Enabling Real-Time Restoration of Compromised ECU Firmware in Connected and Autonomous Vehicles

Josh Dafoe, Harsh Singh, Niusen Chen, and Bo Chen(✉)

Department of Computer Science, Michigan Technological University, Michigan, USA

bchen@mtu.edu

Abstract. With increasing development of connected and autonomous vehicles, the risk of cyber threats on them is also increasing. Compared to traditional computer systems, a CAV attack is more critical, as it does not only threaten confidential data or system access, but may endanger the lives of drivers and passengers. To control a vehicle, the attacker may inject malicious control messages into the vehicle's controller area network. To make this attack persistent, the most reliable method is to inject malicious code into an electronic control unit's firmware. This allows the attacker to inject CAN messages and exhibit significant control over the vehicle, posing a safety threat to anyone in proximity.

In this work, we have designed a defensive framework which allows restoring compromised ECU firmware in real time. Our framework combines existing intrusion detection methods with a firmware recovery mechanism using trusted hardware components equipped in ECUs. Especially, the firmware restoration utilizes the existing FTL in the flash storage device. This process is highly efficient by minimizing the necessary restored information. Further, the recovery is managed via a trusted application running in TrustZone secure world. Both the FTL and TrustZone are secure when the ECU firmware is compromised. Steganography is used to hide communications during recovery. We have implemented and evaluated our prototype implementation in a testbed simulating the real-world in-vehicle scenario.

Keywords: Connected and autonomous vehicles · ECU · CAN bus · flash translation layer · TrustZone · Steganography

1 Introduction

With rapid growth of automotive industries, both automakers and associated government agencies are taking initiatives to support the development and deployment of connected and autonomous vehicles (CAVs). This includes efforts

Y. Chen et al. (Eds.): SmartSP 2023, LNICST 552, pp. 15–33, 2024.
https://doi.org/10.1007/978-3-031-51630-6_2

to improve CAV efficiency and implement public road infrastructures to support V2V (vehicle-to-vehicle) and V2I (vehicle-to-infrastructure) communications. As this technology continues to develop, CAVs have increasing communication pathways in order to make informed decisions in real time. In addition to V2V and V2I developments, autonomous vehicles are increasingly equipped with many sensors, providing input which will be processed in real time. Especially, increased internal communication is observed, with over 70 electronic control units (ECU) communicating via a in-vehicle network. Recently, in order to manage the numerous ECUs, over-the-air updates have been utilized [13,30]. While these developments allow significant improvement in vehicle functions, they have led to increased security vulnerabilities [17,41,47]. As a result, various attacks on the vehicle systems have been performed by both researchers and real-world hackers. For example, from 2010 to 2018, there were 170 reported automotive attacks, with 60 of these happening in 2018. Further, a plurality of attacks were remote which do not require physical access [39].

One typical attack is performed by accessing the internal network of the vehicle, gaining control over it. Adopted broadly, the controller area network (CAN) provides internal communications among the in-vehicle computer systems. If any malicious entity gains access to the CAN, many in-vehicle operations become vulnerable to manipulation. A few methods have been identified by which the aforementioned access may be gained. First, the attacker hacks remotely into the infotainment system or telematics unit, which manage existing network communication with the outside world. They will then attempt to escalate privileges, and enable the injection of malicious messages which control the victim vehicle. This attack method was demonstrated in 2016 when the Keen Security Lab hacked a Tesla model S [36]. Second, the attacker gains access to the CAN bus physically or remotely via compromising the existing mechanisms for CAN access provided by the on board diagnostics (OBD-II) port. The physical attack occurs through connecting a device directly to the OBD-II port, such as a laptop. However, this is often difficult, as during vehicle usage, the attacker cannot be present, and gaining the initial physical access is challenging. The remote attack through an OBD-II port is performed by compromising an OBD-II dongle that the car owner or mechanic connects [16,31,44]. Also, there are now mobile apps which remotely provide diagnostic services by accessing the CAN [24]. In all the aforementioned attack scenarios, the CAN access is likely limited in time, as the hacker cannot be physically present during the vehicle operation, or the persistence of a remote connection is not guaranteed. Consequently, both the remote and physical attackers would prefer to establish a persistent presence within the CAN. To gain this persistent presence, a best choice for the attacker is to inject malicious code into the internal firmware of ECUs. This occurs via exploiting existing vulnerabilities during firmware updates (such as the over-the-air updates [37]), or ECU programming via the OBD-II port [14]. This work therefore focuses on recovering the ECU firmware which has been compromised by such code injection attacks.

To defend against the ECU code injection attacks, a taxonomy with four categories of CAV defense was established [41], including a passive defense and an active defense. The passive defense framework is focused on detecting and isolating CAV malware attacks, and some additional research has been performed in this area. Specifically, to detect the presence of malicious CAN activity, many intrusion detection systems with high success rates have been evaluated [25–27]. These detection methods observe messages on the CAN bus and establish a model for normal behaviors. Using this model, any malicious deviations are detected. Further, the specific ECU that sent the malicious messages can be detected via analysis of signal characteristics [21,35,46]. In [32], it is observed that relying solely on intrusion detection mechanisms results in a delayed response, where a security update eventually may repair the attack, which is insufficient in the CAV scenario. This is because in this scenario, each moment an attack is active there is more threat posed to both the driver and everyone around them. Since the real-time attack response is so essential, [32] further proposes incorporating an active response by sending detection notifications to the infected ECUs, which will repair via switching to a safe mode and rebooting. However, the specific mechanisms for enabling the restoration of the ECU firmware are still missing. This work thus aims to bridge this gap. Our key observation is that through leveraging trusted hardware components equipped with the ECU, it is possible to enable a real-time restoration of the ECU firmware, which has been compromised by the code injection attacks.

Typically, in-vehicle computers (e.g., ECUs) are equipped with low power processors, including ARM Cortex-A or Cortex-M [1–4] CPUs which are broadly equipped with TrustZone capabilities [10,11]. Further, these same computers may use flash memory as external storage [5,8], which is typically managed by the flash translation layer (FTL). The TrustZone is a hardware-level security feature provided by the processor, which can enable the establishment of a trusted execution environment (TEE) isolated from the normal insecure execution environment. In other words, even if the ECU OS is compromised, the execution running in the TrustZone secure world remains uncompromised. The FTL is a piece of trusted flash memory firmware encapsulated inside a flash storage device (e.g., an SSD drive or a microSD card). It stays between the OS and the flash memory hardware, transparently managing the unique hardware nature of flash memory and exposing externally a block access interface. Therefore, the FTL can also remain secure even if the ECU OS is compromised.

Combining the intrusion detection mechanism with the trusted hardware components, we have established a framework which can efficiently restore the ECU firmware to the version right before the code injection attack (note that we refer to this as the "good prior state" or "good firmware version" throughout the paper). After the compromised ECU is detected by a trusted detection module (i.e., a detector), a notification message will be sent by the detector via the CAN. The notification message will arrive at the compromised ECU and be passed to the trusted application running in the TrustZone secure world via the ECU OS. The trusted application will then collaborate with the trusted FTL to

restore the firmware in real time. Our key insight is that whenever a firmware update occurs (e.g., the code injection attack is performed), the FTL, having ultimate control over the underlying storage hardware, can naturally retain an old version of the firmware, due to the out-of-place update feature present in modern flash storage devices. In this way, the FTL can immediately revert to the old good firmware version after the attack. Such a reversion can happen very efficiently as only a small amount of mapping data needs to be restored in the FTL, perfectly meeting the real-time requirement. In addition, to prevent the compromised ECU OS from being aware of the restoration process, all the communications (between the detector and the trusted application, between the trusted application and the FTL) are protected via steganography. In this way, all the communications[1] among them can go through the untrusted ECU OS with the actual purpose being hidden and, the compromised ECU OS typically will not interrupt a seemingly normal process.

2 Background

2.1 Control Area Network

The controller area network (CAN) is a protocol for communication between many nodes connected via two wires where each message is broadcast to all other connected nodes (Fig. 1). Through using this protocol, vehicles are able to greatly reduce the wiring complexity and enable a variable internal network topology. When a node sends a message on the CAN bus, the frame does not include any sender information, but contains a message identifier which describes its type and determines its priority. Based on this identifier, nodes connected to the CAN bus filter out irrelevant messages and accepts those with relevant identifiers. Additionally, each CAN message contains up to 8 bytes of relevant data and commands. The CAN bus is fully accessible, allowing devices or applications to be connected via the on board diagnostics port (OBD-II). In vehicles, CAN is the mechanism for sensors to send data to the main advanced driver assistance systems (ADAS) computer, and for control signals to be sent from the ADAS computer, brake and gas pedals, steering wheel, ignition, etc., to the various ECUs associated with the control operations. Additionally, ECU firmware updates are ultimately sent directly through the CAN bus. Our observation is that when a CAN message is accepted by an ECU, the associated data will be quickly processed [6].

2.2 Flash Memory

Flash memory is broadly used as the external storage device for low-power embedded systems like ECUs [5,8]. This is due to its high throughput, which is necessary in the vehicle scenario, requiring real-time I/O capabilities. Flash

[1] Note that the detector should avoid directly communicating with the FTL via the untrusted ECU OS, which is unusual and hence suspicious.

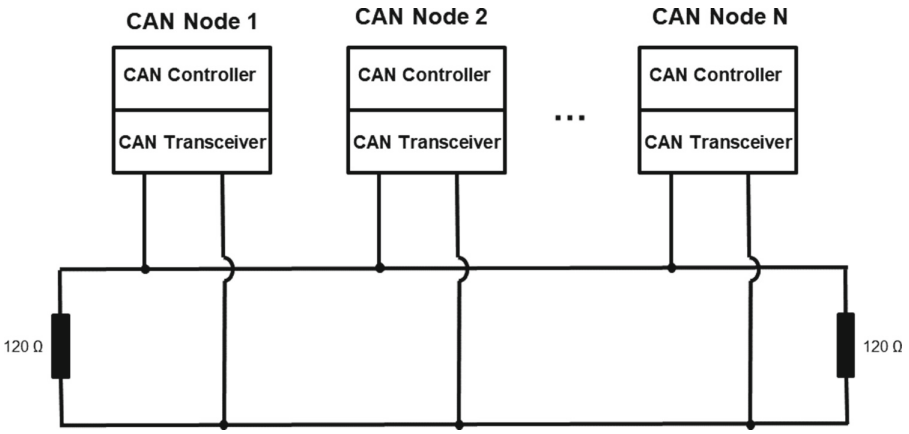

Fig. 1. The topology of a CAN network.

memory (Fig. 2) is organized into a collection of blocks, with each block consisting of smaller pages. However, unique physical characteristics result in differing behavior from hard drive disks (HDD). First, the read/write granularity of flash storage is a page, while the erasure operates on full blocks. Second, each program erase cycle performed on a given block wears down the associated hardware, until a threshold is met and it is considered unreliable and unusable. Due to these special characteristics, in-place updates are expensive. This is because when the data in a single page should be updated, the entire encompassing block must be erased, resulting in further wear. Therefore, an out-of-place update strategy is preferred in which updates are performed by writing the data to a new physical location and marking the old data as invalid. It is also essential to spread program erase cycles throughout the entire storage medium in order to prevent quick wear in any location, so wear-leveling is implemented which handles this. When blocks are invalidated, they are eventually sent to the garbage collector to be erased. Unlike traditional HDDs, the out-of-place update strategy results in different physical locations for the same logical address over time. This is managed by maintaining mappings between physical and logical locations which usually change after each invalidation. All of these firmware components together make up the flash translation layer (FTL), which provides a block access interface externally to the OS. Additionally, the FTL is isolated from the firmware (OS) of its associated ECU by the storage hardware. This isolation provides a guarantee that any computation performed in the FTL will not be compromised even when the ECU firmware is compromised.

2.3 ARM TrustZone

Many ARM processors, such as Cortex-A and Cortex-M CPUs used within automotive ECUs are ARM TrustZone enabled [1–4]. TrustZone establishes a trusted execution environment within a untrusted host. The key idea is to run both

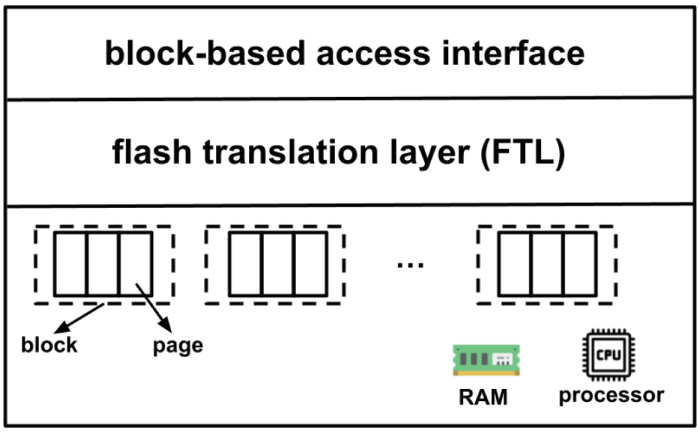

Fig. 2. The architecture of a flash-based block device.

secure (i.e., secure world) and non-secure (i.e., normal world) execution environments on a single processor. The secure world is used to run critical applications with sensitive data, while the normal world can run non-sensitive applications. The two modes are separated by isolating the CPU states and associated memory regions. The architecture of ARM TrustZone is shown in Fig. 3. The communication and interaction between the secure world and the normal world is conducted by secure monitor call (SMC). SMC works as a gateway to ensure invocation of functions and services offered by the secure monitor or secure kernel within the secure world. A salient advantage of TrustZone is that it comes together with the embedded processor and, this hardware-level security feature can be simply utilized without bringing in extra hardware.

2.4 Steganography

Steganography is a mechanism by which to hide some secret message inside of normal data/communications. The secret message is embedded obscurely into original data or messages, such that it goes unnoticed. Different from encryption, this is intended to conceal the fact that a secret message is being sent at all.

3 System and Adversarial Model

3.1 System Model

We consider a connected vehicle with multiple ECUs communicating via the CAN protocol. The ECU is assumed to be equipped with a NAND flash storage device (e.g., an eMMC, a microSD, etc.) on which the ECU firmware is stored. The flash storage device is managed by an FTL, which provides a read/write interface to the ECU OS. The FTL is run on hardware isolated from the OS, so

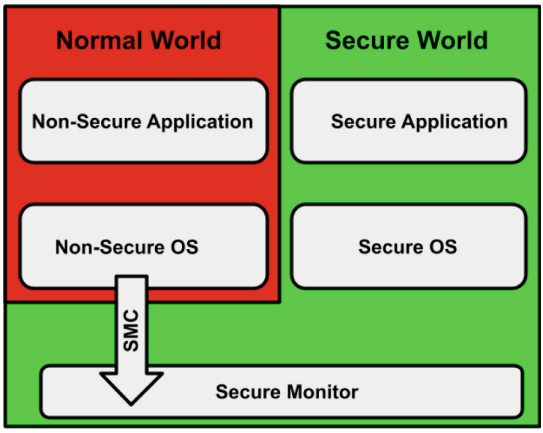

Fig. 3. The architecture of ARM TrustZone.

the computation performed by it is assumed to be secure. Further, each ECU is assumed to be equipped with an ARM processor (Cortex-A or Cortex-M) with TrustZone enabled. Using TrustZone, a trusted world is running in the ECU, on which trusted computation can occur. The trusted world, running trusted applications (TAs) can communicate with untrusted client applications (CAs) running in the untrusted OS (i.e., the potentially compromised ECU firmware). The CAs can perform bidirectional communication with the TA, FTL, and CAN bus. We assume the existence of a trusted in-vehicle computer (IDet) connected to the CAN bus, which performs intrusion detection and signal analysis to detect and localize adversarial ECUs. Note that IDet can communicate directly with the CA via the CAN bus. IDet could be the main ADAS computer or an ECU dedicated to intrusion detection. Our focus in this work is not on malware detection. Therefore, we assume this trusted entity has successfully detected the adversarial ECU [21,25–27,35,46] and we work with the TrustZone and the FTL in the compromised ECU to restore its firmware to a good prior state.

3.2 Adversarial Model

We consider an adversary which can compromise the firmware of an ECU, i.e., by injecting malicious code into the ECU OS. This can be done in a few ways, including remote or physical access to CAN via the OBD-II port, or manipulation of other existing firmware update mechanisms including over-the-air updates. Since the ECU firmware itself is compromised, any detection and recovery mechanisms running in the ECU OS can be averted. This is equivalent to a piece of OS-level malware, which can control the OS of the victim ECU. However, this malware is detectable via intrusion detection of the vehicle, as it must behave maliciously in order to take control of the vehicle, e.g., sending a lot of spoofed CAN messages.

We rely on a few assumptions: 1) The compromised ECU is not able to compromise the TAs running in the TrustZone secure world, which is protected by the processor at the hardware level. This is a common assumption for TrustZone-based applications [23]. 2) The compromised ECU is not able to hack into the FTL, which is isolated by the storage hardware and only presents a limited read/write interface. 3) Before the ECU is compromised, its firmware (OS) is assumed to be healthy. 4) The compromised ECU will not perform DoS attacks, e.g., blocking *regular* communication among CAN, CA, TA, and FTL. Mitigating DoS attacks itself is a hard problem and is out of the scope of this work. In addition, the compromised ECU will not gain any benefits from performing the DoS attacks, as a nonfunctional ECU is an immediate indication of being compromised. In our work, the communications for restoration process are hidden stealthily in the regular communication messages.

4 Design

4.1 Design Overview

Our design consists of four major components (Fig. 4): IDet, CA, TA, and FTL. The IDet (intrusion detector) is running on top of trusted firmware in a secure node, which can communicate with the victim ECU via CAN network. In the victim ECU, there are three components, the CA (client application), the TA (trusted application), and the FTL. The CA is running on top of the untrusted firmware which may be compromised. The TA and the FTL are isolated from the CA by TrustZone hardware and the storage hardware respectively, hence are trusted. We collaborate the aforementioned components in order to restore the compromised ECU firmware to a good prior state after it is compromised.

Our first idea is that the FTL has an ultimate control over the underlying storage hardware and, the previous version of firmware may be maintained and restored. Especially, due to the out-of-place update strategy (Sect. 2.2) in the flash storage, the old version of firmware can be naturally retained in the flash memory blocks, though they will be invalidated when an adversarial update occurs. Since GC eventually erases these invalid blocks, it must be disabled for the old firmware data. Additionally, since the FTL does not know where the firmware is stored, it can be notified before any update occurs, because the firmware is trusted at this point (Sect. 3.2). During recovery, an additional challenge is to find the maintained blocks associated with the good firmware version. These locations can be retrieved by using the mappings associated with the old firmware version, which can be backed up by the FTL.

Our second idea is to securely manage the FTL to restore the ECU firmware even if the entire ECU OS is untrusted. In a vehicle environment, the compromised ECU is able to be identified by another entity (i.e., IDet) outside this ECU in the same vehicle. The IDet needs to inform the FTL to launch the restoration process, but such sensitive messages typically need to go through the CA running on the compromised ECU OS, which will deliver the messages to the

FTL. The direct communication between the IDet and the FTL is very abnormal and the compromised OS will be alerted. Having observed that the data received from the CAN may be processed by the TA running in the TrustZone secure world, and the TA may perform writes to the storage device through the CA [29], our solution is to use the TA as a liaison to forward sensitive messages between the IDet and the FTL. In addition, as the sensitive messages need to go through the untrusted OS, they need to be protected in a plausible manner. Steganography is therefore leveraged to hide the sensitive messages within the regular communications.

Our third idea is to enable the restoration of the ECU firmware when the malware is still present. This is due to the fact that it would be hard for the vehicle user to block the ECU malware once being detected. Upon restoration, the FTL will block all the write requests from the upper layer, and this blocking operation will be canceled once the good firmware has been restored on the external storage and the malware has been removed from the memory.

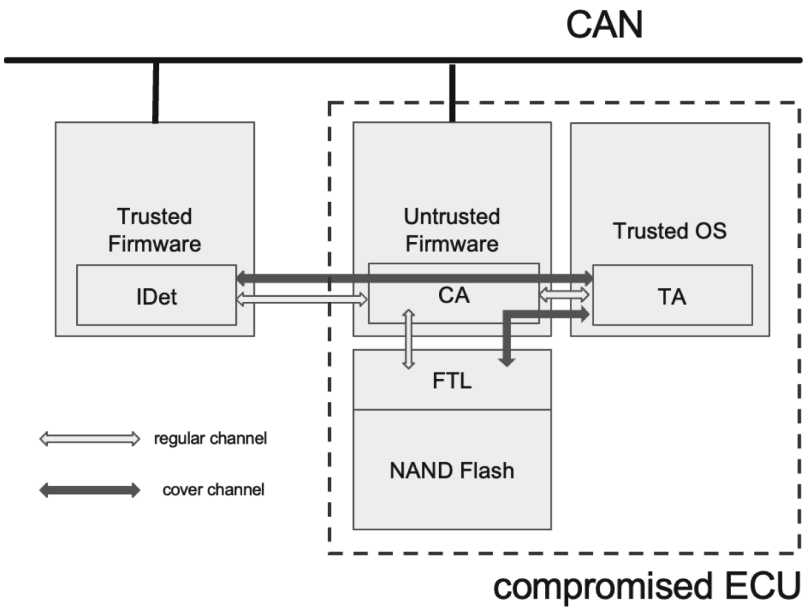

Fig. 4. An overview of our design.

4.2 Design Details

4.2.1 Cover Communications via Steganography. We define a steganographic message $M_s \in \{0,1\}^k$ to be the message produced when a secret message $\beta \in \{0,1\}^l$ is embedded within a regular cover message $\alpha \in \{0,1\}^k$, where l and

k are both positive integers and $l < k$. Along with the steganographic message are the steganographic algorithms associated with generating and decoding it.

To define the steganographic algorithms used in our design, we first introduce the pseudo random permutation π and pseudo random function f, defined as follows (where s is the length of a shared key):

$$\pi : \{0,1\}^s \times \{0,1\}^{\log_2 k} \rightarrow \{0,1\}^{\log_2 k}$$
$$f : \{0,1\}^s \times \{0,1\}^* \rightarrow \{0,1\}^s$$

Our steganographic algorithms used during the encoding and decoding processes are defined in Algorithm 1 (SEncode) and 2 (SDecode) respectively. Note that by $f_k(x)$ (or $\pi_k(x)$) we mean applying f (or π) over x using key k.

Algorithm 1. SEncode

Input: β, α, key, counter
Output: M_s
1: $M_s \leftarrow \alpha$
2: stegKey $\leftarrow f_{\text{key}}$(counter)
3: **for** $i = 0$ to $l - 1$ **do**
4: $\quad j \leftarrow \pi_{\text{stegKey}}(i)$
5: $\quad M_s[j] \leftarrow \beta[i]$
6: **return** M_s

Algorithm 2. SDecode

Input: M_s, key, counter
Output: β
1: stegKey $\leftarrow f_{\text{key}}$(counter)
2: **for** $i = 0$ to $l - 1$ **do**
3: $\quad j \leftarrow \pi_{\text{stegKey}}(i)$
4: $\quad \beta[i] \leftarrow M_s[j]$
5: **return** β

After IDet detects and localizes a compromised ECU, it will send a steganographic message M_{s0} indicating this detection result to the ECU. Since this message will immediately be forwarded to the TA, a key and counter shared between them are used as input to SEncode (Algorithm 1). Further, β is taken to be some secret message, agreed by IDet, TA, and FTL to indicate a malware detection, and α can be any cover message of length k. Unique to the vehicle scenario is that the message is being transmitted over CAN, which has an 8 byte data section that imposes security limitations if using a single CAN frame. Due to this, IDet spreads the k bit message produced by SEncode over $\lceil k \div 64 \rceil$ CAN messages.

After the CA forwards this message (i.e., a collection of $\lceil k \div 64 \rceil$ CAN messages) to the TA, β is extracted using SDecode (Algorithm 2) and is checked against its expected value. If they are identical, a new steganographic message M_{s1} is generated from β with a new α, along with a unique key and counter shared between the TA and the FTL. The TA will send M_{s1} to CA, indicating that it should be written to the FTL. Upon receiving M_{s1}, the FTL will use SDecode to extract β, and check this against its expected value. If they are identical, then firmware restoration will be launched by the FTL. To avoid any replay attacks, both the counter shared between the IDet and the TA, and the counter shared between the TA and the FTL, should be increased by one after each successful restoration.

4.2.2 Firmware Restoration. After receiving the detection notification in the FTL, the old firmware should be restored quickly so that normal operations can resume. To ensure that this is possible, there are two challenges. First, the old firmware must still be present in a recoverable manner on the storage device. Second, the old firmware should be restored quickly to the correct location.

To address the *first* challenge, we exploit the out-of-place updates feature of NAND flash memory. Due to out-of-place updates, during a firmware update the new firmware is written to a different physical location which results in persistence of the old firmware. Normally, when the new data are written to a new physical location, the old blocks are marked as invalid, the mapping from logical address to physical location is updated, and garbage collection (GC) will eventually delete the data. To ensure that the firmware is both maintained and recoverable, we can 1) save the old mappings (from logical to physical location) before an update occurs, and 2) block GC for the blocks associated to relevant saved mappings.

For 1), the FTL reserves a special area for a back up mapping table which stores the saved mappings. Since the ECU firmware is assumed to be trusted prior to the firmware update, a command is sent to a reserved command area by the CA. This command tells the FTL to back up the current mapping tables to the special reserved area. By saving these mappings prior to the update, references to the physical location of the old firmware are maintained. For 2), the data at these physical locations should not be erased. To prevent this, GC is disabled for all blocks invalidated during the firmware update. However, a problem arises when there are multiple firmware updates, as many data blocks will be maintained, but only the mapping tables associated with the last update are preserved. For this reason, GC should be re-enabled on the previously maintained blocks before each firmware update.

To address the *second* challenge, the saved blocks need to be restored. Since the firmware will always boot from the same location, the good firmware should be reverted to this location. To achieve this, the mappings in the reserved area which reference the prior firmware blocks can be restored. Due to this restoration, the same address will now point to the old firmware blocks rather than the malicious firmware. When booting, the ECU will read from the same logical

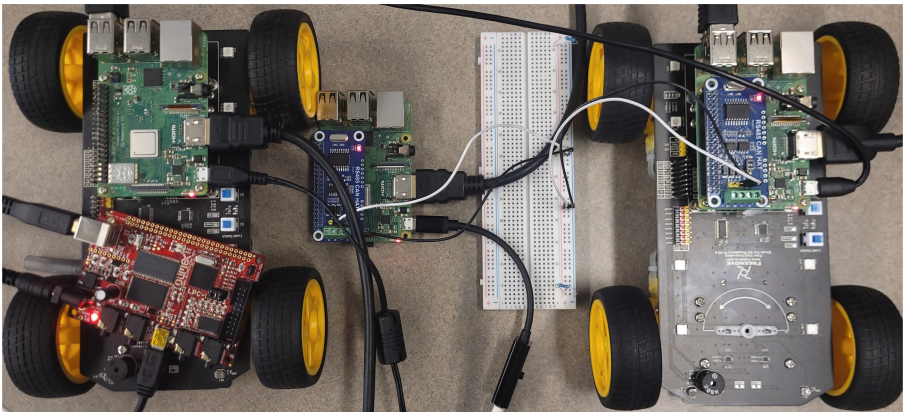

Fig. 5. Our vehicle testbed.

location as before, but it will point to the physical location of the firmware prior to the adversarial code injection.

4.2.3 Malware Removal. After returning the ECU firmware to the good version prior to the code injection attack, the malware may still be running on the CPU and contained in the ECU memory. A problem associated with this is that the malware can again modify the ECU firmware being restored. Different from the scenario of a real-world computer/mobile device [19], it is hard for the user to block/remove the malware from the victim ECU before the firmware restoration in the FTL after having detecting it. To account for this, once the firmware restoration starts in the FTL, any writes on the FTL should be frozen until the malware has been removed from the memory. To remove the malware from the memory, we can reboot [7] the ECU immediately to clear the memory after its firmware is restored and, after the reboot, the FTL can be notified to cancel the freezing operation.

5 Implementation and Evaluation

To construct the testbed (Fig. 5) with all the necessary components for our implementation, we use two different electronic development boards: 1) Raspberry Pi 3B+ [9] (With 1.4GHz 64-bit quad-core ARM Cortex-A53 CPU, and 1GB LPDDR2 SDRAM) with a RS485 CAN HAT, and 2) a high speed USB header development prototype board LPC-H3131 [33] (with ARM9 32-bit ARM926EJ-S, 180Mhz, 32MB of SDRAM, and 512MB NAND flash). One Raspberry Pi (RPi1) is used as IDet, to generate a detection notification, embed it in regular communication via steganography, and send this over the CAN network. The second Raspberry Pi (RPi2) simply receives the CAN messages and forwards

them via a forwarding socket to the third Raspberry Pi[2]. The third raspberry pi (RPi3) acts as the infected ECU and the LPC-H3131 is attached to it via a USB2.0 interface. A client application has been developed to receive CAN messages and forward them to the TA (in RPi3). To develop the TA, we have ported OP-TEE (Open Portable Trusted Execution Environment) [38] to RPi3. In the TA, the forwarded message from the CA is received and the secret notification is extracted. If it is a detection notification, a new message is generated with the command embedded, which is sent back to the CA for writing to the FTL. Additionally, we have ported [40] and modified an open source NAND flash manager OpenNFM [22] to the LPC-H3131. Our OpenNFM modifications consist of backing up the mapping table to a reserved area, reserving a special command area, decoding the embedded command, and rolling back the previous mappings to restore the original data. We choose n (number of CAN messages) to be 16, so for the 64 bit data field of CAN messages, we embed a $l = 64$ bit notification in a 1024 bit message. Consequently, for both embedding and extracting the notification in IDet, TA, and FTL, we employ the feistel network cipher, a pseudo-random permutation, with AES as the round function to produce a $\log_2 1024 = 10$ bit output.

Evaluating Recovery and Communication. To evaluate the time for processing the hidden communications and recovering the ECU firmware to before an attack, we timed three phases: 1) time in IDet (encoding the detection notification), 2) time in TA (extracting the notification in TA, embedding the command in a new message, and sending the new message to FTL via a write request to the CA), and 3) time in FTL (extracting the notification in the FTL and subsequently performing recovery). The time of each phase has been measured 20 times and the results are summarized in Table 1. These results demonstrate that after malicious CAN messages are detected via intrusion detection and signal analysis mechanisms, a hidden detection notification can quickly be transmitted to the compromised ECU and the firmware can be restored to the prior version. Note that the majority of time spent in the FTL is for recovery, with around .01 s for decoding the notification. The time in TA is significantly more than expected. This is likely due to the overhead from computation performed (extracting the notification, generating a new message and embedding the command, sending this back to the CA), context switching overhead, and the limited physical resources available in the secure world.

Table 1. Average time (measured in seconds) in IDet, TA, and FTL.

IDet Time (s)	TA Time (s)	FTL Time (s)
.0012988	7.1065939	1.4264553

[2] This is necessary due to unresolved compatibility issues between the CAN drivers and specific OP-TEE implementation.

Throughput Evaluation To evaluate the impact of our solution on normal operations with the flash storage device, we performed a throughput comparison between the unmodified OpenNFM and our modified version. For these benchmarks, we used the fio benchmarking tool [12] and performed random write (RW), random read (RR), sequential write (SW), and sequential read (SR) benchmarks. Our results from these tests are summarized in Fig. 6.

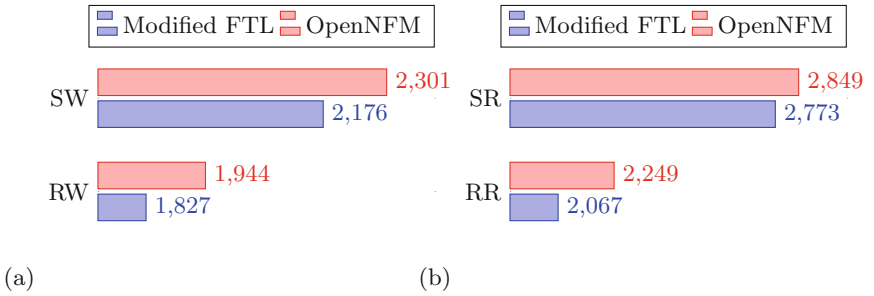

Fig. 6. Throughput comparison (KB/s).

To fully evaluate the impact of the observed differences, we perform a statistical analysis of the results. The values reflected in Fig. 6 are the mean of 30 benchmarks for each type. Evaluations of the differences in throughput are based on the mean values. Subsequently, a T-Test was performed to determine the statistical significance of the observed results for RW (random write), RR (random read), SW (sequential write), and SR (sequential read). The p values were $< .00001$ for all tests, which is less than the .05 level, indicating a statistically significant difference in throughput. These throughput results demonstrate that the implementation of our solution results in a small performance cost on normal read/write operations. Evaluating the extent of these differences, the average difference in throughput is 5.7%.

6 Related Work

6.1 Data Recovery in Embedded Systems

Chen et al. developed FFRecovery [18], an fine-grained data recovery system specifically designed for flash memory devices. FFRecovery employs file system forensics techniques to restore metadata, while utilizing the out-of-place update feature in the flash translation layer to extract raw data. Xie et al. proposed MobiDR, a data recovery framework for mobile devices [45]. MobiDR can recover user data to the corruption point to defend against OS-level malware by utilizing the cloud server's version control capability and the hardware features of the local device. Guan et al. introduced Bolt, a system aimed at restoring the operating system without requiring a reboot. Bolt makes use of ARM TrustZone to backup

the memory snapshot and utilizes a customized firmware to save the snapshot of flash memory. This approach enables efficient recovery of the system to a clean state after it is compromised. Huang et al. presented FlashGuard [28], a firmware-level recovery system against ransomware attacks that provides rapid and efficient data recovery without the need for explicit backups. FlashGuard makes use of the out-of-place update feature in solid state drive to retrieve data encrypted by ransomware. Wang et al. designed TIMESSD [43]. To recover user data that are compromised by the malware, TIMESSD leverages the out-of-place update characteristics in SSD to retain the states of history storage, and the recovery process is achieved via calling rollback functions. Furthermore, Baek et al. [15], Wang et al. [42], and Min et al. [34] have integrated the ransomware detection component into the flash translation layer. This integration can save the space that would otherwise be allocated for backing up invalid data in local storage if ransomware is not detected.

Different from the existing works for data recovery in the embedded systems, our design targets the ECU firmware recovery in a vehicle system which is much more critical than the traditional embedded systems. This type of critical systems has a much higher demand in a few aspects. First, it requires the restoration to be performed in real time, i.e., the extra overhead caused by the design should be minimized. Second, upon restoration, the malware is still present as it is difficult for the user/driver to block the malware in the compromised ECU, i.e., the restoration needs to happen when the malware is still present. Third, the malware detector comes from another entity in the vehicle and the communication between the detector and the FTL need to be adapted to the communication network inside a vehicle. Also, it needs to be protected and avoids being disturbed by the compromised ECU firmware.

6.2 CAN Intrusion and Malicious ECU Defenses

Kwon et al. proposed a mitigation mechanism against CAN intrusion by configuring impacted ECUs and ignoring malicious message IDs [32]. They proposed using intrusion detection to send messages to malicious ECUs and reconfigured them to a good state, but this state provides less functionality, and they did not provide any implementation or mechanism toward providing these guarantees. Han et al. evaluated the use of survival analysis for intrusion detection, with higher than 97% detection accuracy over all tested vehicles [26]. Yang et al. [46] designed a mechanism for detecting spoofing attacks from unrecognized ECUs by authenticating CAN data frame IDs. Using a recurrent neural network (RNN), they authenticated sender identity based on fingerprint signals. Further, Cho and Shin used recursive least squares to construct a baseline of ECU clock behavior for developing an anomaly-based intrusion detection system which identifies the malicious ECU [20].

7 Conclusion

In this work, we have designed a new framework for connected and autonomous vehicles to defend against the ECU code injection attacks, by rolling back the compromised ECU firmware to a good prior state. Our design has taken advantage of various existing hardware features equipped with the ECU to securely manage and efficiently perform the recovery process. We have implemented a prototype for the proposed framework and demonstrated its effectiveness at performing real-time recovery in a simulated in-vehicle testbed.

Acknowledgments. This work was supported by US National Science Foundation under grant number 2225424-CNS, 1928349-CNS, and 2043022-DGE.

References

1. https://www.nxp.com/products/processors-and-microcontrollers/s32-automotive-platform/s32z-and-s32e-real-time-processors:S32Z-E-REAL-TIME-PROCESSORS
2. https://www.xilinx.com/products/boards-and-kits/zcu104.html#information
3. https://www.nxp.com/design/designs/s32g3-vehicle-networking-reference-design:S32G-VNP-RDB3
4. https://www.nxp.com/design/designs/s32k3-automotive-telematics-box-t-box-reference-design-board:S32K3-T-BOX
5. Autonomous vehicle data storage - premio inc. https://premioinc.com/pages/autonomous-vehicle-data-storage
6. Can - Automotive Basics. https://automotivetechis.wordpress.com/2012-06-01-can-basics-faq/
7. Everything you need to know about performing an ECU reset. https://www.way.com/blog/ecu-reset/
8. Memory use in automotive - electronic products. https://www.electronicproducts.com/memory-use-in-automotive/
9. Raspberry pi 3 model b+. https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/
10. Trustzone for cortex-a - arm®. https://www.arm.com/technologies/trustzone-for-cortex-a
11. Trustzone for cortex-m - arm®. https://www.arm.com/technologies/trustzone-for-cortex-m
12. fio (2014). http://freecode.com/projects/fio
13. Applying over-the-air updates in safely automotive ECUS (2021). https://www.nxp.com/company/blog/applying-over-the-air-updates-in-safely-automotive-ecus:BL-OTA-IN-AUTO-ECUS
14. ECU programming guide (2021). https://ecutek.zendesk.com/hc/en-gb/articles/207345569-ECU-programming-guide

15. Baek, S., Jung, Y., Mohaisen, A., Lee, S., Nyang, D.: SSD-insider: internal defense of solid-state drive against ransomware with perfect data recovery. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pp. 875–884. IEEE (2018)
16. Bielawski, R., Gaynier, R., Ma, D., Lauzon, S., Weimerskirch, A.: Cybersecurity of firmware updates. Technical Report DOT HS 812 807, University of Michigan. Transportation Research Institute and University of Michigan, Dearborn and Volkswagen Group of America (Herndon, VA) (October 2020), https://rosap.ntl.bts.gov/view/dot/55729
17. Chattopadhyay, A., Lam, K.Y., Tavva, Y.: Autonomous vehicle: Security by design. IEEE Trans. Intell. Transp. Syst. **22**(11), 7015–7029 (2021). https://doi.org/10.1109/TITS.2020.3000797
18. Chen, N., Dafoe, J., Chen, B.: Poster: data recovery from ransomware attacks via file system forensics and flash translation layer data extraction. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 3335–3337 (2022)
19. Chen, N., Xie, W., Chen, B.: Combating the OS-level malware in mobile devices by leveraging isolation and steganography. In: Zhou, J., et al. (eds.) Applied Cryptography and Network Security Workshops: ACNS 2021. LNCS, vol. 12809, pp. 397–413. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81645-2_23
20. Cho, K.T., Shin, K.G.: Fingerprinting electronic control units for vehicle intrusion detection. In: 25th USENIX Security Symposium (USENIX Security 16), pp. 911–927. USENIX Association, Austin (2016). https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/cho
21. Choi, W., Jo, H.J., Woo, S., Chun, J.Y., Park, J., Lee, D.H.: Identifying ECUS using inimitable characteristics of signals in controller area networks. IEEE Trans. Veh. Technol. **67**(6), 4757–4770 (2018). https://doi.org/10.1109/TVT.2018.2810232
22. Code, G.: Opennfm. https://code.google.com/p/opennfm/
23. Guan, L., et al.: Supporting transparent snapshot for bare-metal malware analysis on mobile devices. In: Proceedings of the 33rd Annual Computer Security Applications Conference, pp. 339–349 (2017)
24. Hackenberg, R., Weiss, N., Renner, S., Pozzobon, E.: Extending vehicle attack surface through smart devices (2017)
25. Hamada, Y., Inoue, M., Ueda, H., Miyashita, Y., Hata, Y.: Anomaly-based intrusion detection using the density estimation of reception cycle periods for in-vehicle networks. SAE Int. J. Transport. Cybersecur. Privacy **1** (2018). https://doi.org/10.4271/11-01-01-0003
26. Han, M.L., Kwak, B.I., Kim, H.K.: Anomaly intrusion detection method for vehicular networks based on survival analysis. Veh. Commun. **14**, 52–63 (2018). https://doi.org/10.1016/j.vehcom.2018.09.004
27. Hoppe, T., Kiltz, S., Dittmann, J.: Applying intrusion detection to automotive it-early insights and remaining challenges. J. Inf. Assur. Secur. (JIAS) **4**, 226–235 (2009)
28. Huang, J., Xu, J., Xing, X., Liu, P., Qureshi, M.K.: Flashguard: leveraging intrinsic flash properties to defend against encryption ransomware. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 2231–2244 (2017)
29. Köhler, J., Förster, H.: Trusted execution environments in vehicles. ATZelektronik worldwide **11**(5), 36–41 (2016). https://doi.org/10.1007/s38314-016-0074-y

30. Kim, B., Park, S.: ECU software updating scenario using OTA technology through mobile communication network. In: 2018 IEEE 3rd International Conference on Communication and Information Systems (ICCIS), pp. 67–72. IEEE (2018)
31. Klinedinst, D.J., King, C.: On board diagnostics: Risks and vulnerabilities of the connected vehicle. CERT Division, Software Engineering Institute, Carnegie Mellon University, April, White paper (2016)
32. Kwon, H., Lee, S., Choi, J., Chung, B.H.: Mitigation mechanism against in-vehicle network intrusion by reconfiguring ECU and disabling attack packet. In: 2018 International Conference on Information Technology (InCIT), pp. 1–5 (2018). https://doi.org/10.23919/INCIT.2018.8584882
33. Ltd., O.: Lpc-h3131. https://www.olimex.com/Products/ARM/NXP/LPC-H3131/. Accessed 30 June 2023
34. Min, D., et al.: Amoeba: an autonomous backup and recovery SSD for ransomware attack defense. IEEE Comput. Archit. Lett. **17**(2), 245–248 (2018)
35. Murvay, P.S., Groza, B.: Source identification using signal characteristics in controller area networks. IEEE Signal Process. Lett. **21**(4), 395–399 (2014). https://doi.org/10.1109/LSP.2014.2304139
36. News, T.H.: Hackers take Remote Control of Tesla's Brakes and Door locks from 12 Miles Away. https://thehackernews.com/2016/09/hack-tesla-autopilot.html
37. Nie, S., Liu, L., Du, Y., Zhang, W.: Over-the-air: how we remotely compromised the gateway, BCM, and autopilot ECUs of tesla cars. Briefing, Black Hat, vol. 91 (2018)
38. OP-TEE. Op-tee documentation. https://optee.readthedocs.io/en/latest/general/about.html Accessed 30 June 2023
39. Stevebell. A Pivotal Year for Black Hat Cyber Attacks on Connected Cars - TU Automotive (2008). https://www.tu-auto.com/2018-a-pivotal-year-for-black-hat-cyber-attacks-on-connected-cars/
40. Tankasala, D., Chen, N., Chen, B.: A step-by-step guideline for creating a testbed for flash memory research via LPC-h3131 and opennfm (2020)
41. Thing, V.L., Wu, J.: Autonomous vehicle security: a taxonomy of attacks and defences. In: 2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 164–170 (2016). https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData.2016.52
42. Wang, P., Jia, S., Chen, B., Xia, L., Liu, P.: Mimosaftl: adding secure and practical ransomware defense strategy to flash translation layer. In: Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, pp. 327–338 (2019)
43. Wang, X., Yuan, Y., Zhou, Y., Coats, C.C., Huang, J.: Project almanac: a time-traveling solid-state drive. In: Proceedings of the Fourteenth EuroSys Conference 2019, pp. 1–16 (2019)
44. Wen, H., Chen, Q.A., Lin, Z.: 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), pp. 949–965. USENIX Association (2020). https://www.usenix.org/conference/usenixsecurity20/presentation/wen
45. Xie, W., Chen, N., Chen, B.: Enabling accurate data recovery for mobile devices against malware attacks. In: 18th EAI International Conference on Security and Privacy in Communication Networks (2022)

46. Yang, Y., Duan, Z., Tehranipoor, M.: Identify a spoofing attack on an in-vehicle can bus based on the deep features of an ECU fingerprint signal. Smart Cities **3**(1), 17–30 (2020). https://doi.org/10.3390/smartcities3010002
47. Zhang, T., Antunes, H., Aggarwal, S.: Defending connected vehicles against malware: challenges and a solution framework. IEEE Internet Things J. **1**(1), 10–21 (2014). https://doi.org/10.1109/JIOT.2014.2302386