2021 IEEE Symposium on Security and Privacy (SP)

CANnon: Reliable and Stealthy Remote Shutdown Attacks via Unaltered Automotive Microcontrollers

Sekar Kulandaivel,[*] Shalabh Jain,[†] Jorge Guajardo,[†] and Vyas Sekar[*]
[*]*Carnegie Mellon University*, [†]*Research and Technology Center, Robert Bosch LLC, USA*
{skulanda, vsekar}@andrew.cmu.edu, {shalabh.jain, jorge.guajardomerchan}@us.bosch.com

2021 IEEE Symposium on Security and Privacy (SP) | 978-1-7281-8934-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/SP40001.2021.00122

Abstract—**Electronic Control Units (ECUs) in modern vehicles have recently been targets for *shutdown* attacks, which can disable safety-critical vehicle functions and be used as means to launch more dangerous attacks. Existing attacks operate either by physical manipulation of the bus signals or message injection. However, we argue that these cannot simultaneously be remote, stealthy, and reliable. For instance, message injection is detected by modern Intrusion Detection System (IDS) proposals and requires strict synchronization that cannot be realized remotely. In this work, we introduce a new class of attacks that leverage the peripheral clock gating feature in modern automotive microcontroller units (MCUs). By using this capability, a remote adversary with purely software control can reliably "freeze" the output of a compromised ECU to insert arbitrary bits at any time instance. Utilizing on this insight, we develop the *CANnon* attack for remote shutdown. Since the *CANnon* attack produces error patterns indistinguishable from natural errors and does not require message insertion, detecting it with current techniques is difficult. We demonstrate this attack on two automotive MCUs used in modern passenger vehicle ECUs. We discuss potential mitigation strategies and countermeasures for such attacks.**

Index Terms—**Automotive security, CAN bus attack, Fault attacks, Glitching attacks**

I. Introduction

Modern in-vehicle networks contain tens of Electronic Control Units (ECUs) that communicate over a shared medium known as the Controller Area Network (CAN) bus. Some of these ECUs that introduce new wireless connectivity (e.g. Bluetooth, cellular, Wi-Fi), which provide a variety of services to vehicle owners, have exposed the in-vehicle network to external network attacks. The feasibility and ease of launching attacks against the CAN bus have been demonstrated by several researchers over the past few years [1]–[4].

The lack of security in in-vehicle networks allows an adversary with access to the CAN bus to arbitrarily insert, modify, and delete messages, allowing an attacker to manipulate the functionality of safety-critical ECUs [1] or limit communication over the bus [5], [6]. While traditional attacks utilize *physical* interfaces to gain bus access, researchers have demonstrated the ability to gain access remotely [4]. This demonstration caused the recall of 1.4 million vehicles and attracted the attention of automotive manufacturers, suppliers, and global regulatory bodies.

As a defense against an evolving threat landscape, academic and industry researchers have proposed a variety of techniques, such as message authentication [7], [8], intrusion detection systems (IDSes) [9]–[12], and secure CAN hardware solutions [13]. Considering the potential societal impact of automotive attacks, regulatory bodies have proposed introducing legal mandates to equip future vehicles with security features, e.g. IDSes [14]. Even hardware defenses in the form of secure transceiver concepts [13] have been proposed to increase security of the in-vehicle CAN bus.

Despite efforts to increase the security of automotive networks, a recent class of attacks demonstrates significant adversarial potential by utilizing the inherent CAN protocol framework to *shut down* safety-critical ECUs. Such attacks introduced by prior work [5], [6], [15] are particularly dangerous due to their ability to disable critical vehicle functionality by shutting down several ECUs from just a single compromised ECU. Additionally, an adversary could use shutdown attacks to launch advanced attacks, e.g. stealthy masquerade attacks [16], [17]. Current shutdown attacks repeatedly trigger the error-handling mechanism on a victim ECU, causing it to enter the *bus-off* error-handling state that shuts down the ECU's CAN communication. This attack is achieved by either physical manipulation of the bus [5], [6] or carefully synchronized and crafted transmissions [15]. However, these proposals either lack stealthiness against existing security proposals [10], [11], [13], require physical access [5], [6], or require strict control (e.g. synchronization) that cannot be achieved in practical remote settings [15].

In this paper, we introduce a fundamentally different approach towards mounting shutdown attacks that, to the best of our knowledge, can evade all existing known defenses. Our attack is facilitated by architectural choices made to improve the integration and efficiency of automotive ECUs and their microcontroller units (MCUs). Modern MCUs typically integrate the CAN interface controller as an on-chip (CAN) peripheral in the same package. This design allows new inputs to the CAN peripheral to be accessible to the application-layer software via an application programming interface (API) and, thus, accessible to a remote adversary that successfully compromises an ECU.

We develop *CANnon*, a method to maliciously exploit one such input, namely the *peripheral clock gating* functionality. This particular API is accessible via software control in most modern automotive MCUs, often included as a valuable feature for performance optimization. We demonstrate how a remote software adversary can employ *CANnon* to utilize the CAN peripheral's clock to bypass the hardware-based CAN protocol compliance and manipulate the ECU output. This capability

DOI 10.1109/SP40001.2021.00122

enables the adversary to inject *arbitrary* bits and signals (as compared to only being able to inject complete CAN-compliant frames) and gain the ability to precisely shape the signals on the CAN bus with bit-level accuracy. We demonstrate that this capability can be used to perform reliable and stealthy shutdown attacks. In other words, the modern MCU design has inadvertently strengthened the capabilities of a remote adversary, who is no longer constrained by CAN protocol compliance.

Our main insight here is the ability to *control* the peripheral's clock signal to "pause" the ECU state in the middle of a transmission (or between state transitions). By exercising this control to selectively pause and resume an ECU's transmission, we can insert an arbitrary bit for a duration and at a time instance of our choice. This bit can be used to overwrite a victim's message and cause it to detect transmission errors. We also illustrate that the pattern of errors produced by *CANnon* is difficult to distinguish from legitimate errors on the CAN bus. Our fine control over malicious *bit* insertion (rather than message insertion) makes the detection of *CANnon* attacks difficult for currently proposed IDSes, as current techniques typically analyze *entire* messages for signs of malicious activity. Additionally, as *CANnon* does not involve spoofing IDs or overwriting the content of a message, even ID-based filtering at the data link layer [13] seems incapable of detecting our attack.[1] Preventing *CANnon*-based attacks require either architectural-level changes, such as isolation or removal of the clock control, or modifying existing approaches to specifically detect *CANnon*-like patterns. In Table I, we summarize existing works and contrast them with *CANnon*, which we further detail in Sec. II-B.

Contributions: In summary, we contribute the following:

- We introduce new methods to exploit the peripheral clock gating API of automotive MCUs to bypass hardware-based CAN protocol compliance and inject arbitrary bits on the bus. In contrast to previous work, we do not exploit diagnostic messages [4], [18], [19] and do not have tight synchronization requirements [15].
- We present three stealthy versions of *CANnon* and discuss modifications to make *CANnon* stealthy against future defenses.
- We illustrate both a basic denial-of-service (DoS) attack and a targeted victim shutdown attack atop two modern automotive MCUs used in passenger vehicles: the Microchip SAM V71 MCU and the STMicro SPC58 MCU. We validate the feasibility of this attack against a 2017 Ford Focus and a 2009 Toyota Prius and achieve a shutdown in less than 2ms.
- We propose several countermeasures to detect/prevent *CANnon* attacks for legacy and future vehicles.

TABLE I: Characteristics of shutdown attacks

Source	**Attack type**	*Remote*	*Reliable*	*Stealthy*
[5], [6]	Direct bit injection		✓	✓
[1], [2], [4]	Diagnostic message	✓	✓	
[15]	Message overwrite	✓		
	CANnon	✓	✓	✓

Organization: The remainder of this paper is organized as follows. We provide relevant background on the CAN protocol and discuss existing shutdown work in Sec. II. In Sec. III, we detail our attack insight, and we then demonstrate two practical applications of the attack in Sec. IV and V. We demonstrate the attack on production ECUs against real vehicles in Sec. VI and illustrate the stealth properties of *CANnon* in Sec. VII. Finally, we propose some countermeasures in Sec. VIII, identify other related work in Sec. IX, and discuss future directions and conclusions in Sec. X.

Disclosure and availability: We disclosed this vulnerability to several automotive stakeholders.[2] Our conversations with the MCU manufacturers (Tier-2 automotive suppliers) reveal their emphasis on software hardening to prevent the adversarial scenario required here; thus, obligations lie with the ECU integrators (Tier-1 automotive suppliers). We have also made the *CANnon* implementation using an Arduino Due board available [20] to encourage further designs using *CANnon* and to test defense strategies.

II. CAN Preliminaries

We start with background on the CAN protocol and highlight characteristics that render CAN nodes vulnerable to shutdown attacks and discuss prior work on such attacks.

A. CAN background

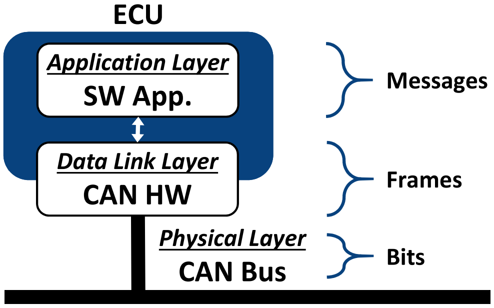


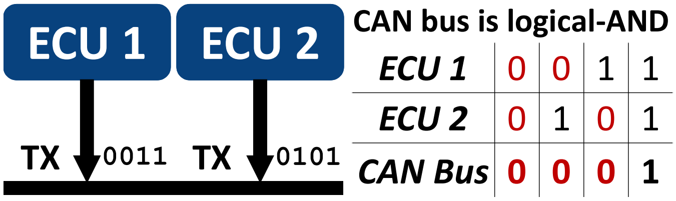
Fig. 1: CAN communication stack

The CAN protocol stack as shown in Fig. 1 is composed of the application layer, data link layer, and the physical layer. The functionality of an ECU (e.g. engine control, driver assistance) is described via high-level software running at the application layer. For actuation and sensing functionality, messages are transmitted and received by the application layer through the lower layers of the communication stack. To send data to another ECU, the application layer creates a CAN message with a priority tag (also referred to as message or arbitration ID) and its payload. The application transfers this

[1] Some recently proposed secure transceiver architectures use such filtering, but it is unclear from publicly available information whether they implement additional countermeasures. We have not evaluated any such products in the market to check their resistance against the *CANnon* attack.

[2] We disclosed the vulnerability to the two MCU manufacturers discussed in this paper. We also performed a broader disclosure via an industry forum.

message to the CAN data link layer, where various control and integrity fields are appended to generate a *frame*, which is transmitted serially via the CAN physical layer. To receive a message, a recipient ECU's data link layer interprets and validates the CAN frame prior to delivery of the message (ID and payload) to the application layer.



CAN bus is logical-AND

ECU 1	0	0	1	1
ECU 2	0	1	0	1
CAN Bus	**0**	**0**	**0**	**1**

Fig. 2: CAN physical layer

The physical layer of the stack, i.e. the physical CAN bus, consists of a broadcast communication medium between multiple ECUs. The bus has two logical states: the dominant (logical-0) state, where the bus is driven by a voltage from the transmitting ECU, and the recessive (logical-1) state, where the bus is passively set. The effective bus state is the logical-AND of all transmitting ECUs' outputs as illustrated in Fig. 2. ECUs connected to the CAN bus communicate at a pre-determined bus speed set by design based on the physical limitations of the bus. The length of each bit is directly determined by the set speed. For example, an ECU communicating at 500Kbps transmits the dominant signal for 2μs to assert a logical-0. Similar to other asynchronous protocols (e.g. Ethernet), CAN nodes rely on frame delimiters for interpreting the start and stop of CAN frames. Each ECU (re)synchronizes its internal clock based on observed transitions on the bus.

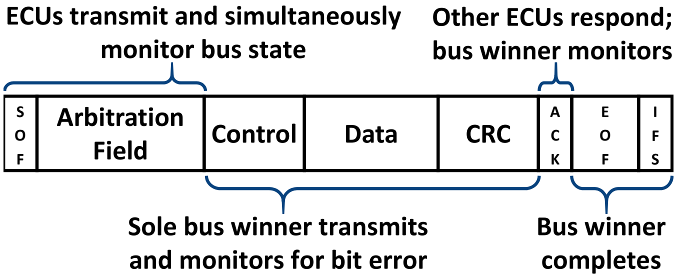

ECUs transmit and simultaneously monitor bus state
Other ECUs respond; bus winner monitors
SOF
Arbitration Field
Control
Data
CRC
ACK
EOF
IFS
Sole bus winner transmits and monitors for bit error
Bus winner completes

Fig. 3: CAN frame format

The CAN data frame illustrated in Fig. 3 has four logical sections: (1) arbitration, (2) data transmission, (3) acknowledgement (ACK), and (4) end-of-frame (EOF) and inter-frame spacing (IFS). Upon detection of an idle bus, an ECU initiates the frame transmission with a dominant start-of-frame (SOF) bit followed by the arbitration ID. Due to CAN's asynchronous nature, multiple ECUs may begin transmission at the same time. While transmitting the ID, an ECU monitors the bus state and stops transmitting if it observes a bit different from the one transmitted. A received dominant bit during a recessive transmission by a node indicates the transmission of a higher-priority message by a different ECU. By the end of arbitration, a single ECU with the highest-priority frame wins access to the bus and continues transmitting. The bus winner transmits the rest of its frame and, for each transmitted bit, monitors that the bus state matches the transmitted bit. During the ACK slot, the transmitter asserts a recessive bit while all receiving ECUs transmit a dominant bit to indicate correct reception. Finally, the sender transmits recessive EOF and IFS bits, where the IFS is the minimum space between two frames on the bus. After the IFS, the bus is idle and holds a recessive state until the next transmission. Each ECU can transmit multiple IDs, but each ID *should* only originate from a single ECU.

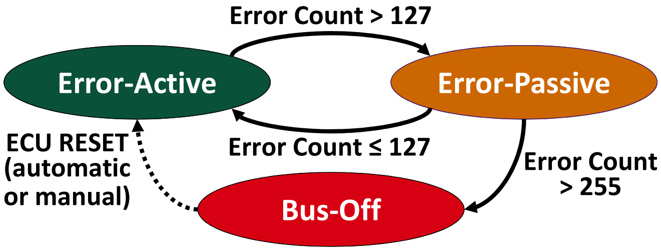


Fig. 4: Three states of error-handling mechanism

Error handling and bus-off state: Error handling is an essential feature of the CAN protocol, providing robustness in automotive environments. The CAN protocol defines several types of errors; we detail two relevant error types, namely the *bit error* and *stuff error*. A bit error occurs when the transmitting node detects a mismatch between a transmitted bit and the bus state (outside of the arbitration and ACK fields). A stuff error occurs in the absence of a stuff bit, which is a bit of opposite polarity intentionally added after every five consecutive bits of the same polarity. When an ECU detects an error, it transmits a 6-bit error flag on the bus that can destroy the contents of the current frame. Depending on the error state of the ECU, the flag may be a sequence of recessive or dominant bits. Each ECU maintains error counters that are incremented upon a transmission error[3] detection and decremented upon a successful transmission. As depicted in Fig. 4, there are three error states based on the error count: (1) error-active, (2) error-passive, and (3) bus-off. An ECU in error-active state represents a "low" error count and transmits a 6-bit *active* (dominant) error flag; an ECU in error-passive indicates a "high" error count and transmits a 6-bit *passive* (recessive) error flag. If enough errors are detected and the count surpasses 255, then an ECU transitions to bus-off, where it will shut down its CAN operations and effectively remove itself from the bus.

B. *Shutdown via errors*

While a large error count in a non-adversarial scenario is indicative of a faulty node and, hence, isolation (or even shutdown) is a logical solution to prevent disruption of the whole network, an adversary can misuse the error mechanism

[3] There is a separate count for reception errors, but it is not relevant to this work. All references to error count refer to the transmission error count.

by causing intentional errors, forcing an ECU to transition into the bus-off state and thus causing the ECU to shut down CAN communication. However, producing intentional errors on the CAN bus without direct access to the physical medium is challenging. One reason is the compliance to the CAN protocol enforced by hardware CAN controllers designed and certified for robustness. Thus, without access to the physical medium, an adversary can only control the ID and payload but *not* the transmitted frame. Nevertheless, recent works (as summarized in Table I) demonstrate limited success operating under these constraints to cause a shutdown.

Errors via physical access: An adversary with physical access can easily bypass the CAN data link layer and inject bits by either sending signals directly to the physical bus or modifying the CAN controller to disobey the protocol. An adversary can also use this access to directly inject dominant bits at any time during a victim's transmission and cause bit errors. Several works [5], [6] use this approach to demonstrate effective shutdown attacks that are difficult to detect as such errors are indistinguishable from genuine bus faults. These attacks have real-time feedback from the bus, enabling a reliable method of shutdown. However, because they require physical access, they are considered impractical both in research and practice as there are easier alternatives to cause harm [12].

Errors via remote access: Prior work [15] demonstrated the ability to overwrite messages and exploit CAN's error-handling mechanism without physical access. Here, an adversary must estimate a victim's message transmission time. As most CAN messages *theoretically* tend to be periodic, an adversary could perform this attack via empirical analysis. Using these estimates, a remote adversary in control of the MCU's software can transmit an attack message at the same time and with the same arbitration ID as the victim. This approach results in two nodes winning control of the bus and intentionally violates the CAN bus protocol. A specially-crafted payload (a dominant bit in place of a victim's recessive bit) can cause the victim to detect a bit error and retransmit its message; by repeating this attack, the victim eventually shuts down. Recent work [21] demonstrates that this attack is not reliable as the deviation of periodic messages varies significantly in practice.

Alternative shutdown mechanisms: While abuse of errors is one method to shut down an ECU, there are other means to shut down ECUs outside the protocol. One method, originally intended for a mechanic to perform ECU testing, exploits diagnostic messages reliably transmitted by a remote adversary [4], [18], [19]. However, such messages use known arbitration IDs and are easily detectable by automotive defense methods.

III. Attack Overview

A. Adversary model

For our attacks, we consider a *remote* adversary that is able to compromise an in-vehicle CAN node capable of transmitting messages on the bus via the CAN stack. As market research estimates that about 150 to 250 million connected cars will be on the road in 2020 [22]–[24], a remote adversary will likely target the infotainment ECU or other ECUs in the vehicle with one or more remote interfaces [25]. We follow the same assumptions of prior work [10], [11], [15], [16], which assume that the adversary can modify the application software on an ECU's MCU and utilize any interfaces or APIs available to this software. However, we assume the adversarial capabilities are limited to *only* software manipulation and do not allow for direct *physical modifications or probing* to any of the vehicle's components; in this work, our targets are unaltered modern passenger vehicles with only original ECUs.

Several prior and recent works [3], [4], [26]–[28] demonstrate the real existence of vulnerabilities to remotely compromise in-vehicle ECUs and gain the ability to take control of physical vehicle functions via CAN transmissions. These works also demonstrate that remote attacks can occur at a large scale since a single vulnerability can be present across hundreds of thousands of vehicles [4]. These real-world demonstrations show that a remote adversary can exploit remote wireless interfaces to modify and/or inject code into software running on a vehicle's MCUs. As outlined in two U.S. agency reports [12], [29], a remote adversary is considered the highest risk factor for the automotive community and passenger safety. Security efforts by vehicle manufacturers, e.g. introduction of IDSes, place significant focus on defending after such an adversary breaches the network [12]. For the remainder of this paper, when we describe the remote adversary, we use API and application-layer control interchangeably to refer to the software instructions that the adversary can control.

Attack goals: In general, a remote adversary will likely target non-safety-critical ECUs (e.g. the head unit or navigation system), which often have remote wireless interfaces to handle multiple high-performance functions. As this adversary likely cannot gain direct compromise of a safety-critical ECU, the adversary will aim to utilize a compromised ECU to influence the functionality of a different (typically safety-critical) ECU in the vehicle without being detected by any deployed network security mechanisms, e.g. IDSes. One way to achieve this attack using the compromised ECU is to shut down a critical ECU and then disable its functions or impersonate it after the shutdown. In this work, we focus on achieving a shutdown of a critical ECU without being detected by state-of-the-art network defenses, i.e. the adversary succeeds if the defense cannot detect an attack *prior* to the shutdown event. As we will demonstrate, the ability to reliably inject an arbitrary bit at an arbitrary time without being detected by vehicle defenses is sufficient to achieve these goals.

Thus, we effectively explore the possibility to construct a reliable remote bit insertion attack, which aims to shut down an ECU, operates as a software application, does not require access or changes to the physical CAN hardware, and deceives even state-of-the-art defenses. Furthermore, although several attacks outlined in Sec. II-B achieve similar goals, to the best of our knowledge, existing shutdown mechanisms cannot simultaneously be remote (performed only at the application layer), reliable (ability to consistently succeed), and stealthy (ability to deceive known defenses). The *CANnon* attack shows that the adversary model used by the industry has *changed* and that the attacker now has *new capabilities* that prior defenses did not consider. The notion of stealth is difficult to characterize, considering the rapid progress in defense mechanisms. For this work, we consider the best-known results for defense as described in Sec. VII.

B. High-level attack insight

Contrast with prior invasive glitch attacks: Creating artificial clock glitches is a common technique to bypass security of MCUs during boot or verification [19] by invasively driving the clock signal line to ground. The idea behind such a technique is to create artificial transitions in the state machines implemented in hardware. As described in Sec. II-B, the difficulty in injecting arbitrary bits is the CAN protocol enforcement by the CAN data link layer, i.e. the CAN controller. Thus, similar to the security logic above, the controller can be viewed as a hardware-based state machine that enforces CAN protocol compliance. Thus, we draw inspiration from the same direction of work but without requiring invasive physical access to the clock.

***CANnon* attack anatomy:** Any finite-state machine (FSM), e.g. the CAN protocol, implemented using sequential logic elements (flip-flops, registers, etc.) relies on the clock signal for state transitions and thus any output transmissions. Therefore, control of the clock signal can be used to force arbitrary deviations from the protocol. As an example, small changes in clock frequency would directly result in a change of the bit duration on the CAN bus.

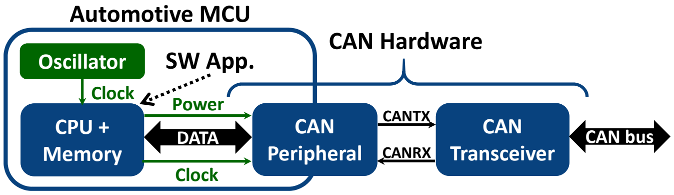


Fig. 5: Modern ECU design includes CAN peripheral that runs off gated clock signal from MCU's oscillator

Clock control is now possible: In an ideal design, the clock signal should not be accessible by a remote adversary. However, for modern ECUs, the MCU is a multi-chip module, where the CAN controller is integrated into the same package as the MCU and is now called a CAN peripheral. A simplified example of the modern ECU architecture is shown in Fig. 5. Additionally, most modern MCU architectures implement power optimization in the form of *peripheral clock gating*. This low-power feature saves energy by shutting down any unwanted peripherals when they are not required, while allowing the main CPU and other critical functions in the MCU to still operate. As the CAN controller is typically attached as a peripheral to the MCU chip, there are controls exposed to cut off the CAN peripheral's clock.

To allow flexibility and control to low-level system designers, most MCUs provide the system designer a small software interface for the controls that allow clock cut-off. As demonstrated in Sec. VI, clock control can be arbitrarily exercised during regular operations, which can also provide a remote adversary in control of the software with the same ability to control the CAN protocol FSM. This control effectively allows an adversary to *gate* the clock and *freeze* the protocol FSM, only to later restart the clock to resume the FSM. Thus, this new capability allows an adversary to arbitrarily manipulate the CAN protocol *without* modifying the hardware.

We note that, in most scenarios, cutting off the clock does not affect any data present in the sequential elements or the outputs of the logic gates. It simply prevents a change in the state of these elements. Also, without architectural changes, the notion of a *frozen state* or disabled clock cannot be recognized or corrected by the CAN controller. An alternative control in the form of power gating may also be available in certain chips, and we investigated exploiting such mechanisms. However, we find that disrupting the power simply resets the peripheral and its buffers/registers, causing the CAN FSM and output to be reset. Ultimately, we discover this attack vector in the driver code for the CAN peripheral. In hindsight, we realize that another factor that enabled our discovery of this vulnerability was our choice in experimental setup (detailed in Sec. VI), which closely resembles the modern MCU architecture, whereas most prior research has continued to use the legacy architecture.

C. Overview of the attack

For any transmission, the CAN controller outputs the bits of the frame serially onto the transmit output pin (CANTX in Fig. 5), where each new bit is triggered by a clock transition. The binary output of the controller is converted to a CAN-compatible analog value on the bus by the transceiver.

Consider the case when the CAN controller is transmitting a dominant logical-0 bit. If the clock is disabled (paused) before the next bit, the CANTX output would continue to be logical-0 until the next clock edge. Thus, the node would continue to assert a dominant signal until the clock signal is resumed. This action effectively allows the transmission of a dominant bit of arbitrary duration. Now consider the opposite case when the CAN controller is transmitting a recessive logical-1 bit. If the clock is disabled, it would continue to assert a recessive value on the bus, i.e. no signal. The rest of the payload resumes transmission only when the clock signal is resumed. This action allows the transmission of the payload at an arbitrary time. Observe that the adversary exploits the controller's inability to sense the bus state when its clock is in the paused state. Thus, resuming the clock resumes the FSM from the point it was stopped, regardless of the current bus state or the controller's transmission output on the bus. This fact is key to disable the back-off arbitration control in CAN controllers and to transmit a signal at an arbitrary time.

IV. Basic Remote Disruption Attack

In what follows, we take a step-wise approach to increase the sophistication of our attack, ultimately demonstrating a controlled victim shutdown. In this section, we begin with a simple application of clock control to disrupt the entire network via a denial-of-service (DoS) attack. This basic disruption also highlights practical constraints that we must consider to design a reliable attack strategy. We note that this basic attack is easy to detect, and current hardware measures can sufficiently protect against it. However, the techniques we describe are the basis for precise and consecutive error injections required for the targeted shutdown attack in Sec. V.

Clock gating at application layer: The primary requirement for this attack is that the MCU must have control over the clock input for its peripherals, e.g. controllers for different protocols, such as CAN, Ethernet, FlexRay, etc. For the attack we present here, we choose a popular hardware device with a high-performance MCU built for networking applications: the Arduino Due board with an AT91SAM3X8EA 32-bit MCU operating at 84 MHz [30]. The Arduino Due offers many networking peripherals (e.g. CAN) and its source code (and CAN drivers) are well-documented, making it ideal for demonstrating our insights. In fact, we find that MCUs marketed as "high-performance" often include peripheral clock gating as a low-power feature available for the system designer (and thus a remote adversary).

Another requirement is that enabling/disabling the clock signal should not reset the peripheral circuitry or change values of its logic elements. Ideally, disabling the clock should only prevent the sequential elements from transitioning to a new state. This fact holds true for basic clock control mechanisms. For the APIs of the automotive MCUs we evaluate in Sec. VI, we find the presence of multiple instructions that control the clock. Typically, for some of the commonly used APIs, MCU designers may implement additional check routines before/after a clock disable instruction to ensure error-free functioning, e.g. check status of transmission, etc. However, these procedures were only implemented for some of the available clock control instructions, and we find at least one instruction that offers a basic control mechanism.

To use the clock control, the adversary must identify which instructions enable an MCU's application to control peripheral clock signals. Typically, manufacturers provide basic driver code for initialization of several peripherals as part of their software development kit (SDK). In such cases, we can discover clock control instructions in the *drivers* for the CAN peripheral. Alternatively, in the event that all clock control instructions are not completely detailed, the reference/programming manuals for a given MCU often outline the steps required to control the peripheral clock and will provide the specific registers that control the clock gating. In the driver for the Arduino Due, we discover the instructions, `pmc_enable_periph_clk()` and `pmc_disable_periph_clk()`, to enable and disable the clock, respectively. These instructions appear prior to low-level configurations, e.g. memory allocation, buffer flushes, etc. However, for another MCU popular in the automotive community, the STMicro SPC58, finding equivalent clock control instructions was more challenging as directly disabling the peripheral clock was not possible. Thus, we use its reference manual to identify specific registers that grants us a similar clock control capability in Sec. VI.

Simple disruption attack: Recall that the CAN bus state is dominant if at least one ECU transmits a dominant bit. As a CAN frame consists of a series of dominant and recessive bits that follow a particular format, no valid information is conveyed from a single state held on the bus. Additionally, such a condition would result in continuous errors in the ECUs due to absence of stuff bits.

Thus, a basic attack we conceive is to *disrupt the bus* by holding the bus in the dominant state. This disruption would prevent any other ECU from transmitting, leading to a DoS of all ECUs in the network. An adversary could perform this action at a critical time (e.g. while the vehicle is in motion) and disrupt key vehicle functionality. For most vehicles, this attack would result in loss of control by the driver.

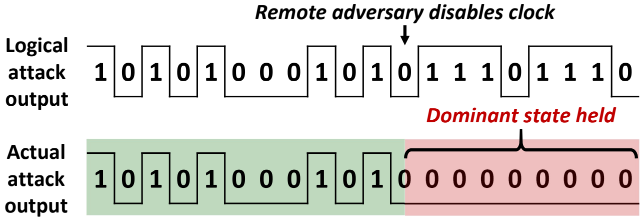


Fig. 6: Holding dominant state disrupts the bus

Using clock control instructions, the adversary could easily achieve this attack by disabling the clock and *freezing* the CAN controller state when it transmits a dominant bit. To launch this attack, a basic method is to target the start-of-frame (SOF) bit:

- Send a message transmission request to the CAN peripheral with any priority and payload.
- Use a timer to delay for half a bit length for the given bus speed so the peripheral starts the transmission of the SOF bit.
- Pause the clock using the disable command to freeze the state of the CAN controller during the SOF bit.

If the bus was initially idle, this sequence would likely lead to the node continuing to hold the dominant state as depicted in Fig. 6. However, there are several practical challenges evident from these basic steps. One critical challenge we encounter is the precise timing required to freeze the controller during the target SOF bit. In practice, the selected delay value only works if the bus was idle when the transmission request was sent and the frame immediately transmitted. In a real scenario, the transmission may start much later, e.g. other bus traffic, scheduling delay, etc. Even minor variations in the timer used to realize the delay period can cause an adversary to miss the SOF bit. Furthermore, any variation in the latency of the actual clock gating effect from the time that the instruction was issued can cause an adversary to miss the SOF bit.

Although there are practical constraints in this attack, the simplicity of this attack (as a result of our attack insight) affords an adversary a great deal of flexibility. For example, missing the SOF bit could be compensated for by using an ID of 0x0 and data payload of 0x0 (essentially a message of all zeros). Thus, freezing the controller during the arbitration or data payload field would also disrupt the bus. However, even this all-zero message has recessive bits due to bit stuffing when converted to a CAN frame. Thus, accidentally encountering those bits due to unreliable timing can cause the attack to fail.

This disruption attack is easy to prevent (if not already prevented) by most modern CAN buses. This disruption attack closely resembles a typical hardware fault encountered in a real CAN bus, i.e. bus held-dominant faults. Thus, several existing CAN transceivers implement built-in mechanisms to prevent the bus from holding a dominant state for a long period of time. This attack demonstrates the practical feasibility of using the clock control to launch an attack. This attack, though potentially dangerous, is highly obstructive for all nodes. It is still short of the goal of this work, which is to target a single ECU with high reliability and without being detected.

V. Reliable Target Victim Shutdown

In this section, we address some of the limitations discussed in the previous section to achieve a *reliable* attack that can *target a specific victim* ECU and quickly shut it down. We detail three variants of the *CANnon* attack and provide solutions to challenges observed in practical scenarios.

A. Reliable clock control

In Sec. IV, we illustrated the difficulty to ensure the clock is paused during a dominant bit. In general, an adversary with unreliable control of the clock cannot precisely ensure what state the controller outputs. Also, unlike the previous attack, a targeted attack usually requires overwriting specific bits of a victim message, thus requiring even more precision. One source of this unreliability is the variation in latency of the clock gating instructions, before the clock is actually paused. Another issue for this attack is that the adversary must track the state of the CAN bus and its own transmissions in order to target specific bits. However, when the CAN controller is in the frozen state, it does not have the ability to observe the CAN bus state. Without feedback, the adversary is virtually blind to the actual peripheral output while performing an attack. Thus, the adversary must keep track of which bit of a compromised ECU's frame it is transmitting at all times.

When the adversary calls a clock gating instruction (either enable or disable), we experimentally find that it takes up to two MCU clock cycles for the instruction to resume the peripheral's output. Thus, the adversary cannot reliably gate the clock near the edge of a bit transition of the attack message. A nonzero latency means that the adversary cannot ensure whether a gating instruction results in the output of the state before or after the state (bit) transition. This latency can thus influence the state of the bus that is held when the controller is frozen. Additionally, an adversary will need to make repeated calls to gating instructions within a single frame transmission by the compromised ECU. If the adversary loses precision in their clock control at any time, they could lose track of which bit the compromised ECU is currently transmitting.

Improving precision: To address the challenge of reliable clock control, the adversary can take advantage of the fact that the MCU's clock operates at a much higher frequency than the CAN peripheral's clock. We utilize the MCU's hardware-based timer, operating it at a frequency equal to the bus speed. This timer creates interrupts at the middle of each CAN bit, which allows us to track exactly which bit the compromised ECU is transmitting. Prior to starting the timer, the adversary must first detect when the compromised ECU attempts to send a frame; from this point, the adversary should delay half of a bit time before starting the timer interrupt. Our solution is to gate the clock as close to the *middle of a CAN bit*, giving the adversary maximum distance from bit transition edges. With an interrupt at the middle of each bit, the adversary can reliably track the bus state and control the clock with bit-level precision.

B. Insertion of a single bit

The precise clock control described so far can be used to insert a single bit on the bus. As described in the previous section, simply disabling the clock is not sufficient for the adversary to relinquish bus control. It must be ensured that the clock is disabled during a recessive transmission so that the adversary can continue its attack at a later time (recall that a recessive output does not influence the bus). Since the adversary only has clock control at the middle of a bit, the following steps are required to inject *a single dominant bit*, assuming the compromised ECU is currently paused at the recessive state: (1) enable clock a half-bit time before recessive-to-dominant edge, (2) wait one bit time to enter dominant bit, (3) wait another bit time to enter recessive bit, and (4) pause clock a half bit-time after dominant-to-recessive edge. Thus, the adversary must use such a pattern of bits within its payload, i.e. a dominant bit between two recessive bits.

However, this attack pattern introduces another unique challenge. As described earlier, the ECU reads bus state after each transition. Thus, if the adversary stops its attack during a dominant transmission by the victim, the compromised ECU will raise an error since it transmitted a recessive bit (a stopping requirement for the adversary) but observed a dominant transmission. This error will cause the attack ECU to reset its transmission so we must investigate methods to overcome this challenge as discussed below.

C. Causing an error on the victim

We now discuss how to exploit clock gating to induce just a single error on a victim. Our goal is to trick the victim into detecting an error in the transmission of its own frame, causing its transmit error counter to increase. To achieve this, the adversary must induce an error *after* the victim wins arbitration and becomes the sole transmitter. As detailed in Sec. II, a node transmitting on the bus simultaneously checks the bus for any bit errors. Thus, the adversary can simply overwrite a victim's recessive bit with a dominant bit using the steps outlined in the previous section, tricking the victim into thinking it caused the error. To successfully achieve this, there are two practical challenges that the adversary must consider: (1) it must account for the victim response, i.e. error flag transmission, and (2) it should identify bits in the victim frame that can be reliably targeted.

Victim's error response: When the adversary overwrites a victim's recessive bit with a dominant bit, the victim will detect a bit error and immediately transmit an error frame. Depending on the state of the victim's error counter, this error frame can be a series of six dominant or recessive bits. However, as outlined in Sec. V-B, an adversary cannot stop its attack during a victim's dominant transmission. Thus, an adversary cannot stop the attack if it expects the victim to transmit an active (dominant) error flag.

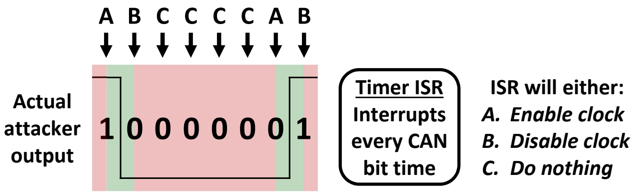


Fig. 7: Use timer ISR to convert a single logical-0 bit to a 6-bit error frame

To resolve this, the adversary can exploit their clock control to *expand* a single dominant bit into a series of six additional dominant bits, or an active-error flag. To generate an active-error flag from a single dominant bit, we perform four steps as depicted in Fig. 7:

1) With clock paused on a recessive bit, the adversary resumes clock for one bit time (or until the next timer interrupt).
2) After the recessive-to-dominant edge, the adversary pauses clock so the compromised ECU holds dominant state.
3) After five timer interrupts, the adversary resumes clock.
4) The compromised ECU's output transitions from dominant to recessive, and the adversary pauses the clock at the next interrupt and is ready for the next attack.

By simultaneously transmitting the flag as the victim transmits its flag, both flags will overlap, and the compromised ECU's transition from dominant to recessive will occur when the bus state is recessive due to the recessive end of the error flag. This approach enables the attacker to maintain an error-free transmit state on the compromised ECU so it may prepare for the next error injection. In scenarios where there are multiple nodes on the bus, the length of the error frame may be longer and thus the dominant duration by the attacker should be adjusted accordingly.

Targeting victim frames: A challenge we face is determining which bit to overwrite during a victim frame. Assuming that the adversary can determine the starting point of the victim's transmission, identifying the location of general recessive bits may be difficult due to variations in the payload and stuff bits. Recall that, during the paused clock state, an attacker has no source of feedback from the bus. Thus, we must identify some invariant about the victim's transmission for the adversary to exploit. We borrow an insight from prior work [15] to target the control fields, which often are static as data payloads do not change length. Alternatively, the adversary could analyze several frames prior to attack and target bits in the data payload that remain static. However, as the stuff bits can vary, it is preferable to use the initial control bits for attack.

D. *Shutting down victims with* CANnon

We now stitch together the components described above to transition a victim into the bus-off state. To achieve the shutdown attack against a specific victim ECU, the adversary must cause enough errors to forcibly transition the victim into a bus-off state. The goal here is to produce an attack that operates as fast as possible. For now, we assume that victim transmissions are periodic, which is often the case according to prior work [21], and thus the period can be used to estimate a victim ECU's SOF bit transmission time. As depicted in Fig. 8, the *CANnon* attack consists of two phases: a *loading* phase, where the attacker prepares the attack, and a *firing* phase, where the error frames are generated.

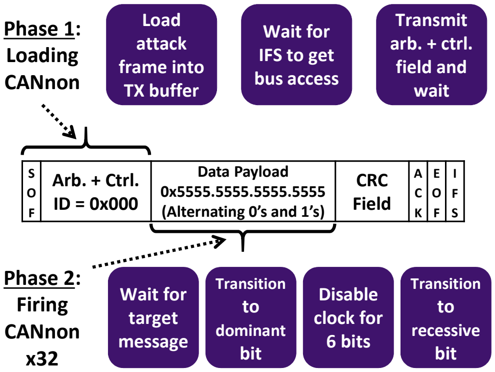

Phase 1: Loading CANnon
Load attack frame into TX buffer
Wait for IFS to get bus access
Transmit arb. + ctrl. field and wait
SOF
Arb. + Ctrl. ID = 0x000
Data Payload 0x5555.5555.5555.5555 (Alternating 0's and 1's)
CRC Field
ACK
EOF
IFS
Phase 2: Firing CANnon x32
Wait for target message
Transition to dominant bit
Disable clock for 6 bits
Transition to recessive bit

Fig. 8: Two-part approach to the *CANnon* attack

Loading the *CANnon*: To be able to transmit any arbitrary signal on the bus, the CAN controller must first win arbitration. Since the adversary only controls the software and is unable to modify the CAN controller, the compromised ECU's FSM should be in the same state (sole arbitration winner) before the adversary can attempt to transmit arbitrary bits. Thus, in the loading phase, the adversary aims to trick the compromised ECU into thinking that it is allowed to transmit on the bus as preparation for the firing phase. To do this, the adversary loads the attack frame (of a specially selected ID and payload) into the CAN peripheral's transmit buffer and waits for the compromised ECU to win the bus. In this attack, the adversary waits for completion of the arbitration phase and transmission of the control bits, before pausing the clock during the first payload bit, which can be designed to be recessive. At this point, the adversary is ready to start the *firing phase* of the attack while waiting for victim messages to appear on the bus.

To ensure a quick transition into the firing phase, the adversary can set the arbitration ID of the attack frame to 0x0, giving it highest priority on the bus. This ID ensures that the compromised ECU wins control of the bus as soon as the bus is idle. Then, to transition a victim into bus-off, the adversary needs to inject a dominant bit 32 times using a single attack frame. Thus, the data payload should be set to contain 64 bits of data with a pattern of alternating 0's and 1's, or a payload of 0x5555.5555.5555.5555. This payload gives the adversary 32 dominant bits to use for an attack and 32 recessive bits to temporarily pause the attack between victim transmissions. An additional benefit is that having a consistent pattern simplifies the logic for the adversary.

It should be noted that a different attack payload can still be utilized to achieve the same attack, albeit in a slightly sub-optimal manner. Any deviation from a payload of alternating dominant and recessive bits would require the attacker to reload another attack frame before shutting down the ECU.

Firing the *CANnon*: In the firing phase, the adversary utilizes the strategy described earlier to convert a single dominant bit into an active error flag, which will overwrite the recessive bits of the victim message. The adversary must wait for a victim message to appear on the network by waiting for its next periodic transmission that wins bus arbitration. The adversary then overwrites its selected recessive bit, causing the victim to detect an error and increment its error counter by 8. After detection of the error, the victim will immediately attempt to retransmit the failed frame. The adversary repeats this firing phase against 31 back-to-back victim retransmissions until the victim's error count reaches 256 (8x32) and enters the bus-off state. Thus, after the adversary causes an error in the first victim transmissions (using its period), targeting the retransmissions is significantly easier for the adversary.

E. Alternative CANnon *implementations*

Although the strategy described above is an efficient method to force the compromised CAN controller to transmit, we also describe alternative methods that achieve a shutdown attack using different parts of the CAN frame to highlight the flexibility an adversary has for the *CANnon* attack.

Firing with SOF bit: Instead of the above two-phase approach, imagine if the adversary could just skip to the firing phase. Our insight here is to use the SOF bit but with a different approach from Sec. IV. By stopping the clock right *before* a SOF transmission, the adversary can inject a dominant SOF bit during a victim's recessive bit. Since the SOF is only transmitted after a bus idle, the adversary can only transmit a SOF when it knows the bus is idle. Once bus idle is detected, the compromised CAN controller will load the registers to prepare for frame transmission. The adversary can pause the clock right when the transmit registers are loaded (experimentally, we find this to be two CAN bit times), effectively stopping the transmitter before it sends a SOF.

However, as the SOF is only a single bit, the error active flag from the victim will cause an error on the compromised ECU, forcing it to retransmit. Instead of avoiding this retransmission, the adversary can exploit it. The victim's error flag will cause the compromised ECU to think it simply lost arbitration. The adversary can then wait for a bus idle to occur and perform its attack again. Bus idle will not be observed until after the victim successfully retransmits so the adversary will need to target the periodic victim transmissions instead of its retransmissions from the loading/firing attack. While this attack is not as fast as the loading/firing attack, it does enable the *CANnon* attack on alternative MCU architectures as explained in Sec. VI.

Firing with ACKs: Instead of using data frame transmissions to attack a victim ECU, the adversary could exploit another scenario where the compromised ECU transmits a dominant bit: the acknowledgement (ACK) slot. To acknowledge a correctly received data frame, an ECU will set a dominant bit during the ACK slot of the data frame. Our idea here is the adversary could pause the compromised ECU right before it transmits the ACK bit for a victim's frame (the bit before the ACK slot is a recessive CRC delimiter bit). Suppose the CAN peripheral offers a SOF bit interrupt, which we observe in a number of automotive MCUs [30], [31]. If the adversary knows when the victim frame transmission starts and can determine when the CRC delimiter bit occurs in the frame, the adversary can pause the clock before the ACK slot and resume the clock just a few bit times later during the EOF, causing an error on the victim. The challenge here is that the adversary must precisely predict when an ACK will occur and the number of bits in the victim frame. Thus, victim frames that contain static or predictable data make an ideal target.

F. Practical challenges

We now discuss approaches to solving two practical challenges we encounter when launching *CANnon* against real vehicles in Sec. VI, one of which is a new capability resulting from the peripheral clock gating vulnerability.

Period deviations in victim frames: Up to now, we make the assumption that victim frame transmissions will be periodic. However, in practice, prior work [21] has found that period deviation is nonzero, which makes it difficult for the adversary to predict victim transmission times and thus perform the shutdown attack. Using insights from prior work [15], we could estimate when a victim message will initially appear on the bus. However, these insights relied on other messages in the network that would transmit immediately before the victim message, which is not always guaranteed. Likewise, even considering these circumstances, this approach has been found to not be reliable [21].

We introduce a new capability that permits an adversary to *guarantee* when a victim message appears on the CAN bus. We first revisit an observation made in Sec. IV during tests on real vehicles. When the compromised ECU holds a dominant state, all other ECUs will queue their frames waiting to transmit during bus idle. Upon releasing this dominant state, all transmitting ECUs will attempt to clear their queues. We find that these queued frames appear on the bus in a predefined order: by their arbitration ID. Our insight here is to determine which messages should arrive in a given range of time prior to launching our attack. By holding the dominant state for this range of time,[4] we can predict the ordering of messages and thus predict the start of the victim transmission.

Interruptions by higher-priority messages: Another practical challenge we encounter when launching *CANnon* against real vehicles is that higher-priority messages can interrupt the attack. If the adversary targets a victim frame with a low priority, we find that higher-priority messages can interrupt the repeated retransmissions by the victim. As the adversary expects the victim retransmissions to occur back-to-back, these interruptions can cause the attack to fail by causing collateral damage against unintended victims. Thus, the adversary could use prior work [21] to identify all source IDs of a victim ECU and simply select the highest-priority message, minimizing the chance of interruption by a higher-priority message. Additionally, prior work [21] also finds that safety-critical ECUs tend to transmit higher-priority frames so our adversary is already incentivized to target higher-priority frames.

VI. Evaluation

In this section, we demonstrate *CANnon* using two automotive MCUs found in modern vehicles and launch shutdown attacks against a variety of targets, including two real vehicles. We also detail experiments to highlight the reliability and stealth of *CANnon*.

A. Experimental setup

To demonstrate the significance of this attack, we launch *CANnon* from automotive MCUs used by modern vehicles, and we target real ECUs from two real vehicles. In this work, we do not explicitly show the ability to compromise an in-vehicle ECU remotely as this has been the focus of a large number of papers [3], [4], [26]–[28]. Rather, we build our attack on the assumption that these existing techniques would be successful in remotely compromising the software of automotive ECUs.

One of the key factors that enabled the discovery of this vulnerability was our choice of experimental setup, which is likely why, to the best of our knowledge, this incidence has not been studied before. In this work, we initially used the Arduino Due board, which closely resembles the capabilities of modern automotive MCUs. However, if we look at prior work in the field [6], [10], [11], [15]–[17], [32], [33], we find widespread use of the legacy design of automotive ECUs, namely an Arduino Uno board with a standalone controller. Thus, as a result of *our choice of experimental setup*, none of these prior works could have identified the *CANnon* vulnerability; where the industry moved to a modern design, prior research has continued to use the legacy design.

Automotive MCUs: In addition to the Arduino, we test *CANnon* on evaluation boards for two automotive MCUs from Microchip and STMicro (commonly known as ST). These boards will serve as the compromised in-vehicle ECUs as they are used in modern production vehicles. In fact, STMicro is one of the top five semiconductor suppliers for the automotive industry [34], and its MCUs are likely to be in many modern vehicles. The features and architectures they offer are likely generalizable to other automotive MCUs as they are both marketed as high-performance networking MCUs, which are two key features we identify in Sec. IV. While we do not evaluate boards from every MCU supplier, we find multiple references to software APIs for peripheral clock gating in reference manuals and market reports [35]–[39].

Specifically, we evaluate: (1) the Microchip SAM V71 Xplained Ultra board, which uses an ATSAMV71Q21 32-bit MCU operating at 150 MHz and is designed for in-vehicle infotainment connectivity [31], [40], and (2) the STMicro SPC58EC Discovery board, which uses an SPC58EC80E5 32-bit MCU operating at 180MHz and is designed for automotive general-purpose applications [41], [42]. It is likely that other MCUs in the same family (i.e. SAM V MCUs and ST SPC5 MCUs) share the same peripheral clock gating vulnerability as demonstrated by similarities within an MCU family's reference manuals [30], [31]. Consequently, the Arduino Due board identified in Sec. IV uses an AT91SAM3X8EA 32-bit MCU operating at 84 MHz from an older series of the same SAM family [30].

For the SPC58 MCU, we encountered a challenge in finding a clock enable/disable function. All clock functions *requested* the peripheral to essentially give the MCU permission to disable the peripheral's clock. Upon receiving the request, the peripheral waits for all transmission operations to stop before the MCU can disable its clock. However, we found an alternative approach to directly control the clock on the SPC58 that bypasses this request procedure. In fact, this alternative contradicts the expected implementation as described in the SPC58's reference manual [42]. The SPC58 utilizes operating modes that change the configurations for the MCU. In particular, we focus on two modes: the DRUN mode, which permits all peripherals to run normally, and the SAFE mode, which stops the operation of all active peripherals. We find that a transition to DRUN is equivalent to enabling the CAN peripheral's clock and a transition to SAFE effectively disables the peripheral's clock without permission from the peripheral itself. A limitation introduced here is that a clock enable could not occur soon after a clock disable[5], but the SOF-based attack from Sec. V successfully works for the SPC58.

Real vehicle testbed: Additionally, we demonstrate *CANnon* against two real vehicles: a 2009 Toyota Prius and a 2017 Ford Focus. We connect to the CAN bus by accessing the bus via the vehicle's On-Board Diagnostics (OBD) port to emulate a remotely-compromised ECU. We also identify the mapping of arbitration IDs to source ECUs using details from prior work [21]. We only launch the *CANnon* attack against vehicles while they are parked to avoid any potential safety concerns. Note that these vehicles have their engine running with all ECUs actively transmitting onto the network. As detailed in prior work [21], vehicles tend to transmit mostly periodic

[4] Where prior work required injecting a message to guarantee transmission time of a victim, we can simply disrupt the bus to "simulate" an injected message.

[5] The transition to SAFE mode (or effectively disabling the clock) includes several processes that must complete for safety reasons before the peripheral clocks can be enabled.

messages, and we find that these transmissions start when the engine is started. Even if the vehicle is taken on a drive, only the data payloads change rather than periodic transmission rates. We also launch *CANnon* against an Arduino Due, a PeakCAN USB device, and a 2012 Ford Focus powertrain ECU in a variety of synthetic network setups.

B. CANnon *against real vehicles*

Basic disruption on real vehicles: We launch the basic disruption attack against both real vehicles using the SAM V71 and SPC58 evaluation boards. As discussed in Sec. IV, ECUs often implement a time-out feature that prevents a CAN transceiver from holding the dominant state for an extended period of time. We experimentally find that we can maintain up to 1ms of dominant state on the bus with at least 4μs of recessive in-between on both vehicles. We find that this attack prevents ECUs from communicating on the bus and will trigger malfunction lights on the instrument panel and even diagnostic codes that indicate loss of ECU communication.

Powertrain ECU shutdown in 2017 Focus: We demonstrate a shutdown attack with the V71 MCU using the loading/firing attack in Sec. V. The powertrain ECU transmits several arbitration IDs, but we select the highest-priority ID using methods from prior work [21]. In our pre-analysis of the victim transmission times, we find that a majority of the powertrain ECU's IDs will transmit back-to-back. With our technique for guaranteeing transmission times in Sec. V, we hold the dominant bit when we expect the victim to appear (for approximately 50μs). Upon release of the dominant bit, the target victim frame will be the first frame to transmit and, thus, we launch our firing phase on that frame. We target the control field and perform this attack 32 times, allowing us to shut down the powertrain ECU in about 2ms. Although the powertrain ECU does auto-recover, the ability to shut down the ECU quickly demonstrates the speed of our attack.

Power steering ECU shutdown in 2009 Prius: We demonstrate a shutdown attack with the SPC58 MCU using the SOF-based attack in Sec. V as the SPC58 cannot enable the clock immediately after disabling it. The target victim is a power steering ECU that transmits three IDs: 0x262, 0x4C8, and 0x521. We choose the ID with the smallest period (0x262 with period of 20ms) and find that its period deviation is quite small using methods from prior work [21]. As the SOF approach requires a successful transmission between each attack, this shutdown is significantly longer since we do not target retransmissions. We shut down the power steering ECU after 700ms, and we find that it remains permanently offline.

C. Attack reliability

One important aspect of a reliable attack is repeatability. We envision an adversary who purchases the same MCU that the compromised ECU uses as preparation for their remote exploit and shutdown attack. After tuning attack parameters to the specific MCU (e.g. number of MCU cycles prior to SOF transmission), the adversary hopes that the tuned parameters will be similar to that of the real victim MCU. We find that properly tuned attack code across multiple copies of our test MCUs over a few months could repeatedly produce the same output to the bus. We attribute this success to the strict specifications that ECU hardware must follow in the manufacturing stage.

We now compare the reliability of *CANnon* using the hardware timer interrupt centered on each CAN bit versus manually counting MCU clock cycles. In this experiment, we use the Microchip and Arduino Due boards to transmit active error frames at repeated intervals. We transmit these frames against an Arduino Due victim that sends its frames every 10ms with zero deviation in the period. Using a hardware timer to launch our attack, we find that both the Microchip and Arduino Due boards can shut down the victim 100% of the time. However, if we try to perform the active frame transmissions by manually keeping count of MCU clock cycles, we only achieve the attack 10% of the time due to variations discussed in Sec. V.

We also compare the reliability to guarantee victim transmission time versus prior work [15] that overwrites messages using injected messages to predict victim transmission. Here, we use the Arduino Due board to target three different victims: (1) another Arduino Due, (2) a PeakCAN device, and (3) a 2012 Ford Focus powertrain ECU. Using our method, we can achieve a shutdown of all three victims using all three of our MCUs 100% of the time. However, using prior work to perform the message overwrite attack, we only succeed for the Arduino Due and PeakCAN device. On the powertrain ECU, we cannot achieve even a single success as its transmissions exhibit significant period deviation.

D. Stealth analysis

We now compare the stealth of *CANnon* versus the state-of-the-art message overwrite attack [15]. We construct three simple detection methods at each layer of the CAN stack based on existing defenses.[6] The goal of either shutdown attacker is to achieve a victim shutdown without the detection method alerting *prior* to the shutdown itself. Our experimental setup involves three Arduino Due boards: (1) the victim ECU, (2) the detection ECU, and (3) the compromised ECU. The detection ECU also transmits its own messages to simulate other traffic on the bus. We perform each test 1,000 times, and we operate all tests at 500Mbps, use a shared 12V supply to power the boards, and observe the bus traffic using a logic analyzer.

For all of the experiments below, we follow the configuration of prior work [15]: the victim ECU transmits ID 0x11 every 10ms, the detection ECU transmits ID 0x7 and 0x9 every 10ms, and the compromised ECU monitors the bus and attempts to attack. To simulate noise from a real vehicle, we intentionally set the deviation of ID 0x11 to 0.15ms as the best-case minimum deviation found by prior work [21]. For all experiments with the overwrite attack, the compromised

[6] We do not demonstrate *CANnon* against complete implementations of existing defenses, which monitor only entire CAN messages or frames, as they are ineffective by construction as detailed in Sec. VII.

ECU injects ID 0x9 around the expected transmission time of 0x11 to set up their attack.

Versus timing-based IDS: We first test the overwrite attack and *CANnon* against a timing-based IDS that alerts if frames transmit outside of their expected period. Timing-based IDSes also include ML-based [43] and traffic anomaly methods [44] as they analyze timestamps to detect illegitimate transmissions. We set the detection threshold for period deviation to be 10% (e.g. 1ms for a 10ms period) following prior work [9]. We program our detection ECU to measure the inter-arrival time between frames for a given ID and alert if the measured time exceeds 10% of the expected period. For *CANnon*, the compromised ECU attacks using the data payload and employs the *dominant-hold* technique from Sec. V to guarantee victim transmission time. Out of 1,000 attempts, we find that our detection ECU alerts to *every* attempt by the overwrite attack but does not alert to any of the *CANnon* attacks. *CANnon* only needs to hold the dominant state for 0.15ms *once* to guarantee the first victim transmission and cause an error. The overwrite attack injects new messages onto the network, exceeding the expected deviation threshold. *CANnon* achieves a shutdown in just 2ms before the next transmission should occur.[7]

Versus a "secure transceiver:" As secure transceivers are not currently in production, we modify the detection ECU to act as the secure transceiver. It will read each logical bit transmitted and received by the compromised ECU by directly connecting between the MCU's CAN peripheral and the CAN transceiver following prior work [21]. If an ECU sends an illegitimate arbitration ID, it will produce an alert in real-time immediately after the arbitration field transmits. For *CANnon*, the compromised ECU attacks via the SOF bit method as the secure transceiver could detect the data payload attack.[8] Out of 1,000 attempts, we find that our secure transceiver alerts to *every* attempt by the overwrite attack but does not alert to any of the *CANnon* attacks. *CANnon* only injects a SOF bit as its attack and does not transmit any arbitration ID, while the additional message transmissions in the overwrite attack cause our secure transceiver to alert immediately.

Versus a frame-level voltage IDS: Following observations from prior work [10], [11], [45], we modify the detection ECU to directly measure the CAN bus voltages to detect an attack. The CAN medium is a differential pair called CAN low and CAN high that typically exhibit around 1.5 and 3.5 voltages for a dominant bit, respectively (recessive state causes both to exhibit 2.5 volts). The key insight from prior work is to measure the voltage of the dominant bits throughout an entire frame. With the message overwrite attack [15], the start of an overwritten frame has two transmitters and ends with a single transmitter, i.e. the compromised ECU. Thus, the attack exhibits a larger differential at the start and a smaller differential at the end of an overwritten frame. We build a voltage IDS that alerts if the dominant bits exhibit a sudden drop in dominant differential voltage during a single frame. Out of 1,000 attempts, we find that our IDS alerts to *every* attempt by the overwrite attack but does not alert to any of the *CANnon* attacks. *CANnon* only injects a single error flag in the *middle* of a frame and, thus, this approach to voltage IDS does not detect our attack.

VII. Stealth Against Network Defenses

Next, we discuss why *CANnon* evades state-of-art defenses and also how to tackle future *CANnon*-aware defenses.

A. Deceiving state-of-the-art defenses

Many approaches exist that can defend against shutdown attacks. We group these defenses into three classes based on the layer in the CAN communication stack they operate on.

Defenses at application layer: Many IDSes are software applications, limited to reading messages passed up the communication stack by CAN hardware. These run on any ECU and do not require special hardware, making them an attractive solution. For instance, they can use statistical techniques based on message timings and content [9], [16], [17], [46]. A recent U.S. agency report [12] discusses how companies working closely with automakers have access to proprietary information on the expected content of CAN messages, enhancing their ability to create application-layer defenses. Another class of IDS that makes use of this proprietary information are machine learning [43] and traffic anomaly IDSes [44], which analyze message timing and payload to detect an attack.

Application-layer IDSes can detect both the diagnostic message and message overwrite attacks in Table I as they require transmitting additional CAN frames on the bus. As such, any application-layer defenses that measure message timing or content cannot detect our attack since we do not transmit entire CAN frames or significantly disrupt valid transmitted frames. *CANnon* operates quickly and can shutdown ECUs in just a couple milliseconds (well under the minimum period observed by prior work [21]) as demonstrated in Sec. VI.

Defenses at data link layer: Recent industry solutions propose secure CAN transceivers [13] that operate at the data link layer. These transceivers can *prevent* a compromised ECU from attacking a CAN bus by either: (1) invalidating frames with spoofed CAN IDs, (2) invalidating frames that are overwritten by a compromised ECU, and (3) preventing attacks that flood the bus with frame transmissions. Attacks that require physical access are outside their scope.

These transceivers are a promising approach to defending against the diagnostic message and message overwrite attacks in Table I as the transceivers would destroy any illegitimate frames based on their IDs. As the loading phase in our loading/firing attack transmits a specific arbitration ID (0x0), these transceivers would also detect an illegitimate ID from the compromised ECU and raise an alarm. However, the two attack alternatives (SOF and ACK attacks) do not produce an arbitration ID and could not be detected by pure ID-based filtering concepts as demonstrated in Sec. VI.

[7] This fast ability to shutdown could act as a useful stepping-stone to future work on masquerade attacks.

[8] *CANnon* could technically use any arbitration ID (even a legitimate ID), but we assume that the adversary wants to use ID 0x0 to minimize wait for bus idle.

Defenses at physical layer: Another approach for IDSes is to directly access the physical layer, e.g. measuring bus voltages. These defenses detect sudden changes over a series of CAN frames (or even a single frame) by creating a profile of the expected voltages [10], [11], [45]. These works find that each ECU applies a unique voltage that is measurable across an entire CAN frame. If an illegitimate transmitter attempts to spoof a victim's message, the voltage measured across the frame could identify a potential attack.

This approach can detect the message overwrite attack because a single frame will start with two simultaneous transmitters followed by only the overwriting compromised ECU; a distinctive change in voltage for the rest of the frame indicates an attack. However, in regard to physical-layer defenses that measure voltage, *CANnon* does not require overwriting a frame from the SOF onwards and, thus, prior work [45] would not detect a sudden change in the voltage from the start of a single data frame as demonstrated in Sec. VI.

B. Deceiving CANnon*-aware defenses*

We now discuss how *CANnon* could remain stealthy against even future *CANnon*-aware defenses. We discuss defenses that might seem appealing at a glance, but we will show that this attack will likely require architectural countermeasures discussed in Sec. VIII.

Tracking error interrupts at application layer: Up to now, we have discussed how application-layer defenses that only monitor messages do not detect *CANnon*. However, there is another source of signals from the lower CAN stack layers: error interrupts. We envision a *CANnon*-aware defense that uses these interrupts to identify potentially malicious sources of error. This defense tracks errors based on their frequency and for which messages they occur during in an attempt to find a pattern representative of a shutdown attack. Existing work [32] can detect when a shutdown occurs by tracking error flags, but it cannot determine if the errors were caused maliciously or by legitimate bus faults. We now discuss a couple modifications that similar work could implement to detect a malicious attack. We also discuss how our adversary can thwart those efforts by making it challenging for defenses to detect *CANnon* while maintaining a low false positive rate:

1) *Tracking number of errors per ID:* One potential defense is to track the number of errors that occur when a particular message ID is transmitted. However, our adversary could use prior work [21] to identify all source IDs from an ECU by simply monitoring the bus and tracking message timestamps. Our adversary could then target all IDs from a victim ECU, making an error seem consistent across all transmissions and difficult to differentiate from a legitimate fault.
2) *Checking for multiple errors in short time:* Another defense is to check for multiple errors in a short amount of time, which is an invariant of prior work [15]. While the loading/firing attack causes multiple errors in a matter of milliseconds, an adversary can extend this attack over a longer period of time. An active error flag will increment the victim error counter by eight; to recover from an error, a successful transmission from a victim will decrement the error counter by one. Our adversary could launch an error flag for one of every seven successful transmissions from a victim, giving us an effective increase of one for the transmit error count. By repeating this attack 256 times, the adversary could allow up to 1792 successful transmissions by a victim and still succeed in a shutdown.

VIII. Countermeasures

As illustrated, *CANnon*-based attacks are stealthy against existing security methods. Here, we describe some directions for potential countermeasures. Since the attack relies on two broad ideas, namely clock control and error exploitation, the countermeasures described can be seen to prevent one of these problems, i.e. prevent clock control or detect specific error patterns or error transmitter patterns.

Detecting bit-wise voltage spikes: Overwriting a message causes a sudden voltage change in the dominant bit. Thus, one approach to detect such an attack is tracking per-bit voltages at the physical layer. Changes in the middle of message transmissions could indicate potential adversary activity. However, since random faults or genuine errors flags can cause similar behaviour, such a method would require additional identification of patterns in the voltage changes, e.g. behaviour periodicity. Some recent work that uses transition characteristics for network fingerprinting [33] could be modified in this direction.

Forced clear of transmit buffers: As observed in Sec. III, the ability to resume a message transmission is a key factor for successfully exploiting the controller. Thus, the attack can simply be prevented by disabling such behavior, i.e. resetting/clearing all buffers upon clock gating. Such a countermeasure allows the flexibility of being deployed at either the hardware or software level. If hardware changes are permitted, this approach can be achieved by designing reset logic based on the clock signal. In software, this approach can be achieved by flushing the peripheral transmit buffers upon clock stop. A modification of this idea for safety is present in SPC58, whereby a clock stop request is completed based on the feedback from the CAN peripheral.

On-chip power analysis: The automotive industry takes another approach to protecting their ECUs from remote adversaries: host-based IDSes [12]. One host-based detection method for *CANnon* could be a separate secure chip that monitors the power usage of the MCU. Since disabling the peripheral clock induces a drop in power, a host-based IDS could detect potentially malicious actions. This approach should operate outside of the MCU and could include logic to identify when power drops are *not* expected, e.g. while in motion, while vehicle not asleep, etc.

Removal of CAN peripheral clock gating: The main feature that enables *CANnon* in modern MCUs is peripheral clock gating. Rather than offering a peripheral for CAN, modern MCUs could simply utilize a separate always-on clock domain for the

CAN peripheral or require standalone CAN controllers, which receive a clock signal from a separate oscillator. Assuming the other peripherals do not share this vulnerability, they could remain unchanged by removing clock gating for just CAN.

IX. Other Related Work

Side-channel attacks and fault attacks: *CANnon* has some similarity to fault attacks on cryptographic algorithm implementations available in secure processors, which can completely break the security of secure systems [47]–[49]. Fault attacks [47], [48] are a subset of side-channel attacks, which disrupt normal execution of a (software or hardware) process to compromise a system. Fault attacks typically require physical access to the device under attack to successfully induce a fault. To our knowledge, the RowHammer attack [50] is the only other attack that is able to successfully produce a *remote* fault. In contrast, *CANnon* remotely induces "faults" through software-only access to the peripheral clock API of unaltered automotive MCUs.

Security API attacks: Attacks on secure embedded processor APIs were first discovered in prior work [51] (see other work [52] for an up-to-date survey). The idea behind these attacks was "to present valid commands to the security processor, but in an unexpected sequence" in such a way that "it is possible to obtain results that break the security policy envisioned by its designer" [51]. Although similar in aim, our work is fundamentally different as *CANnon* takes advantage of low-level clock-control APIs in current automotive MCUs that are used to save energy and improve performance (not input secure key material). *CANnon* does not target a secure processor either, and it focuses in subverting an interface not externally available to a human subject as in other work [51] (i.e. in *CANnon*, an attacker must first compromise the MCU and gain control of its software).

X. Discussion

In this work, we introduced *CANnon*, a novel attack method to exploit the benign clock gating mechanism in automotive MCUs for exerting arbitrary control over the CAN bus. We illustrated several methods to achieve precise clock control and use it to cause remote shutdown attacks. Despite focusing on a single example class here, we envisage that such a methodology can be used for other powerful attacks. With the increasing software code base for a modern vehicle, it can be expected that there exist exploitable vulnerabilities in connected ECUs. *CANnon*-based techniques allow the compromise of a single ECU to influence all ECUs on the bus in a stealthy manner. This reason makes the existence of *CANnon*-relevant interfaces very dangerous.

In this work, we illustrated the attack capability on two lines of automotive-grade MCUs, and we believe that several other lines from independent manufacturers can be susceptible to this attack. We would strongly encourage the research community to identify similar gaps in other processors. Since this attack exploits a fundamental architectural feature, changes to mitigate such a class of attacks poses an interesting problem. We illustrated some directions for such changes here. However, designing specific modifications to future security systems would require further investigation.

CANnon not only enables new attack methodologies, but it can also be viewed as a capability that can be integrated into existing software systems for testing. Enabling such investigations is one of our key motivations for making the tool widely available [20]. We illustrate a few future directions below.

Expanding existing tools: Recent work [21] demonstrates a network mapper for the CAN bus. This approach requires the tool to operate from customized hardware rather than an in-vehicle ECU. By using *CANnon* to target and shut down ECUs for destination mapping, network mapping could run on existing ECUs without modification. Further, the *CANnon* method could be utilized by a genuine node, e.g. an Intrusion Prevention System (IPS), to remove malicious messages from the bus. Prior to *CANnon*, such IPS capabilities typically require hardware changes.

Clock control for other peripherals: Future work could investigate the impact of the *CANnon*-like vulnerability for other peripherals. It is possible that other bus protocols, including transport layer protocols that use CAN for the data link layer (e.g. GMLAN, MilCAN, UAVCAN, etc.), are likely vulnerable to a network participant that maliciously holds the state of the bus. For example, the Local Interconnect Network (LIN) bus implements the same logical bus states as the CAN bus and is likely vulnerable to the basic remote disruption attack.

Non-standard CAN: Automakers are starting to implement extended CAN and CAN-FD protocols. These protocols rely on the same principles as standard CAN and thus are vulnerable to *CANnon*. Future work could investigate unique implications related to these other CAN implementations (e.g. perhaps the higher bit rate for the data payload in CAN-FD could enable unique derivations of the *CANnon* attack).

Acknowledgments

This work was funded in part by the PITAXVIII PITA award and the CNS-1564009 NSF IoT award. We thank the anonymous reviewers for their helpful suggestions.

References

[1] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*, vol. 4. San Francisco, 2011, pp. 447–462, http://static.usenix.org/events/sec11/tech/full_papers/Checkoway.pdf.

[2] 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 *2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 447–462, https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5504804.

[3] S. Nie, L. Liu, and Y. Du, "Free-fall: hacking tesla from wireless to can bus," *Briefing, Black Hat USA*, pp. 1–16, 2017, https://paper.seebug.org/papers/Security%20Conf/Blackhat/2017_us/us-17-Nie-Free-Fall-Hacking-Tesla-From-Wireless-To-CAN-Bus-wp.pdf.

[4] C. Miller and C. Valasek, “Remote exploitation of an unaltered passenger vehicle,” *Black Hat USA*, vol. 2015, p. 91, 2015, http://illmatics.com/Remote%20Car%20Hacking.pdf.

[5] P.-S. Murvay and B. Groza, “Dos attacks on controller area networks by fault injections from the software layer,” in *Proceedings of the 12th International Conference on Availability, Reliability and Security*. ACM, 2017, p. 71, http://www.aut.upt.ro/~pal-stefan.murvay/papers/dos-attacks-controller-area-networks-fault-injections-from-software-layer.pdf.

[6] A. Palanca, E. Evenchick, F. Maggi, and S. Zanero, “A stealth, selective, link-layer denial-of-service attack against automotive networks,” in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2017, pp. 185–206, https://www.politesi.polimi.it/bitstream/10589/126393/1/tesi_palanca.pdf.

[7] A. Van Herrewege, D. Singelee, and I. Verbauwhede, “CANAuth - A Simple, Backward Compatible Broadcast Authentication Protocol for CAN bus,” in *ECRYPTWorkshop on Lightweight Cryptography 2011*, 2011.

[8] B. Groza, P. Murvay, A. V. Herrewege, and I. Verbauwhede, “Libra-can: A lightweight broadcast authentication protocol for controller area networks,” in *Cryptology and Network Security, 11th International Conference, CANS 2012*, J. Pieprzyk, A. Sadeghi, and M. Manulis, Eds., vol. 7712. Springer, December 12-14, 2012, pp. 185–200.

[9] H. M. Song, H. R. Kim, and H. K. Kim, “Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network,” in *2016 international conference on information networking (ICOIN)*. IEEE, 2016, pp. 63–68, https://ieeexplore.ieee.org/iel7/7422341/7427058/07427089.pdf.

[10] K.-T. Cho and K. G. Shin, “Viden: Attacker identification on in-vehicle networks,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1109–1123, https://arxiv.org/pdf/1708.08414.

[11] W. Choi, K. Joo, H. J. Jo, M. C. Park, and D. H. Lee, “Voltageids: Low-level communication characteristics for automotive intrusion detection system,” *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 8, pp. 2114–2129, 2018, https://ieeexplore.ieee.org/iel7/10206/4358835/08306904.pdf.

[12] An assessment method for automotive intrusion detection system performance. https://rosap.ntl.bts.gov/view/dot/41006.

[13] B. Elend and T. Adamson, “Cyber security enhancing can transceivers,” in *Proceedings of the 16th International CAN Conference*, 2017.

[14] J. Wilson and T. Lieu, “Security and privacy in your car study act of 2017 — H. R. 701,” 2017, available at https://www.congress.gov/115/bills/hr701/BILLS-115hr701ih.pdf.

[15] K.-T. Cho and K. G. Shin, “Error handling of in-vehicle networks makes them vulnerable,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1044–1055, https://rtcl.eecs.umich.edu/wordpress/wp-content/uploads/ktcho_busoff_CCS_16.pdf.

[16] ——, “Fingerprinting electronic control units for vehicle intrusion detection,” in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 911–927, https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_cho.pdf.

[17] S. U. Sagong, X. Ying, A. Clark, L. Bushnell, and R. Poovendran, “Cloaking the clock: emulating clock skew in controller area networks,” in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*. IEEE Press, 2018, pp. 32–42, https://ieeexplore.ieee.org/iel7/8429083/8443707/08443719.pdf.

[18] C.-W. Lin and A. Sangiovanni-Vincentelli, “Cyber-security for the controller area network (can) communication protocol,” in *2012 International Conference on Cyber Security*. IEEE, 2012, pp. 1–7.

[19] C. Smith, *The Car Hacker’s Handbook: A Guide for the Penetration Tester*. No Starch Press, 2016, http://opengarages.org/handbook/.

[20] Cannon. https://github.com/sksecurity/cannon.

[21] S. Kulandaivel, T. Goyal, A. K. Agrawal, and V. Sekar, “Canvas: Fast and inexpensive automotive network mapping,” in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 389–405, https://www.usenix.org/system/files/sec19-kulandaivel.pdf.

[22] T. Ring, “Connected cars–the next targe tfor hackers,” *Network Security*, vol. 2015, no. 11, pp. 11–16, 2015.

[23] Gartner says by 2020, a quarter billion connected vehicles will enable new in-vehicle services and automated driving capabilities. https://www.gartner.com/en/newsroom/press-releases/2015-01-26-gartner-says-by-2020-a-quarter-billion-connected-vehicles-will-enable-new-in-vehicle-services-and-automated-driving-capabilities.

[24] The car in the age of connectivity: Enabling car to cloud connectivity. https://spectrum.ieee.org/telecom/wireless/the-car-in-the-age-of-connectivity-enabling-car-to-cloud-connectivity.

[25] C. Miller and C. Valasek, “A survey of remote automotive attack surfaces,” *black hat USA*, vol. 2014, p. 94, 2014, http://illmatics.com/remote%20attack%20surfaces.pdf.

[26] Experimental security research of tesla autopilot. https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Research_of_Tesla_Autopilot.pdf.

[27] Car hacking research: Remote attack tesla motors. https://keenlab.tencent.com/en/2016/09/19/Keen-Security-Lab-of-Tencent-Car-Hacking-Research-Remote-Attack-to-Tesla-Cars/.

[28] New car hacking research: 2017, remote attack tesla motors again. https://keenlab.tencent.com/en/2017/07/27/New-Car-Hacking-Research-2017-Remote-Attack-Tesla-Motors-Again/.

[29] D. Wise, “Vehicle cybersecurity dot and industry have efforts under way, but dot needs to define its role in responding to a real-world attack,” *Gao Reports. US Government Accountability Office*, 2016.

[30] Microchip sam 3x family of mcus. http://ww1.microchip.com/downloads/en/devicedoc/atmel-11057-32-bit-cortex-m3-microcontroller-sam3x-sam3a_datasheet.pdf.

[31] Microchip sam v family of automotive mcus. http://ww1.microchip.com/downloads/en/DeviceDoc/SAM-E70-S70-V70-V71-Family-Data-Sheet-DS60001527D.pdf.

[32] S. Longari, M. Penco, M. Carminati, and S. Zanero, “Copycan: An error-handling protocol based intrusion detection system for controller area network,” in *ACM Workshop on Cyber-Physical Systems Security & Privacy (CPS-SPC’19)*, 2019, pp. 1–12, https://re.public.polimi.it/retrieve/handle/11311/1104918/427927/CopyCAN.pdf.

[33] M. Kneib, O. Schell, and C. Huth, “Easi: Edge-based sender identification on resource-constrained platforms for automotive networks,” https://dl.acm.org/doi/pdf/10.1145/3338499.3357362.

[34] Automotive semiconductor market - growth, trends, and forecast (2020 - 2025). https://www.mordorintelligence.com/industry-reports/automotive-semiconductor-market.

[35] Nxp mcus. https://www.nxp.com/docs/en/application-note/AN4240.pdf.

[36] Renesas mcus. https://www.renesas.com/us/en/products/synergy/hardware/microcontrollers/glossary.html.

[37] Fujitsu mcus. https://www.fujitsu.com/downloads/EDG/binary/pdf/find/25-5e/5.pdf.

[38] Cypress mcus. https://www.cypress.com/products/fm4-32-bit-arm-cortex-m4-microcontroller-mcu-families.

[39] Infineon mcus. https://www.infineon.com/dgdl/Infineon-TC1767-DS-v01_04-en.pdf?fileId=db3a30431be39b97011bff8570697bdb.

[40] Sam v71 xplained ultra evaluation kit. https://www.microchip.com/DevelopmentTools/ProductDetails/PartNO/ATSAMV71-XULT.

[41] Spc58ec-disp discovery board. https://www.st.com/en/evaluation-tools/spc58ec-disp.html?ecmp=tt12221_gl_social_jul2019.

[42] St spc5 family of automotive mcus. https://www.st.com/en/automotive-microcontrollers/spc5-32-bit-automotive-mcus.html.

[43] K. Zhu, Z. Chen, Y. Peng, and L. Zhang, “Mobile edge assisted literal multi-dimensional anomaly detection of in-vehicle network using lstm,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4275–4284, 2019.

[44] M. Russo, M. Labonne, A. Olivereau, and M. Rmayti, “Anomaly detection in vehicle-to-infrastructure communications,” in *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*. IEEE, 2018, pp. 1–6.

[45] M. Foruhandeh, Y. Man, R. Gerdes, M. Li, and T. Chantem, “Simple: Single-frame based physical layer identification for intrusion detection and prevention on in-vehicle networks,” 2019, http://u.arizona.edu/~yman/papers/simple_acsac19.pdf.

[46] C. Young, H. Olufowobi, G. Bloom, and J. Zambreno, “Automotive intrusion detection based on constant can message frequencies across vehicle driving modes,” in *Proceedings of the ACM Workshop on Automotive Cybersecurity*. ACM, 2019, pp. 9–14, https://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=1066&context=ece_conf.

[47] D. Boneh, R. A. DeMillo, and R. J. Lipton, “On the importance of checking cryptographic protocols for faults (extended abstract),” in *Advances in Cryptology - EUROCRYPT ’97*, ser. LNCS, W. Fumy, Ed., vol. 1233. Springer, May 11-15, 1997, pp. 37–51.

[48] E. Biham and A. Shamir, “Differential fault analysis of secret key cryptosystems,” in *Advances in Cryptology - CRYPTO ’97*, ser. LNCS, B. S. K. Jr., Ed., vol. 1294. Springer, August 17-21, 1997, pp. 513–525.

[49] S. P. Skorobogatov and R. J. Anderson, “Optical fault induction attacks,” in *Cryptographic Hardware and Embedded Systems - CHES 2002*, ser. LNCS, B. S. K. Jr., Ç. K. Koç, and C. Paar, Eds., vol. 2523. Springer, August 13-15, 2002, pp. 2–12.

[50] Y. Kim, R. Daly, J. Kim, C. Fallin, J. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, “Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors,” in *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014*. IEEE Computer Society, June 14-18, 2014, pp. 361–372.

[51] M. Bond and R. J. Anderson, “Api-level attacks on embedded systems,” *IEEE Computer*, vol. 34, no. 10, pp. 67–75, 2001.

[52] R. J. Anderson, *Security engineering - a guide to building dependable distributed systems (2. ed.).* Wiley, 2008.