CANSentry: Securing CAN-Based Cyber-Physical Systems against Denial and Spoofing Attacks

Abdulmalik Humayed[1,3], Fengjun Li[1], Jingqiang Lin[2,4], and Bo Luo[1]

[1] The University of Kansas, Lawrence, KS, USA
[2] School of Cyber Security, Univ. of Science and Technology of China, Hefei, China
[3] Jazan University, Jazan, Saudi Arabia
[4] Institute of Information Engineering, Chinese Academy of Sciences, China
ahumayed@jazanu.edu.sa;{fli,bluo}@ku.edu;linjingqiang@iie.ac.cn

Abstract. The Controller Area Network (CAN) has been widely adopted as the *de facto* standard to support the communication between the ECUs and other computing components in automotive and industrial control systems. In its initial design, CAN only provided very limited security features, which is seriously behind today's standards for secure communication. The newly proposed security add-ons are still insufficient to defend against the majority of known breaches in the literature. In this paper, we first present a new stealthy denial of service (DoS) attack against targeted ECUs on CAN. The attack is hardly detectable since the actions are perfectly legitimate to the bus. To defend against this new DoS attack and other denial and spoofing attacks in the literature, we propose a CAN firewall, namely CANSentry, that prevents malicious nodes' misbehaviors such as injecting unauthorized commands or disabling targeted services. We implement CANSentry on a cost-effective and open-source device, to be deployed between any potentially malicious CAN node and the bus, without needing to modify CAN or existing ECUs. We evaluate CANSentry on a testing platform built with parts from a modern car. The results show that CANSentry successfully prevents attacks that have shown to lead to safety-critical implications.

1 Introduction

The *Controller Area Network* (*CAN*), also referred to as the *CAN bus*, has been widely adopted as the communication backbone of small and large vehicles, ships, planes, and industrial control systems. When CAN was originally developed, its nodes were not technically ready to be connected to the external world and thus assumed to be isolated and trusted. As a result, CAN was designed without basic security features such as encryption, authentication that are now considered essential to communication networks [14, 19]. The protocol's broadcast nature also increases the likelihood of attacks exploiting these security vulnerabilities. Therefore, a malicious message injected by a compromised electronic control unit (ECU) on the bus, if it conforms to CAN specifications, will be treated the same as a legitimate message from a benign ECU and broadcasted over the bus. Moreover, wireless communication capabilities have been added to CAN nodes for expanded functionality (e.g., TPMS, navigation, entertainment systems) without carefully examining the potential security impacts [46, 43, 21, 20]. This enlarges the attack vector for remote attacks by allowing some previously physically isolated units to connect to external entities through wireless connections. Various attacks have been reported in recent years, ranging from simple bus denial attacks [12, 36] and spoofing attacks [27, 32] to more sophisticated bus-off [42, 23] and arbitration denial attacks [27, 3, 12, 36]. In response to the vulnerabilities and attacks, significant research efforts have been devoted to automobile and CAN security. Solutions such as message authentication and intrusion detection, akin to those designed for Internet security, have been proposed to secure CAN [49, 50]. However, these proposals suffer from major efficiency issues. For instance, the authentication-based schemes deploy cryptographic keys among ECUs to generate MACs, which inevitably incur non-negligible processing overhead and significantly impact CAN's transmission speed. Moreover, such defense mechanisms can be defeated if the adversary exploits compromised ECUs to nullify authentication or MAC frames [3]. The intrusion detection-based mechanisms leverage ECUs' behavioral features to detect abnormal frames on the bus, hence, they have to collect a sufficient number of frames by monitoring the bus and ECUs, which inevitably results in delays in the detection.

In this paper, we first present a new stealthy arbitration denial attack against CAN, which takes seemingly legitimate actions to prevent selected CAN nodes from sending messages to the bus. This attack bypasses CAN controllers to inject deliberately crafted messages without triggering packet-based detectors, which allows the adversary to extend the length of the attack and cause severe damages. For instance, the attack can block all steering and breaking messages from being sent to the bus without causing any bus error, while keeping other sub-systems such as the engine operating normally. We implement the attack using an STM32 Nucleo-144 board as the attacker and an instrument panel cluster from a 2014 passenger car as the victim and demonstrate a successful attack.

Moreover, we propose a new defense mechanism, namely CANSentry, against the family of general CAN denial and spoofing attacks. This system-level control mechanism addresses the fundamental CAN security problem caused by the protocol's broadcast nature and lack of authentication without demanding any modification to the CAN standard or the ECUs. CANSentry leverages the malicious behavior of the attacks, e.g., inconsistency between attack ECU state and the bus state, to filter frames from high-risk ECUs (i.e., a few *external-facing* ECUs) in a bit-by-bit fashion and block unauthorized frames from being sent to the bus. We implement a cost-efficient prototype of CANSentry using the STM32 Nucleo-144 development board and demonstrate its effectiveness against recent denial and spoofing attacks including the one we propose in this paper.

The contributions of this paper are three-fold: (1) we demonstrate a stealthy selective arbitration denial attack against CAN that prevents selected ECUs from transmitting to the CAN bus without triggering any error or anomaly. (2) More importantly, we design a proof-of-concept CAN firewall, CANSentry, which is deployed between high-risk CAN nodes and the bus to defend against bus/ECU denial attacks and ECU spoofing attacks. And (3) we implement the proposed CAN firewall using low-cost hardware and demonstrate its effectiveness on a platform consisting of components from modern passenger cars.

The rest of the paper is organized as follows: we introduce the CAN bus and CAN attacks in Section 2 and the threat model in Section 3. Then, we present the stealthy selective arbitration denial attack in Section 4, and CANSentry design and implementation in Section 5, followed by security analysis in Section 6. Finally, we conclude the paper in Section 7.

2 Background and Related Work

2.1 The Control Area Network (CAN)

CAN has been widely used in many distributed real-time control systems. While components differ, the main concepts/mechanisms remain similar in all CAN applications. A CAN network consists of nodes interconnected by a differential bus. Each node is controlled by a microcontroller (MCU), a.k.a. electronic control unit (ECU) in automotive CAN networks. A node connects to the bus through a controller and a transceiver. The *controller* is a stand-alone circuit or an MCU module, which implements the protocol, e.g., encoding/decoding frames and error handling. The *transceiver* converts between logic data and physical bits.

Frame Prioritization. In CAN, frame prioritization is realized by the *arbitration ID* (a.k.a. *CAN ID*). Fig. 1 (a) shows a simplified CAN frame. It starts with the 11-bit arbitration ID (or 29-bit in the extended format), which determines the frame's priority on the bus as well as the frame's relevance to receivers, based on which each receiver decides to accept or ignore the frame. When multiple nodes attempt to send to the bus simultaneously, the lowest ID indicates the highest priority and wins the *arbitration*. To support prioritization of frames, the CAN specification defines *dominant* and *recessive* bits, denoted by "0" and "1", respectively. Whenever a dominant and a recessive bit are sent at the same time by different nodes, the dominant bit will *dominate* the bus. This mechanism allows CAN to resolve real-time conflicts during arbitration and transmission.

CAN Error Handling. Error handling in CAN allows nodes to autonomously detect and resolve transmission errors without third-party intervention. It also supports fault confinement and the containment of defective nodes. There are five types of errors in CAN: bit, ACK, stuff, form, and CRC errors. Each ECU is responsible for detecting and keeping track of both transmission and receiving errors with two error counters, *transmit error counter* (TEC) and *receive error counter* (REC). Depending on the role of the ECU, one of the counters will increase when an error is detected or decrease after a successful transmission or reception. The ECU determines its error state according to the values of both counters, as shown in Fig. 1 (b). The transitions between error states allow nodes to treat temporary errors and permanent failures differently. In particular, when a node encounters persistent transmission errors that may affect other nodes, it switches to bus-off state, resulting in its complete isolation from the CAN bus.

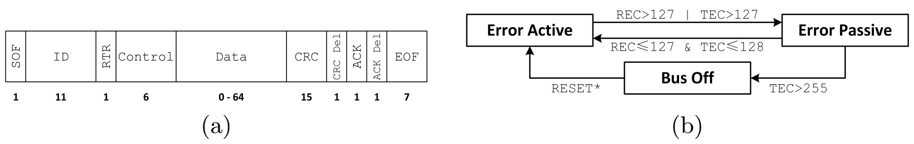


Fig. 1: The CAN Bus: (a) Format of a CAN data frame; (b) CAN error state transition. *: RESET or reception of 128 occurrences of 11 recessive bits.

2.2 Existing CAN Attacks

In this paper, we focus on *denial* and *spoofing* attacks against CAN. Denial attacks are further classified into bus, ECU, and arbitration denial.

Bus Denial. A naive approach is to flood the bus with a stream of 0x0 IDs so the bus is always occupied with this highest priority ID (*BD1*) [27, 12]. Another approach is to occupy the bus by transmitting a stream of dominant bits and prevent any ECU from transmitting. However, this bit-stream does not conform to CAN protocol, hence, it cannot be dispatched by CAN controllers in regular mode. [12] exploited the *Test Mode* in some CAN controllers to flood the bus with dominant bits (*BD2*). [36] built a malicious ECU without the CAN controller to allow them to launch the dominant bit-stream attack (*BD3*).

ECU Denial. The attacker attempts to force an ECU to a bus-off state (`TEC>255`) and eliminates all the functionalities managed by the target ECU. Four types of ECU denial attacks have been proposed in the literature: (1) CAN Controller Abuse (*ED1*). [12] exploited the *ID Ready Interrupt* feature and the *Test Mode* on some ECUs to inject dominant bits when the target ID attempts to transmit. This would trigger transmission bit errors at the target ECU, as it detects the discrepancies between what it sends and what it sees on the bus. Repeating this attack would gradually increase the transmitter's TEC, and eventually force it to bus-off. (2) Malicious Frames (*ED2*). An adversarial ECU may send a frame with identical contents to the targeted ECU's, except replacing a recessive bit with a dominant one, to trigger a bit error at the transmitting ECU [3, 12]. (3) Bypassed CAN Controller (*ED3*). Adversaries directly connect malicious/compromised ECUs to CAN transceivers to inject arbitrary bit streams to the bus. [42, 36, 23] implemented this attack to overwrite recessive bits from the target ECU by dominant ones to trigger bit errors at the transmitting ECU.

Arbitration Denial. The adversarial ECU attempts to inject frames with the lowest possible ID (0x0) to win the arbitration over any other CAN ID and prevent all non-0x0 IDs from sending to the bus, so that the bus becomes completely non-functional [12]. It could also target on a selected ID by transmitting an ID of higher priority whenever that ID starts transmission [22]. We categorize both cases as *AD1*. [36] monitored the bus with an MCU connected through a CAN transceiver (without CAN controller) to detect the target ID and then inject a dominant bit that replaces a recessive bit in the ID field (*AD2*). The target ID loses arbitration and is unaware of the attack, however, the attack results in an incomplete frame that causes a form error. In Section 4, we propose a stealthier version so that no error frame is generated (*AD3*).

Control	**Features**				**Effectiveness against attacks**									
	Inj.	Aper.	RT	Cost	BD1	BD2	BD3	ED1	ED2	ED3	AD1	AD2	AD3	Spoof
C1	✗	✓	✗	✓	*D*	*D*	*D*	*D*	*D*	*D*	*D*	*D*	*D*	*D*
C2	✗	✓	✗	✗	*D*	-	-	-	*D*	-	*D*	-	-	*D*
C3	✗	✗	✗	✓	*D*	-	-	-	*D*	-	*D*	-	-	*D*
C4	✗	✓	✗	✓	-	-	-	*P*	*P*	*P*	*P*	*P*	*P*	*P*
C5	✗	✓	✓	✓	*P*	-	-	-	*P*	-	*P*	-	-	*P*
C6	✗	✓	✗	✗	*P*	-	-	-	*D*	-	*D*	-	-	*P*
C7	✗	✓	✗	✗	*P*	-	-	-	*D*	-	*P*	-	-	*P*
CANSentry	✓	✓	✓	✓	*P*	*P*	*P*	*P*	*P*	*P*	*P*	*P*	*P*	*P*

Features: **Inj.**: preventing injection of incomplete frames or random bits, **Aper.**: handling aperiodic attacks, **RT**: real-time defense; **Cost**: low cost.
Effectiveness: **D**: Detect, **P**: Prevent, **-**: No protection
Controls: C1: Anomaly-based IDS [19, 13, 38, 52, 51, 37]; **C2**: Voltage-based IDS [35, 6, 7, 5, 4, 10]; **C3**: Time-based IDS [15, 4]; **C4**: CAN-ID Obfuscation [22, 59, 17, 24, 29, 58]; **C5**: Counterattacking [8, 30, 28, 48]; **C6**: Authentication [41, 44, 18, 53, 54]; **C7**: Application-level Firewalls [45, 26].

Table 1: Summary of CAN security control mechanisms: features and effectiveness against attacks discussed in Section 2.2.

Spoofing Attacks. Due to the lack of authentication in CAN and the broadcast nature of the bus, a compromised ECU could easily send CAN frames with any ID, including IDs that belong to other legitimate/critical ECUs. In [32, 27], attackers compromised an ECU through a remote channel, and sent CAN frames to unlock doors, stall the engine, or control the steering wheel. In the *masquerade attack*, the attacker first disables the target ECU and then transmits its frames [40]. In the *conquest attack*, the target ECU is fully compromised to transmit legitimate frames but with malicious intentions [40]. Lastly, [31] sent spoofed diagnostic frames to force a target ECU into diagnostic session, which would stop transmitting frames until the diagnostic session is terminated.

2.3 Existing Controls and Limitations

IDS for CAN. Due to the high predictability of CAN traffic, anomaly detection becomes a viable solution. For example, [19, 13, 38, 52, 51, 37] monitor CAN traffic and detect frame injection attacks based on the analysis of traffic pattern and packet contents. Meanwhile, due to the lack of sender authentication in CAN, a number of fingerprinting approaches are proposed to identify senders based on physical layer properties. For example, ECUs could be profiled and identified with time/clock features [15, 4] or electric/voltage features [35, 5, 7, 6, 25, 10]. These approaches associate frames with their legitimate senders. When a mismatch is detected, an attack is assumed. They are effective against the spoofing attacks and some of the denial attacks, as shown in Table 1. The IDS mechanisms do not provide real-time detection except [10], which detects a malicious frame before it completes transmission. In addition, they all assume that packets conform to CAN protocol. Therefore, they cannot recognize many random bit or incomplete frame injection attacks discussed in Section 2.2.

CAN-ID Obfuscation. Many attacks discussed in Section 2.2 target specific CAN IDs. Several solutions have been proposed to obfuscate CAN IDs to confuse or deter the attackers, e.g., ID-Hopping [22, 59], ID Randomization [17, 24], ID Obfuscation [29], and ID Shuffling [58]. The effectiveness of these solutions rely on the secrecy of the obfuscation mechanism, i.e., the assumption that the obfuscated CAN IDs are known to the legitimate ECUs but not the attackers. This assumption may not be practical in real world applications.

Counterattacking. The counterattack mechanisms attempt to stop CAN intrusions by attacking the source. In [30], owners of CAN IDs (legitimate ECUs) would detect spoofed frames on the bus, and interrupt the sender by sending an error frame. [8] proposed a similar mechanism that induced collisions with the spoofed frames until the attacking ECU was forced into bus-off. [28] proposed a counterattack technique with a central monitoring node , which shared secret keys with legitimate ECUs for authentication and spoof detection. [48] launched a bus-off attack against the adversarial ECU of the attacks proposed in [3]. The counterattack approaches need to add detection and counterattacking capacity to all ECUs, so that they can protect their own CAN IDs. This is not cost-effective, and may be impractical for some mission-critical ECUs.

Authentication. Cryptography-based solutions have been proposed to enable node (ECU) authentication, and to support data confidentiality and integrity [54, 55, 34, 60]. In addition, controls against attacks from external devices have been introduced [9, 16, 47, 57]. They require encryption, key management, and collaboration between ECUs, which implies significant changes to the existing CAN infrastructure. Additional communication overhead and delays are also expected, which is undesired in real-time environments such as CAN.

Firewalls. The idea of an in-vehicle firewall was first suggested in [56] to enforce authentication and authorization of CAN nodes, but it did not articulate any technical details behind the concept. Application-level firewalls have been introduced in [45, 26], which require intensive modification at the CAN node (in both software and hardware) and system-wide cryptographic capabilities to CAN. This is impractical in real world CAN systems. Finally, the attacks performed by the skipper model [36, 42] cannot be prevented using such solutions.

In Table 1, we compare different CAN defense approaches with our CANSentry solution in terms of features and capabilities of detecting or preventing the attacks discussed in Section 2.2. We also like to note that existing countermeasures do not consider cases where the attacker abuses or bypasses the CAN controller to send arbitrary bit streams or incomplete frames, e.g. [12, 36, 23]. Meanwhile, controls that transmit frames to the bus, e.g., counterattacks, are also vulnerable to denial attacks that would nullify their effectiveness. Lastly, many existing solutions require overhead such as renovating the protocol, upgrading all or many ECUs, employing statistical learning for IDS, etc., which makes them less cost-efficient and impractical.

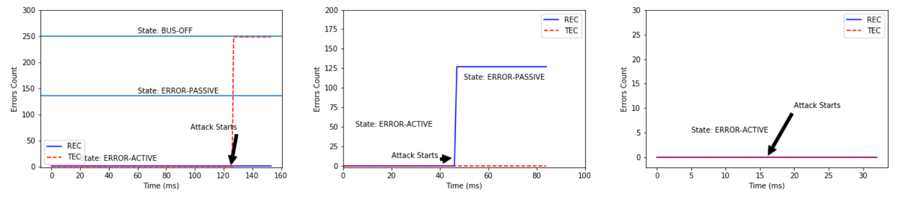
3 The Threat Model

Attackers' Objectives. In this paper, we study two attacks, *denial* and *spoofing.* In denial attacks, the attacker aims to nullify certain functionality of an ECU or the CAN bus, such as forcing an ECU into bus-off, stopping a CAN ID from sending to the bus, or occupying the bus. A *stealthy* denial is to achieve the goal without being noticed by the target ECU or the bus. In spoofing, the attacker aims to spoof a selected ID to send frames to the bus and deceive the receiver, which can trigger undesired consequences ranging from displaying incorrect information to disabling the brakes or the steering wheel [27, 32, 39].
Attacker's Capabilities and Limitations. Attackers with local or remote access have been demonstrated in the literature [27, 2, 11, 32]. We consider two attacker models with different capabilities, based on which, different attacks could be realized that vary in impact and sophistication. In both cases, we assume that the attacker knows the design and specifications of the targeted system (automobile) model through open documentation or reverse engineering. For example, the attacker knows the IDs and functionalities of the ECUs.
1. CAN Abusers. The attacker has a local or remote access to an ECU. The attacker obtains full control of all software components of the ECU but cannot alter the hardware. The attacker is assumed to be able to sniff the bus and inject frames to abuse one or more of the CAN's basic functionalities: arbitration and error handling. Note that the integrity of the CAN controller is preserved, so that all the injected frames conform to the CAN protocol. When an attacker transmits a frame with a high priority ID, the frame wins arbitration over legitimate low priority IDs. On the other hand, the attacker can transmit an almost identical frame with a small difference to trigger error handling and eventually force a target ECU to bus-off state. The attacker may also abuse a feature provided by some CAN controllers, e.g., the "Test Mode", to inject dominant bits [12].
2. The Skippers. An attacker can sniff and inject arbitrary CAN traffic by skipping the CAN controller. This allows her to inject bits at any time without having to be limited by the rules enforced by CAN controllers. This could be achieved through direct or remote access: (1) Direct Access: the attacker could connect a malicious MCU and a CAN transceiver to the OBD-II port (e.g., [42, 36]), or connect the MCU through an aftermarket OBD-II adapter (e.g. [11]). This approach requires brief physical access to the vehicle. (2) Remote Access: the attacker may first compromise an ECU (usually one with remote access capabilities such as WiFi or cellular), and then exploit certain hardware design vulnerabilities to directly connect to the bus (e.g. [32]). In particular, [12] showed that 78% of the MCUs deployed in vehicles have built-in CAN controllers, which connect to CAN transceivers through GPIO pins. A compromised MCU could change GPIO configurations to disconnect the CAN controller, and then directly connect itself to the transceiver. In both cases, the attacker "skips" the CAN controller, analyzes the traffic in a bit-by-bit fashion in a process known as "Bit Banging", and injects arbitrary bit-streams to the bus.

We assume that the attacker does not have unlimited and uncontrolled access to alter or to break the integrity of CAN hardware. The attacker could access the bus through any legitimate access point, or through existing hardware/software vulnerabilities, but she cannot alter the hardware to create new vulnerabilities or access points. That is, the attacker may have brief access to the car to connect a malicious ECU to the OBD-II port. However, she cannot disassemble the system to weld a new port or a new attacking device directly to the bus.

4 The Stealthy Selective Arbitration Denial Attack

In this section, we improve the design of the arbitration denial attack presented in [36] with two added features: stealthiness and inexpensive hardware implementation. We present the implementation of the new attack and evaluate it with an Instrument Panel Cluster (IPC) from a used passenger car.
Attack Objectives. The objective of the attack is to stealthily prevent frames with specific IDs from being sent to the bus. The attack is expected to have the following features: (1) Selective: only specific IDs are denied, while other ECUs/frames all function as expected. (2) Stealthy: the attack should conform to CAN standard and should not trigger any error on the bus. Hence, the attacker controls the damage and extends the length of the attack. And (3) Practical: the attack should not require expensive hardware or extended access to the bus.
Adversary Model. The attacker needs to bypass CAN controllers' restrictions to perform bit-by-bit analysis/manipulation on the bus. Therefore, the attacker connects directly to the CAN transceiver, i.e., the *Skipper* model in Section 3.
Challenges. First, most of existing CAN tools (e.g., CANoe, VehicleSpy, SocketCAN only work with full CAN frames. However, we need to monitor the bus at bit-level and inject bits at any arbitrary time. Next, the attack requires a high degree of precision to the bit-level timing. Last, since the skipper model operates without a CAN controller, any unexpected operation delay, premature injection, or malformed CAN frame will result in bus errors, which may render the attack unsuccessful or detectable.
The Attack. The proposed attack passively monitors the bus to detect a specific CAN ID in the arbitration phase. The attacker waits for the last recessive bit in the target ID, and overwrites it with a dominant one, to beat the targeted ECU in arbitration. The attacker completes the transmission with a fake frame, so that it would not trigger any error flag (as in [36]). Hence, the malfunctioning on the bus cannot be detected, and the attacking ECU cannot be identified.
Attack Implementation and Evaluation. Most of the current CAN research that require precise timing use automotive-grade micro-controllers or other expensive tools. We use an open-source tool, CANT [1], which facilitates the synchronization with the bus and bit-level analysis/manipulation on the bus. We connect the following to the CAN bus: (1) attacker: an STM32 Nucleo-144 board (connected through OBD-II) running CANT; (2) victim: the IPC of a used 2014 passenger car; (3) other nodes: simulated by BeagleBone Black (BBB) micro-controllers. In the experiments, when we launch the proposed attack against ID 0x9A (turn signals), the turn signals become unresponsive regardless of the status of the turn signal lever, since their control messages are blocked.



(a) (b) (c)

Fig. 2: Comparison of the stealthiness of CAN denial attacks: (a) ECU denial; (b) arbitration denial; (c) stealthy selective arbitration denial.

We compare the stealthiness of the proposed attack with the other denial attacks discussed in Sec. 2.2: ECU denial [12, 42] and arbitration denial[12, 36]. We assign a BBB microcontroller as the victim ECU and capture its errors (TEC, REC and error state). The ECU denial attack (Fig. 2 (a)) causes a sharp increase of TEC at the victim ECU, to force it into bus-off state. It also increases the REC counters on other ECUs. The arbitration denial attack (Fig. 2 (b)) interrupts the victim ECU and causes a form error, which also drives all ECUs into error passive mode, since incomplete frames are detected on the bus. Finally, our attack (Fig. 2 (c)) is stealthy as it achieves the arbitration denial without causing any error on any ECU.

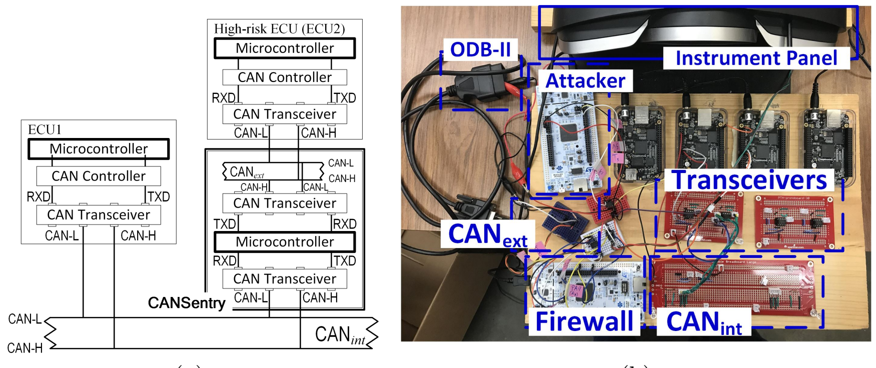
5 CANSentry: A Firewall for the CAN Bus

Next, we present CANSentry, an efficient and low-cost firewall for the CAN bus to defend against the attacks discussed in Sections 2.2 and 4.

5.1 The Architecture of the CANSentry Firewall

The objective of CANSentry is to prevent an attacker node (either a CAN abuser or a skipper) from sending malicious frames onto the CAN bus without introducing any practical delay. This requires monitoring and filtering all the messages in real time, since we cannot block any ECU from accessing the bus before an abnormal activity is detected. To address this challenge, we propose a segmentation-based approach to separate high-risk CAN nodes, i.e., a few ECUs with interfaces to the external network, from the rest of the bus, using a firewall.

As shown in Fig. 3(a), CANSentry is deployed between each high-risk node and the main bus, which logically divides the original CAN bus into two segments, the *internal* bus CAN_{int} and the *external* bus CAN_{ext}. A high-risk ECU directly connects to CAN_{ext}, which further connects to CAN_{int} through the CANSentry firewall. Both segments send and receive messages following the original CAN specifications, where the bi-directional firewall functions mainly as a relay to transmit legitimate messages between CAN_{int} and CAN_{ext} without causing any collision. Therefore, CANSentry monitors the current transmission state of the main bus and decides to forward or block the messages from CAN_{ext}.



(a) (b)

Fig. 3: The CANSentry Approach: (a) Firewall architecture; (b) Implementation.

Introducing a firewall component for network segmentation enforces the necessary security control, which is missing in the current CAN standard, to regulate the activities of the high-risk, potentially vulnerable nodes. This defense mechanism could fundamentally overcome the vulnerabilities caused by the broadcast nature and lack of authentication/authorization capacities of the CAN bus by preventing the attacker node from broadcasting unauthorized frames. Note that CANSentry does not demand any modifications to the CAN standard, or require any changes to the existing ECUs, which makes it easily adoptable.

5.2 Firewall Principle and Rules

The CANSentry firewall monitors the states of the internal and external buses. Similar to other network firewalls, it checks bi-directional traffic against a set of rules and enforces the forwarding or blocking actions. However, the design of the firewall rules is not straightforward. We have to leverage the features of CAN denial and spoofing attacks to derive appropriate firewall rules for prevention.

States and State Transitions of CAN Bus and Nodes. We examine the relationship between the states of a CAN node and the corresponding states of the CAN bus, as seen by the firewall. As shown in Fig. 4(a), each CAN node transits between four legitimate states. In particular, the RECEIVE state denotes the node is listening to the bus when another node is transmitting, while the IDLE state denotes the node is listening to the bus and waiting for other nodes to transmit. We combine them as the node takes the same action in both states. Moreover, the ARBITRATION state denotes the transmission of CAN ID bits and the TRANSMIT state denotes the transmission of all other data and control bits. Correspondingly, the CAN bus also transits between four legitimate states as shown in Fig. 4(b). For the internal bus, we further define TRANSMIT$_{int}$ and TRANSMIT$_{ext}$ states to distinguish the transmission due to an (directly connected) internal node or an (firewalled) external node, respectively.

From the aspect of bus state transitions, we can interpret the attacks discussed in Section 2.2 as exploiting malicious messages to compromise specific

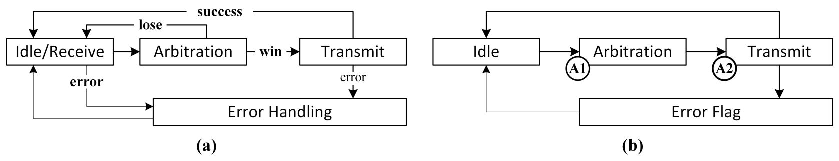


Fig. 4: State transitions of a CAN node and the CAN bus.

bus states. In particular, in the spoofing and arbitration denial attacks, the adversary node (e.g., a malicious abuser or skipper ECU) compromises the bus Arbitration state (denoted as “A1” in Fig. 4(b)) by injecting a partial or complete fake frame with a fraudulent CAN ID to falsely win the arbitration. In the ECU denial attacks, the attacker node compromises the bus Transmit state by injecting messages with deliberately crafted data bits to falsely trigger the target node into transmission errors. Finally, the bus denial attack is a special case of compromising either Arbitration or Transmit states with a stream of dominant bits to take over the entire bus.

Firewall Rules. The fundamental *principle* of the firewall is to ensure that at any time high-risk nodes on the external bus operate in a state consistent with the state of the internal bus. So, for each bus state, we derive the corresponding consistent node states based on the CAN protocol. In particular, (i) for the bus Idle state, the consistent node states are Idle/Receive, Arbitration and Transmit; (ii) for the bus Arbitration state, the consistent node state are Idle/Receive and Arbitration; and (iii) for the bus Transmit or ErrorFlag states, the consistent node state is only the Idle/Receive state. Finally, we derive a set of firewall rules following the above principle. Similar to network firewall enforcement, the rules will be executed in order such that the traffic blocked by an upper rule will not be evaluated by the lower rules.

$\mathbb{R}1$: When the internal bus is in either Transmit_{int} or ErrorFlag state, the firewall always *forwards* the traffic (bits or frames) from CAN_{int} to CAN_{ext} and *blocks* the traffic from external to internal, regardless of high-risk node's state.

$\mathbb{R}2$: When the internal bus is in either Idle or Arbitration state, the firewall *forwards* all the traffic from CAN_{int} to CAN_{ext}. It also *forwards* traffic that has a CAN ID in the *arbitration whitelist* and conforms to CAN specifications from CAN_{ext} to CAN_{int}. It *blocks* all other traffic from external to internal.

$\mathbb{R}3$: When the internal bus is in the Transmit_{ext}, the firewall *forwards* the traffic from CAN_{ext} to CAN_{int} that conforms CAN specifications, and *blocks* all traffic from CAN_{int} to CAN_{ext}, except for error flags.

Discussions. First, we assume that the traffic on the internal bus conforms to CAN specifications, because all the nodes behind the firewalls are considered low risk and more trustworthy than external nodes. Regulated by CAN controllers, their activities should not deviate from the CAN protocol. Meanwhile, in $\mathbb{R}1$, the firewall blocks all the traffic from CAN_{ext} to CAN_{int} so that it may block legitimate error flags generated by a benign external node. This may mistakenly block all error flags originated in CAN_{ext}, since we cannot distinguish if it

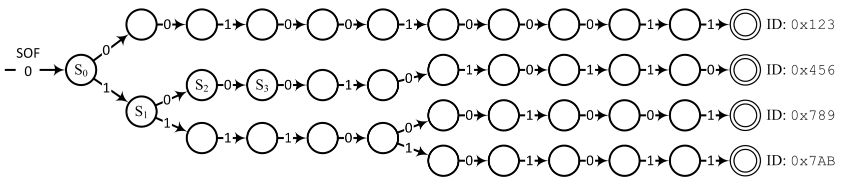


Fig. 5: An example DFA for CAN ID matching.

comes from a benign or compromised external ECU. However, the impact of this blocking to both external and internal nodes is negligible: it does not affect the error counters of the external node, meantime, the internal node causing the error can still receive enough reliable error flags from internal nodes. Lastly, when the internal bus is in TRANSMIT$_{ext}$ (i.e., an external node won the arbitration and is sending to the bus), no legitimate traffic should be sent by any other node on the internal bus. So, we block the CAN$_{int}$-to-CAN$_{ext}$ circuit in $\mathbb{R}3$ as a protective measure. It is worth noting that error flags are handled as a special case – they are collected by an error buffer in the firewall, and sent to CAN$_{ext}$ when identified. For example, when an internal node detects any error and sends out an error flag, it will trigger $\mathbb{R}2$ to block the CAN$_{ext}$-to-CAN$_{int}$ circuit and forward the error flag to the transmission ECU on CAN$_{ext}$. Error flags will arrive at CAN$_{ext}$ with a 5-bit delay, which would not cause any issue in error handling.

Rule Enforcement. Each firewall monitors both internal and external buses and detects the current internal bus state and external bus state (almost equivalent to the external node state since there is only one node on CAN$_{ext}$). Since $\mathbb{R}1$ and $\mathbb{R}3$ only involve two firewall actions, *forward* and *block*, the implementation is straightforward. However, $\mathbb{R}2$ requires the firewall to check the CAN ID originated from the external node against a pre-determined whitelist during the arbitration. The key challenge is to detect the malicious bit as soon as possible before a spoofed ID wins the arbitration. Therefore, we construct a *deterministic finite automaton* (DFA) to enforce this bit-by-bit ID matching.

Formally, a DFA is defined as a 5-tuple $(Q, q_0, \Sigma, \delta, F)$, where Q is a finite set of states, $q_0 \in Q$ denotes the initial state, Σ is a finite set of input symbols, known as the input alphabet, $\delta : Q \times \Sigma \rightarrow Q$ is a transition function, and $F \subseteq Q$ denotes the accept states. To start the arbitration, a node always issues a Start-of-Frame (SOF) for hard synchronization, so the input of q_0 is always 0. $\Sigma = \{0, 1\}$ since the input is a bit stream. Since each firewall is in charge of one high-risk node on the external bus, it maintains a set of CAN IDs to which the external ECU is allowed to send messages. Based on the ID set, we derive the transition function δ and the accepted states $F = \{\text{CAN IDs}\}$. For example, Fig. 5 shows the DFA of a firewall whose whitelist contains four CAN IDs: "0x123", "0x456", "0x789" and "0x7AB". For a spoofed ID "0x481", its fifth bit (i.e., "1") will be rejected by the state S_3 of the DFA, then the firewall will block all the remaining bits of this ID.

In practice, we cannot wait until an Arbitration ID reaches an accept state to disseminate it to the bus, since the arbitration phase is expected to be precisely synchronized bit-by-bit among all competing ECUs. To be precise, the proposed CAN firewall implements a Moore machine [33], which is a DFA that generates an output at each state. At each DFA state, an output bit will be disseminated to CAN_{int} instantly so that it competes with other nodes on the bus. Lastly, when a spoofed ID is rejected by the DFA, the prefix bits have already been sent to CAN_{int}. We will further discuss this and its impact to security in Section 6.

5.3 Implementation and Evaluation

To enable an efficient bit-by-bit monitoring and manipulation of bit streams transmitted over CAN, CANSentry is constructed with one MCU and two CAN transceivers, where one transceiver interfaces between the firewall and CAN_{int} and the other interfaces between the firewall and CAN_{ext}, as shown in Fig. 3(a). As the main component of the firewall, the MCU has a filter module, which implements firewall rules and the DFA for CAN ID matching, and then enforces the forwarding or blocking decisions for CAN traffic.

Hardware and costs. We use the open-source tool CANT [1] on an STM32 Nucleo-144 development board to implement the proof-of-concept CANSentry. The MCU board is connected to CAN_{int} and CAN_{ext} through two designated transceivers. The hardware cost is about \$20 to us, which could be significantly lowered in mass production. Meanwhile, we only need to deploy firewalls to external-facing ECUs in a vehicle, which is very limited in number.

Transmission delay. We evaluate the transmission delay introduced by CANSentry. Fig. 6 depicts the received bit streams on the CAN_{int} and CAN_{ext} buses when a frame is relayed by the firewall from internal to external (i.e., Fig. 6(a)) and vice versa (i.e., Fig. 6(b)). In Fig. 6(b), the firewall processing includes DFA-based ID matching. Obviously, in both cases, there is no noticeable delay that could cause any bit error or synchronization error.

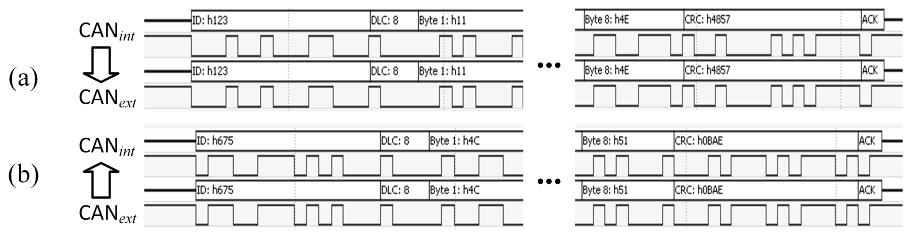


Fig. 6: Traffic between two buses: (a) CAN_{int} to CAN_{ext}; (b) CAN_{ext} to CAN_{int}.

Effectiveness against Attacks. As shown in Fig. 3(b), CANSentry is connected to two CAN transceivers interfacing CAN_{ext} and CAN_{int}, respectively. We use another STM32 Nucleo-144 board to simulate the attacks, which is connected to CAN_{ext} through ODB-II. We further select an ECU from the Instrument Panel Cluster (IPC) from a used 2014 passenger car as the victim ECU. The IPC is

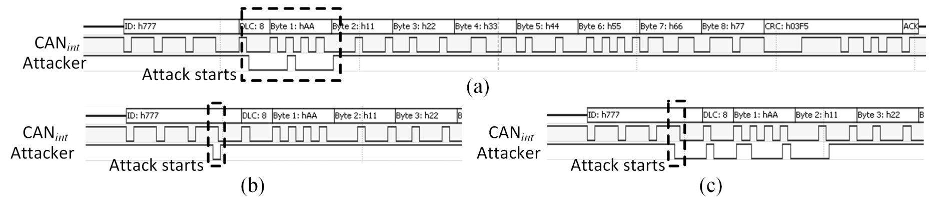


Fig. 7: Evaluation of firewall performance under: (a) bus-off attack; (b) arbitration denial attack; (c) the proposed stealthy selective arbitration denial attack.

connected to CAN$_{int}$. We simulate ten attacks discussed in Section 2.2 (Table 1) to evaluate the performance of CANSentry. In all the attacks, the attacker attempts to inject illegal bits that violate the CAN protocol (e.g., continuous dominant bits in bus-denial attack and arbitration-denial attack), a spoofed frame (e.g., spoofing attack and bus-denial attack), or a spoofed CAN ID (e.g., selective arbitration-denial attack). To evaluate the effectiveness of CANSentry, we monitor the bit stream on the `Tx` pin of the attacker ECU (i.e., the injected bits from the attacker), and the ones on the internal bus CAN$_{int}$, which shows the traffic on the main bus after the attacks, as shown in Fig. 7.

In the experiments, CANSentry was able to block all attack attempts from the adversary ECU, while not interfering with the normal operations on the internal bus CAN$_{int}$. Due to space limit, we only demonstrate the results of three attacks. Fig. 7(a) shows a bus-off attack, when a skipper-type attacker injects arbitrary bits (denoted in the block with dashed-line) to trigger a receive bit error at the victim ECU, this attempt is blocked by CANSentry (i.e., no change is observed on the internal bus). Meanwhile, when the attacker injects a dominant bit “0” to win the arbitration and block the transmission attempt of the victim ECU (Fig. 7(b)), or even covers his trace with a complete frame (from a spoofed CAN ID) and a valid CRC (Fig. 7(c)), the malicious actions are prevented by CANSentry and thus have no effect on the internal bus. Finally, it is worth pointing out that although we evaluate CANSentry in the in-vehicle network setting, it can be adopted to secure any CAN-based system.

6 Security Analysis and Discussions

Now we analyze the security guarantees of CANSentry and its effectiveness against threats introduced in Sec. 3, and discuss the remaining attack surfaces.

Security Analysis of Arbitration. A Deterministic Finite-state Automaton (DFA) is used to filter arbitration IDs (in binary strings).

Theorem 1. *Let $L(M)$ be the list of arbitration IDs on the whitelist, which is used to generate DFA M. When M is correctly generated, no CAN frames carrying an ID field that is not in $L(M)$ shall be disseminated to the bus.*

Theorem 1 roots on automata theory – a binary string I is accepted by M if and only if $I \in L(M)$. The firewall allows the transmission of the rest of the CAN frame only when I is accepted by M, and I wins arbitration. Theorem 1 ensures that CAN spoofing attacks are effectively blocked: when a malicious/compromised ECU tries to send CAN frames with IDs not in the whitelist, the attempt is rejected in the arbitration phase.

Theorem 2. *Let I_m be the lowest arbitration ID accepted by DFA M, any (partial) ID output from M cannot win arbitration against a target ID I_t that has higher priority than I_m (i.e., $I_t < I_m$).*

During the arbitration phase, the DFA generates an intermediate output bit at each state and transmits the output to CAN_{int}. It is possible that the prefix of a spoofed ID ($I_s \notin L(M)$) being sent to CAN_{int}. For instance, if the firewall only allows 0b1010110011, the ID 0b1010001011 will be denied at bit 5. However, its prefix 0b1010 will be sent to CAN_{int}. Theorem 2 shows that adversaries cannot exploit this mechanism to launch arbitration denial attacks against high-priority IDs. Please refer to Appendix A for the proof of this theorem.

Security Analysis of Data Frame Transmission. The external node is authorized to transmit its frame to the bus only when it wins arbitration on CAN_{int}. The firewall enforces **R3** to forward traffic from CAN_{ext} to CAN_{int}. For the simplicity of the design and the portability of the firewalls, we do not further audit the validity of the data frame. When CAN_{ext} loses arbitration, the firewall moves to enforce **R1**, which blocks traffic from CAN_{ext} to CAN_{int}. Therefore, an adversary node cannot inject dominate bit(s) to the bus to interrupt CAN frame from other nodes, which may eventually drive other nodes into bus-off. Such ECU denial and bus denial attacks are denied at the firewall.

Security of CANSentry. The physical and software security of CANSentry relies on the following factors: (1) CANSentry nodes are deployed in a physically isolated environment, i.e., inside the car. It is difficult for an adversary to bypass/alter it unless he has extended time to physically modify the in-vehicle components. (2) CANSentry only has two network interfaces: CAN_{ext} and CAN_{int}. The limited communication channel and the simplicity of CAN makes it impractical to compromise the operations of the firewall from CAN_{ext}. And (3) the simplicity of the firewall makes it unlikely to have significant software faults.

Updates. We do not consider *remote* updates to CANSentry in this paper, to keep the simplicity of CANSentry and to avoid adding a new attack vector. In case a CANSentry-protected ECU is *remotely* updated to send CAN packets with new IDs, such packets will be blocked. In practice, we are not aware of any existing remote update that adds new CAN IDs to the high-risk (e.g., remotely accessible) ECUs. Meanwhile, when the vehicle is updated in the shop, the corresponding CANSentry could be easily updated to add new CAN IDs to the whitelist.

Remaining Attack Surfaces. An adversary node may send to CAN_{int} using IDs on the whitelist, which may potentially block frames from higher CAN IDs.If the adversary node keeps sending at high frequency, it becomes arbitration denial attacks to nodes with *lower priority.* CANSentry cannot detect or block this attack. Fortunately, this may not be a severe problem, since: (1) the attack paths from known attacks (e.g. [32, 39, 27] are all attacking from low priority nodes (i.e., components with wireless interfaces and externally-facing vulnerabilities) to high priority nodes. Attacks from the other direction appear to be meaningless. (2) The *high-risk nodes*, i.e., nodes that have shown to be compromisable by remote adversaries, mostly belong to low priority nodes (entertainment units, navigation, etc), which are only allowed to send CAN frames with high arbitration IDs. (3) Arbitration denial attacks from such nodes are detectable with a simple IDS, which runs on the same chip as the firewall, monitors traffic from CAN_{int}, and informs the firewall to take actions when attacks are detected.

When a spoofing attempt is blocked by CANSentry, a partial frame, which only contains the first few bits of the arbitration ID, will be observed on the bus. An adversary node could then intentionally inject malformed frames to CAN_{int}. Other receiving nodes on the bus will detect *form errors* and raise error flags. This will increase RECs at the receiving nodes, and may drive them into error passive mode, in the worst case. This is not a severe problem since: (1) To force a node into error passive mode, `REC>127` is needed. This requires continuous attack attempts from a adversary node (assuming that the node manages to avoid increasing its own TEC). (2) Such persistent attacks could be easily detected by a simple counter or IDS embedded in CANSentry. And (3) nodes in error passive mode still perform their functions (with a performance penalty), and they can recover from error passive mode when frames are correctly received.

In the attack model, internal ECUs are considered low risk. While we are unaware of any attack that compromises an internal ECU with only brief physical access, we cannot completely rule out this possibility. Lastly, a small number of units in modern automobiles, such as GM's OnStar, are both remotely compromisable and capable of sending high priority frames, e.g., stopping the vehicle by shutting down fuel injection. When such systems are compromised, they may take over control of the cars even with the existence of CANSentry. From the CAN bus perspective, they are fully authorized to send such frames. Defending against such attacks is outside of the scope of CANSentry.

7 Conclusion

In this paper, we first summarize existing DoS and spoofing attacks on the CAN bus, then implement and evaluate a stealthy selective arbitration denial attack. We present a novel CAN firewall, CANSentry, to be deployed between each high-risk ECU (external-facing ECUs with remote attack vectors) and the internal bus. It defends against attacks that violate CAN standards or abuse the error-handling mechanism to achieve malicious goals. To the best of our knowledge, CANSentry is the first mechanism to not only detect but also prevent denial attacks at data link layer that have not been addressed yet, in addition to preventing the traditional spoofing and denial attacks. Despite the simple yet powerful design, CANSentry effectively prevents the attacks with no noticeable overhead at a low cost with no need to modify CAN standard nor existing ECUs.

With hardware implementation and evaluation, we demonstrate the effectiveness of the firewall against the bus/ECU denial attacks and ECU spoofing attacks.

Acknowledgements

We would like to thank the anonymous reviewers for their constructive comments. Fengjun Li and Bo Luo were sponsored in part by NSF CNS-1422206, DGE-1565570, NSA Science of Security Initiative H98230-18-D-0009, and the Ripple University Blockchain Research Initiative. Jingqiang Lin was partially supported by National Natural Science Foundation of China (No. 61772518) and Cyber Security Program of National Key RD Plan of China (2017YFB0802100).

References

1. Grimm co. cant. `https://github.com/bitbane/CANT`.
2. S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*, 2011.
3. Kyong-Tak Cho and Kang G Shin. Error handling of in-vehicle networks makes them vulnerable. In *ACM CCS*, pages 1044–1055. ACM, 2016.
4. Kyong-Tak Cho and Kang G Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *USENIX Security Symposium*, 2016.
5. Kyong-Tak Cho and Kang G Shin. Viden: Attacker identification on in-vehicle networks. In *ACM CCS*, 2017.
6. Wonsuk Choi, Hyo Jin Jo, Samuel Woo, Ji Young Chun, Jooyoung Park, and Dong Hoon Lee. Identifying ecus using inimitable characteristics of signals in controller area networks. *IEEE Trans. on Vehicular Tech.*, 67(6):4757–4770, 2018.
7. Wonsuk Choi, Kyungho Joo, Hyo Jin Jo, Moon Chan Park, and Dong Hoon Lee. Voltageids: Low-level communication characteristics for automotive intrusion detection system. *IEEE TIFS*, 13(8):2114–2129, 2018.
8. Tsvika Dagan and Avishai Wool. Parrot, a software-only anti-spoofing defense system for the can bus. *ESCAR EUROPE*, 2016.
9. Andrea Dardanelli, Federico Maggi, Mara Tanelli, Stefano Zanero, Sergio M Savaresi, R Kochanek, and T Holz. A security layer for smartphone-to-vehicle communication over bluetooth. *IEEE embedded systems letters*, 5(3):34–37, 2013.
10. Mahsa Foruhandeh, Yanmao Man, Ryan Gerdes, Ming Li, and Thidapat Chantem. Simple: single-frame based physical layer identification for intrusion detection and prevention on in-vehicle networks. In *ACSAC*, pages 229–244, 2019.
11. Ian Foster, Andrew Prudhomme, Karl Koscher, and Stefan Savage. Fast and vulnerable: A story of telematic failures. In *USENIX WOOT*, 2015.
12. Sibylle Fröschle and Alexander Stühring. Analyzing the capabilities of the can attacker. In *ESORICS*, pages 464–482, 2017.
13. Mabrouka Gmiden, Mohamed Hedi Gmiden, and Hafedh Trabelsi. An intrusion detection method for securing in-vehicle can bus. In *IEEE STA*, 2016.
14. Rachana Ashok Gupta and Mo-Yuen Chow. Networked control system: Overview and research trends. *IEEE Trans. on Industrial Electronics*, 57(7):2527–2535, 2010.
15. Subir Halder, Mauro Conti, and Sajal K Das. Coids: A clock offset based intrusion detection system for controller area networks. In *ICDCN*, 2020.

16. Kyusuk Han, Swapna Divya Potluri, and Kang G Shin. On authentication in a connected vehicle: Secure integration of mobile devices with vehicular networks. In *ACM/IEEE ICCPS*, pages 160–169, 2013.
17. Kyusuk Han, André Weimerskirch, and Kang G Shin. A practical solution to achieve real-time performance in the automotive network by randomizing frame identifier. In *Proc. Eur. Embedded Secur. Cars (ESCAR)*, pages 13–29, 2015.
18. Oliver Hartkopp and R MaCAN SCHILLING. Message authenticated can. In *Escar Conference, Berlin, Germany*, 2012.
19. Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. Security threats to automotive can networks — practical examples and selected short-term countermeasures. In *SAFECOMP*, 2011.
20. Abdulmalik Humayed, Jingqiang Lin, Fengjun Li, and Bo Luo. Cyber-physical systems security—a survey. *IEEE Internet of Things Journal*, 4(6), 2017.
21. Abdulmalik Humayed and Bo Luo. Cyber-physical security for smart cars: taxonomy of vulnerabilities, threats, and attacks. In *ACM/IEEE ICCPS*, 2015.
22. Abdulmalik Humayed and Bo Luo. Using id-hopping to defend against targeted dos on can. In *SCAV Workshop*, 2017.
23. Kazuki Iehira, Hiroyuki Inoue, and Kenji Ishida. Spoofing attack using bus-off attacks against a specific ecu of the can bus. In *IEEE CCNC*, 2018.
24. Khaled Karray, Jean-Luc Danger, Sylvain Guilley, and M Abdelaziz Elaabid. Identifier randomization: An efficient protection against can-bus attacks. In *Cyber-Physical Systems Security*, pages 219–254. Springer, 2018.
25. Marcel Kneib and Christopher Huth. Scission: Signal characteristic-based sender identification and intrusion detection in automotive networks. In *ACM CCS*, 2018.
26. G. Kornaros, O. Tomoutzoglou, and M. Coppola. Hardware-assisted security in electronic control units: Secure automotive communications by utilizing one-time-programmable network on chip and firewalls. *IEEE Micro*, 38(5):63–74, 2018.
27. K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, et al. Experimental security analysis of a modern automobile. In *IEEE S&P*, 2010.
28. Ryo Kurachi, Yutaka Matsubara, Hiroaki Takada, Naoki Adachi, Yukihiro Miyashita, and Satoshi Horihata. Cacan-centralized authentication system in can (controller area network). In *Int. Conf. on ESCAR*, 2014.
29. Martin Lukasiewycz, Philipp Mundhenk, and Sebastian Steinhorst. Security-aware obfuscated priority assignment for automotive can platforms. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 21(2):32, 2016.
30. Tsutomu Matsumoto, Masato Hata, Masato Tanabe, Katsunari Yoshioka, and Kazuomi Oishi. A method of preventing unauthorized data transmission in controller area network. In *IEEE VTC*, 2012.
31. Charlie Miller and Chris Valasek. Adventures in automotive networks and control units. *Def Con*, 21:260–264, 2013.
32. Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015:91, 2015.
33. Edward F Moore. Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153, 1956.
34. P. Mundhenk, A. Paverd, A. Mrowca, S. Steinhorst, M. Lukasiewycz, S. A. Fahmy, and S. Chakraborty. Security in automotive networks: Lightweight authentication and authorization. *ACM TODAES*, 22(2):1–27, 2017.
35. Pal-Stefan Murvay and Bogdan Groza. Source identification using signal characteristics in controller area networks. *IEEE Signal Processing Letters*, 21(4), 2014.

36. Pal-Stefan Murvay and Bogdan Groza. Dos attacks on controller area networks by fault injections from the software layer. In *ARES*. ACM, 2017.
37. Michael Müter and Naim Asaj. Entropy-based anomaly detection for in-vehicle networks. In *IEEE Intelligent Vehicles Symposium*, 2011.
38. Sandeep Nair Narayanan, Sudip Mittal, and Anupam Joshi. Obd_securealert: An anomaly detection system for vehicles. In *IEEE SMARTCOMP*, 2016.
39. Sen Nie, Ling Liu, and Yuefeng Du. Free-fall: hacking tesla from wireless to can bus. *Briefing, Black Hat USA*, pages 1–16, 2017.
40. N. Nowdehi, W. Aoudi, M. Almgren, and T. Olovsson. Casad: Can-aware stealthy-attack detection for in-vehicle networks. *arXiv:1909.08407*, 2019.
41. Stefan Nürnberger and Christian Rossow. –vatican–vetted, authenticated can bus. In *CHES*, pages 106–124. Springer, 2016.
42. A. Palanca, E. Evenchick, F. Maggi, and S. Zanero. A stealth, selective, link-layer denial-of-service attack against automotive networks. In *DIMVA*, 2017.
43. Jonathan Petit and Steven E Shladover. Potential cyberattacks on automated vehicles. *IEEE Trans. on Intelligent Transportation Systems*, 16(2):546–556, 2015.
44. Andreea-Ina Radu and Flavio D Garcia. Leia: A lightweight authentication protocol for can. In *ESORICS*, 2016.
45. Syed Rizvi, Jonathan Willet, Donte Perino, Seth Marasco, and Chandler Condo. A threat to vehicular cyber security and the urgency for correction. *Procedia computer science*, 114:100–105, 2017.
46. I. Rouf, R. Miller, H. Mustafa, T. Taylor, S. Oh, W. Xu, M. Gruteser, W. Trappe, and I. Seskar. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In *USENIX Security Symposium*, 2010.
47. F. Sagstetter, M. Lukasiewycz, S. Steinhorst, M. Wolf, A. Bouard, W. R Harris, S. Jha, T. Peyrin, A. Poschmann, and S. Chakraborty. Security challenges in automotive hardware/software architecture design. In *DATE*. IEEE, 2013.
48. Daisuke Souma, Akira Mori, Hideki Yamamoto, and Yoichi Hata. Counter attacks for bus-off attacks. In *SafeComp*, 2018.
49. Ivan Studnia, Vincent Nicomette, Eric Alata, Yves Deswarte, Mohamed Kaaniche, and Youssef Laarouchi. Survey on security threats and protection mechanisms in embedded automotive networks. In *IEEE/IFIP DSN*, 2013.
50. Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz. Anomaly detection in automobile control network data with long short-term memory networks. In *IEEE DSAA*, 2016.
51. Andreas Theissler. Detecting known and unknown faults in automotive systems using ensemble-based anomaly detection. *Knowledge-Based Systems*, 123, 2017.
52. Daxin Tian, Yuzhou Li, Yunpeng Wang, Xuting Duan, Congyu Wang, Wenyang Wang, Rong Hui, and Peng Guo. An intrusion detection system based on machine learning for can-bus. In *INISCOM*, 2017.
53. Jo Van Bulck, Jan Tobias Mühlberg, and Frank Piessens. Vulcan: Efficient component authentication and software isolation for automotive control networks. In *ACSAC*, pages 225–237, 2017.
54. Anthony Van Herrewege, Dave Singelee, and Ingrid Verbauwhede. Canauth-a simple, backward compatible broadcast authentication protocol for can bus. In *ECRYPT Workshop on Lightweight Cryptography*, volume 2011, 2011.
55. Qiyan Wang and Sanjay Sawhney. Vecure: A practical security framework to protect the can bus of vehicles. In *IEEE Intl. Conf. on IOT*, 2014.
56. Marko Wolf, André Weimerskirch, and Christof Paar. Security in automotive bus systems. In *Workshop on Embedded Security in Cars*, 2004.

57. Samuel Woo, Hyo Jin Jo, and Dong Hoon Lee. A practical wireless attack on the connected car and security protocol for in-vehicle can. *IEEE Transactions on intelligent transportation systems*, 16(2):993–1006, 2014.
58. Samuel Woo, Daesung Moon, Taek-Young Youn, Yousik Lee, and Yongeun Kim. Can id shuffling technique (cist): Moving target defense strategy for protecting in-vehicle can. *IEEE Access*, 7:15521–15536, 2019.
59. Wufei Wu, Ryo Kurachi, Gang Zeng, Yutaka Matsubara, Hiroaki Takada, Renfa Li, and Keqin Li. Idh-can: A hardware-based id hopping can mechanism with enhanced security for automotive real-time applications. *IEEE Access*, 2018.
60. Tobias Ziermann, Stefan Wildermann, and Jurgen Teich. Can+: A new backward-compatible controller area network (can) protocol with up to 16× higher data rates. In *DATE*. IEEE, 2009.

A Proof of Theorem 2

Theorem 2. *Let I_m be the lowest arbitration ID accepted by DFA M, any (partial) ID output from M cannot win arbitration against a target ID I_t that has higher priority than I_m (i.e., $I_t < I_m$).*

Proof. Assume that the adversary attempts to spoof arbitration ID I_s ($I_s < I_t$) to win against I_t. Let $Bit(S, i)$ be the i-th bit of string S. If the longest common prefix of I_m, I_s, and I_t is an i-bit string P, then the first i bits of I_s would be accepted in M (since $Prefix(I_s, i) = Prefix(I_m, i)$) and sent to CAN accordingly. I_t and I_s would tie in the first i bits of arbitration (since $Prefix(I_s, i) = Prefix(I_t, i)$). At bit $i + 1$, we have the following three conditions:

• If $Bit(I_s, i{+}1) < Bit(I_m, i{+}1)$, I_s will be rejected by M, since: (1) I_s cannot be accepted by the DFA branch that contains I_m, since $Bit(I_s, i{+}1) \neq Bit(I_m, i{+}1)$; (2) if there exist another DFA branch (with ID I_n) that accepts $Bit(I_s, i + 1)$, then we have $I_n < I_m$ (since they are identical in the first i bits and $I_n < I_m$ at bit $i + 1$). This violates our assumption that I_m be the lowest arbitration ID accepted by M. Therefore, such I_n and the corresponding DFA branch does not exist. I_s will be rejected by M, and I_t wins arbitration against I_s.

• If $Bit(I_s, i{+}1) > Bit(I_m, i{+}1)$, then we have $I_s > I_m$, since they are identical in the first i bits and $I_s > I_m$ at bit $i + 1$. This violates our assumption that $I_s < I_t < I_m$.

• If $Bit(I_s, i + 1) = Bit(I_m, i + 1)$, then $Bit(I_s, i + 1)$ will be sent to the bus. Meanwhile, we need $Bit(I_s, i + 1) \neq Bit(I_t, i + 1)$, otherwise $Prefix(I_s, i + 1)$ is a longer common prefix than P. In case $Bit(I_s, i + 1) > Bit(I_t, i + 1)$, I_s would lose arbitration against I_t. In case $Bit(I_s, i + 1) < Bit(I_t, i + 1)$, then we have $I_m = I_s < I_t$. This violates our assumption that $I_t < I_m$.

In summary, with the existence of M, any (partial) output generated by $I_s < I_m$ cannot win arbitration against a higher priority ID $I_t < I_m$. □