ShadowAuth: Backward-Compatible Automatic CAN Authentication for Legacy ECUs

Sungwoo Kim
sk@purdue.edu
Purdue University
West Lafayette, Indiana, USA

Gisu Yeo
charcode@pusan.ac.kr
Pusan National University
Busan, Republic of Korea

Taegyu Kim
tgkim@psu.edu
The Pennsylvania State University
University Park, Pennsylvania, USA

Junghwan "John" Rhee
jrhee2@uco.edu
University of Central Oklahoma
Edmond, Oklahoma, USA

Yuseok Jeon
ysjeon@unist.ac.kr
UNIST
Ulsan, Republic of Korea

Antonio Bianchi
antoniob@purdue.edu
Purdue University
West Lafayette, Indiana, USA

Dongyan Xu
dxu@purdue.edu
Purdue University
West Lafayette, Indiana, USA

Dave (Jing) Tian
daveti@purdue.edu
Purdue University
West Lafayette, Indiana, USA

ABSTRACT

Controller Area Network (CAN) is the de-facto standard in-vehicle network system. Despite its wide adoption by automobile manufacturers, the lack of security design makes it vulnerable to attacks. For instance, broadcasting packets without authentication allows the impersonation of electronic control units (ECUs). Prior mitigations, such as message authentication or intrusion detection systems, fail to address the compatibility requirement with legacy ECUs, stealthy and sporadic malicious messaging, or guaranteed attack detection. We propose a novel authentication system called ShadowAuth that overcomes the aforementioned challenges by offering backward-compatible packet authentication to ECUs without requiring ECU firmware source code. Specifically, our authentication scheme provides transparent CAN packet authentication without modifying existing CAN packet definitions (e.g., J1939) via automatic ECU firmware instrumentation technique to locate CAN packet transmission code, and instrument authentication code based on the CAN packet behavioral transmission patterns. ShadowAuth enables vehicles to detect state-of-the-art CAN attacks, such as bus-off and packet injection, responsively within 60ms without false positives. ShadowAuth provides a sound and deployable solution for real-world ECUs.

CCS CONCEPTS

• **Security and privacy** → **Security protocols**; **Authentication**; *Embedded systems security*; **Intrusion detection systems**.

ASIA CCS '22, May 30–June 3, 2022, Nagasaki, Japan
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9140-5/22/05.
https://doi.org/10.1145/3488932.3523263

KEYWORDS

control area network; authentication; binary analysis; electronic control unit

ACM Reference Format:
Sungwoo Kim, Gisu Yeo, Taegyu Kim, Junghwan "John" Rhee, Yuseok Jeon, Antonio Bianchi, Dongyan Xu, and Dave (Jing) Tian. 2022. ShadowAuth: Backward-Compatible Automatic CAN Authentication for Legacy ECUs. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (ASIA CCS '22), May 30–June 3, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3488932.3523263

1 INTRODUCTION

Modern vehicles use the Control Area Network (CAN) protocol to operate electronic control units (ECUs). Those ECUs are responsible for controlling driving operations, such as engine and brake controllers [17], and sharing vehicle status (e.g., driving speed) among in-vehicle devices [34]. The CAN standard was originally designed without security in mind. For this reason, it lacks authentication during the communications between ECUs. Therefore, if any ECU is compromised, the entire CAN bus is subject to attacks, as shown in previous works where attackers compromise ECUs over an Internet connection [16], Bluetooth [61], and phone calls [64]. Compromised ECUs can cause vehicle malfunctions by falsifying the car status data (e.g., revolutions per minute (RPM)), impersonating other ECUs (e.g., engine, transmission, and brake control), potentially leading to the lost control of a vehicle [34].

Although attacks often start from external inputs, the ECUs without direct external inputs are not exceptions from potential attacks. Typically, safety-critical ECUs, such as the ones for engines, transmission, and brakes, are not directly accessible from external input interfaces (such as Bluetooth). However, attackers can still reach them indirectly after compromising other connected ECUs. As a result, an attack launched from non-safety-critical ECUs can eventually impersonate [16, 34, 61] safety-critical ECUs to inject malicious CAN packets or suppress [7, 50] safety-critical ECUs from sending the CAN packets.

The root cause of CAN insecurity is its inability to reject malicious CAN packets. Mitigating such malicious packets requires identifying the sender of each packet and verifying whether it is an expected sender. Message authentication code (MAC) is one approach to authenticate CAN packets. Specifically, MAC protects benign ECUs from impersonation attacks by simply ignoring the CAN packets with invalid credentials, as proposed in the previous works [4, 14, 18, 21, 22, 30, 36, 43, 46, 47, 56–58].

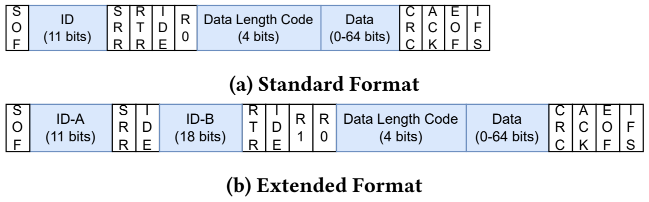
However, there are currently two challenges deterring wide adoption of MAC into CAN packets: the incompatibility with (1) the existing CAN packet definitions (e.g., SAE's J1939 [26] or manufacturer definitions [28]) and (2) the real-time constraint of vehicle operations due to high latency of packet acceptance. These incompatibilities mandate redesign of every firmware to adopt MAC. Unfortunately, the number of ECUs in each vehicle ranges from thirty to hundreds produced by third-party vendors in multiple tiers [15]. It is prohibitively infeasible for every vendor to roll out new firmware, implying numerous administrative challenges and compliance efforts. In addition, such a redesign requires a significant coordination effort among different vendors to ensure interoperability among different ECUs.

To overcome the aforementioned challenges, we propose ShadowAuth, a novel CAN authentication framework to support backward-compatible packet authentication, using automatic ECU binary firmware instrumentation. ShadowAuth has three novel features: (1) enabling message authentication in a way that is backward-compatible with the existing CAN packet definitions, (2) automatically embedding authentication functionality into the binary firmware without requiring source code, and (3) adhering to the real-time constraint of vehicle operations.

To automatically embed authentication functionality into ECU firmware, ShadowAuth locates CAN packet transmission functions using both static and dynamic analysis guided by their packet transmission behavioral patterns through a CAN peripheral device (i.e., inspecting instructions to control a CAN peripheral device).

Furthermore, we propose a new MAC scheme to provide full compatibility with (1) legacy packet definitions and the CAN standard and (2) real-time constraints. Our scheme introduces only one extra packet type called "authentication packet" without modifying the legacy "operational packet" (e.g., a brake control packet). Such an authentication packet is used to authenticate each operational packet. Specifically, our scheme authenticates operational packets paired with the corresponding authentication packets based on an *"accept-first-authenticate-later"* policy, which prioritizes the acceptance of each operational packet first to comply with the real-time constraints of vehicle control operations without deferring the process of operational packets.

We applied ShadowAuth to three different ARM-based ECUs supporting a wide range of real-world automotive vendors, such as BMW, Chevrolet, and Ford. Our evaluation results show that ShadowAuth successfully implants authentication functions into the existing ECU firmware without requiring any source code or prior knowledge of the firmware. Furthermore, ShadowAuth shows an effective detection of impersonation and bus-off attacks within 60ms in the worst case without any false positives. We believe our backward-compatible design makes ShadowAuth practical and deployable for existing vehicle hardware without any redesign of the existing CAN packet definitions.



(a) Standard Format

(b) Extended Format

Figure 1: CAN Data Frame Description

In summary, the main contributions of this paper are the following:

- We propose ShadowAuth, a fully backward-compatible, real-time, and low-overhead authentication scheme for CAN bus.
- We design an automatic and architecture-agnostic method to implant an authentication function into existing ECU firmware via binary instrumentation.
- Our evaluation shows that ShadowAuth can mitigate intrinsic vulnerabilities of CAN, such as the bus-off attacks within 60ms with 100% accuracy.

Our code is available at https://github.com/purseclab/ShadowAuth.

The remainder of this paper is organized as follows: Section 2 explains the background; Section 3 defines the threat and trust model; Section 4 presents the design of ShadowAuth; Section 5 shows the detailed implementation of ShadowAuth; Section 6 presents our evaluation result; Section 7 discusses observations from our evaluation, current limitations, and future directions of this work; Section 8 discusses related works, and Section 9 concludes the paper.

2 BACKGROUND

2.1 Control Area Network

CAN data frame. A CAN data frame consists of CAN fields, as shown in Figure 1. ECU firmware is responsible for writing the `ID`, `Data`, and `DLC` fields. The others are updated by the CAN peripheral device attached to the ECU board. The `ID` field (or CAN ID) indicates the type of content of the data field. A single ECU can send multiple CAN IDs, and the same CAN ID can be shared by multiple ECUs. The `DLC` field represents the length of `Data`. The `Data` field contains vehicle status information. Specifically, each offset of this field contains the value of different vehicle status, such as the speed of a wheel, the fuel level, or the degree of acceleration. Further, this field can be encoded in various formats. Manufacturers can use open standards (e.g., J1939 [26]) or make their own definitions (e.g., Volkswagen transport protocol [28]) to indicate which `ID` field describes which `Data` field, the offsets, and the lengths to describe status values. The descriptions of other fields can be found in [29]. In this paper, we will use CAN data frame and CAN packet interchangeably.

Bus topology. CAN uses a bus topology that does not distinguish between the sender nodes and the receiver nodes. In this network, a node in the CAN bus broadcasts its packets to all nodes, including the sender itself. This topology has the advantage of reducing the frame size because frames do not have to contain the sender's and receiver's information. Every receiver node receives every packet regardless of relevance. As discussed before, this simple design allows impersonation attacks due to missing authentication of senders and receivers.

Priority control. The CAN bus arbitrates simultaneous accesses by prioritizing CAN frames using the ID field. When two or more ECUs attempt to send packets together, the packet with a lower ID number has a higher priority over the others. However, if their IDs are the same and the packets conflict in other fields (e.g., data field), one ECU perceives that the CAN bus reads a different bit to its transmission. In this case, the ECU stop sending the frame and start to transmit an error frame. In such a manner, the CAN standard prevents the simultaneous transmission of the same CAN ID number.

Error handling. When two or more ECUs try to send CAN frames simultaneously, their frames will collide, and an error happens. If an ECU continuously generates errors (e.g., frame collision) in a short time, the node detaches itself from the CAN bus to protect the bus and other nodes. Each ECU has an error counter which increases whenever the node encounters errors. It has two error counters: Transmit Error Counter (TEC) and Receive Error Counter (REC). The TEC increases when an ECU fails to transmit its frame, and the REC increases when an ECU fails to receive any frame. Both counters decrease when ECUs succeed in sending and receiving a frame. These changes in the counter value cause an ECU state transition.

Specifically, the default state of an ECU is the *error active* state if an ECU's TEC **and** REC are less than 128. In this state, an ECU sends an *active error frame* when it fails to transmit frames. In particular, an *active error frame* starts with six dominant sequential bits ($000000_{(2)}$). Those bits prevent other ECUs from finishing a frame transmission, causing other ECUs to retransmit frames.

If either TEC **or** REC becomes greater than or equal to 128 but less than 256, an ECU changes its state to the *error passive* state, where an ECU sends a *passive error frame* when this ECU fails to transmit frames instead of an *active error frame*. Since the *passive error frame* starts with six sequential recessive bits ($111111_{(2)}$), all the other ECUs are not aware of the errors due to the analog features in the CAN bus [50]. Unlike an *active error frame*, a *passive error frame* does not cause other ECUs to retransmit frames.

An ECU changes its state to the *bus-off* state if an ECU's TEC **or** REC reaches 256. In the *bus-off* state, ECUs detach themselves from the CAN bus and stop transmitting or receiving frames. By exploiting the aforementioned feature, a compromised ECU can deliberately trigger packet collision on the CAN bus, eventually forcing a victim ECU to switch to the *bus-off* state [7, 50].

2.2 Electronic Control Unit

Safety-critical ECU. For the safety of drivers, passengers, and pedestrians, many countries, including the United States, mandate vehicle manufacturers to apply ECUs to safety features such as Anti-lock Brake System (ABS) and Electronic Stability Control (ESC) (a.k.a. Electronic Stability Program (ESP)). These ECUs help drivers control vehicles in hazardous situations. For instance, ABS repetitively applies and releases brakes more than ten times per second to wheels with different pressures to protect vehicles from skidding on a road. ESC detects the driver's intent by collecting sensor values such as the location of a steering wheel, a wheel speed, and a vehicle gradient. If the driver's intent is different from the vehicle's status, ESC stabilizes the vehicle by controlling the engine power and brake pressures.

Memory Mapped Input Output. MMIO is the medium to communicate with peripheral devices attached to the board. Specifically, ECU firmware communicates with peripheral devices by accessing the MMIO addresses of an attached peripheral device with memory read and write instructions [32] (e.g., `LDR` and `STR` for ARM). Through these instructions, the CAN bus peripheral devices such as TLE8888, SN65HVD232, and MCP2517 interpret digital and analog signal.

Approach	Requirements		
	New Packet Definition	Delay in Delivery (Time)	New H/W
CANAuth [22]	✓	✓(N/A)	
Nilsson et al. [41]	✓	✓(N/A)	
LCAP [21]	✓	✓(N/A)	✓
TOUCAN [4]	✓	✓(5.79μs)	
VeCure [58]	✓	✓(50μs)	
CaCAN [36]	✓	✓(2.2-3.2μs)	✓
SECU [57]	✓		
LiBrA-CAN [18]	✓		
S2CAN [46]	✓	✓(75μs)	
MAuth-CAN [30]		✓(500μs)	
LiEA [47]		✓(N/A)	
HLPSL [14]		✓(N/A)	✓
vatiCAN [43]		✓(3300μs)	
VulCAN [56]		✓(201μs)	✓
ShadowAuth			

Table 1: Comparison of Previous MAC Approaches

2.3 Hash-based Message Authentication Code

Hash-based Message Authentication Code (HMAC) guarantees packet integrity using a cryptographic hash function. In this paper, we use a hash scheme based on a Pre-Shared Key (PSK) to authenticate packets. Specifically, the nodes sharing the same key and sequence numbers can authenticate packets via validating whether their senders are legitimate. As shown in Table 1, previous HMAC-based approaches are not backward-compatible due to (1) their requirements of packet format changes, (2) the violation of the real-time constraint caused by HMAC computation upon packet arrival, or (3) dependency of new hardware.

3 THREAT MODEL

We assume impersonation attacks through CAN bus communication from not safety-critical ECUs (e.g., infotainment unit) with a prior compromise as adversaries. The malicious packets can imitate and suppress the functionalities of ECUs [7, 50], such as wheels, airbags, or brakes, as well as launching "replay" attacks where an attacker can replay packets observed from a CAN bus as demonstrated in prior works [34].

Our work introduces an ECU application monitoring the CAN bus and processing authentication. The integrity of this ECU is assumed, as well as the firmware update capability of ECUs. Meanwhile, we do not consider any physical access to the protected vehicle or direct access to the CAN bus from an attacker (e.g., hardware attack). Also, we do not consider attacks that happen at the start of vehicles because they are easily noticeable by drivers. Furthermore, attacks against ECU firmware update is out of scope [23, 37, 38]. Also, ShadowAuth targets ECU manufacturers who have legal permission to modify the firmware running in the devices they produce. This threat model aligns with the typical ones used in the previous CAN security research [7–9, 33, 50].

As an example scenario, a driver accidentally clicks a phishing website in her vehicle infotainment unit, leading to a compromise of the system [16]. Consequently, an attacker gains the ability to send any packet on the CAN bus. To incur a car accident, the attacker impersonates a cruise control system's packets to increase the driving speed drastically. Unfortunately, the driver cannot immediately reduce her car's speed properly because the attacker prevents an ESC from sending and receiving packets through a bus-off attack [7, 50]. While she immediately turned her steering wheel, the ESC could not maintain the vehicle's balance because the CAN packets of the steering wheel's location could not reach the ESC, resulting in an accident.

4 DESIGN

There are three challenges to achieve authentication while maintaining compatibility with legacy systems.

C1: Compatibility with existing packet definitions. Implementing authentication requires extra information in CAN packets to exchange authentication packets among ECUs. Trimming existing `Data` fields or appending additional fields for authentication conflicts with the existing standards and causes incompatibility with the CAN network used in the existing vehicles. For example, J1939 [26] is a standardized CAN packet definition widely adapted by heavy-duty vehicles. As J1939 defines all offsets and length of the data in CAN packets, appending verification information to some `Data` fields requires the modification of the packet definition and all participating ECUs.

C2: Real-time constraint. Any extended delay in a CAN packet delivery might cause a safety issue because it can slow down the real-time response time of safety-critical functions such as engines and brakes. Thus, any defense systems for CAN buses, including authentication mechanisms or intrusion detection systems, should be cautious about any delay of packets.

C3: Source code constraint. Source code dependency of securing ECU solutions imposes more deployment challenges. A vehicle consists of hardware and software components from a variety of vendors [15], and the manufacturer might not have some code access to ECU firmware. For instance, the absence of legacy build tools, build tool features, third-party source code, or vendor support, introduces the necessity of binary rewriting at post-compile and link time [44, 62].

ShadowAuth has two phases of operations in an ECU's life cycle, as shown in Figure 2. In the firmware instrumentation phase, ShadowAuth implants packet authentication functions into firmware using static and dynamic analyses. To this end, ShadowAuth converts the ECU firmware into an intermediate representation (IR) and finds the CAN transmission function based on common CAN packet transmission behavior. ShadowAuth then generates trampoline code and rewrites the existing firmware (Section 4.1), enabling ECUs to send authentication packets. When an impersonation attack occurs, the packet verification in CAN Packet Authentication (Section 4.2) will fail, and ShadowAuth considers the CAN bus compromised and notifies a driver accordingly.

4.1 ECU Firmware Instrumentation

As the first step, ShadowAuth identifies CAN transmission functions automatically through both static and dynamic analyses. ShadowAuth first uses static analysis for shortening the list of CAN transmission function candidates through the following three heuristic rules:

(1) ShadowAuth detects the functions that can potentially write data into CAN peripheral devices. Firmware sets MMIO addresses first and writes data into those addresses. Since many of such addresses cannot be determined statically, ShadowAuth conservatively identifies functions that include instructions writing into memory (e.g., `str` for the ARM architecture). (2) ShadowAuth further chooses the functions whose total memory writing size is larger than 79 bits which are the minimum size to send a CAN packet, including the `ID` field (whose size is either 11 bits or 29 bits), the `Data` field (whose size is 64 bits), and lastly the `DLC` field (whose size is 4 bits), as illustrated in Figure 1 and Section 2.1. (3) ShadowAuth selects the functions with a basic block writing at least 64 bits of data into the memory. A CAN transmission function should write the content of the `Data` field to send CAN packets, which usually happens in a single block in general.

To confirm whether a candidate function actually generates CAN packets at runtime, we use dynamic analysis and configure two ECU boards for the CAN bus. The first one is running the target ECU firmware to be protected. The other one is the CAN bus monitor watching CAN packets. We set the breakpoints at the entry address of each function candidate through the debugger module attached to the target ECU board. The CAN bus monitor of ShadowAuth verifies whether the watched function indeed generates CAN packets. Once a CAN packet is detected, the watched function candidate is considered as a CAN packet transmission function.

Knowing all the CAN transmission functions, ShadowAuth instruments the ECU firmware via implanting the authentication code into the identified CAN transmission functions. Specifically, ShadowAuth uses the detour-based approach [5, 24, 48] to instrument the authentication code into the existing ECU firmware. A simplified example in Figure 3 illustrates the workflow. A `call` instruction of the candidate function (i.e., `send_packet`) is at the address `0x04` as shown on the left. This function is responsible for sending "operational" packets (e.g., break control packets). ShadowAuth inserts a jump code right after this call to generate and transmit "authentication" packets.

As shown in Figure 3, the `add` instruction at the address `0x8` is replaced with a `jmp` instruction to redirect the control flow to the trampoline code at `0x100`. The trampoline code first executes the original instruction overwritten by the `jmp` instruction. Then, it generates an HMAC by calling `calc_hmac` and uses the identified

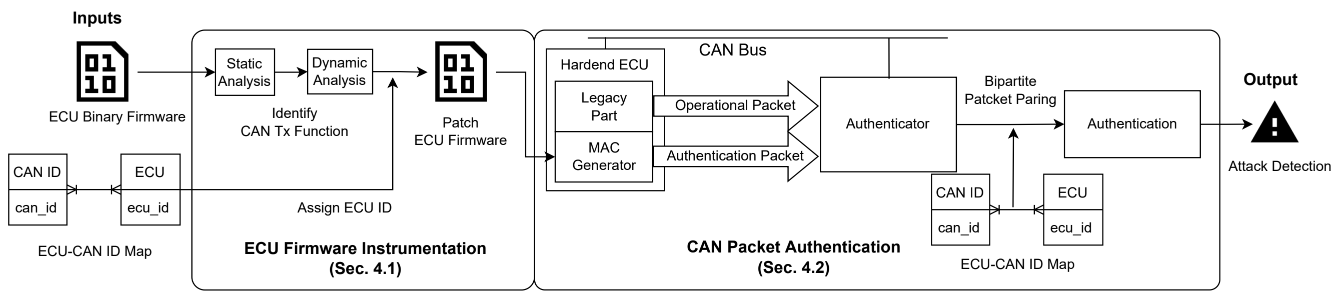
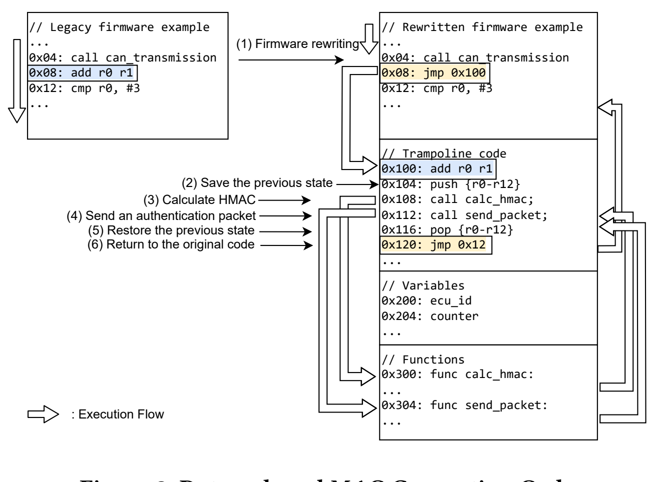


Figure 2: Overview of ShadowAuth



```
// Legacy firmware example
...
0x04: call can_transmission
0x08: add r0 r1
0x12: cmp r0, #3
...
```

```
// Rewritten firmware example
...
0x04: call can_transmission
0x08: jmp 0x100
0x12: cmp r0, #3
...

// Trampoline code
0x100: add r0 r1
0x104: push {r0-r12}
0x108: call calc_hmac;
0x112: call send_packet;
0x116: pop {r0-r12}
0x120: jmp 0x12
...

// Variables
0x200: ecu_id
0x204: counter
...

// Functions
0x300: func calc_hmac:
...
0x304: func send_packet:
...
```

(1) Firmware rewriting
(2) Save the previous state
(3) Calculate HMAC
(4) Send an authentication packet
(5) Restore the previous state
(6) Return to the original code
: Execution Flow

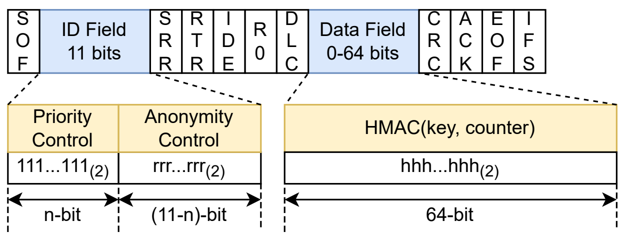
Figure 3: Detour-based MAC Generation Code

CAN transmission function (i.e., send_packet) to send an authentication packet. The trampoline then returns to the address 0x12 to resume the original control flow.

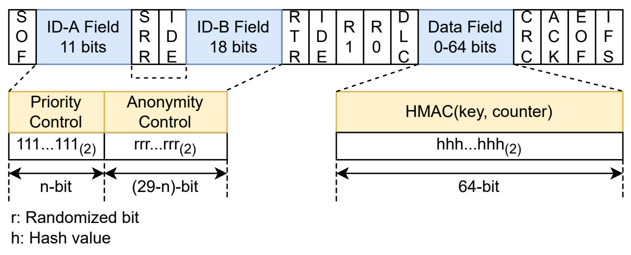
4.2 CAN Packet Authentication

In this section, we introduce the packet authentication process of ShadowAuth, which consists of the *HMAC generator* (embedded in existing ECUs' firmware, as described in Section 4.1) and *Authenticator*, a monitoring ECU on the CAN bus, as presented in Figure 2. Since the Authenticator is software, the manufacturers can deploy it atop gateway ECUs [53], minimizing the real-world deployment cost. After each unmodified previous operational packet is sent by an ECU, the HMAC generator sends an authentication packet to authenticate the operational packet on the CAN bus. This asynchronous design of our authentication scheme using authentication packets sends additional authentication packets after the corresponding original operational packets, helping ShadowAuth achieve compatibility with legacy systems (**C1**). To support this design, our authentication scheme includes ECU ID, counter, ECU-CAN ID map, and packet matching algorithm.

ECU ID. An ECU ID is assigned a unique identifier for each ECU, only shared with the Authenticator. The Authenticator knows all ECU IDs, but each ECU only knows its own ECU ID. Based on its ECU ID, an ECU generates authentication packets to help the Authenticator authenticate operational packets.



(a) Authentication Packet for Standard CAN Frame



(b) Authentication Packet for Extended CAN Frame

Figure 4: Authentication Packet Definition

Counter. An ECU maintains a counter to prevent replay attacks. Specifically, the counter starts from 0 and increases by 1 when an ECU sends an authentication packet. When authentication succeeds, the Authenticator synchronizes its counter per ECU.

This counter is reset to 0 when a vehicle restarts. This design makes ShadowAuth compatible with real-world ECUs [25] because many ECUs cannot store the counter value after a vehicle is turned off due to a lack of persistent memory (e.g., flash memory).

ECU-CAN ID map. An ECU-CAN ID map represents which ECUs are allowed to send which CAN IDs. We can obtain this map by leveraging the prior work [10, 35, 39]. Alternatively, ECU manufacturers can use their domain knowledge to find an ECU-CAN ID map. After we assign an ECU ID to each ECU and obtain the list of CAN IDs for each ECU, we build the ECU-CAN ID map and save it within the Authenticator.

Authentication packet structure. Authentication packets are defined using the standard and extended CAN frames, as shown in Figure 4. The CAN ID field consists of two sections, priority control and anonymity control. To prioritize operational packets without deterring their processing, we set the priority control field of the authentication packets as the lowest priority (**C2**), minimizing the real-time impact. Also, we randomize the anonymity control

Operational Packets	ECU-CAN ID Map	Possible Scenario	Possible Counter	
			ECU ID X	ECU ID Y
$o_1 o_2$	$X - o_1$	XX	+2	+0
	$Y - o_1$	XY	+1	+1
	$X - o_2$	YX	+1	+1
	$Y - o_2$	YY	+0	+2

Table 2: Example of Scenarios for Two Operational Packets

field to randomize the sending order of two or more authentication packets, resulting in unpredictable delays for each packet. This design prevents replay and correleation attacks which we will describe in Section 4.3. Authentication packets store an HMAC in the `data field` with length up to 64 bits.

Accept-first-authenticate-later policy. ShadowAuth follows an asynchronous *Accept-first-authenticate-later* policy. Specifically, (1) A sender ECU broadcasts an operational packet first followed by a corresponding authentication packet. (2) A receiver ECU and the Authenticator accept the operational packet and only the Authenticator accepts the authentication packet. (3) the Authenticator authenticates the operational packet with the authenticate packet on behalf of the receiver ECU. (4) If authentication fails, the Authenticator fires an alarm, indicating a potential attack detected.

This *Accept-first-authenticate-later* policy introduces the following advantages: First, it does not change legacy CAN packet definitions (**C1**). Second, this policy helps ECUs the real-time constraint of ECU firmware by minimizing the workload of authentication (**C2**). The HMAC generator sends authentication packets only after the prioritized operational packets are sent. Meanwhile, there is no extra overhead for receivers because a CAN peripheral device (attached to an ECU board) filters authentication packets automatically.

Authentication. Authentication succeeds if the Authenticator finds out the correct match between operational and authentication packets. Specifically, it calculates HMAC with an ECU ID and counter from the ECU-CAN ID map with the same cryptographic hash function as the HMAC generator. If the HMAC is identical to the authentication packet, we call it is a "match" between two packets, meaning that the operational packet is authentic.

Otherwise, the authentication fails if packets are not matched after a certain time. In particular, we defined the timeout value, the maximum waiting time to be matched for each packet. Theoretically, the longest waiting time occurs when all ECUs try to simultaneously send their operational packets. Since the CAN bus prioritizes operational packets, the authentication packet should wait for the number of ECU multiplied by the time to send an operational packet in the worst case. For instance, if a vehicle uses the J1939 standard and one hundred ECUs, an authentication packet arrives at most $0.6 \times 100ms$ after. Thus, the Authenticator can guarantee all packets will be matched within $60ms$ unless attacks happen.

Packet match. To match packets, the Authenticator should infer the correct sender ECU's ID and counter. However, these are not trivial because the ECU-CAN ID map might return one or more ECU IDs for one CAN ID, and one ECU ID might have one or more possible counters, causing multiple scenarios, as shown in Table 2. For instance, if there are two operational packets (e.g., o_1 and o_2) and two ECUs (e.g., X and Y) are eligible to send them,

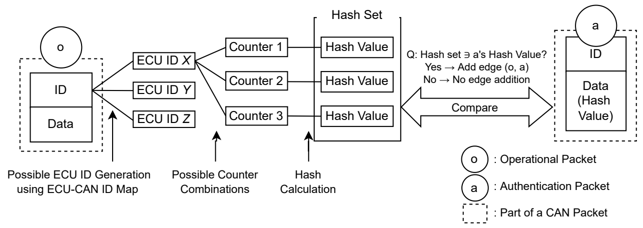


Figure 5: Temporal Pairing Process

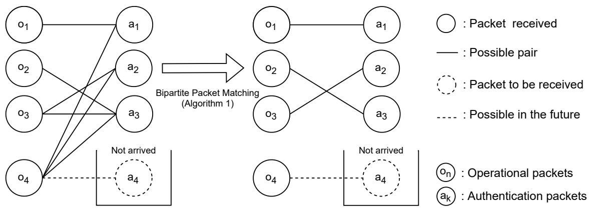


Figure 6: Exact Matching Process

the Authenticator should consider four scenarios, generating three possible counters(e.g., +2, +1, and +0) per ECU. Based on these scenarios, the Authenticator temporarily pairs all possible packets as shown in Figure 5. Each edge in Figure 6 represents a scenario that could have happened. For example, the edge $o_1 - a_1$ means the sender ECU of o_1 also could have sent a_1.

Based on Algorithm 1, the Authenticator verifies which combination of those scenarios can happen simultaneously. Specifically, the goal of Algorithm 1 is to find the vertex cover [6] of Figure 6. For instance, $o_1 - a_1$ and $o_4 - a_1$ cannot happen together because a_1 is overlapped. As shown in Algorithm 1, the arguments of the authentication function are a set of operational and authentication packets (V_o and V_a), all possible scenarios (E), and a packet requiring immediate authentication due to the timeout (o_t). As described in Line 2-3, it picks an operational and authentication packets (o and a) with the fewest adjacent nodes (o_1 and a_1). If both nodes are adjacent, the Authenticator temporarily pairs them by removing both vertices from V_o and V_a (Line 5 and 10-13). Authentication succeeds when the Authenticator succeeds in pairing the timeout packet (Line 6-8). Otherwise, authentication fails when there is no authentication packet corresponding to the timeout packet (Line 16-18).

4.3 Security Analysis

The design of ShadowAuth reflects the trade-off between security and real-world deployment constraint. We analyze a number of common attacks against CAN in the context of ShadowAuth, and specific attacks towards ShadowAuth.

Packet injection. To calculate valid HMACs, both an ECU ID and counter value are required as described in Section 4.2. Since the ECU ID is stored on the ECU's memory, extracting ECU ID requires access to firmware (e.g., through the physical access or hijacking firmware updates) which is out of the scope of our threat model, as mentioned in Section 3. Consequently, attackers could not inject valid operational packets without generating valid authentication packets.

Algorithm 1: Bipartite Packet Matching

```
Input:
$V_o$, // a set of operational packet
$V_a$, // a set of authentication packet
$E$, // a set of edges in Figure 6
$o_t$ // a packet requiring authentication
Output: True or False // True if authentication is successful
Function IsAuthenticated($G, o_t$):
    for $o \in V_o$ do
        for $a \in V_a$ do
            // Choose packets having less edges than others
            if $(o, a) \in E$ then
                if $o == o_t$ then
                    // If packet $o_t$ is authenticated by $a$
                    return True
                end
                // Temporary pairs $o$ with $a$
                $V_o = V_o - o$
                $V_a = V_a - a$
                break
            end
        end
        if $o \in V_o$ then
            // Even if temporal pairing fails
            return False;
        end
    end
```

Replay attacks. As described in Section 4.2, when HMAC generator sends an authentication packet, the counter in the next authentication packet is increased. As a result, attacks naïvely replaying the previously sent authentication packets will not be matched with any of the hash values that the Authenticator has.

However, advanced replay attacks across driving sessions can send both operational packets and their corresponding authentication packets to evade detection because the counter always starts from 0, and the ECU ID is not changed. Theoretically, if the attacker ECU can memorize all packets from the previous driving session (e.g., using a permanent storage), it can replay all the valid authentication packets. However, this attack mandates the two following requirements: (1) identifying which packets are the victim ECU's authentication packets and (2) preventing the victim ECU from sending any packets. Fortunately, satisfying both requirements is non-trivial because the attacker ECU has to infer a victim ECU's ID unless the ECU-CAN ID map is known. Moreover, the attacker ECU needs to launch a bus-off attack to stop the victim ECU from sending any operational packets.

Brute-force attacks. The short length of HMAC is the culprit of brute-force attacks, including finding collisions, enumerating all authentication packets, and brute-forcing HMAC. We chose this design as the trade-off between the security of HMAC and the deployment requirement. For instance, each CAN data frame is 64 bits, and we decided to use only one data frame for authentication to minimize the potential impact on the CAN. Nevertheless, ShadowAuth tolerates brute-force attacks. Furthermore, hash collision attacks can be mitigated by various methods such as using more data frames to pump up the 64-bit HMAC into 128-bit or even more (with risk of overloading the CAN network) or leveraging over-the-air ECU ID updates, as discussed in previous work [46].

Attackers can enumerate all HMACs to neutralize the authentication process, exploiting that one of the HMACs is *valid* (a.k.a. online brute-force attack). Fortunately, the Authenticator easily detects this attack through the timeout mechanism. Specifically, all authentication packets are supposed to be matched with operational packets. However, almost all enumerated authentication packets do not match with operational packets, leading to the timeout that the Authenticator considers the attack presence as described in Section 4.2

Additionally, the attacker can recover an ECU ID and a counter from an HMAC by calculating possible HMACs with all the combinations of ECU IDs and counters (a.k.a. offline brute-force attack). Nevertheless, ShadowAuth tolerates this attack because the synchronization between the victim ECU's and Authenticator's counter will be broken even if attackers manage to brute-force an HMAC and inject malicious authentication packets. As mentioned in Section 4.2, the ECU's counter only increases when it sends an authentication packet. Thus, if attackers impersonate a victim ECU by sending an authentication packet with a *valid* HMAC, only the Authenticator's counter increases, which breaks the counter synchronization. Eventually, the victim ECU's authentication packets will expire, leading to the authentication failure.

Bus-off attacks. Since the bus-off attacks also require packet injection, the Authenticator mitigates the attacks. To launch the attacks, an attacker ECU deliberately and consistently makes packet collisions with a victim ECU's packets, transitioning the victim ECU from the *error active* to *error passive* and *bus-off* state. Unfortunately, the Authenticator cannot detect the attacks when the victim is in the *error active* state because *active error frames* halt both ECU's transmission as described in Section 2. Nevertheless, if the victim ECU reaches the *error passive* state, the attacker ECU becomes responsible for sending authentication packets because the victim ECU's *passive error frames* do not hinder the attacker's packets anymore. Still, the attacker ECU fails to send authentication packets without the ECU-CAN ID map. Additionally, bus-off attacks cannot target the Authenticator because it does not send any CAN frame to the bus.

ECU-CAN ID map recovery. Since the ECU-CAN ID map is stored in the Authenticator's firmware, the map extraction requires physical access to an ECU inside of the target vehicle or firmware hijacking through MITM attacks during an over-the-air update. However, physical access is out of scope, as discussed in Section 3. On the other hand, firmware hijacking through MITM attacks can be mitigated, as discussed in Section 7. Therefore, we consider both of them are out of scope.

The correlation between operational and authentication packets might help recover the ECU-CAN ID map. However, the unpredictable time to send authentication packets prevents the correlation. Specifically, our authentication packets have the lowest priority than any of operational packets to minimize the potential negative performance impact on the CAN bus (e.g., congestion on the CAN bus). Furthermore, the CAN ID of authentication packets is randomized, as described in Section 4.2 to send the packets with an unpredictable delay, preventing the temporal correlations. Therefore, there is no strong correlation between the two types of packets because authentication packets are sent with an unpredictable delay. We will evaluate in Section 6 the delay of authentication packets.

5 IMPLEMENTATION

We implemented a static analysis prototype with a python script and pypcode [2] library. Also, we implemented dynamic analysis using GDB through STLink [52] and Black Magic Probe [1] to set hardware breakpoints. During this dynamic analysis, we monitored CAN bus based on Arduino Uno and a CAN bus shield with a MCP2515 CAN transceiver. Inspired by detours [24], we implemented our binary rewriter with python script with a BLAKE3's C implementation [45]. Further, we implemented the Authenticator with python script with a BLAKE3's python implementation [45].

6 EVALUATION

Experimental setup. We performed the backward-compatibility and security analysis evaluations with the existing CAN packet definition, SAE J1939 [26].

We performed the binary analysis evaluation on a set of ARM-based open-source ECU firmware, rusEFI [49], Styreenhet [65], and Rabbit ECU [13]. rusEFI uses TLE8888 as a CAN peripheral device and STM32F407 as the mainboard powered by a 168MHz 32-bit ARM Cortex-M4 processor with 1 MB flash memory and 192 KB SRAM. Styreenhet uses MCP2551 as a CAN transceiver and STM32F407 as the main board. Rabbit ECU uses SN65HVD232 as a CAN transceiver and Arduino. SN65HVD232 is an 84MHz ARM Cortex-M3 processor with 512 KB flash memory and 100 KB SRAM.

We evaluated our performance impact on the CAN traffic from two different real cars (2014 Kenworth T270 and 2015 Kenworth T660) [11], which contains over 37 million CAN packets and 186 CAN IDs. The 2014 Kenworth T270 traffic was collected during a 4-day cross-country round trip from Colorado to Michigan and back. The 2015 Kenworth T660 traffic was collected during driving around an industrial block with some hard braking maneuvers.

Research questions. In this section, we present our evaluation results, answering the following research questions:

- **RQ1**: How effectively does ShadowAuth prevent attacks, maintaining backward-compatibility? (Section 6.1)?
- **RQ2**: How well does our binary analysis find CAN transmission functions (Section 6.2)?
- **RQ3**: How much performance impact on ECUs is there due to ShadowAuth's authentication? (Section 6.3)?

6.1 Attack Detection

Based on the related works, we classified the CAN bus attacks into the following two categories: **packet injection** [16, 34, 61, 64], and **bus-off attacks** [7, 50]. In this section, we present how ShadowAuth detects all these attacks using two realistic case studies. Further, we will show the compatibility of ShadowAuth, studying the case of J1939.

6.1.1 *Case Study 1: Falsify a fuel level via an impersonation attack.*

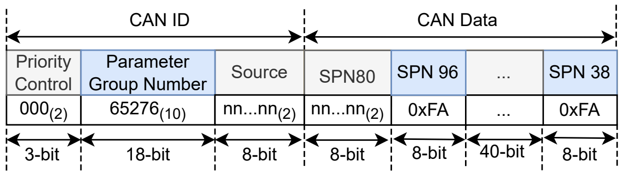


Figure 7: Attack Packet for Case Study 1

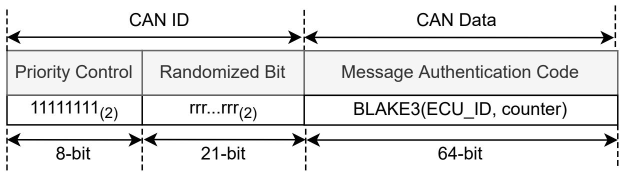


Figure 8: An Example of an Authentication Packet

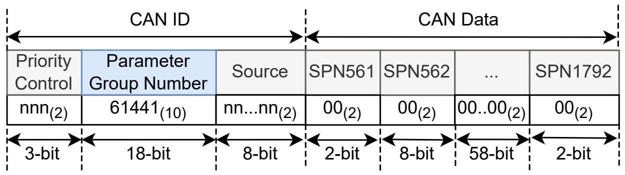


Figure 9: Attack Packet for Case Study 2

The goal of this attack is to confuse a driver by falsifying a fuel level. A compromised ECU consistently sends CAN packets that are compliant with the standard protocol formats. For instance, the attack packets follow the parameter group number 65276 (`PGN65276`), and the suspect parameter numbers 38 and 96 (`SPN38`, and `SPN96`), which are responsible for a fuel level in the J1939 standard [27], as illustrated in Figure 7.

The attack packet uses a particular value, `0xFA` in the SPN38 and SPN96 fields, denoting the current fuel level as 100%. The driver would fail to recognize the vehicle's status and run out of gas eventually. Note that there is no difference between falsified and legitimate packets because the CAN and J1939 standards do not require authentication. Hence, any packet following the standard can be easily forged.

Nonetheless, ShadowAuth defeats this attack as follows. First, ShadowAuth generates the list of sender ECUs of packets from the ECU-CAN ID map and calculates expected authentication packets from each sender using a hash of ECU ID with a counter as described in Figure 8. When the monitor ECU receives the falsified packet, ShadowAuth starts a timer for the authentication packet to arrive. Since the attacker cannot forge the authentication packets, the timer will expire eventually and ShadowAuth will report the authentication failure.

6.1.2 *Case Study 2: Deactivating ECUs via a bus-off attack.*

In this case, an adversary attempts to make ECUs unavailable using bus-off attacks [7, 50], e.g., detaching a brake controller from a CAN bus. If this attack is successful, a driver fails to activate brakes even if she pushes the brake pedal because her pedal cannot send out the pedal position to the brake unit.

The detail of the attack packet is described in Figure 9. Since the bus-off attacks target a brake controller, the attack packet uses the parameter group number 61441 (`PGN61441`) to control the brake position. The attack packet will collide with legitimate packets from the victim ECU, which leads the attacker and victim ECU's towards the `error passive` state. Fortunately, the victim ECU firstly turns into the `error passive` state, resulting in the attacker ECU's successful transmission. In this step, the Authenticator detects the bus-off attacks due to the missing authentication packets.

6.1.3 *Case Study 3: CAN packet compatibility.*

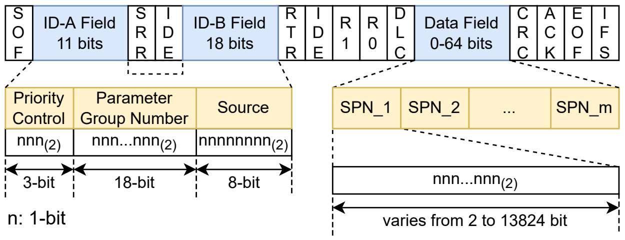


Figure 10: J1939 Packet Description

We tested the compatibility of our implementation with the existing CAN network by running ShadowAuth with the SAE J1939 [26] CAN packet definition, which is widely used for heavy-duty vehicles. Figure 10 illustrates the J1939 packet and its three fields. A suspect parameter number defines the data types, field offsets, and lengths for certain purposes. For instance, `SPN96` depicts the fuel level through 8-bit length data starting from the second byte of `Data` field. A parameter group number describes the list of SPN types and their offsets in the `Data` field. For example, when the PGN is 65276, the SPN data in the data field represents the dashboard information, such as washer fluid, fuel level, and engine oil pressure. Another field, priority control, prioritizes packets. The source field claims the sender of the packet, but it may not be accurate.

Since the authentication packets use the specific CAN IDs that J1939 packets do not use, ShadowAuth is compatible with the J1939 standard. As shown in Figure 8, we filled the authentication packet's `prioritize field` with sequential eight recessive bits ($11111111_{(2)}$) to avoid the conflict with J1939. It is the lower priority than the J1939's lowest priority, `PGN 65279`. Further, the rest content of the CAN ID (21-bit) is randomized as shown as (`0brrr...rrr`), maintaining the compatibility and hiding from replay and correlation attacks, as described in Section 4.3.

6.2 Effectiveness of Firmware Analysis

In this section, we present how effective our static and dynamic analyses are.

Static Analysis. The goal of our static analysis is to reduce the number of function candidates for dynamic analysis by distinguishing CAN transmission functions from others. Figure 11 shows the filtering rate: 69% (rusEFI), 45% (MS3), and 69% (Styreenhet).

Dynamic analysis. In dynamic analysis, CAN bus activities are monitored while ShadowAuth determines which functions are actively used for CAN transmission. Specifically, ShadowAuth generated hardware breakpoints on all functions candidates. We validate the final results by checking with the source code manually. All

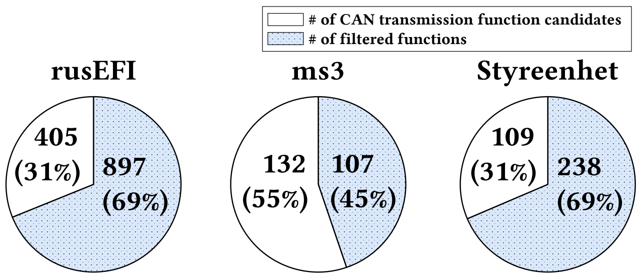


Figure 11: Non-target functions filtered by static analysis.

CAN transmission functions are detected with 100% accuracy for all tested platforms without ground truth study. Note that the source code is only used for the evaluation of accuracy. Our static and dynamic analyses do not require source code.

6.3 Runtime Overhead

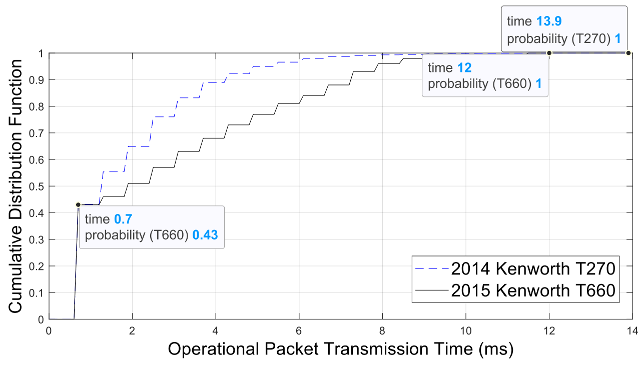


Figure 12: CDF of Operational Packet Transmission

To understand the runtime overhead introduced by ShadowAuth, we measured its impact on the CAN bus traffic and overhead of HMAC generation.

Impact on the CAN bus traffic. Since the authentication packets have lower priority than operational packets, the CAN bus conveys authentication packets when there is no on-going operational packet on the CAN bus. Hence, we measured how much time ECUs should wait before sending authentication packets. As shown in Figure 12, 43% of 2014 Kenworth T270 and 2015 Kenworth T660's authentication packets did not wait, meaning that the bus was already empty and ECUs can send authentication packets right after operational packets. Moreover, all authentication packets are sent in 14.0ms, far below the theoretic worst 60ms described in Section 4.2.

Overload of HMAC. We measured the CPU time and the code size of HMAC generator. Our results show that our HMAC generator incurs $4\mu s$ of execution, which is negligible compared to the minimum value of real ECU's transmission period, $1.5ms$ [50]. The code size increased by 212KB which is feasible for the commercial ECU's flash memory, e.g., TC1767, 2MB. 86% (182KB out of 212KB) of the space overhead is taken by the BLAKE3's algorithm, which varies by implementations. To further reduce the code size, we can alternatively choose other hashing algorithms such as SHA2, SHA3,

IDS	Trusted Base	False Positive (%)	False Negative (%)
CIDS [8]	Clock skew	0.055	0
VIDEN [9]	Voltage level	0.2	0.2
EASI [33]	Voltage edge	0	0.03

Table 3: Attack Detection Comparison with IDSs

and RIPEMD-160. For instance, RIPEMD-160 can take only 14KB which is less than 1%.

7 DISCUSSION

Rewriting ECU firmware of real-world ECUs. Rewriting real-world ECUs' firmware could be illegal in some countries, e.g., the United States [54, 55]. For this consideration, we conducted our experiment using open-source ECUs and validated our evaluation using the ground truth. For real-world use cases, we target the real automotive vendors, which should not have such administrative challenges. If secure boot is used to guarantee the ECU firmware integrity, we expect a re-signing process after the binary instrumentation from the vendor directly.

Burst traffic consideration for CAN bus. Most in-vehicle CAN bus traffic is periodic [9], and there is no traffic burst on the CAN bus. While ShadowAuth introduces additional packets due to authentication, they typically spread out along the traffic. By sending short random delays (e.g., a few ms) before sending authentication packets, ShadowAuth can diversify the traffic load. Based on our experiments, in practice, it is a rare to have traffic with any congestion. It is possible to sense the traffic load of the bus before sending out authentication packets to avoid the rare traffic bursts.

ECU firmware extraction during updates. A firmware update accesses its raw memory content. Therefore, extracting the firmware, including any keys during an update, is possible. Existing firmware update protection techniques [20, 23, 37, 38, 40, 42] can be used to mitigate this attack and complement ShadowAuth.

8 RELATED WORK

Packet authentication. There have been multiple prior works to address the missing authentication of CAN buses and ECUs [4, 14, 18, 21, 22, 30, 36, 41, 43, 47, 56–58]. Several approaches [4, 18, 21, 22, 36, 41, 57, 58] require significant manual effort to modify packet definitions and standards, which does not overcome **C1**. Specifically, they use a portion of the `Data` field for HMAC, which is already reserved for other purposes, as presented in Section 4. Thus, those methods impose high costs due to their backward incompatibility.

Another group of approaches [4, 14, 21, 22, 30, 41, 43, 46, 47, 56, 58] delays CAN packets delivery. As discussed in **C2**, if there is an excessive acceptance delay, the vehicle operations cannot be properly controlled and may lead to accidents. Those approaches did not evaluate the influence of such a long acceptance delay to the vehicle control, despite their use of costly cryptographic functions such as Elliptic-Curve [14]. In particular, ShadowAuth has a broader attack coverage than S2-CAN [46], covering remote attacks.

Intrusion detection system. Existing CAN intrusion detection systems (IDS) [8, 9, 33] have evolved to detect attacks by utilizing the physical characteristics of CAN packets, such as clock skews, voltage levels, and voltage edges. Their advantages include fast detection time and easy deployment by attaching additional hardware to the existing CAN bus. However, these side-channel-based approaches cannot guarantee 100% of accuracy, leading to inevitable false positives or false negatives, as shown in Table 3. Compared to such approaches, ShadowAuth achieves the authentication of CAN packets, guaranteeing attack detection.

Binary rewriter. There have been a large number of rewriter approaches introduced for multiple computer architectures such as x86 [3, 12, 24, 59, 60, 63] and ARM [19, 31, 51]. ShadowAuth used the detour-based binary rewriting approach [24, 51] because other approaches [3, 19, 31, 59, 60, 63] are relatively unstable compared to the detour-based approaches. For instance, binaries compiled from the C++ language contain multiple sections to store meta-information [59]. Hence, supporting C++ binary rewriting is an unsolved problem for non-detour-based approaches. ShadowAuth's design also avoids certain corner cases such as PC-relative jumps to make a proper patch, as mentioned in the previous work [31].

9 CONCLUSION

We propose ShadowAuth, a backward-compatible authentication scheme based on HMAC and automated binary rewriting. By using asynchronous authentication packets, ShadowAuth provides message authentication in the *existing legacy* CAN bus without violating the current CAN packet definitions and protocols. Moreover, ShadowAuth automatically implants this authentication scheme into the legacy firmware by accurately discovering CAN packet transmission functions and utilizing a trampoline-based binary rewriting mechanism. ShadowAuth achieves CAN packet authentication in 60ms after a packet arrival without source code dependency and packet acceptance latency.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and Dr. Haehyun Cho for their valuable comments and suggestions. We also thank the open-source contributors and maintainers for their technical help. This project was supported in part by Office of Naval Research (ONR) under Grant N00014-18-1-2674 and N00014-20-1-2128, National Science Foundation (NSF) under Award Number CNS-2145744, and Defense Advanced Research Projects Agency (DARPA) under contract number N6600120C4031. This project was also supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2021R1F1A1049822). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. Any opinions, findings, and conclusions in this paper are those of the authors and do not necessarily reflect the views of our sponsors.

REFERENCES

[1] 1BitSquared. 2021. *Black Magic Probe V2.1.* https://1bitsquared.com/products/black-magic-probe
[2] angr. 2021. *pypcode.* https://github.com/angr/pypcode

[3] Erick Bauman, Zhiqiang Lin, Kevin W Hamlen, et al. 2018. Superset Disassembly: Statically Rewriting x86 Binaries Without Heuristics.. In *NDSS*.

[4] Giampaolo Bella, Pietro Biondi, Gianpiero Costantino, and Ilaria Matteucci. 2019. TOUCAN: A ProTocol tO SecUre Controller Area Network. In *Proceedings of the ACM Workshop on Automotive Cybersecurity* (Richardson, Texas, USA) *(AutoSec '19)*. Association for Computing Machinery, New York, NY, USA, 3–8. https://doi.org/10.1145/3309171.3309175

[5] Bryan Buck and Jeffrey K. Hollingsworth. 2000. An API for Runtime Code Patching. *Int. J. High Perform. Comput. Appl.* 14, 4 (Nov. 2000), 317–329. https://doi.org/10.1177/109434200001400404

[6] Jianer Chen, Iyad A. Kanj, and Ge Xia. 2010. Improved upper bounds for vertex cover. *Theoretical Computer Science* 411, 40 (2010), 3736–3756. https://doi.org/10.1016/j.tcs.2010.06.026

[7] Kyong-Tak Cho and Kang G. Shin. 2016. Error Handling of In-Vehicle Networks Makes Them Vulnerable. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 1044–1055. https://doi.org/10.1145/2976749.2978302

[8] Kyong-Tak Cho and Kang G. Shin. 2016. Fingerprinting Electronic Control Units for Vehicle Intrusion Detection. In *Proceedings of the 25th USENIX Conference on Security Symposium* (Austin, TX, USA) *(SEC'16)*. USENIX Association, USA, 911–927.

[9] Kyong-Tak Cho and Kang G. Shin. 2017. Viden: Attacker Identification on In-Vehicle Networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1109–1123. https://doi.org/10.1145/3133956.3134001

[10] Wonsuk Choi, Hyo Jin Jo, Samuel Woo, Ji Young Chun, Jooyoung Park, and Dong Hoon Lee. 2018. Identifying ECUs Using Inimitable Characteristics of Signals in Controller Area Networks. *IEEE Transactions on Vehicular Technology* 67, 6 (2018), 4757–4770. https://doi.org/10.1109/TVT.2018.2810232

[11] Jeremy Daily. 2021. *Heavy Vehicle CAN Data*. https://www.engr.colostate.edu/~jdaily/J1939/candata.html

[12] Sushant Dinesh, Nathan Burow, Dongyan Xu, and Mathias Payer. 2020. RetroWrite: Statically Instrumenting COTS Binaries for Fuzzing and Sanitization. In *2020 IEEE Symposium on Security and Privacy (SP)*. 1497–1511. https://doi.org/10.1109/SP40000.2020.00009

[13] Rabbit ECU. 2021. *Rabbit ECU Project*. https://mdac.com.au/rabbit-ecu-project/

[14] Samir Fassak, Younes El Hajjaji El Idrissi, Noureddine Zahid, and Mohamed Jedra. 2017. A secure protocol for session keys establishment between ECUs in the CAN bus. In *2017 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. 1–6. https://doi.org/10.1109/WINCOM.2017.8238149

[15] R Fletcher, A Mahindroo, N Santhanam, and A Tschiesner. 2020. The case for an end-to-end automotive-software platform. *McKinsey & Company* (2020).

[16] I.Matteucci G.Costantino. 2020. KOFFEE - Kia OFFensivE Exploit Attack Surface in Automotive IoT. https://sowhat.iit.cnr.it/pdf/IIT-20-2020.pdf

[17] Robert Bosch GmbH. 2021. *Engine Control Unit*. https://www.bosch-mobility-solutions.com/en/solutions/control-units/eengine-control-unit/

[18] Bogdan Groza, Stefan Murvay, Anthony van Herrewege, and Ingrid Verbauwhede. 2012. LiBrA-CAN: A Lightweight Broadcast Authentication Protocol for Controller Area Networks. In *Cryptology and Network Security*, Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 185–200.

[19] Dongsoo Ha, Wenhui Jin, and Heekuck Oh. 2018. REPICA: Rewriting Position Independent Code of ARM. *IEEE Access* 6 (2018), 50488–50509. https://doi.org/10.1109/ACCESS.2018.2868411

[20] Subir Halder, Amrita Ghosal, and Mauro Conti. 2020. Secure over-the-air software updates in connected vehicles: A survey. *Computer Networks* 178 (2020), 107343. https://doi.org/10.1016/j.comnet.2020.107343

[21] A. Hazem and H. Fahmy. 2012. LCAP-A Lightweight CAN Authentication Protocol for Securing In-Vehicle Networks.

[22] A. V. Herrewege, Dave Singelée, and I. Verbauwhede. 2011. CANAuth - A Simple, Backward Compatible Broadcast Authentication Protocol for CAN bus.

[23] Irina Hossain and Syed Masud Mahmud. 2007. Analysis of a Secure Software Upload Technique in Advanced Vehicles using Wireless Links. In *2007 IEEE Intelligent Transportation Systems Conference*. 1010–1015. https://doi.org/10.1109/ITSC.2007.4357797

[24] Galen Hunt and Doug Brubacher. 1999. Detours: Binary Interception of Win32 Functions. In *Proceedings of the 3rd Conference on USENIX Windows NT Symposium - Volume 3* (Seattle, Washington) *(WINSYM'99)*. USENIX Association, USA, 14.

[25] Infineon. 2013. *SAK-TC1767-256F80HR AD*. https://www.infineon.com/cms/en/product/microcontroller/legacy-microcontroller/other-legacy-mcus/audo-family/tc1767-audo-future/sak-tc1767-256f80hr-ad/

[26] SAE International. 2021. *SAE J1939 Standards Collection*. https://www.sae.org/publications/collections/content/j1939_dl/

[27] SAE International. 2021. SAE J1939 Standards Collection. https://www.sae.org/standardsdev/groundvehicle/j1939a.htm

[28] SAE International. 2022. *VW Transport Protocol 2.0 (TP 2.0) for CAN bus*. https://jazdw.net/tp20

[29] ISO 11898:2015 2015. *Road vehicles — Controller area network (CAN)*. Standard. International Organization for Standardization, Geneva, CH.

[30] Hyo Jin Jo, Jin Hyun Kim, Hyon-Young Choi, Wonsuk Choi, Dong Hoon Lee, and Insup Lee. 2020. MAuth-CAN: Masquerade-Attack-Proof Authentication for In-Vehicle Networks. *IEEE Transactions on Vehicular Technology* 69, 2 (2020), 2204–2218. https://doi.org/10.1109/TVT.2019.2961765

[31] Taegyu Kim, Chung Hwan Kim, Hongjun Choi, Yonghwi Kwon, Brendan Saltaformaggio, Xiangyu Zhang, and Dongyan Xu. 2017. RevARM: A Platform-Agnostic ARM Binary Rewriter for Security Applications. In *Proceedings of the 33rd Annual Computer Security Applications Conference* (Orlando, FL, USA) *(ACSAC 2017)*. Association for Computing Machinery, New York, NY, USA, 412–424. https://doi.org/10.1145/3134600.3134627

[32] Taegyu Kim, Vireshwar Kumar, Junghwan Rhee, Jizhou Chen, Kyungtae Kim, Chung Hwan Kim, Dongyan Xu, and Dave (Jing) Tian. 2021. PASAN: Detecting Peripheral Access Concurrency Bugs within Bare-Metal Embedded Applications. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 249–266. https://www.usenix.org/conference/usenixsecurity21/presentation/kim

[33] Marcel Kneib, Oleg Schell, and Christopher Huth. 2020. EASI: Edge-Based Sender Identification on Resource-Constrained Platforms for Automotive Networks. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/easi-edge-based-sender-identification-on-resource-constrained-platforms-for-automotive-networks/

[34] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. 2010. Experimental Security Analysis of a Modern Automobile. In *2010 IEEE Symposium on Security and Privacy*. 447–462. https://doi.org/10.1109/SP.2010.34

[35] Sekar Kulandaivel, Tushar Goyal, Arnav Kumar Agrawal, and Vyas Sekar. 2019. CANvas: Fast and Inexpensive Automotive Network Mapping. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 389–405. https://www.usenix.org/conference/usenixsecurity19/presentation/kulandaivel

[36] Ryo Kurachi, Yutaka Matsubara, Hiroaki Takada, Naoki Adachi, Yukihiro Miyashita, and Satoshi Horihata. 2014. CaCAN - Centralized Authentication System in CAN.

[37] S.M. Mahmud, S. Shanker, and I. Hossain. 2005. Secure software upload in an intelligent vehicle via wireless communication links. In *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.* 588–593. https://doi.org/10.1109/IVS.2005.1505167

[38] Karim Mansour, Wael Farag, and Mohamed ElHelw. 2012. AiroDiag: A sophisticated tool that diagnoses and updates vehicles software over air. In *2012 IEEE International Electric Vehicle Conference*. 1–7. https://doi.org/10.1109/IEVC.2012.6183181

[39] Pal-Stefan Murvay and Bogdan Groza. 2014. Source Identification Using Signal Characteristics in Controller Area Networks. *IEEE Signal Processing Letters* 21, 4 (2014), 395–399. https://doi.org/10.1109/LSP.2014.2304139

[40] D. K. Nilsson and U. E. Larson. 2008. Secure Firmware Updates over the Air in Intelligent Vehicles. In *ICC Workshops - 2008 IEEE International Conference on Communications Workshops*. 380–384. https://doi.org/10.1109/ICCW.2008.78

[41] Dennis K. Nilsson, Ulf E. Larson, and Erland Jonsson. 2008. Efficient In-Vehicle Delayed Data Authentication Based on Compound Message Authentication Codes. In *2008 IEEE 68th Vehicular Technology Conference*. 1–5. https://doi.org/10.1109/VETECF.2008.259

[42] Dennis K. Nilsson, Lei Sun, and Tatsuo Nakajima. 2008. A Framework for Self-Verification of Firmware Updates over the Air in Vehicle ECUs. In *2008 IEEE Globecom Workshops*. 1–5. https://doi.org/10.1109/GLOCOMW.2008.ECP.56

[43] Stefan Nürnberger and Christian Rossow. 2016. – vatiCAN – Vetted, Authenticated CAN Bus. In *Cryptographic Hardware and Embedded Systems – CHES 2016*, Benedikt Gierlichs and Axel Y. Poschmann (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 106–124.

[44] Pádraig O'Sullivan, Kapil Anand, Aparna Kotha, Matthew Smithson, Rajeev Barua, and Angelos D. Keromytis. 2011. Retrofitting Security in COTS Software with Binary Rewriting. In *Future Challenges in Security and Privacy for Academia and Industry*, Jan Camenisch, Simone Fischer-Hübner, Yuko Murayama, Armand Portmann, and Carlos Rieder (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 154–172.

[45] Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. 2021. *Blake3, one function, fast everywhere.* https://raw.githubusercontent.com/BLAKE3-team/BLAKE3-specs/master/blake3.pdf

[46] Mert D. Pesé, Jay W. Schauer, Junhui Li, and Kang G. Shin. 2021. *S2-CAN: Sufficiently Secure Controller Area Network*. Association for Computing Machinery, New York, NY, USA, 425–438. https://doi.org/10.1145/3485832.3485883

[47] Andreea-Ina Radu and Flavio D. Garcia. 2016. LeiA: A Lightweight Authentication Protocol for CAN. In *Computer Security – ESORICS 2016*, Ioannis Askoxylakis, Sotiris Ioannidis, Sokratis Katsikas, and Catherine Meadows (Eds.). Springer

International Publishing, Cham, 283–300.

[48] Ted Romer, Geoff Voelker, Dennis Lee, Alec Wolman, Wayne Wong, Hank Levy, Brian Bershad, and Brad Chen. 1997. Instrumentation and Optimization of Win32/Intel Executables Using Etch. In *Proceedings of the USENIX Windows NT Workshop on The USENIX Windows NT Workshop 1997* (Seattle, Washington) *(NT'97)*. USENIX Association, USA, 1.

[49] rusEFI. 2021. *A GPL open source Engine Management System.* https://rusefi.com/

[50] Khaled Serag, Rohit Bhatia, Vireshwar Kumar, Z. Berkay Celik, and Dongyan Xu. 2021. Exposing New Vulnerabilities of Error Handling Mechanism in CAN. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity21/presentation/serag

[51] Shellphish. 2017. *Patcherex, Shellphish's automated patching engine, originally created for the Cyber Grand Challenge.* https://github.com/angr/patcherex/

[52] STMicroelectronics. 2021. *ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32.* https://www.st.com/en/development-tools/st-link-v2.html

[53] Telecommunication Standardization Sector Of International Telecommunication Union. 2020. *Guidelines for an intrusion detection system for in-vehicle networks.* https://www.sae.org/publications/collections/content/j1939_dl/

[54] United States Congress. 2020. Clean Air Act Section 203(a)(3)(B); 42 U.S. Code § 7522.

[55] United States Environmental Protection Agency. 2020. EPA Tampering Policy: The EPA Enforcement Policy on Vehicle and Engine Tampering and Aftermarket Defeat Devices under the Clean Air Act. https://www.epa.gov/sites/default/files/2020-12/documents/epatamperingpolicy-enforcementpolicyonvehicleandenginetampering.pdf

[56] Jo Van Bulck, Jan Tobias Mühlberg, and Frank Piessens. 2017. VulCAN: Efficient Component Authentication and Software Isolation for Automotive Control Networks. In *Proceedings of the 33rd Annual Computer Security Applications Conference* (Orlando, FL, USA) *(ACSAC 2017)*. Association for Computing Machinery, New York, NY, USA, 225–237. https://doi.org/10.1145/3134600.3134623

[57] Eric Wang, William Xu, Suhas Sastry, Songsong Liu, and Kai Zeng. 2017. Hardware Module-Based Message Authentication in Intra-vehicle Networks. In *2017 ACM/IEEE 8th International Conference on Cyber-Physical Systems (ICCPS)*. 207–216.

[58] Qiyan Wang and Sanjay Sawhney. 2014. VeCure: A practical security framework to protect the CAN bus of vehicles. In *2014 International Conference on the Internet of Things (IOT)*. 13–18. https://doi.org/10.1109/IOT.2014.7030108

[59] Shuai Wang, Pei Wang, and Dinghao Wu. 2016. UROBOROS: Instrumenting Stripped Binaries with Static Reassembling. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 236–247. https://doi.org/10.1109/SANER.2016.106

[60] Richard Wartell, Vishwath Mohan, Kevin W. Hamlen, and Zhiqiang Lin. 2012. Securing Untrusted Code via Compiler-Agnostic Binary Rewriting. In *Proceedings of the 28th Annual Computer Security Applications Conference* (Orlando, Florida, USA) *(ACSAC '12)*. Association for Computing Machinery, New York, NY, USA, 299–308. https://doi.org/10.1145/2420950.2420995

[61] Haohuang Wen, Qi Alfred Chen, and Zhiqiang Lin. 2020. 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)*. USENIX Association, 949–965. https://www.usenix.org/conference/usenixsecurity20/presentation/wen

[62] Matthias Wenzl, Georg Merzdovnik, Johanna Ullrich, and Edgar Weippl. 2019. From Hack to Elaborate Technique—A Survey on Binary Rewriting. *ACM Comput. Surv.* 52, 3, Article 49 (jun 2019), 37 pages. https://doi.org/10.1145/3316415

[63] David Williams-King, Hidenori Kobayashi, Kent Williams-King, Graham Patterson, Frank Spano, Yu Jian Wu, Junfeng Yang, and Vasileios P. Kemerlis. 2020. Egalito: Layout-Agnostic Binary Recompilation. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 133–147. https://doi.org/10.1145/3373376.3378470

[64] Guoming Zhang, Chen Yan, Xiaoyu Ji, Tianchen Zhang, Taimin Zhang, and Wenyuan Xu. 2017. DolphinAttack: Inaudible Voice Commands. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 103–117. https://doi.org/10.1145/3133956.3134052

[65] Knut Ørland. 2015. *Styreenhet.* https://github.com/ION-Racing/Styreenhet