RIDAS: Real-time identification of attack sources on controller area networks

Jiwoo Shin and Hyunghoon Kim, *Soongsil University;* Seyoung Lee, Wonsuk Choi, and Dong Hoon Lee, *Korea University;* Hyo Jin Jo, *Soongsil University*

https://www.usenix.org/conference/usenixsecurity23/presentation/shin

This paper is included in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

RIDAS: Real-time identification of attack sources on controller area networks

Jiwoo Shin* *Soongsil University*
Hyunghoon Kim* *Soongsil University*
Seyoung Lee *Korea University*
Wonsuk Choi *Korea University*
Dong Hoon Lee *Korea University*
Hyo Jin Jo† *Soongsil University*

Abstract

Researchers have responded to various cyber attacks on controller area network (CAN) by studying technologies for identifying the source of an attack. However, existing attack source identification technologies have shown significantly lower accuracy depending on changes in the vehicle environment (temperature, humidity, battery level, etc.), or have proven to be circumvented by identification-aware attackers, or do not provide real-time identification. A real-time attack node identification technology that cannot be bypassed by an attacker while not being affected by changes in the vehicle environment is essential for cyber attack response technologies such as node isolation, security patch, digital forensics, etc.

To meet this need, we propose a novel real-time attack node identification method, called RIDAS, which can identify the attack source by using the error handling rule of CAN. RIDAS injects bit errors into the abnormal messages that have been detected by an existing intrusion detection system (IDS). The source that sent the abnormal message become the error passive state defined in CAN in which it cannot send consecutive messages. RIDAS then sequentially inspects all electronic control units (ECU) and identifies the node in the error passive state by checking the *priority reduction* phenomenon that occurs in that state. Moreover, RIDAS address two challenging issues, *identification robustness* and *identification errors*. Our experimental results, conducted on both a CAN bus prototype and one real vehicle, have demonstrated that RIDAS can accurately identify an attack source while remaining unaffected by changes in the vehicle's environment. Additionally, RIDAS is able to deal with *RIDAS-aware* attackers.

1 Introduction

Over years of research and development, autonomous driving technologies such as the advanced driver assistance system (ADAS) have become prevalent in automotive IT systems. Despite the advantages of ADAS, they introduce numerous security issues. For example, researchers have demonstrated that a vehicle can be remotely controlled through an in-vehicle network after compromising an electronic control unit (ECU) with a vulnerable remote interface [2, 9–11, 14, 18, 22–24, 28, 36–38].

To address these security concerns, various methods for detecting cyber attacks on vehicles have been introduced [3, 8, 15, 17, 21, 26, 29, 33, 34]. Although these intrusion detection systems (IDS) can detect cyber attack attempts on in-vehicle networks, they struggle to locate the attack source, i.e., a compromised ECU. This is mainly because in-vehicle networks are largely based on a Controller Area Network (CAN), which adopts a bus network topology, and because message source identifiers can be easily altered by attackers. Identifying the source of the attack is one necessary step that must be performed for security incident management, including network isolation, software patch, and so on. In previous studies, researchers have proposed several methods to identify an attack source on a CAN based on voltage-based fingerprints [5, 7, 16]. Voltage measurement allows a monitor device like an oscilloscope to fingerprint minor inherent discrepancies in voltage signals of different ECUs when they transmit CAN messages. However, identification using voltage-based fingerprints can result in errors because these fingerprints are subject to change as a result of varying environmental factors like power supply level, ambient temperature, humidity, and more. Moreover, an attack that manipulates the fingerprint information may occur during the periodically performed fingerprint update process. Other researchers have also found that voltage-based fingerprints can be corrupted by two compromised ECUs [1].

In light of this, in this paper, we propose real-time identification of attack sources (RIDAS). This approach identifies a compromised ECU based on the error confinement rule in the CAN standard. RIDAS is the first method that can effectively handle *identification robustness* and *identification errors* issues existing solutions have not adequately addressed.

Identification robustness. RIDAS is resilient to variable

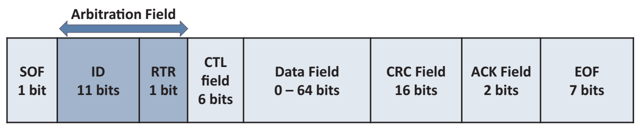
*Co-first authors
†Corresponding author (hyojin.jo@ssu.ac.kr)

environments and can identify an attack ECU while driving without affecting any vehicle operations. This is achieved by a system composed of four modules: a TEC emulation module, an attack handling (AH) module, a naive attack source identification (NASI) module and an *RIDAS-aware* attack source identification (RASI) module. When an attack message is detected by an existing IDS, a continuous transmission error on that message is intentionally generated a pre-defined number of times by the AH module before message transmission is completed. This places the ECU detected by the IDS into the error passive state, in which CAN messages cannot be continuously transmitted due to the suspend transmission time defined in the CAN standard. When the AH module creates a detection alert, the NASI module intentionally generates a bit error up to k_{add} times for every representative message of all ECUs in order to find the error passive ECU. (k_{add} is a system parameter.) Furthermore, RIDAS includes a RASI module that can deal with RIDAS-aware attackers.

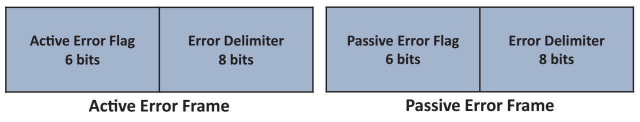
Identification error. RIDAS can handle false positives from an intrusion detection system, which is an inherent problem that most IDSs have. In the event that the AH module introduces bit errors into a normal CAN message, the NASI module must rectify the error by determining whether the ECU, which enters the error passive state due to a pre-defined number of consecutive errors, is a normal node or an attack node. The NASI module utilizes an ECU mapping table that links each CAN message with its source for this purpose. If the identifier pair (i.e., the abnormal identifier targeted by the AH module and the representative identifier of the error passive ECU identified by the NASI module) does not match any entry in the ECU mapping table, then the node identified by the NASI module is considered a normal node. Otherwise, the ECU identified by the NASI module is classified as an attack node.

RIDAS has been validated on a CAN prototype and a real vehicle. Our evaluation results show that RIDAS not only addresses both *identification robustness* and *identification error*, but can also identify an attack ECU without affecting driving. In summary, this paper makes the following contributions:

- Development of a novel method called RIDAS that can deal with both *identification robustness* and *identification errors* by using the error passive state of CAN error handling (Sections 4.3, 4.4 and 4.5)
- Proposal of a methodology that deals with *RIDAS-aware* attackers (Section 4.6)
- Evaluation of RIDAS on a CAN bus prototype and one real vehicle (Section 5).



(a) CAN data frame



(b) CAN error frame

Figure 1: The structure of the CAN data frame and error frame

2 Background

2.1 Controller Area Network

The CAN, one of a number of the in-vehicle networks, was developed by Bosch in 1986. There are four different types of CAN frames: a data frame, a remote frame, an error frame, and an overload frame. Note that we focus on the data frame, remote frame, and error frame. The structure of the data frame is shown in Fig. 1a and consists of seven fields: a start of frame (SOF), an arbitration field, a control (CTL) field, a data field, a cyclic redundancy check (CRC) field, an acknowledge (ACK) field, and an end of frame (EOF). The arbitration field contains an identifier (ID) and a remote transmission request (RTR) and plays an important role in the data frame. The priority of the data frame is determined by the bit-wise arbitration of the arbitration field, such that the lower the identifier of the arbitration field, the higher the priority. Additionally, by using the arbitration field, each node can decide whether to receive the data frame or not. If a CAN data frame's RTR bit is set to 1, it is known as a remote frame. When a remote frame is transmitted, the receiver ECU responds to the request by sending a data frame. The error frame is used to notify the detected errors and consists of an error flag and an error delimiter. As shown in Fig. 1b, there are two types of the error frames that depend on the error state of the node, i.e., an active error frame and a passive error frame. The active error frame and passive error frame have an error flag with six dominant bits and six recessive bits, respectively. The error delimiter is eight recessive bits on both frames.

2.2 CAN Error Handling

2.2.1 Error Type

The CAN protocol defines five types of errors:

- Bit error: The CAN node monitors the bus as it sends a bit to the bus. A bit error occurs if the monitored bit and the transmitted bit are different, with the exception of the arbitration field.

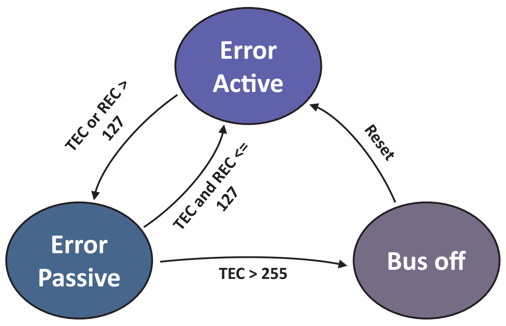


Figure 2: Error confinement mechanism

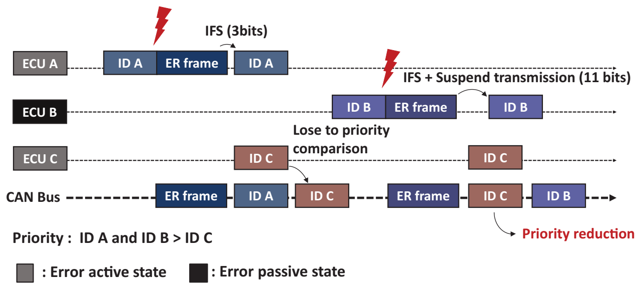


Figure 3: Priority reduction

- Stuff error: The CAN frame should be encoded using the bit stuffing method, except for the EOF. If six consecutive equal bits are observed, a stuff error occurs.
- CRC error: If the verification of the CRC value of the received frame fails, a CRC error occurs.
- Form error: A form error occurs when an illegal bit violates the format of a CAN frame.
- ACK error: If the node receiving the CAN frame does not transmit the dominant bit (0) in the ACK slot, an ACK error occurs.

2.2.2 Fault Confinement

Each ECU maintains two error counters: the transmit error counter (TEC) and the receive error counter (REC). If an error is detected during transmission, the TEC is incremented by 8. Conversely, if an error is detected during reception, the REC changes incrementally by 1. When a message is successfully transmitted, both counters decrease by 1. The CAN defines three error states based on the values of the TEC and the REC: error active, error passive, and bus-off. The CAN error states and their corresponding counters are depicted in Fig. 2.

Error active state. By default, all ECUs start in this state. A node in this state has a minimum idle time 3-bit idle time duration between consecutive frame transmissions and sends an active error frame to alert other nodes.

Error passive state. An ECU enters this state when the value of either the TEC or the REC surpasses 127. A node in this state requires an additional 8-bit duration, known as suspend transmission time, in addition to the 3-bit duration for consecutive frame transmissions. It also sends a passive error frame to notify other nodes.

Bus-off state. An ECU enters this state when the TEC value exceeds 255. The node can return to the error active state after observing at least 128 instances of recessive bits on the CAN bus or after a CAN controller reset.

2.3 Priority Reduction

As mentioned in section 2.1, the lower the identifier in the arbitration field, the higher the priority of a message. However, if there is an error passive node on the CAN bus, a high-priority message from the passive node may be transmitted later than a low-priority message due to the suspend transmission time. This phenomenon is referred to as *priority reduction* in [32].

For example, if an intentional error is injected on the message transmission of an error passive node, retransmission is immediately attempted due to the message transmission failure. However, in order to retransmit the message, it waits for the duration of the suspend transmission time, which is a penalty time in the error passive state. During this time, other nodes may transmit messages with a lower priority than the retransmission message. Thus, the *priority reduction* can be used to differentiate between a message transmitted by a passive node and a message transmitted by an active node. The principle of priority reduction is shown in Fig. 3.

2.4 Unified Diagnostic Services

The Unified Diagnostic Services (UDS) is a diagnostic communication protocol used for diagnosing vehicle status or updating an ECU, and is defined by ISO 14229-1 [13]. UDS allows functions such as diagnostic session control and ECU reset to be performed using diagnostic messages over the CAN. Generally, diagnostic messages use identifiers in the range of 0x700 to 0x7FF and include the service identifier (SID) to be diagnosed in the data field. Specifically, one diagnostic identifier is assigned to each ECU, and diagnostic messages are communicated through UDS request and response messages.

3 System Model

3.1 Assumptions

System assumption. In general, the network structures adopted by vehicle manufacturers are divided into several buses— including powertrain, comfort, body, etc.—and these separate buses are connected via a gateway. Thus, we assume that RIDAS is installed as an additional node on the individual buses or is integrated into the gateway. Additionally, we assume that RIDAS maintains an ECU mapping table for a target vehicle, which maps each CAN identifier to its source ECUs. The table includes a list of ECUs installed on the vehicle, the identifiers (CAN IDs) assigned to each ECU, and the message transmission cycle of each identifier.

Adversary assumption. It is important to note that an ECU can be compromised by vulnerable connectivity interfaces such as USB, cellular, WiFi, or Bluetooth [2, 24]. Therefore, a cyber attack can be performed on vehicles if an attacker can access network buses through a physically or remotely compromised ECU. However, in our system model, we do not consider attackers who have physically accessed the network buses through an additional node (e.g., a device plugged into the OBD-II port) because this node can be easily recognized by drivers. Since RIDAS is not designed to detect attack attempts on the CAN bus, we assume attacks are detected by a CAN IDS.[*] Furthermore, we assume that remote attackers cannot compromise RIDAS or the ECU mapping table.

3.2 Threat Model

An attacker can perform masquerade attacks to control vehicle operations by injecting attack messages into the CAN bus through a compromised ECU. In this paper, we consider two different types of masquerade attackers: *naive* and *RIDAS-aware* attackers.

A *naive* attacker does not have any knowledge of how RIDAS identifies attack sources. Thus, the attacker uses the default setting of the CAN controller and does not attempt to tamper with the CAN controller settings to evade RIDAS. The *naive* attacker simply injects attack messages whenever he/she wants.

A *RIDAS-aware* attacker, however, understands how an attack source can be identified by RIDAS. Thus, the attacker leverages his/her knowledge to evade RIDAS by modifying the CAN controller settings. Since RIDAS utilizes the CAN's error handling (i.e., the characteristic of the error passive state) to identify attack sources, the attacker can choose one of the following three ways—CAN controller reset, one-shot mode, and fast message transmission—as a method to decrease the TEC.

- CAN controller reset: This method sets the TEC to zero by initializing all registers, including the TEC register of the CAN controller.
- One-shot mode: In this mode, even if there is message transmission failure due to a CAN error, the message is not retransmitted.

[*]The detection of cyber attacks on the CAN bus is beyond the scope of this paper.

- Fast message transmission: As the TEC is decremented by 1 for each successful message transmission, a fast message transmission leads to a quick decrease in the TEC.

4 RIDAS

4.1 Design Challenges

In order to identify an attack node by checking the *priority reduction* of the error passive state, the following challenges should be considered.

Challenge 1. How can we identify an attack node on the CAN bus within such a short time in which the error passive state can be detected? In the default setting of a CAN controller, an ECU in the error passive state can revert to the error active state when its own TEC drops below 128 through successful transmission of messages. The time it takes for a node in the error passive state to convert to the error active state varies from tens of milliseconds to several thousand milliseconds, depending on the node's error state (i.e., the TEC and REC states), the number of identifiers assigned to the node, and the message transmission cycle of the identifiers. Therefore, the process of distinguishing the attack node from the normal nodes through the *priority reduction* phenomenon should, in the worst case, be performed within tens of milliseconds.

Challenge 2. How can we identify an attack node that can control its own TEC? If an attacker exploits the CAN controller's reset function or the one-shot mode function, the compromised ECU cannot be in the error passive state. In this case, it is impossible to identify the attack node by checking the *priority reduction* phenomenon of the error passive state. In addition, the attacker can transmit a large volume of messages with the fastest transmission cycle to drastically decrease the TEC, which shortens the duration of the error passive state.

Challenge 3. How can we deal with identification errors originating from false positives, an inherent problem of IDSs? In order to identify an attack source using the *priority reduction* of the error passive state, an attack message from the attack node must be detected by an IDS and destroyed by a bit error before its transmission is complete. However, since false positives from IDSs are an inherent problem, it is difficult for IDSs to address this issue completely. Therefore, attack source identification should be designed to be robust against IDS false positives.

4.2 Overview

The architecture of RIDAS is shown in Fig. 4. RIDAS is composed of four modules: a TEC emulation module, an attack handling (AH) module, a *naive* attack source identification (NASI) module, and a *RIDAS-aware* source identification

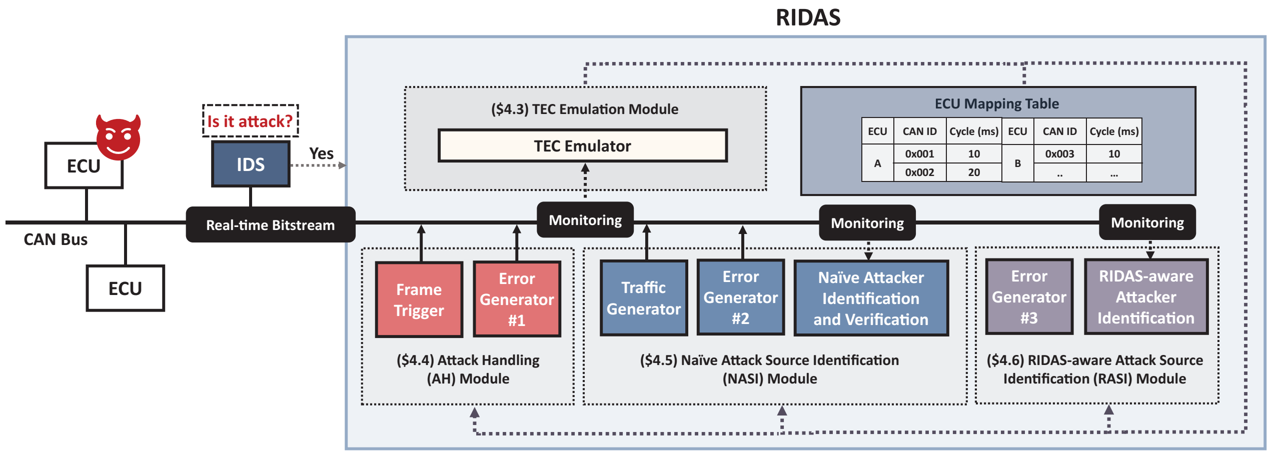

RIDAS
Is it attack?
ECU
IDS
Yes
CAN Bus
ECU
Real-time Bitstream
($4.3) TEC Emulation Module
TEC Emulator
ECU Mapping Table
ECU | CAN ID | Cycle (ms) | ECU | CAN ID | Cycle (ms)
A | 0x001 | 10 | B | 0x003 | 10
0x002 | 20 | .. | ...
Monitoring
Monitoring
Monitoring
Frame Trigger
Error Generator #1
Traffic Generator
Error Generator #2
Naïve Attacker Identification and Verification
Error Generator #3
RIDAS-aware Attacker Identification
($4.4) Attack Handling (AH) Module
($4.5) Naïve Attack Source Identification (NASI) Module
($4.6) RIDAS-aware Attack Source Identification (RASI) Module

Figure 4: RIDAS system overview

Table 1: Notations

Notations	Descriptions
k_{init}	The number of bit errors injected by the error generator #1 of the AH module.
k_{add}	The number of additional bit errors injected by the error generator #2 of the NASI module to observe a priority reduction.
$TEC^{t_s}_{estimate}$	The estimated value of TEC at the time, t_s, when RIDAS was started.
$TEC^{t_c}_{estimate}$	The estimated value of TEC at the current time, t_c.
TEC_{base}	The TEC value that should be reduced using remote frames after a compromised node is identified.

module (RASI).

The TEC emulation module estimates the TEC of each ECU by monitoring the error frames of all CAN messages.

The AH module consists of two components: 1) an error generator #1, which generates bit errors on abnormal CAN messages detected by an IDS to transition a suspicious ECU from an error active state to the error passive state, and 2) a frame trigger that instigates the transmission of CAN packets from a specific ECU using a report frame or a UDS request message, aiming to reduce the ECU's TEC below a predefined threshold.

The NASI module comprises three components: 1) a traffic generator that generates background traffic, and 2) an error generator #2 that introduces suitable additional bit errors based on the ECU's TEC state, as estimated by the TEC emulation module, to monitor *priority reduction*, and 3) *naive* attacker identification and verification, which can identify an ECU in the error passive state in real-time and validate the identification result.

The RASI module is composed of two components: 1) *RIDAS-aware* attack identification and 2) an error generator #3. The *RIDAS-aware* attack identification checks for a drastic TEC decrease resulting from the use of the reset function, one-shot mode setting, or fast message transmission by a compromised node. Specifically, it monitors the CAN bus to: 1) observe changes in the transmission cycle of CAN messages not transmitted during the CAN controller's reset process, 2) check for retransmission on an erroneous message due to the one-shot mode setting, and 3) detect excessive message injections, which could decrease the TEC. If a failed transmission message is not retransmitted or fast message transmission attempts are detected, the error generator #3 produces a bit error to locate the compromised ECU enabling one-shot mode or to prevent the TEC value of the compromised ECU from decreasing. The system flows and notations of RIDAS are described in Fig. 5 and Table 1, respectively.

4.3 TEC Emulation

The goal of the TEC Emulation module is to monitor the CAN bus in real-time and measure the TEC of each ECU. The CAN bus may experience transmission errors due to various factors such as temperature variations, collision events, and CAN wiring issues. Consequently, an ECU's TEC may not always be zero. Therefore, if RIDAS's error generators cause a transmission error while the ECU's TEC is in a non-zero state, the ECU could transition to the bus-off state. To prevent an ECU from entering the bus-off state, a TEC emulator capable of emulating the TEC of all ECUs is necessary. Algorithm 1 in Appendix A.1 illustrates how the TEC emulator measures each ECU's TEC by counting the error frames and monitoring the CAN bus.

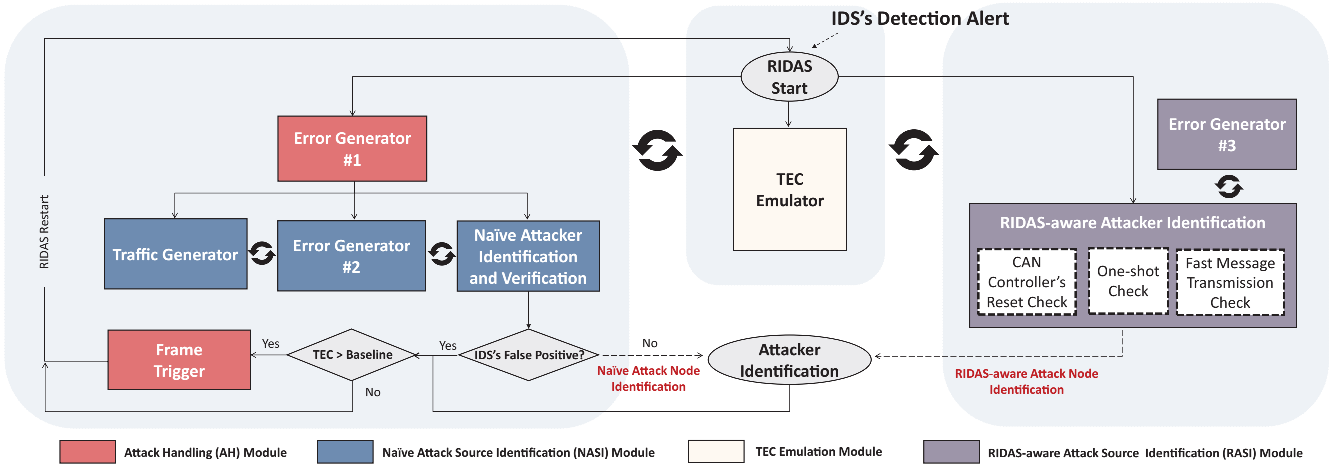

IDS's Detection Alert
RIDAS Start
Error Generator #1
TEC Emulator
Error Generator #3
Traffic Generator
Error Generator #2
Naïve Attacker Identification and Verification
RIDAS-aware Attacker Identification
CAN Controller's Reset Check
One-shot Check
Fast Message Transmission Check
RIDAS Restart
Frame Trigger
Yes
TEC > Baseline
Yes
IDS's False Positive?
No
Attacker Identification
No
Naïve Attack Node Identification
RIDAS-aware Attack Node Identification
Attack Handling (AH) Module
Naïve Attack Source Identification (NASI) Module
TEC Emulation Module
RIDAS-aware Attack Source Identification (RASI) Module

Figure 5: RIDAS system flows

4.4 Attack Handling (AH)

When an IDS detects an abnormal message, the AH module's error generator #1 must inject bit errors k_{init} times before the transmission of the attack message completes, to transition the suspect node into an error passive state. The value of k_{init} should be set between 16 and 32. After k_{init} bit errors have been injected into the suspect node, its TEC reaches a value between 128 and 256, and this value is defined as TEC_{init}.

If the IDS falsely identifies messages from normal ECUs as attacks, normal nodes could be forced into a bus-off state by error generator #1. To prevent this, before restarting RIDAS, the frame trigger checks all TEC values, $\text{TEC}^{t_c}_{estimate}$, obtained from the TEC emulation module. ($\text{TEC}^{t_c}_{estimate}$ represents the estimated TEC value at the current time, t_c.) If $\text{TEC}^{t_c}_{estimate}$ exceeds TEC_{base}, a RIDAS system parameter, the frame trigger sends a remote frame or a UDS request message to the respective ECUs to lower their TEC values below TEC_{base}. The total number of remote frames or UDS request messages sent by the frame trigger can be calculated by the following equation.

$$\#\, trigger\ frames = \sum_{\forall ECU \in \mathbb{E}} \left\lceil \frac{(\text{TEC}^{t_c}_{estimate} - \text{TEC}_{base})}{8} \right\rceil$$

, where $\mathbb{E}$ represents a set of all ECUs installed in a vehicle.

4.5 Naive Attack Source Identification (NASI)

The main objective of the NASI module is to identify the node that has transitioned to the error passive state due to the AH module's actions.

When the AH module activates the NASI module, the traffic generator begins generating background traffic. At the same time, the error generator #2 introduces k_{add} bit errors into the transmission of each ECU, where k_{add} is the value dynamically calculated as follows.

$$k_{add} = \left\lceil \frac{(\text{TEC}^{t_s}_{estimate} - \text{TEC}^{t_c}_{estimate})}{8} \right\rceil + k_{check}$$
$$,(0 \le \left\lceil \frac{(\text{TEC}^{t_s}_{estimate} - \text{TEC}^{t_c}_{estimate})}{8} \right\rceil < 16 - k_{check})$$

, where k_{check} is a system parameter that is determined by the ratio of the background traffic and is the minimum number of bit error injections required to observe *priority reduction* in nodes that have transitioned to the error passive state, and $\text{TEC}^{t_s}_{estimate}$ represents the estimated value of TEC at the time when starts, expressed as t_s.

The ECU interrupted by the error generator #2 will retransmit the interrupted message k_{add} times. During this retransmission process, the NASI module monitors the node's *priority reduction* to determine if it has transitioned to the error passive state. If the ECU performing k_{add} retransmissions is not the abnormal node targeted by the AH module, it should not be in the error passive state, and thus, no *priority reduction* should occur. Conversely, *priority reduction* can be observed during the *suspend transmission time* of the abnormal node because k_{init} bit errors have been injected by the AH module. Once a node in the error passive state is identified, the ECU mapping table is used to verify whether the node is indeed an attack node. For instance, if the abnormal ID detected by the IDS and the error passive node's representative ID identified by NASI do not violate the ECU mapping table, this node is not an attack node, but a normal node that the IDS falsely detected. Otherwise, the error passive node identified by NASI is an attack node.

Successfully observing *priority reduction* depends on four conditions: 1) the default number of bit error injections (k_{check}), 2) the representative ID for each ECU, 3) the scheduling of the ECU inspection order, and 4) the background CAN traffic. To increase the probability of the NASI module observing *priority reduction*, these four conditions must be managed as follows.

4.5.1 The default number of bit error injections (k_{check})

In order for the NASI module to check *priority reduction* of a node in the error passive state, the default number of bit error injections (k_{check}) must be determined. Suppose we define an event $\mathbb{A}$ where *priority reduction* does not occur in a single bit error injection when there is an error passive node. The expected number of bit error injections (k_{check}) for a given probability p can be calculated as follows.

$$\begin{aligned} \text{minimize:} \quad & k_{check} \\ \text{subject to:} \quad & (1 - Pr(\mathbb{A}))^{k_{check}} \geq p \\ & k_{check} \geq 1 \;\; (k_{check} \text{ is an integer}) \end{aligned}$$

, where $Pr(\mathbb{A})$ represents the probability of the event $\mathbb{A}$, and p is a system parameter in RIDAS that represents the probability of successful identification by the NASI module. p is typically set to a value that approximates 1, such as 0.99999.

However, since it is not feasible to accurately obtain $Pr(\mathbb{A})$ from the CAN bus due to various factors like the execution time of the NASI module, the number of attack message transmissions, and the number of normal message transmissions, k_{check} for each vehicle type should be experimentally determined as demonstrated in Section 5.3.

4.5.2 The representative ID for each ECU

To identify an ECU in the error passive state, the NASI module sequentially generates k_{add} bit errors on each ECU to observe the *priority reduction*. Therefore, it is necessary to select a representative CAN ID of each ECU to be used as a target for bit error generation. To complete the NASI module inspection as quickly as possible, the representative ID for each ECU is set to the CAN ID with the fastest transmission cycle among the CAN IDs assigned to that ECU. If an ECU has two or more CAN IDs with the same fast transmission cycle, the CAN ID with higher priority is selected as the representative ID. An example of selecting a representative ID for each ECU is shown in Table 2.

4.5.3 ECU inspection order Scheduling

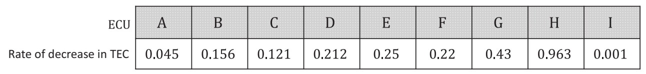
The ECU inspection by the NASI module is carried out sequentially, one at a time. However, if the ECU that gets inspected last by the NASI module is the abnormal node targeted by the AH module, even if k_{add} bit errors are injected, the error passive state of that node may not be maintained due to the delay caused by the inspection time of other ECUs. To address this issue, the inspection order of the NASI module should be scheduled in descending order according to the TEC decrease rate for each ECU. The TEC decrease rate depends on the number of CAN IDs assigned to the ECU and the message transmission cycle of the CAN IDs. Thus, the TEC decrease rate, TDR, can be calculated through the following formula.

$$\text{TDR}_{\text{ECU}_i} = \sum_{\forall \text{cid}_{i,j} \in \text{ECU}_i} \frac{1}{P_{\text{cid}_{i,j}}}$$

, where ECU_i is the i-th ECU on a vehicle, $\text{cid}_{i,j}$ is the j-th CAN ID of ECU_i, and $P_{\text{cid}_{i,j}}$ is the message transmission period of $\text{cid}_{i,j}$. An example of inspection order scheduling is shown in Fig. 6.

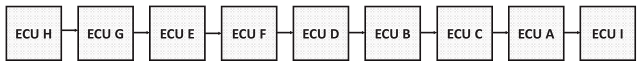
Table 2: The example of selecting a representative ID (The red color indicates the representative ID of each ECU)

ECU	CAN ID	Transmission Time (ms)
X	0x101	10
	0x102	10
	0x104	100
	0x201	10
	0x202	50



ECU	A	B	C	D	E	F	G	H	I
Rate of decrease in TEC	0.045	0.156	0.121	0.212	0.25	0.22	0.43	0.963	0.001

(a) The TEC decrease rate of ECUs



(b) ECU inspection order

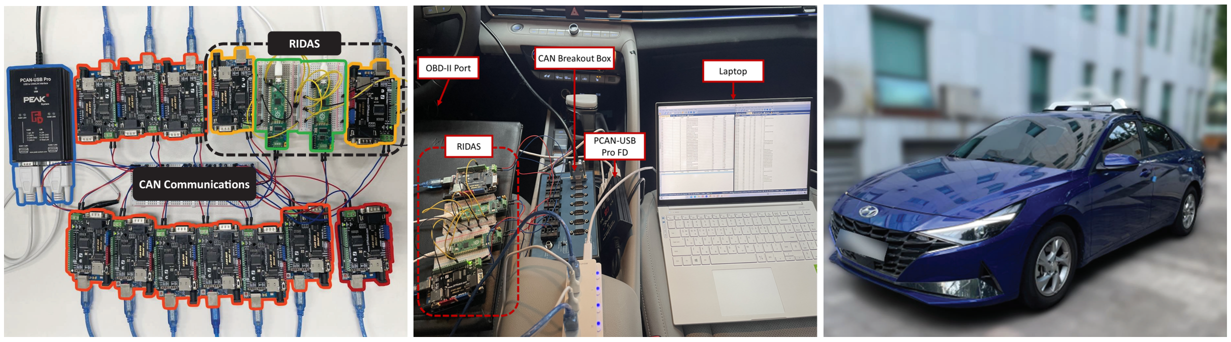
Figure 6: Inspection schedule of the test vehicle (Vehicle A)

4.5.4 Background CAN Traffic

The generation of background traffic is a crucial component of the NASI module's success in observing *priority reduction*. This is because *priority reduction* occurs probabilistically and is dependent on the number of CAN messages that have lower priority than the representative ID of each ECU. The traffic generator creates background traffic comprised of CAN IDs that have a lower priority than all other CAN IDs. This background traffic thus enhances the probability of *priority reduction*.

4.6 RIDAS-aware Attack Source Identification (RASI)

The RASI module is designed to deal with a *RIDAS-aware* attacker who can rapidly decrease the TEC. There are three methods to quickly change the TEC: 1) resetting the CAN



(a) CAN bus prototype (b) Real Vehicle A (Inside) (c) Real Vehicle A (Outside)

Figure 7: Environment setup for the CAN bus prototype and real vehicle

Table 3: The reset time of three CAN controllers

Device	CAN Controller	CAN Transceiver	Reset Time (ms)
Arduino Uno with CAN Bus Shield[†]	MCP2515	TJA1050	≤ 1
Arduino Due[‡]	SAM3X8E	SN65HVD230	
ESP32[§]	SJA1000		

controller, 2) using the one-shot mode, and 3) sending a large volume of data at a high transmission rate. To address the aforementioned attack methods employed by a *RIDAS-aware attacker*, the RASI module performs three processes as follows:

4.6.1 Response to a CAN controller reset

When a *RIDAS-aware* attacker resets the CAN controller of a compromised ECU, it typically takes about 1ms for the compromised ECU to restart its own CAN controller, as shown in Table 3. Despite the short reset time, the RASI module can detect the change in transmission cycles of certain CAN packets originating from the reset CAN controller when a reset event occurs. Therefore, the RASI module can examine the CAN IDs whose transmission cycle has changed and identify the corresponding node using the ECU mapping table.

4.6.2 Response to one-shot mode

If a compromised ECU is set to one-shot mode, the message that failed to be transmitted by error generator #1 is not retransmitted due to the one-shot mode setting. In this case, the RASI module checks for message retransmission, rather than *priority reduction*. To do this, the RASI module selects at least one of the identifiers assigned to each ECU by referring to the ECU mapping table. When messages containing the selected identifiers are transmitted, error generator #3 generates a bit error on them, and the RASI module checks if these messages are retransmitted. For example, if message retransmission does not occur after a bit error is injected into a message containing a specific identifier, the RASI module can check this by monitoring changes in the message transmission cycle of the identifier. Then, the RASI module can identify the ECU to which this identifier is assigned by referring to the ECU mapping table. Finally, the RASI module can determine that the ECU is the compromised node set to the one-shot mode by the *RIDAS-aware* attacker.

4.6.3 Response to fast message transmission

If a compromised node injects a large volume of messages in a fast transmission cycle, the duration of the compromised node's error passive state is shortened due to the TEC decrease. Thus, whenever the compromised node's TEC decreases by 8, a bit error is injected to restore the node's TEC to its original value. To achieve this, the RASI module detects a fast message transmission attempt aimed at reducing the TEC when the CAN bus load is maintained at 70-100%, which is higher than the normal CAN bus load range of 40-60%, excluding background traffic.

In the case of attacks sending a large number of messages with a single identifier, the attack identifier can be easily identified because this identifier is observed more frequently on the CAN bus than other identifiers. However, since the RASI module does not know which ECU is transmitting the fast cycle message, it injects a bit error into each RID of all ECUs whenever the messages with the corresponding identifier are successfully transmitted 8 times.

In the case of attacks involving a large number of messages with multiple identifiers, the RASI module selects all identifiers that are being transmitted in a fast transmission cycle. Then, by counting the total number of message transmissions for messages with the corresponding identifiers on the CAN bus, a bit error is injected for each RID of all ECUs every $(8 \times n + 1)$-th message transmission, where n is a non-negative integer.

[†]https://github.com/DFRobot/DFRobot_MCP2515
[‡]https://github.com/collin80/due_can
[§]https://github.com/miwagner/ESP32-Arduino-CAN

5 Evaluation

5.1 Experimental Setup

CAN DBC. Since there is no available CAN DBC, i.e., Database CAN, for the target real vehicle (Hyundai Avante CN7), we collected CAN traffic from the target vehicle and obtain the ECU mapping table using the existing mapper solution [19]. As a result, the test vehicle was found to have nine ECUs, and each ECU was indexed from A to I, as shown in Table 6 in the Appendix.

CAN bus prototype. Fig. 7a shows the environment of the CAN bus prototype, which consists of four components; 1) nine nodes that simulate ECUs on the target vehicle, 2) a compromised node (a *naive* or *RIDAS-aware* attacker), 3) RIDAS, and 4) the monitoring tool.

The nine normal nodes, highlighted in orange, and the compromised node, highlighted in red, are composed of Arduino UNOs with CAN bus shields. The TEC Emulation module, AH module, NASI module, and RASI module, highlighted in green and yellow, are implemented in four devices, which are two CANPico boards and two Arduino UNOs with CAN bus shields[¶]. The monitoring tool, highlighted in blue, uses a device called PCAN-USB Pro. All devices are connected to the CAN bus, and RIDAS's modules communicate with each other through the UART. The nine nodes and the traffic generator are configured to send CAN messages as shown in Table 6 and Table 7 in the Appendix, respectively.

Real vehicle. To set up this environment, we used a real vehicle (referred to as Vehicle A), Hyundai Avante CN7, as shown in Fig. 7c. The environment is presented in Fig. 7b, and the configuration is the same as the CAN bus prototype except for the nine nodes simulating ECUs on the target vehicle. RIDAS is connected through the OBD-II port of the vehicle.

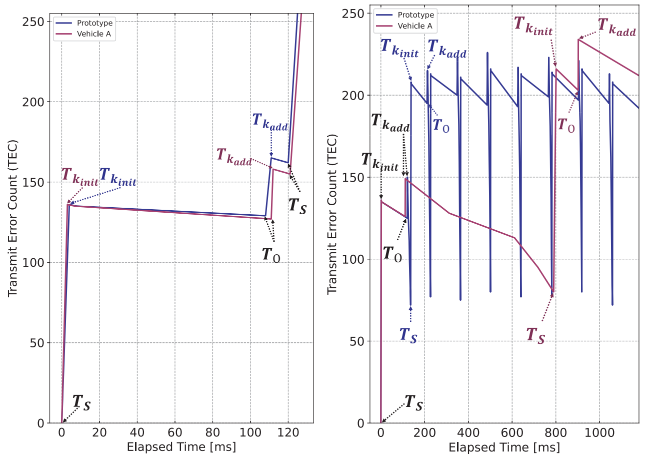
5.2 Evaluation of AH's frame trigger

To evaluate the effect of frame triggers, we designated ECU A in Vehicle A as an attack node and measured the TEC of ECU A. Fig. 8a shows the variation in the TEC value of ECU A without the frame trigger, while Fig. 8b shows the variation in the TEC of ECU A with the frame trigger. In this experiment, TEC$_{base}$ was set to 79. Remote frames were used in the prototype environment, while UDS request messages were used in Vehicle A. [‖] Fig. 8 shows that the frame trigger allows RIDAS to reduce the TEC value of ECU A, thereby preventing it from reaching the bus-off state by RIDAS restart.

Moreover, in the prototype environment, it took approximately 10ms for the TEC to decrease to TEC*base* after being increased by both *kinit* and k_{add}. However, in Vehicle A, it took approximately 678ms. The time difference stems from the difference in using remote frames and UDS request messages. Experimental results show that the ECU's response time to remote frames is significantly faster than its response time to UDS request messages. This means that, in Vehicle A, it can be inferred that the time required for the restart after RIDAS termination is over 678ms for Vehicle A However, if software updates that support remote frames are applied in Vehicle A, the time required for RIDAS to restart can be reduced to a maximum of 10ms.

¶AH module's frame trigger and NASI module's traffic generators are highlighted in yellow color.

‖As Vehicle A ECUs do not support remote frames, we substituted these with UDS diagnostic request messages. For ECU A, the diagnostic request ID is 0x7B3 and the diagnostic response ID is 0x7BB. This diagnostic ID pair was obtained experimentally.



(a) TEC (w/o frame trigger) (b) TEC (w/ frame trigger)

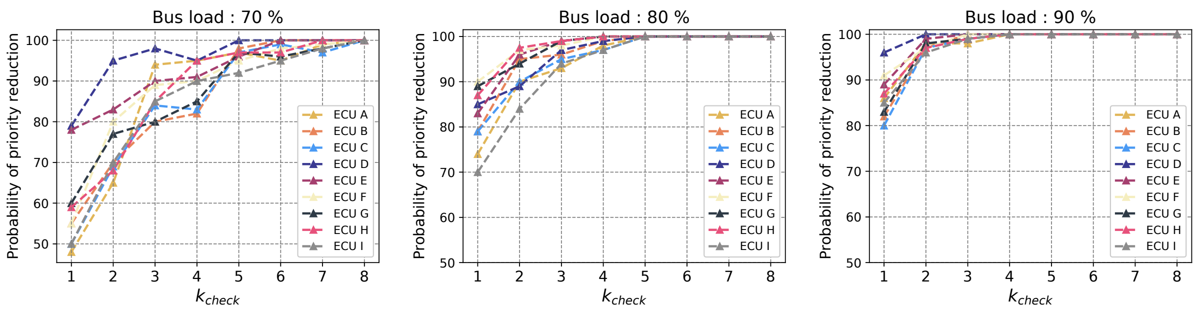
Figure 8: TEC of ECU A (TEC$_{base}$ = 79, $k_{init} = 17$, $k_{check} = 5$) (T_s: The time at which RIDAS starts; $T_{k_{init}}$: The time at which the TEC increase is completed due to injecting k_{init} bit errors; $T_{k_{add}}$: The time at which the TEC increase is completed due to injecting k_{add} bit errors; T_o:The time at which the observation of *priority reduction* is completed.)

Table 4: The RID information of each ECU

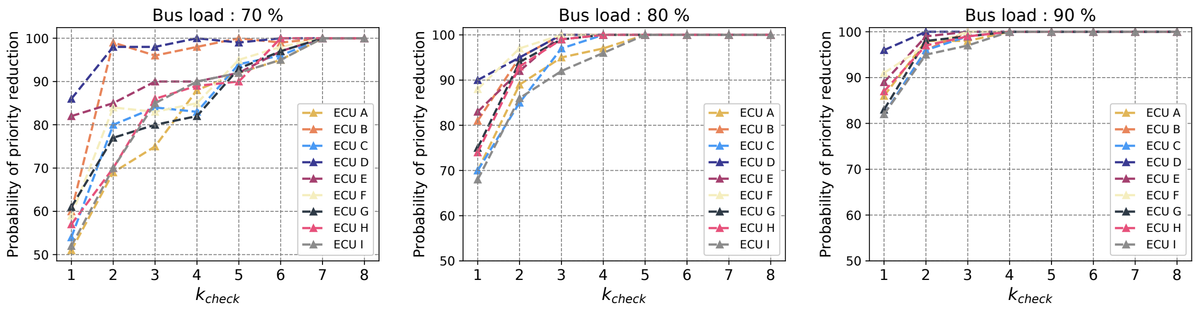
CAN bus prototype & Real Vehicle A									
ECU	A	B	C	D	E	F	G	H	I
RID	0x48A	0x340	0x391	0x389	0x251	0x130	0x153	0x260	0x563
Transmission Cycle (ms)	50	10	20	20	10	10	10	10	600

5.3 Evaluation of NASI

In this section, we experimentally measured the success rate of the NASI module's identification of error passive nodes. This rate depends on the CAN bus load and the number of bit error injections, k_{check}. We also discuss the scheduling of the NASI module's inspection order and the module's completion time for each ECU. To evaluate the NASI module, we set the representative identifier (RID) for each ECU installed in Vehicle A. The RID information used in the experiment is shown in Table 4.



(a) Prototype



(b) Vehicle A

Figure 9: The probability of *Priority Reduction* according to the bus load and k_{check}

5.3.1 Priority Reduction according to the CAN bus load and k_{check}

The probability of *Priority Reduction* correlates positively with the CAN bus load and k_{check}. To assess the probability of *Priority Reduction* according to the CAN bus load and k_{check}, we maintained the bus load at 70%, 80%, or 90% and conducted a *Priority Reduction* experiment corresponding to k_{check}. For this test, we generated background traffic with the lowest priority, a CAN ID of 0x600, to maintain the bus load at levels from 70% to 90%.

Then, after setting each ECU to an error passive state, k_{check} bit errors were injected into the ECU to observe if *Priority Reduction* occurred. We conducted this test 100 times for each ECU and k_{check}. Fig. 9 depicts the average probability of *Priority Reduction* (i.e., the identification probability of the error passive node) according to k_{check} when the CAN bus load is maintained at 70%, 80%, or 90%. The probability of *Priority Reduction* reached 100% for all ECUs when k_{check} was set to 8 with a CAN bus load of 70%, when k_{check} was set to 5 with a bus load of 80%, and when k_{check} was set to 4 with a bus load of 90%.

As a result, we confirmed that a higher volume of background traffic increased the likelihood of observing *Priority Reduction*, which corresponded to a lower number of bit error injections, k_{check}.

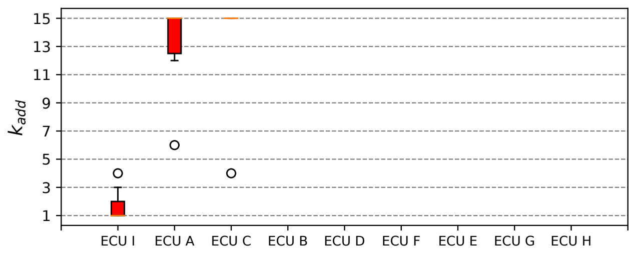
Table 5: The completion time of the NASI module ($TEC_{base} = 79$, $k_{check} = 5$, background traffic = 80%)

	Compromised ECU	A	B	C	D	E	F	G	H	I
Completion Time (ms)	Prototype	164.6	75.7	106.3	73	42.4	43.7	33.2	**5.4**	**563.5**
	Vehicle A	150.1	75.8	99.6	72.3	38.4	40.1	29.7	**5.1**	**458.67**

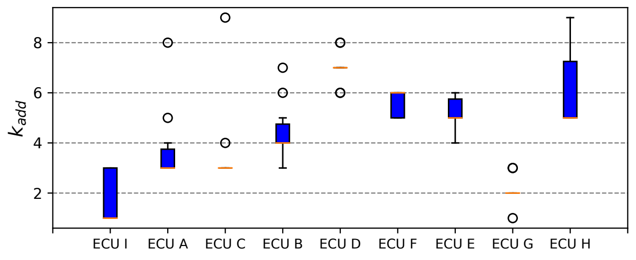
5.3.2 Evaluation of NASI module's inspection order

As outlined in section 4.5, the NASI module sequentially injects bit errors into the RIDs of all ECUs and identifies the attack source by detecting the ECU in the error passive state. Therefore, the attack node must be in an error passive state when the NASI module conducts an inspection on the compromised node. This is why it is crucial to schedule the NASI module's inspection order according to the TEC decrease rate.

The inspection schedules of the NASI module used in the experiment are set in ascending and descending orders based on the TEC decrease rate. As depicted in Fig. 10, when the inspection schedule is set in ascending order, the additional bit errors (k_{add}) for ECU B, D, F, E, G, and H exceed 16. On the other hand, if the inspection schedule is set in descending order, the additional bit errors (k_{add}) for all ECUs fall within the range of 1 to 10. This shows that it is more advantageous to arrange the NASI inspection sequence in descending order based on the TEC decrease rate. The NASI inspection sequence in descending order is shown in Fig. 6.



(a) Ascending order



(b) Descending order

Figure 10: k_{add} according to the ECU inspection schedule ($\text{TEC}_{base} = 79$, $k_{check} = 5$, background traffic = 80%)

5.3.3 The completion time of NASI module

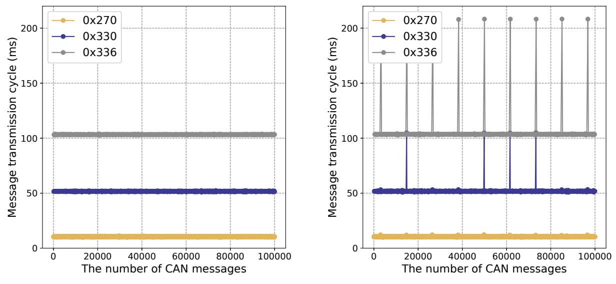
Table 5 presents the average completion time of the NASI module in both the CAN bus prototype and Vehicle A. If the compromised node is ECU H, which is first in the NASI inspection order, the identification of this node (i.e., ECU H) is completed in approximately 5ms in Vehicle A. Conversely, if the compromised node is ECU I, which is last in the NASI inspection order, we confirmed that the inspection completion time does not exceed 500ms in Vehicle A.

5.4 Evaluation of RASI

In this subsection, we also evaluate the performance of the RASI module. In this evaluation, since we cannot modify the CAN controllers of ECUs installed on the actual vehicle without compromising them, we used a simulated *RIDAS-aware* attacker to evaluate the RASI module. To simulate the *RIDAS-aware* attacker, three CAN IDs (i.e., 0x270, 0x330, and 0x336) were assigned to a virtual ECU, termed ECU L. This information can be found in Table 8 in the Appendix.

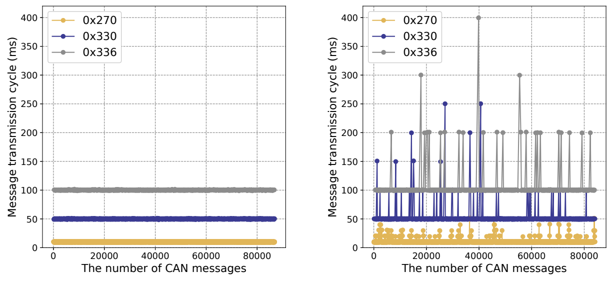
5.4.1 Evaluation of response to an ECU reset

Each time an ECU's CAN controller is reset, there is a notable change in the message transmission cycle of the reset ECU due to the rescheduling of packet transmission. To examine the variation in the message transmission cycle affected by a CAN controller reset, we implemented a CAN controller reset every five seconds using ECU L and compared the message transmission cycle both with and without



(a) w/o CAN controller reset (b) w/ CAN controller reset

Figure 11: Message transmission cycle variations of ECU L (reset off vs. reset on)



(a) Normal mode (b) One-shot mode

Figure 12: Message transmission cycle variations of ECU L (Default mode vs. One-shot mode)

periodic resets.

Fig. 11 illustrates the variations of message transmission cycles in two CAN controller settings: (a) the CAN controller without a reset, and (b) the CAN controller with a periodic reset. As depicted in Fig. 11a, there is no significant change in the transmission cycle of messages sent from the CAN controller without a reset. However, Fig. 11b demonstrates a noticeable change in the transmission cycle of messages sent from the CAN controller with a periodic reset. Hence, we confirmed that an ECU resetting the CAN controller can be identified by using the message transmission cycle variation and the ECU mapping table. However, an attacker who precisely adjusts the timing of the CAN controller reset to match the packet transmission schedule may be difficult to identify by looking at variations in the message transmission cycle. This scenario is discussed further in Appendix A.2.

5.4.2 Evaluation of response to the use of one-shot mode

In this experiment, we compared two test cases: an ECU in normal (default) mode and an ECU in one-shot mode, to observe that messages failing to be transmitted are not retransmitted in the one-shot mode. We injected bit errors into messages with CAN IDs of ECU L in both cases and measured the message transmission cycle. As depicted in Fig. 12, in default mode, there were minimal variations in the message transmission cycle even when bit errors were injected on the ECU. However, for the ECU in one-shot mode, the change in the message transmission cycle was noticeable when bit errors were injected.

Specifically, in the one-shot mode, the messages that have been interrupted by the bit-wise arbitration process are not retransmitted. Therefore, we confirmed that an ECU in one-shot mode can be easily identified by using the message transmission cycle variation and the ECU mapping table.

5.4.3 Evaluation of response to the fast message transmission

When a fast message transmission attack is detected, the RASI module injects an additional bit error into the RID of all ECUs every time the number of injected attack messages becomes a multiple of eight. In this context, the number of additional bit errors, $k_{add}^{\mathrm{ECU}_i}$, for any ECU$_i$ must be calculated using the following equation.

$$k_{add}^{\mathrm{ECU}_i} = \left\lceil \frac{(\mathrm{TDR}_{\mathrm{ECU}_i} \times \Delta T_{\mathrm{ECU}_i})}{8} \right\rceil + \left\lceil \frac{(ATR \times \Delta T_{\mathrm{ECU}_i})}{8} \right\rceil + k_{check}$$

, where $k_{add}^{\mathrm{ECU}_i}$ represents the number of additional bit errors for ECU$_i$, TDR$_{\mathrm{ECU}_i}$ represents the TEC decrease rate of ECU$_i$, $\Delta T_{\mathrm{ECU}_i}$ represents the elapsed time between the initial injection of RIDAS bit errors and the inspection start time of ECU$_i$, and ATR represents the number of attack messages transmitted per unit of time (ms).

Given that $k_{add}^{\mathrm{ECU}i}$ must be at least 1 and less than 16, the maximum time value of ΔTECU$_i$ that satisfies this condition is depicted in Fig. 13. According to Fig. 13, the shortest time, approximately 44.81 ms, is $\Delta T_{\mathrm{ECU}H}$, while the longest time, around 87.85 ms, is ΔTECU$_I$. If the maximum time value of $\Delta T_{\mathrm{ECU}_i}$ is shorter than the start time of inspection for each ECU, the RASI module may not complete its operation regarding the fast transmission attack in one cycle and may need to perform several cycles. For example, as shown in Table 5, ECUs B, D, E, F, G, and H of Vehicle A can be inspected in the first RIDAS operation. If fast message transmission is detected again, the remaining ECUs A, C, and I, which were not covered in the first RIDAS operation, will be inspected in subsequent RIDAS operations.

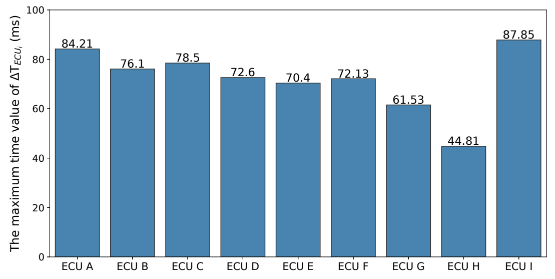


Figure 13: The maximum time value of $\Delta T_{\mathrm{ECU}_i}$ under fast message transmission attacks ($k_{check} = 5, ATR = 1$)

6 Discussion

6.1 Intrusion detection system

Note that the higher the detection accuracy of the IDS, the better the performance of RIDAS's error passive node identification. An IDS that can be integrated with RIDAS must detect attacks before the attack packets are completely transmitted. Thus, existing IDSs such as [4] and [30], which detect attacks based on the transmission period of CAN packets, can be applied to RIDAS. Although research has been conducted on the circumvention of periodic message-based IDSs, RIDAS can distinguish false positives generated by the IDS. In light of this, the advantage of RIDAS is that it can correct an incorrectly detected message as the normal message using the ECU mapping table. However, further research is necessary on IDS based on non-periodic messages in addition to periodic message-based IDS. In future research, we plan to study lightweight IDSs and attempt to integrate them into RIDAS.

6.2 Handling RIDAS's failure

RIDAS's RASI module handles *RIDAS-aware* attackers who are capable of rapidly changing the TEC. Through experiments, we evaluated the performance of the RASI module in detecting attacks using CAN controller reset, one-shot mode setting, and fast message transmission. However, we observed that if a fast transmission attacker is present, the time it takes for the TEC of the attack node to decrease can be significantly shortened, and as a result, the attack node may not be immediately identified by the NASI module. Despite this, such attacks can still be frequently detected by the IDS, and the attack node can then be identified through multiple RIDAS operations. Future research is needed to reduce the operation time of RIDAS either by using the frame trigger or by performing simultaneous inspections on multiple ECUs, rather than inspecting one ECU at a time.

6.3 Other Attacks on RIDAS

Direct TEC manipulation attack. If an attacker can directly modify the value of the TEC register without resetting the CAN controller, RIDAS may be unable to identify the attack node based on the error passive state characteristic. However, in general, the TEC registers of commonly used CAN controllers such as MCP2515, SAM3X8E, SJA1000, STM32F4, and TMS470R1x, are readable but not writable, and thus it is impossible to directly modify the value. Making the TEC register writable would require a hardware redesign. Therefore, assuming an attacker could directly manipulate the TEC register in the current CAN controller environment may be unrealistic.

Advanced one-shot mode attack. If an attacker enables one-shot mode for a single attack packet and then disables it thereafter, RIDAS might not be able to identify the attack source. This is because the RASI module, which identifies one-shot mode attackers based on the retransmission of normal packets, does not observe one-shot mode patterns. However, if the attack packet sent with one-shot mode is accurately detected by an IDS, it will be destroyed by RIDAS's bit error injection and not retransmitted. As a result, the packet from an attacker using an advanced one-shot mode is less likely to be successfully transmitted on the CAN bus.

ID reuse attack. RIDAS is incapable of identifying attack sources that reuse their own CAN ID to inject attack packets. Since attacks that use the CAN IDs of other ECUs are more common than attacks that reuse one's own ID, RIDAS was designed to identify nodes performing masquerade attacks. Nevertheless, ID reuse attacks can also cause significant damage to the vehicle operations. Therefore, further research is required to identify nodes executing ID reuse attacks.

7 Related works

Automotive Intrusion Detection Systems. Automotive IDS has been considered a promising solution to combat cyber-attacks on the CAN bus, as it doesn't necessitate modifications to the vehicle. The IDS's self-adaptation facilitates its direct application to real vehicles, and researchers have extensively studied the implementation of automotive IDS frameworks [12, 20, 25, 35]. The periodicity of CAN messages is the most commonly used characteristic because most CAN messages are periodically transmitted to the CAN network, and the injection of malicious CAN messages significantly increases their transmission frequency [25, 35].

ECU Identification. Several automotive IDS methods have been proposed that offer ECU identification in addition to in-vehicle intrusion detection [6–8, 27]. These methods leverage the unique characteristics of each ECU for identification. If the ECU characteristics previously obtained and the characteristics monitored in the CAN network do not align, it would be considered an anomaly, and ECU identification is performed. Murvay and Groza proposed a method that utilizes the electrical characteristics of CAN messages generated by ECUs [27]. However, they do not account for scenarios where CAN messages are simultaneously transmitted from multiple ECUs, such as during the arbitration phase, which results in mixed characteristics. Choi et al. [7] subsequently proposed a method for ECU identification using electrical characteristics, with arbitration phase consideration. Although this method identifies ECUs with high accuracy, it necessitates an extended CAN frame format to allow for an additional 19-bit extended identifier field.

Recently, Choi et al. [8] and Kneib et al. [16] both proposed improved intrusion detection and identification methods based on the electrical properties of CAN messages. However, all methods utilizing the electrical characteristics of ECUs face the significant issue of device aging. Moreover, these electrical characteristics can be easily affected by various external factors, such as temperature or electromagnetic interference. Instead of leveraging the electrical characteristics, Cho and Shin [3] proposed a method to utilize the unique clock skew of each ECU. Unfortunately, Sagong et al. [31] demonstrated that this method could be evaded by an advanced attacker emulating an ECU clock skew, which ultimately enables the attacker to continue injecting malicious CAN messages undetected. Cho and Shin [5] also proposed a method for identifying a compromised ECU after an attack is detected by IDS. However, despite being a state-of-the-art approach to identifying malicious transmission sources, this method shares the same limitations as the previously mentioned methods, as it is based on the electrical characteristics of individual ECUs.

8 Conclusion

Automotive manufacturers have a practical interest in adopting both intrusion detection and firmware updates to enhance the security of in-vehicle networks. However, automotive IDSs typically cannot identify the compromised ECU even when they successfully detect automotive intrusions. Given this, several attack node identification methods have been proposed. Yet, the proposed methods have shown to be unreliable as the success rate of attack node identification is significantly affected by changes in a vehicle's environment, or the method does not address false positives of the intrusion detection system. Consequently, it is challenging to accurately determine which ECU requires an update in real-time.

To tackle this issue, we propose a new scheme named RIDAS, which identifies the attack source using the *priority reduction* of an ECU's error passive state. Our experiments on a CAN bus prototype and a real vehicle demonstrate that RIDAS is capable of identifying the attack source without impacting driving, and we confirm that RIDAS is robust against changes in a vehicle's environment. We believe that attack node identification through RIDAS is an important advancement in bridging the gap between attack detection and attack response (e.g., isolation, security patch, digital forensics, etc.).

Acknowledgments

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2021R1C1C1011960, No. 2021R1A4A1029650).

References

[1] Rohit Bhatia, Vireshwar Kumar, Khaled Serag, Z. Berkay Celik, Mathias Payer, and Dongyan Xu. Evading Voltage-Based Intrusion Detection on Automotive CAN. In *Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2021.

[2] 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)*, pages 6–6, San Francisco, CA, 2011. USENIX Association.

[3] 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, Austin, TX, 2016. USENIX Association.

[4] 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, Austin, TX, August 2016. USENIX Association.

[5] 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*, CCS '17, pages 911–927, New York, NY, USA, 2017. ACM.

[6] 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. ACM, 2017.

[7] W. Choi, H. J. Jo, S. Woo, J. Y. Chun, J. Park, and D. H. Lee. Identifying ECUs through Inimitable Characteristics of Signals in Controller Area Networks. *IEEE Transactions on Vehicular Technology*, 67(6):4757 – 4770, Feb. 2018.

[8] W. Choi, K. Joo, H. J. Jo, M. C. Park, and D. H. Lee. VoltageIDS: Low-Level Communication Characteristics for Automotive Intrusion Detection System. *IEEE Transactions on Information Forensics and Security*, 13(8):2114–2129, Aug. 2018.

[9] Evenchick Eric. An Introduction to the CANard Toolkit. *Black Hat Asia (2015)*, 2015.

[10] Ian Foster, Andrew Prudhomme, Karl Koscher, and Stefan Savage. Fast and Vulnerable: A Story of Telematic Failures. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, Washington, D.C., 2015. USENIX Association.

[11] Zhiqiang Lin Haohuang Wen, Qi Alfred Chen. 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)*, Boston, MA, Aug. 2020. USENIX Association.

[12] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. Security threats to automotive can networks–practical examples and selected short-term countermeasures. In *International Conference on Computer Safety, Reliability, and Security*, pages 235–248. Springer, 2008.

[13] ISO ISO. 14229-1: 2013 road vehicles–unified diagnostic services (uds)–part 1: Specification and requirements, 2013.

[14] Hyo Jin Jo, Wonsuk Choi, Seoung Yeop Na, Samuel Woo, and Dong Hoon Lee. Vulnerabilities of Android OS-Based Telematics System. *Wireless Personal Communications*, 92(4):1–20, Feb. 2016.

[15] S. Katragadda, P. J. Darby, A. Roche, and R. Gottumukkala. Detecting Low-Rate Replay-Based Injection Attacks on In-Vehicle Networks. *IEEE Access*, 8:54979–54993, 2020.

[16] Marcel Kneib and Christopher Huth. Scission: Signal characteristic-based sender identification and intrusion detection in automotive networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 787–800, New York, NY, USA, 2018. Association for Computing Machinery.

[17] Marcel Kneib, Oleg Schell, and Christopher Huth. EASI: Edge-Based Sender Identification on Resource-Constrained Platforms for Automotive Networks. In *Network and Distributed Systems Security (NDSS) Symposium 2020*, Feb. 2020.

[18] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental Security Analysis of a Modern Automobile. In *IEEE Symposium on Security and Privacy 2010 (IEEE S&P 2010)*, pages 447–462, May 2010.

[19] Sekar Kulandaivel, Tushar Goyal, Arnav Kumar Agrawal, and Vyas Sekar. CANvas: Fast and inexpensive automotive network mapping. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 389–405, Santa Clara, CA, August 2019. USENIX Association.

[20] Congli Ling and Dongqin Feng. An algorithm for detection of malicious messages on can buses. In *2012 National Conference on Information Technology and Computer Science*. Atlantis Press, 2012.

[21] S. Longari, D. H. N. Valcarcel, M. Zago, M. Carminati, and S. Zanero. CANnolo: An Anomaly Detection System based on LSTM Autoencoders for Controller Area Network. *IEEE Transactions on Network and Service Management*, pages 1–1, 2020.

[22] Amit Kr Mandal, Federica Panarotto, Agostino Cortesi, Pietro Ferrara, and Fausto Spoto. Static analysis of Android Auto infotainment and on-board diagnostics II apps. *Software: Practice and Experience*, 49(7):1131–1161, 2019.

[23] Sahar Mazloom, Mohammad Rezaeirad, Aaron Hunter, and Damon McCoy. A Security Analysis of an In-Vehicle Infotainment and App Platform. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, Austin, TX, 2016. USENIX Association.

[24] Charlie Miller and Chris Valasek. Remote Exploition of an Unaltered Passenger Vehicle. *Black Hat USA 2015*, Aug. 2015.

[25] Michael R Moore, Robert A Bridges, Frank L Combs, Michael S Starr, and Stacy J Prowell. Modeling inter-signal arrival times for accurate detection of can bus signal injection attacks: a data-driven approach to in-vehicle intrusion detection. In *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*, pages 1–4. ACM, 2017.

[26] P. Murvay and B. Groza. TIDAL-CAN: Differential Timing Based Intrusion Detection and Localization for Controller Area Network. *IEEE Access*, 8:68895–68912, 2020.

[27] Pal-Stefan Murvay and Bogdan Groza. Source identification using signal characteristics in controller area networks. *IEEE Signal Processing Letters*, 21(4):395–399, 2014.

[28] Sen Nie, Ling Liu, and Yuefeng Du. Free-Fall: Hacking Tesla from Wireless to CAN Bus. In *Blackhat USA 2017*, Jul. 2017.

[29] H. Olufowobi, C. Young, J. Zambreno, and G. Bloom. SAIDuCANT: Specification-Based Automotive Intrusion Detection Using Controller Area Network (CAN) Timing. *IEEE Transactions on Vehicular Technology*, 69(2):1484–1494, 2020.

[30] Habeeb Olufowobi, Clinton Young, Joseph Zambreno, and Gedare Bloom. Saiducant: Specification-based automotive intrusion detection using controller area network (can) timing. *IEEE Transactions on Vehicular Technology*, 69(2):1484–1494, 2020.

[31] Sang Uk Sagong, Xuhang Ying, Andrew Clark, Linda Bushnell, and Radha Poovendran. Cloaking the clock: emulating clock skew in controller area networks. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 32–42. IEEE Press, 2018.

[32] Khaled Serag, Rohit Bhatia, Vireshwar Kumar, Z Berkay Celik, and Dongyan Xu. Exposing new vulnerabilities of error handling mechanism in {CAN}. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 4241–4258, 2021.

[33] Hyun Min Song, Jiyoung Woo, and Huy Kang Kim. In-vehicle network intrusion detection using deep convolutional neural network. *Vehicular Communications*, 21:1–13, Jan. 2020.

[34] Shahroz Tariq, Sangyup Lee, Huy Kang Kim, and Simon S. Woo. CAN-ADF: The controller area network attack detection framework. *Computers Security*, 94:101857, 2020.

[35] 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.

[36] Chris Valasek and Charlie Miller. Adventures in automotive networks and control units. *Defcon 21*, 2013.

[37] Haohuang Wen, Qingchuan Zhao, Qi Chen, and Zhiqiang Lin. Automated Cross-Platform Reverse Engineering of CAN Bus Commands From Mobile Apps. In *Network and Distributed Systems Security (NDSS) Symposium 2020*, Feb. 2020.

[38] S. Woo, H.J. Jo, and D.H. Lee. A Practical Wireless Attack on the Connected Car and Security Protocol for In-Vehicle CAN. *Intelligent Transportation Systems, IEEE Transactions on*, 16(2):993–1006, Apr. 2015.

A Appendix

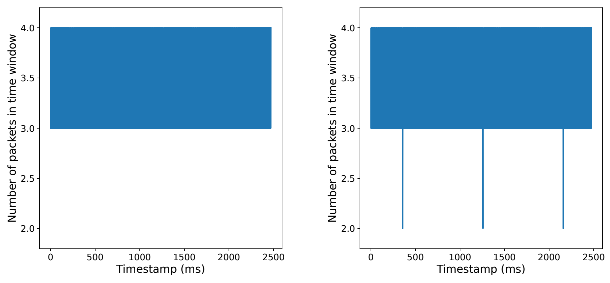
A.1 TEC emulation algorithm

The transmit error counter (TEC) emulation algorithm shown in Algorithm 1 is explained as follows. Initially, the TEC emulator creates a virtual TEC table (V) to emulate the TEC values of all ECUs, setting all TECs to zero. Next, mask and match values are generated to distinguish the four types of bitstreams: arbitration field, active error frame, passive error frame, and EOF. For instance, if a received bitstream is 16 bits, the match value should be set as 0x17FF to check

Algorithm 1 Algorithm for the TEC Emulation module

```
function EMULATOR($T$: an ECU mapping table of a target vehicle)
    $I_S, I_C \leftarrow$ mask and match for the arbitration field
    $E_S, E_C \leftarrow$ mask and match for the end of frame
    $A_S, A_C \leftarrow$ mask and match for the active error frame
    $P_S, P_C \leftarrow$ mask and match for the passive error frame
    $V \leftarrow$ a virtual TEC table for emulating TEC of ECUs
    $arb\_id, idx, cnt \leftarrow$ local variables in TEC Emulator
    while true do
        $B \leftarrow read\_bitstream()$
        // Checking what bitstream is
        if $B \wedge I_S = I_C$ then
            $arb\_id \leftarrow destuff\_identifier(B)$
            // index of ECU with $arb\_id$ in $T$
            $idx \leftarrow find(arb\_id, T)$
        else if $B \wedge A_S = A_C$ or $B \wedge P_S = P_C$ then
            $cnt \leftarrow cnt + 1$
        else if $B \wedge E_S = E_C$ then
            if $cnt = k_{init}$ then
                // ALL: index of all ECUs
                $V[ALL] \leftarrow V[ALL] + (k_{init} \times 8) - 1$
            else
                $V[idx] \leftarrow V[idx] + (cnt \times 8) - 1$
            $cnt \leftarrow 0$
        // RIDAS ends with
        // when a node is identified
        if NASI's identification signal then
            $N \leftarrow index\_of\_compromised\_node()$
            $notify\_to\_AH(N)$
            // ALL-N: index except for $N$
            $V[ALL - N] \leftarrow V[ALL - N] - (k_{init} \times 8)$
        // when fast transmission is detected
        if RASI's fast transmission signal then
            $notify\_to\_AH(ALL)$
            $V[ALL] \leftarrow 0$
        // when an ECU reset is detected
        if RASI's reset signal then
            $N \leftarrow index\_of\_compromised\_node()$
            $V[N] \leftarrow 0$
```

whether the bitstream is end of frame (EOF). The reason is that when an ECU has successfully received a data frame, the value from the CRC delimiter to the inter-frame space (IFS) will be 0x17FF because the acknowledge (ACK) slot is set to dominant. Additionally, the mask value is set to 0x1FFF because the match value must be obtained by AND operating the bitstream with the mask value. Consequently, the TEC emulator can determine the received bitstream's type using the mask and match values. If the received bitstream is an arbitration field, the TEC emulator extracts the identifier (arb_id) from the bitstream and locates the index (idx) of the ECU that owns the identifier using the ECU mapping table (T). If the bitstream is an active or passive error frame, the emulator counts the number of error frames (cnt). If the bitstream is an EOF, the emulator increases the TEC of the ECU, which is idx in V, by $(cnt \times 8) - 1$. However, if cnt equals k_{init} (Section 4.4), the TEC of all ECUs in V is increased by $(k_{init} \times 8) - 1$ because RIDAS cannot identify the compromised ECU before RIDAS inspection.



(a) ECU A without a reset (b) ECU A with reset four times

Figure 14: Identification of ECU A's reset through a remote frame

The TEC emulator is designed to operate continuously, unlike other modules in RIDAS. This allows it to inform the attack handling (AH) module's frame trigger and the native attack source identification (NASI) module's error generator about each ECU's TEC status. the emulator notifies the AH module of the frame trigger's target (Section 4.4) and sets the TEC of V as follows: If the compromised node performed fast message transmission (Section 4.6.3), the emulator sets all ECUs' TEC in V to 0. Otherwise, it decreases the TEC of all ECUs, except for the compromised ECU, by $(k_{init} \times 8)$. Additionally, when the RASI module signals a reset of an ECU, the TEC value of the ECU where the reset occurred is set to 0.

A.2 ECU reset identification issue

In the RASI module of RIDAS, an ECU reset can be identified by monitoring changes in the message transmission cycles. However, it becomes challenging to detect a CAN controller's reset by variations in the message transmission cycle when dealing with an attacker who accurately adjusts the timing of CAN controller resets according to the packet transmission schedule.

To address this issue, RIDAS can incorporate the use of a frame trigger by sending a remote frame or a UDS request message to prompt CAN message transmission to a specific ECU. Consequently, if RIDAS's traffic generator uses the remote frame or UDS request message as background traffic, additional CAN messages can be stimulated. To evaluate the identification experiment of the CAN controller's reset using the frame trigger, we used the remote frame of ECU A. In this experiment, the remote frame of ECU A was set to be transmitted at a cycle of 5 ms, and the number of data frames generated by ECU A was counted for each time window period (e.g., 15 ms in this case). As depicted in Fig. 14, when there was no CAN controller reset, approximately three or four CAN messages from ECU A were observed during one time window. However, when resets were performed four times, windows containing only two CAN messages were observed three times.

Therefore, it was confirmed that the frame trigger can be utilized to identify an ECU with a CAN controller reset. Especially, if the frame trigger randomly schedules the transmission time of the triggering message, an attacker may not be able to respond to that message due to a CAN controller's reset, since the attacker cannot predict the random timing of the triggering message transmission.

A.3 RIDAS's effect on normal CAN messages

RIDAS could potentially influence the message transmission cycle of normal CAN messages, given that it generates additional traffic via the traffic generator and error generators to identify the compromised ECU. To evaluate RIDAS's impact on the message transmission cycle of normal CAN messages, we measured the transmission time for all messages both with and without RIDAS. As indicated in Table 6, we observe that the difference in the message transmission cycle variation with and without RIDAS is extremely minimal. Interestingly, we noticed the variation in the message transmission cycle of CAN packets even under normal conditions without RIDAS. This confirmed that RIDAS exerts only a minor effect on a normal message's transmission cycle.

Table 6: RIDAS's effect on normal CAN messages

ECU	CAN ID	Transmission Time (ms)					
		Without RIDAS			With RIDAS		
		min	max	mean	min	max	mean
A	0x48A	46.51	53.50	49.99	45.77	54.09	49.99
	0x48C	196.50	203.17	199.99	194.68	205.37	199.99
	0x58B	44.77	55.22	49.99	43.68	56.51	49.99
B	0x340	8.53	11.46	10.00	8.52	11.45	10.00
	0x42D	97.93	101.71	100.01	97.09	102.92	100.01
	0x484	66.26	73.48	70.00	66.29	73.91	70.00
	0x485	46.50	53.51	50.00	46.03	53.99	50.00
	0x4A7	496.89	502.18	500.05	496.41	503.62	500.05
	0x53E	96.65	103.46	100.01	95.31	104.78	100.01
C	0x391	17.40	22.59	19.98	16.81	23.12	19.98
	0x4F1	14.85	24.38	19.98	13.84	26.09	19.98
	0x50C	95.52	103.95	99.94	93.61	104.83	99.93
	0x52A	39.70	204.15	198.57	194.54	202.98	199.87
	0x5B0	995.38	1002.37	999.42	993.79	1002.88	999.38
	0x5CD	195.10	204.29	199.88	194.00	203.82	199.87
D	0x389	15.73	24.26	20.00	15.19	24.73	20.00
	0x38D	17.66	22.35	20.00	17.39	22.90	20.00
	0x420	16.14	23.85	20.00	15.67	24.26	20.00
	0x421	15.91	24.08	20.00	16.57	23.42	20.00
	0x483	197.45	202.36	200.02	196.49	203.58	200.02
	0x4A2	496.99	502.95	500.05	496.26	504.00	500.05
	0x50A	196.89	203.02	200.02	196.30	203.67	200.02
E	0x251	8.55	11.44	10.00	9.00	11.00	10.00
	0x2B0	8.55	11.45	10.00	8.40	11.50	10.00
	0x381	17.96	22.03	20.00	17.61	22.31	20.00
F	0x130	9.50	10.46	10.00	9.44	10.57	10.00
	0x140	9.51	10.46	10.00	9.44	10.56	10.00
	0x495	95.83	104.33	100.06	94.87	104.80	100.06
	0x500	95.96	104.01	100.06	95.30	105.41	100.06
G	0x153	9.10	10.93	10.00	9.05	10.92	10.00
	0x164	9.03	10.95	10.00	9.03	10.95	10.00
	0x220	8.84	11.16	10.00	8.80	11.20	10.00
	0x394	17.61	22.29	20.00	16.51	23.55	20.00
	0x47F	17.18	22.42	20.00	15.63	24.35	20.00
	0x490	45.42	54.56	50.00	45.64	54.32	50.00
	0x507	98.81	101.59	100.00	95.92	104.09	100.00
H	0x043	995.92	1002.06	999.34	996.94	1001.10	999.29
	0x07F	998.88	1000.76	999.99	998.71	1001.05	999.99
	0x260	6.56	13.58	10.00	5.69	14.42	10.00
	0x329	5.79	14.15	10.00	6.15	13.86	10.00
	0x356	7.52	12.48	9.99	8.40	11.77	10.00
	0x366	5.82	13.92	9.99	5.70	14.40	9.99
	0x367	6.86	13.41	9.99	7.05	12.96	9.99
	0x368	7.43	12.49	9.99	7.68	12.42	9.99
	0x386	16.46	23.78	20.00	16.84	23.05	19.99
	0x387	15.70	24.19	20.00	15.61	25.08	20.00
	0x410	198.93	200.85	199.99	198.30	201.57	199.99
	0x412	198.61	201.49	199.99	195.15	204.89	199.99
	0x436	47.03	53.04	49.99	46.88	53.42	49.99
	0x44E	197.20	202.72	199.99	197.51	202.46	199.99
	0x453	15.88	24.18	19.99	16.64	23.46	19.99
	0x470	15.88	24.18	19.99	16.46	23.43	19.99
	0x479	97.62	102.37	99.99	96.63	103.37	99.99
	0x492	43.56	56.76	50.00	43.45	56.68	50.00
	0x49F	196.17	203.56	199.99	196.89	202.66	199.99
	0x4A9	196.10	204.03	199.99	194.89	205.07	199.99
	0x4C9	197.31	202.85	199.99	197.36	202.06	199.99
	0x4CB	197.83	202.22	199.99	197.07	202.25	199.99
	0x50E	196.95	203.07	199.99	195.69	204.37	199.99
	0x520	95.14	104.87	99.99	95.98	104.06	99.99
	0x53B	197.84	201.78	199.99	197.68	201.99	199.99
	0x53F	195.98	204.13	199.99	197.13	202.76	199.99
	0x541	96.98	102.72	99.99	96.48	103.43	99.99
	0x544	195.63	204.39	199.99	194.96	205.16	199.99
	0x553	196.85	203.21	199.99	196.51	203.49	199.99
	0x559	197.20	202.74	200.00	196.52	203.21	199.99
	0x568	95.49	104.50	99.99	94.91	104.98	99.99
	0x572	196.12	203.74	199.99	196.66	203.04	199.99
	0x57F	1994.93	2004.87	1999.99	1995.76	2003.18	1999.98
	0x593	194.75	208.82	199.99	190.69	209.66	199.99
	0x5A6	196.03	203.75	199.99	196.64	202.57	199.99
	0x5BE	997.79	1001.86	999.99	996.18	1002.00	999.99
I	0x563	595.16	604.58	600.00	596.23	603.72	600.00

Table 7: An ECU mapping table of the traffic generator

ECU	CAN ID	Transmission Time (ms)
		mean
J	0x600	1
	0x601	1
	0x602	1
K	0x603	1
	0x604	1
	0x605	1

Table 8: An ECU mapping table of a compromised ECU

ECU	CAN ID	Transmission Time (ms)
L	0x270	10
	0x330	50
	0x336	100