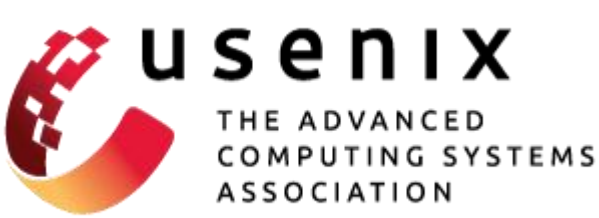


浙江大学
ZHEJIANG UNIVERSITY

Harness: Transparent and Lightweight Protection of Vehicle Control on Untrusted Android Automotive Operating System

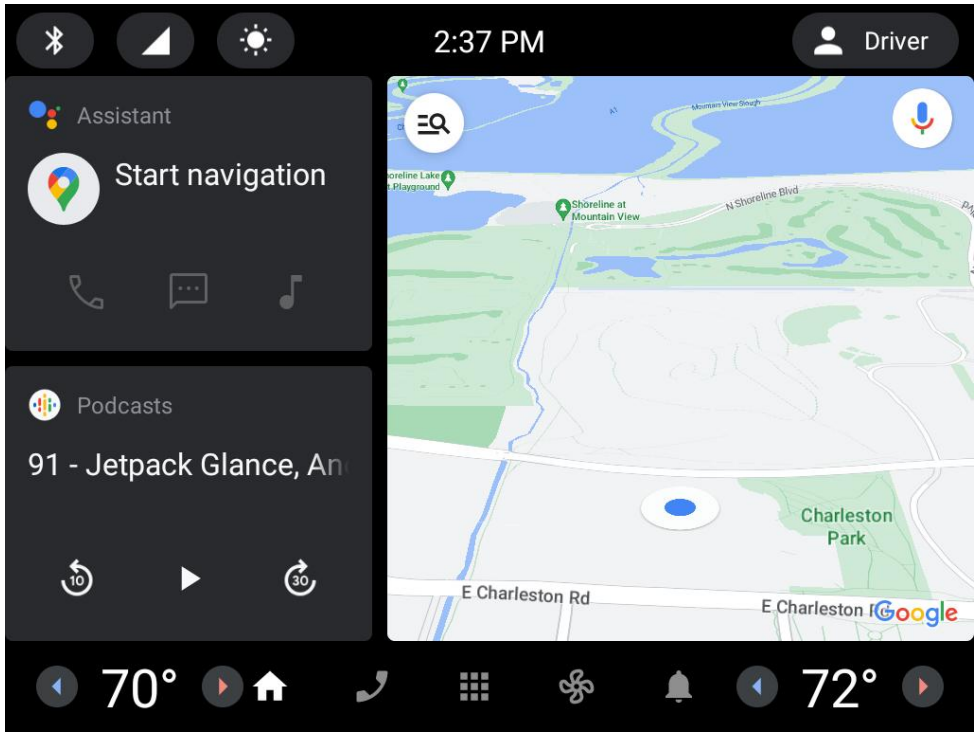
Haochen Gong, Siyu Hong, Shenyi Yang, Rui Chang,
Wenbo Shen, Ziqi Yuan, Chenyang Yu, and Yajin Zhou

Zhejiang University

Background

Modern in-vehicle infotainment (IVI) systems become more feature-rich

- For example, Android Automotive Operating System (AAOS)

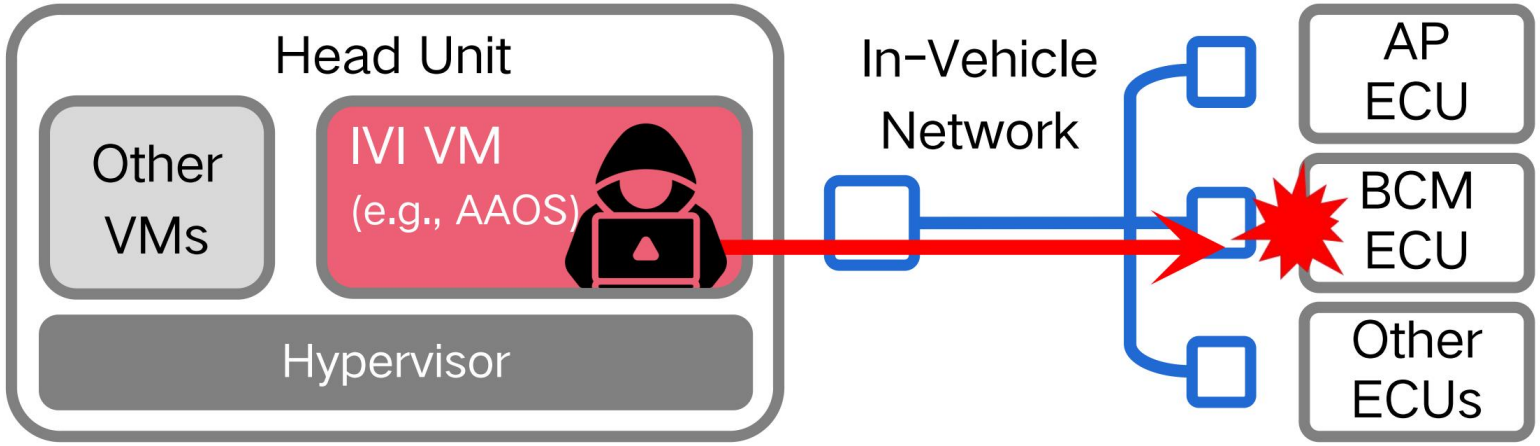


Android Automotive OS
https://developer.android.com/training/cars/platforms/automotive-os

- Touch screen, voice assistant
- Wireless connectivity (e.g., bluetooth)
- Third-party apps (Android ecosystem)
- Vehicle control (e.g., seats, locks, AC)

Background

But what's the cost?



Security Issues

- Android's rich functionality and complexity expand the attack surface and bloat the trusted computing base (TCB)
- IVIs are often connected to the vehicle's electronic control units (ECUs) through the in-vehicle network (IVN)

Attacks on the IVI OS can be the gateway to full vehicle control

- e.g. door lock, window/seat position, gear shift, parking break

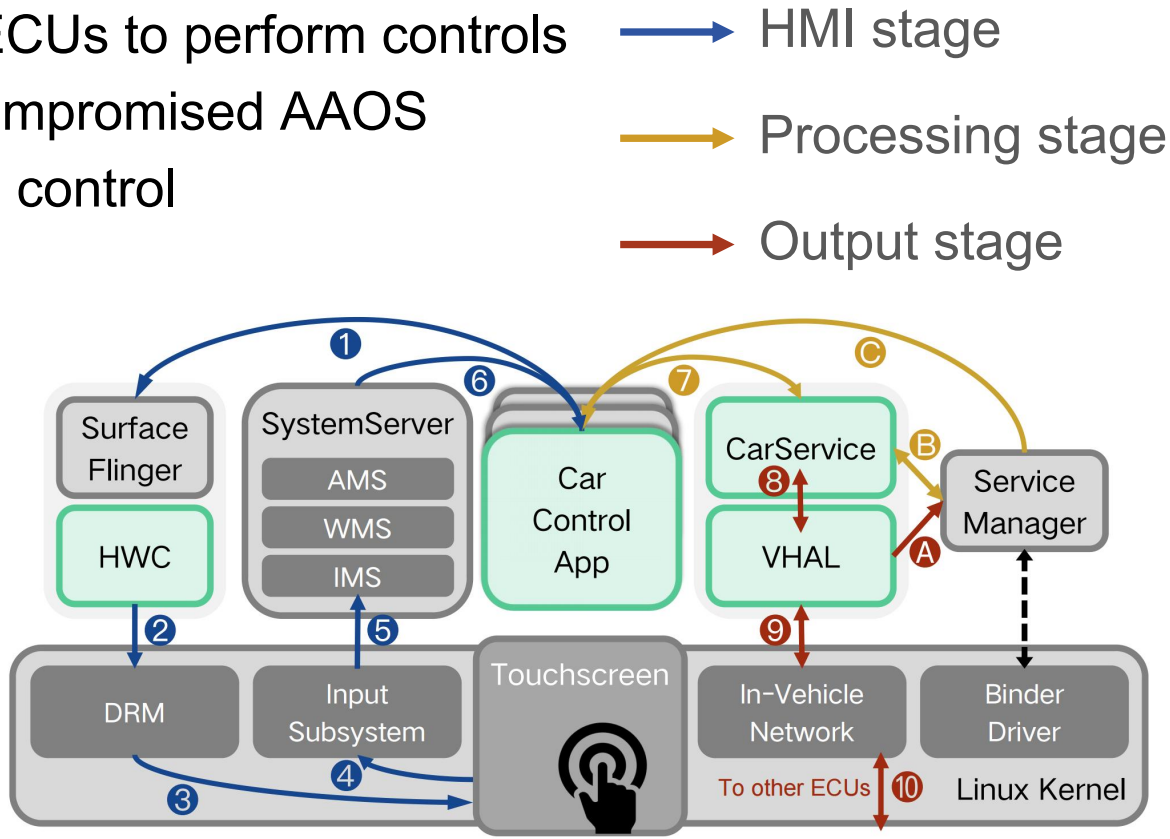
Vehicle Control Chain and Attack Surface

The vehicle control chain can be divided into three stages

- **HMI stage**: User interacts with car apps via touchscreen
- **Processing stage**: Car apps call the car APIs of the CarService
- **Output stage**: The VHAL communicates with ECUs to perform controls

We identified 13 potential attack strategies on a compromised AAOS — any one of them could cause the vehicle to lose control

Adversary	Attack Strategy	Affected Nodes/Steps	Requirements
Guest kernel	**A1**: Illegal register access	All processes	R2
	A2: Illegal page mapping	All processes	R2, R3, R4
	A3: Invalid syscall return (Iago)	All processes	R2, R3, R4
	A4: Device driver hijacking	All processes (I/O, IPC)	R3, R4
User process	**A5**: Service forging	①, ⑥, ⑦, ⑧, Ⓐ, Ⓑ, Ⓒ	R3, R4
	A6: Malicious data dispatching	①, ⑥, ⑦, ⑧, Ⓐ, Ⓑ, Ⓒ	R3, R4
Both	**A7**: Illegal memory access	All processes	R2, R3, R4
	A8: Unauthorized syscall	②, ⑨	R4, R1
	A9: Executable file tampering	All processes	R2
	A10: Config file tampering	CarService, VHAL	R2
	HA1: Unauthorized API call	⑦, ⑧	R3
	HA2: GUI confusion	①, ②, ③	R4
	HA3: Touch input forging	④, ⑤, ⑥	R4

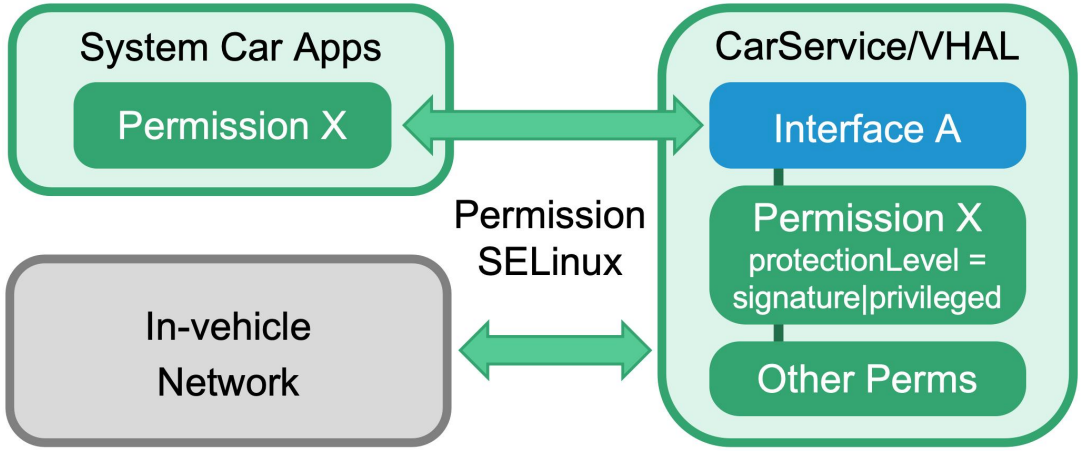


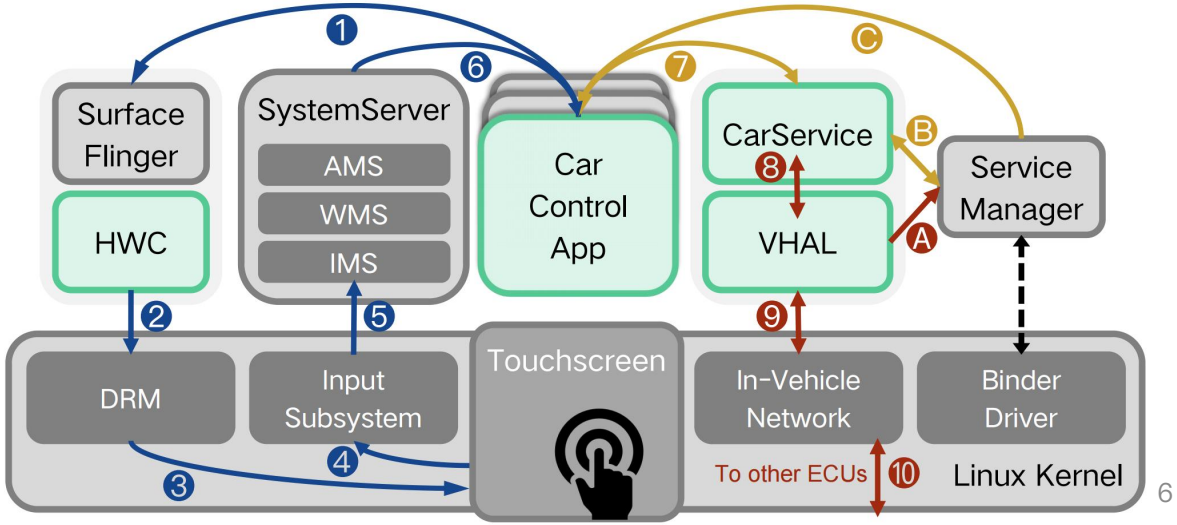
Threat Model & Assumption

- AAOS (kernel & system services) and third-party apps are untrusted
  - The attacker can compromise the AAOS
- The hardware, hypervisor, and system car components are trusted
  - The fundamental components for vehicle control
- Assume the hypervisor prevents DMA attacks
- Assume the guest OS can boot securely
- Attacks against AI components/remote control are out of scope
- Physical/side-channel/DoS attacks are out of scope

Observation & Insight

- AAOS explicitly defines **security-critical car interfaces** through the *protectionLevel* attribute in its permission system
  - Accessible **only** by **trusted components** (system apps/services)
- This defines a **clear security boundary**, and we can:
  - **Isolate** car-related trusted **components**
  - **Protect** their inter-process/IVN **communications**
  - **Restrict** external access to their **interfaces**





Design Goals

Key idea: Reduce the TCB for vehicle control

- **Isolate** car-related **trusted components** from AAOS
- **Protect** their inter-process/IVN **communications**
- **Restrict** external access to their **interfaces**

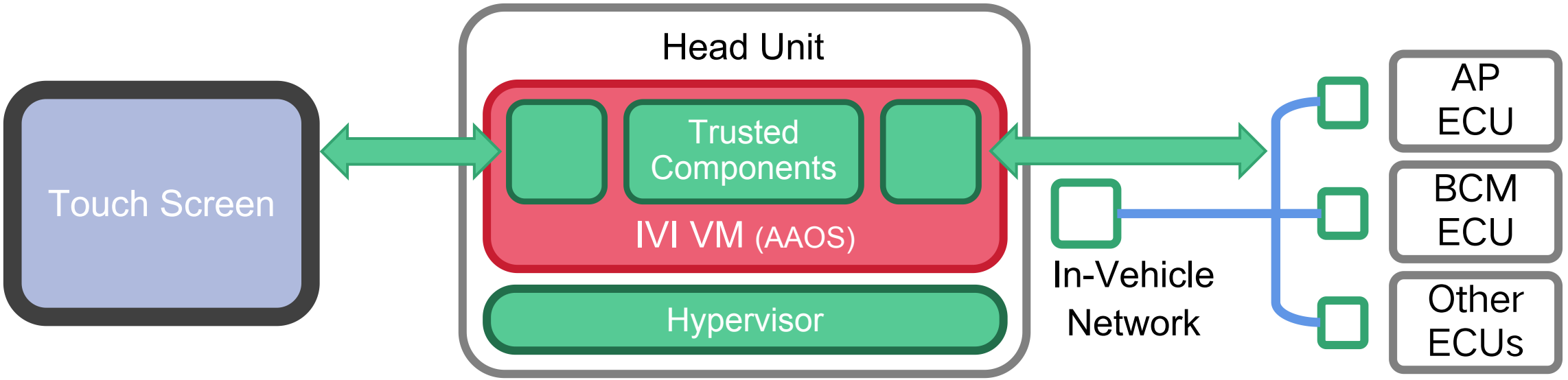
Security	Efficiency (lightweight)	Usability (transparent)
The attacker is prevented from performing vehicle control through the software outside the isolation domain	The performance overhead of the solution should be acceptable; The impact on unprotected software should be negligible	The solution should be easy to deploy and should maintain a seamless user experience

Harness Overview

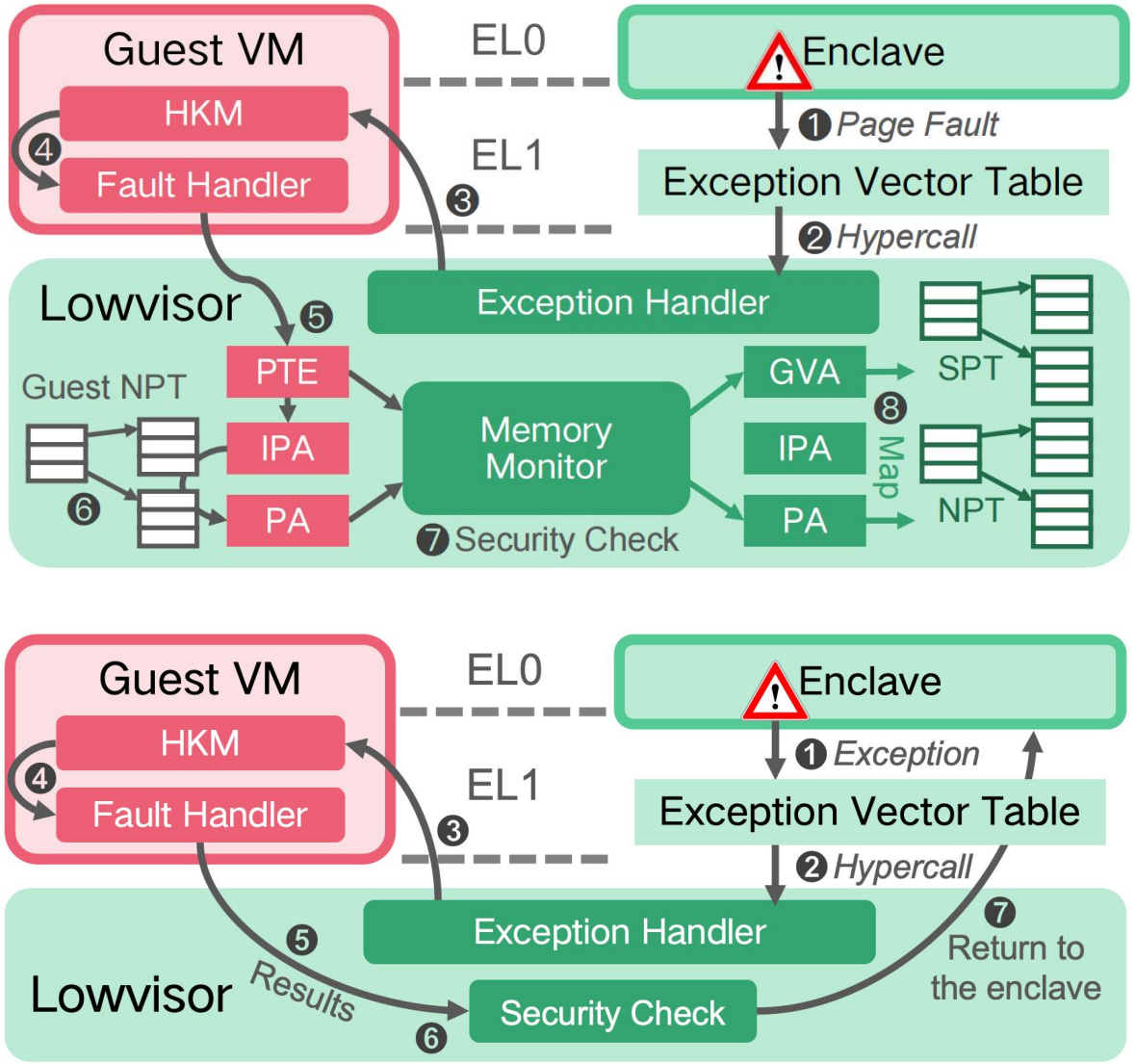
Harness ensures that vehicle controls remain firmly in the user's hands

Full-chain protection:
Human Machine Iterface (HMI) ⇒ Userspace Car Components ⇒ In-Vehicle Network

- Isolate trusted components, protect their communications and interfaces



Harness Overview



How to isolate trusted car components?

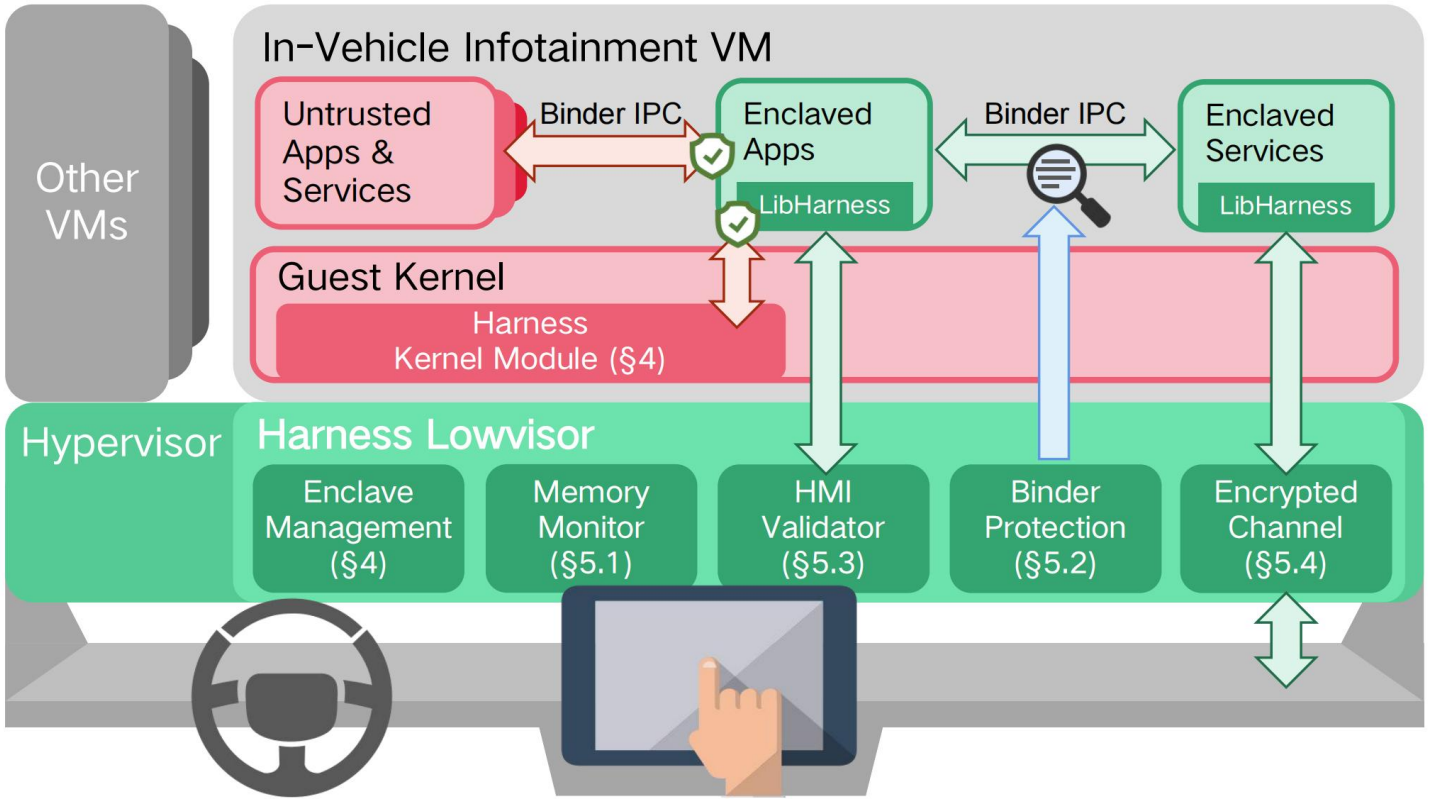
Hardware virtualization-based enclave

- At process granularity
- Stage-2 paging-based memory isolation
- Each enclave has its *own IPA space*
- Enclaves share a protected exception vector table for Harness to intercept

Lowvisor: The core of Harness

- A hypervisor module for managing and protecting enclaves
- Forward some exceptions (e.g., page fault and syscall) to the guest kernel to handle and check the results

Harness Overview



Green: trusted Red: untrusted

Runtime support for enclaves?

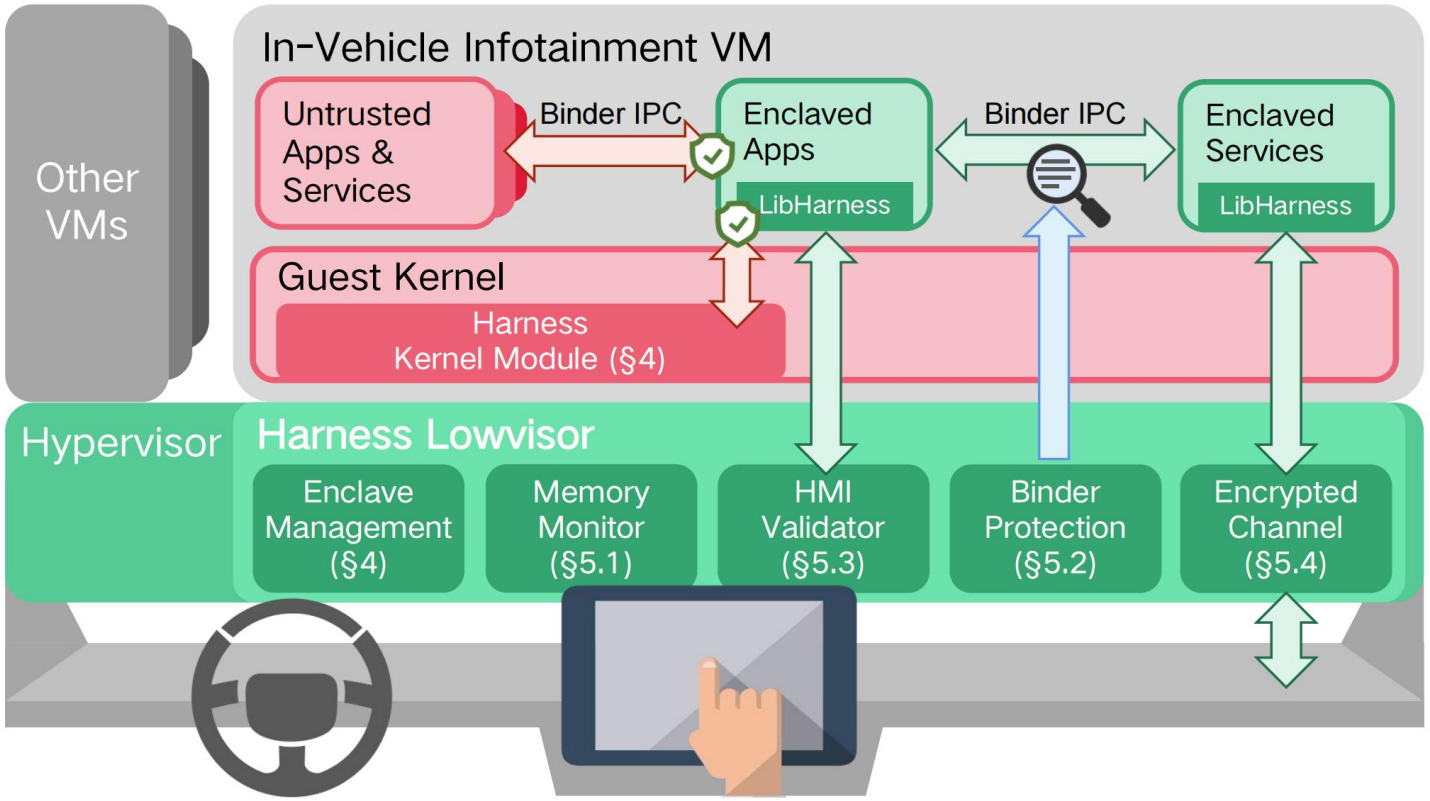
Harness Kernel Module

- Bridge the gap between enclaves and the untrusted kernel

LibHarness

- Make Harness transparent to userspace software developers
  - Support unmodified Android components (the usability goal)

Harness Overview



Green: trusted Red: untrusted

Which components to isolate and protect?

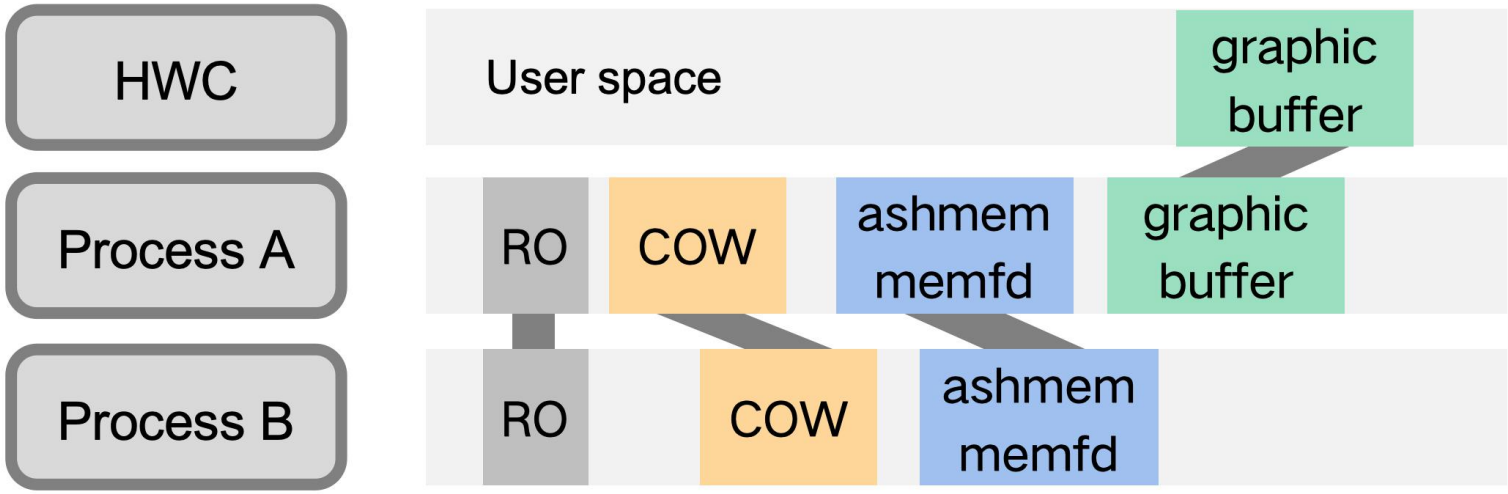
Minimal protection domain

- Trusted userspace components
  - System car apps
  - CarService, VHAL
  - HWC: secure display
  - A new Zygote: fork new enclaves
- Authenticate in-domain components using a whitelist and signatures
- Interface protection
  - IVN: encrypted channel between the VHAL and ECUs
  - API, HMI of enclaved components

Challenges and Key Techniques

C1: Android components heavily rely on shared memory

- Implicit sharing: non-writable/copy-on-write memory
  - e.g., library, file, parent-child process
- Explicit sharing: through syscall (MAP_SHARED) and IPC (fd, handle)
  - e.g., ContentProvider (ashmem, memfd), graphic buffer



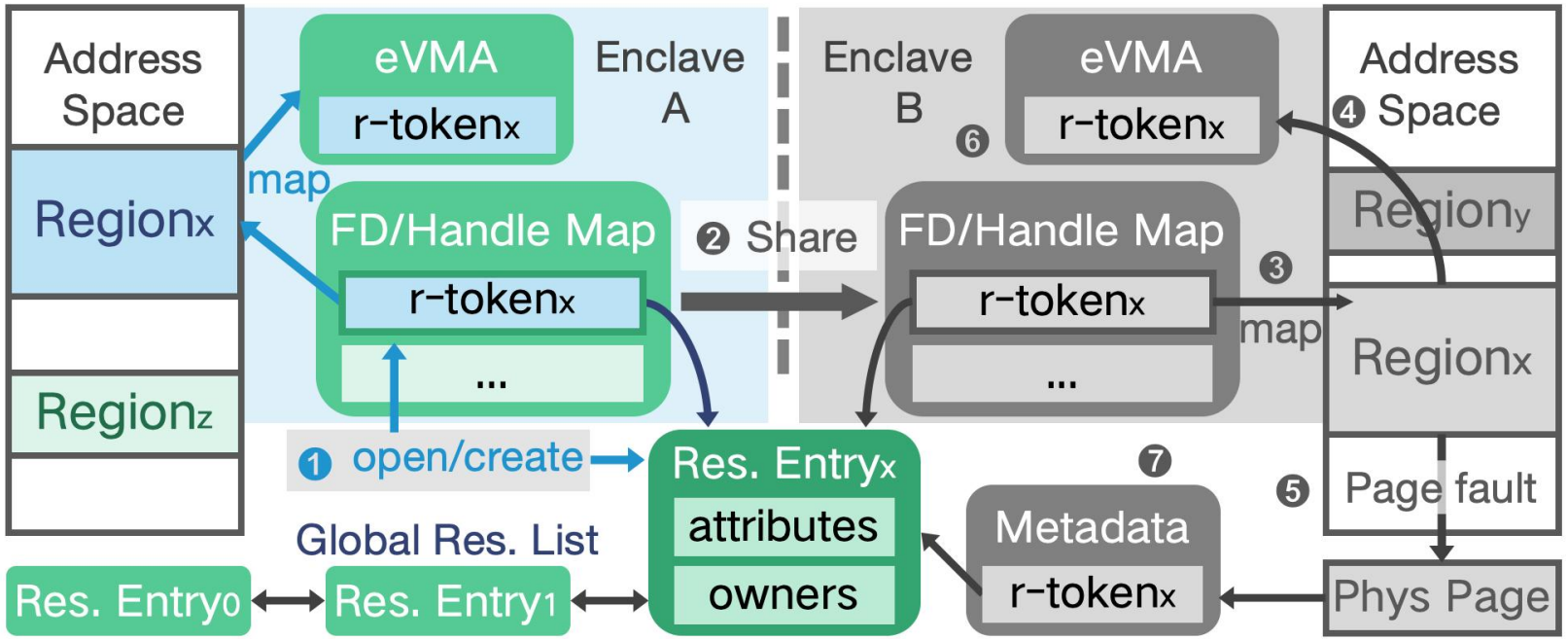
Challenges and Key Techniques

C1: Android components heavily rely on shared memory

How to identify which memory region can be shared?

Sharing-aware memory monitor: a Lowvisor module

- Interpose on enclaves' syscall for memory management
- Record enclaves' intents on memory mapping/sharing



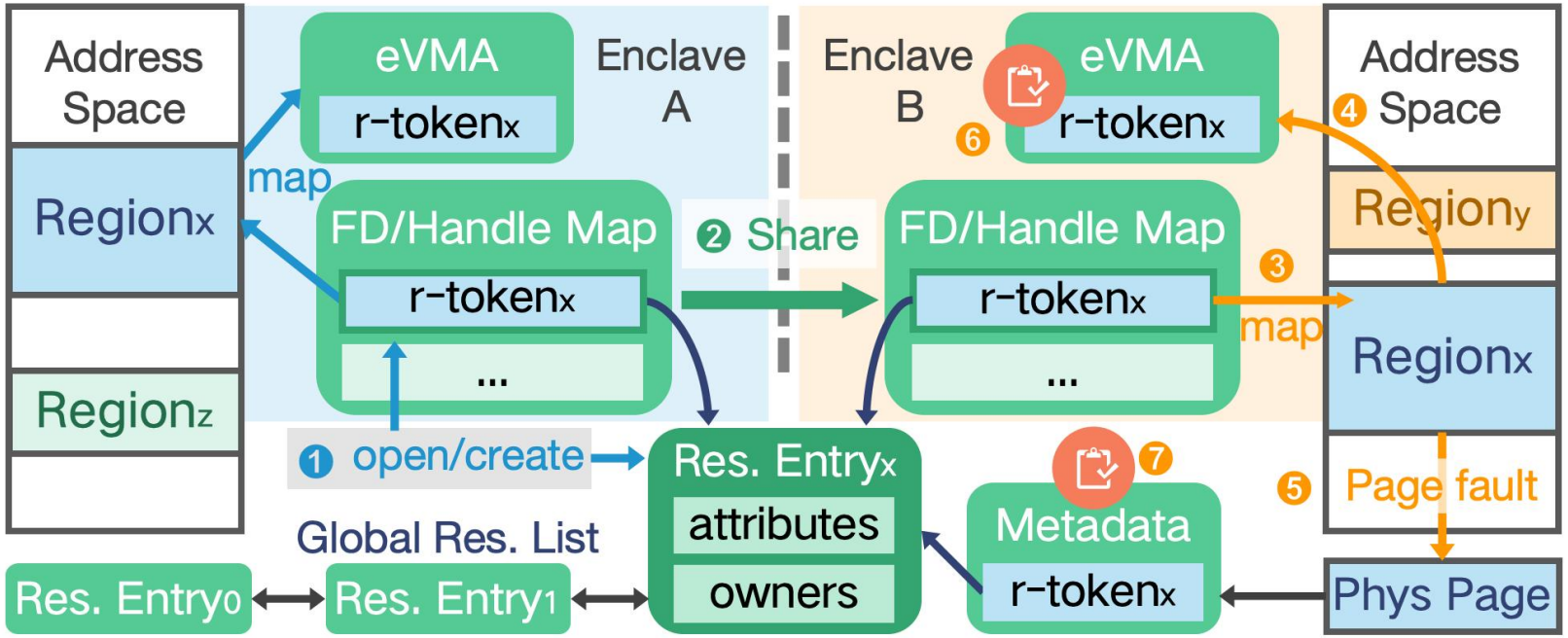
Challenges and Key Techniques

C1: Android components heavily rely on shared memory

How can enclaves share memory securely?

Ownership-based access control via resource tokens

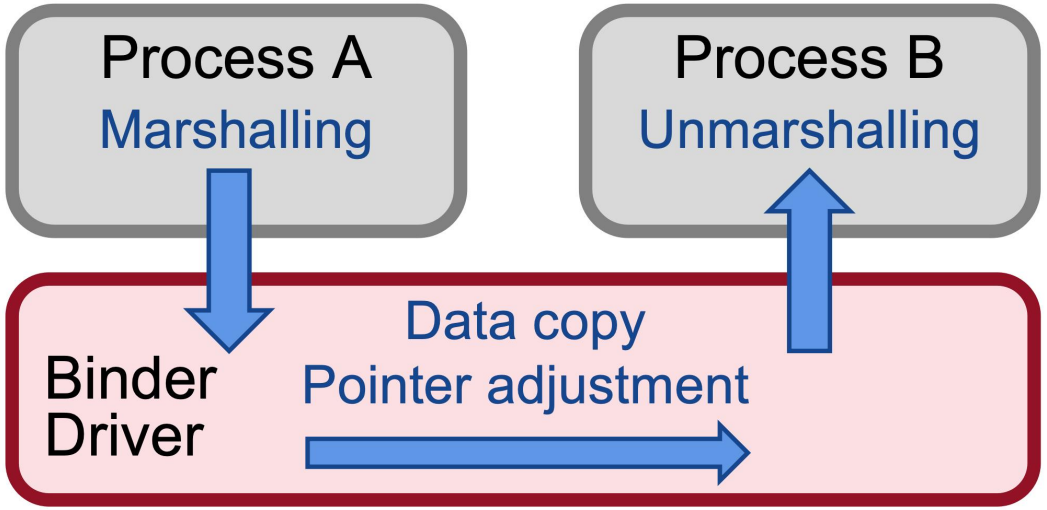
- The owner of a resource can grant the r-token to other enclaves
- Each physical page records the corresponding r-token in metadata



Challenges and Key Techniques

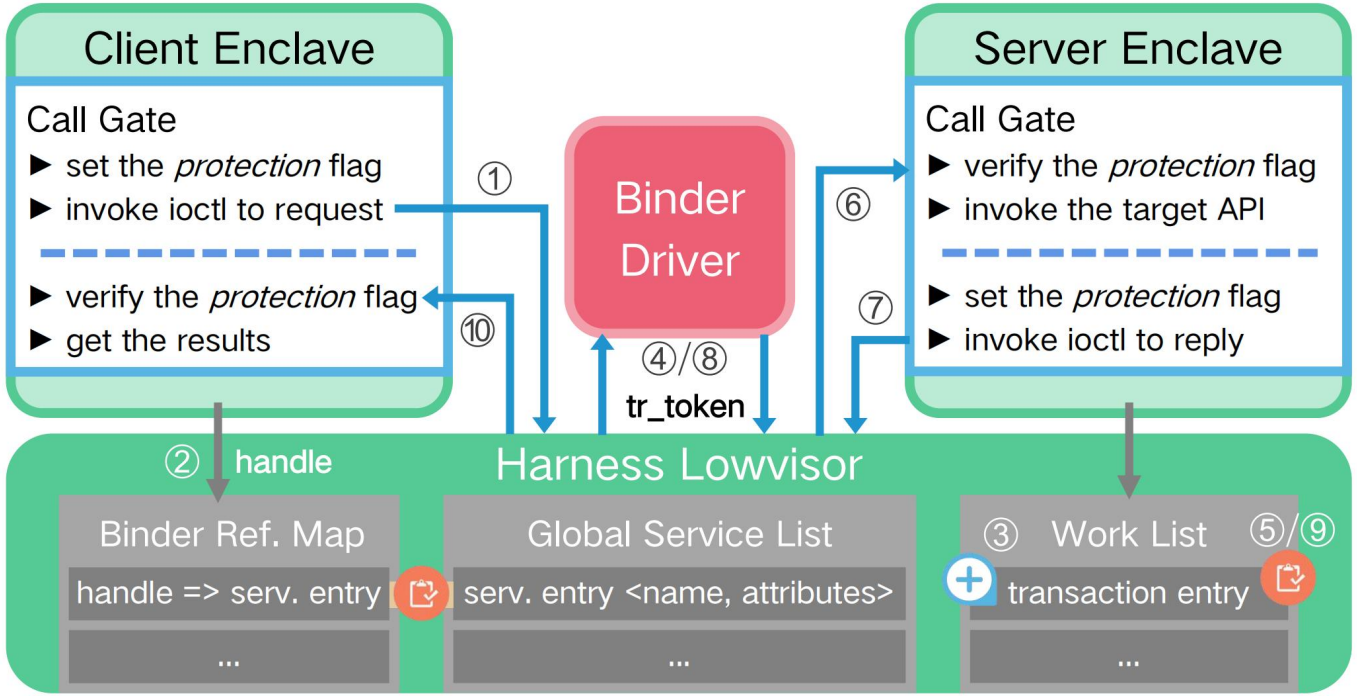
C2: Binder IPC is complex and performance-sensitive

- Binder is an essential mechanism for Android interfaces
  - Frequently used by isolated components
- Binder has a complex software stack
  - Library + kernel driver (untrusted)
    - data serialization
    - data copy
    - pointer adjustment
- Therefore, cryptography-based approach is unsuitable



Challenges and Key Techniques

C2: Binder IPC is complex and performance-sensitive



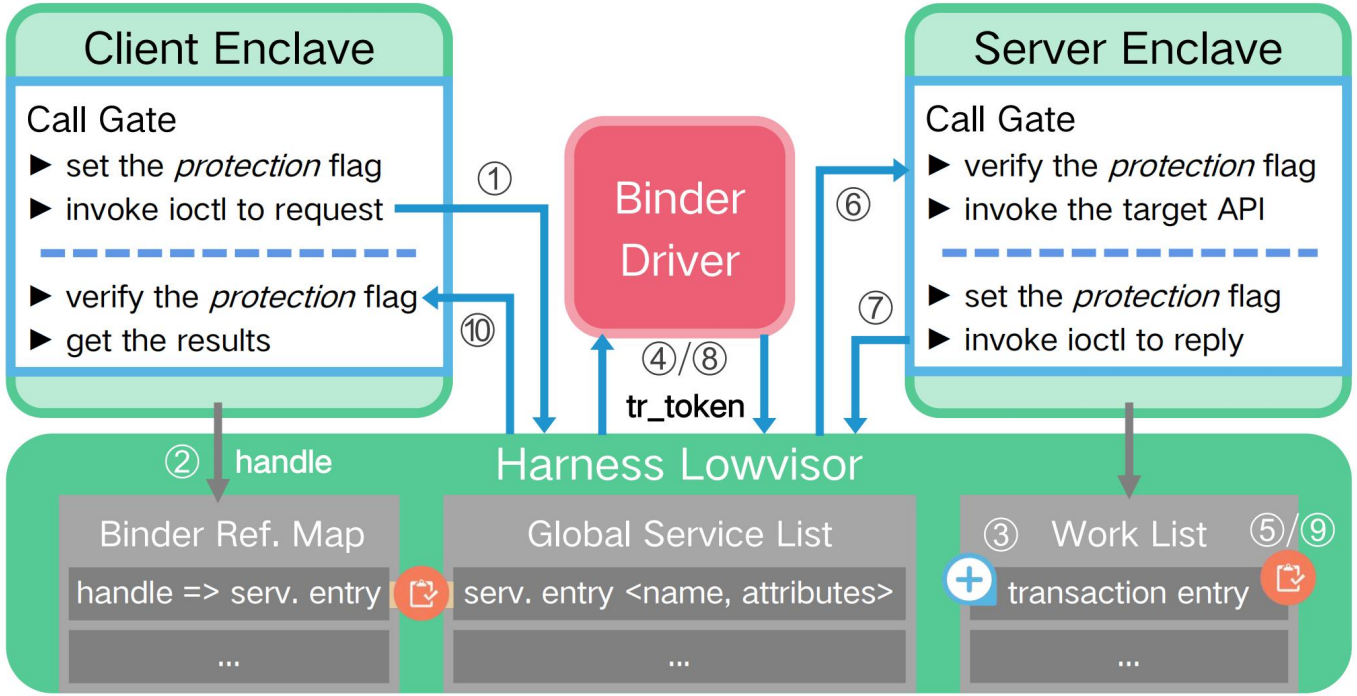
How to protect IPC (content & flow)?

Lowvisor-based Binder transaction tracking and mediation

- Services registered by enclaves
- Services queried by enclaves
- Working transaction
- Data copy + pointer adjustment
  - No longer depend on the driver

Challenges and Key Techniques

C2: Binder IPC is complex and performance-sensitive



To ensure usability, we must allow enclaves to communicate with untrusted components (e.g., SystemServer)

How to prevent malicious IPC?

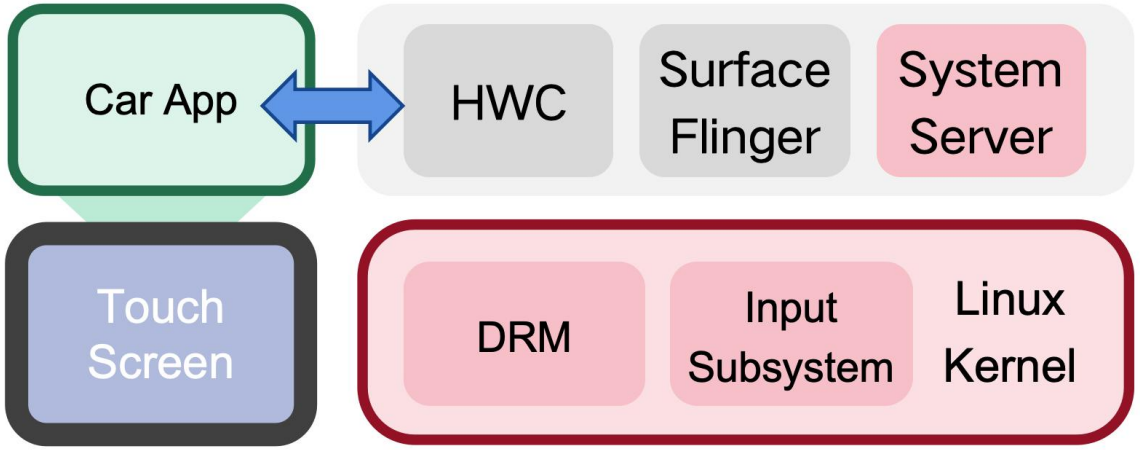
AIDL-based lightweight authentication

- Developer marks sensitive interfaces
- AIDL compiler generates call gates
- Call gates and Lowvisor cooperate to authenticate via protection flags

Challenges and Key Techniques

C3: The HMI is challenging to protect transparently

- Existing approaches impose usability burdens on users and deployment burdens on manufacturers
- The HMI software stack includes kernel and vulnerable system services
  - We should not simply isolate all related software components



Challenges and Key Techniques

C3: The HMI is challenging to protect transparently

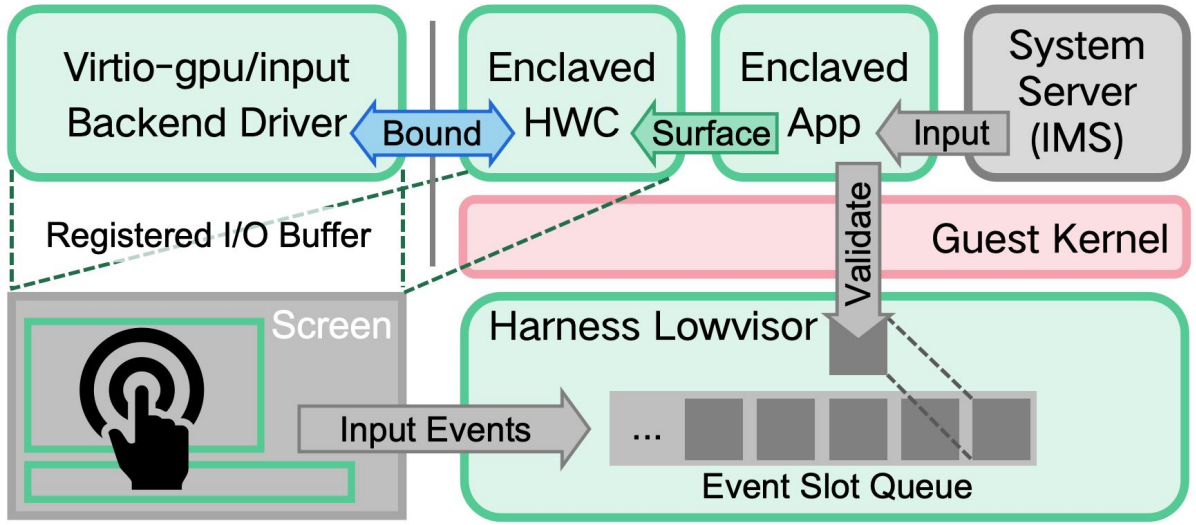
How to protect the GUI of trusted car apps?

Graphic buffer isolation

- Isolate enclaved apps' graphic buffers from the AAOS
- A buffer can only be accessed by the HWC and an enclaved app

Virtio device binding

- Bind virtio devices to enclaves to block malicious I/O control



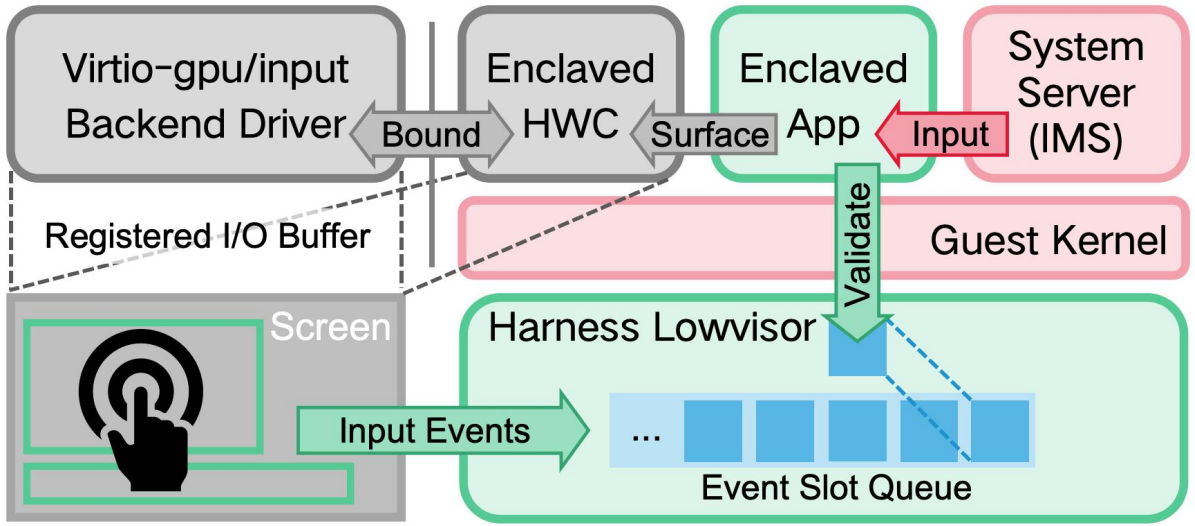
Challenges and Key Techniques

C3: The HMI is challenging to protect transparently

How to protect the touch input of trusted car apps?

Touch input validation

- The processing of touch input events by IMS can be reversed
- Record hardware touch input events targeting an enclaved app
- Compare the events the app received against the recorded ones



Harness Implementation

- Raspberry Pi 5
  - 4 x Cortex A76
  - 8GB RAM
- Google Cuttlefish
  - IVI VM: 4 cores, 6GB
  - android-13.0.0-r41
  - kernel: android13-5.15

Module	Feature	LoC
Harness Lowvisor	Enclave management	1,897
	Syscall/signal support	1,873
	Memory management	950
	Resource monitor	858
	Binder protection	1,213
	HMI validation	248
	Others	484
	Total	7,523
HKM / Guest Kernel	Total / Modification	1,726 /173
LibHarness / AOSP	Total / Modification	79 / 914
Host Kernel	Modification	114

Evaluation: Security

Potential attacks

- 13 attack strategies

Real-world attacks

- 26 CVEs
- 3 attack paths[1]

Attack simulations

- CPU, memory, I/O
- API, IPC, files

Type	CVEs
Privilege Escalation	2016-9337, 2018-9322, 2021-0303, 2021-39738, 2022-20144, 2022-42430, 2022-42431, 2023-20927, 2023-32155
Arbitrary Code Execution	2015-5611, 2019-9977, 2020-28656, 2021-23906, 2021-23907, 2021-23908, 2021-23909, 2023-32156, 2023-32157
Memory Corruption	2022-33300, 2022-40539
Tapjacking/Overlay	2021-0583, 2021-1036, 2021-1039, 2021-1040, 2022-20212, 2022-20214

[1] Jing et al. - Revisiting automotive attack surfaces: a practitioners' perspective. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 80–80. IEEE Computer Society, 2023.

Evaluation: Performance

Micro benchmarks

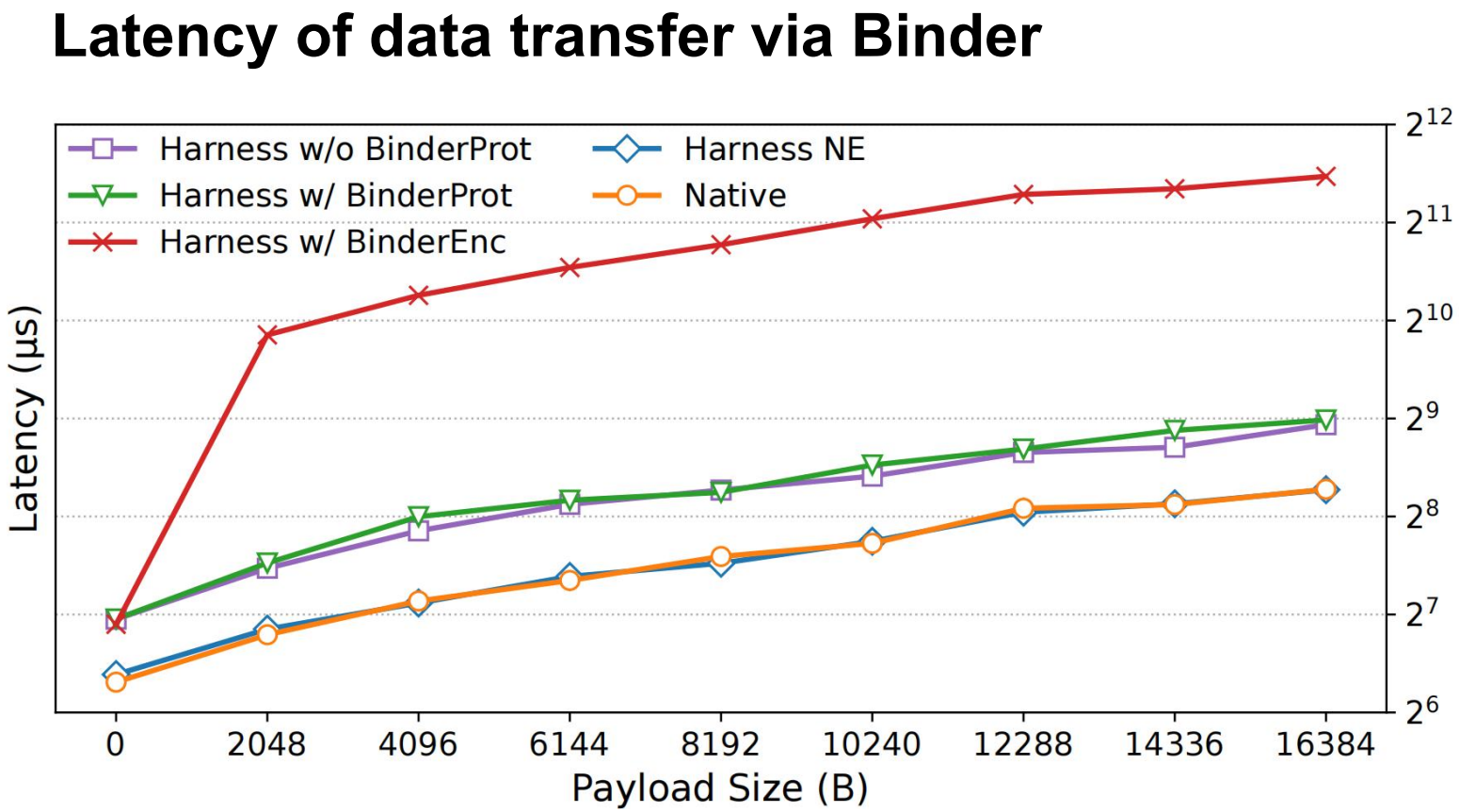
Harness overhead on enclaved components

LMBench	Encl. App Startup (Cold/Hot)	Touch Input (Down/Up)	Memory Consumption
3.68x	13.5/29.7%	10.8/21.9%	< 3.9%

Harness overhead on non-enclaved components

LMBench	Encl. App Startup (Cold/Hot)	Touch Input (Down/Up)	Memory Consumption
1.36%	–2.0/5.8%	–1.36/1.81%	--

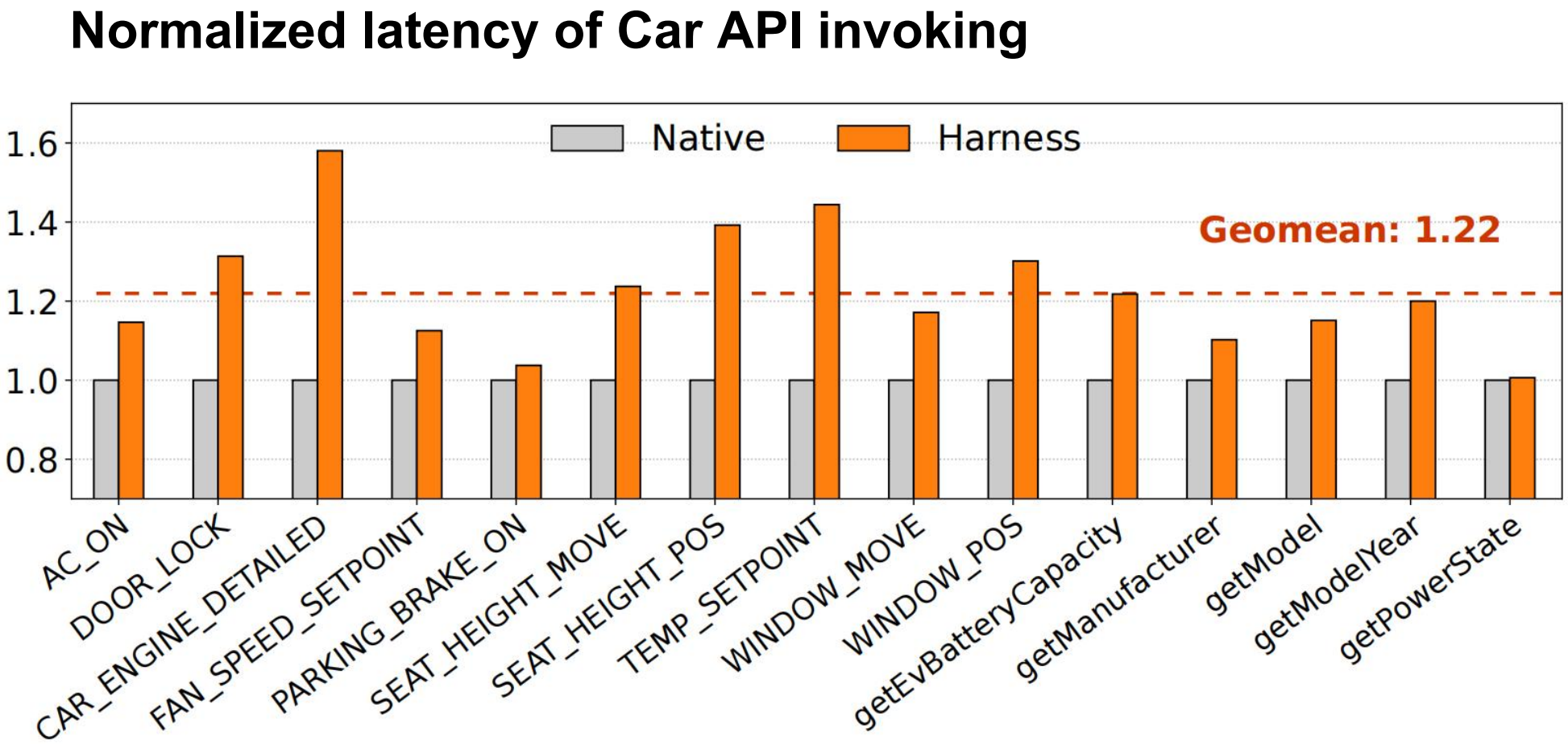
Evaluation: Performance



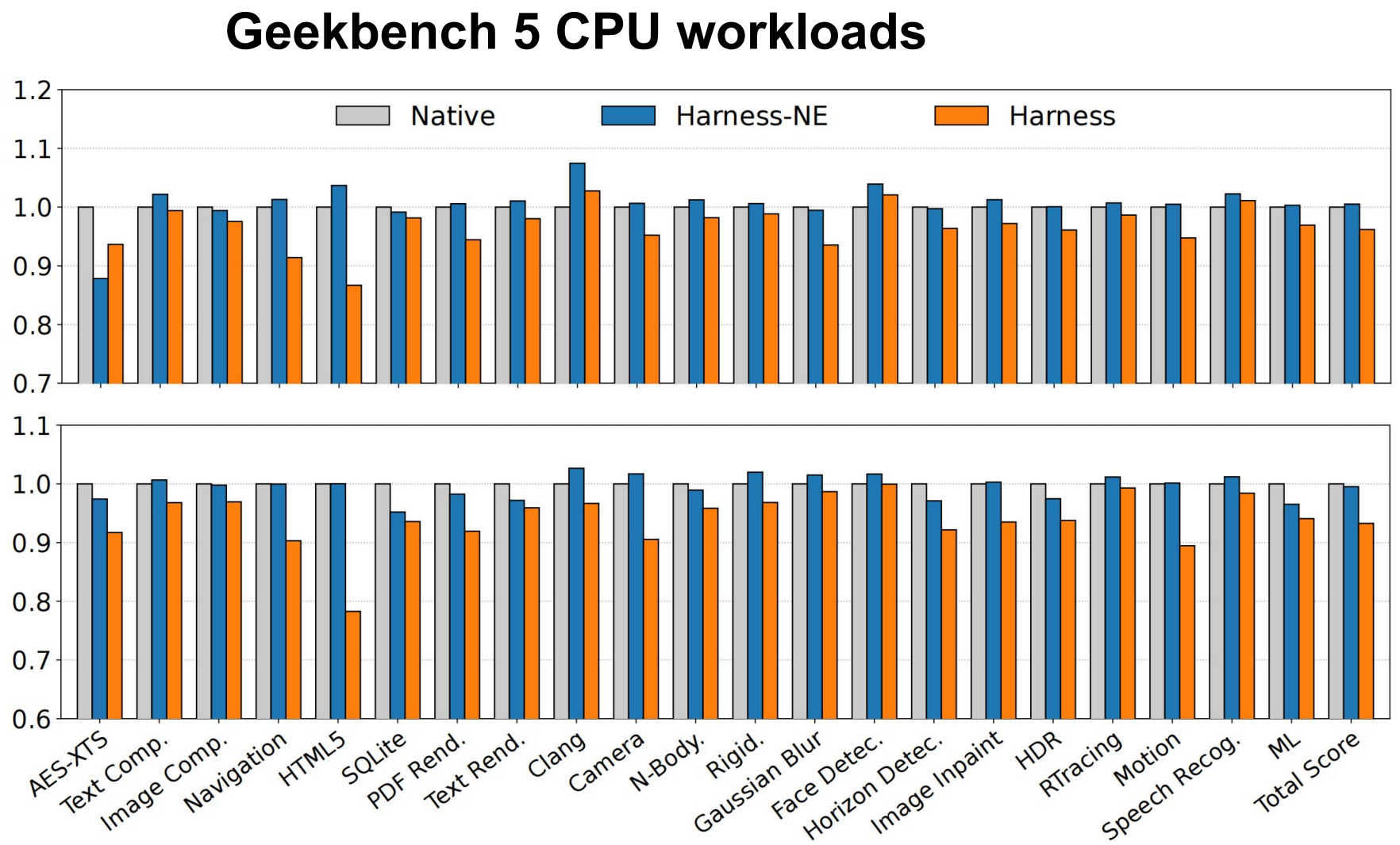
60% slowdown, 4.5x faster than cryptographic method

Negligible impact on non-enclaved components

Evaluation: Performance



Evaluation: Performance



Single-core

Enclave: 3.35%

Non-enclave: -0.57%

Multi-core

Enclave: 6.09%

Non-enclave: 0.46%

Conclusion

Harness leverages hardware virtualization to enclave and protect security-critical car components in AAOS

- Full-chain protection: HMI ⇒ Car Components ⇒ IVN
- Minimal performance overhead and resource consumption
  - Negligible impact on unprotected software
- No intrusive modifications on userspace components
  - Easy to deploy
  - Seamless user experience

浙江大学 ZHEJIANG UNIVERSITY

Thanks for listening!

Harness protects **vehicle control** from **compromised AAOS** —
securely, efficiently, and transparently

Haochen Gong
gonghaochen@zju.edu.cn
Zhejiang University

Source code: https://doi.org/10.5281/zenodo.14723474