are better. DRA is undefined for variants without any human-style intervention and is reported as 0.00 for completeness.

	Nominal				CAN Injection				LiDAR Spoofing			
Condition	**TSR↑**	**TSV↓**	**DRA↓**	**ASR↓**	**TSR↑**	**TSV↓**	**DRA↓**	**ASR↓**	**TSR↑**	**TSV↓**	**DRA↓**	**ASR↓**
RAIL (full)	**0.85**	**0.75**	0.04	**0.08**	**0.68**	**1.02**	0.37	**0.34**	**0.80**	**0.62**	0.03	**0.11**
Shield only	0.82	0.81	**0.00**	0.12	0.64	1.18	**0.00**	0.39	0.77	0.71	**0.00**	0.15
Surrogate only	0.76	0.94	0.11	0.16	0.52	1.54	0.61	0.53	0.70	0.88	0.19	0.24
Neither (base)	0.71	1.08	**0.00**	0.21	0.41	1.87	**0.00**	0.60	0.58	1.19	**0.00**	0.31

Table 10: IRI weight configuration sensitivity on MetaDrive.

Weights (w_C, w_T, w_O)	**SR↑**	**SV↓**	**DR↓**	**AS↑**
Uniform $(0.33, 0.33, 0.33)$	0.82	0.81	0.0049	21.6
TTC-heavy $(0.2, 0.6, 0.2)$	0.84	0.70	0.0058	21.2
CURV-heavy $(0.5, 0.25, 0.25)$	0.79	0.93	0.0087	22.0
OOD-heavy $(0.25, 0.25, 0.5)$	0.81	0.83	0.0039	21.0
Default $(0.3, 0.4, 0.3)$	**0.86**	**0.74**	**0.0026**	**22.7**

Operator Burden. Table 12 summarizes workload indicators for human-in-the-loop baselines. RAIL achieves the largest mean time between takeovers (MTBT) and the lowest cumulative operator active time, indicating that risk-triggered shielding resolves many hazards before human intervention is required. This shifts the operator role from continuous supervision to rare exception handling. The false-positive rate, defined as shielded steps not followed by a safety event within 50 steps, is 14.2%, meaning 85.8% of activations align with impending hazards. This selectivity reduces workload and supports trust calibration. Formal subjective workload assessments are left to future work.

6 Insights and Practical Implications

This section distills the principal empirical insights and discusses the practical implications of the RAIL architecture. Broader extensions and future research directions are deferred to the concluding section.

Core Empirical Insights. First, explicit per-cue risk decomposition closes a gap left by implicit cost-based approaches.

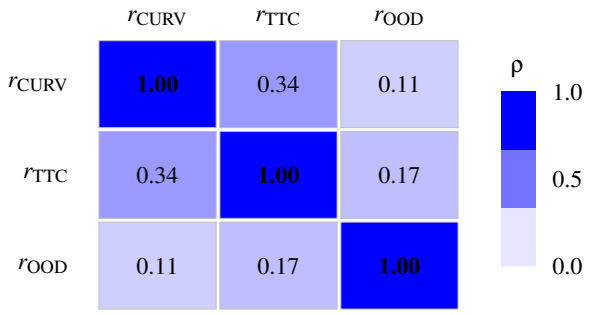


Figure 5: Pairwise Spearman correlations between IRI cues.

Table 11: Per-step computational overhead of RAIL components (mean ± std over 1,000 steps).

Component	Operation	Time (ms)
r_{CURV}	Subtract + sigmoid	0.012 ± 0.004
r_{TTC}	Divide + exp	0.010 ± 0.003
r_{OOD}	Mahalanobis ($\mathbb{R}^{72}$)	0.168 ± 0.037
IRI fusion	Noisy-OR (3 terms)	0.006 ± 0.002
Bandit selection	Softmax (3 arms)	0.014 ± 0.005
Action blending	Linear interp ($\mathbb{R}^2$)	0.005 ± 0.002
Total RAIL overhead		$\mathbf{0.215 \pm 0.043}$

While safe reinforcement learning methods reduce aggregate violations, they do not expose which failure mode triggered intervention. In contrast, RAIL attributes each shield activation to a dominant cue and threat layer, improving targeted adaptation and post-hoc auditability. Second, sensitivity analysis shows stable behavior across thresholds and weight configurations: success rate remains above 0.79 for all tested $\tau \in [0.1, 0.5]$ and weight vectors, with the default setting ($\tau = 0.3$, weights $0.3/0.4/0.3$) lying on the safety-efficiency Pareto frontier. Third, low pairwise cue correlations ($\rho \leq 0.34$) empirically support the conditional-independence assumption behind Noisy-OR fusion and validate the complementarity design principle.

Collectively, these findings demonstrate that structured, decomposed risk modeling enables simultaneous gains in safety, interpretability, and efficiency without sacrificing robustness.

Architectural and Deployment Implications. RAIL's decomposed Intrusion Risk Index provides a transparent forensic trace aligned with automotive cybersecurity documentation requirements, enabling traceable linkage between observed anomalies and mitigation actions. The cue interface is modular: new sensing or anomaly-detection modules can be integrated via monotone $[0, 1]$ mappings without redesigning the policy. The shield library is likewise composable, allowing certified low-level controllers to operate as shield arms. In this role, RAIL functions as a coordination layer over heterogeneous safety mechanisms rather than replacing them.

Table 12: Operator workload indicators on MetaDrive (test episodes).

Method	MTBT (steps)↑	Active time (s)↓	FP rate (%)↓
HACO [25]	142	8.7	–
HAIM [19]	186	6.2	–
RAIL (Ours)	**310**	**3.1**	14.2

MTBT = mean time between takeovers; Active time = cumulative operator engagement per episode; FP rate = shield activations with no safety event within 50 steps using.

Current Scope and Constraints. The present evaluation relies on a scripted human surrogate and two simulators, which capture diverse traffic and intrusion scenarios but do not encompass hardware-in-the-loop or field validation. Attack models are periodic rather than adaptive, and formal guarantees are limited to passivity, monotonicity, and bounded deviation.

Sim-to-real deployment considerations. Three deployment aspects remain open for vehicle hardware. First, real LiDAR exhibits multipath returns, weather-induced sparsity, and calibration drift that can shift the Mahalanobis baseline (μ, Σ), motivating periodic online recalibration of the perception cue. Second, the per-step overhead measured in Table 11 is small relative to a 100 ms control cycle, but real vehicle buses introduce additional CAN arbitration and sensor-fusion latencies that must be budgeted in a production timing analysis. Third, RAIL is intended to sit above existing low-level safety controllers such as automatic emergency braking and electronic stability control, not to replace them; shield arms can be instantiated with certified controllers where formal guarantees are required. These boundaries motivate the broader research agenda outlined in the conclusion.

7 Conclusions and Future Work

Building on the empirical insights and practical implications discussed above, we introduced RAIL, a risk-aware human-in-the-loop framework for autonomous vehicles operating under cyber-physical threats. RAIL fuses heterogeneous runtime cues into an Intrusion Risk Index using Noisy-OR aggregation, yielding a single yet decomposable measure of threat severity. When risk exceeds the activation threshold, a contextual bandit selects an interpretable shield, and a blending executor interpolates between nominal and safeguarded actions with authority proportional to risk. Training integrates dual reward shaping that penalizes both safety violations and unnecessary shielding, while a risk-prioritized replay buffer revisits rare but safety-critical transitions.

Across MetaDrive and CARLA and 12 baselines spanning reinforcement learning, constrained safe RL, offline learning, imitation learning, and prior human-in-the-loop methods, RAIL matches expert-level success rates (0.85 on MetaDrive and 0.41 on CARLA) while achieving the lowest test and

training safety violations using 30K interaction steps. Under Controller Area Network injection and LiDAR spoofing, RAIL reduces attack success rates by up to 48% and operator disengagements by up to 50% relative to the strongest human-in-the-loop baseline, while maintaining the highest task success rate under attack. These results show that explicit per-cue risk modeling converts threat awareness into graded, auditable control actions rather than binary fail-safes, supporting safer and more transparent human-AI collaboration.

Several directions remain for future investigation. Adaptive thresholding, online cue-weight learning, and adversarial training against white-box and persistent attackers could allow the operating point to adjust to changing traffic density, sensor conditions, and threat intensity, extending RAIL beyond the bounded-burst regime studied here. Extending the Intrusion Risk Index to multi-agent settings would enable cooperative risk sharing through compact vehicle-to-everything cue exchange. Hardware-in-the-loop validation on a drive-by-wire platform with real Controller Area Network traffic and physical sensors would further assess timing constraints and sim-to-real robustness. Finally, controlled human factors studies are needed to quantify how graded authority and selective shielding influence trust calibration and cognitive workload, complementing the log-based indicators reported here.

Ethical Considerations. RAIL is defensive in nature, aimed at improving the safety and cyber-physical resilience of autonomous vehicles. All experiments are conducted in simulation (MetaDrive and CARLA) with no real vehicles, communication buses, or human subjects. The attack models evaluated, including CAN bus injection and LiDAR spoofing, are grounded in prior automotive security research [7, 8, 27, 39] and do not introduce novel offensive techniques. While we describe attack parameterizations to support reproducibility, the underlying vulnerabilities are well documented. We believe the defensive contributions of this work, including interpretable risk decomposition, graded shielding, and reduced operator burden, outweigh any residual dual-use risk.

Acknowledgments

This work was supported by the Army Research Office under Grant No. W911NF-24-2-0241.

Use of Generative AI

Generative AI tools were used for editorial purposes in this manuscript, and all outputs were inspected by the authors to ensure accuracy and originality.

References

[1] ABDO, A., MALEK, S. M. B., ZHAO, X., AND ABU-GHAZALEH, N. Avmon: securing autonomous vehicles by learning control invariants and residual prediction. In *Symposium on Vehicle Security and Privacy (VehicleSec)* (2024).

[2] ACHIAM, J., HELD, D., TAMAR, A., AND ABBEEL, P. Constrained policy optimization. In *International conference on machine learning* (2017), PMLR, pp. 22–31.

[3] ALORAINI, F., JAVED, A., AND RANA, O. Adversarial attacks on intrusion detection systems in in-vehicle networks of connected and autonomous vehicles. *Sensors 24*, 12 (2024), 3848.

[4] ALSHIEKH, M., BLOEM, R., EHLERS, R., KÖNIGHOFER, B., NIEKUM, S., AND TOPCU, U. Safe reinforcement learning via shielding. In *Proceedings of the AAAI conference on artificial intelligence* (2018), vol. 32.

[5] AMES, A. D., XU, X., GRIZZLE, J. W., AND TABUADA, P. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control 62*, 8 (2016), 3861–3876.

[6] BAIN, M., AND SAMMUT, C. A framework for behavioural cloning. In *Machine intelligence 15* (1995), pp. 103–129.

[7] CAO, Y., XIAO, C., CYR, B., ZHOU, Y., PARK, W., RAMPAZZI, S., CHEN, Q. A., FU, K., AND MAO, Z. M. Adversarial sensor attack on lidar-based perception in autonomous driving. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security* (2019), pp. 2267–2281.

[8] CHECKOWAY, S., MCCOY, D., KANTOR, B., ANDERSON, D., SHACHAM, H., SAVAGE, S., KOSCHER, K., CZESKIS, A., ROESNER, F., AND KOHNO, T. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX security symposium (USENIX Security 11)* (2011).

[9] CHO, K.-T., AND SHIN, K. G. Fingerprinting electronic control units for vehicle intrusion detection. In *25th USENIX security symposium (USENIX Security 16)* (2016), pp. 911–927.

[10] CHOI, J., CASTAÑEDA, F., TOMLIN, C., AND SREENATH, K. Reinforcement learning for safety-critical control under model uncertainty, using control lyapunov functions and control barrier functions. *Robotics: Science and Systems XVI* (2020).

[11] CHOUDHARY, G., SHARMA, V., YOU, I., YIM, K., CHEN, R., AND CHO, J.-H. Intrusion detection systems for networked unmanned aerial vehicles: A survey. In *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)* (2018), IEEE, pp. 560–565.

[12] FISAC, J. F., AKAMETALU, A. K., ZEILINGER, M. N., KAYNAMA, S., GILLULA, J., AND TOMLIN, C. J. A general safety framework for learning-based control in uncertain robotic systems. *IEEE Transactions on Automatic Control 64*, 7 (2018), 2737–2752.

[13] GETTMAN, D., AND HEAD, L. Surrogate safety measures from traffic simulation models. *Transportation research record 1840*, 1 (2003), 104–115.

[14] GREENBERG, I., CHOW, Y., GHAVAMZADEH, M., AND MANNOR, S. Efficient risk-averse reinforcement learning. *Advances in Neural Information Processing Systems 35* (2022), 32639–32652.

[15] HA, S., XU, P., TAN, Z., LEVINE, S., AND TAN, J. Learning to walk in the real world with minimal human effort. *arXiv preprint arXiv:2002.08550* (2020).

[16] HAARNOJA, T., ZHOU, A., ABBEEL, P., AND LEVINE, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning* (2018), Pmlr, pp. 1861–1870.

[17] HAMAD, M., FINKENZELLER, A., KÜHR, M., ROBERTS, A., MAENNEL, O., PREVELAKIS, V., AND STEINHORST, S. React: Autonomous intrusion response system for intelligent vehicles. *Computers & Security 145* (2024), 104008.

[18] HO, J., AND ERMON, S. Generative adversarial imitation learning. *Advances in neural information processing systems 29* (2016).

[19] HUANG, Z., SHENG, Z., MA, C., AND CHEN, S. Human as ai mentor: Enhanced human-in-the-loop reinforcement learning for safe and efficient autonomous driving. *Communications in Transportation Research 4* (2024), 100127.

[20] ISO/SAE 21434, C. Road vehicles—cyber security engineering, 2021.

[21] JOHANSEN, I. L., AND RAUSAND, M. Foundations and choice of risk metrics. *Safety science 62* (2014), 386–399.

[22] KASTNER, T., ERDOGDU, M. A., AND FARAHMAND, A.-M. Distributional model equivalence for risk-sensitive reinforcement learning. *Advances in Neural Information Processing Systems 36* (2023), 56531–56552.

[23] KELLY, M., SIDRANE, C., DRIGGS-CAMPBELL, K., AND KOCHENDERFER, M. J. Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)* (2019), IEEE, pp. 8077–8083.

[24] KUMAR, A., ZHOU, A., TUCKER, G., AND LEVINE, S. Conservative q-learning for offline reinforcement learning. *Advances in neural information processing systems 33* (2020), 1179–1191.

[25] LI, Q., PENG, Z., AND ZHOU, B. Efficient learning of safe driving policy via human-ai copilot optimization. *arXiv preprint arXiv:2202.10341* (2022).

[26] MANDLEKAR, A., XU, D., MARTÍN-MARTÍN, R., ZHU, Y., FEI-FEI, L., AND SAVARESE, S. Human-in-the-loop imitation learning using remote teleoperation. *arXiv preprint arXiv:2012.06733* (2020).

[27] MILLER, C., AND VALASEK, C. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA 2015*, S 91 (2015), 1–91.

[28] NAGARAJAN, J., MANSOURIAN, P., SHAHID, M. A., JAEKEL, A., SAINI, I., ZHANG, N., AND KNEPPERS, M. Machine learning based intrusion detection systems for connected autonomous vehicles: A survey. *Peer-to-Peer Networking and Applications 16*, 5 (2023), 2153–2185.

[29] NO, U. R. 155-cyber security and cyber security management system. *The United Nations Economic Commission for Europe (UNECE)* (2021).

[30] PAREKH, D., PODDAR, N., RAJPURKAR, A., CHAHAL, M., KUMAR, N., JOSHI, G. P., AND CHO, W. A review on autonomous vehicles: Progress, methods and challenges. *Electronics 11*, 14 (2022), 2162.

[31] PASQUALETTI, F., DÖRFLER, F., AND BULLO, F. Attack detection and identification in cyber-physical systems. *IEEE transactions on automatic control 58*, 11 (2013), 2715–2729.

[32] PEARL, J. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[33] PENDLETON, M., GARCIA-LEBRON, R., CHO, J.-H., AND XU, S. A survey on systems security metrics. *ACM Computing Surveys (CSUR) 49*, 4 (2016), 1–35.

[34] QUEENEY, J., AND BENOSMAN, M. Risk-averse model uncertainty for distributionally robust safe reinforcement learning. *Advances in Neural Information Processing Systems 36* (2023), 1659–1680.

[35] REN, H., AND HUANG, T. Adversarial example attacks in the physical world. In *International Conference on Machine Learning for Cyber Security* (2020), Springer, pp. 572–582.

[36] SAUNDERS, W., SASTRY, G., STUHLMUELLER, A., AND EVANS, O. Trial without error: Towards safe reinforcement learning via human intervention. *arXiv preprint arXiv:1707.05173* (2017).

[37] SCHULMAN, J., WOLSKI, F., DHARIWAL, P., RADFORD, A., AND KLIMOV, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[38] SHA, L. Using simplicity to control complexity. *IEEE Software 18*, 4 (2001), 20.

[39] SHIN, H., KIM, D., KWON, Y., AND KIM, Y. Illusion and dazzle: Adversarial optical channel exploits against lidars for automotive applications. In *International conference on cryptographic hardware and embedded systems* (2017), Springer, pp. 445–467.

[40] STOOKE, A., ACHIAM, J., AND ABBEEL, P. Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning* (2020), PMLR, pp. 9133–9143.

[41] SULTANA, R., GROVER, J., TRIPATHI, M., AND SHARMA, P. Ladetects: Local and adaptive data-centric misbehavior detection framework for vehicular technology security. *IEEE Open Journal of Vehicular Technology* (2024).

[42] TAMAR, A., CHOW, Y., GHAVAMZADEH, M., AND MANNOR, S. Policy gradient for coherent risk measures. *Advances in neural information processing systems 28* (2015).

[43] TAMAR, A., GLASSNER, Y., AND MANNOR, S. Optimizing the cvar via sampling. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2015), vol. 29.

[44] YANG, Z., JIN, H., TANG, Y., AND FAN, G. Risk-aware constrained reinforcement learning with non-stationary policies. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems* (2024), pp. 2029–2037.





USENIX VehicleSec '26 Artifact Appendix: Risk-Aware Human-in-the-Loop Framework with Adaptive Intrusion Response for Autonomous Vehicles

Dawood Wasif[1], Terrence J. Moore[2], Seunghyun Yoon[3], Hyuk Lim[3], Dan Dongseong Kim[4], Frederica F. Nelson[2], Jin-Hee Cho[1]

[1]Virginia Tech [2]U.S. Army Research Laboratory [3]Korea Institute of Energy Technology
[4]University of Queensland

A Artifact Appendix

A.1 Abstract

This artifact provides the complete implementation of RAIL, a risk-aware human-in-the-loop framework for autonomous vehicles that unifies runtime cyber–physical risk estimation, adaptive shielding, action arbitration, and risk-prioritized learning. It contains the RAIL framework (the IRI risk score via weighted Noisy-OR cue fusion, a contextual bandit for shield selection, a risk-conditioned action-blending executor, and a risk-prioritized replay buffer), the `RAILEnv` wrapper with CAN-injection and LiDAR-spoofing attacks, all comparison baselines, and end-to-end scripts. It supports five paper claims (C1–C5; see Major Claims).

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact is defensive and poses no security or privacy risk to evaluators. All experiments run entirely in simulation (MetaDrive and CARLA 0.9.9.4); no real vehicles, physical CAN buses, hardware sensors, or human subjects are involved. The CAN-injection and LiDAR-spoofing attacks reproduce previously documented automotive cyber–physical scenarios as parameterized simulator perturbations and add no novel offensive capability. Human demonstration data for some baselines was collected by a co-author via keyboard control in simulation only. No personal data is collected or processed.

A.2.2 How to access

The artifact is permanently archived on Zenodo with a stable DOI: **10.5281/zenodo.20451598** (`https://doi.org/10.5281/zenodo.20451598`).

A.2.3 Badges

This artifact was awarded the *Artifacts Available* badge (source code, configuration files, and the human-demonstration dataset publicly archived on Zenodo with a stable DOI) and the *Artifacts Functional* badge (all major components described in the paper, with documentation and end-to-end scripts that run the reported experiments).

A.2.4 Hardware dependencies

A modern x86-64 CPU ($\geq$4 cores, 8+ recommended for Ray parallelism); $\geq$16 GB RAM (32 GB with CARLA); one NVIDIA GPU with $\geq$8 GB VRAM (all results obtained on an RTX 3090, 24 GB; `--num-gpus=0` allows slow CPU-only runs); $\geq$20 GB disk (plus 15–30 GB for CARLA); and a connected display or virtual framebuffer (`Xvfb`) for rendered runs (set `use_render: False` for headless training). Training takes $\sim$2.5 h per method (MetaDrive, 30 K budget) and $\sim$3 h (CARLA) on one RTX 3090.

A.2.5 Software dependencies

Ubuntu 20.04/22.04 and Python 3.7, managed with Conda. Core libraries: `metadrive-simulator==0.2.4`, `ray==1.0.0`, `tensorflow==2.4.0` (with `tensorflow-probability==0.11.1`), `gym==0.19.0`, the standard SciPy-stack packages, and CUDA 11.0 with cuDNN. The CARLA experiment (Table 5) additionally needs the CARLA 0.9.9.4 server and `di-engine==0.2.0`; the GAIL/HG-DAgger/IWR baselines need `pytorch==1.7.1` in a separate Conda environment.

A.2.6 Benchmarks

All experiments are simulation-only. MetaDrive 0.2.4 uses procedurally generated maps (50 training maps, seeds 100–149; 50 held-out maps, seeds 500–549); CARLA uses Town02/Town05 routes via DI-Drive. No external dataset is required for generation. A human-demonstration

dataset (`data/human_demos.pkl`, 49 K steps collected in simulation) is included in the Zenodo archive for the imitation/offline-RL baselines and is regenerable via the provided script.

A.3 Set-up

A.3.1 Installation

Obtain the code from the Zenodo DOI, then create the MetaDrive environment:

```
conda create -n RAIL python=3.7 -y
conda activate RAIL
pip install -e .
conda install cudatoolkit=11.0 -y
conda install -c nvidia cudnn -y
```

Optionally create two more environments (same `python=3.7`, `cudatoolkit=11.0`): `RAIL-carla` (add `di-engine==0.2.0`, `markupsafe==2.0.1`) for the CARLA experiment, and `RAIL-torch` (add `pytorch==1.7.1`) for the GAIL/HG-DAgger/IWR baselines.

A.3.2 Basic Test

Verify the installation and environment wrapper with a short quick-start run:

```
conda activate RAIL
cd RAIL/scripts/run_rail
python train_rail_keyboard_easy.py \
  --num-gpus=1
```

Expected: Ray starts, MetaDrive launches, and runtime metrics such as `success`, `takeover`, and `disturbance` print to the console; $\sim$5000 steps finish in under 15 minutes on a single GPU.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *Nominal MetaDrive performance.* RAIL reaches expert-level success while attaining the lowest safety and training-time violations at the 30 K interaction budget. Proven by (E1); Table 4.

(C2): *CARLA generalization.* RAIL transfers to CARLA with TSR $= 0.41$ and TR $= 1609.70$ at only 8 K effective steps. Proven by (E2); Table 5.

(C3): *Adversarial robustness.* Under CAN injection and LiDAR spoofing, RAIL attains the lowest attack-success and disengagement metrics. Proven by (E3); Table 6.

(C4): *Complementary components.* Removing any component (a cue, the bandit, cue weighting, or shield-effort regularization) degrades $\geq$1 key metric. Proven by (E4); Table 7.

(C5): *Sensitivity stability.* Performance is stable across IRI thresholds and cue weights, with the default on the safety–efficiency frontier. Proven by (E5); Tables 8–9.

A.4.2 Experiments

(E1): *Nominal benchmarking* [$\sim$2.5 h/method]: run `train_rail.py` in `scripts/run_rail/`, plus the baseline `train_*.py` scripts in `run_haim/` and `run_baselines/` (HAIM-DRL, HACO, SAC/PPO $\pm$Lagrangian, CQL with `ray==1.2.0`, and GAIL/HG-DAgger/IWR in `RAIL-torch`). Matches Table 4 within $\pm$0.05 TSR / $\pm$0.10 TR over 5 seeds, ordering preserved (validates C1).

(E2): *CARLA generalization* [$\sim$3 h]: start `./CarlaUE4.sh -carla-rpc-port=9000`, then in `RAIL-carla` run `train_rail_in_carla.py` (baselines: `train_haim_drl_in_carla.py`, `train_haco_in_carla.py`). TSR within $\pm$0.03, TR within $\pm$50 of Table 5 (validates C2).

(E3): *Adversarial robustness*: train with `train_rail.py --attack-mode=cyber` (CAN) and `=physical` (LiDAR); evaluate via `eval_rail.py --ckpt-idx 200 --num-episodes 50 --attack-mode {cyber,physical}`. ASR/DRA within $\pm$0.05 of Table 6 (validates C3).

(E4): *Ablation*: in the `train_rail.py` config, zero a cue (`iri_w_ttc`/`iri_w_curv`/`iri_w_ood`=0), use a fixed dominant-cue arm (no bandit), set all weights to 0.333, or set `lambda5`=0. Each degrades $\geq$1 metric (most when TTC is dropped), per Table 7 (validates C4).

(E5): *Sensitivity*: sweep `iri_tau`$\in \{0.1, \ldots, 0.5\}$ and the cue weights per Table 9, re-running `train_rail.py` after each edit. SR stays $>$0.79; default near the best trade-off (validates C5).

A.5 Notes on Reusability

RAIL is modular: add risk cues in `scripts/utils/iri_calculator.py` (return $[0,1]$ values and register them in `IRICalculator.compute()`); add shield actions in `shield_bandit.py`; reuse `RAILEnv` in environments exposing curvature mismatch, obstacle-closing risk, and LiDAR-like ranges; and pair the runtime safety layer with other RL backbones, since the wrapper and replay buffer are learner-independent. All major parameters are configuration-exposed for systematic sweeps.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.

ARTIFACT EVALUATED
USENIX
AVAILABLE

Reversing a Brake ECU Firmware Update

Ben Gardiner
National Motor Freight Traffic Association Inc.

Abstract

This paper presents a reverse engineering analysis of a firmware update for the Bendix EC80 Brake ECU, a critical component in heavy-duty vehicles. Unlike prior research focusing on CAN or UDS, this study isolates vulnerabilities in the legacy J2497 Power Line Carrier (PLC4TRUCKS) interface. We analyze the "ID9363" updater executable to extract firmware and perform differential binary analysis of the S12X architecture firmware images. Our research identifies specific changes in the recall that address undocumented vulnerabilities in J1587 PID processing and interrupt handling on J2497.

We demonstrate that the patched functionality contained critical flaws, including a write-where primitive and denial-of-service (DoS) conditions. These vulnerabilities were validated on a test bench and in a moving truck, where exploitation caused the loss of speedometer, dynamic steering, and automatic shifting. The findings confirm the safety recall remediation was also a security patch, mitigating risks in legacy protocol implementations. This work motivates the necessity of scrutinizing legacy interfaces in general in modern heavy vehicle safety systems, and the J2497 interface in particular.

1 Introduction

Introduced circa 2001, the J2497 [1] (aka 'PLC4TRUCKS') powerline databus remains the only industry-standard means to satisfy the trailer ABS warning light requirement of FMVSS 121 [2], S5.1.6.2(b). As such, J2497 has been present in all towing application Class 8 vehicles in North America from 2001 to the present.

Previous research on J2497 ECU security [3] [4] [5] [6] revealed that J2497 is wirelessly reachable and trailer brake controller ECUs implement diagnostics functions inside J1587 Data Link Escapes beyond FMVSS 121 requirements. However, current threat models assume tractor brake controllers only process necessary LAMP messages on J2497.

The 2024 Bendix EC80 safety recall [7] revealed safety impacts to tractor ECUs from J2497 reception, suggesting the EC80 processed J2497 traffic beyond necessary LAMP messages. The safety recall was remediated via a firmware update of the EC80 using the 'ID9363' updater executable.

We performed binary differential analysis on the S12X firmware (FW) pre- and post-recall in 3 types of the EC80 (one for each OEM affected by the recall). Results show extensive processing of J1587 messages received by the tractor brake controller over J2497 in the removed code. This code contained vulnerabilities including hardcoded secrets protecting traction control configuration, buffer overflows, and memory corruption leading to denial of service (of the vehicle) and remote code execution. Finally, we identified the removal of PID handlers discoverable through simple fuzzing. This suggests the safety recall also functions as a security patch, preventing attackers with wireless, 'adjacent' [6], access to J2497 from exploiting these vulnerabilities. This paper focuses exclusively on changes that were 'fixed' by this patch; no '0day' vulnerabilities are revealed as all have been patched.

Contributions. In summary, we make the following contributions:

- **FW Update Process**: We reverse engineered the operations performed by ID9363 from several captures of the UDS [8] traffic (Section 3)
- **EC80 Firmware Partitions**: The flash layout/partitioning and fixed addresses of the EC80 are revealed within the constraints of the S12X architecture (Section 4)
- **S12X Reverse Engineering**: We present novel methods for reversing the S12X banked memory architecture. We detail memory map reconstruction, symbolic analysis, and opportunistic function identification, overcoming significant tooling gaps (Sections 5, 6 and 7)
- **The Security Patch**: The functionality removed by the

patch is confirmed on all three types of EC80 units acquired and the vulnerabilities present in the removed functionality are itemized (Section 8)

- **Exploitability of Removed Functionality**: We evaluate the exploitability of each of the patched vulnerabilities from a mix of: bench and in-motion vehicle testing (Sections 9 and 10).

2 Background

In late 2024, three Class 8 vehicle OEMs in North America issued a safety recall [9] [10] [11]; these were the three OEMs that integrate the Bendix EC80 brake controller. Bendix identified potential memory corruption causing the ECU to go offline and worked with OEMs to deploy a firmware update for affected trucks – those with Electronic Stability Program (ESP) or Automatic Traction Control (ATC) features – estimated at 450,000 units at the time of this writing [12]. The timeline of this recall is summarized below in Figure 1 (see also [13] [14] [15]). See [16] for further details on the recall. The recall was later expanded in October 2025 to cover some units which were sold as aftermarket equipment [17] [18] [1].

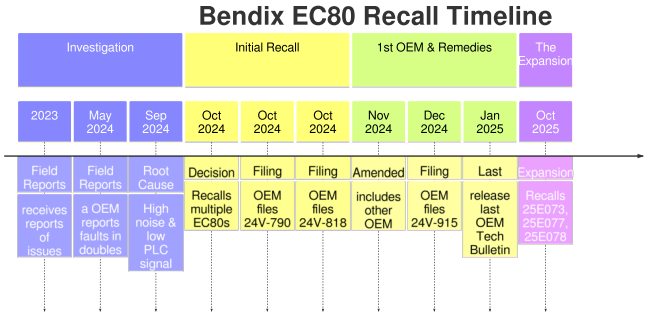


Figure 1: Bendix Recall Timeline.

The firmware update was distributed to fleets via a Windows executable, 'ID9363' [19] [20] [21]. We examine the ID9363 update process and the resulting firmware changes in three EC80 ECUs (one per OEM).

The firmware update, or 'patch', purportedly prevents J2497 noise from triggering memory corruption. However, due to CVE-2022-26131 [6], it is possible to wirelessly inject signals onto J2497. It is also possible to inject from compromised connected devices such as the increasingly common trailer telematics devices. We examine if these memory corruptions could be triggered by malicious actors using wireless (or wired) injection on J2497.

J1587 [22] serves as (roughly) the application layer for J2497, while J1708 [23] defines the original physical and data link layers (i.e. "framing"). The J1708 physical layer is replaced with a Power Line Carrier (PLC) modulated over the 12V auxiliary power line in J2497, but it inherits the J1708 data link and J1587 application layers. A frame can only be a maximum of 21 bytes long and must end with a one-byte checksum (the two's complement of the sum of the preceding bytes). In software, the frames are received byte-by-byte over a standard UART peripheral via a J1708 to J2497 converter chip (the Intellon SSCP485 [24]).

For the J1587 application layer, the first byte is defined by J1708 and is the Message Identification (MID), which identifies the sender or the message type. MIDs 0x0A (LAMP ON), 0x0B (LAMP OFF), and 0x57 (Active Trailer ABS Event) are required by the J2497 standard. For other J1587 messages the bytes following the MID are Parameter Identification (PID), which specify the type of attached data (e.g., VIN, wheel speed, diagnostics) followed by parameters (payload). Some PIDs have a fixed data length (like a 1-byte speed value), while others are variable length. PIDs data enables transmission of vehicle signals, diagnostic commands, and proprietary data.

The Bendix EC80 is a heavy-duty vehicle Electronic Control Unit (ECU) responsible for Anti-lock Braking (ABS), ATC, and ESP functions. It controls pneumatic circuits through external pressure modulation valves connected to PCB-mounted FET drivers. During initialization, absent critical Diagnostic Trouble Codes (DTCs), modulators undergo individual testing, producing a 'roll call' of small air chuffs [2].

Table 1 provides an overview of the EC80 units acquired for this study. The complexity of the EC80 firmware varies with the integrated features. The middle target, `2ec80`, represents a medium complexity unit and was selected as the primary target for reverse engineering in this paper. All findings have been validated across all the target firmwares.

The EC80 units affected were released in April 2020 [20]. Given its role in preventing rollovers and maintaining stability, brake controllers are expected to have a high Automotive Safety Integrity Level (ASIL), likely ASIL C or D, to meet OEM safety targets. While specific ASIL targets for EC80 vehicles are unpublished, Hazard Analysis and Risk Assessment (HARA) of brake controllers [25] identified ASIL D scenarios. While focused on autonomous systems, this analysis notes the architecture applies to non-autonomous vehicles, stating the brake system is critical regardless of autonomy. Furthermore, the literature for a competing product, the mBSP, explicitly states compliance with ISO 26262 [26] up to ASIL D [27].

A Threat and Risk Assessment (TARA) of Class 8 vehicles [28] ranked brake controllers third, after telematics and gateway devices, for cybersecurity priority. The EC80 con-

[1] all recalls for this same issue known at this time: 24V780000, 24V818000, 24V915000, 25E073000, 25E077000, 25E078000.

[2] In a bench environment the same successful power-on results in a clicking of a failsafe 'diagonal' relay.

Target Name	1ec80	2ec80	3ec80
FW version, before	Z228999	Z266494	Z286098
FW version, after	Z300822	Z302578	Z302579
Sticker Name	EC80ESP+	EC80ESP	EC80ESP
Non-sticker Feature	J1708	-	-
Sticker Features	6S/6M	6S/6M	4S/4M
	PLC	PLC	PLC
	2nd CAN	2nd CAN	-
	-	CAN Gateway	-
	Integrated TPMS	-	-
HW Date Code	20220723	20230507	20221019
SW Date Code	3H0922G	3E2323G	2F1423G

Table 1: Information on the three EC80s acquired and tested. While specific part numbers and OEM integrators are omitted, the firmware versions presented are consistent with recall documentation and can be retrieved from the target ECU via a Read Data By Identifier request (0xF194). The order here matches Figure 2 left to right.

nects to multiple interfaces, including "Untrustworthy Network Domains" like J2497 [29]. Because it is not explicitly intended to transport, translate, or filter traffic, it is classified as an "unintended gateway" device [29] [3]. Consequently, it must satisfy cybersecurity requirements NGW-S-001 (Security Assurance), NGW-S-002 through -005 (Won't Transport, Translate, Filter, etc.) [29].

The EC80 uses two S12X microcontrollers (as can be seen in the PCBs pictured in Figure 2). Both are specifically the NXP MC9S12XEQ512 [30], a 16-bit microcontroller from the S12X family; however, the 'left' MCU is in a smaller footprint 80-pin QFP package compared to the larger 'right' MCU in a 144-pin LQFP package.

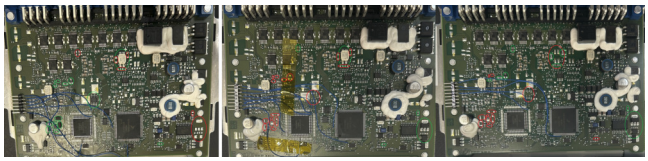


Figure 2: The PCBs of the three EC80s acquired and tested. A 0.1" header is installed for access to the BDM programming pins on the right S12X MCU. The added/removed components are highlighted in green/red, respectively.

The S12X [31] features an XGATE peripheral coprocessor to offload high-speed data transfer and logic processing from the main CPU12X core. The S12X employs a paging architecture using PPAGE, EPAGE, and RPAGE registers to map 16-bit logical windows into a 23-bit global address space. The device includes 512 KiB of PFLASH (Program Flash) residing at global addresses 0x780000 - 0x7FFFFF and 32 KiB of DFLASH (Data Flash) located at 0x100000 - 0x107FFF. Additionally, it features a 4 KiB Emulated EEPROM (EEE) buffer RAM mapped to global addresses 0x13F000 - 0x13FFFF, which allows the firmware to treat the DFLASH as random-access non-volatile memory.

3 ID9363 Update Process

We observed several EC80 firmware updates. The new firmware is readable in cleartext from CAN databus captures; however, analyzing the update requires the pre-update firmware contents. While some ECUs offer a UDS 'upload' service, we confirmed the EC80 does not. From the traffic we reconstructed the seed-key exchange routine in closed-form for all DSC,SA sessions (see Appendix A), confirming no firmware upload service exists in the observed sessions (DSC=2,SA=5) (DSC=3,SA=1) (DSC=3,SA=7) (DSC=3,SA=3) [4].

We used the BDM interface to dump PFLASH, DFLASH, and EEE (emulated in DFLASH) from both MCUs to obtain the pre-update firmware. We found that both the XPROG tool [32] and PROGS12Z [33] worked well. XPROG creates flat binary files of PFLASH, DFLASH and EEE which correspond to their global address extents. PROGS12Z creates .s19 files [34] which contain the global address extents of PFLASH, DFLASH, and EEE.

We found that although the S12X used in the EC80 supports read-out protection, it was not enabled. We confirmed that the data transferred over CAN matches the PFLASH contents after update. Only the 'right' MCU (larger package) receives code changes; the 'left' MCU's PFLASH remained unchanged. It did, however, receive some DFLASH / EEE changes during the update process. We focus solely on the 'right' MCU's code changes.

A visualization of the byte-by-byte difference of PFLASH before and after update of a 2ec80 is presented in Figure 3. Byte differences are marked by pink fill, red outline boxes, and are numerous (this is clearly not a micropatch [35]). The "Drive Block"s 1 and 2 are labeled. The PPAGE numbers corresponding to the PFLASH offset are labeled at the bottom for reference. The same style of visualization will be reused throughout the paper to illustrate successive analysis steps. In this figure function prolog estimates are plotted as green circles (in subsequent figures the functions resulting from firmware analysis will replace these).

Despite extensive byte differences, the high-level source code changes may be concise. Small machine code changes and linking order can cause massive byte differences. Byte-level analysis cannot reveal specific code changes or isolate areas

[3] Although the 'Z' version which implements a 'CAN Gateway' (see Table 1) could be considered a '(intentional) gateway device'

[4] memory leaks, other unintended upload functionality in the bootloader or application are out of the scope of this paper

of interest, as changes span 'Drive Blocks' 1 and 2. Understanding the changes requires disassembling and comparing the pre- and post-update images.

4 EC80 Bootloaders and Application

Global PFLASH byte differences (Figure 3) and ID9363 updater traffic analysis (Table F.1) indicate a J1939 [36] CAN UDS [8] bootloader in the skipped region. We label this the 'late' bootloader as it does not contain the reset vector. Code between the reset vector and this late bootloader is labeled the 'early' bootloader. Its code likely resides in some subset of PPAGE 0xFF (aka "Drive Block 2 of 2") which has a fixed mapping in the S12X memory space.

The bootloaders presumably hand-off to the 'application'. Its code is very likely in the "Drive Block" 1 of 2 region, although it could have some code in the other "Drive Block" as well to make use of the fixed mapping of PPAGE 0xFF.

5 Limitations of Static Analysis

The S12X primarily uses a 16-bit memory space with banked windows (PPAGE, DPAGE, RPAGE) to access global memory [30], rather than direct global addressing (Figure 5) [5]. Disassemblers must model this mapping to correctly resolve page-dependent references to the global address space. Fixed mappings, like PPAGE 0xFF at 0xC000-0xFFFF, are straightforward. The reset vector is in this address so starting analysis with a disassembler is likewise straightforward. In all EC80 firmwares, execution quickly encounters a call instruction with a 3-byte target: a 16-bit offset and a PPAGE. At this point the disassembler needs to have the code in the correct location in its memory model.

[5] although specific, slower execution, instructions exist for direct global address access.

There are not many examples of S12X architecture reversing available. Of course the seminal work of [37] did include S12X reversing of a power assist steering module, and it did include uncovering the use of hardcoded passwords there for UDS [8] seed-key exchange (the security access $27 service). But the 'how' of creating dumps of PFLASH, DFLASH, RAM and assembling it for a reverse engineering tool (e.g. IDA Pro as used by [37]) are not available. The recent analysis by Pulse Security of a motorcycle ECU [38] demonstrates a way to assemble an S12X target binary image for analysis in Ghidra [39] before switching to hunting for numeric tables in Flash. Ghidra does not support the XGATE coprocessor, whereas IDA Pro [40] does; therefore IDA Pro was selected for this analysis. The IDA Pro support team was kind enough to provide a code snippet of how PPAGE is modeled in the IDA linear address space model. It corresponds to how all of Ghidra, the HSW12 open source assembler [41], and the original S12X debugger, HiWave [42] treat it: use PPAGE as the most significant byte in a three-byte address (e.g. the gray boxes in Figure 6). The same is done for RPAGE and EPAGE [6]. Because RPAGE and PPAGE windows are discontiguous in the 16-bit local space this does not cause collisions. This addressing scheme is very useful and will be used in the remainder of the paper (i.e. any three byte addresses other than global addresses imply the first byte is the page and the remaining bytes are the offset within it). IDA also expects to be able to address the global memory space in the exact same address values in its linear memory map (the peach boxes in Figure 6); however, this is only used for instructions that do global memory space access.

We used IDA Pro Python scripts to map PFLASH to global addresses and copy it to high-byte and fixed PPAGE locations in the 16-bit local space (Figure 6, Appendix I.0.1).

[6] This causes IDA linear space collisions between EPAGE 0x10/0x13 and DFLASH global addresses. No workaround exists; users must be cautious if firmware uses both.

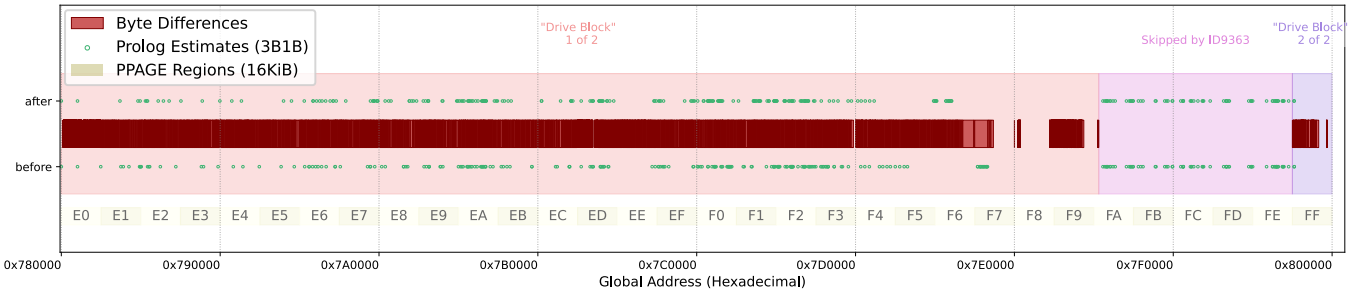


Figure 3: Comparison of the PFLASH of 2ec80 before and after ID9363 update. The unchanged extents illustrate that both the interrupt vector table at the end of PFLASH (0x7E8800) and the region 0x7E8800 - 0x7FC000 were skipped by the firmware update. Many other locations where the PFLASH contents are unchanged were found to have value 0x3F '?', the Software Interrupt ('SWI') instruction, which we believe is used as a padding value. The number of 0x3F bytes increased, suggesting function deletion by the update. Function prolog estimate locations across the update suggest shifting to lower addresses which (also) suggests function deletion by the update (examined in Section 8).

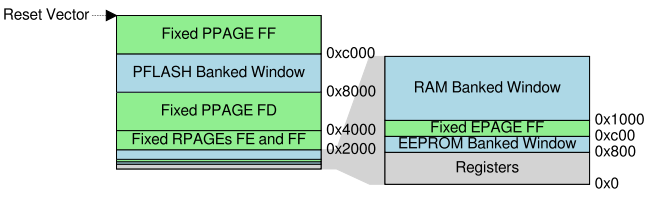


Figure 5: Local 16-bit memory map of S12X by linkerscope. Green for fixed page areas and blue for paged windows.

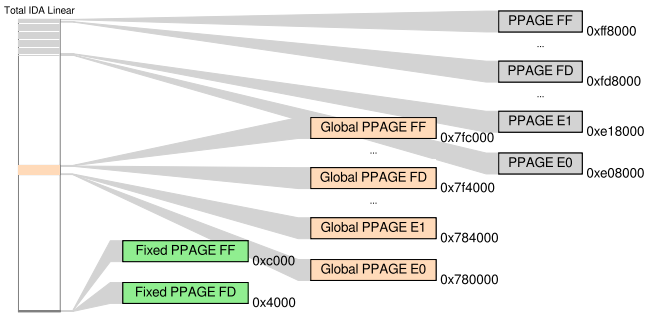


Figure 6: Illustration of the copies of PFLASH placed in the IDA Linear Address space memory map for some success in static analysis.

We then created an entrypoint for every interrupt vector in 0xFF80 - 0xFFFF, including reset, CAN, Programmable Interrupt Timer (PIT), Serial Communications Interface (SCI), and all other peripherals with an interrupt. IDA Pro analysis results are shown in Figure 4, with function extents replacing the prolog estimates used previously in Figure 3.

Analysis proceeds past three-byte calls, confirming correct PPAGE installation. While analysis reaches the late bootloader, it fails to reach application code. Starting from the reset vector often yields incomplete coverage due to unresolved data-dependent jump or call targets in many firmwares and the EC80 is no exception.

We focus on J2497 processing changes, so understanding the entire EC80 bootloader is unnecessary. The EC80 receives J2497 data via an Intellon SSCP485 [24] chip connected to the S12X SCI2 peripheral. However, the SCI2 interrupt vector of the default table points to a dummy function, and no discovered functions reference SCI2 data registers.

6 Limitations of Dynamic Analysis

Since static analysis failed to reveal SCI2 access, we used the unlocked BDM for dynamic analysis.

The debugger software we had access to is HiWave [42]. Breakpoints on SCI2 register reads were never triggered. We found that breakpoints cease functioning after the late bootloader entrypoint (0xFBADB1). The cause is unknown; possibilities include `BGND` instruction execution or resets triggered by the secondary ('left') processor. Attempts to reset both MCUs in tandem failed. Despite evidence of debug-aware firmware (e.g. watchdog feeding after `STOP`), we abandoned further investigation to focus on the firmware patch.

BDM allows reading S12X address spaces without interrupting execution (mostly [7]). We used HiWave's [42] scripting interface (Appendix I.0.10) to create hexdumps and convert

[7] BDM accesses global address space bypassing the MPU. It is non-intrusive only during free bus cycles, otherwise stealing cycles or stalling the CPU.

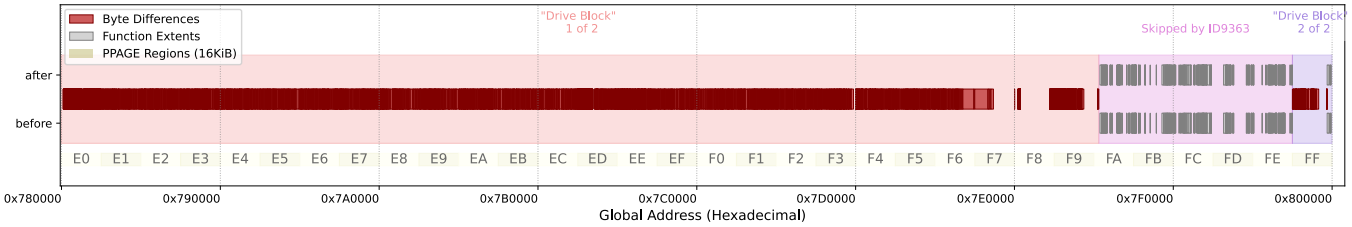


Figure 4: Visualization of what the IDA Pro auto-analysis achieved after creating entrypoints for each of the handlers in the default interrupt vector table. The lack of functions in all but the 'early' and 'late bootloader' regions illustrates that auto-analysis is not able to follow the exeution flow into the 'application' region.

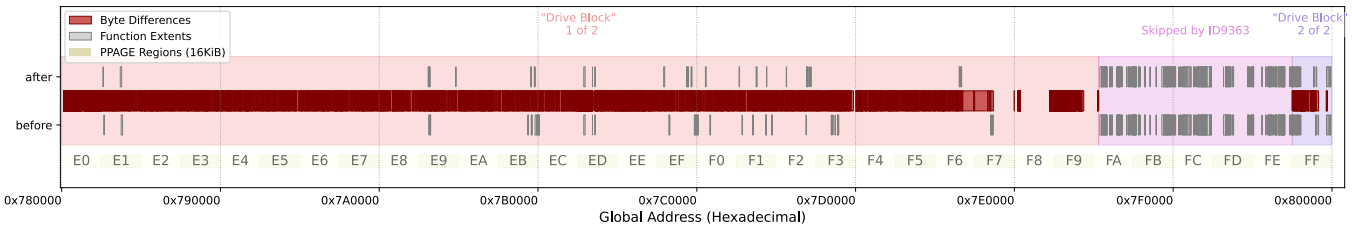


Figure 7: Visualization of what the IDA Pro auto-analysis achieved after creating entrypoints for each of the handlers in an interrupt vector table given by IVBR=0xF7. Compared with Figure 4, this figure shows several functions in the 'application' region; however, they represent a small fraction of the firmware's total application code, the upper bound of which is given by the byte differences. Similar to the prologs in Figure 3, it also shows correlated, left-shifted function extents which (again) suggest code deletion by the update; however, the low overall coverage limits our confidence.

them to .s19 files. This allowed us to read 'live' RAM and register contents without breakpoints and load them into IDA.

Polling registers revealed that the application uses an IVBR of 0xF7, unlike the initial bootloader's default 0xFF. The S12X has the capability to switch the location of its interrupt vector table [8]; although static analysis at the stage shown in Figure 4 missed the IVBR assignment, we had confirmed the application uses an interrupt vector table at 0xF710-0xF7F9.

We updated the IDA Pro database with RAM dumps and entrypoints for the new vectors (Figure 7). Analysis still covers much of the late bootloader, as the application frequently calls back into it (e.g., for programming mode).

We can now access application interrupt handlers. Target `2ec80` supports XGATE [9], PIT, CAN0, CAN4, Enhanced Capture Timer (ECT), SPI1, and SCI2. We focus solely on SCI2. The SCI2 ISR, `sub_C12B`, calls functions where IDA Pro analysis fails due to lack of 'register tracking' for S12X. For now, we manually resolved data-dependent `call` instructions (e.g., `call [-$2662,y]`) that IDA failed to handle. We also manually resolved jump tables, which IDA Pro does not support on S12X.

Then IDA Pro analysis revealed the application's J2497 message reception operation [10]. The details of the receiver architecture are captured in Appendix B; but the dataflow of received packets from the SCI2 peripheral to consumers is simpler: frames are built by appending to a receive buffer at 0x3BF5 and frame reception is signaled with the frame counter / semaphore at 0x3BF4. Figure G.1 illustrates this dataflow. This IDA analysis identified only the 'publish' side. No consumers of the frame counter or receive buffer were found. Firmware typically splits interrupt handling into a lightweight top-half and a main-thread bottom-half to minimize interrupt context cycles.

7 Binary Diffing

Interrupt context execution was explored, but the main execution thread remained invisible. Analysis failed to trace the late bootloader hand-off to the application [11]. Searching for function prolog byte sequences yielded excessive false positives. We observed function pointer arrays in 'Drive Blocks' 1 and 2. Indirect call instructions (e.g., `call [-$2662,y]`) read a 3-byte address (PPAGE offset + PPAGE value) from a resolved memory location. The compiler/linker used here always emits a trailing zero byte. Scanning for these patterns, filtering for valid PPAGE values and offsets (0x8000-0xC000), significantly reduced false positives. We further filtered out isolated function pointers, as they typically occur in arrays. See Appendix I.0.4 for the script used. Then analysis discovered the main-thread bottom-half function for J2497 message processing at sub_EEB989 (see Figure G.2).

The main thread processing in `sub_EEB989`, handles J2497 LAMP ON (0x0A), LAMP OFF (0x0B), and Active Trailer ABS Event (0x57) commands, ignoring payloads. It consumes contiguous 0x0A, 0x0B, or 0x57 messages. Messages with other MIDs are passed to `sub_F096F9`. J1587 messages can contain one or more PID payloads. This function splits J1587 messages into PID payloads and passes the payloads to `sub_F2A3E1`, which dispatches each to a handler from PFLASH tables starting at the 0xDC3D 'context structure' (see Appendix C for details). This data flow processing is illustrated in Figure G.2.

To analyze six firmware images (three pairs) with BinDiff [43], we automated IDB creation (see I.0.1), segment setup,

[8] Except for Power-on, Clock Monitor, and COP Watchdog reset vectors.

[9] We analyzed XGATE code by mapping PFLASH and RAM in IDA. XGATE handled only ECT interrupts (channels 0-7) and was irrelevant to J2497 reception, so analysis ceased. Ghidra would suffice here, but we had at this point invested in IDA-based automation.

[10] One firmware change is observable now but will be detailed in Section 8.

[11] Presumably due to at least a data-dependent call target but possibly due to this and other reasons. The result was good enough to identify the functions that consume the receive buffer 0x3BF5 and Frame Counter 0x3BF4: `sub_EEB98` and `sub_F096F9`; these are functions that check for LAMP messages and then process non-LAMP J1587 messages, respectively.

interrupt vector definition, and function search. We scripted the identification and marking of PID tables and handlers (Appendix I.0.5). We excluded the late bootloader to focus BinDiff on the application firmware.

Lack of 'register tracking' hindered automatic analysis, requiring manual resolution of indirect call targets. We implemented rudimentary concolic analysis to parse disassembly and symbolically track execution, substituting concrete values from PFLASH, DFLASH, or RAM when needed and where available. See Appendix I.0.2 for the script used. We also implemented basic jump table detection and creation by parsing disassembly and creating manual code xrefs (Appendix I.0.3) [12].

To ensure BinDiff fidelity, we aimed for zero disassembly errors. We found numerous disassembler errors in PPAGE 0xFD functions. Hiveplots [44] of IDA Pro xrefs revealed invalid data writes to PPAGE 0xFD from functions in other PPAGEs. These writes are invalid as PFLASH requires sector erasure. Runtime polling of the MMCCTL1 register confirmed that the application maps 0x4000 to fixed RAM, not PPAGE 0xFD. We adjusted the IDA database segment setup, as shown in Figure 8.

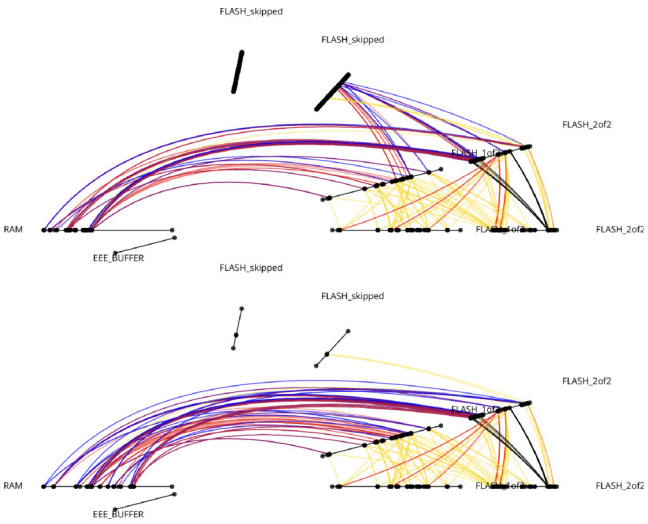


Figure 8: Hiveplot panel comparing IDA Pro analysis achieved before (above) and after (below) a change of the 0x4000 mapping from Flash to RAM. The cross-references (xrefs) are drawn as edges directed counter-clockwise (CCW); each region with executable code is doubled to show intra-region xrefs; the "Drive blocks" 1 and 2 are shown co-linearly and the skipped region is shown lifted above them; yellow edges are code xrefs, red edges are data reads, blue are data writes and black are interrupt vector references.

These steps achieved function coverage of 80% (1ec80), 60% (2ec80), and 75% (3ec80) of byte differences in PPAGEs E0-F9. We then automatically performed BinDiff on all three target pairs. Bindiff was able to support the S12X architecture and give reasonable insight; however, its shortcomings when applied to this firmware caused spurious matches e.g. it matched the SCI2 handler in a before image with the SCI3 handler in an after image.

We ultimately created a QBinDiff [45] analysis, the results of which are in Figure 9. The key to successful QBinDiff analysis was to 'anchor' (force a match) all the interrupt handlers. This yielded improvements over the Bindiff results. Less than 1% of discovered functions were low-confidence matches. Over 98% of functions remained unchanged, while ~110 - ~140 were modified or deleted (Table 2).

	1ec80	2ec80	3ec80
New	0	0	0
Deleted	104	127	123
Modified	11	12	13
Unchanged	1109	1100	1031
Low Confidence	12	13	14

Table 2: QBinDiff results summary of all three targets.

Three-way BinDiff analysis showed that matched functions dropped from 968 (pre-update) to 878 (post-update) across all 'Z' versions, corroborating function deletion. The patch makes very similar changes in all three cases: of the deleted functions, 87 were common to all three updates. The next section, 8, analyzes these changes. Fewer functions were deleted in the 1ec80 update because it retained J1587 Transport Protocol (TP) management functions which were deleted in the others; presumably, because they are used due to the J1708 interface which 1ec80 has and the other targets do not.

Term	Definition	BinDiff Terminology
Function Extents (matched, unchanged)	Functions present in both binaries with identical logic and structure.	Matched, Similarity = 1.0
Function Extents (matched, changed)	Functions present in both binaries but with some modifications to instructions or structure.	Matched, Similarity < 1.0, Confidence >= 0.95
Function Extents (unmatched)	Functions present in one binary but not the other (new or deleted).	Unmatched (Primary/Secondary)
< 95% confidence Function Extents	Functions that might be matched but with low confidence; ignored for high-assurance analysis.	Matched, Confidence < 0.95

Table 3: Table defining the categories of functions used in Figure 9 and the analysis presented in Sections 8 and 9.

[12] because the actual jump table operation "probably can't be scripted" according to IDA Pro support.

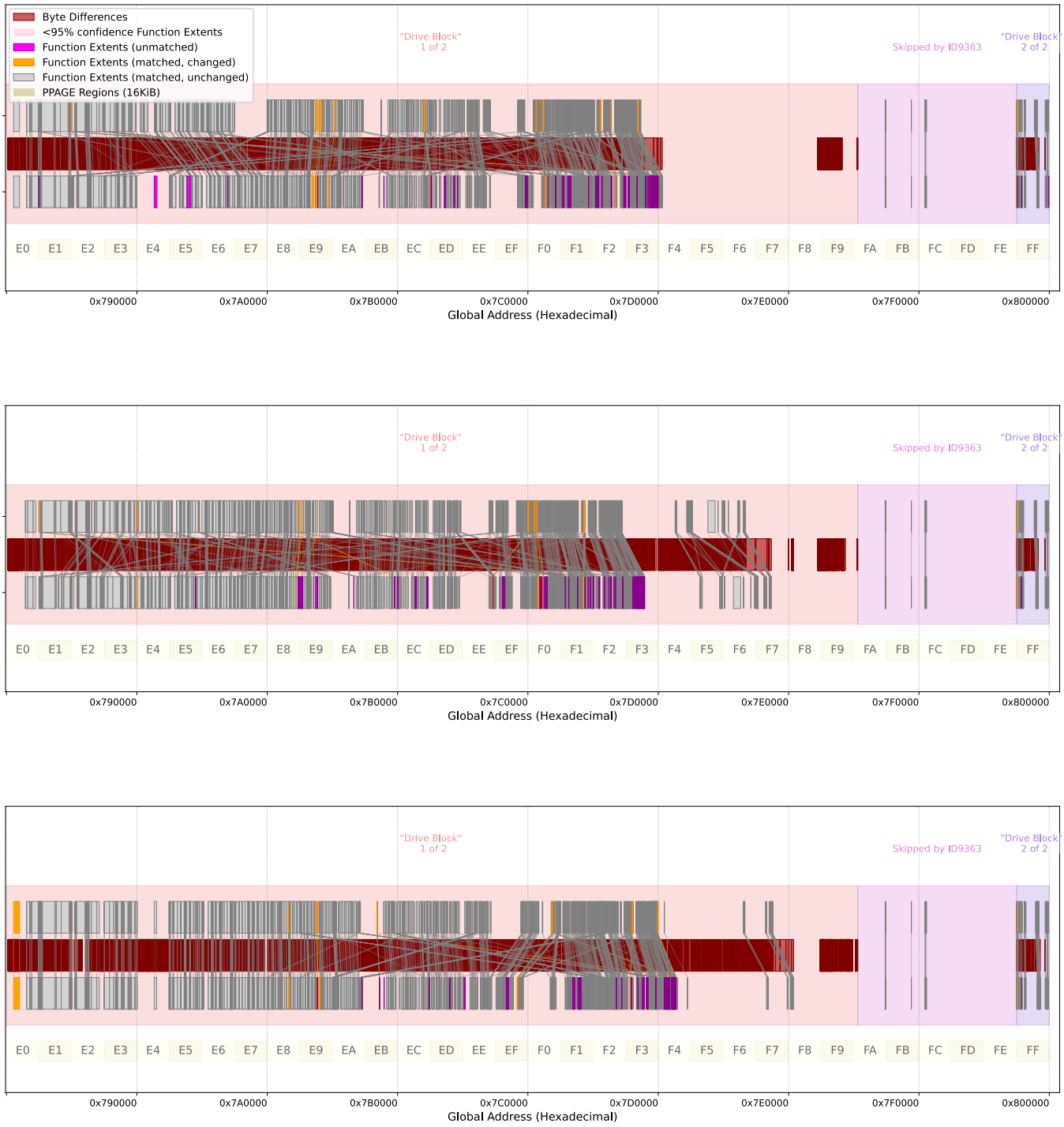


Figure 9: A visualization of the QBinDiff analyses of all three of the firmware updates (when limited to the APPLICATION parts of the PFLASH), overlaid on the byte differences of each. The varying complexity of the firmwares mentioned in the introduction is clearly illustrated here (compare to Table 1); target 3ec80 (top) shows the least complexity while 1ec80 (bottom) shows the most. Links between matched (by QBinDiff) functions are plotted across pre- and post-update. Despite noise from small/null function matches, the links show clear function shifting to lower addresses and numerous unmatched functions in the pre-update images (colored magenta), suggesting multiple function deletions. More conclusively: there are no (known) spurious matches and zero 'new' functions post-update: further suggesting multiple function deletions in the update.

8 Changes Made in Patch

Manual BinDiff analysis, validated by call tree analysis script (Appendix I.0.6), categorized all the deleted functions in all three firmwares as one of:

1. All J1587 PID processing present in the pre-update image (Appendix C), excluding LAMP and active ABS event processing.
2. SCI2 UART EDGE interrupt handling (Appendix B).
3. Secondary J1587 features, including the diagnostic code manager (`sub_E9858A`), J1587 Transport Protocol (TP) connection management (`sub_EF905B`), and TP timeouts (`sub_EC8116`). [13]

Modified functions primarily remove calls to these deleted functions:

1. In `sub_EEB989` (LAMP/ABS check), the call to `sub_F096F9` and subsequent PID processing is removed (Figure 10, EBB 0xEEBA20), consequently removing all linked PID processing tables and functions.
2. In `sub_C12B` (SCI2 ISR), the call to EDGE interrupt handling is removed (Figure 11), effectively eliminating the 'RXEDGIF Branch' (Figure G.3) and related setup/processing.

The bulk of the deletions were due to the removal of all PID processing on J2497 (see Table C.1 for the full list). The disassembly of all deleted functions (for target 2ec80) is provided in Appendix E.

9 Exploitability of Removed Functionality

We reviewed deleted functions (primarily PID handlers, Section 8) for vulnerabilities using static and dynamic analysis. Dynamic analysis was limited to fuzzing due to breakpoint limitations (Section 6). We identified several vulnerabilities in the deleted code.

The PID 0xC2 handler was exploitable for Denial-of-Service (DoS) and Remote Code Execution (RCE); both verified, see below. A DoS condition could be triggered by a J2497 message like `89C2FE`, where a large 'n' parameter (0xFE) caused a crash. For RCE, the handler processes a length parameter specifying the total size of data following the length parameter. The data is copied to a 0x30-byte stack buffer without length check. The return address is 0x31 bytes from this buffer, creating a classic buffer overflow [46] (see Listing of `sub_F1B623` in appendices) with one intervening byte: the count of a loop executed after the copy. While J2497 frame limits restrict direct attacker-controlled data to 18 bytes, the copy size is fully controllable.

[13] J1587 TP was not functional even before the update, confirmed with static analysis and testing.

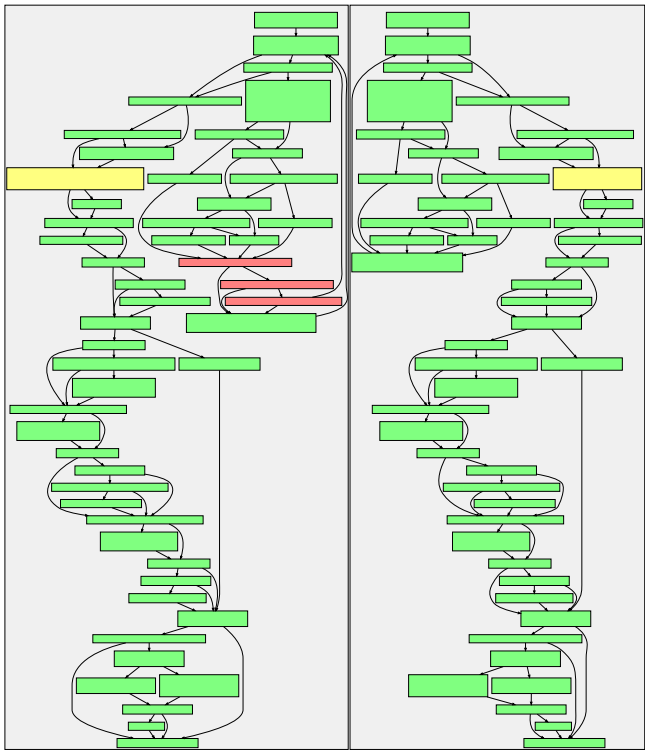


Figure 10: BinDiff of the LAMP etc check function before and after update. The modified EBB has a two-line change: remove a call to `sub_F096F` (Process PID payloads) and remove load of an argument to that call. The deleted EBBs were removed because with the call to PID processing removed there is no longer any need to handle LAMP and Active ABS MIDs at a higher priority.

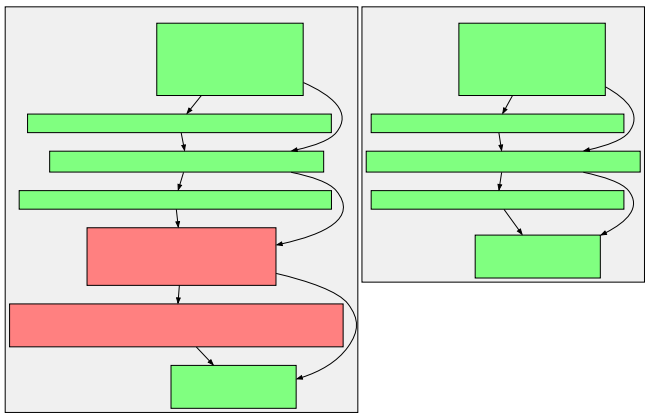


Figure 11: BinDiff of the SCI2 Handler before and after update. The deleted EBBs were responsible for servicing SCI2 EDGE interrupts.

Furthermore, the adjacent region is attacker-controlled, as J2497 frames are contiguous (with an injected frame-length byte) in a 0x54-byte FIFO buffer: an attacker can control the entire 0x53 byte copy by first sending a complete 0x52 byte buffer as a message (results in additional injected length byte and added checksum byte) starting with `89c280` which fills the buffer at 0x3BF5 (but does not get passed to higher application layers because it is longer than 21 bytes) then sending `89c280` by itself. The second message will get written to the same location as the previous `89c280` bytes in the receive buffer and will be passed to higher layers (with the remaining buffer in-tact). The wrinkle here is that any reception of other messages on the J2497 databus will interfere with the layout in the buffer: messages received before will move the start position of the first and/or second messages, reducing available buffer space and messages received in-between will move the start of the second message, removing the adjacency of the attacker-controlled buffer. A typical J2497 databus does contain a stream of LAMP messages and e.g. 0xC2 and other lower priority messages. There is a means to inhibit reception: during the development of the keyhole mitigation [47] the ideal standing sinusoid frequency was discovered to inhibit message reception. An attacker can transmit this sinusoid interference (to empty the buffer) and then transmit their exploit in a blanking of the sinusoid. They can also abuse the Intellon SSCP485 receiver feature where the pre-amble is superfluous to send the two body-only messages in a sequence quicker than 'normal' J2497 databus transmitters can achieve.

The RCE was confirmed on the 2ec80 and 3ec80 targets in a bench environment. This confirms that although the S12X has an MPU which could inhibit execution from data buffer and/or stack areas, it was not configured to do so. The following listing shows a simple proof of concept with NOP placeholders. The first code block is copied to the stack (as described above); the return address is overwritten with the address of the first code block in the receive FIFO 0x3BF5. Execution flows into the receive buffer original copy of the code block and jumps over the return address and intervening byte there to use an additional area for more code in the buffer.

```
c2_cmd          = [0x89, 0xc2,
                   0x30     # buffer size
                   + 1      # 2nd loop limit
                   + 3]     # ret addr on stack
payload_in_buff = ( 0x3bf5  # handy static address
                  + 1  # 1 for the injected length
                  + len(c2_cmd) )  # skip the c2 cmd

code_1 = list(bytes.fromhex(
  "A7"        # 3BF9: NOP          ; needed for jmp
  "1410"    + # 3BFA: SEI          ; Disable ints
  "A7" * 43 + # 3BFF: NOP NOP...
  "2004"      # 3C22: BRA   04     ; to CONTINUE
              #                    ; PC (3C29) + 4
))
sec_loop_lim = [0x00]  # skip handler's 2nd loop
ret_address  = [0xf2,  # PPAGE for later return
                       # to f2a462 via jump
                (payload_in_buff>>8) & 0xFF,
                (payload_in_buff   ) & 0xFF]

code_2 = list(bytes.fromhex(
              # CONTINUE:
  "A7" * 23 + # 3C2D: NOP NOP...
              #       ; Reduce this for resilience
  "10EF"      # CLI           ; Enable Interrupts
  "06A462"    # JMP   $A462   ; Return control
))

tosend = bytearray(c2_cmd + code_1
         + sec_loop_lim + ret_address + code_2)
you_send(tosend)  # fills the entire 0x3bf5 buffer
you_send(tosend[0:20])  # small enough to trigger
                        # reception of 89c2 command
```

This PoC uses the receive FIFO buffer at 0x3BF5 as the location for hosting the executable code and only queues one message (the `[0:20]` slice above). If other bytes are received before the `SEI` is executed they will be written into the buffer after 0x3BF5+20. The interfering sinusoid mentioned above can be reasonably effective at accomplishing this. A more robust approach could use the stack only (if the SP address in the 0xC2 handler were known [14]) or an approach could send multiple messages and skip over the injected length bytes. Long-running payloads will require feeding the watchdog peripheral and there are many examples of this in the firmware.

The PoC leaves 66 bytes of usable payload; this is enough to transmit a valid J1939 [36] CAN frame. The basic form of this is 85 bytes long using MOVB for data move. This can be reduced to 62 bytes by using STD 2,X+ instead. And it can be golfed into a 43 byte payload by abusing the stack for PULX; however, using PULX is not practical on the targets which use the XGATE coprocessor during their runtime (such as 2ec80) because the stack is shared. The payload can also be reduced to 39 bytes by re-using the firmware's CAN-sending function `sub_E9919A`. This was confirmed using the HSW12 assembler [41]. Also possible is misconfiguring the CAN transceiver to a bad bps which will crash that CAN bus segment and this a much smaller payload. This will silence all ECUs on the bus – until bus recovery fixes the problem, if at all. More is possible. As is always the case with RCE: anything is possible [48]. The space limitation here is also only a minor problem since an attacker can find many RAM areas to write longer payloads 'in pieces' as well.

Returning to the vulnerabilities in the other deleted code, the PID 0xED [15] handler can also crash the ECU. Sending a rapid sequence of messages with a large length parameter triggered this vulnerability. Bench and closed-track testing confirmed that greater than 10 repeated `89EDFE` messages trig-

[14] it was not for us. Because there was no dynamic analysis possible this PoC was developed iteratively in a crash/no-crash loop

[15] which is Trailer VIN PID, not the previously allocated PID: Entry Assist Control #1.

gered the crash (128x was used to be sure). The PID 0xED handler (`sub_F2BA46`) causes this by copying the PID payload (length-specified) into a static buffer at 0x5BDC without bounds checking (see Listing for `sub_F2BA46`s). This buffer has a 26-byte safe limit before overwriting critical structures at 0x5BF6 and 0x5BF8. These locations are used by `sub_F38241` for a conditional `memcpy` (`sub_E642`), with the condition seemingly controllable via the PID 0xB4 handler. Like the 0xC2 handler, the attacker controls copy size and can control the adjacent bytes in the receive FIFO buffer.

Theoretically, sequential 0xED and 0xB4 PID payloads (potentially in one J2497 message) could establish a write primitive. The write primitive pointer and data are 26 and 28 bytes from the 0xED handler's buffer start. Successful exploitation requires careful J2497 receive buffer grooming to overcome the 18-byte control limit. This write primitive makes RCE feasible, given unused RAM regions and fixed-address function pointers in the firmware [16].

Deleted SCI2 EDGE interrupt code theoretically presented a write-where primitive. Exploitation involves: 1) Triggering an EDGE interrupt and timeout to partially reset the receive buffer (by `sub_F1B035`), setting a buffer full flag (see Table G.1) but leaving the write offset (0x3C4B) un-reset. 2) Consuming a subsequent message clears the buffer full flag without resetting the offset, potentially exceeding the 0x54 buffer limit. 3) `sub_EC803E` (append data) contained a write-before-check flaw. Data was written to the receive buffer (0x3BF5) at the current offset (0x3C4B) before boundary validation. This allowed writes beyond the limit. 4) Positioning the write offset at +0x54 and sending a 4-byte frame allowed setting arbitrary upper bound (0x3C49) and current offset (0x3C4B) values. Subsequent data would be written to the new offset address.

The PID 0xC7 handler contained a hardcoded password check. Receiving the matching password disables traction control, as per J1587 specification [22]. Bench testing confirmed that message `ACC703004B42` satisfies this password check [17].

Component	Vulnerability	Impact
PID 0xC2	Buffer Overflow	DoS (Verified), RCE (Verified)
PID 0xED	Unbounded Copy	DoS (Verified), RCE (Theoretical)
PID 0xC7	Hardcoded Credential	Auth Bypass (Verified)
SCI2 EDGE	OOB Write	DoS (Theoretical), RCE (Theoretical)

Table 4: Summary of vulnerabilities in removed functionality.

Exploits of these RCE vulnerabilities (verified and theoretical) will depend on the specific 'Z' firmware version memory layout. The ID9363 updater contains 13 'Z' versions (covering a total ~450,000 units [12]); however, exploits of the DoS vulnerabilities and hardcoded credential will not. Testing shows they worked in all three tested 'Z' versions and likely affect all versions.

Reverse engineering is not required to discover all these vulnerabilities. A simple fuzzing script discovered the PID 0xC2 and 0xED crashes. These were verified on the bench and in-motion (Section 10). See Appendix I.0.7 for the script used.

10 Notes on In-Motion Vehicle Tests

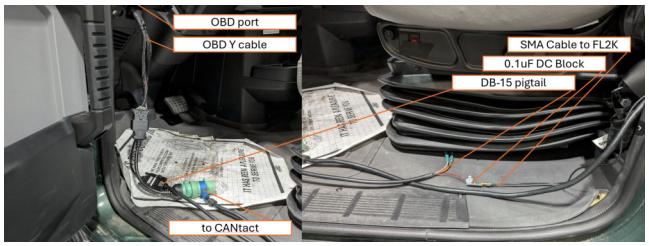


Figure 12: Right: a 10uF DC block resulting in AC-coupling of the FL2K SDR (not pictured) connected to a power amplifier (not pictured). Left: Connection of the same to the in-cab on board diagnostic (OBD) port.



Figure 13: Top Left: failure state of instrument cluster after any and all the ECU crashes described in Section 10 while still in-motion. Wide: in-cab picture of in-motion vehicle test with black bar redacting.

Onsite testing used a host-provided tractor that did not perfectly match the source vehicle of the vulnerable firmware but whose EC80 part number matched perfectly – because the host did not have any EC80s that were not already updated by ID9363. We manually flashed the target ECU with the vulnerable firmware via BDM. Then the target ECU was part-calibrated to the tractor using OEM tools, and baseline ABS functionality was confirmed via hard braking before further testing.

[16] It would also be possible to send CAN data with a write primitive and without RCE.

[17] `4B42` is ASCII 'KB', presumably for Knorr-Bremse, Bendix's parent company (since 2024 after a joint venture starting in 1993).

Testing focused on the crashes in the handlers for J1587 PIDs 0xC2 and 0xED, identified during bench testing. We injected J2497 signals using an FL2K Software Defined Radio (SDR) transmitter [49] [50] and power amplifier. Due to insufficient J2497 filtering at the diagnostic port, we were able to use a simple in-cab setup: AC-coupled via a 10uF capacitor to the diagnostic port's VBAT pin [18] (Figure 12). As shown in [4], this simulates wireless injection [6].

We conducted in-motion tests below 5 mph and at ~9 mph — fast enough to trigger ABS pulsing but safe for occupants. At 9 mph, after observing the ECU crash via CAN, the driver initiated a hard brake to check for ABS pulsing. In all cases, CAN traffic ceased. ECU recovery always required a battery disconnect [19]. The crash consistently caused loss of speedometer, steering assist, and shifting (Figure 13). The results are summarized in Table 5. The scripts used to trigger the 0xED and 0xC2 DoS vulnerabilities can be found in Appendices I.0.9, and I.0.8, respectively.

One difference emerged: in the PID 0xED case, the EC80 stopped streaming signal data but still responded to UDS TP requests. This was not observed in the PID 0xC2 case or during PID 0xED bench testing.

We repeated these in-motion tests on a second vehicle: owned by a fleet and from a different OEM than the previous. This EC80 was 'factory installed' and was not yet updated by 'ID9363'. During these tests we were able to test also the 0xC7 'backdoor' impacts. In this vehicle the impact of crashing the ABS controller did not include loss of speedometer nor steering assist nor shifting; but other impacts were the same. This was also true for the 0xC7 'backdoor'.

Vuln.	Target	Speed	UDS	Notes	ABS Disabled
0xED	1ec80	$<$ 5 mph	Yes	Loss of shift, steering assist, speedo. Multiple cluster faults	–
0xED	3ec80	$<$ 5 mph	No	Mulitple Cluster faults.	–
0xED	1ec80	$>$ 5 mph	Yes	Loss of shift, steering assist, speedo	**Yes**
0xED	3ec80	$>$ 5 mph	No	Mulitple Cluster faults.	**Yes**
0xC2	1ec80	$<$ 5 mph	No	Loss of speedo, steering assist	–
0xC2	3ec80	$<$ 5 mph	No	Mulitple Cluster faults.	–
0xC2	1ec80	$>$ 5 mph	No	Loss of shift, steering assist, speedo	**Yes**
0xC2	3ec80	$>$ 5 mph	No	Mulitple Cluster faults.	**Yes**
0xC7	3ec80	$>$ 5 mph	No	Multiple Cluster faults including "Dyno Mode". Loss of shifting in Reverse	**Yes**

Table 5: In-motion test results (In all cases streaming data stopped).

11 Random Noise Sufficiency

The EDGE SCI2 interrupt vulnerability could obviously have been triggered by noise: constant framing errors and/or parity errors could cause false 0x00 bytes to fill the receive buffer and edge interrupts firing coincident with these could cause the partial reset and eventual overwrite of pointers. It is not as clear for the PID processing handlers' vulnerabilities.

As was mentioned in Appendix B, the only receiver error checking is if the checksum is consistent. These checksums are the two's complement of the sum of all bits in the message. The checksum is not able to detect many-bit errors, and e.g. fails to detect 'symmetric' errors (pairs of bit flips). It is easy to imagine the checksum failing to prevent reception of invalid messages in noisy conditions.

There are not any established models of bit errors or of spurious data reception of the Intellon SSCP485 [24] that we are aware of. However, from previous research [53] it is known to be possible for a message on the bus to have several of its bits flipped and/or be truncated on reception in the presence of noise.

It is very typical of trailer brake controllers to stream PID 0xC2 and so it could be imagined that bit flips of an existing 0xC2 message could cause a large value of 'n' and hence the same DoS crash we observed in Section 10.

Trailer brake controllers do not typically stream 0xED messages [20]. Nor do they stream 0xC7 messages. It is, however, possible that the 0xC2 (or other streaming messages) get bit flipped to 0xED or 0xC7 and truncated to the right length. This could not be reasonably described as a likely possibility though.

12 Related Work

Binary analysis of bare-metal firmware often faces unique constraints. While [54] uses emulation, we employ minimum-viable concolic analysis where emulation was unavailable. We address BinDiff's small-function mismatch issues [55] by applying QBinDiff [56] with manual interrupt handler anchoring to enable useful results. Furthermore, our findings confirm that silent security patching [57] is occuring also in heavy vehicle firmware.

[18] this was confirmed by observing Trailer ABS fault dash light status and 0xF001 PGN with `python -m can.viewer -i cantact -c 0 -b 500000 --filter=00F00100:00FFFF00` (using python-can [51] and CANtact [52]).

[19] This is a relevant condition because some tractors are configured without a battery disconnect switch, requiring a manual removal of the battery leads or a manual cycling of a fuse to recover the ECU.

[20] although they can send them on power up and on request at power-up of the tractor

Seminal work [58] identified "standard" vulnerabilities like buffer overflows that, we show, persist in modern components. We extend J1708 security research [59] to J2497 and apply the wireless write attack [53] to tractor equipment. Complementing J1939 diagnostic research [60], we demonstrate that mission-time (as distinct from diagnostics) data processing defects in J2497 (as opposed to J1939) directly impact tractor safety.

13 Generalizable Contributions

Our work provides several contributions that generalize across different architectures and target devices. First, we demonstrate that hiveplots can be effectively used to survey the current disassembly of a binary. This visualization technique is particularly useful for identifying 'configuration' issues and spotting segmented areas in bare-metal firmware images. Second, we found that the use of manual anchoring in QBinDiff for each of the interrupt handlers is a critical step in achieving useful and accurate binary diff results in its application to bare metal firmware. Finally, our approach of performing live RAM dumping is generally applicable to any case where a debugger is not fully functional. This technique is critical for discovering dynamically resolved functions, particularly when function pointers or their offsets are stored exclusively in RAM.

14 Conclusions

This work demonstrates binary differential analysis on legacy automotive architectures, using heuristics and symbolic analysis to overcome tooling limitations. Our analysis reveals several vulnerabilities were patched by the ID9363 update.

While the vendor emphasized safety compliance regarding random noise, our analysis confirms a broader security impact. Random line noise is unlikely to generate the valid J1587 frames and frame sequences required to trigger these vulnerabilities. Specifically, the patch removes the tractor ECU's J1587 parsing stack (except LAMP/ABS event handling), eliminating vulnerable handlers for PIDs 0xC2, 0xED, and 0xC7.

We classify the removed functionality as critically flawed, containing unauthenticated memory corruption (PIDs 0xC2, 0xED) and hardcoded weak authentication (PID 0xC7). In-motion testing validated the severity: exploiting the PID 0xC2 buffer overflow caused immediate ABS Denial of Service (DoS), physically manifesting as loss of steering assist, speedometer, shifting and ABS pulsing. The PID 0xED handler induced the same DoS via memory corruption and the 0xC7 handler also via abuse of weak authenticated functionality. Bench testing confirmed RCE via the memory corruption in the PID 0xC2 handler.

The ID9363 update mitigates risks from attackers with wireless adjacency or other forms of J2497 network access, improving the compliance to requirements NGW-S-001 through -005 [29]. Bendix's proactive recall and firmware update represent a commendable and responsible action. This is particularly important as users are believed to prioritize safety recalls over security patches. However, the lack of distinct CVE assignments for these flaws may obscure the patch's security criticality. What is recommended – by ISO/IEC 29147 [61] and CISA [62] – is that vulnerabilities which are patched should be communicated to users so that they can make their own informed risk calculations.

The features removed align with the security architecture for tractor J2497 reception [21] detailed in all of:

1. the ATA TMC position paper on next generation tractor trailer interfaces [63]: "Pertaining to PLC communications as described in SAE J2497: only the MID 10 and 11 lamp messages, MID 125 J2497 identification, and MID 87 active ABS event shall be permitted on new tractor and trailer equipment…"
2. the CISA mitigations in advisory ICSA-25-021-03 [64]: "To most effectively mitigate general vulnerabilities of the powerline communication, any [tractors] utilizing J2497 technology should disable all features where possible, except for backwards-compatibility with LAMP ON detection only."
3. the latest version of SAE J2497 [1]: "For tractor PLC units, it is recommended that the software implements only reception of the LAMP ON and OFF messages … All other reception of messages should be performed on other network types."

Given the potential for a) malicious J2497 signal injection and b) compromised J2497-capable telematics units, we strongly advocate that all tractor J2497-receiving equipment, especially brake controllers, adopt the security-conscious approach demonstrated by the EC80 update.

Acknowledgments

We would like to thank our colleagues at the National Motor Freight Traffic Association Inc. whose expertise and dedication have been invaluable to the completion of this research. Particularly Anne Zachos for the continuous assistance in analysis and onsite testing.

We would like to thank AIS for access to their Class 8 vehicle multiple times during this research. Thank you to Hannah Silva and Jesse Norton for sharing S12X development materials and EC80 experience. Thank you to Chris York for the

[21] for tractor J2497 transmission the same sources recommended that the tractor mitigate attacks on older trailer equipment; this patch does not appear to add any such mitigations.

trailhead. Thank you to Jonatan Mars for critical support at a critical time. We would also like to thank many industry engineers for their support of this research – the rest of whom would prefer not to be named. Without the support of the engineers at these vehicle OEMs and suppliers this work would not be possible. Last but not least, this work would also not be possible without the support of the member fleets of the NMFTA Inc.

We used various large language models (Gemini [65] 2, 2.5-pro, 3-pro) for tasks such as: drafting Python code and Mermaid diagrams. We acknowledge the open-source tools used: Pandoc [66], Matplotlib [67], Hiveplotlib [68], Zynamics BinDiff [43], Quarkslab QBinDiff [45], Quokka [69], Python 3 [70], python-can [51], py-hv-networks [71], RP1210 python [72], Osmocom FL2K [50], Linkerscope [73], Mermaid [74], and Graphviz [75].

Open Science - Availability

The complete code listings of scripts mentioned in the paper are available in the Artifact Appendix of this paper https://doi.org/10.5281/zenodo.19664811. With the exception of the QBinDiff analysis which will be made available later as 'howto' document pull request to the Quarkslab QBinDiff repository docs.

None of the firmware images used in the analysis will be made available. The reverse engineering of this work focused only on the machine code patched by the firmware update. This work makes no claims on the presence nor absence of vulnerabilities in the remainder of the firmware domains; hence, there is insufficient guarantee of safety of sharing the firmwares. In lieu of the firmware, we provide the following in the Artifact Appendix:

1. disassembly listings (with instruction bytes included) for the deleted and select modified functions only
2. PID Handler and Context Tables deleted in the update
3. an .s19 file containing only the above plus the interrupt vectors for the application

The ID9363 updater executable will not be made available because it contains copies of all the new firmware images.

The ID9363 firmware update CAN logs will not be made available because they contain the entire updated firmware in the clear as well as seed-key exchange pairs which cannot be guaranteed to be insufficient to reconstruct the seed-key routines in closed-form.

Ethical Considerations

This research prioritizes disclosure and minimizes potential harm. Prior to commencing the technical analysis, in early 2025, the manufacturer of the brake ECU was contacted and informed of our intent to conduct research on their firmware update for the safety recall and requested to collaborate. Again, as the analysis neared completion, the manufacturer was contacted in an attempt to provide them with an overview of our findings. Two of the three OEMs have received the results in this paper. NHTSA and Transport Canada have also been informed of this research and results.

Crucially, this paper discusses only functions and vulnerabilities addressed by the manufacturer's safety recall. Only deleted function code is shared. Limited side-by-side code modifications are shared to limit disclosure of code in the field. The PoC provided can be used to confirm RCE but does not disclose the details of CAN bus or other physical control by the ECU. The updated firmware has been deployed on ~310,000 units [12] at the time of this writing. These vulnerabilities are confirmed absent in updated firmware versions Z300822, Z302578, Z302579 (one for each affected OEM). Thus, disclosing these vulnerabilities introduces no new risks. By focusing on already-patched issues, this research aims to contribute to the broader understanding of automotive security without inadvertently exposing users to unmitigated risks.

Finally, the in-vehicle testing was conducted in a controlled environment, utilizing a closed test track for in-motion vehicle testing. All testing was performed with the explicit cooperation and support of an OEM, ensuring that safety protocols were rigorously followed and that no public roads or operational vehicles were subjected to unverified or potentially hazardous conditions.

References

[1] SAE Truck and Bus Control and Communications Network Committee, "J2497 power line carrier communications for commercial vehicles." Warrendale, PA, 2026.

[2] *Federal motor vehicle safety standard no. 121, air brake systems*. Washington, D.C.: National Highway Traffic Safety Administration.

[3] MITRE Corporation, "CVE-2020-14514." 2020. Accessed: Nov. 2025. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-14514

[4] B. Gardiner, "Disclosure of confirmed remote write to J2497 aka PLC4TRUCKS," *NMFTA, Alexandria, VA, Letter, March*, 2022.

[5] National Motor Freight Traffic Association, "Actionable mitigations options v9." 2022. Accessed: Nov. 2025. [Online]. Available: https://nmfta.org/wp-content/media/2022/11/Actionable_Mitigations_Options_v9_DIST.pdf

[6] MITRE Corporation, "CVE-2022-26131." 2022. Accessed: Nov. 2025. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-26131

[7] Bendix Commercial Vehicle Systems LLC, "24E086 chronology." 2024. Accessed: Nov. 2025. [Online]. Available: https://static.nhtsa.gov/odi/rcl/2024/RMISC-24E086-5355.pdf

[8] *ISO 14229: Road vehicles – unified diagnostic services (UDS)*. Geneva, Switzerland: International Organization for Standardization, 2020.

[9] PACCAR Incorporated, "Safety recall report 24V-915." 2024. Accessed: Nov. 2025. [Online]. Available: https://static.nhtsa.gov/odi/rcl/2024/RCLRPT-24V915-6438.PDF

[10] Navistar, Inc., "Safety recall report 24V-818." 2024. Accessed: Nov. 2025. [Online]. Available: https://static.nhtsa.gov/odi/rcl/2024/RCLRPT-24V818-4283.PDF

[11] Volvo Trucks North America, "Safety recall report 24V-790." 2024. Accessed: Nov. 2025. [Online]. Available: https://static.nhtsa.gov/odi/rcl/2024/RCLRPT-24V790-3386.PDF

[12] National Highway Traffic Safety Administration, "NHTSA recalls by manufacturer." Accessed: Nov. 2025. [Online]. Available: https://datahub.transportation.gov/Automobiles/NHTSA-Recalls-by-Manufacturer/mu99-t4jn

[13] National Highway Traffic Safety Administration, "Technical service bulletin 10194446." 2024. Accessed: Nov. 2025. [Online]. Available: https://dot.report/bulletins/10194446

[14] National Highway Traffic Safety Administration, "Technical service bulletin 10176745." 2024. Accessed: Nov. 2025. [Online]. Available: https://dot.report/bulletins/10176745

[15] National Highway Traffic Safety Administration, "Technical service bulletin 10222229." 2024. Accessed: Nov. 2025. [Online]. Available: https://dot.report/bulletins/10222229

[16] National Motor Freight Traffic Association, "Bendix EC80 recall: Safety and security implications." 2025. Accessed: Jan. 2026. [Online]. Available: https://nmfta.org/bendix-ec80-recall-safety-and-security-implications/

[17] Bendix Commercial Vehicle Systems LLC, "Safety recall report 25E-073." 2025. Accessed: Nov. 2025. [Online]. Available: https://static.nhtsa.gov/odi/rcl/2025/RCLRPT-25E073-3346.pdf

[18] Land Line Media, "Defective bendix ECUs have prompted recall of nearly half a million trucks with latest paccar recall." 2025. Accessed: Nov. 2025. [Online]. Available: https://landline.media/defective-bendix-ecus-have-prompted-recall-of-nearly-half-a-million-trucks-with-latest-paccar-recall/

[19] Bendix Commercial Vehicle Systems LLC, "Technical bulletin TCH-27-007." 2024. Accessed: Nov. 2025. [Online]. Available: https://www.bendix.com/media/services-and-support/product-action-center-pdfs/tch_27_007_en_000.pdf

[20] Bendix Commercial Vehicle Systems LLC, "Technical bulletin TCH-27-006." 2024. Accessed: Nov. 2025. [Online]. Available: https://www.bendix.com/media/services-and-support/product-action-center-pdfs/tch_27_006_en_000.pdf

[21] Bendix Commercial Vehicle Systems LLC, "Technical bulletin TCH-27-008." 2025. Accessed: Nov. 2025. [Online]. Available: https://www.bendix.com/media/services-and-support/product-action-center-pdfs/tch-27-008_en_000.pdf

[22] *J1587: Electronic data interchange between microcomputer systems in heavy-duty vehicle applications*. Warrendale, PA: SAE International, 2013.

[23] *J1708: Serial data communications between microcomputer systems in heavy-duty vehicle applications*. Warrendale, PA: SAE International, 2016.

[24] Intellon Corporation, *SSC P485 PL transceiver IC data sheet*. 1997.

[25] S. Behere, X. Zhang, V. Izosimov, and M. Törngren, "A functional brake architecture for autonomous heavy commercial vehicles." 2016. Available: https://legacy.sae.org/publications/technical-papers/content/2016-01-0134/

[26] *ISO 26262: Road vehicles – functional safety*. Geneva, Switzerland: International Organization for Standardization, 2018.

[27] ZF Friedrichshafen AG, "mBSP XBS factsheet." 2024. Accessed: Nov. 2025. [Online]. Available: https://www.zf.com/public/org/ZF_CVS_mBSP_XBS_Factsheet_EN_296135.pdf

[28] Vehicle Cybersecurity Working Group (VCRWG), National Motor Freight Traffic Association, "NMFTA vehicle cybersecurity requirements." 2024. Accessed: Nov. 2025. [Online]. Available: https://github.com/nmfta-repo/nmfta-vehicle_cybersecurity_requirements

[29] B. Gardiner, J. Maag, and K. Tindell, "Security requirements for vehicle security gateways," SAE International, 2024-01-2806, 2024. Accessed: Nov. 2025. [Online]. Available: https://www.sae.org/papers/security-requirements-vehicle-security-gateways-2024-01-2806

[30] NXP Semiconductors, “MC9S12XEQ512 data sheet.” Accessed: Nov. 2025. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/MC9S12XEP100.pdf

[31] NXP Semiconductors, “MC9S12XE family reference manual.” Accessed: Nov. 2025. [Online]. Available: https://www.nxp.com/docs/en/reference-manual/MC9S12XERM.pdf

[32] ELDB, “XPROG-box.” Accessed: Nov. 2025. [Online]. Available: https://www.eldb.eu/

[33] PEmicro, “PROGS12Z flash programmer software.” Accessed: Nov. 2025. [Online]. Available: https://www.pemicro.com/

[34] Motorola, “Motorola s-record description (PDF).” Accessed: Nov. 2025. [Online]. Available: https://deramp.com/downloads/mfe_archive/060-Standards%20and%20Specifications/Hex%20Data%20Formats/Motorola%20S%20Record.pdf

[35] DARPA, “Assured micropatching (AMP).” Accessed: Nov. 2025. [Online]. Available: https://www.darpa.mil/program/assured-micropatching

[36] *J1939: Serial control and communications heavy duty vehicle network*. Warrendale, PA: SAE International, 2018.

[37] C. Miller and C. Valasek, “Adventures in automotive networks and control units,” IOActive, 2014. Accessed: Nov. 2025. [Online]. Available: https://www.ioactive.com/wp-content/uploads/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf

[38] Pulse Security, “Reversing the ducati 696 ECU part 2.” 2022. Accessed: Nov. 2025. [Online]. Available: https://pulsesecurity.co.nz/articles/ducati-696-part2

[39] National Security Agency, *Ghidra*. (2025). Available: https://ghidra-sre.org/

[40] Hex-Rays, *IDA pro*. (2026). Available: https://hex-rays.com/ida-pro/

[41] hotwolf, *HSW12*. Accessed: Nov. 2025. [Online]. Available: https://github.com/hotwolf/HSW12

[42] NXP Semiconductors, *HiWave debugger*. 2010.

[43] Zynamics, *BinDiff*. Available: https://www.zynamics.com/bindiff.html

[44] M. Krzywinski, I. Birol, S. J. Jones, and M. A. Marra, “Hive plots—rational approach to visualizing networks,” *Briefings in bioinformatics*, vol. 13, no. 5, pp. 627–644, 2011.

[45] R. Cohen, R. David, R. Mori, F. Yger, and F. Rossi, “Improving binary diffing through similarity and matching intricacies,” in *Proc. Of the 6th conference on artificial intelligence for defense*, 2024.

[46] A. One, “Smashing the stack for fun and profit,” *Phrack Magazine*, vol. 7, no. 49, 1996, Available: http://phrack.org/issues/49/14.html

[47] National Motor Freight Traffic Association, “j2497-keyhole.” Accessed: Nov. 2025. [Online]. Available: https://github.com/nmfta-repo/j2497-keyhole

[48] Masterjun, “Super mario world "total control" TAS.” 2014. Accessed: Nov. 2025. [Online]. Available: https://tasvideos.org/2513M

[49] T. Yapo, “FL2K experiments.” Accessed: Nov. 2025. [Online]. Available: https://hackaday.io/project/164346-fl2k-sdr

[50] Osmocom, “Osmo-FL2k project.” Accessed: Nov. 2025. [Online]. Available: https://osmocom.org/projects/osmo-fl2k

[51] python-can Developers, *Python-can*. (2024). Available: https://python-can.readthedocs.io/

[52] E. Evenchick, “CANtact.” Accessed: Nov. 2025. [Online]. Available: https://cantact.io/

[53] B. Gardiner, “Mitigating PLC4TRUCKS remote write,” in *Proceedings of the 9th escar USA conference*, Ypsilanti, MI: escar, 2022. Available: https://escar.info/downloads

[54] R. Tsang, D. Joseph, S. Salehi, *et al.*, “FFXE: Dynamic control flow graph recovery for embedded firmware binaries,” in *33rd USENIX security symposium (USENIX security 24)*, 2024, pp. 5573–5590.

[55] A. Marcelli, M. Graziano, X. Ugarte-Pedrero, Y. Fratantonio, D. Balzarotti, and A. Francillon, “How machine learning is solving the binary function similarity problem,” in *USENIX security symposium*, 2022.

[56] E. Mengin and R. David, “QBinDiff : Un outil de diffing binaire flexible et modulaire,” in *SSTIC 2020 (symposium sur la sécurité de l’information et des communications)*, 2020.

[57] J. Ruge, J. Classen, F. Gringoli, and M. Hollick, “Frankenstein: Advanced wireless fuzzing to exploit new bluetooth escalation targets,” in *USENIX security symposium*, 2020.

[58] K. Koscher *et al.*, “Experimental analysis of a modern automobile,” in *2010 IEEE symposium on security and privacy*, 2010.

[59] B. Hass, Y. Burakova, L. Millar, and A. Weimerskirch, “Truck hacking: An experimental analysis of the SAE J1939 standard,” in *10th USENIX workshop on offensive technologies (WOOT 16)*, Austin, TX: USENIX Association, 2016.

[60] R. Chatterjee, C. Green, and J. Daily, “Exploiting diagnostic protocol vulnerabilities on embedded networks in commercial vehicles,” in *Symposium on vehicles security and privacy (VehicleSec)*, 2024.

[61] *ISO/IEC 29147: Information technology – security techniques – vulnerability disclosure*. Geneva, Switzerland: International Organization for Standardization, 2018.

[62] Cybersecurity and Infrastructure Security Agency, "Shifting the balance of cybersecurity risk: Principles and approaches for security-by-design and -default," 2023. Accessed: Nov. 2025. [Online]. Available: https://www.cisa.gov/resources-tools/resources/shifting-balance-cybersecurity-risk-principles-and-approaches-security-design-and-default

[63] Technology & Maintenance Council, "Position paper 2024-3: Next generation tractor-trailer technical needs," American Trucking Associations, 2024. Accessed: Nov. 2025. [Online]. Available: https://tmc.trucking.org/sites/default/files/TMC_PP-2024_3_NEXTGEN_TRACTOR_TRAILER_TECHNICAL_NEEDS%20.pdf

[64] Cybersecurity and Infrastructure Security Agency, "ICS advisory (ICSA-25-021-03) bendix EC-80." 2025. Accessed: Nov. 2025. [Online]. Available: https://www.cisa.gov/news-events/ics-advisories/icsa-25-021-03

[65] Google DeepMind, "Gemini: A family of highly capable multimodal models." 2023. Accessed: Nov. 2025. [Online]. Available: https://deepmind.google.com/technologies/gemini/

[66] J. MacFarlane, *Pandoc*. (2026). Available: https://pandoc.org/

[67] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[68] Hiveplotlib Developers, *Hiveplotlib*. (2025). Available: https://github.com/hiveplotlib/hiveplotlib

[69] Quarkslab, *Quokka*. (2024). Available: https://github.com/quarkslab/quokka

[70] Python Software Foundation, *Python programming language*. (2026). Available: https://www.python.org/

[71] TruckHacking organization (originally by Haystack), *Py-hv-networks*. (first published 2016). Available: https://github.com/TruckHacking/py-hv-networks

[72] dfieschko, *RP1210*. (2019). Available: https://github.com/dfieschko/RP1210

[73] LinkerScope Developers, *LinkerScope*.

[74] K. Sveidqvist, *Mermaid*. (2026). Available: https://mermaid.js.org/

[75] Graphviz Authors, *Graphviz*. (2026). Available: https://graphviz.org/

[76] Wireshark Foundation, *Wireshark*. Available: https://www.wireshark.org/

A Appendix: ID9363 Update Process Further Details

The update process is executed by the ID9363 updater executable, where we captured the J1939 [36] traffic sent between it and the target ECU. The update process uses UDS [8] on J1939 [36]; in all cases we observed the traffic was unicast, default priority, between the vehicle diagnostic adapter at address 0xF9 and the brake controller at 0x0B: resulting in the two extended (29-bit) CAN IDs 0x18DA0BF9 and 0x18DAF90B. The CAN traffic was filtered on those two IDs and processed with Wireshark [76] to reconstruct the ISO-TP multi frame messages used by UDS and then these were analyzed to reconstruct the ID9363 update process, a simplified summary of which is reproduced in Table F.1 (the remainder of this appendix is available in appendix F at https://doi.org/10.5281/zenodo.19664811).

B Appendix: J2497 Reception Architecture of EC80

The reception is driven by events from the SCI2 and PIT hardware peripherals as illustrated in Figure G.3, PIT being used to manage timeouts. SCI2 is a UART that supports several interrupt event types and the receiver state machine makes use of 'data ready' (RDRF), IDLE, and edge (RXEDGIF) events. The entire state machine is distributed over several functions which all access address 0x3C52, a set of control bitfields, summarized in Table G.1 (the remainder of this appendix is available in appendix G at https://doi.org/10.5281/zenodo.19664811).

C Appendix: J1587 PID Processing

Table C.1 shows all the PID processing which are removed by the firmware update. These were discovered and compiled by walking PID handler data structures found in the firmware and detailed in the remainder of this appendix – which is available in appendix H at https://doi.org/10.5281/zenodo.19664811

D Appendix: Code Listings

The remainder of this appendix is available in appendix I at https://doi.org/10.5281/zenodo.19664811.

E Appendix: Disassembly Listings

The remainder of this appendix is available in appendix J at https://doi.org/10.5281/zenodo.19664811.

J1708 message	PID / PID Request Description	Data Input Size
mm0031 / mm803188	Request for 0x31 49 ABS Control Status	one byte data (broadcast req) / zero bytes data (unicast req)
mm003E / mm803E88	Request for 0x3E 62 Retarder Inhibit Status	one byte data / zero bytes data
mm0054 / mm805488	Request for 0x54 84 Road Speed	one byte data / zero bytes data
mm0097 / mm809788	Request for 0x97 151 TC Control State	one byte data / zero bytes data
mm009E / mm809E88	Request for 0x9E 158 Battery Potential (V), Switched	one byte data / zero bytes data
mm00A8 / mm80A888	Request for 0xA8 168 Battery Potential (V)	one byte data / zero bytes data
mm00C2 / mm80C288	Request for 0xC2 194 Transmitter System Diagnostic Code and Occurrence Count Table	one byte data / zero bytes data
mm00C4 / mm80C488	Request for 0xC4 196 Diagnostic Data/Count Clear Response	one byte data / zero bytes data
mm00C7 / mm80C788	Request for 0xC7 199 Traction Control Disable State	one byte data / zero bytes data
mm00D1 / mm80D188	Request for 0xD1 209 ABS Control Status, Trailer	one byte data / zero bytes data
mm00D6 / mm80D688	Request for 0xD6 214 Vehicle Wheel Speeds	one byte data / zero bytes data
mm00E9 / mm80E988	Request for 0xE9 233 Unit Number (Power Unit)	one byte data / zero bytes data
mm00EA / mm80EA88	Request for 0xEA 234 Software Identification	one byte data / zero bytes data
mm00ED / mm80ED88	Request for 0xED 237 Vehicle Identification Number	one byte data / zero bytes data
mm00F3 / mm80F388	Request for 0xF3 243 Component Identification	one byte data / zero bytes data
mm2A	42 Pressure Switch Status	1 byte data
mm46	70 Parking Brake Switch Status	1 byte data
mm54	84 Road Speed	1 byte data
mm75	117 Brake Primary Pressure	1 byte data
mm76	118 Brake Secondary Pressure	1 byte data
mmA9	169 Cargo Ambient Temperature	2 bytes data
mmB4	180 Trailer Weight	2 bytes data
mmC2	194 Transmitter System Diagnostic Code and Occurrence Count Table	variable bytes data. byte count, followed by sets of diagnostic data
mmC303	195 Diagnostic Data Request / Clear	3 bytes data
mmC7	199 Traction Control Disable State	variable length data. byte count, state flags, ASCII 'access code' of 0-15 characters selected by the manufacturer ...
mmD1	209 ABS Control Status, Trailer	variable bytes data, up to 3x5 bytes for 5 trailers byte count, followed by the status bytes
mmED	237 Vehicle Identification Number	variable bytes data. byte count, followed by the VIN ASCII
mmF5	245 Total Vehicle Distance	4 bytes data
mmF7	257 Total Engine Hours	4 bytes data
mmFE88C5 / mmFE88C6	254 Proprietary DLE	permitted lengths of only 0xC5 and 0xC6
mmFF73	115 Trailer Pneumatic Supply Line Pressure	one byte data
mmFF7B	123 Door Status	one byte data

Table C.1: J1587 PIDs supported by the before update firmware. 'mm' refers to any MID value except 0A, 0B, or 57, which are handled earlier in processing.

ARTIFACT EVALUATED
USENIX
AVAILABLE

Artifact Appendix
Reversing a Brake ECU Firmware Update

Ben Gardiner, National Motor Freight Traffic Association Inc.

A Artifact Appendix

A.1 Abstract

The purpose of the artifact(s) is to share IDA Pro / python scripts which we hope will be useful to others and also to support the claim that the firmware update to remediate the recall was in fact a security patch. There are some artifacts which we cannot safely release: i.e. the firmware files. The before and after firmware files of the ECUs would be required for functional and/or reproducible badges. And these firmware files will not be released because – as mentioned in the paper – we cannot make any guarantees on the absence of other vulnerabilities in neither the before nor after ECU firmwares. For this reason and the fact that the scripts require a paid license to IDA Pro 9.1, we request only confirmation of availability of the artifacts.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifacts submitted contains documentation and disassembly of vulnerabilities that have been patched in the wild through a safety recall. The provided disassembly is limited to the functions removed during the update to minimize the risk of exposing code from the updated, currently deployed firmware. All provided scripts are for offline static analysis or safe, controlled test-bench use.

A.2.2 How to access

The artifacts are submitted alongside this appendix document.

- **IDA Scripts: `ida-*.py`** Reverse engineering scripts (IDA Python). These are standalone scripts produced by inlining functions from an internal EC80-specific analysis library (which is not released in full to prevent leaking potentially attacker-beneficial information):
  - `ida-pflash-s12x-setup.py`: Sets up the memory map for the S12X processor in IDA Pro, handling the PFLASH and PPAGE banking.
  - `ida-concolic-s12x-resolve.py`: Performs a basic concolic analysis to resolve indirect calls.
  - `ida-jumptables-s12x-resolve.py`: Detects and creates jump tables by parsing the disassembly and creating manual code xrefs.
  - `ida-funcptr-s12x-discovery.py`: Scans the firmware image for potential function pointers to identify additional functions.
  - `ida-pid-ec80-naming.py`: Walks the PID handler tables and names the handler functions based on the PID and context.
  - `ida-deletedfn-report.py`: Validates the call tree to identify deleted functions.
- **Disassembly:** The disassembly of the deleted/modified functions (`disassembly-of-changed-functions.md`).
- **S19 File: `changes-2ec80.s19`** An S19 memory dump containing the global addresses of the deleted functions, PID and context tables, and interrupt vectors from the pre-update `2ec80` target. Note that this file contains only the global addresses of these items in PFLASH; it has not been 'expanded' to the paged format expected by many tools.
- **Testing Scripts: `j*.py`** scripts for fuzzing the J2497 ingress of the target and for testing susceptibility to simple PoCs of the vulnerabilities.
  - `j1587-pid-fuzzer.py`: Used to fuzz the J1587 PID handlers and discover the vulnerabilities in the 0xC2 and 0xED PID Handlers.
  - `j2497-sdr-badones-transmit.py`: Used to trigger the 0xC2 and 0xED vulnerabilities during in-motion tests. It emits the test signal on an attached FL2K SDR. (Expects the `j2497-keyhole` project to be installed as a package or run from the same directory).

These files are available at Zenodo DOI https://doi.org/10.5281/zenodo.19664811.

A.2.3 Badges

We apply for the **Artifacts Available** badge only. We cannot apply for the Functional or Reproducible badges because the full firmware files cannot be released as mentioned in the paper.

A.2.4 Hardware dependencies

- A standard desktop or laptop computer is sufficient for offline static analysis and reviewing the disassembly and scripts. A Windows 10 or 11 x64 machine is required for the `j1587-pid-fuzzer.py` test script.
- A FL2K VGA adapter with a SMA connector adapter is required for the `j2497-sdr-badones-transmit.py` test script
- A RP1210 VDA with J2497 support (or one with J1708 support and a converter) and any python-can CAN adapter is required for the `j1587-pid-fuzzer.py` test script.
- A generic USB HID relay switch (VID: 0x16C0, PID: 0x05DF) is required for power cycling the target in the `j1587-pid-fuzzer.py` test script.

A.2.5 Software dependencies

- **Python 3.12 or later**
- **IDA Pro** (with S12X architecture support) for running the IDA Python scripts, if desired.
- **py-hv-networks, RP1210, python-can, scapy, bitstring, hidapi, numpy, tabulate, and bincopy** pypi packages must be installed for the test and setup scripts.
- **Your own S12X firmware .s19 file** is required for running the IDA Pro scripts.
- **osmo-fl2k** is required to provide the `fl2k_file` executable used by the `j2497-sdr-badones-transmit.py` script.

A.3 Evaluation Workflow

The provided `changes-2ec80.s19` file can be loaded into an S12X disassembler to verify the structure of the deleted code. Please note that it is not sufficient for reproducing the execution of any of the scripts. The provided `disassembly-of-changed-functions.md` file contains disassemblies which can be used for reference if a working S12X disassembler is not available.

A.3.1 Major Claims

- **(C1):** The ID9363 update removed substantial J1587 PID processing functionality from the EC80 firmware [1].
- **(C2):** The removed functionality contained critical vulnerabilities, including buffer overflows and out-of-bounds writes (e.g., in handlers for PIDs `0xC2` and `0xED`). This is proven by the experiment (E1).
- **(C3):** The removed functionality contained a hardcoded credential (PID `0xC7`). This is proven by the experiment (E1).

A.3.2 Experiments

- **(E1):** [Verifying Removed Functionality] [30 human-minutes + 0 compute-hour + 0GB disk]: Reviewing the disassembly allows manual verification of claims C2 and C3.
  - **How to:** Review the disassemlby of the .s19 file provided or text of disassembly file provided.
  - **Preparation:** Ensure the `changes-2ec80.s19` file can be disassembled or the `disassembly-of-changed-functions.md` can be referenced.
  - **Execution:**
    1. Open the provided disassembly files.
    2. Observe the `Context at 0xDC3D` and `PID Table at 0xD9DD` sections to confirm that the following functions are PID handlers.
    3. Locate `sub_F1B623` and `sub_F2BA46` to verify claim C2.
    4. Locate `sub_F2AB7A` to verify claim C3.
    5. Load the `.s19` file to confirm byte sequences match the provided disassembly.
  - **Results:** Reviewing the text and `.s19` file is sufficient to corroborate the claims manually.

A.4 Notes on Reusability

This artifact uses standard Python 3 and IDA Pro environments. The scripts can be adapted for analyzing similar S12X-based firmware images.

Additionally, for users wishing to extract their own firmware via the HiWave debugger, memory can be dumped with the following script/commands (note: set the command window's cache limit to unlimited first):

```
DMM CACHINGOFF
DMM WRITEREADBACKOFF
HCS12X_MAP4000 RAM
db 0x4000'L..0x7FFF'L
db 0x0f8000'G..0x0FBFFF'G
```

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.

[1] Note that we cannot provide an artifact to confirm the claim that the code is deleted because that would require releasing the 'after' firmware image.