CVShield: Guarding Sensor Data in Connected Vehicle with Trusted Execution Environment

Shengtuo Hu
University of Michigan
shengtuo@umich.edu

Qi Alfred Chen
University of California, Irvine
alfchen@uci.edu

Jiwon Joung
University of Michigan
computer@umich.edu

Can Carlak
University of Michigan
ccarlak@umich.edu

Yiheng Feng
University of Michigan
yhfeng@umich.edu

Z. Morley Mao
University of Michigan
zmao@umich.edu

Henry X. Liu
University of Michigan
henryliu@umich.edu

ABSTRACT

The emerging Connected Vehicle (CV) technology enables vehicles to wirelessly exchange safety and mobility information (e.g., location and speed) with traffic infrastructure and other vehicles. Existing CV applications heavily rely on sensor inputs (e.g., GPS). However, previous work has shown that the attacker can cause severe congestion or increased safety risks by compromising vehicles and broadcasting falsified sensor data. Thus, it is highly desirable to ensure the integrity of sensor data.

In this paper, to prevent compromised vehicles from sending falsified sensor data, we propose a system *CVShield*, which utilizes the recent advances in hardware-assisted security (e.g., ARM TrustZone). *CVShield* can ensure the integrity of the sensor data from their reading to their transmission at the vehicle side. In general, we relocate all codes that are related to sensor data reading, processing, encapsulation, and transmission from the rich execution environment (REE) into the trusted execution environment (TEE). However, manually extracting code sections is laborious and error-prone. Also, we should minimize the size of the trusted computing base (TCB) in TEE to reduce the attack surface. To achieve these goals, we propose to leverage program slicing to automatically extract code sections and eliminating irrelevant codes in large codebases. Our initial results demonstrate that *CVShield* can support GPS data reading, and our optimization can eliminate the time overhead introduced by context switches of TrustZone.

KEYWORDS

Connected Vehicle; TrustZone; sensor; spoofing; TEE

ACM Reference Format:
Shengtuo Hu, Qi Alfred Chen, Jiwon Joung, Can Carlak, Yiheng Feng, Z. Morley Mao, and Henry X. Liu. 2020. *CVShield*: Guarding Sensor Data in Connected Vehicle with Trusted Execution Environment. In *Second ACM Workshop on Automotive and Aerial Vehicle Security (AutoSec '20), March 18, 2020, New Orleans, LA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3375706.3380552

AutoSec '20, March 18, 2020, New Orleans, LA, USA

ACM ISBN 978-1-4503-7113-1/20/03…$15.00
https://doi.org/10.1145/3375706.3380552

1 INTRODUCTION

With the emerging Connected Vehicle (CV) technology [19], Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) wireless communication enables vehicles to exchange important safety and mobility information with other entities in real time. While CV technology can greatly benefit transportation mobility and safety, such dramatically increased connectivity inevitably increases the attack surface of CV devices (e.g., vehicles, infrastructures). For example, it is easy for the attacker to send falsified data, interfering with the CV ecosystem. Chen et al. [7] have demonstrated that, by sending falsified location and speed data, one single attack vehicle can create massive traffic congestion.

Since one malicious sender can reach numerous receivers, it is challenging to ensure that all these receivers have proper and timely protections on the spoofing attack. In a CV environment, the diversity of the receivers (e.g., vehicles, infrastructures, and pedestrians) further increases the challenges. For example, some CV receivers (e.g., pedestrians) may not have the computation power to deploy the anomaly detection system. Thus, to fundamentally solve the problem, it is necessary to prevent data spoofing from the sources: malicious vehicles, even if the system is compromised.

To prevent data spoofing at the vehicle side, our goal is to ensure the integrity of the critical CV data (e.g., location and speed) from the incoming sensor reading to the outgoing transmission at the On-Board Unit (OBU). However, the challenge is to provide such integrity guarantee in the presence of potential software-layer compromise in in-vehicle systems. In-vehicle systems are known to have a large attack surface, including a broad range of Electronic Control Units (ECUS) (e.g., CD players, Bluetooth, and cellular radio) [6] and more recently malware in the IVI (In-Vehicle Infotainment) system [18]. More seriously, vehicle owners can compromise their own vehicles. Thus, as long as in-vehicle systems are not vulnerability-free, such compromises are always achievable in practice.

To address this challenge, we design and implement *CVShield* that leverages the recent advances in hardware-assisted security to provide strong security guarantee. More specifically, *CVShield* uses a hardware feature called Trusted Execution Environment (TEE) (e.g., ARM TrustZone [3]). With this hardware feature, the execution environment is split at the processor level to a normal world and a secure world. The normal world runs a commodity OS, which provides a Rich Execution Environment (REE), and is completely isolated from the secure world running the TEE. Thus, security-critical data and code can be put in the TEE so that their confidentiality and integrity can be guaranteed even if the commodity OS is compromised. Such isolation can effectively reduce the attack surface from the whole OS to the code and data residing in TEE, making it much harder to compromise. Also, since the size of the code and data in TEE is much smaller, it is easier to ensure its correctness through formal verification or manual review.

CVShield can protect the pipeline of sensor data (1) reading, (2) processing, (3) encapsulation, and (4) transmission in CV (§4). For (1) and (4), *CVShield* first disables the normal world from accessing peripherals directly, as TEE can control all memory/peripheral accesses and interrupts received by the normal world. Then, *CVShield* relocates drivers of peripherals into TEE, which include sensors and CV network interface, so that only TEE is capable of interacting with security-critical peripherals. (2) and (3) are often handled by user-space applications. However, the codebases of these applications (e.g., `gpsd` [11]) are usually large, so putting them as a whole into TEE will significantly increase the TCB size. Therefore, *CVShield* analyzes these applications only to extract necessary code sections. Since manual extraction of security-critical code sections is laborious and error-prone, we propose to utilize the idea of program slicing [20] to extract code sections of sensor data processing automatically. Apart from protecting the pipeline, *CVShield* also exposes trusted APIs to the normal world for sensor information reading and CV packet transmission, as other applications in the normal world should be able to retrieve sensor data and exchange CV packets. For example, a trusted GPS API will provide the latest location information. For the overall design, we should not violate the real-time constraints of CV [12], which requires the vehicle to broadcast Basic Safety Messages (BSMs) every 100 ms. Overall, our research goals are summarized as follows:

- Design and implement a TEE-based system *CVShield* to protect CV sensor data integrity and prevent CV spoofing attack at the vehicle side.
- Leverage program slicing to reduce TCB size and extract code sections on sensor data processing automatically.
- Optimize the overall system performance to meet the real-time requirement of CV.

2 BACKGROUND

In this section, we introduce the necessary technical background about CV technology [19] and ARM TrustZone [3].

2.1 CV Technology

CV network, based on Dedicated Short Range Communications (DSRC), provides connectivity in support of mobile and stationary CV applications, which offers users (e.g., drivers) greater situational awareness of events, potential threats, and imminent hazards, intending to enhance the safety, mobility, and convenience of everyday transportation [13]. The Basic Safety Message (BSM) defined in SAE J2735 [9] is used by a variety of applications, such as Forward Collision Warning (FCW), Cooperative Adaptive Cruise Control (CACC), to exchange safety data regarding vehicle state (e.g., location and speed). The transmission rate of BSM is typically 10 times per second [12].

In the CV network, there are two basic types of devices: (1) On-Board Unit (OBU) in a roaming vehicle and (2) stationary Road-Side Unit (RSU) along the road. Usually, these devices are ARM embedded devices and install Linux operating systems [8]. As shown in Figure **??**, the OBU is mounted in a roaming vehicle and connected with in-vehicle sensors like GPS and the in-vehicle network such as Controller Area Network (CAN). The RSU is a stationary unit along the road and connected with larger infrastructures or core networks such as the Internet. Roaming vehicles with OBUs installed can not only directly communicate with each other (i.e., V2V) but also communicate with RSUs (i.e., V2I), collectively called Vehicle-to-Everything (V2X) communication.

2.2 ARM TrustZone

ARM TrustZone is a hardware-enforced isolation technique enabled on ARM Cortex processors [3]. As shown in Figure 1, It creates two isolated execution environments: a trusted execution environment (TEE) and a rich execution environment (REE). TEE contains privileged permissions and can access to reserved memory regions. The TrustZone Address Space Controller (TZASC) [1] provided by ARM TrustZone can partition portions of memory such that they are available only to the secure world. Besides, i.MX provides a TrustZone compatible component, the Central Security Unit (CSU), that extends the secure/non-secure access permission to peripherals. The CSU can be used to enable secure-only access for different peripherals. Any invalid accesses will result in an asynchronous external abort exception, similar to a device interrupt. With the present of TEE, asynchronous hardware interrupts can also be routed directly into the secure world, which allows sensors and CV network interface to map all their interrupts to the secure world.

Regardless of privilege, the normal world processes cannot access the instructions and memory in the secure world. In order to execute secure instructions, the normal world must trigger context switch to the secure world using a special `smc` instruction, which generates a synchronous exception and suspends execution in the normal world [2]. The secure monitor handles the exception; then, the secure world is activated for execution.

OP-TEE. OP-TEE [16] is an open source implementation of TEE, which usually works with a non-secure Linux kernel running on ARM. OP-TEE implements TEE Internal Core API v1.1.x which is the API exposed to Trusted Applications, and the TEE Client API v1.0, which is the API describing how to communicate with a TEE. Those APIs are defined in the GlobalPlatform API specifications [10].

3 THREAT MODEL

In this paper, we trust the device hardware and exclude the threat of hardware attacks (e.g., GPS spoofing [21]). Both secure and non-secure kernels should be able to correctly load the properties of

hardware devices (e.g., physical addresses and buses, interrupts, etc.); otherwise, neither kernel can exchange data with sensors.

Following existing works [7, 17], to ensure the functionality of TEE, we assume that the boot ROM and boot loader are trusted so that the secure kernel can be faithfully loaded. On the other hand, the non-secure OS, system services, and all user-space applications in the normal world may be malicious. This is possible, because previous works [6, 14] have already shown that in-vehicle systems can be compromised physically or remotely. Note that attacks that aim at compromising TCB (i.e., all code that runs in TEE) are out of the scope of this paper.

4 PROPOSED APPROACH

As presented in § 1, we aim at protecting sensor data integrity and prevent spoofing attacks at the vehicle side, by relocating code sections of sensor data reading, processing, encapsulation, and transmission into TEE. The following are design goals of *CVShield*:

(1) *Data correctness*: the normal world should not be able to directly access security-critical peripherals (e.g., GPS, CV network interface).
(2) *Functionality*: applications in the normal world should be able to access sensor information (e.g., GPS location, vehicle speed) and transmit CV packets.
(3) *Low complexity*: irrelevant code sections should be removed from TEE in order to reduce TCB size.
(4) *Usability*: code extraction and relocation should be automated and should try to exclude human efforts, which may be laborious and error-prone.

Besides four design goals, we should consider the real-time constraints of the CV network while implementing *CVShield*.

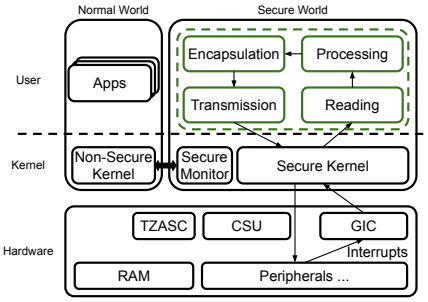

Figure 1: *CVShield* extends the trust boundary to protect code sections related to sensor data in green boxes.

To achieve Goal (1), we first utilize a TrustZone-specific feature to configure access permissions of security-critical peripherals. In § 2.2, we mention that i.MX products are equipped with a TrustZone compatible component, CSU. Although *CVShield* prototype is built on i.MX6 SoC, if other SoCs provide similar security functionality, *CVShield* is general and can be ported to other SoCs. Then, to allow TEE to exchange data with peripherals, we relocate necessary device drivers into TEE, such as the serial device driver required by GPS. After that, by programming the ARM General Interrupt Controller (GIC), *CVShield* registers interrupt handlers for protected peripherals. When new data arrives at the peripherals, only the secure world can receive the interrupt and retrieve the data.

For Goal (2), we first need to port code sections of sensor data processing and encapsulation into TEE. Then, we expose trusted APIs to the normal world, so applications in the normal world can still retrieve sensor information (e.g., location and speed) and transmit CV packets. In this case, *CVShield* can understand the raw sensor data and extract the sensor information; also, it can validate outgoing CV packets before the transmission. However, unlike device drivers, programs of sensor data processing and encapsulation usually have large codebases and contain irrelevant code sections, which may violate Goal (3). For example, `gpsd` [11] is a commonly used daemon that receives data from a GPS receiver and provides the data back to multiple user-space applications. Parsing GPS data and generating location reports is just a small portion of it. Thus, we propose to utilize program slicing [20] to remove irrelevant parts. Static program slicing takes the source code and a slicing criterion (e.g., a call-site of some function) as input. It then performs static analysis to generate a program slice that may affect the values of the slicing criterion. For instance, the potential slicing criterion for `gpsd` can be the function of generating location reports. Also, program slicing can help us reduce human efforts in code extraction, so Goal (4) can be guaranteed.

To ensure the real-time requirement in the CV environment, for each component in Figure 1, we break down the time overhead and optimize the performance case by case. In § 5.2, we show our efforts on optimizing sensor data reading, which can eliminate overhead introduced by context switches between the normal world and the secure world.

5 INITIAL RESULTS

In this section, we present our current progress on implementing *CVShield*. Also, we conduct experiments to evaluate the performance on the current implementation.

5.1 Testbed Setup

Our testbed is based on the Boundary Devices Nitrogen6Q development board [5], namely the SABRE Lite board. The overall hardware of this board is similar to commercial OBUs [8]. The board has 1GB of memory and contains an i.MX6 SoC with a quad-core ARM Cortex-A9 processor with TrustZone security extensions. We use Linux kernel version 4.9.128 [4], provided by Boundary Devices, as the non-secure kernel. The secure kernel is based on OP-TEE [15].

5.2 Current Implementation and Performance

As an initial step, our current implementation is at the stage of protecting sensor data reading. By modifying OP-TEE, we integrate CSU driver and enable TZASC-380 driver. We reuse `imx_uart` driver in OP-TEE codebase for GPS I/O, which provides read/write capabilities of the serial device. Since the serial device transmits data byte by byte, we further wrap it up to provide C-style I/O API. To understand the performance of sensor reading, we expose `read_s` API to the normal world and measure its performance with the baud rate of 115200. In the normal world, we develop a small test program that repeatedly calls the exposed API. For different sizes of data, the test program invokes the read function 1000 times. We measure the execution time of each function call and calculates the average value. Table 1 summarizes the results on exposed APIs. For

a real GPS device, the polling interval is about 0.044 seconds, and the average size of fetched data is around 7.91 bytes. Our current implementation can handle real GPS data and only takes around 0.686 ms to read 7.91 bytes of data for each polling.

Table 1: Performance of exposed serial device I/O APIs.

Size (bytes)	10	100	1000	10000
`read_s` (ms)	0.87	8.79	87.98	880.02
`read_mem` (ms)	0.79	8.71	87.91	879.90

Optimization. We analyze the execution time of the exposed API (see Figure 2) and propose a passive communication mechanism based on the shared memory, because we want to reduce introduced overhead as much as possible. In comparison with the normal case (Figure 2a), the extra time overhead in Figure 2b comes from (1) context switches (around 0.041 ms) and (2) memory copy between the normal world and the secure world. In § 2, we mention that the context switch (i.e., `smc` instruction) generates a synchronous exception and suspends the execution in the normal world. To avoid synchronous context switches, we design a passive communication mechanism that utilizes the asynchronous hardware interrupts. As shown in Figure 2c, the secure kernel handles interrupts from hardware devices using an idle CPU core. Then, the secure kernel writes data into a read-only memory chunk that is shared with the normal world. In this case, the applications do not need to trigger context switches, but only monitor the shared memory (e.g., an array) and process the latest data accordingly. Based on this design, we implement a sensor data reading API `read_mem`. By comparing two rows in Table 1, we confirm that our optimization can reduce the time overhead caused by context switches.

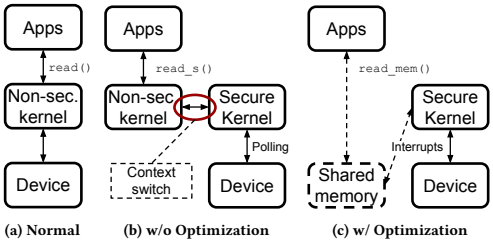

Figure 2: Optimization of sensor data reading. (a) does not incorporate TEE, while (b) shows a trusted API of sensor reading. (c) avoids context switches and eliminates the time overhead via shared memory.

6 CONCLUSION AND FUTURE PLANS

Rapid advances in CV technology have increased the probability of cyberattacks (e.g., spoofing attacks) [7]. In this position paper, we propose a TEE-based system *CVShield* to protect sensor data integrity. Overall, *CVShield* relocates security-critical code sections of sensor data reading, processing, encapsulation, and transmission into TEE so that the malicious attacker in the normal world cannot modify and transmit falsified data. To automate the security-critical code extraction and minimize TCB size, we utilize program slicing to remove code sections that are irrelevant to sensor data processing. To ensure the functionalities of normal-world applications, we expose trusted services to provide the latest sensor information and CV packet transmission capability with the normal world. Also, we consider optimization during the design and development phases to ensure the efficiency of *CVShield*.

For future plans to finalize *CVShield*, we would like to:

(1) Incorporate more peripherals (e.g., CAN bus, CV network interface, Camera, LiDAR) into TEE and present experimental results to prove the effectiveness of the defense scheme.
(2) Explore whether program slicing can be extended to code sections of sensor reading and transmission to further reduce human efforts on porting device drivers, which requires the program analysis on kernel modules.
(3) Evaluate end-to-end performance on the whole system to understand the bottleneck and optimize the system to meet the real-time constraints in the CV network.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for providing valuable feedback on our work. This research was supported in part by an award from Mcity at University of Michigan, and by the National Science Foundation under grant CNS-1929771.

REFERENCES

[1] ARM Ltd. 2019. Address Space Controllers – Arm. https://tinyurl.com/uxjdnfz.
[2] ARM Ltd. 2019. SMC Calling Convention - ARM Infocenter. https://tinyurl.com/y6gkrpo3.
[3] ARM Ltd. 2019. TrustZone – Arm Developer. https://tinyurl.com/vj29ybd.
[4] Boundary Devices. 2019. boundarydevices/linux-imx6. https://tinyurl.com/w66kfkj.
[5] Boundary Devices. 2019. i.MX6 ARM Development Board. https://boundarydevices.com/product/bd-sl-i-mx6/.
[6] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. 2011. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *Proc. USENIX Security*.
[7] Qi Alfred Chen, Yucheng Yin, Yiheng Feng, Z. Morley Mao, and Henry X. Liu. 2018. Exposing Congestion Attack on Emerging Connected Vehicle based Traffic Signal Control. In *Proc. NDSS*.
[8] Cohda Wireless. 2019. Cohda MK5 OBU. https://tinyurl.com/y6qepj6h.
[9] Cross-Cutting Technical Committee. 2016. Dedicated Short Range Communications (DSRC) Message Set Dictionary™ Set. *SAE International* (Mar. 2016).
[10] GlobalPlatform. 2019. GlobalPlatform Homepage - GlobalPlatform. https://globalplatform.org/.
[11] GPSD project. 2019. biiont/gpsd. https://github.com/biiont/gpsd.
[12] John Harding, Gregory Powell, Rebecca Yoon, Joshua Fikentscher, Charlene Doyle, Dana Sade, Mike Lukuc, Jim Simons, Jing Wang, et al. 2014. *Vehicle-to-Vehicle Communications: Readiness of V2V Technology for Application.* Technical Report. https://tinyurl.com/yc579x68
[13] IEEE 1609 Working Group. 2019. IEEE Guide for Wireless Access in Vehicular Environments (WAVE) Architecture. *IEEE Std 1609.0-2019 (Revision of IEEE Std 1609.0-2013)* (2019).
[14] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. 2010. Experimental Security Analysis of a Modern Automobile. In *Proc. IEEE S&P*.
[15] Linaro Ltd. 2019. OP-TEE/optee_os. https://tinyurl.com/rurhgcl.
[16] Linaro Ltd. 2019. Open Portable Trusted Execution Environment - OP-TEE. https://www.op-tee.org/.
[17] He Liu, Stefan Saroiu, Alec Wolman, and Himanshu Raj. 2012. Software abstractions for trusted sensors. In *Proc. MobiSys*.
[18] Sahar Mazloom, Mohammad Rezaeirad, Aaron Hunter, and Damon McCoy. 2016. A Security Analysis of an In-Vehicle Infotainment and App Platform. In *USENIX WOOT*.
[19] U.S. Department of Transportation (US DOT). 2019. Intelligent Transportation Systems - Connected Vehicle Basics. https://tinyurl.com/yxjj98vr.
[20] Mark Weiser. 1981. Program slicing. In *Proc. ICSE*.
[21] Kexiong (Curtis) Zeng, Shinan Liu, Yuanchao Shu, Dong Wang, Haoyu Li, Yanzhi Dou, Gang Wang, and Yaling Yang. 2018. All Your GPS Are Belong To Us: Towards Stealthy Manipulation of Road Navigation Systems. In *Proc. USENIX Security*.