From Attack to Defense: Toward Secure In-vehicle Networks

by

Kyong Tak Cho

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2018

Kyong Tak Cho

ktcho@umich.edu

ORCID iD: 0000-0002-0830-0361

To my wife, sister, father, and mother

ACKNOWLEDGEMENTS

I would like to respectfully express my sincere gratitude to my advisor, Professor Kang G. Shin, for his support, patient guidance, and unending inspiration over the past five years. During my doctoral study, he always guided me to the right direction in which I can grow professionally and personally. He kept patience in me so that I can rise on my own feet as an independent researcher. Most of all, he was an outstanding mentor and friend whom I could always ask for advices on different aspects of life.

I would also like to thank Professors J. Alex Halderman, Peter Honeyman, and Florian Schaub for serving on my doctoral committee and giving me constructive comments and suggestions.

I thank my labmates in the Real-Time Computing Laboratory (RTCL) for their camaraderie and discussion. In particular, I am thankful to Dongyao Chen. He has been a great friend whom I worked with on many new projects and also enjoyed talking with on career paths and life in general. A word of gratitude also goes to other past and current members of RTCL: Duc Bui, Chun-Yu Chen, Seunghyun Choi, Hoon Sung Chwa, Kassem Fawaz, Huan Feng, Arun Ganesan, Juncheng Gu, Suining He, Sunmin Kim, Youngmoon Lee, Taeju Park, Mert Pese, Tim Trippel, Yu-chih Tung, Chaocan Xiang, Xiufeng Xie, and Hamed Yousefi. I also thank my collaborators and former mentors: Li Zhao, Manoj Sastry, and Lily Yang at Intel Labs, and Yu-seung Kim at Ford Research & Innovation Center.

Most importantly, I would like to express my deep gratitude to my wife, sister, father, and mother. My wife, Soohyun Hwang, has been a friend and supporter, always giving me indescribable encouragement and happiness throughout this Ph.D. journey. She went through all the ups and downs with me and without her unwavering love and support, I could not have finished this work. No words can express my gratitude towards my father Heekeun Cho, my mother Soonmi Choi, and my sister Haram Cho. They have constantly provided me endless love, encouragement, and support over the years. This dissertation is dedicated to these people.

Finally, I would like to thankfully acknowledge the support of the US National Science Foundation under Grants CNS-1329702, CNS-1446117, CNS-1505785, and CNS-1646130, and the DGIST Global Research Laboratory Program through NRF funded by MSIP of Korea.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

New security breaches in vehicles are emerging due to software-driven Electronic Control Units (ECUs) and wireless connectivity of modern vehicles. These trends have introduced more remote surfaces/endpoints that an adversary can exploit and, in the worst case, use to control the vehicle remotely. Researchers have demonstrated how vulnerabilities in remote endpoints can be exploited to compromise ECUs, access in-vehicle networks, and control vehicle maneuvers. To detect and prevent such vehicle cyber attacks, researchers have also developed and proposed numerous countermeasures (e.g., Intrusion Detection Systems and message authentication schemes). However, there still remain potentially critical attacks that existing defense schemes can neither detect/prevent nor consider. Moreover, existing defense schemes lack certain functionalities (e.g., identifying the message transmitter), thus not providing strong protection for safety-critical ECUs against in-vehicle network attacks. With all such unexplored and unresolved security issues, vehicles and drivers/passengers will remain insecure.

This dissertation aims to fill this gap by 1) unveiling a new important and critical vulnerability applicable to several in-vehicle networks (including the Controller Area Network (CAN), the *de-facto* standard protocol), 2) proposing a new Intrusion Detection System (IDS) which can detect not only those attacks that have already been demonstrated or discussed in literature, but also those that are more acute and cannot be detected by state-of-the-art IDSes, 3) designing an attacker identification scheme that provides a swift pathway for forensic, isolation, security patch, etc., and 4) investigating what an adversary can achieve while the vehicle's ignition is *off*.

First, we unveil a new type of Denial-of-Service (DoS) attack called the *bus-off attack* that, ironically, exploits the error-handling scheme of in-vehicle networks. That is, their fault-confinement mechanism — which has been considered as one of their major advantages in providing fault-tolerance and robustness — is used as an attack vector. Next, we propose a new anomaly-based IDS that detects intrusions based on the extracted fingerprints of ECUs. Such a capability overcomes the deficiency of existing IDSes and thus detects a wide range of in-vehicle network attacks, including those existing schemes cannot. Then, we propose an attacker identification scheme that provides a swift pathway for forensic, isolation, and security patch. This is achieved by fingerprinting ECUs based on CAN voltage measurements. It takes advantage of the fact that voltage outputs of each ECU are slightly different from each other due to their differences in supply voltage, ground voltage, resistance values, etc. Lastly, we propose two new attack methods called the *Battery-Drain* and the *Denial-of-Body-control attacks* through which an adversary can disable parked vehicles with the ignition off. These attacks invalidate the conventional belief that vehicle cyber attacks are feasible and thus their defenses are required only when the vehicles ignition is on.

CHAPTER I

Introduction

Contemporary vehicles are usually equipped with 40–100 Electronic Control Units (ECUs), which are interconnected on various in-vehicle networks to exchange data for making maneuvering decisions [19]. Such internally networked ECUs improve response time, safety, control precision, and fuel-efficiency for vehicles.

Internally wired ECUs are now being connected/exposed to *external* entities. Through vehicle-embedded Bluetooth, Wi-Fi, and cellular connections, drivers and passengers are provided with various types of infotainment and other applications/services. Moreover, Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communications are used to introduce new functionalities such as remote diagnostic/prognosis, crash avoidance, and traffic management to enhance safety and mobility, and reduce environmental impact [27, 39]. Thanks to its substantial benefits to vehicles and drivers/passengers, external connectivity has been continuously gaining popularity and even starting to be considered for a regulation [12].

1.1 Security Threats for Connected Vehicles

Vehicles with more external interfaces, however, prove to be a double-edged sword. While they provide a wide range of benefits, they also open up more remote surfaces/endpoints that an attacker can exploit and, in the worst case, control the vehicle [29, 31, 69, 70, 76]. Researchers have demonstrated how vulnerabilities in such endpoints are exploited to compromise an ECU, access the in-vehicle network, and take control of the vehicle (e.g., control its brake or steering) [31, 50, 56, 58]. These vulnerabilities, unfortunately, seem to be inevitable due to the inherent nature of automotive manufacturing: in-vehicle components and software codes are developed and written by different organizations, and thus vulnerabilities emerge naturally at interface boundaries [31]. Such a reality of vehicle cyber attacks has made automotive security one of the most critical issues that need to be resolved by industry, academia, and governments.

Checkoway *et al.* [31] experimentally evaluated various remote attack vectors (e.g., CD, PassThru, Bluetooth, Cellular) and have shown that they can indeed be exploited to remotely compromise ECUs and thus seize the control of a vehicle. Moreover, the authors of [57] analyzed internal network architectures of 20 different vehicles, and have shown the practicability and feasibility of remote attacks. Exploiting the compromised ECUs, researchers have also shown to be able to control vehicle maneuvers by packet injection in the in-vehicle network [50, 56]. Recently, researchers have been able to compromise and remotely kill a Jeep Cherokee running on a highway [25, 58], which triggered a recall of 1.4 million vehicles. In 2016 and 2017, researchers were able to hack Tesla model S and X cars, and control their maneuvers [26]. The authors of [76] also succeeded in a remote attack via a vehicle's tire pressure monitoring system (TPMS). An ECU is shown in [56] to be reprogrammable by injecting forged diagnostic messages. Also proposed is new hardware that can generate/fabricate magnetic fields, spoof the wheel speed sensor of a running vehicle, thus activating the Anti-lock Braking System (ABS) [82].

1.2 State-of-the-Art Defense

As a countermeasure against attacks on in-vehicle networks, two main lines of defense networks — akin to those in the Internet security — have been pursued: *message authentication* and *intrusion detection*.

1.2.1 Message Authentication

Providing message authentication in in-vehicle networks is, however, difficult due to limited space available for appending a Message Authentication Code (MAC) in their messages' data field. For example, in Control Area Network (CAN) — the *de facto* standard in-vehicle network protocol — its payload field provides space for only up to 8 bytes of data, making it difficult to implement strong security primitives.

Several schemes have been proposed to overcome such a difficulty. Truncating MAC across multiple frames has been proposed in [83] to overcome the limited room for MAC. Instead of appending MAC in the limited data field, exploitation of an out-of-band channel was suggested in [86] for exchanging authentication messages. The authors of [71] proposed a delayed authentication protocol that uses multiple 16-bit CRC fields to include a 64-bit CBC-MAC with the KASUMI algorithm.

Although such preventive measures provide some degree of security, they alone cannot guarantee complete security due to their inability to handle certain critical attacks, e.g., Denial-of-Service (DoS). Moreover, their operations not only require a significant amount of processing power but also increase message latencies and bus utilization. Since in-vehicle networks must operate in real time and ECUs are resource-limited for cost reasons, unlike in the Internet, these "costs" of preventive measures hindered their adoption [33, 65].

1.2.2 Intrusion Detection

To overcome such limitations of preventive measures, different Intrusion Detection Systems (IDSs) have been proposed. The authors of [65] proposed a method of measuring the entropy of an in-vehicle network and used the result as a specification of the behavior for an IDS. Similarly, a method of modeling the distribution of message intervals was proposed in [56] to define a norm behavior. In-vehicle sensors were exploited in [66] to verify message range, frequency, correlation, consistency, etc. Abnormal measurements on brake-related sensors were detected by using the tire-friction model [34]. The authors of [45] monitored not only message frequency but also obvious misuse of message IDs as well as low-level communication characteristics.

The essence of such state-of-the-art IDSs is to monitor the contents and the periodicity of in-vehicle messages and verify whether there are any significant changes in them. Since they are either constant or predictable in in-vehicle networks, such approaches are feasible in most circumstances.

1.3 Problems

Various efforts from the academia, industry, and governments have been made to make vehicles more secure. However, as vehicle/automotive security directly relates to drivers/passengers' safety, we must further enhance the security of in-vehicle networks — the core of vehicles. So, it is essential to investigate 1) whether there exist any undiscovered attacks/vulnerabilities on in-vehicle networks and 2) whether state-of-the-art defenses provide a high enough level of security.

1.3.1 Undiscovered Vulnerabilities of In-vehicle Networks

With control of a compromised ECU, an attacker is shown to be able to control the vehicle maneuver via packet injection in the in-vehicle network(s). However, in order to launch such existing/known attacks, the attacker is required to first reverse-engineer messages (i.e., figure out the meaning/purpose of messages) or their checksum algorithms. Since the messages and the implemented checksum algorithms are different for different vehicle manufacturers and models, such reverse-engineering can be very painstaking, when the adversary wants to mount attacks on different vehicles. Based on the facts that not only mounting vehicle cyber attacks is very painstaking and thus difficult (especially for a relatively novice adversary) but also there exist various state-of-the-art defense schemes against them, one might (mis-)believe that vehicles with such defense schemes are already secure enough. However, as there is no one silver bullet in solving (any) security problems, we must think from the adversary's perspective and question whether there remain any undiscovered vulnerabilities of in-vehicle networks. If so, vehicles and its drivers/passengers still remain insecure.

1.3.2 Failure of Intrusion Detection and Attacker Identification

While investigating whether there remain any unknown vulnerabilities of in-vehicle networks, we must also understand the level of security that state-of-the-art defense schemes can provide. That is, it is imperative to figure out which functionalities they lack in mitigating any possible threats.

While state-of-the-art defense schemes provide a certain level of security, they all lack one important functionality: they do not know whether or not the messages on the in-vehicle network were sent by the genuine transmitter, and hence cannot detect any changes of the message transmitter. This is because in-vehicle networks are mostly configured as broadcast buses and their messages do not carry any information on the transmitters. If in-vehicle messages do not carry any information on their transmitters, one may question how IDSs can identify them and detect acute attacks/intrusions such as when the attacker ECU masquerades another.

Even in those scenarios where the state-of-the-art defense schemes are capable of determining whether or not there is an intrusion in the in-vehicle network, another important feature that they lack is determining *which* ECU is actually mounting the attack, i.e., incapable of identifying the attacker ECU. Accurate identification of an attacker, however, is imperative as it provides a swift pathway for forensic, isolation, security patch, etc. No matter how well an IDS detects the presence of an intrusion in a vehicle, if we still do not know which ECU is mounting the attack and hence which ECU to isolate/patch, the vehicle remains insecure and unsafe. It is much better and more economical to isolate/patch the attacker ECU, than blindly treating *all* ECUs as (possible) attackers.

1.4 Dissertation Contributions

Motivation and Approach

Enhancing the security of in-vehicle networks and thus vehicles is of vital importance since it directly relates to drivers/passengers' safety. Thus, it is important to look at the problem from two different vantage points: *attack* and *defense*. That is, we must think from the attacker's perspective on how s/he might exploit vulnerabilities in attacking vehicles, design their countermeasures, and help OEMs/suppliers implement them before the vulnerabilities are actually exploited by an adversary. Also, we must think from the defender's perspective and see whether there are any threats that state-of-the-art schemes fail to handle. Through such an understanding, we must design new defense mechanisms that can mitigate those threats, thus securing vehicles better.

Focus of This Dissertation

In-vehicle networks can be in the form of Local Interconnect Network (LIN), Controller Area Network (CAN), CAN with Flexible Data-Rate (CAN-FD), Media Oriented Systems Transport (MOST), FlexRay, Ethernet, etc. Among such various protocols, in this dissertation, we primarily focus on CAN, since it is the *de facto* standard for in-vehicle networks due to its maturity, low cost, efficiency, and robustness. CAN was mandated by the U.S. regulations to be outfitted on all 2008 and newer model-year vehicles. Almost every new European car also comes with the CAN bus equipped. In 2016 alone, an annual installation of 1.8 billion CAN interfaces were reported to be made [30]. In fact, CAN is used in not only passenger cars but also in trucks, buses, ships, planes, drones, submarines, and even in prosthetic limbs. Considering such a ubiquity of CAN in various safety-critical systems and its direct relationship with the people's safety, it is imperative to properly secure CAN. Although this dissertation focuses on CAN, the new attacks and defenses proposed in this dissertation are applicable not only to CAN but also to other in-vehicle networks, which we discuss further in the following chapters.

The Dissertation Statement

We enhance in-vehicle network security by discovering new attacks that are yet unknown but practical and by designing new defenses that outperform the state-of-the-art.

In this dissertation, we look at the vehicle security problem from two perspectives: as an attacker and a defender. Specifically, we (i) unveil three new attacks that existing defense mechanisms cannot detect nor prevent, (ii) design and implement a new IDS that can detect various types of attacks — including those that state-of-the-art schemes cannot — and (iii) design and implement an attacker identification scheme that provides a swift pathway for forensic, isolation, security patch, etc.; something that existing defense schemes have not been able to achieve.

1.4.1 Error Handling of In-vehicle Networks Makes Them Vulnerable

We discover a new important vulnerability called the *bus-off attack* that, ironically, exploits CAN's error-handling scheme. That is, its fault-confinement mechanism, which has been considered as one of its major advantages in providing fault-tolerance and robustness, is used as an attack vector. The attacker periodically injects attack messages to the in-vehicle network, deceives an uncompromised ECU into thinking it is defective, and eventually forces itself or even the whole network to shut down. This is an important attack that must be thwarted, since the attack, once an ECU is compromised, is easy to be mounted without reverse-engineering messages/checksums, while its prevention is very difficult. We analyze its practicability and demonstrate the attack on a CAN bus prototype and two real vehicles. Based on our analysis and experimental results, we propose a defense mechanism to prevent the bus-off attack.

1.4.2 CIDS: Fingerprinting ECUs for Vehicle Intrusion Detection

To overcome the limitations of state-of-the-art IDSs and defend against various vehicle attacks, we propose a new anomaly-based IDS, called *Clock-based IDS* (`CIDS`). The need of `CIDS` for vehicles is motivated through an analysis of three representative in-vehicle network attacks — fabrication, suspension, and masquerade attacks. Our analysis shows that state-of-the-art IDSs are insufficient, especially in detecting the masquerade attack due to the absence of the transmitters information in messages. `CIDS` overcomes these limitations of existing IDSs by fingerprinting in-vehicle ECUs. `CIDS` monitors the intervals of (commonly seen) periodic in-vehicle messages, and then exploits them to estimate the clock skews of their transmitters which are then used to fingerprint the transmitters. Based on the thus-obtained fingerprints, `CIDS` constructs a norm model of ECUs clock behaviors using the Recursive Least Squares (RLS) algorithm and detects intrusions with a Cumulative Sum (CUSUM) analysis. This enables `CIDS` to detect not only attacks that have already been demonstrated or discussed in literature, but also those that are more acute and cannot be detected by state-of-the-art IDSs. Our experimental evaluations on a CAN bus prototype and 3 real vehicles show that `CIDS` detects various types of in-vehicle network intrusions with a low false-positive rate of 0.055%.

1.4.3 Viden: Attacker Identification on In-Vehicle Networks

Although state-of-the-art defense schemes may detect the presence of an intrusion, they fail to identify which ECU actually mounted the attack. To meet this need, we developed a novel scheme, called `Viden` (*Voltage-based attacker identification*), which can identify the attacker ECU by measuring and utilizing voltages on the in-vehicle network. The first phase of `Viden`, called *ACK learning*, determines whether or not the measured voltage signals really originate from the genuine message transmitter. `Viden` then exploits the voltage measurements to construct and update the transmitter ECUs voltage profiles as their fingerprints. It finally uses the voltage profiles to identify the attacker ECU. Since `Viden` adapts its profiles to changes inside/outside of the vehicle, it can pinpoint the attacker ECU under various conditions. Moreover, its efficiency and design-compliance with modern in-vehicle network implementations make `Viden` practical and easily deployable. Our extensive experimental evaluations on both a CAN bus prototype and two real vehicles have shown that `Viden` can accurately fingerprint ECUs based solely on voltage measurements and thus identify the attacker ECU with a low false identification rate of 0.2%.

1.4.4 Who Killed My Parked Car?

We discover and apply two new practical and important cyber attacks — called *Battery-Drain* (BD) and *Denial-of-Body-control* (DoB) — on real vehicles, invalidating the conventional belief that vehicle cyber attacks are feasible and thus their defenses are required only when the vehicle's ignition is on. The former can drain the vehicle battery while the latter can prevent the vehicle owner/driver from starting or even opening/entering his car. These attacks will likely delude forensics since their symptoms are very similar to typical mechanical/electronic failures, thus triggering unnecessary repairs, extending service outage, or even blaming wrong parties (vehicle OEMs/suppliers) for the problem.

We first analyze how operation (e.g., normal, sleep, listen) modes of ECUs are defined in various in-vehicle network standards and how they are implemented in real vehicles. From this analysis, we discover that an adversary can exploit the wake-up function of in-vehicle networks—which was originally designed for enhanced user experience/convenience (e.g., remote diagnosis, remote temperature control)—as an *attack vector*. Ironically, the key battery-saving feature in in-vehicle networks makes it easier for an attacker to wake up ECUs and, therefore, mount BD and/or DoB attacks.

Via extensive experimental evaluations on various real vehicles, we show that by mounting the BD attack, the adversary can increase the average battery consumption by at least 12.57x, drain the car battery within a few hours in the worst case, and thereby immobilize/cripple the vehicle. We also demonstrate the DoB attack on a real vehicle, showing that the attacker can disable communications between the vehicle and its key fob by indefinitely shutting down an ECU, thus making the driver unable to start and/or even enter the car.

1.4.5 Organization of Dissertation

This dissertation is organized as follows. The first chapter unveils the bus-off attack vulnerability and details its countermeasure. The second and third chapter propose `CIDS` and `Viden`, respectively. The fourth chapter unveils the two new attacks — BD and DoB attack — which can be mounted while the car is parked with its ignition off. Finally, we conclude the dissertation in the fifth chapter where we also discuss future research directions.

CHAPTER II

Error Handling of In-vehicle Networks Makes Them Vulnerable

2.1 Introduction

To detect and prevent vehicle cyber attacks, various types of security solutions, such as Message Authentication Code (MAC) and Intrusion Detection Systems (IDSs) for in-vehicle networks — akin to those in the Internet security — have been proposed [45, 65, 71, 83]. These solutions provide a certain level of security, but there still remain critical, uncovered vulnerabilities specific to the automotive domain.

In this chapter, we unveil a critical uncovered vulnerability called the *bus-off attack*. Bus-off attack is a new type of Denial-of-Service (DoS) attack which, ironically, exploits the error-handling scheme of in-vehicle networks in shutting down a vehicle. That is, their fault-confinement mechanism — which has been considered as one of their major advantages in providing fault-tolerance and robustness — is used as an attack vector. The attacker periodically injects attack messages to the in-vehicle network, deceives an uncompromised ECU into thinking it is defective, and eventually forces itself or even the whole network to shut down. In addition to its severe consequences, the following unique characteristics of the proposed bus-off attack differentiate itself from previously known attacks and make it a critical threat which must be countered.

1. The bus-off attack is easy to mount since the attacker is not required to reverse-engineer messages (i.e., figure out the meaning/purpose of messages) or their checksum algorithms for launching it. Thus, the attacker can easily mount the attack on various vehicles regardless of their manufacturer or model. This is in sharp contrast to previously demonstrated attacks, which required painstaking reverse-engineering procedures in order to take control of a vehicle [50,56,58].
2. As the symptoms of the bus-off attack resemble those of system errors such as improper termination [17], bit flip [85], and bit drop [67], it deceives state-of-the-art IDSs to think the network is erroneous, while it is actually under attack.
3. Although MACs for in-vehicle network messages may thwart most of the previously known attacks, they cannot prevent the proposed bus-off attack since it nullifies their functionalities. That is, not only contemporary insecure in-vehicle networks but also prospective *security-enhanced* ones will still be vulnerable to the bus-off attack.
4. Since the attack relies solely on low-level safety features of in-vehicle networks, it is independent of actual implementation subtleties of different ECUs.

In this chapter, we will primarily focus on *what* an adversary can do with a compromised ECU, rather than *how* the ECU was compromised in the first place, which has been covered well elsewhere [31,41,50,56,58].

This work makes the following main contributions:

- Discovery of a new Denial-of-Service threat — bus-off attack — on in-vehicle networks which exploits their error-handling mechanism as an attack vector;
- Analysis and characterization of real CAN bus traffic, and the proof of the practicability of the bus-off attack;
- Implementation and demonstration of the bus-off attack on a CAN bus prototype and on two real vehicles; and
- Development and evaluation of a countermeasure that can detect and prevent the bus-off attack.

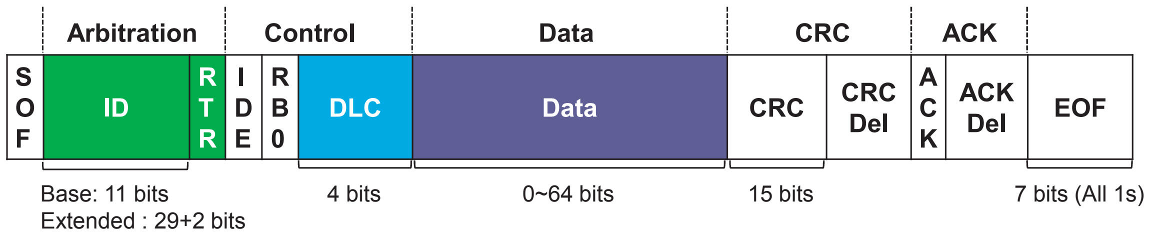


Figure 2.1: Format of a CAN data frame.

The rest of the chapter is organized as follows. Section 2.2 provides the required background on CAN, and Section 2.3 details the proposed bus-off attack. Section 2.4 discusses the feasibility of the attack and Section 2.5 evaluates the attack on a CAN bus prototype and real vehicles. Section 2.6 discusses the limitations of state-of-the-art solutions in preventing the bus-off attack, and Section 2.7 details a new defense mechanism against it. Section 2.8 discusses further the severity of the bus-off attack as well as its applicability to other in-vehicle networks. Finally, we conclude the chapter in Section 2.9.

2.2 Primer on CAN

For completeness, we first review the main features of CAN related to the proposed attack; see the CAN specification [2] for the features that are not discussed here.

2.2.1 CAN Frames

CAN interconnects ECUs/nodes through a message broadcast bus. Each node broadcasts periodic (and occasionally sporadic) data frames on the CAN bus to provide retrieved data. The transmitted data is received by one or more nodes on the bus and then utilized for maintaining data consistency and for vehicle control decisions.

Frame format. Each CAN frame is basically a sequence of dominant (0) and recessive (1) bits, and belongs to one of four different types: *data* frame which is used for sending retrieved data; *remote* frame for requesting transmission of a specified message; *error* frame used to indicate detected errors via error flags; and *overload* frame to inject delay between

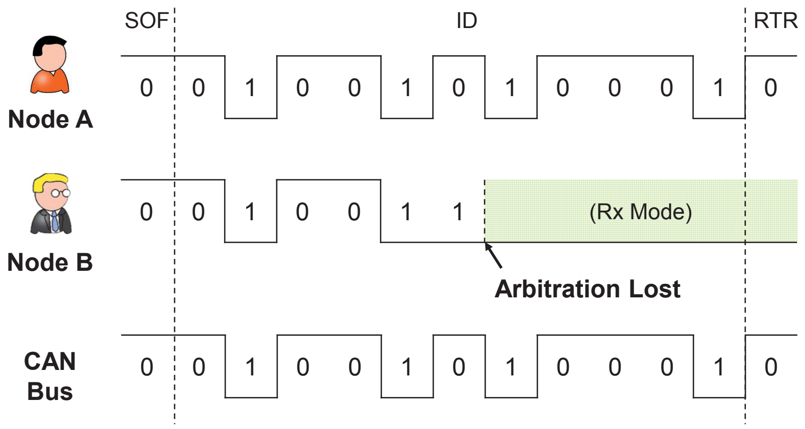


Figure 2.2: Example of CAN arbitration phase.

frames. Fig. 2.1 shows the base format of a CAN data frame. A data frame can carry up to 8 bytes of data, the length of which is specified in the 4-bit Data Length Code (DLC). For most passenger cars, a 1-byte checksum of each message is contained in the last byte of its data field [58]. Although the checksum is not part of the CAN specification, car manufacturers implement it using their own algorithms to provide an additional layer of protection to the CAN Cyclic Redundancy Check (CRC). Consecutive transmissions of CAN frames are separated by a 3-bit Inter-frame Space (IFS).

Message ID. Instead of containing the transmitter/receiver address, a CAN frame contains a unique ID, which represents its priority and meaning. For example, a frame containing wheel speed values might have ID=0x01 and frame containing battery temperature values might have ID=0x20. Only one ECU is assigned to transmit a given ID at a time, and the ID values are defined to be distinct from each other by the manufacturer. The base frame format has an 11-bit ID, whereas an extended format has a 29-bit ID. Since the use of base format is much more prevalent, we focus on the base format in this dissertation. Note, however, that the attack model proposed in this chapter is not dependent on the type of format.

2.2.2 Arbitration

Once the CAN bus is detected idle, a node with data to transmit, starts its frame transmission (Tx) by issuing a Start-of-Frame (SOF). SOF provides *hard* synchronization between ECUs to make bitwise transmission and reception feasible. At that time, one or more other nodes may also have buffered data to transmit, and may thus concurrently access the bus. In such a case, the CAN protocol resolves the access contention via *arbitration*.

During transmission, each node sends its frame one bit at a time and monitors the actual output on the CAN bus. In the arbitration phase, since frame IDs are unique and the CAN bus logically behaves as a wired-AND gate (e.g., 0 & 1 = 0), some contending nodes would see a dominant (0) bit even though it has transmitted a recessive (1) bit. In such a case, they lose arbitration, withdraw from bus contention, and switch to receiver mode. In the end, only *one* arbitration-winner node is allowed to continuously access the bus for data transmission. This process enables higher-priority frames (i.e., lower IDs) to be transmitted before lower-priority ones. Once the arbitration winner has completed transmission of its frame ending with an End-of-Frame (EOF), after a 3-bit time of IFS, the bus becomes free again for access, i.e., idle. At that time, nodes that have buffered data or had previously lost arbitration, start another round of arbitration for access. For completeness, we illustrate in Fig. 2.2 how arbitration is done in CAN.

2.2.3 Error Handling

Error handling is built in the CAN protocol and is important for its fault-tolerance. It aims to detect errors in CAN frames and enables ECUs to take appropriate actions, such as discarding a frame, retransmitting a frame, and raising error flags. The CAN protocol defines no less than 5 different ways of detecting errors [2].

- Bit Error: Every transmitter compares its transmitted bit with the output bit on the CAN bus. If the two are different, a *bit error* has occurred, except during arbitration.

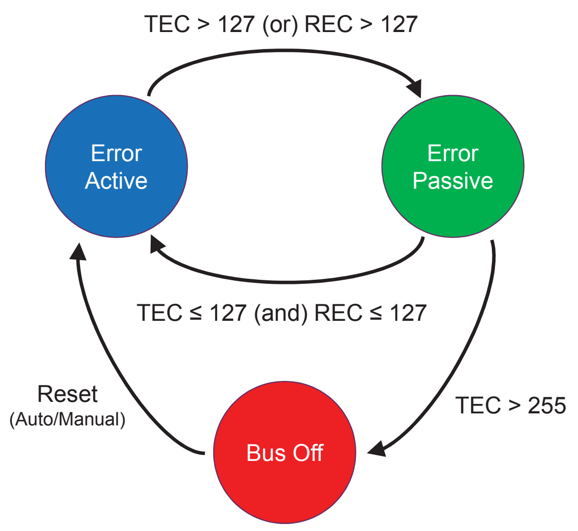


Figure 2.3: State diagram of fault confinement in CAN.

- Stuff Error: After every five consecutive bits of the same polarity, an opposite polarity bit is stuffed for maintaining *soft* synchronization. Violation of this incurs a *stuff error*.
- CRC Error: If the calculated CRC is different from the received CRC, a *CRC error* is raised.
- Form Error: If the fixed-form bit fields (e.g., CRC delimiter, ACK delimiter, EOF, IFS) contain at least one illegal bit, a *form error* has incurred.
- ACK Error: When a node transmits a message, any node that has received it issues a dominant bit in the ACK slot. If none replies, an *ACK error* is raised.

Error counters. For any detected errors, the perceived node transmits an error frame on the bus and increases one of the two error counters it maintains: *Transmit Error Counter* (TEC) and *Receive Error Counter* (REC). There are several rules governing the increase/decrease of these counters, but in essence, a node that detects an error during transmission increases TEC by 8, whereas if perceived during reception, REC is increased by 1 [2]. Moreover, for any error-free transmission and reception, TEC and REC are decreased by 1, respectively.

Fault confinement. To confine serious errors disrupting bus communications, each ECU's error mode is managed as in Fig. 2.3. All ECUs start in an *Error-active* mode and switch between different modes depending on their TEC and REC values. When TEC or REC exceeds 127 due to consecutive errors, the node becomes *Error-passive*, and only returns to its initial state when both go below 128. When TEC exceeds the limit of 255, the corresponding ECU — which must have triggered many transmit errors — enters the *Bus-off* mode. Upon entering this mode, to protect the CAN bus from continuously being distracted, the error-causing ECU is forced to shut down and *not* participate in sending/receiving data on the CAN bus at all. It can be restored back to its original error-active mode, either automatically or manually. However, since bus-off is usually an indication of serious network errors and may not be fixed by mere automatic re-initialization of the CAN controller, a user-intervened recovery or even a controlled shut-down of the entire system is recommended [75].

Bus-off recovery. The reasons for such different options are 1) the bus-off recovery mechanism depends on the software stack being used, i.e., how the system is designed by the manufacturer, and 2) ECUs have different ASILs (Automotive Safety Integrity Levels). Since the bus-off is a serious problem, in most cases, vehicle systems are designed to first enter a "limp home" mode (when it occurs) with all their parameters set to pre-set values, and thus run with reduced functionality (e.g., limited engine RPM). In this mode, warning lamps are lit up on the dashboard to alert the driver, and the vehicle runs only for some time before it is properly serviced by an OEM-authorized service center. Depending on the severity of the underlying issue, i.e., which ECU was shut down, a vehicle in the limp home mode will later be totally disabled.

Error flags. When an error is detected, the perceived node indicates to others on the bus via an error flag, which comes in two forms: active and passive. For any perceived errors, nodes that are in error-active mode issue an active error flag which consists of 6 *dominant* bits. So, the transmitted frame causes other nodes to violate the bit-stuffing rule, transmit their own error frame caused by the stuff error, and terminate any on-going transmissions or receptions.

For nodes that are in error-passive mode, they operate in the same way as error-active ones, except that they issue a passive error flag which consists of 6 *recessive* bits and have an 11 (not 3) bit-time of IFS if they were the transmitter of the previous message [2]. An error-passive node tries to signal its passive error flag until it actually observes 6 recessive bits on the bus, i.e., an indication of the error flag being properly sent. Since recessive bits are overwritten on the CAN bus by dominant bits, the thus-issued passive error flags may persist until the end of a frame.

2.3 Attack Model

We first discuss the adversary model under consideration, and then uncover a new vulnerability of in-vehicle networks.

2.3.1 Adversary Model

We consider an adversary whose objective is to shut down uncompromised (healthy) in-vehicle ECUs with a minimal number of message injections. Such an objective precludes other types of attacks (e.g., flooding) which, albeit their greater impact, require a large number of message injections and are thus easier to be detected. In Section 2.8, we will discuss more on such attacks, highlighting the severity of the bus-off attack. As in previously discussed attacks [31, 50, 56, 58], we assume that the adversary can physically/remotely compromise an in-vehicle ECU through numerous attack surfaces and means, and thus gain its control. In contrast, we do *not* require the adversary under consideration to reverse-engineer messages or checksums in order to achieve its goal of shutting down an ECU. Since the messages and the implemented checksum algorithms are different for different vehicle manufacturers and models, such reverse-engineering can be very painstaking, when the adversary wants to mount attacks on different vehicles.

Once an ECU is compromised, we consider the adversary to be capable of performing at least the following malicious actions. The adversary can *inject* any message with forged

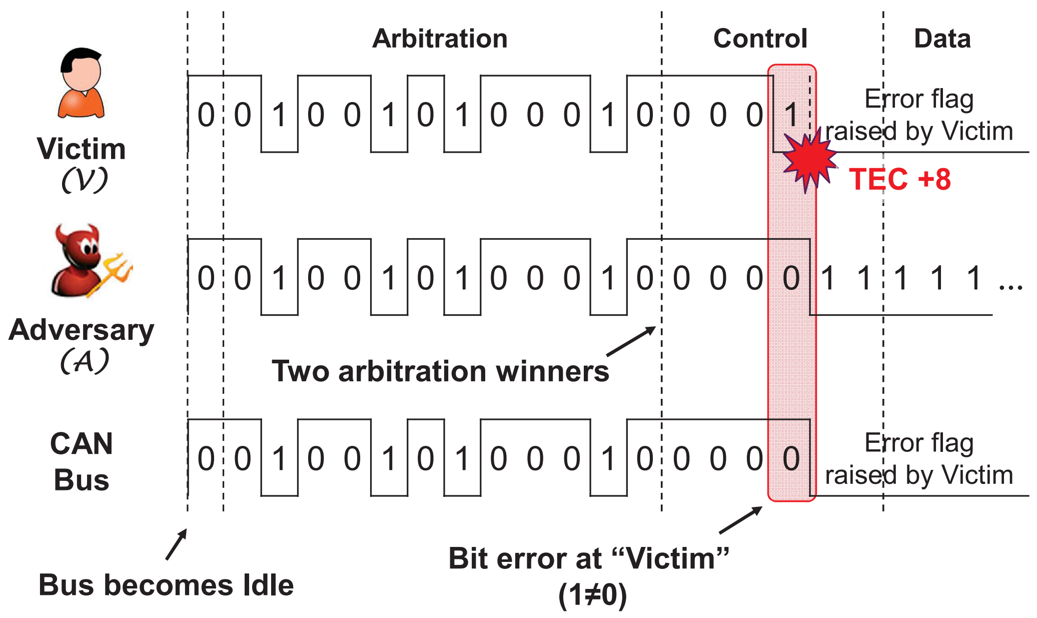


Figure 2.4: Example of a bus-off attack.

ID, DLC, and data on the bus as they are managed at user level. Also, since CAN is a broadcast bus, the adversary can *sniff* messages on CAN. Restrictions of message filters are detailed in Section 2.4.2. These are the *basic* capabilities of an adversary who has the control of a compromised ECU. Practicability of such an adversary model has already been proved and demonstrated in [31, 50, 56].

2.3.2 Bus-off Attack

We now introduce the *bus-off attack* which exploits the following feature of CAN: CAN's error handling automatically isolates defective or "misbehaving" ECUs — whose TEC>255 — into *bus-off* mode. Specifically, by iteratively injecting attack messages, the adversary coerces the TEC of an uncompromised/healthy victim ECU to continuously increase — deceiving it to think it is defective — and in the end, triggers the CAN fault confinement to force the victim or even the entire network to shut down.

Increasing the victim's TEC. Suppose message $\mathbb{M}$ is periodically sent by some victim ECU $\mathbb{V}$. Then, an adversary $\mathbb{A}$ can succeed in a bus-off attack by injecting an attack message, which satisfies the following conditions.

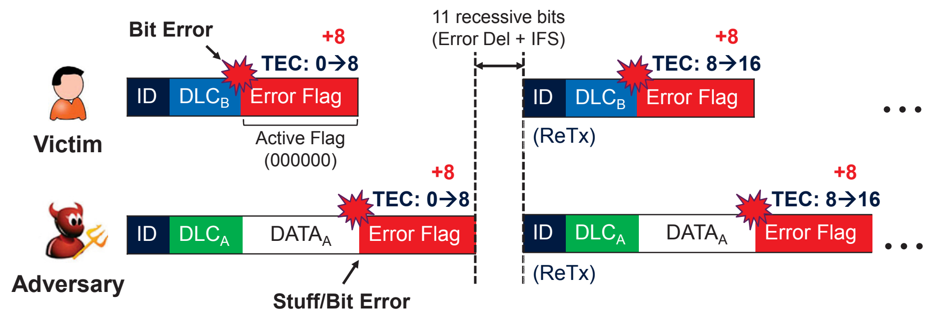
C1. ID – *Same* ID as message $\mathbb{M}$;

C2. Timing – Transmitted at the *same* time as $\mathbb{M}$; and

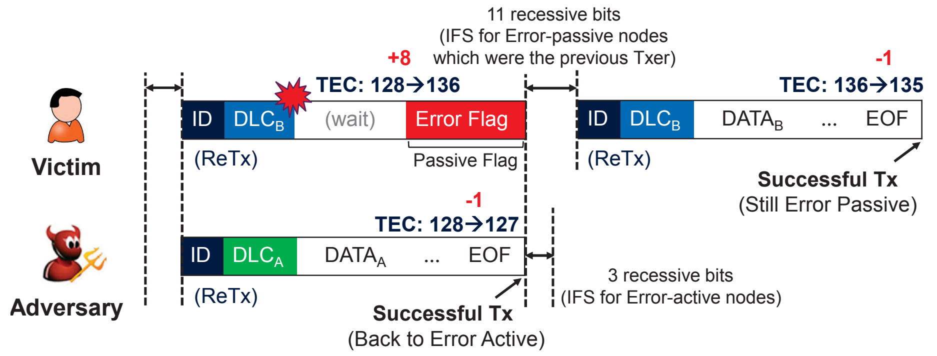
C3. Contents – Having at least one bit position in which it is dominant (0), whereas it is recessive (1) in $\mathbb{M}$. All preceding bits should be the same as $\mathbb{M}$.

To describe the general attack model, for now we assume that the adversary can transmit a message that satisfies C1–C3. Details of how and why C1–C3 can be met will be discussed in Section 2.4. As shown in Fig. 2.4, when the bus becomes idle, not only the victim transmits message $\mathbb{M}$ but also the adversary transmits an attack message satisfying C1–C3. So, not one but two transmitters — $\mathbb{A}$ and $\mathbb{V}$ sending messages with identical IDs — win arbitration, and thus *concurrently* send their bit values of control, data, etc., on the bus. The two nodes' bitwise transmissions are synchronized in virtue of hard and soft synchronizations. Since the attack message meets C3, $\mathbb{V}$ sees an opposite polarity on the bus to the one it transmitted (this happens after an arbitration). As a result, the victim $\mathbb{V}$ experiences a bit error *forced* by $\mathbb{A}$, thus increasing its TEC by 8. By repeating this bus-off attack on the victim's messages, the adversary can make the victim's TEC to continuously increase, and force the victim to enter bus-off mode and disconnect from the bus. Although the victim is error-free in transmitting messages other than the targeted one, since TEC increases by 8 upon detection of each error but decreases only by 1 for each error-free transmission, an iterative bus-off attack rapidly increases the victim's TEC. The entire process of an iterative bus-off attack consists of the following two phases.

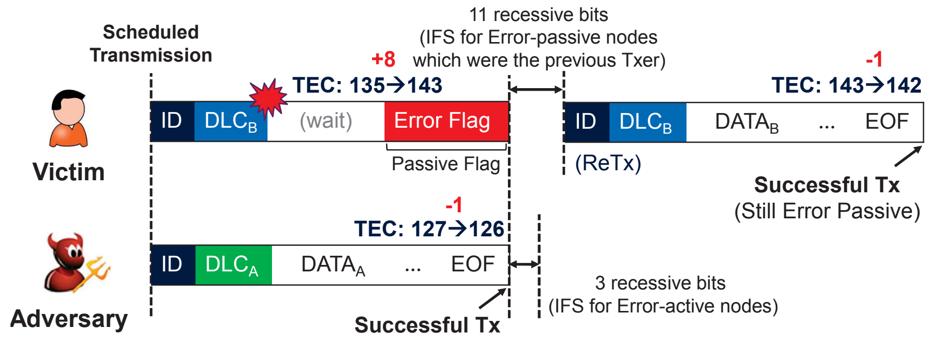
Phase 1 – Victim in error-active. Both adversary and victim nodes start in their default mode, error-active. After observing messages on the CAN bus, the adversary targets one of them for a bus-off attack. We refer to such a message as the *target message* and its transmitter as the victim. As mentioned earlier, the adversary then injects its attack message at the same time as the target message to increase the victim's TEC. Thus, as shown in Fig. 2.5a, the victim experiences bit error, transmits an *active* error flag, and increases its



(a) Phase 1 – Victim in error-active mode.



(b) Transition from Phase 1 to 2.



(c) Phase 2 – Victim in error-passive mode.

Figure 2.5: Two phases of bus-off attack.

TEC by 8. Since an active error flag consists of 6 consecutive dominant bits (i.e., 000000), either a stuff or bit error is triggered at the adversary node, and its TEC also increases by 8. After the error delimiter and IFS, the CAN controllers of the adversary and the victim automatically retransmit the Tx-failed messages again at the same time. So, the exact same bit error recurs until they both enter error-passive mode. What is significant about the attack in Phase 1 is that the adversary can coerce the victim to become error-passive with just one message injection.

Phase 1 to 2. After 16 (re)transmissions, as shown in Fig. 2.5b, both the adversary and the victim become error-passive when their TEC=128. Again, for the retransmitted message, bit error occurs at the victim node. However, since the victim is now in error-passive mode, it attempts to deliver a *passive* error flag which consists of 6 recessive bits. At that time, since the adversary transmits its frame, the attempt to deliver the error flag will persist until the adversary's EOF. In contrast to Phase 1, the adversary node experiences no error and thus succeeds in transmitting its frame, whereas the victim will succeed later during its retransmission. In total, due to a bit error ($+8$) and a successful retransmission (-1), the victim's TEC changes $128 \rightarrow 136 \rightarrow 135$, whereas the adversary's changes $128 \rightarrow 127$. Accordingly, the adversary returns to error-active, while the victim remains error-passive. Up to this point, all but the first change in TEC are achieved via automatic retransmissions by the CAN controller, i.e., *the controller does it all for the attacker!*

Phase 2 – Victim in error-passive. Fig. 2.5c illustrates Phase 2 of the bus-off attack in which only the victim is error-passive. Once the scheduled interval of the target message has elapsed, the victim again transmits that message, and thus at the same time, the adversary re-injects its attack message. Since the victim is still in error-passive mode, as it was the case when transitioning from Phase 1 to 2, the adversary can decrease its TEC further by 1. On the other hand, the victim's TEC is again increased by 7 ($= +8 - 1$), thus keeping the victim in error-passive mode. In Phase 2, the adversary iterates this process for every periodically transmitted target message until the victim is eventually forced to bus off, i.e., TEC$>$255. This implies that the periodicities of the attack and the target messages are the same. As a result, the victim ECU becomes disconnected, and in the worst case, the entire network shuts down [75].

Although CAN messages' ID values do not contain information on their actual transmitters, their values and intervals together imply the messages' priority and safety-criticality. That is, if the attacker targets a message sent with high priority (i.e., a low ID value) and small message intervals, then the attacker would most likely disconnect a *safety-critical* ECU that sends important messages related to, for example, vehicle acceleration or braking.

Alternative bus-off attacks. The adversary may attempt to iterate this process for not only every periodic transmission but also every retransmission (as in Phase 1). However, as shown in Fig. 2.5c, since error-passive nodes, which were the transmitter of the previous message, have a longer IFS than error-active nodes, the adversary cannot synchronize its injection timing with the victim's retransmission, thus failing to mount the attack.

Yang [89] reported that an uncovered fatal error can sometimes occur due to a misspecification of CAN; if a new frame is transmitted on the bus during the error delimiter subsequent to a passive error flag issued by an error-passive node, a form error incurs to that node, thus increasing its TEC. If the adversary were to exploit such an inherent specification error in CAN to mount a bus-off attack, it may be simpler to coerce the victim to bus off. However, the occurrence of that fatal error depends on what type of new frame is transmitted at that time as well as how high the bus load is. Note that it would be very difficult for an adversary to control such factors. That is, it is too restrictive for the attacker to achieve the bus-off attack in that way. By contrast, we propose and discuss a more general method for mounting the bus-off attack, i.e., succeeding the attack under any bus condition.

2.4 Feasibility of Bus-off Attack

To mount the bus-off attack, the adversary has to inject an attack message that satisfies conditions on ID, timing, and contents. We now discuss how and why each condition can be met, in the ascending order of difficulty.

2.4.1 Different Contents

To trigger a bit error at the victim and increase its TEC, the attack message must first satisfy C3: having at least one bit position in which its signal is dominant (0), whereas the victim's is recessive (1), and all preceding bits are identical. Since the bus-off attack also requires the attack and target messages to have the same ID (i.e., C1 – identical arbitration fields), the mismatch in C3 must occur in either the control or the data field. Note that this is infeasible in other fields such as CRC and ACK, because they are determined by the CAN controller, not by the user/adversary. Since CAN messages normally have DLC set to at least 1 and non-zero data values, one simple but most definite way for the adversary to cause a mismatch is to set the attack message's DLC or data values to all 0s. Also, given that DLC for each CAN ID is usually constant over time, the attacker can learn the value and set its attack message's DLC accordingly. This way, the adversary can satisfy C3.

2.4.2 Same ID

The next difficult task is to meet C1, which requires the attack message to have the same ID as the target. That is, the adversary must know in advance the ID used by the target message. The fact that favors the adversary is that CAN is a broadcast bus system. However, each ECU cannot acquire the IDs of all received messages except those passed through its message filter. We distinguish a *received* message from an *accepted* message, depending on whether it passed through the filter and then arrived at a user-level application. Since an adversary can read contents only from accepted messages, meeting C1 depends on how the filter is set at the compromised ECU.

Empty message filter. Some ECUs in vehicles have almost, if not completely, empty message filters so that they can receive, accept, and process almost all messages on the bus, thus making it trivial to satisfy C1. A typical example of these ECUs is the telematic unit, which has to operate in that way to provide a broad range of features, e.g., remote diagnostics, anti-theft. Another interesting aspect of this ECU is that, from a security viewpoint, it is regarded as one of the most vulnerable ECUs due to its wide range of external/remote attack surfaces. Note that several researchers have already shown the practicability of compromising the telematic unit [31, 41, 50, 56]. So, this implies that an adversary can compromise some empty filter ECUs more easily than those with non-empty filters, thus satisfying C1.

Non-empty message filter. Although the message filter of a compromised ECU is preset to accept messages with only a few different IDs, it does not restrict the adversary in attacking them. Furthermore, by directly modifying the message filter, the adversary can also mount the attack on messages that would usually have been filtered out. The two most common CAN controllers — Microchip MCP2515 and NXP SJA1000 — both allow modification and disabling of message filters through software commands, when the ECU is in configuration mode [10, 15]. For ECUs with the Microchip MCP2515 CAN controller, the configuration mode can be entered not only upon power-up or reset but also via user instructions through the Serial Peripheral Interface (SPI). Through the SPI, it is also possible for the user to read/write the CAN controller registers, including the filter register [10]. Thus, such user-level features for configuring the CAN controller allow attackers to easily enter configuration mode via software commands, and modify/disable the message filters, thus satisfying C1.

2.4.3 Tx Synchronization

Even though the adversary knows the ID and contents to use for his attack message, he should also know exactly *when* to send it. Unless the adversary is capable of sending the attack message at the same time as the target message, not only once but iteratively, he would fail to cause a bit error, or increase the victim's TEC.

Difficulties in synchronizing the Tx timing. C2 requires the transmission of the attack and the target messages to be synchronized with less than a bit resolution. If the attack timing is wrong by even one bit, there won't be two arbitration winners and the attack

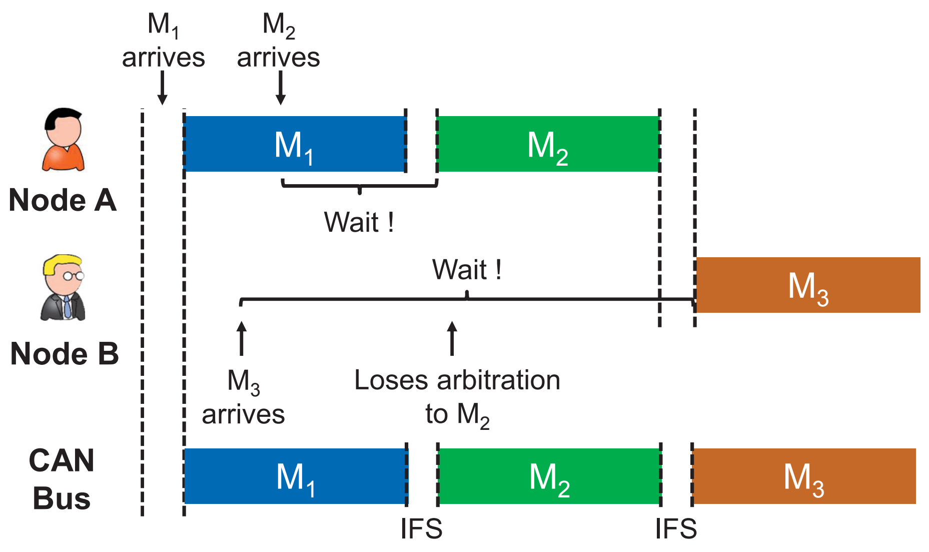


Figure 2.6: Example of preceded IDs.

would thus fail. For synchronizing the timing of its transmission, the adversary may utilize the fact that CAN messages are usually sent at fixed intervals. For example, once the adversary learns that the target message is sent every T ms, it can attempt to transmit its attack message when T ms has elapsed since the target's last transmission. However, such an approach would be inaccurate due to jitters. Since jitters make the actual message periodicities deviate from their preset values [46], albeit leveraging message periodicity to fulfill C2, the attacker would have difficulties in synchronizing the transmission of its attack message with the target's.

Preceded ID. In order to overcome these difficulties, the adversary can exploit another fact of CAN: nodes, which have either lost arbitration or had new messages buffered while the bus was busy, attempt to transmit their messages as soon as the bus becomes idle.

We define a *preceded ID* of $\mathbb{M}$ as the ID of the message that has completed its transmission right before the start of $\mathbb{M}$'s. Consider an example where node A transmits messages with ID=$\mathbb{M}_1$, $\mathbb{M}_2$, and node B transmits a message with ID=$\mathbb{M}_3$ which has the lowest priority among them. As shown in Fig. 2.6, if these messages are arriving and being queued at the depicted times, $\mathbb{M}_1$ and $\mathbb{M}_2$ would be the preceded IDs of $\mathbb{M}_2$ and $\mathbb{M}_3$, respectively, with only a 3-bit IFS separating the corresponding message pairs. In

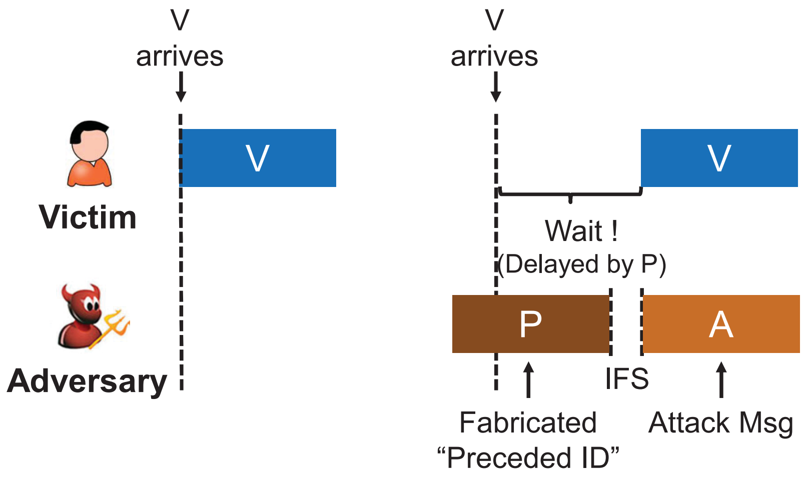


Figure 2.7: Fabricated preceded ID.

other words, the transmissions of $\mathbb{M}_2$ and $\mathbb{M}_3$ are forced to be buffered until their preceded ID messages have been transmitted on the bus. Since message priorities and periodicities do not change, such a feature implies that one particular CAN message may always be followed by another specific message, i.e., there is a *unique* preceded ID for that specified one. As an example, if the periodicities of their transmissions are either same or integer multiples (e.g., 5ms for $\mathbb{M}_1$, $\mathbb{M}_2$, and 10ms for $\mathbb{M}_3$), then $\mathbb{M}_2$ would always be the preceded ID of $\mathbb{M}_3$, i.e., be the *unique preceded ID*. Hence, regardless of jitter, the exact timing of message transmissions becomes rather predictable and even determinative: 3 bit-time after the preceded ID's completion.

Bus-off attack by exploiting preceded IDs. In the above example, to attack $\mathbb{M}_3$, the adversary can monitor the CAN bus, learn its preceded ID of $\mathbb{M}_2$ or even $\mathbb{M}_2$'s preceded ID of $\mathbb{M}_1$, and buffer an attack message with ID=$\mathbb{M}_3$ when receiving one of them. Then, its CAN controller would always transmit the attack message after $\mathbb{M}_2$'s transmission (i.e., concurrently with the target message), and the adversary will thus succeed in the bus-off attack. Likewise, the adversary can target $\mathbb{M}_2$ by buffering its attack message with ID=$\mathbb{M}_2$, as soon as it receives its preceded ID of $\mathbb{M}_1$. If the preceded ID is unique, then the bus-off attack can be iterated for its every reception and thus consecutively increase the victim's TEC.

Even though the target message does not have such a preceded ID, an adversary can fabricate it in order to synchronize the timing and thus succeed in mounting the attack. Consider an example shown in Fig. 2.7 where a victim node periodically transmits message $\mathbb{V}$, which has no preceded IDs. In such a case, just before the transmission of $\mathbb{V}$, the adversary can inject some message $\mathbb{P}$ and an attack message $\mathbb{A}$, sequentially. Hence, $\mathbb{V}$'s transmission gets delayed until the completion of $\mathbb{P}$, i.e., the adversary fabricates $\mathbb{P}$ as the preceded ID of $\mathbb{V}$, and thus the attack message is synchronized with its target. Our evaluation results will later demonstrate the efficiency of bus-off attacks based on the above approaches.

2.4.4 Preceded IDs in Actual CAN Traffic

The key point in meeting C2 and succeeding in the bus-off attack is leveraging the preceded IDs of the target message. Depending on the configuration and scheduling of messages, some target messages may (or may not) have a preceded ID. We first consider the case in which the adversary is targeting messages with *genuine* (unfabricated) preceded IDs. Hence, it is essential to verify their existence in actual CAN traffic as well as usefulness in satisfying C2. That is, the following questions should be answered:

- **Existence –** Are there such preceded IDs in real in-vehicle network traffic?
- **Uniqueness –** If yes, how many distinct preceded IDs are there for a specified message?
- **Pattern –** If more than one, are there any patterns in the preceded IDs which can be utilized?

We answer these questions via an analysis of actual CAN traffic data. We use CAN data that was recorded from a 2010 Toyota Camry by Ruth *et al.* [78]. During a 30-minute drive, the data was logged by a Gryphon S3 and Hercules software [78]. According to the logged data, there were 42 distinct messages transmitted on the CAN bus: 39 of them sent periodically at intervals ranging from 10ms to 5 secs, and 3 of them sent sporadically.

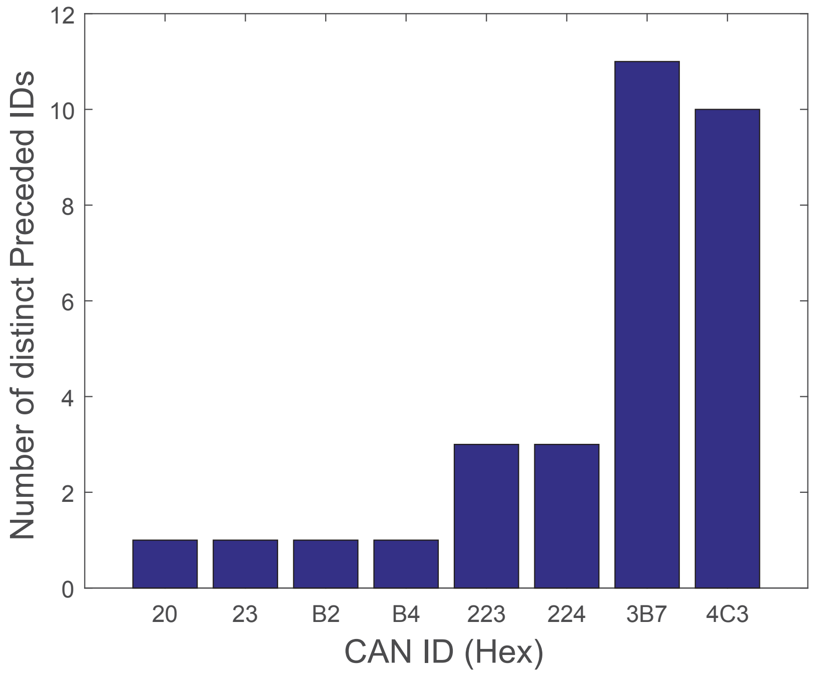


Figure 2.8: Distinct preceded IDs.

Existence and uniqueness. Of these, we first identify the ones of interest to us: messages that are *always* sent right after another's completion, i.e., have preceded IDs (e.g., $\mathbb{M}_2$ or $\mathbb{M}_3$ in Fig. 2.6). Of the 39 periodic messages seen on the bus, we were able to find 8 of such type. Fig. 2.8 shows the number of distinct preceded IDs of the ones labeled in the x-axis. For example, message 0xB2 had a unique preceded ID 0xB0, i.e., 0xB2 always followed 0xB0, where both were sent every 10ms. On the other hand, message 0x3B7 had 11 different kinds of those IDs. The result showing that 10% of the periodic messages have a unique preceded ID answers the question of their existence as well as uniqueness in actual CAN traffic, and thus implies that the bus-off attack exploiting genuine preceded IDs is indeed feasible in actual vehicles. Since in-vehicle messages have fixed priorities, are sent periodically, and some have to be sent consecutively by an ECU (e.g., two messages containing front and rear wheel speed values), we believe preceded IDs are prevalent in all other types of passenger cars as well.

Patterns in preceded IDs. Another interesting aspect of the CAN bus traffic, which

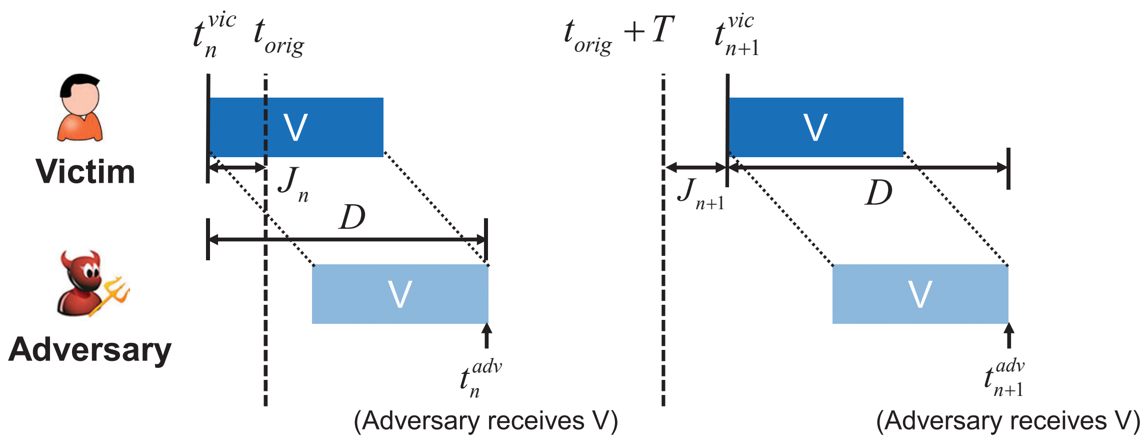


Figure 2.9: Timing of preceded ID message injection.

caught our attention was that there were notable *patterns* in the preceded IDs. As shown in Fig. 2.8, message 0x223 with periodicity 30ms had 3 distinct preceded IDs, meaning that observing those IDs may not help determine the transmission timing of 0x223. However, we were able to extract an interesting pattern in their transmission: the 6n-th transmission of 0x20 was the unique preceded ID of the 2n-th transmission of 0x223, where n is an integer. That is, even though a unique preceded ID was not observable for every transmission, it was for every n-fold transmission, i.e., there exists a pattern in preceded IDs. Therefore, by observing the CAN bus traffic, acquiring knowledge of genuine preceded IDs, and thus meeting not only C1, C3 but also C2, the adversary can succeed in mounting bus-off attack.

2.4.5 Fabrication of Preceded IDs

Even though the targeted messages do not have a preceded ID, as shown in Fig. 2.7, the adversary can fabricate it to meet C2. Therefore, we will henceforth refer preceded ID messages to ones which are fabricated by the adversary. The injection timing, quantity, and the contents of the preceded ID message are important for the adversary to mount a bus-off attack with preceded IDs.

Injection timing. To succeed in the bus-off attack via fabrication of preceded IDs, it is essential for the adversary to inject that fabricated message right before the target message. In other words, the adversary is required to estimate when the target message would be transmitted on the bus. Although most in-vehicle messages have fixed periodicity, randomness incurred from jitters makes such estimation rather difficult. As shown in Fig. 2.9, consider a target message $\mathbb{V}$ with periodicity of T, which is expected to be transmitted at times t_{orig}, $t_{orig} + T$, and thereafter. Note that T is a predefined and constant value for periodic messages. However, due to the jitters of J_n and J_{n+1} — caused by variations in the transmitter node's clock drift, task scheduling, execution time, etc. [46] — the victim's messages are transmitted on the bus at times t_n^{vic} and t_{n+1}^{vic}, where n is the sequence index. From the adversary's perspective, due to an incurred delay of $\mathbb{D}$ from transmission and reception, it receives message $\mathbb{V}$ at times t_n^{adv} and t_{n+1}^{adv}. Note that $\mathbb{D}$ includes delays for message transmission, propagation, and processing. Since the number of bits in a certain message is almost a constant and the bit timing of CAN already takes into account of the signal propagation on the bus, without loss of generality, we assume $\mathbb{D}$ to be constant for a given message $\mathbb{V}$ [2,68].

Thus, the only remaining randomness in the timing of message transmission is jitter (e.g., J_n). Jitter is known to follow a Gaussian distribution due to randomness in thermal noise, which also follows a Gaussian, and the Central Limit Theorem, i.e., composite effects of many uncorrelated noise sources approach a Gaussian distribution [46]. So, we can consider J_n and J_{n+1} as outcomes of a Gaussian random variable $J \sim \mathbb{N}(0, \sigma_v^2)$. Thus, the times when the adversary receives $\mathbb{V}$ can be expressed as:

$$\begin{aligned} t_n^{adv} &= t_n^{vic} + \mathbb{D} = t_{orig} + J_n + \mathbb{D} \\ t_{n+1}^{adv} &= t_{n+1}^{vic} + \mathbb{D} = t_{orig} + T + J_{n+1} + \mathbb{D}, \end{aligned} \tag{2.1}$$

where $J_n < 0$ and $J_{n+1} > 0$ in Fig. 2.9. Then

$$t_{n+1}^{vic} = t_n^{adv} + T - \mathbb{D} + J_{n+1} - J_n = t_n^{adv} + T - \mathbb{D} + J^*, \tag{2.2}$$

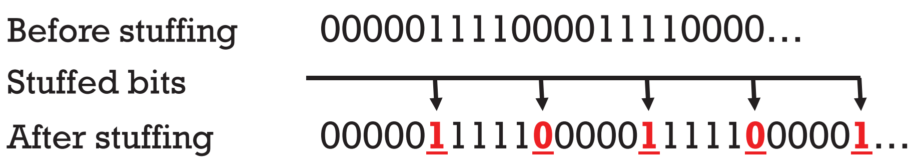


Figure 2.10: Maximizing Tx duration via bit-stuffing.

where $J^* \sim \mathbb{N}(0, 2\sigma_v^2)$ since its outcomes are $J_{n+1} - J_n$. Note that in Eq. (2.2), J^* is the only random variable whereas others are either constant or measurable by the adversary. Such an equation shows that the adversary can indeed obtain an approximate estimation of when the victim would transmit its message, i.e., the target message, at the *next* sequence. As shown in Fig. 2.7, for the fabrication of preceded IDs to be effective, the adversary has to 1) start transmission of its preceded ID message(s) before the target and 2,3) hold the CAN bus, i.e., make the bus busy, until it becomes sure that the attack and target messages would synchronize. That is, the adversary must meet the following three conditions:

$$\begin{aligned}
&1)\ t_{fab} < \min(t_{n+1}^{vic}) = t_n^{adv} + T - \mathbb{D} + \min(J^*)\\
&2)\ t_{fab} + \mathbb{H} > \max(t_{n+1}^{vic}) = t_n^{adv} + T - \mathbb{D} + \max(J^*)\\
&3)\ \mathbb{H} = \kappa\mathbb{F} > \max(J^*) - \min(J^*),
\end{aligned} \tag{2.3}$$

where t_{fab} denotes when the adversary starts to inject its fabricated preceded ID message(s). Moreover, $\mathbb{H}$ denotes the duration of the adversary holding the bus, which is equivalent to κ preceded ID messages each sent for a duration of $\mathbb{F}$. Since J^* is a bounded Gaussian random variable, the boundaries can be approximated as $|max(J^*)| = |min(J^*)| \simeq \mathbb{I}\sqrt{2}\sigma_v$, where σ_v is measurable, and $\mathbb{I}$ an attack parameter. Since J^* is Gaussian, setting $\mathbb{I} = 3$ would provide a 99.73% confidence and $\mathbb{I} = 4$ a 99.99% confidence. In total, to fully exploit the fabricated preceded IDs, the adversary has to start injecting them prior to $t_n^{adv} + T - \mathbb{D} - \mathbb{I}\sqrt{2}\sigma_v$. Note that the adversary should not lower t_{fab} beyond $\max(t_{n+1}^{vic}) - \mathbb{H}$, which can be set once $\mathbb{H}$ is determined.

Number of messages. Once satisfying 1) in Eq. (2.3), the adversary has to satisfy 3)

— occupy the bus at least for the duration of $\max(J^*) - \min(J^*) = 2\sqrt{2}\mathbb{I}\sigma_v$, which can be met via $\kappa(\geq 1)$ injections of preceded ID messages. Since the adversary's objective is to mount the bus-off attack with a minimal number of injections, κ should be kept to minimum by maximizing $\mathbb{F}$. To maximize $\mathbb{F}$, i.e., the duration of its preceded ID message occupying the bus, the adversary can exploit the bit-stuffing rule of CAN: after every 5 consecutive bits of the same polarity (e.g., 00000), an opposite polarity bit is stuffed. By fabricating its preceded ID message with DLC=8 and the data field as shown in Fig. 2.10, the adversary can maximize the number of stuffed bits and thus $\mathbb{F}$ to at least $\mathbb{F}^* = (8L + 44 + \lfloor 8L/4 \rfloor)/S_{bus} = 124/S_{bus}$, where 44 denotes the number of bits exterior to the data field, L the DLC=8, and S_{bus} the bus speed. Note that at least $2L$ bits are added to the fabricated message according to the CAN's bit-stuffing rule. Hence, if we consider a CAN bus with $S_{bus} = 500$Kbps, using a single injected preceded ID message, the adversary can take control of the bus for at least 0.248ms. Such analyses suggest that for targeting a message with a jitter deviation of σ_v, the adequate number of preceded ID messages can be expressed as $\kappa = \left\lceil \frac{\max(J^*) - \min(J^*)}{\mathbb{F}^*} \right\rceil = \left\lceil \frac{2\sqrt{2}\mathbb{I}\sigma_v S_{bus}}{124} \right\rceil$. For example, if $\sigma_v = 0.025$ms and $S_{bus} = 500$Kbps, the adversary is only required to inject $\kappa = \lceil 0.8554 \rceil = 1$ preceded ID message with $\mathbb{I} = 3$, i.e., 99.73% confidence. To ensure the effectiveness of the fabricated preceded IDs with a near-perfect confidence, the adversary can set $\mathbb{I} = 4$ and thus inject $\kappa = \lceil 1.1405 \rceil = 2$ messages at time t_{fab}. Our evaluation will later show that one injection of fabricated ID messages ($\kappa = 1$) can be sufficient for a bus-off attack.

Contents of the preceded ID message. Other than the control and data fields, the adversary also has to carefully decide which ID to use for fabricating the preceded ID messages. If only one preceded ID message is to be used for the attack, the adversary can exploit the next seemingly free ID. To be as elusive as possible, the ID value can be changed for each attempt of attack and be chosen from those least frequently sent on the bus. If two preceded ID messages are to be used, the adversary can similarly inject the first one with any free ID but should inject the second one with an ID having higher priority

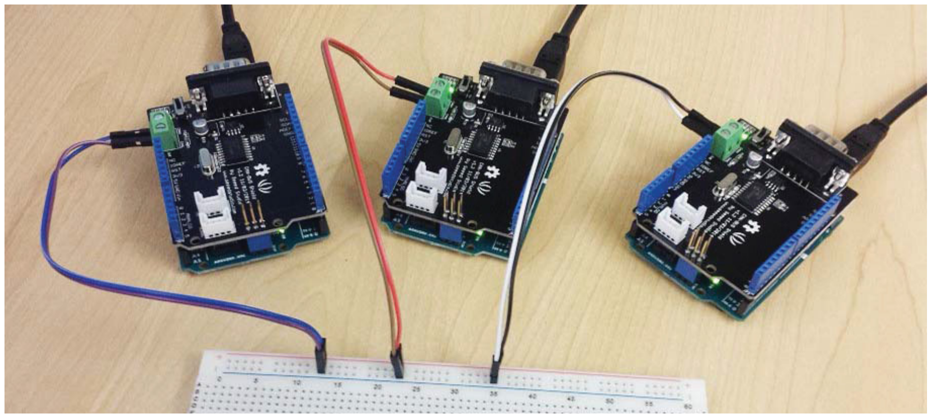


Figure 2.11: CAN bus prototype.

(smaller value) than the target.

2.5 Evaluation

We now evaluate the feasibility of the proposed bus-off attack on a CAN bus prototype and two *real* vehicles. For in-depth analyses of the bus-off attack, we first evaluate the attack on the CAN bus prototype, and then extend the evaluation to real vehicles.

2.5.1 Bus-off Attack

As shown in Fig. 2.11, we configured a CAN prototype in which all 3 nodes were connected to each other via a 2-wire bus. Each node consists of an Arduino UNO board and a SeeedStudio CAN bus shield, which is composed of a Microchip MCP2515 CAN controller with SPI interface, MCP2551 CAN transceiver, and a 120Ω terminal resistor to provide CAN bus communication capabilities.

Evaluation setup. The CAN bus prototype was set up to operate at 500Kbps as in typical in-vehicle CAN buses. Three interconnected nodes were each programmed to replicate the scenario shown in Fig. 2.6, which is also commonly seen in actual CAN bus traffic: every 10ms, node *A* was programmed to send two consecutive messages with ID=0x07 and 0x09, and *B* to send one message with a lower priority of ID=0x11. Node

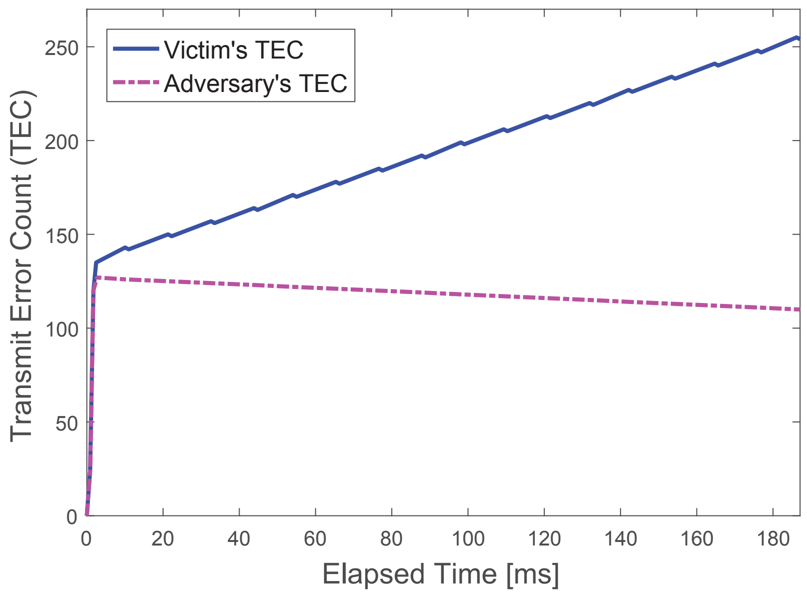


Figure 2.12: TECs during a bus-off attack.

B buffered its message when a message with ID=0x07 was received. Note that 0x07 and 0x09 become the genuine preceded IDs of 0x09 and 0x11, respectively. We will later extend this evaluation of the bus-off attack to the case without assuming the availability of genuine preceded IDs. Last, but not least, the third node was programmed as an attacker which learns any preceded IDs on the bus and repetitively launches the bus-off attack. In our evaluation, message 0x11 from *B* was set as the attacker's target message. Hence, the attack message was set up to have the same ID=0x11 but with a different DLC (=0). Since the targeted message had a period of 10ms, the bus-off attack was repeated at the same time interval. For every message transmission/reception, the Transmit Error Count (TEC) of each node was read from its CAN controller register. The entire procedure of an iterative bus-off attack was re-initiated once a node entered the bus-off mode, and was examined 1,000 times.

Changes in TEC during a bus-off attack. Fig. 2.12 shows how the TECs of the victim and the adversary change during an iterative bus-off attack. All 1,000 examinations showed near identical changes as shown in Fig. 2.12. In the initial stage of the attack,

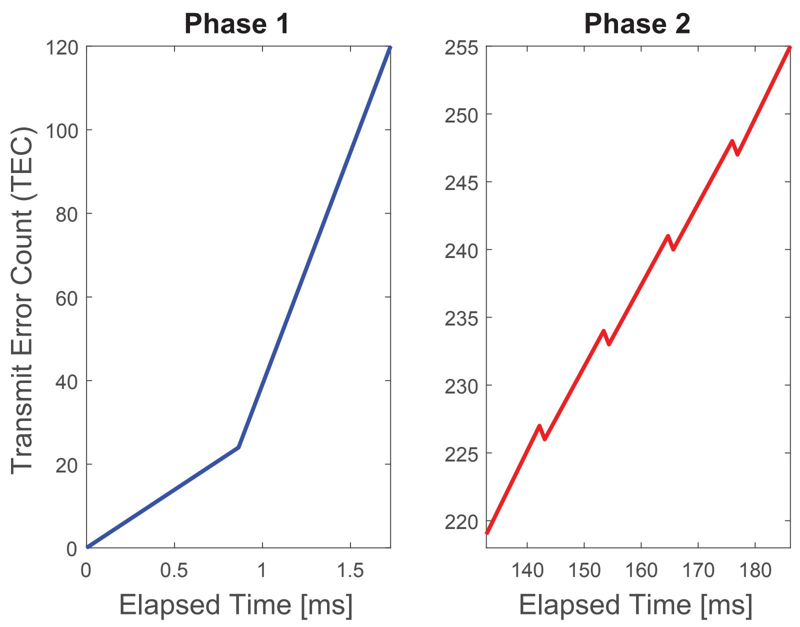


Figure 2.13: Victim's TEC during Phase 1 and 2.

there was a steep rise in the TEC of both nodes. This is because in Phase 1, with just one attack message, bit errors incurred for not only the initial transmission but also all subsequent retransmissions as depicted in Fig. 2.5a. Once the victim became error-passive, i.e., TEC>127, the attack entered its second phase. Here, with successful message transmissions, the adversary was able to recover back to, and remain as error-active. On the other hand, the victim experienced iterative bit errors during its transmissions, and eventually entered the bus-off mode when its TEC exceeded 255.

Properties of attack Phase 2. Fig. 2.13 shows a magnified plot of the changes in the victim's TEC during the attack. In Phase 1, TEC monotonically increased due to the errors in all (re)transmissions. On the other hand, once the attack entered Phase 2, the difference in error mode lets the victim succeed in its transmission only after experiencing a bit error. Thus, whenever an attack message was injected into the bus by the attacker, the victim's TEC was first increased by 8 and then immediately decreased by 1. The net TEC increase of 7 each time eventually forced the victim to be disconnected from the bus. These results confirm the properties of Phase 2 discussed in Section 2.3.2.

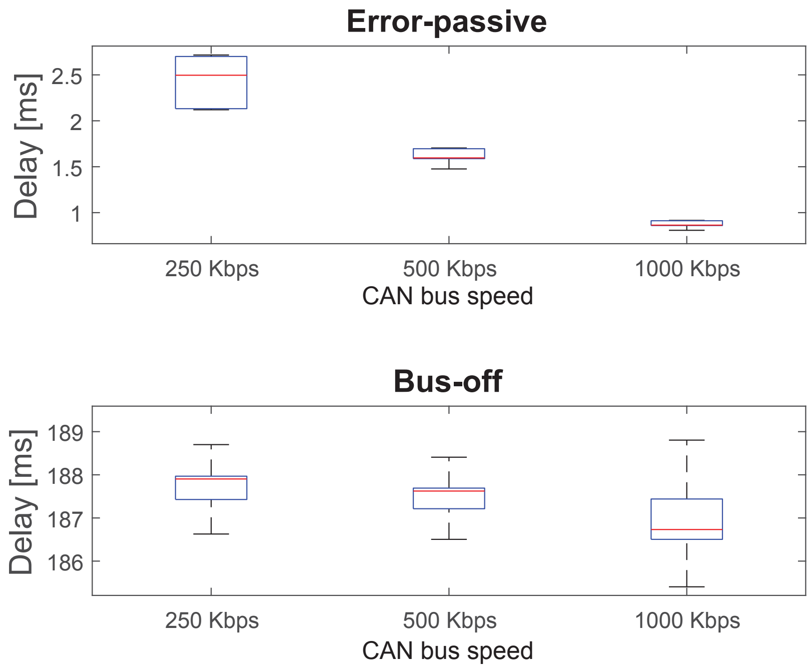


Figure 2.14: Delays of bus-off attack under different bus speeds.

2.5.2 Attack Under Different Bus Conditions

The bit-rate of the CAN bus can vary from 125Kbps to 1Mbps, depending on its purpose, and its bus load can also vary with time. Thus, in order to examine the practicability of a bus-off attack under different bus conditions, we conducted the same experiment in Section 2.5.1 1,000 times, while varying the speed and load of the bus. Each time we measured the average delays of the bus-off attack that forces a victim to enter error-passive and bus-off modes, as they represent the following metrics: (1) the required number of attack messages, and (2) the probability of the attack's success. If there was at least one attempt in which the attack failed, then the maximum deviation in delays would be at least the inter-attack interval.

Different bus speeds. Fig. 2.14 shows box plots of the average delays of the bus-off attack in coercing the victim to become error-passive and bus-off. The bus speed was varied from 250Kbps to its maximum of 1Mbps. As shown in Fig. 2.14 (top), for all bus speeds, it took much less than 10ms for the adversary to coerce the victim to become error-passive.

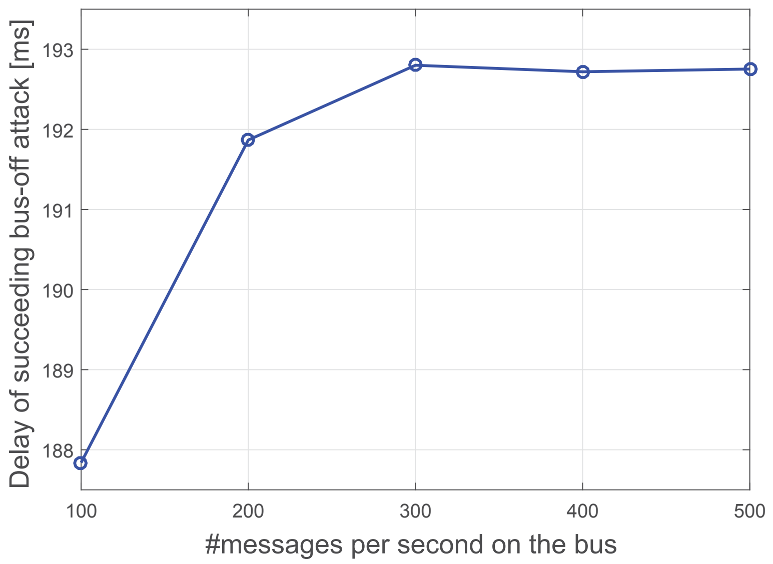


Figure 2.15: Delays of bus-off attack under different bus loads.

Since the attack message was injected every 10ms, this implies that only a single injection of the attack message was required. Also, observing that the maximum deviation was less than 10ms, all attempted bus-off attacks succeeded, irrespective of the bus speed. The attacker was able to make the victim enter error-passive mode faster with the increase of bus speed, because a higher bus speed enabled frame (re)transmissions to complete more quickly. Fig. 2.14 (bottom) shows the total delay of the victim eventually becoming bus-off at different bus speeds. For all settings, the maximum deviation was less than 10ms, again implying a 100% success probability of the attack.

Different bus loads. Not only the speed but also the load of the bus was varied in evaluating the attack. To generate different bus loads, the node that was neither the adversary nor the victim injected 100∼500 messages per second. Their IDs were randomly chosen among the set of unused ID values, and their DLCs were set randomly between 1 and 8. Fig. 2.15 shows the average total delay of coercing the victim node to eventually bus off under the given bus loads. As the bus load increases, the overall delay is shown to rise, because some of the randomly injected messages won arbitration over the target and attack messages and thus delayed their transmission. Note, however, that since they both

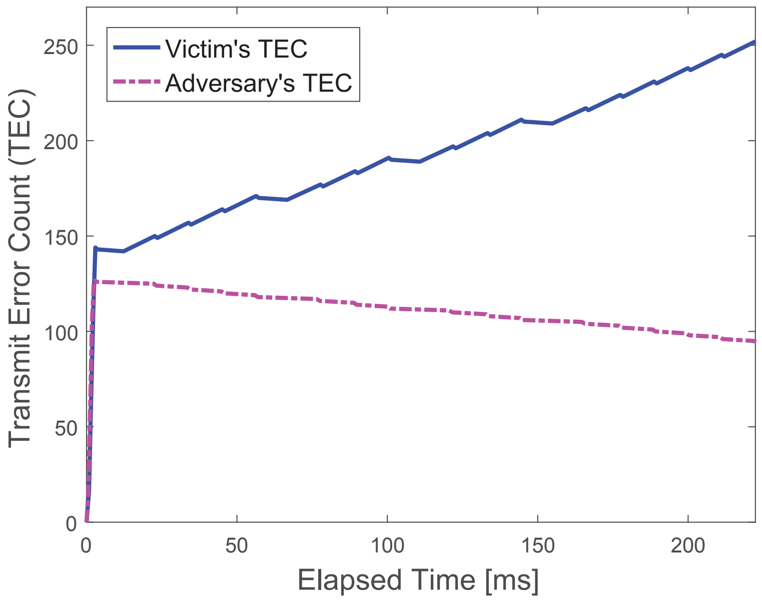


Figure 2.16: Changes in TECs when the preceded ID is fabricated.

had the same IDs, they both won/lost the arbitration. Therefore, in all 1000 examinations with different bus loads, all trials of bus-off attack succeeded regardless of the bus load.

2.5.3 Periodicity vs. Preceded ID

To mount a bus-off attack, the attack message has to satisfy conditions C1–C3. Of these, C2 is the most difficult to meet, but can be satisfied by exploiting either of the following three scenarios:

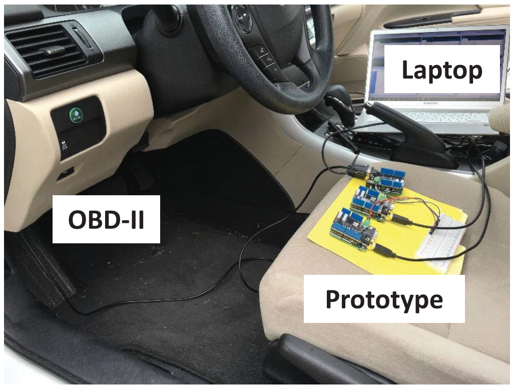
- **Periodicity** – measure the Tx interval of the target message and exploit it for synchronizing the transmission timing.
- **Genuine ID** – assuming genuine preceded IDs are available, exploit them for the bus-off attack.
- **Fabricated ID** – fabricating and thus exploiting the preceded ID of a target message.

We evaluated all of these in order to verify their accuracies and efficiencies. For the first scenario, the adversary and the victim were programmed to send messages every 10ms with the same ID but different DLC values. The first transmissions from both nodes were initiated by a reference message sent by the non-victim node. For the second scenario of exploiting genuine preceded IDs, the nodes were programmed equivalently as discussed in Section 2.5.1. Finally, for the third scenario, we programmed the adversary to fabricate one preceded ID message per attack as shown in Fig 2.7. For each attack scenario, we examined 50,000 bus-off attack trials.

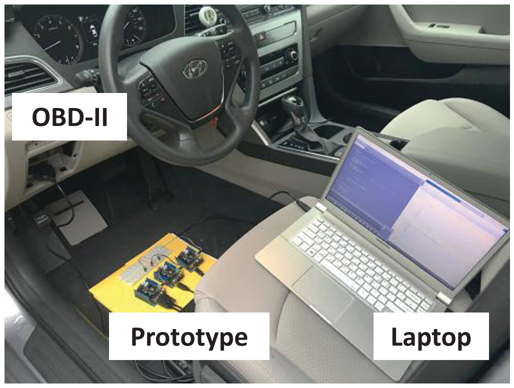
When periodicity was exploited to synchronize the Tx timing for a bus-off attack, due to jitters, only 58 out of 50,000 trials (0.12%) were able to trigger a bit error at the victim, thus eventually not triggering a bus-off. On the other hand, when genuine preceded IDs were assumed to be present and thus were exploited, all 50,000 trials succeeded in increasing the victim's TEC. Even without assuming that there is a genuine preceded ID for the target message, the adversary increased the victim's TEC 45,127 times (out of 50,000 trials) by fabricating it, i.e., a 0.9025 success probability. Note that in achieving such a high probability, one fabricated preceded ID was sufficient since the jitter deviation $\sigma_v = 0.023$ms. Although some attempts failed, due to the high success rate and the nature of change in TEC (i.e., $+8$ in TEC in case of error and -1 in the absence of error), iterative bus-off attacks eventually forced the victim to bus off as shown in Fig. 2.16. One can see that the change in TEC is slightly different from the one in Fig. 2.12 due to some failed attempts. These results show that a preceded ID — regardless of whether genuine or fabricated — is a good indicator for determining the exact timing of a specific message, and is indeed useful for mounting a bus-off attack.

2.5.4 Bus-off Attack on Real Vehicles

To evaluate the feasibility of bus-off attack further, we also conducted experiments on two real vehicles, 2013 Honda Accord and 2016 Hyundai Sonata shown in Figs. 2.17a and 2.17b. During our experiments, the vehicles were immobilized for safety in an isolated and controlled environment. As shown in Fig. 2.17, through the On-Board Diagnostic (OBD-II) port, we were able to connect our CAN bus prototype to their in-vehicle CAN buses, both



(a) Communication with ECUs in a 2013 Honda Accord.

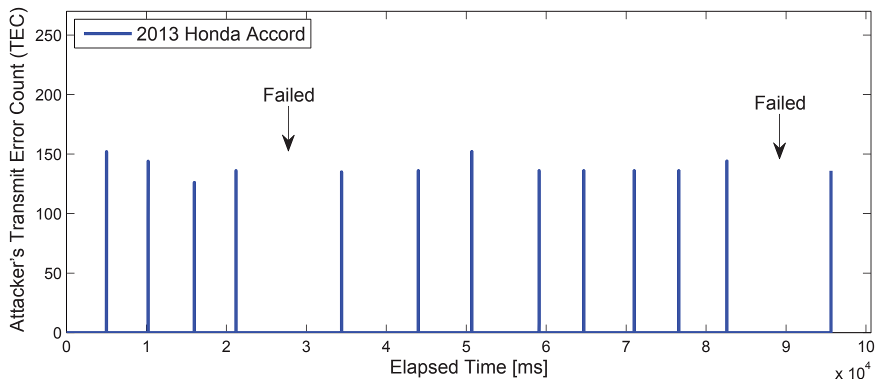


(b) Communication with ECUs in a 2016 Hyundai Sonata.

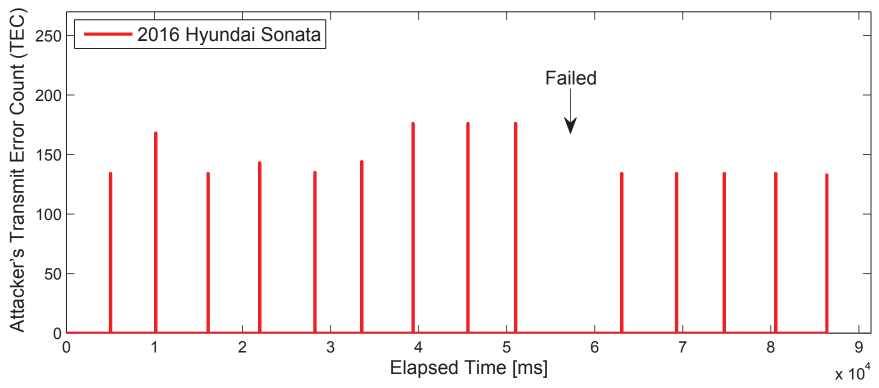
Figure 2.17: Bus-off attack experiments on (a) Honda Accord 2013 and (b) Hyundai Sonata 2016.

of which run at 500Kbps. Thus, the 3 prototype nodes were able to read all 40 distinct broadcast messages from the Honda Accord's CAN bus and 58 distinct messages from the Hyundai Sonata's CAN bus. Moreover, the nodes were capable of injecting and delivering arbitrary messages to the in-vehicle ECUs.

Increasing the TEC of a real in-vehicle ECU. We first experimentally show that an attacker can synchronize its Tx timing with a real ECU, increase its TEC, and thus succeed in launching a bus-off attack. For the evaluation on the Honda Accord, one prototype node was programmed as the attacker mounting a bus-off attack on one of its ECUs, which sent message 0x295 every 40ms. The attacker node made 15 attempts of the bus-off attack on that ECU by fabricating the preceded ID of 0x295. For the evaluation on the Hyundai Sonata, the node was programmed to mount the attack on an ECU which sent message



(a) 2013 Honda Accord.



(b) 2016 Hyundai Sonata.

Figure 2.18: Transmit error counts of ECUs in the two real vehicles are increased via bus-off attack.

0x164 every 10ms. Similarly, 15 attempts of the bus-off attack via fabrication of preceded ID were made.

Due to restrictions in accessing the error counters of real in-vehicle ECUs, we validate the attack on real ECUs by exploiting the fact that in Phase 1 of the bus-off attack, the attacker's and the victim's TECs increase *equivalently* and both eventually exceed 127 within a very short period. So, Fig. 2.18 shows (i) how the attacker's TEC changed during Phase 1 of 15 bus-off attack attempts under both settings, and (ii) how the TEC of the in-vehicle victim ECU changed. In contrast, during Phase 2, the attacker's TEC does not reflect the victim's TEC. Since the results from only Phase 1 are valid, once the attacker's TEC exceeds 127, the node is reset to initialize its error counters. For the real ECU's

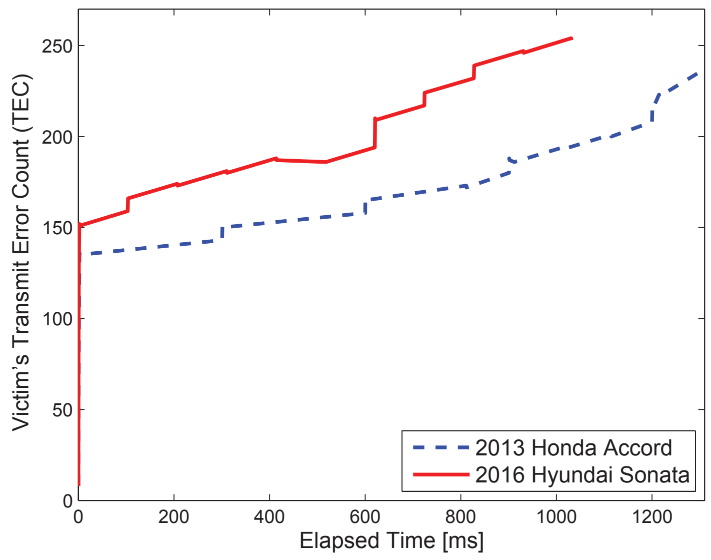


Figure 2.19: Iterative bus-off attack in a Honda Accord and a Hyundai Sonata.

TEC to be re-initialized so as to iterate Phase 1, the adversary mounted the attack every 5 secs, thus allowing the real ECU to decrease its TEC back to 0 as a result of its error-free transmissions during this period.

One can see from Fig. 2.18a that in 13 out of 15 attempts, the attacker's and hence the Honda Accord ECU's (victim's) TEC rose steeply, thus making both ECUs enter error-passive mode. As the attack was mounted via one fabricated preceded ID, 2 out of 15 attempts failed. The steep rises of TEC were again due to automatic retransmissions by the attacker node's and the in-vehicle ECU's CAN controllers. Similarly, Fig. 2.18b shows that, except for one attempt, the Hyundai Sonata ECU's TEC always increased when the bus-off attack was mounted. Although we could only show the result from Phase 1, it implies that the attacker can synchronize the Tx timing with a real ECU, and iterate such a process to continuously increase its TEC, eventually forcing it to disconnect from the in-vehicle network. Moreover, it shows that the attack can succeed regardless of the vehicle model/year, and corroborates an important fact of the bus-off attack: there is no need to reverse-engineer messages or checksums for mounting the bus-off attack, thus making it easier for the adversary to launch the bus-off attack.

Forcing an in-vehicle ECU to bus off. To further demonstrate the feasibility of the bus-off attack on real vehicles, we also evaluated a scenario in which one of the CAN prototype nodes was made to be the victim. The period of the target message sent by the victim was set to 50ms. Since the three CAN prototype nodes were capable of exchanging messages with real ECUs in both vehicles, they were successfully *added* to their in-vehicle CAN networks, and hence operate/act as if they were real ECUs. As a result, an attack on one of them would be equivalent to an attack on a real ECU. Unlike the prototype setting, however, their bus loads were significantly higher — due to traffic generated by real in-vehicle ECUs — during the attack, i.e., the attack was evaluated in highly complex CAN bus traffic. Fig. 2.19 shows the changes in TECs of the victim node being attacked on the 2013 Honda Accord and the 2016 Hyundai Sonata. Under both settings, through iterative bus-off attacks, the victim became error-passive within 2.4ms and eventually entered bus-off mode. Compared to the prototype setting, since the target message was set to have a larger interval and the bus loads were much higher, the overall delays of the victim entering error-passive and bus-off were larger.

These results on the two real vehicles confirm that the bus-off attack is indeed a severe, real problem.

2.6 Related Work

As a countermeasure against attacks on in-vehicle networks, message authentication and intrusion detection systems (IDSs) have been the two main lines of defense.

As we had discussed in Chapter I, providing message authentication for CAN is difficult due to the limited space available for appending a MAC in its data field. Moreover, the requirement of real-time communication and processing makes the provision of authentication a non-trivial problem. Although, several schemes have been proposed to overcome these difficulties, in the proposed bus-off attack, the adversary causes a bit error at the victim *during* message reception. Thus, even though a MAC is appended to the message, its functionalities will be nullified. Similarly, the functionalities of message checksums are also nullified.

Other than authentication methods, IDSs have also been proposed as countermeasures. The essence of state-of-the-art IDSs is to monitor the periodicity and contents of messages, and verify whether there are any significant changes to them. Although such IDSs are capable of detecting attacks to some extent, they overlooked the fact that message periodicity can change even in a normal or uncompromised environment. For example, if an error had occurred due to hardware/software fault, then that message would automatically be retransmitted by the CAN controller, thus changing its transmission interval [77]. As a result, since an abnormal periodicity may be due to a system error as well as an attack, it is unreasonable to directly map such a symptom to an attack. For example, 16 consecutive error frames can occur due to not only a bus-off attack but also improper CAN termination [17]. This implies that for the proposed bus-off attack, the IDS may not tell if the error was due to an attack or a system error.

2.7 Countermeasures

The proposed bus-off attack is an important vulnerability, especially in view of its capability of nullifying MACs and checksums, and also deceiving IDSs to think there is a system error although the network is actually under attack. We propose a new defense mechanism which leverages the following features of the bus-off attack for its prevention.
In Phase 1, due to CAN's automatic retransmission,

F1. at least two *consecutive* errors occur during the transmission of frames. Thus, we watch for consecutive error frames with an active error flag.

In Phase 2, due to the difference in error modes,

F2. at the time when the (error-passive) victim's TEC increases, a message with the same ID will be successfully transmitted by some ECU on the bus.

F1 indicates a bus-off attack. We first simulated the probability of F1 under an

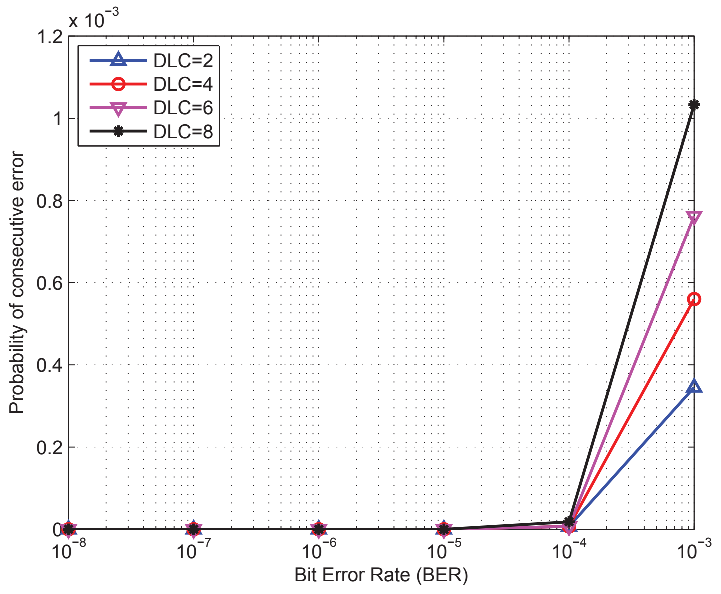


Figure 2.20: Probability of two consecutive errors.

uncompromised condition. We leveraged the same error model in [77] where bit error occurrences follow a Bernoulli distribution. In our simulation, for a given DLC and Bit Error Rate (BER), we randomly generated 100,000 different CAN messages and measured how many of them satisfy F1. The simulation result, plotted in Fig. 2.20, shows that even under an unusually high BER of 10^{-3} (usually $10^{-5} \sim 10^{-7}$ [77]), the maximum probability of F1 was only 0.11%. So, under a normal condition, the probability of F1 occurring 16 times in a bursty manner can be considered 0, whereas it was 1.0 during a bus-off attack. Considering this large discrepancy, we can consider F1 to indicate a bus-off attack. However, since F1 can also occur due to severe system errors such as improper bus termination [17], bit flip [85], and bit drop/insertion [67], F1 alone cannot be a definitive evidence of a bus-off attack.

F2 is the evidence. After occurrence of F1, once an error-passive ECU experiences a bit error again when transmitting a message with ID=$\mathbb{M}$, it can further monitor the CAN bus and check if there was any successful transmission of another message with the same ID=$\mathbb{M}$, i.e., occurrence of F2. This can only occur when two or more ECUs are sending the same message at the same time, which is not allowed on CAN and thus infeasible even

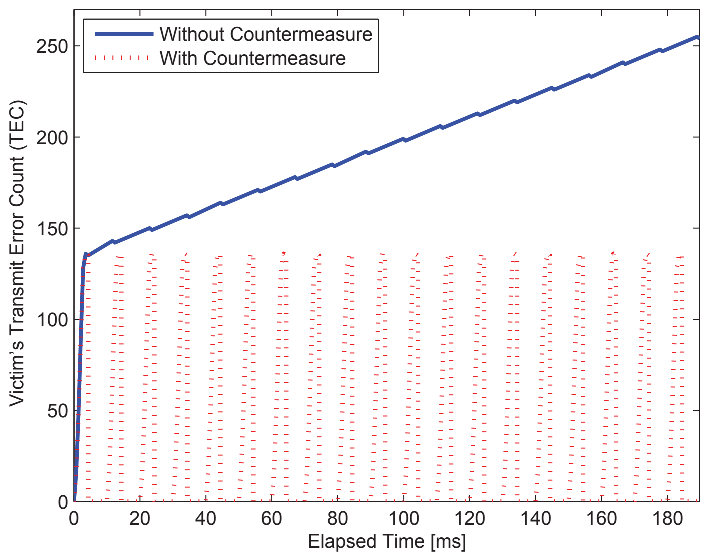


Figure 2.21: Efficiency of the proposed countermeasure.

in a severely erroneous network, while it is possible under a bus-off attack. From both F1 and F2, we can thus verify the occurrence of a bus-off attack, i.e., the observed symptoms are not caused by a system error. Following this reasoning, we propose the following defense mechanism: an ECU or its error counters are reset whenever F2 is observed after N consecutive error frames. We use $N = 16$ to reset the victim ECU upon occurrence of both F1 and F2.

Efficiency of the proposed countermeasure. We evaluated the bus-off attack on our CAN bus prototype with the proposed countermeasure. According to the proposed defense, the nodes were programmed to reset when F2 was observed after 16 consecutive and bursty error frames. During an iterative bus-off attack, Fig. 2.21 shows how the victim's TEC changed with and without the proposed defense. By observing the presence of bursty error frames on the CAN bus, the victim's error mode mostly stayed as error-active, which is in sharp contrast to the case without any countermeasure, and also successfully transmitted its messages, efficiently preventing the bus-off attack.

Alternative countermeasures. In our proposed countermeasure, we looked for consecutive error *frames* (i.e., F1) to prevent the bus-off attack. We may also consider a countermeasure which verifies if consecutive errors incur at the same *bit* position, instead of *frames*. As this is, in fact, the strongest evidence of the bus-off attack, we may use it as the condition of triggering an ECU reset. It may also be considered as a new rule in CAN for *not* increasing the error counters. Then, as the detection of a bus-off attack can be made much faster with the stronger evidence, unwanted retransmissions, which were at least 16 in our proposed countermeasure, can be reduced further. Note that these retransmissions may affect other messages in meeting their hard/soft deadlines. Albeit effective, such a countermeasure accompanies numerous challenges and limitations: 1) detecting individual bit errors would require changes in hardware, thus incurring expensive development cost and 2) detecting which bit has currently changed and comparing it with the previously changed one would require additional memory in the CAN transceiver chip.

The proposed bus-off attack exploits the *periodic* feature of in-vehicle network messages for synchronizing its transmission time with the victim's. It is thus limited to only periodic messages but it is very effective since most in-vehicle messages are periodic. Therefore, an alternative countermeasure can be to transmit periodic messages with some random factors added to their Tx times (e.g., adding random jitters), making them somewhat aperiodic. However, adding random factors to Tx times can create serious unexpected problems such as priority inversion, message sequence inversion, and deadline violation [37]. As this incurs detrimental effects on the message scheduling mechanism, we did not consider it as a possible countermeasure against the bus-off attack.

2.8 Discussion

Severity of the bus-off attack. Since contemporary in-vehicle networks are not equipped with security mechanisms, an adversary can mount not only the bus-off attack but also other types of (previously covered) attacks on in-vehicle networks. For example, the attacker can simply inject arbitrary messages on the CAN bus or monopolize the network by continuously sending the highest-priority frames. However, the following facts make the bus-off attack a more severe problem and an attack that an adversary might favor over other attacks. Previously demonstrated/covered attacks either show obvious misuse of message IDs or significantly increase the message frequency, which can easily be detected and then removed by existing IDSs [45, 56, 65]. In contrast, the proposed bus-off attack can be mounted without misusing IDs and requires only a small increase in message frequency — up to a frequency that renders the bus-off attack feasible even during a system error (e.g., bit drop/insertion). This may cause the existing IDSs to be confused whether the symptom is due to an attack or a system error. More importantly, unlike previously demonstrated attacks, the adversary can mount a bus-off attack without any knowledge on the meanings or purposes of messages, making their reverse-engineering unnecessary. Also, the attacker succeeds before the victim verifies a message's checksum, and hence can use an arbitrary checksum in the attack message, i.e., no need to reverse-engineer the implemented checksum algorithm. Even if MACs were used for in-vehicle network messages, their functionalities could likewise be nullified. Requiring far less painstaking (than reverse engineering) efforts — especially when the adversary wants to mount attacks on various types of vehicles — differentiates the bus-off attack from previously known vehicle attacks and can also be a strong motivation for the adversary to prefer the bus-off attack to others.

Limitations of the bus-off attack. In order to succeed in the bus-off attack, the attacker must synchronize its transmission timing with the victim's. To achieve this, the proposed bus-off attack leverages the CAN protocol's buffering strategy via a genuine/fabricated preceded ID message. As mentioned before, however, its exploitation is only feasible when the target message (from the victim) is sent periodically. In other words, the attacker cannot force a victim to bus off when he is sending messages aperiodically. Meanwhile, since most messages in CAN are sent periodically, the attacker can succeed in mounting the bus-off attack on most ECUs in the in-vehicle network.

Vulnerability of other in-vehicle networks. Although most modern vehicles are equipped with CAN, some may deploy other protocols, such as CAN-FD, TTCAN, and FlexRay for more complex operations. Note that CAN-FD is an enhanced version of CAN, providing flexible and higher data rates as well as a larger data field [67]. Since CAN-FD's basic components, arbitration, and error handling all conform to those in CAN, it is also vulnerable to the proposed bus-off attack. What makes CAN-FD more interesting is that an attacker can monitor the newly introduced Extended Data Length (EDL) and Bit Rate Switch (BRS) fields, recognize which messages are sent with high bit rates or large payloads — safety-critical messages — and target them for a bus-off attack.

TTCAN is a session-layer protocol in which its message transmissions are based on a static schedule, time-triggered paradigm to provide Tx determinism [51]. As a result, all ECUs transmit their messages only at their assigned time slots and are periodically synchronized through a broadcast reference message. Since the allocated time slots are predefined in a schedule matrix and are stored in each node, the attacker — having control of a vehicle running TTCAN — is provided with the knowledge of what messages are sent and when, thus making the bus-off attack easier.

In FlexRay, which is designed to be more reliable than CAN as in CAN-FD, the error modes are divided into 3 different modes: normal-active, normal-passive, and halt. Normal-active mode is essentially equivalent to the error-active mode of CAN, whereas normal-passive mode differs from the CAN's error-passive mode, as it does not allow the nodes to transmit in that mode. Accordingly, FlexRay becomes invulnerable to the proposed bus-off attack.

2.9 Conclusion

In this chapter, we discovered a new vulnerability, called the *bus-off attack*, of in-vehicle networks. The attack exploits their error-handling scheme to disconnect an uncompromised ECU and/or even shut down the entire in-vehicle network. We analyzed its practicability and demonstrated the attack on a CAN bus prototype and two real vehicles. Based on our analysis and experimental results, we proposed a defense mechanism to prevent the bus-off attack. Even though the proposed attack has not yet been seen in the wild, it is easy to mount and also directly related to drivers/passengers' safety, and should thus be countered with high priority. Moreover, the facts that the proposed attack can nullify state-of-the-art solutions and is easy to launch, make it even more important to design and deploy its countermeasures. Thus, we recommend concerted efforts from both academia and industry to account for this vulnerability in the design of in-vehicle networks.

CHAPTER III

Fingerprinting Electronic Control Units for Vehicle Intrusion Detection

3.1 Introduction

Many researchers have been able to demonstrate various types of vehicle attacks, some forcing mass recalls from car OEMs. As a countermeasure against such attacks, defense schemes in the forms of *message authentication* or *intrusion detection* have been proposed. Although these provide a certain level of security, there still remain critical attacks which existing schemes can neither detect nor prevent, for two main reasons: 1) in-vehicle messages do not carry information on their transmitters, and thus one cannot tell whether they originate from genuine transmitters; and 2) lack of the transmitters' information makes it very difficult or impossible for state-of-the-art IDSs to identify which ECU has mounted an attack.

To overcome these limitations and defend against various vehicle attacks, we propose a new anomaly-based IDS, called *Clock-based IDS* (`CIDS`). The need of `CIDS` for vehicles is motivated through an analysis of three representative in-vehicle network attacks — fabrication, suspension, and masquerade attacks. Our analysis shows that state-of-the-art IDSs are insufficient, especially in detecting the masquerade attack due to the absence of the transmitters' information in messages. `CIDS` overcomes these limitations of existing

IDSs by fingerprinting in-vehicle ECUs. Researchers have proposed various schemes for fingerprinting network devices by estimating their clock skews through the timestamps carried in their control packet headers [47, 49, 73, 90]. However, since such embedded timestamps are not available for in-vehicle networks making them inapplicable, `CIDS` fingerprints in-vehicle ECUs in a very different way.

`CIDS` monitors the intervals of (commonly seen) periodic in-vehicle messages, and then exploits them to estimate the clock skews of their transmitters which are then used to fingerprint the transmitters. That is, instead of assuming or requiring timestamps to be carried in messages for fingerprinting, `CIDS` exploits the periodic feature (seen at receivers) of in-vehicle network messages for fingerprinting transmitter ECUs. This makes `CIDS` invulnerable to attackers who use faked timestamps and thus clock skews — a problem that timestamp-based fingerprinting schemes cannot handle. Based on the thus-obtained fingerprints, `CIDS` constructs a norm model of ECUs' clock behaviors using the Recursive Least Squares (RLS) algorithm and detects intrusions with a Cumulative Sum (CUSUM) analysis. This enables `CIDS` to detect not only attacks that have already been demonstrated or discussed in literature, but also those that are more acute and cannot be detected by state-of-the-art IDSs. Our experimental evaluations show that `CIDS` detects various types of in-vehicle network intrusions with a low false-positive rate of 0.055%. Unlike state-of-the-art IDSs, if an intrusion is detected in `CIDS`, its fingerprinting capability facilitates identification of the (compromised) ECU that mounted the attack. We validate these capabilities of `CIDS` through experimental evaluations on a CAN bus prototype and on real vehicles.

Although `CIDS`'s focus is on securing CAN, its applicability is not limited to it; we also discuss its applicability to other in-vehicle network protocols in Section 3.6. Considering the ubiquity of CAN and its direct relationship with the drivers/passengers' safety, it is critically important to build as capable a CAN bus IDS as possible.

This work makes the following contributions:

- Development of a novel scheme of fingerprinting ECUs by exploiting message periodicity;
- Proposal of `CIDS`, which models the norm behavior of in-vehicle ECUs' clocks based on fingerprints and then detects in-vehicle network intrusions;
- Implementation and validation of `CIDS` on a CAN bus prototype as well as on 3 real vehicles.

The rest of the chapter is organized as follows. Section 3.2 provides the necessary background of CAN and IDS-related work, and Section 3.3 details the attack model we consider. Section 3.4 details the design of `CIDS`, which is evaluated in Section 3.5 on a CAN bus prototype as well as on three real vehicles. Section 3.6 discusses `CIDS` further, such as its overhead and extension to emerging in-vehicle networks. Finally, we conclude the chapter in Section 3.7.

3.2 Background

For completeness, we first provide the necessary background on the CAN protocol that is yet unexplained but relevant to `CIDS`. Then, we discuss the related work on security solutions for in-vehicle networks.

3.2.1 Synchronization in CAN

Each ECU broadcasts periodic (or occasionally sporadic) frames on the CAN bus to deliver retrieved sensor data. For proper bitwise message transmission and reception, hard and soft *bit* synchronizations are achieved, respectively, by using the Start-of-Frame (SOF) signal and bit stuffing in CAN frames [2]. Although these provide alignment of bit edges for message exchange, they do not synchronize the *clocks* of ECUs, i.e., CAN lacks clock synchronization. Thus, since time instants for ECUs are provided by their own quartz crystal clocks, these clocks, in reality, run at different frequencies, resulting in random drifting of clocks: a drift of 2400ms over a period of 24 hours is possible [61].

3.2.2 Deficiency of State-of-the-art Defense Schemes

In the area of Internet security, cryptographic message authentication provides strong protection against forgery. Thus, researchers have attempted to borrow such approaches from the domain of Internet security to address in-vehicle network security problems. Although such preventive measures provide some degree of security, they alone cannot guarantee complete security due to their inability to handle certain critical attacks, e.g., Denial-of-Service (DoS). Moreover, their operations not only require a significant amount of processing power but also increase message latencies and bus utilization. Since in-vehicle networks must operate in real time and ECUs are resource-limited for cost reasons, unlike in the Internet, these "costs" of preventive measures hinder their adoption [65]. More importantly, when an adversary has full access to any data stored in RAM and/or FLASH, including data used for implementing security mechanisms (e.g., shared secret keys), some cryptographic solutions become incapable [57].

To overcome such limitations of preventive measures, different IDSs have been proposed. Although existing IDSs are capable of detecting most attacks, they fail to cover some critical attacks which are more acute, and thus are not sufficient to provide security. We will elaborate on such shortcomings of state-of-the-art IDSs while analyzing the attack scenarios under consideration in Section 3.3.3.

3.3 Attack Model

We first discuss the adversary model under consideration, and then the three representative attack scenarios.

3.3.1 Adversary Model

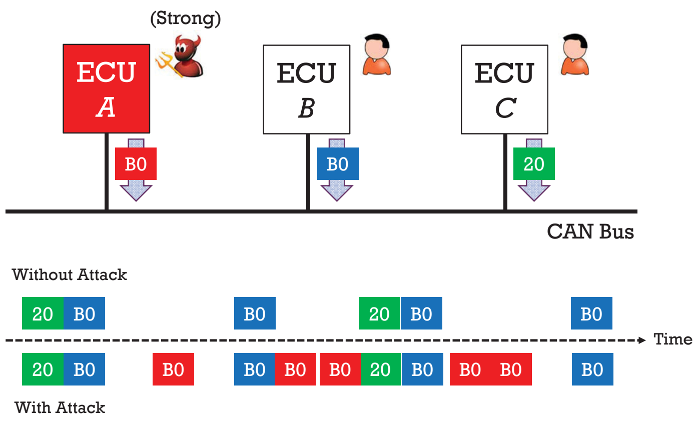
Adversaries can physically/remotely compromise more than one in-vehicle ECU via numerous attack surfaces and means [31]. We consider an adversary who wants to manipulate or impair in-vehicle functions. The adversary can achieve this by either injecting arbitrary messages with a spoofed ID into the in-vehicle network, which we refer to as *attack messages*, or by stopping/suspending message transmissions of the compromised ECU.

Strong and weak attackers. Depending on their hardware, software, and attack surfaces, ECUs of different vehicles have different degrees of vulnerabilities, thus providing attackers different capabilities. So, we consider two different types of compromised ECUs: *fully* and *weakly* compromised ECUs.

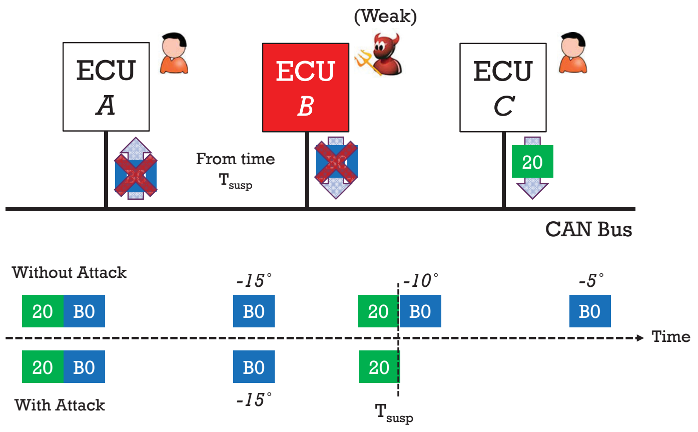
Through a *weakly* compromised ECU, the attacker is assumed to be able to stop/suspend the ECU from transmitting certain messages or keep the ECU in listen-only mode, but cannot inject any fabricated messages. We call such an attacker with limited capabilities a *weak attacker*, and will use this term interchangeably with "weakly compromised ECU".

In contrast, with a *fully* compromised ECU, the attacker is assumed to have full control of it and access to memory data. Thus, in addition to what a weak attacker can do, the attacker controlling a fully compromised ECU can mount attacks by injecting arbitrary attack messages. We call such an attacker with more attack capabilities a *strong attacker*, and will use this term interchangeably with a "fully compromised ECU". Even when preventive security mechanisms (e.g., MAC) are built into the ECUs, since the strong attacker has full access to any data stored in their memory, including data used for implementing security mechanisms (e.g., shared secret keys), it can disable them [57]. On the other hand, a weak attacker can only stop, or listen to message transmissions, but cannot start a new one.

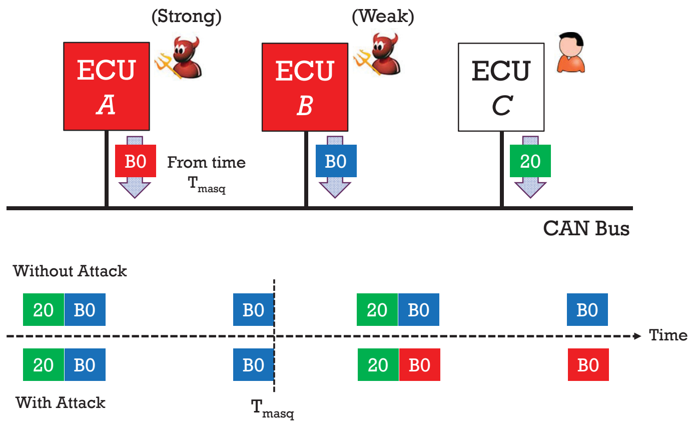
Foster *et al.* [41] have recently proved the possible existence of these two types of attackers in in-vehicle networks. They have shown that the firmware versions of telematics units can affect/limit the attacker's capabilities in injecting and monitoring in-vehicle network messages. Specifically, for a certain firmware version of the telematics unit, an attacker having control of that ECU was shown to be able to receive CAN messages



(a) Fabrication attack.



(b) Suspension attack.



(c) Masquerade attack.

Figure 3.1: Three representative attack scenarios on in-vehicle networks.

but unable to send the messages. On the other hand, for some other firmware versions, the attacker was capable of both sending and receiving CAN messages to and from the in-vehicle network. In other words, the firmware version of an ECU determines which type of an attacker — strong or weak — it can become, if compromised.

To further comprehend how and why these two different types of attackers can exist, let's consider one of the most common CAN controllers, Microchip MCP2515 [10]. For ECUs with such a controller, various operation modes like configuration, normal, and listen-only can be selected by user instructions through the Serial Peripheral Interface (SPI). Thus, user-level features for configuring the CAN controller allow attackers to easily enter different modes (e.g., listen-only mode for a weak attacker). In contrast, there are no such features allowing attackers to easily inject forged messages. In other words, the specification of the ECU hardware/software, if compromised, can *restrict* the adversary to become a weak attacker only. Note that the required functionalities of a strong attacker subsume those of a weak attacker. It is thus easier for an adversary to become a weak attacker than a strong attacker, let alone researchers have already demonstrated how to create such a strong attacker [31, 50, 56, 57].

3.3.2 Attack Scenarios

Based on the adversary model discussed so far, we consider the following attack scenarios that can severely impair in-vehicle functions: *fabrication*, *suspension*, and *masquerade*.[1]

Fabrication attack. Through an in-vehicle ECU compromised to be a strong attacker, the adversary fabricates and injects messages with forged ID, DLC, and data. The objective of this attack is to override any periodic messages sent by a legitimate safety-critical ECU so that their receiver ECUs get distracted or become inoperable. For example, as shown in Fig. 3.1a, the strong attacker $\mathbb{A}$ injects several attack messages with ID=0xB0, which is usually sent by a legitimate ECU $\mathbb{B}$, at a high frequency. Thus, other nodes which normally receive message 0xB0 are forced to receive the fabricated attack messages more often than the legitimate ones. We refer to such a case as $\mathbb{A}$ mounting a fabrication attack

[1] In this dissertation, we focus on only these three attack scenarios and do not consider others as they may be less feasible or harmful, or be detectable by existing IDSs.

on message 0xB0 or its genuine transmitter $\mathbb{B}$. Demonstrated attacks such as controlling vehicle maneuver [50] and monopolizing the CAN bus with highest priority messages [45] exemplify a fabrication attack.

Suspension attack. To mount a suspension attack, the adversary needs only one weakly compromised ECU, i.e., become a weak attacker. As one type of Denial-of-Service (DoS) attack, the objective of this attack is to stop/suspend the weakly compromised ECU's message transmissions, thus preventing the delivery/propagation of information it acquired, to other ECUs. For some ECUs, they must receive certain information from other ECUs to function properly. Therefore, the suspension attack can harm not only the (weakly) compromised ECU itself but also other receiver ECUs. An example of this attack is shown in Fig. 3.1b where the weak attacker having control of the Electric Power Steering ECU $\mathbb{B}$ stops transmitting its measured steering wheel angle value. So, the Electronic Stability Control (ESC) ECU $\mathbb{A}$, which requires the steering wheel angle value from $\mathbb{B}$ for detecting and reducing the loss of traction, no longer receives its updates and thus malfunctions.

Masquerade attack. To mount a masquerade attack, the adversary needs to compromise two ECUs, one as a strong attacker and the other as a weak attacker. The objective of this attack is to manipulate an ECU, while shielding the fact that an ECU is compromised. Fig. 3.1c shows an example where the adversary controls a strong attacker $\mathbb{A}$ and a weak attacker $\mathbb{B}$. Until time T_{masq}, the adversary monitors and learns which messages are sent at what frequency by its weaker attacker, e.g., $\mathbb{B}$ sends message 0xB0 every $20ms$. Since most in-vehicle network messages are periodic and broadcast over CAN, it is easy to learn their IDs and intervals. Once it has learned the ID and frequency of a message, at time T_{masq}, the adversary stops the transmission of its weak attacker and utilizes its strong attacker $\mathbb{A}$ to fabricate and inject attack messages with ID=0xB0. Stopping $\mathbb{B}$'s transmission and exploiting $\mathbb{A}$ for transmission of attack messages are to overcome the weak attacker's inability of injecting messages. After T_{masq}, the original transmitter of 0xB0, $\mathbb{B}$, does not send that message any longer, whereas $\mathbb{A}$ sends it *instead* at its original frequency. So, when the CAN bus traffic is observed, the frequency of message 0xB0 remains the same, whereas its transmitter has changed. We refer to such a case as $\mathbb{A}$ mounting a masquerade attack on message 0xB0 or its original transmitter $\mathbb{B}$.

In fact, in order to attack and remotely stop a Jeep Cherokee running on a highway, Miller *et al.* [58] had to control its ABS collision prevention system by mounting a *masquerade* (not fabrication) attack. In contrast to other vehicles which they had previously examined (e.g., Toyota Prius), the Jeep Cherokee's brake was not controllable via the fabrication attack as its ABS collision prevention system, which was the attack vector for engaging brakes, was switched off when the fabrication attack was mounted. On the other hand, when mounting the masquerade attack, the system was not switched off, thus allowing them to control the Jeep Cherokee's braking maneuver.

Using masquerade attacks, the adversary can not only inject attack messages from the compromised/impersonating ECU but also cause other severe problems, significantly degrading the in-vehicle network performance. The impersonating ECU sending a message instead of another ECU implies that it would generate more messages to periodically transmit than before, making its transmit buffer more likely overloaded. This may, in turn, lead to severe consequences, such as non-abortable transmission requests [37], deadline violation [48], and significant priority inversion [68]. Moreover, the original sequence of messages may also change, thus failing to meet the requirement of some in-vehicle messages to be sent sequentially in a correct order for proper vehicle operations. These network problems from a masquerade attack occur while not deviating much from the norm network behavior (e.g., message frequency remains the same). This is in sharp contrast to the cases of mounting a fabrication attack or a suspension attack, which may also incur similar problems. Such problems have been identified to be critical since they degrade the real-time performance of CAN significantly, and thus undermine vehicle safety [37,48,68]. The masquerade attack can thus cause more problems to the in-vehicle network than just injecting attack messages.

3.3.3 Defense Against the Attacks

When the fabrication or suspension attack is mounted, the frequency of certain messages significantly and abnormally increases or decreases, respectively. Thus, if state-of-the-art IDSs [45, 56, 65, 66], which monitor the message frequencies, were to be used, the attacks can be detected.

When mounting the masquerade attack, however, the adversary does not change the original frequency of messages. Thus, the adversary may use this attack to evade state-of-the-art IDSs. Moreover, if the adversary does not change the content of messages as well, it can behave like a legitimate ECU. However, the adversary may later mount other types of attacks (e.g., a fabrication attack) through the impersonating ECU. Hence, defending against the masquerade attack implies not only detecting the attack reactively, but also preventing other attacks *proactively*.

3.4 Clock-Based Detection

Although state-of-the-art IDSs are capable of detecting some basic attacks such as fabrication attack and suspension attack, they fail to detect more sophisticated ones such as the masquerade attack for the following reasons.

- *No authenticity* — CAN messages lack information on their transmitters. So, existing IDSs do not know whether or not the messages on the CAN bus were sent by the genuine transmitter, and hence cannot detect any changes of the message transmitter.
- *Inability of identifying a compromised ECU* — Lack of the transmitter's information makes it very difficult or impossible for state-of-the-art IDSs to identify which of compromised ECUs mounted an attack.

If CAN frames do not carry any information on their transmitters, how could an IDS identify them and detect intrusions such as the masquerade attacks? Which behavior of CAN should the IDS model for detection of such intrusions? We answer these questions by developing a novel IDS, `CIDS`, which exploits message frequency to fingerprint the transmitter ECUs, and models a norm behavior based on their fingerprints for intrusion detection. We focus on detecting intrusions in *periodic* messages as most in-vehicle messages are sent periodically [68, 78].

3.4.1 Fingerprinting ECUs

For each in-vehicle ECU in CAN, the time instants of periodic message transmissions are determined by its quartz crystal clock [61]. We follow the nomenclature of clocks of the NTP specification [59] and Paxson [74]. Let $\mathbb{C}_{true}$ be a "true" clock which reports the true time at any moment and $\mathbb{C}_i$ be some other non-true clock. We define the terms "clock offset, frequency, and skew" as follows.

- **offset:** difference in the time reported by clock $\mathbb{C}_i$ and the true clock $\mathbb{C}_{true}$. We define *relative offset* as the offset between two non-true clocks.
- **frequency:** the rate at which clock $\mathbb{C}_i$ advances. Thus, the frequency at time t is $\mathbb{C}_i'(t) \equiv d\mathbb{C}_i(t)/dt$.
- **skew:** difference between the frequencies of clock $\mathbb{C}_i$ and the true clock $\mathbb{C}_{true}$. We define *relative skew* as the difference in skews of two non-true clocks.

If two clocks have relative offset and skew of 0, then they are said to be *synchronized*. Otherwise, we consider they are *unsynchronized*. Since CAN lacks clock synchronization, it is considered to be unsynchronized.

Clock skew as a fingerprint. The clock offsets and skews of unsynchronized nodes depend solely on their local clocks, thus being distinct from others. As others have also concluded [47, 49, 90], clock skews and offsets can therefore be considered as fingerprints of nodes. Various studies have exploited this fact to fingerprint physical devices [47, 49, 73, 90]. However, they are not applicable to our problem as they exclusively rely on the timestamps carried in the packet headers, which are, as discussed before, not available in in-vehicle networks. Kohno *et al.* [49] considered an alternative to embedded timestamps:

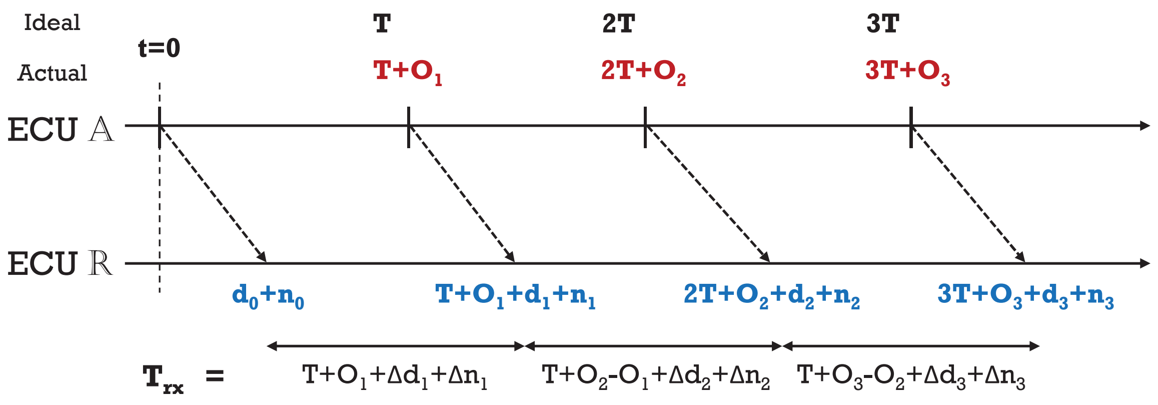


Figure 3.2: Timing analysis of message arrivals.

using Fourier Transform for clock skew estimation. However, as their approach relies on the unique characteristics of the Internet (e.g., multi-hop delays, large network topology), it cannot be directly applied to in-vehicle networks.

To build an effective IDS, which can detect various types of attack including the masquerade attack, it should be capable of verifying the transmitter of each message. However, since such information is not present in CAN messages, we must fingerprint ECUs with other “leaked” information. Unlike the existing approaches that exploit embedded timestamps, we exploit *message periodicity* to extract and estimate the transmitters’ clock skews, which are then used to fingerprint the transmitter ECUs.

Clock skew estimation. Consider an ECU $\mathbb{A}$ which broadcasts a message every T *ms* and an ECU $\mathbb{R}$ which periodically receives that message. From the perspective of $\mathbb{R}$, since only its timestamp is available, we consider its clock as the true clock. As shown in Fig. 3.2, due to the clock skew, periodic messages are sent at times with small offsets from the ideal values (e.g., T, $2T$, $3T$, $\cdots$). Let $t = 0$ be the time when the first message was sent from $\mathbb{A}$, and O_i be the clock offset of $\mathbb{A}$ when it sends the i-th message since $t = 0$. Then, after a network delay of d_i, ECU $\mathbb{R}$ would receive that message and put an arrival timestamp of $iT + O_i + d_i + n_i$, where n_i denotes the noise in $\mathbb{R}$’s timestamp quantization [90]. Thus, the intervals between each arrival timestamp, $T_{rx,i} = T + \Delta O_i + \Delta d_i + \Delta n_i$, where ΔX_i denotes the difference of X between step i and $i-1$, and $O_0 = 0$. Since the change in O_i within one time step is negligible and n_i is a zero-mean Gaussian noise term [11], the expected value of the timestamp intervals, $\mu_{T_{rx}} = \mathrm{E}[T_{rx,i}]$, can be expressed as:

$$\begin{aligned}\mu_{T_{rx}} &= \mathrm{E}[T + \Delta O_i + \Delta d_i + \Delta n_i] \\ &= T + \mathrm{E}[\Delta O_i + \Delta d_i + \Delta n_i] \\ &\approx T,\end{aligned} \tag{3.1}$$

where the second equality holds since T is a pre-determined constant. Since the data lengths of CAN periodic messages, i.e., DLCs, are constant over time, for now, we consider $\mathrm{E}[\Delta d_i] = 0$. Later in Section 3.4.4, we will discuss the case when d_i is not constant, and how it may affect the performance of `CIDS`.

Based on the arrival timestamp of the first message, $d_0 + n_0$, and the average of timestamp intervals, $\mu_{T_{rx}}$, we extrapolate and determine the *estimated* arrival time of the i-th message as $i\mu_{T_{rx}} + d_0 + n_0$, whereas the actual *measured* arrival time is $iT + O_i + d_i + n_i$. As we are estimating subsequent arrival times, $\mu_{T_{rx}}$ is determined by past measurements. Since T is constant over time and thus again $\mu_{T_{rx}} \approx T$, the average difference between the estimated and measured times is:

$$\mathrm{E}[\mathbb{D}] = \mathrm{E}[i(T - \mu_{T_{rx}}) + O_i + \Delta d + \Delta n] \approx \mathrm{E}[O_i]. \tag{3.2}$$

That is, from message periodicity, we can estimate the *average clock offset*, $\mathrm{E}[O_i]$, which will indeed be distinct for different transmitters. Since clock offset is slowly varying and non-zero [47, 90], $\mathrm{E}[O_i] \neq 0$, whereas $\mathrm{E}[\Delta O_i] = 0$.

If ECU $\mathbb{R}$ were to determine the average clock offset for every N received messages, since it is derived in reference to the first message (of N messages), it represents only the average of *newly* incurred offsets. Thus, to obtain the total amount of incurred offset, which we call the *accumulated clock offset*, the absolute values of the average clock offsets have to be summed up. By definition, the slope of the accumulated clock offset would

Algorithm 1 Clock skew estimation with RLS

```
1:  Initialize: $S[0] = 0$, $P[0] = \delta I$
2:  function SkewUpdate($t, e$)                                    ▷ RLS algorithm
3:      $G[k] \leftarrow \frac{\lambda^{-1}P[k-1]t[k]}{1+\lambda^{-1}t^2[k]P[k-1]}$
4:      $P[k] \leftarrow \lambda^{-1}(P[k-1] - G[k]t[k]P[k-1])$
5:      return $S[k] \leftarrow S[k-1] + G[k]e[k]$
6:  end function
7:  for $k^{th}$ step do
8:      $a_0 \leftarrow$ arrival timestamp of most recently rxed message
9:      $n \leftarrow 1$
10:     while $n \le N$ do
11:         if current time $\gg a_{n-1}$ then
12:             /* No longer receives the message */
13:             $a_n, \cdots, a_N \leftarrow$ significantly high values
14:             $T_n, \cdots, T_N \leftarrow$ significantly high values
15:             break
16:         else
17:             $a_n \leftarrow$ arrival timestamp of $n^{th}$ message
18:             $T_n \leftarrow a_n - a_{n-1}$                             ▷ Timestamp interval
19:             $n \leftarrow n + 1$
20:         end if
21:     end while
22:     $\mu_T[k] \leftarrow \frac{1}{N}\sum_{i=1}^{N} T_i$                  ▷ Avg. timestamp interval
23:     $O[k] \leftarrow \frac{1}{N-1}\sum_{i=2}^{N} a_i - (a_1 + (i-1)\mu_T[k-1])$
24:     $O_{acc}[k] \leftarrow O_{acc}[k-1] + |O[k]|$                      ▷ Accumulated offset
25:     $e[k] \leftarrow O_{acc}[k] - S[k-1]t[k]$                          ▷ Identification error
26:     $S[k] \leftarrow$ SkewUpdate($t, e$)                               ▷ Clock skew
27: end for
```

thus represent the clock skew, which is constant as we will show and as others have also concluded [49, 63, 74, 87]. This enables `CIDS` to estimate the clock skew from arrival timestamps and thus fingerprint the message transmitter for intrusion detection. We will later show, via experimental evaluations on a CAN bus prototype and on 3 real vehicles, that the thus-derived clock skew is indeed a fingerprint of an in-vehicle ECU.

3.4.2 CIDS — Per-message Detection

By determining the clock skew from observation of message intervals, transmitter ECUs can be fingerprinted. We exploit this in designing `CIDS`, a clock-based IDS for in-vehicle networks which detects intrusions in two different ways: *per-message* detection and *message-pairwise* detection, where the latter supplements the former in reducing false positive/negative results. Next, we first discuss per-message detection and then pairwise detection.

Modeling. For a given message ID, `CIDS` derives the accumulated clock offset inherent in the arrival timestamps. Since clock skew is constant, the accumulated clock offset is linear in time, and hence `CIDS` describes it as a linear regression model. A linear parameter identification problem is thus formulated as:

$$O_{acc}[k] = S[k] \cdot t[k] + e[k], \tag{3.3}$$

where at step k, $O_{acc}[k]$ is the accumulated clock offset, $S[k]$ the regression parameter, $t[k]$ the elapsed time, and $e[k]$ the identification error. The regression parameter S represents the slope of the linear model and thus the *estimated clock skew*. The identification error, e, represents the residual which is not explained by the model. In `CIDS`, O_{acc}, S, t, and e are updated every N messages, i.e., kN messages are examined up to step k.

To determine the unknown parameter S, we use the Recursive Least Squares (RLS) algorithm [43], which uses the residual as an objective function to minimize the sum of squares of the modeling errors. Hence, in RLS, the identification error skews towards 0, i.e., has 0 mean. We will discuss the computational overhead of RLS as well as other possible solutions in Section 3.6.

Algorithm 1 describes how the clock skew is estimated using RLS. First, `CIDS` measures the arrival times and their intervals of N messages for a given ID. If the intended message has not been received for a long time — possibly due to suspension attack — as in line 13–14, `CIDS` sets the remaining timestamp and interval values significantly higher. Once N values are measured, `CIDS` determines the accumulated clock offset and accordingly, the identification error. Based on the thus-derived value, the gain, G, and the covariance, P, are updated with RLS for identifying the regression parameter S, i.e., estimate clock skew.

This procedure of clock skew estimation continues iteratively during the operation of `CIDS` and, if uncompromised, outputs an identification error skewed towards 0 and a constant clock skew. This way, the *norm clock behavior* of the transmitter can be described as a linear model with the clock skew being the slope. In RLS, a forgetting factor, λ, is used to give exponentially less weights to older samples and thus provide freshness. In `CIDS`, we set λ=0.9995.

Detection. For a given message ID, `CIDS` runs RLS for clock skew estimation, constructs a norm model on clock behavior, and verifies whether there are any abnormal measurements deviating from it, i.e., intrusions.

Consider a fabrication attack in which the adversary injects an attack message with ID=0x01, which is originally sent every 10*ms* by some ECU. The fabrication attack significantly increases the absolute average difference between the estimated and measured arrival times of 0x01. As a result, due to a sudden increase in the rate at which the accumulated clock offset changes, a high identification error results. Similarly, when the suspension attack is mounted, the absolute average difference also increases and thus a high error is also incurred. When a masquerade attack is mounted, since the adversary sends the message through a different ECU than its original one, the increase rate of accumulated clock offset, i.e., clock skew, suddenly changes and also results in a high identification error. In summary, unlike when the mean of identification error should usually skew towards 0, which is the norm clock behavior, its mean suddenly shifts towards a high non-zero value when there is an intrusion.

`CIDS` exploits the Cumulative Sum (CUSUM) method, which derives the cumulative sums of the deviations from a target value to detect sudden shifts. Since it is cumulative, even minor drifting from the target value leads to steadily increasing or decreasing cumulative values. It is therefore optimal in detecting small persistent changes and is widely used for change-point detection [28]. `CIDS` detects intrusions via CUSUM as follows. At each step of clock skew estimation, `CIDS` updates the mean and variance of the identification errors (e), μ_e and σ_e^2, respectively. In `CIDS`, these values represent the CUSUM target values of e (i.e., norm clock behavior), and thus require proper tracking. Hence, as a precaution of abnormal values incurring from an attack to be reflected into the target values, μ_e and σ_e^2 are updated only if $|\frac{e-\mu_e}{\sigma_e}| < 3$. Then, per derived identification error e, the upper and lower control limits of CUSUM, L^+ and L^- are updated as [88]:

$$\begin{aligned} L^+ &\leftarrow \max\left[0, L^+ + (e-\mu_e)/\sigma_e - \kappa\right] \\ L^- &\leftarrow \max\left[0, L^- - (e-\mu_e)/\sigma_e - \kappa\right] \end{aligned} \tag{3.4}$$

where κ is a parameter reflecting the number of standard deviations `CIDS` intends to detect. Note that κ can be learned offline, or by monitoring normal in-vehicle traffic. If either of the control limits, L^+ or L^-, exceeds a threshold Γ_L, a sudden positive or negative shift in value has been detected, respectively, and thus `CIDS` declares it as an intrusion. As the general rule of thumb for CUSUM is to have a threshold of 4 or 5 [62], we set $\Gamma_L = 5$.

3.4.3 CIDS — Message-pairwise Detection

In addition to per-message detection, `CIDS` also alarms intrusions via message-pairwise detection, which examines the *correlation* between the average clock offsets in two periodic messages. Consider two messages M_1 and M_2 periodically sent by an ECU $\mathbb{A}$. Since these messages originate from the same transmitter, their instantaneous average clock offsets are likely equivalent. Thus, the correlation coefficient, ρ, between their average clock offsets (derived per step) would show a high value close to 1, i.e., correlated. On the other hand, if the two messages were sent by different ECUs, $\rho \simeq 0$, i.e., uncorrelated.

Modeling and detection. If clock offsets in two messages are highly correlated ($\rho > 0.8$), their relationship can be linear. So, `CIDS` describes them as a linear regression model: $O_{M_2}[k] = \alpha O_{M_1}[k] + e_{corr}[k]$, where O_{M_i} denotes the average clock offset of message M_i at step k, α the regression parameter, and $e_{corr}[k]$ the identification error. As per-message detection, message-pairwise detection is also based on a linear model.

Thus, we apply the same detection method, CUSUM. Since message-pairwise detection seeks intrusions from a different perspective than per-message detection, it reduces false positive/negative results. Note, however, that message-pairwise detection is only applicable when two messages' clock offsets are highly correlated, whereas per-message detection is applicable to any periodic message. Moreover, albeit effective, it requires pairwise computations. Therefore, we use message-pairwise detection as an *optional* feature of CIDS. We will later show via experimental evaluations how message-pairwise detection further improves the performance of CIDS.

3.4.4 Verification

To reduce possible false positives/negatives, CIDS also performs a verification process. Suppose that a possible intrusion was alarmed due to a high identification error when verifying message V_i, the i-th message of V. Although such a high error can be due to an intrusion, it can also be due to an incorrect computation of average clock offset. In Section 3.4.1, we considered $\mathrm{E}[\Delta d_i] = 0$ and could thus extract and determine the average clock offset. Although this is true in most cases, occasionally $\mathrm{E}[\Delta d_i] \neq 0$, which affects the accuracy of deriving the true clock offset and thus the detection result. In CAN, $\mathrm{E}[\Delta d_i] \neq 0$ only occurs if the transmission of V_i was delayed due to the bus being busy or its transmitter losing arbitration when attempting to send V_i. Note that the latter also results in the bus being busy before the transmission/reception of V_i. Thus, CIDS also checks if the possibility of $\mathrm{E}[\Delta d_i] \neq 0$ is the main cause of a (possibly false) alarm of intrusion by verifying whether the CAN bus was busy right before receiving V_i. This way, CIDS enhances its detection accuracy. However, as discussed before, usually $\mathrm{E}[\Delta d_i] = 0$ in an actual CAN bus due to its high speed, its messages having short lengths, and low bus load. In other words, the nature of CAN bus communication helps CIDS reduce false positives/negatives.

3.4.5 Attacker Identification

When an intrusion is detected for some message ID, `CIDS` can also identify which compromised ECU mounted the attack. It can extract the clock skew for that attacked message ID, compare it with other clock skew values extracted from other message IDs, and exploit the comparison result in determining whether they originated from the same transmitter. This way, `CIDS` can at least reduce the scope of ECUs which may (or may not) have mounted the attack, thus facilitating attacker identification

3.5 Evaluation

We now validate that clock skews can be used as fingerprints of transmitter ECUs, and evaluate the performance of `CIDS` on a CAN bus prototype and real vehicles.

CAN bus prototype: Similarly to the set up we had in Chapter 2.5, the CAN bus prototype was configured to operate at a 500Kbps bus speed, interconnecting three CAN nodes. The main differences were how the nodes were programmed. The first node $\mathbb{A}$ was programmed to send messages 0x11 and 0x13 every 50*ms*, and the second node $\mathbb{B}$ to send message 0x55 at the same frequency. The third node $\mathbb{R}$ was programmed to run `CIDS`.

Real vehicle: The 2013 Honda Accord (Fig. 2.17a (left)) that we had used for the bus-off attack evaluation is also used for our experiments of `CIDS` in an isolated and controlled (for safety) environment. Again, via the On-Board Diagnostic (OBD-II) system port [13], we connected our CAN bus prototype nodes — which function as an adversary or `CIDS` — to the in-vehicle network. Through the OBD-II port, the three nodes were able to communicate with real ECUs.

CAN log data: To further validate that `CIDS`'s fingerprinting is applicable to other vehicles, we also refer to CAN traffic data logged from a Toyota Camry 2010 by Ruth *et al.* [78] and data logged from a Dodge Ram Pickup 2010 by Daily [36]. Both data were logged through a Gryphon S3 and Hercules software. In the Toyota Camry 2010, there were 42 distinct messages transmitted on the CAN bus: 39 of them sent periodically at intervals ranging from 10*ms* to 5 seconds, and 3 of them sent sporadically. In the Dodge Ram Pickup 2010, there were 55 distinct messages which were all sent periodically on the CAN bus.

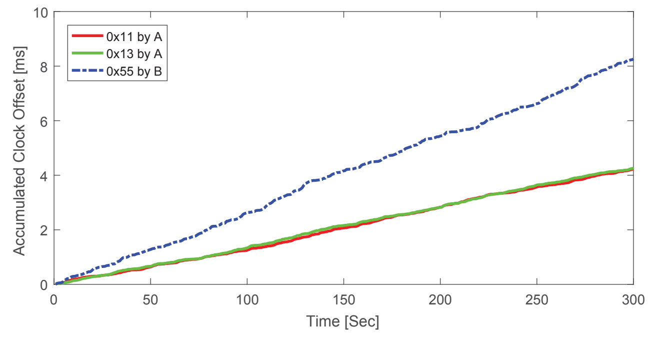
In order to identify which messages originate from the same real ECU and thus exploit it as a ground truth, we used the naive method discussed in [68]. The messages, which originate from the same ECU and have the same preset message interval, were shown to have the same number of transmissions on the bus, when traced for at least a few minutes. Such a method can be an alternative to fingerprinting, but it requires pairwise comparisons and cannot be completed in real time as required in the design of `CIDS`, which is essential for intrusion detection in real in-vehicle networks.

While running `CIDS`, we determined offsets and skews for every 20 received samples, i.e., $N = 20$, and set $\kappa = 5$.

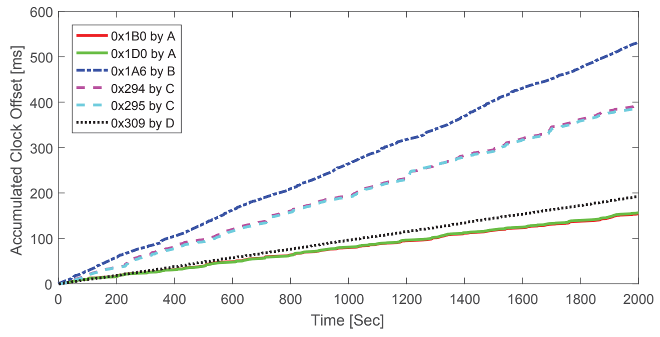
3.5.1 Clock Skew as a Fingerprint

We first evaluate the validity of `CIDS`'s fingerprinting of the transmitter ECUs based on the estimated clock skews. We evaluate skew estimates in microseconds per second ($\mu s/s$) or parts per million (ppm).

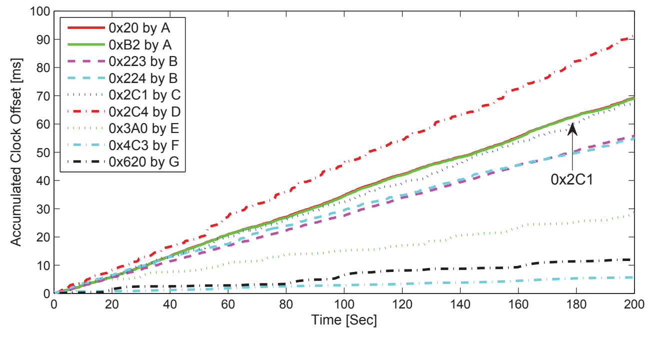
CAN bus prototype. Fig. 3.3a plots our evaluation results of `CIDS`'s fingerprinting on the CAN bus prototype: accumulated clock offsets of messages 0x11, 0x13, and 0x55. Note that the slopes in this figure represent the estimated clock skews. All the derived accumulated clock offsets were found to be linear in time, i.e., constant estimated skews. Messages 0x11 and 0x13, both of which were sent from node $\mathbb{A}$, exhibited the same constant clock skew of 13.4ppm. On the other hand, the message 0x55 sent from a different node $\mathbb{B}$ showed a different clock skew of 27.2ppm. Thus, the clock skews derived by `CIDS` can be used to differentiate ECUs.



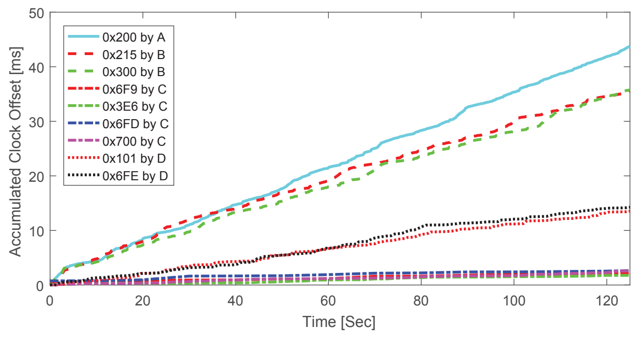
(a) CAN bus prototype.



(b) Honda Accord 2013.



(c) Toyota Camry 2010.



(d) Dodge Ram Pickup 2010.

Figure 3.3: Accumulated clock offsets derived by CIDS in different evaluation settings.

Honda Accord 2013. For CIDS's evaluation on a real vehicle, the CAN prototype nodes logged the in-vehicle CAN traffic of the Honda Accord 2013, and ran CIDS on messages 0x1B0, 0x1D0, 0x1A6, 0x294, 0x295, and 0x309. The approach in [68] was adopted to verify that messages {0x1B0, 0x1D0} were sent from the same ECU, {0x294, 0x295} both sent from another ECU, whereas others were sent from different ECUs. Utilizing these facts, one can conclude from Fig. 3.3b that the clock offsets and the skews derived in CIDS are equivalent *only* for those messages sent from the same ECU; 0x1B0 and 0x1D0 showed a skew of 78.4ppm, 0x294 and 0x295 showed a skew of 199.8ppm, while messages 0x1A6 and 0x309 showed very different skews of 265.7ppm and 95.78ppm, respectively. This result again shows that clock skews between *different* ECUs are distinct and can thus be used as the fingerprints of the corresponding ECUs.

Toyota Camry 2010. To show that the applicability of CIDS's fingerprinting is not limited to the specific vehicle model used, we also conducted experiments on a different vehicle: running CIDS's fingerprinting on the Toyota Camry logged data. Similarly to the real vehicle evaluation in Section 3.5.1, the approach in [68] was used as the ground truth. It was verified that messages {0x20, 0xB2} within the CAN log data were all sent from some ECU $\mathbb{A}$. Also, {0x223, 0x224} were both sent from some ECU $\mathbb{B}$, whereas 0x2C1, 0x2C4, 0x3A0, 0x4C3, and 0x620 were each sent from a different ECU. As shown in Fig. 3.3c, messages 0x20 and 0xB2 both showed a clock skew of approximately 345.3ppm, whereas 0x223 and 0x224 showed a different clock skew of 276.5ppm. 0x2C4, 0x3A0, 0x4C3, and 0x620 showed very different clock skews of 460.1ppm, 142.5ppm, 26.1ppm and 58.7 ppm, respectively.

We made an interesting observation on message 0x2C1, showing a clock skew of 334.1ppm, which was different from the skews of messages {0x20, 0xB2} only by 3%, despite the fact that it was sent by a different ECU. This may confuse CIDS in determining whether they were sent by the same ECU or not. However, in such a case, CIDS can further examine the correlation between clock offsets and can thus fingerprint with a higher accuracy, which we will discuss and evaluate further in Section 3.5.4.

Dodge Ram Pickup 2010. We also ran `CIDS`'s fingerprinting on the CAN log data of a Dodge Ram Pickup 2010. For this vehicle, it was verified that message 0x200 was sent from some ECU $\mathbb{A}$, $\{$0x215, 0x300$\}$ sent from $\mathbb{B}$, $\{$0x6F9, 0x3E6, 0x6FD, 0x700$\}$ sent from $\mathbb{C}$, and $\{$0x101, 0x6FE$\}$ sent from $\mathbb{D}$. Fig. 3.3d shows that `CIDS` determined that 0x200 has a clock skew of 351.7ppm, $\{$0x215, 0x300$\}$ to have approximately 295.3ppm, $\{$0x6F9, 0x3E6, 0x6FD, 0x700$\}$ to have 24.5ppm, and $\{$0x101, 0x6FE$\}$ to have 110.3ppm, thus correctly fingerprinting their transmitters.

These results of a Toyota Camry and a Dodge Ram Pickup CAN log data again affirm the fact that the clock skews derived by `CIDS` are diverse and can indeed be used as fingerprints of in-vehicle ECUs. Moreover, they show that `CIDS`'s fingerprinting is not limited to a specific vehicle model, and can thus be applied to other vehicle models.

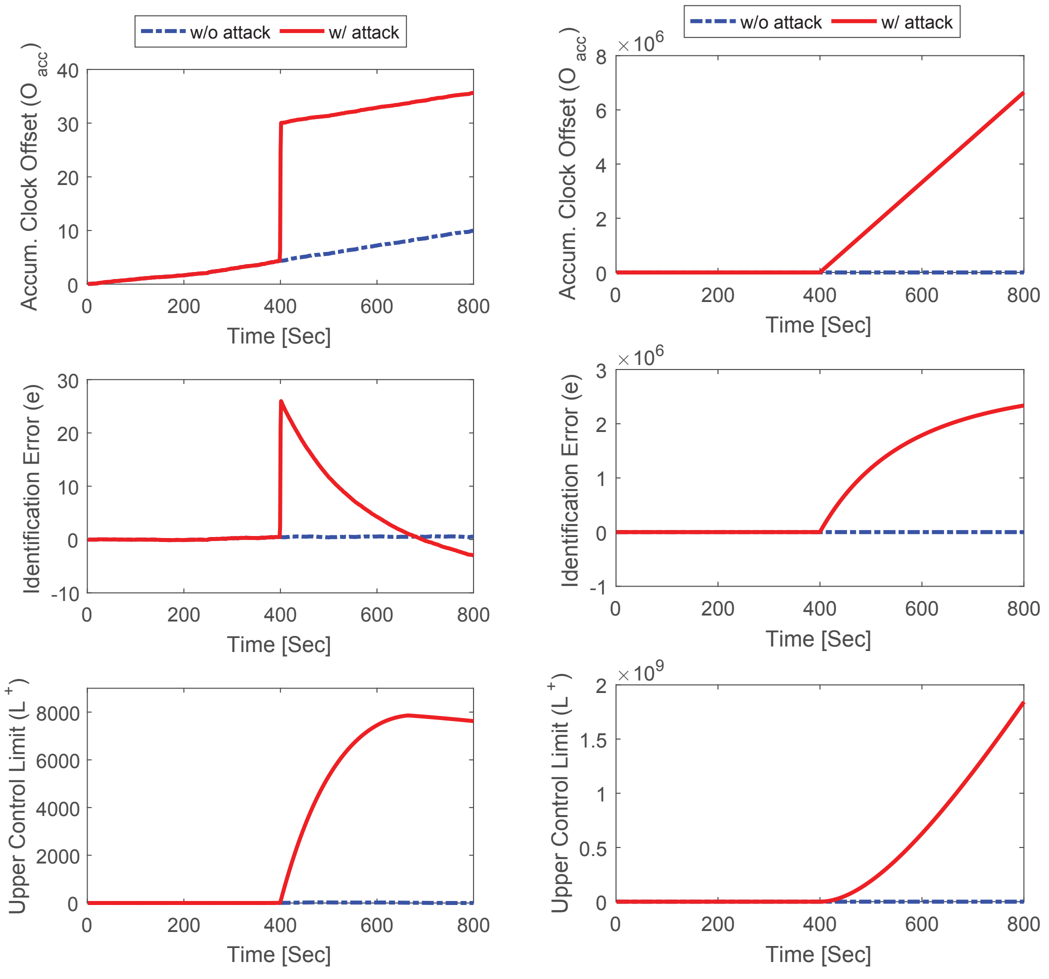
3.5.2 Defending Against Fabrication and Suspension Attacks

On both the CAN bus prototype and the real vehicle setting (Honda Accord 2013), we launch the fabrication and suspension attacks, and evaluate `CIDS`'s effectiveness in detecting them.[2] To this end, we consider `CIDS` to only perform per-message detection, and will later evaluate `CIDS` with message-pairwise detection.

CAN bus prototype. For evaluation of `CIDS` defending against fabrication attack on the CAN bus prototype, $\mathbb{B}$ was programmed to inject a fabricated message at $t = 400$ secs with ID=0x11, which is a periodic message usually sent by $\mathbb{A}$, i.e., $\mathbb{B}$ launches a fabrication attack on $\mathbb{A}$. ECU $\mathbb{R}$ was running `CIDS` on message 0x11 and derived accumulated clock offset (O_{acc}), identification error (e), and control limits (L^+, L^-). For the suspension attack, $\mathbb{A}$ was instead programmed to stop transmitting 0x11 at $t = 400$ secs.

Fig. 3.4a shows how such values changed for message 0x11 in the presence and absence of a fabrication attack. As soon as $\mathbb{B}$ mounted a fabrication attack, as discussed

[2]As the attacks cannot be emulated using the CAN log data, we do not consider their use for evaluating `CIDS` against the attacks.



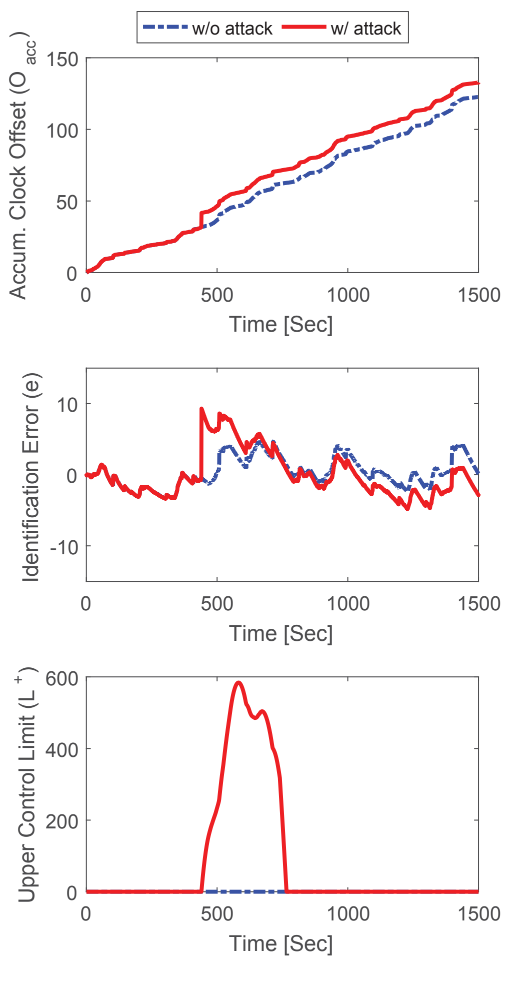
(a) Fabrication attack.

(b) Suspension attack.

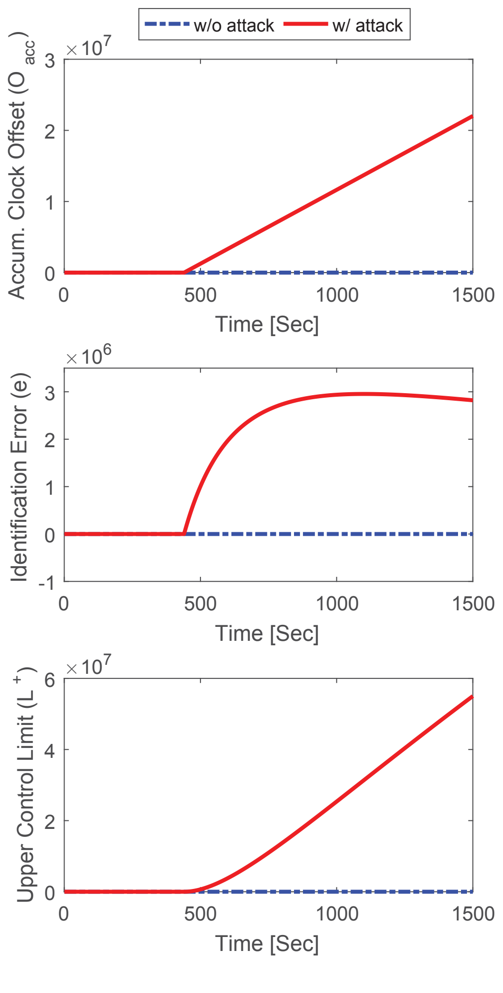
Figure 3.4: CIDS defending fabrication attack (left) and suspension attack (right) in a CAN bus prototype.

in Section 3.4.2, there was a sudden positive shift in the accumulated clock offset, thus yielding a high identification error. Due to such a shift, the upper control limit, L^+, of CUSUM suddenly increased and exceeded its threshold Γ_L =5, i.e., detecting an intrusion. Similarly, Fig. 3.4b shows that since the suspension attack also shifted the accumulated clock offset significantly, CIDS was able to detect the attack.

Real vehicle. To evaluate CIDS against the fabrication attack under the real vehicle setting, one CAN prototype node $\mathbb{R}$ was programmed to run CIDS, and another node $\mathbb{A}$ as an adversary mounting the attack on a real ECU. The attack was mounted by injecting



(a) Fabrication attack.

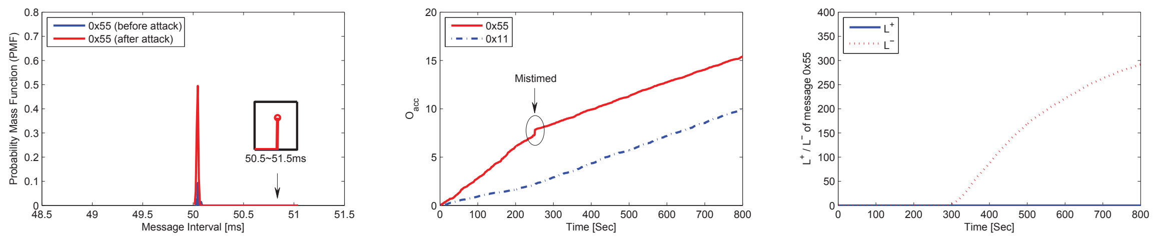


(b) Suspension attack.

Figure 3.5: CIDS defending fabrication attack (left) and suspension attack (right) in a Honda Accord 2013.

a fabricated attack message with ID=0x1B0, which was sent every 20*ms* by some real in-vehicle ECU, i.e., $\mathbb{A}$ mounted the fabrication attack on a Honda Accord ECU sending 0x1B0. For the suspension attack, the message filter of $\mathbb{R}$ was reset at $t = 420$ secs so as to no longer receive 0x1B0, thus emulating the suspension attack.

Fig. 3.5a shows how accumulated clock offsets (O_{acc}), identification errors (e), and upper control limits (L^+) changed for both cases of with and without a fabrication attack. Again, the attack message injected at around $t = 420$ secs caused a sudden increase in O_{acc} and e, thus increasing L^+ to exceed $\Gamma_L = 5$. As a result, CIDS declares the detection of an


(a) CAN bus prototype.

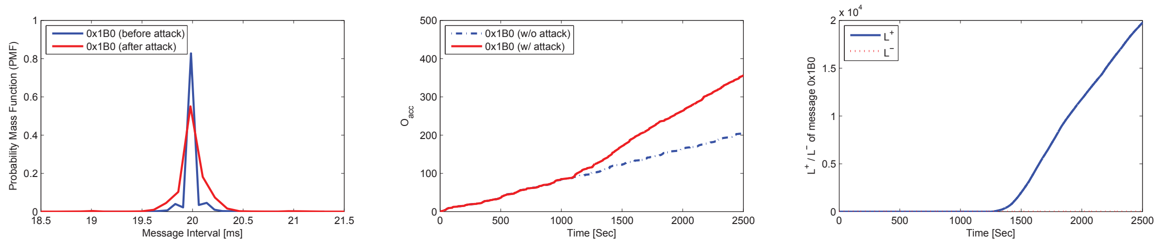

(b) Real vehicle.

Figure 3.6: Masquerade attack — Probability mass function of message intervals (left), changes in accumulated clock offsets (middle), and control limits (right) derived in CIDS.

attack. After the attack, since 0x1B0 was still periodically sent by the real in-vehicle ECU, the clock skew — i.e., the slope of O_{acc} graph — remains unchanged. Similarly, as shown in Fig. 3.5b, the suspension attack increases the offset values, thus causing L^{+} to exceed the threshold, i.e., the suspension attack was detected by CIDS.

3.5.3 Defending Against Masquerade Attack

We now evaluate the performance of CIDS in detecting a masquerade attack.

CAN bus prototype. To evaluate CIDS's defense against the masquerade attack in the CAN bus prototype, nodes $\mathbb{A}$ and $\mathbb{B}$ were considered to have been compromised as strong and weak attackers as in Fig. 3.1c, respectively. $\mathbb{A}$ was programmed to mount a masquerade attack on $\mathbb{B}$, i.e., stop $\mathbb{B}$ transmitting message 0x55 and instead send it through $\mathbb{A}$ onwards, once $T_{masq} = 250$ secs had elapsed. As usual, messages 0x11 and 0x13 were periodically sent by $\mathbb{A}$, and CIDS was run by $\mathbb{R}$.

Fig. 3.6a (left) shows the Probability Mass Function (PMF) of the intervals of message 0x55: before and after the attack was mounted. In contrast to the fabrication attack, since the attacker sent the attack message at its original frequency after masquerading, the distribution did not deviate much from that before the attack. However, at T_{masq}, since there was some delay when the transmitter was switched from one node to another, the first masquerade attack message was sent $51.04ms$ after its previous transmission, whereas it should have been approximately $50ms$ which is the preset message interval of 0x55. Due to such a slightly *mistimed* masquerade attack, the PMF graph shows a message interval with an abnormal deviation from the mean. We will later evaluate the perfectly *timed* masquerade attack — a much more severe case than a mistimed attack — on a real vehicle, and show the efficacy of `CIDS` in detecting it.

The resulting changes in O_{acc}, L^{+}, and L^{-} at $\mathbb{R}$ are also shown in Fig. 3.6a (middle and right). The change in the ECU transmitting message 0x55 caused the slope (i.e., clock skew) in O_{acc} graph to change after the attack was mounted. Since the measurements of O_{acc} after T_{masq} significantly deviated from their expected values, which is determined by the estimated clock skew of $t < T_{masq}$, the CUSUM lower control limit, L^{-}, in `CIDS` exceeded the threshold, thus declaring detection of an intrusion. Since the transmitter of 0x55 was changed (to ECU $\mathbb{A}$), its clock skew after $t = T_{masq}$ was equivalent to the clock skew in 0x11. Accordingly, via its attacker identification, `CIDS` identifies the compromised ECU to be ECU $\mathbb{A}$. Unlike the previous results, since the change in slope was negative, persistent identification error with high negative values caused L^{-} to exceed the threshold.

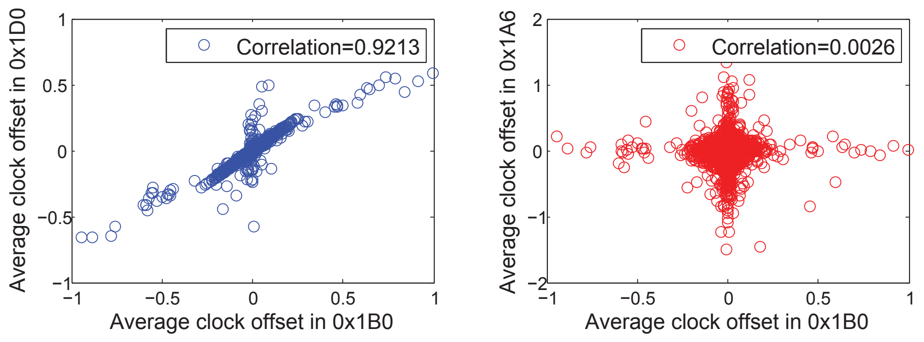
Real vehicle. To evaluate `CIDS`'s defense against the masquerade attack in a real vehicle, we consider a scenario in which real in-vehicle ECUs $\mathbb{V}_1$ and $\mathbb{V}_2$ transmitting 0x1A6 and 0x1B0 are compromised as a strong and a weak attacker, respectively. Of the three CAN prototype nodes ($\mathbb{A}$, $\mathbb{B}$, and $\mathbb{R}$), which were connected to the real in-vehicle network via OBD-II, we programmed node $\mathbb{R}$ to run `CIDS` on in-vehicle message 0x1B0 and another node $\mathbb{B}$ to simply log the CAN traffic. To generate a scenario of real ECU $\mathbb{V}_1$ mounting a masquerade attack on real ECU $\mathbb{V}_2$, $\mathbb{R}$ was programmed further to receive message 0x1A6 instead of 0x1B0, but still record the received messages' ID to be 0x1B0, once $T_{masq} = 1100$ seconds had elapsed. That is, we let $\mathbb{R}$ interpret 0x1A6 as 0x1B0 for $t > T_{masq}$, i.e., the transmitter of 0x1B0 changes from $\mathbb{V}_2$ to $\mathbb{V}_1$. Such a change in interpretation was achieved by programming $\mathbb{R}$ to modify its message acceptance filter from only accepting 0x1B0 to only accepting 0x1A6. Since 0x1B0 and 0x1A6 were observed to be always transmitted nearly at the same time, such a setting replicates the *timed* masquerade attack. During such a process, $\mathbb{B}$ continuously logged 0x1B0 so that we can obtain a reference for circumstances when no attacks are mounted.

Fig. 3.6b (left) shows the PMF of the message intervals of 0x1B0 before and after the attack. Since the message periodicity remained the same, the distribution of the messages intervals did not change. Moreover, since we considered a timed masquerade attack, in contrast to the result in Fig. 3.6a, there were no such abnormal message intervals. Such a result indicates that state-of-the-art IDSs, which try to find abnormal message frequencies, cannot detect such an attack. Although the distribution of message intervals remained unchanged, due to the change in ECU transmitting 0x1B0 ($\mathbb{V}_2 \rightarrow \mathbb{V}_1$), the accumulated clock offset suddenly exhibited a different trend in its change, i.e., a different clock skew after the attack. Here, the original trend in offset changes was determined by the data obtained from $\mathbb{B}$. So, as shown in Fig. 3.6b (right), `CIDS` was able to detect a sudden shift in its identification error and thus outputted a high level of CUSUM upper control limit, i.e., an intrusion detection. `CIDS`'s capability of detecting various types of masquerade attack is evaluated further in Section 3.5.5.

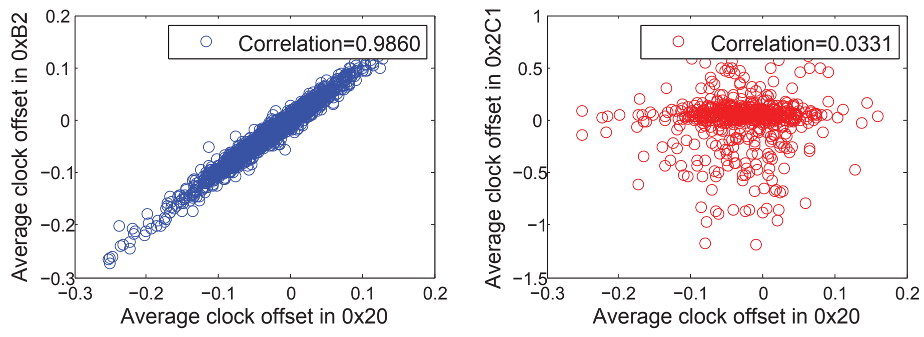
In conclusion, through its modeling and detection processes, `CIDS` can detect not only the fabrication attack but also the masquerade attack, i.e., is capable of doing not only what existing solutions can do, but also more.

3.5.4 Message-pairwise Detection

We evaluate the feasibility and efficiency of message-pairwise detection in `CIDS`. To validate its practicability in the real-world, we first examine whether there exists pairs of messages inside real vehicles with correlated clock offsets — the condition for `CIDS` to run



(a) Honda Accord 2013.



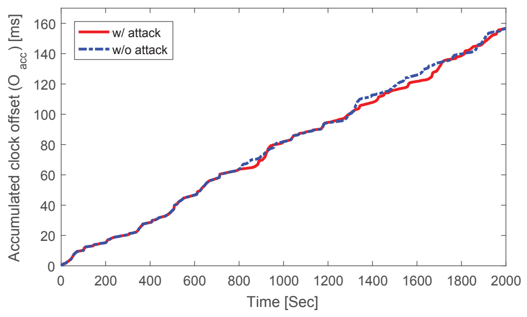
(b) Toyota Camry 2010.

Figure 3.7: Correlated and uncorrelated clock offsets.

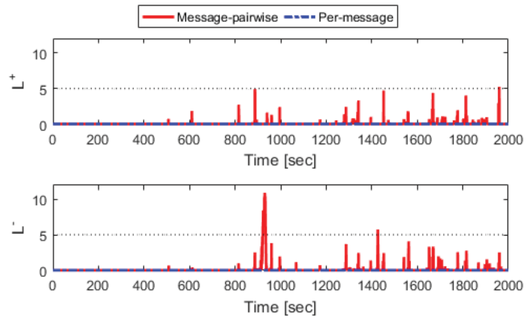
message-pairwise detection.

Fig. 3.7a shows two cases of correlated and uncorrelated clock offsets of in-vehicle messages collected from the Honda Accord 2013. Fig. 3.7a (left) shows that the average clock offsets of messages 0x1B0 and 0x1D0, which were determined to have been sent from the same ECU, showed a high correlation of 0.9213, i.e., linear relationship. In contrast, as shown in Fig. 3.7a (right), average clock offsets of messages 0x1B0 and 0x1A6, which were sent every 20*ms* from different ECUs, showed a near 0 correlation.

By the Birthday paradox, some ECUs in the vehicle may probably have near-equivalent clock skews — as it was for messages 0x20 and 0x2C1 in the examined Toyota Camry 2010 (see Fig. 3.3c). Although clock skews may be near-equivalent, instantaneous clock offsets of two different ECUs cannot be near-equivalent and are thus uncorrelated as they run different processes. The results in Fig. 3.7b corroborate such a fact by showing that clock offsets of messages 0x20 and 0xB2, which were sent by the same ECU, had a high



(a) Accumulated clock offset.



(b) Control limits.

Figure 3.8: Defense against the worst-case masquerade attack via message-pairwise detection.

correlation of 0.9860, whereas offsets of messages 0x20 and 0x2C1 — sent by different ECUs with similar clock skews — had a low correlation of 0.0331. Thus, for messages with near-equivalent clock skews, CIDS can further examine the correlation between their clock offsets, and correctly determine their transmitters.[3] These facts and observations indicate the feasibility and efficiency of message-pairwise detection in CIDS.

To show that message-pairwise detection can support per-message detection in decreasing false positives/negatives by examining offset correlations, we consider a scenario in which an attacker $\mathbb{V}_1$ has mounted a masquerade attack on a Honda Accord ECU $\mathbb{V}_2$ at $t_{masq} = 800$ secs. We refer to $\mathbb{V}_2$ as the ECU which originally transmits message 0x1B0. To consider the *worst case* in detecting the masquerade attack, we assume that the clock skews of $\mathbb{V}_1$ and $\mathbb{V}_2$ are nearly equivalent, similarly to messages 0x20 and 0x2C1 in the Toyota Camry. We replicated such a worst-case scenario by randomly permuting the acquired offset values of 0x1B0 for $t > t_{masq}$, and considering the permuted values to be output from $\mathbb{V}_1$. As shown in Fig. 3.8a, this leads to a situation where the clock skew does not change even though the message transmitter has been changed from one ECU to another. Although the clock skew remained equivalent at $t = t_{masq}$, the correlation between offsets of 0x1B0 and 0x1D0 suddenly dropped from 0.9533 to 0.1201, i.e., a linear to

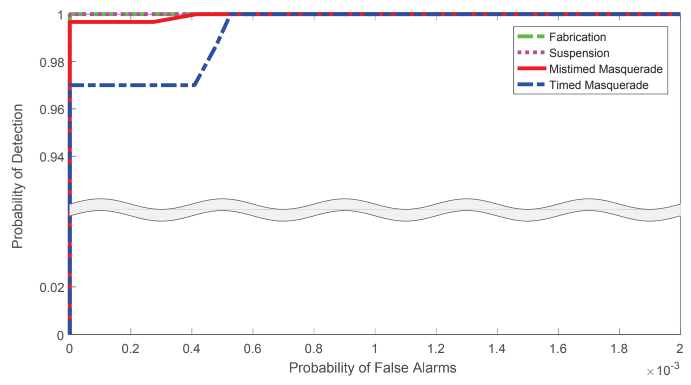
[3]If the two ECUs' clock behaviors are still not distinguishable, CIDS can be set up to exclude them for examination so that the risk of false positives significantly decreases. However, this may impact CIDS's capability of detecting attacks mounted through those ECUs.

non-linear relationship. As a result, as shown in Fig. 3.8b, the control limits in CIDS's message-pairwise detection exceeded the threshold $\Gamma_L = 5$. On the other hand, since the clock skews before and after the attack were equivalent, per-message detection was not able to detect the intrusion.

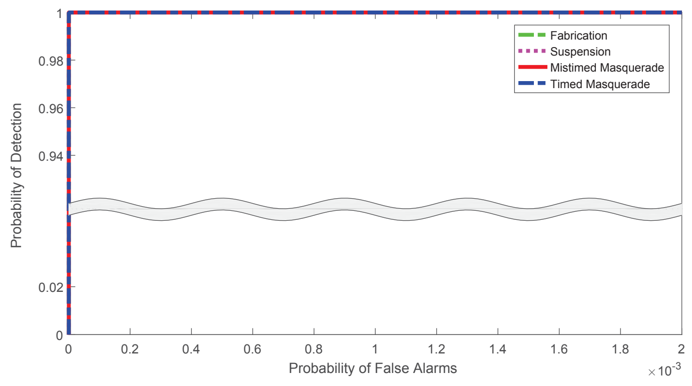
3.5.5 False Alarm Rate

We also examined the false alarm rate of CIDS under the real vehicle setting. The results obtained from the CAN bus prototype are omitted due to their insignificance, i.e., not many false alarms occurred due to its less complex bus traffic. Based on data recorded for 30 minutes from the Honda Accord 2013 — approximately 2.25 million messages on the CAN bus — four attack datasets were constructed to each contain 300 different intrusions. The intrusions either had different injection timings, suspension timings, or changes in clock skews: each in the form of fabrication attack, suspension attack, mistimed masquerade attack, and timed masquerade attack. For each dataset, we varied the κ parameter of CIDS to acquire one false positive rate (false-alarm rate) and one false negative rate ($1-$detection rate).

Fig. 3.9a shows the Receiver Operating Characteristic (ROC) curve of CIDS, which represents its trade-off between false alarm and detection, executing *only* per-message detection on the attack datasets. Clearly, CIDS is shown to be able to detect fabrication, suspension, and masquerade attacks with a high probability. Since the timed masquerade attack is the most difficult to detect, it showed the highest false positive rate among all the attack scenarios considered: a false positive rate of 0.055% while not missing any anomalies (100% true positives). Even for false positives $< 0.055\%$, 97% of the anomalies were detected by CIDS. However, these false positives can be of great concern for in-vehicle networks. Therefore, to eliminate such false positives, CIDS can additionally run message-pairwise detection. Fig. 3.9b shows the ROC curve of CIDS executing not only per-message detection but also message-pairwise detection for further verification.



(a) Per-message detection.



(b) Per-message + Message-pairwise detection.

Figure 3.9: ROC curves of `CIDS` in the real vehicle.

Accordingly, `CIDS` was able to detect all types of attacks considered without having any false positives, which is in contrast to `CIDS` with only per-message detection, i.e., all false positives were eliminated via message-pairwise detection.

3.6 Discussion

Discussed below are the overhead, deployment, limitations, and applications of `CIDS`.

Identification algorithm. To estimate clock skew, one can also use other algorithms than RLS, such as Total Least Squares (TLS) and Damped Least Squares (DLS), which perform orthogonal linear and non-linear regression, respectively. Although they might identify the clock skew with a higher accuracy than RLS, their gains are offset by the accompanying high complexity. TLS requires Singular Value Decomposition (SVD), which is computationally expensive, and DLS requires a large number of iterations for curve fitting. RLS is known to have a computation complexity of $\mathbb{O}(N^2)$ per iteration, where N is the size of the data matrix. However, in `CIDS`, only a scalar clock offset is exploited for identification, and thus the computational complexity is relatively low.

Defeating `CIDS`. There may be several ways the adversary may attempt to defeat `CIDS`. First, the adversary may try to compromise the ECU running `CIDS` and disable it. However, if *cross-validation* for `CIDS` was to be exploited, such an attempt can be nullified. For the detection of intrusions, `CIDS` only requires an ECU to record the timestamps of message arrivals. Such a low overhead makes it feasible for `CIDS` to be installed distributively across several in-vehicle ECUs for cross-validation. Suppose using `CIDS`, ECU $\mathbb{A}$ monitors attacks on messages $\{M_1, M_2\}$, ECU $\mathbb{B}$ monitors $\{M_2, M_3\}$, and ECU $\mathbb{C}$ monitors $\{M_1, M_3\}$. Since `CIDS` regards the *receiver*'s time clock as the true clock, cross-validation provides multiple perspectives of clock behaviors for each message ID, e.g., two different perspectives of M_2 from $\mathbb{A}$ and $\mathbb{B}$. Thus, even when an ECU running `CIDS` gets compromised, cross-validation via `CIDS` can handle such a problem.

Another way the adversary may try to defeat `CIDS` is to adapt to how its algorithm is running and thus deceive it. The adversary may figure out the clock skew of the target ECU and then heat up or cool down the compromised ECU so that its clock skew changes to match that of the target. In such a case, the clock skew can be matched and thus may bypass `CIDS`'s per-message detection. However, as discussed in Section 3.5.4, unless the adversary also matches the instantaneous clock offset, which is affected by the ECU's *momentary* workload and temperature, `CIDS` can detect the intrusion via message-pairwise detection.

Upon intrusion detection. False alarms for intrusion detection systems, especially in in-vehicle networks, are critical. Thus, `CIDS` should also deal with them as accurately as possible. To meet this requirement, if an intrusion has been determined, even after going through the verification process, `CIDS` can follow the following steps for further examination:

1. If an intrusion was detected while using only per-message detection, examine it further via message-pairwise detection.
2. If still alarmed as an intrusion and the attacked ECU is a safety-critical ECU, go straight to step 4.
3. If not, communicate with other ECUs for cross-validation as they would provide different perspectives of the clock skew results. If communicating with other ECUs incurs too much overhead (in terms of bus load, processing overhead, etc.), send traffic data for a remote diagnosis.
4. Request re-patching of firmware and advise the driver to stop the vehicle.

Limitation of `CIDS`. `CIDS` is shown to be effective in detecting various types of in-vehicle network intrusions. One limitation of `CIDS` might be that since it can only extract clock skews from periodic messages, it would be difficult to fingerprint ECUs which are sending aperiodic messages. That is, if the attacker injects messages aperiodically, although `CIDS` can still detect the intrusion, it would not be able to pinpoint where the attack message came from, i.e., finding the source of the attacks launched with or on aperiodic messages. Recall that `CIDS` can achieve this only for periodic messages. In future, we would like to find new features other than clock skew, which can fingerprint ECUs, regardless of whether they send messages periodically or aperiodically.

Applicability to other in-vehicle networks. Although most modern in-vehicle networks are based on CAN, some may be equipped with other protocols, such as CAN-FD, TTCAN and FlexRay, for more complex operations. CAN-FD is an enhanced version of CAN, providing flexible and higher data rates [3]. Since its basic components conform to CAN and thus also lacks synchronization, `CIDS` can be applied to CAN-FD. For protocols such as TTCAN [51] and FlexRay [55], nodes are periodically synchronized for determinative timing of message exchanges. The interval between two consecutive synchronizations depends on how each protocol is deployed [68]. For TTCAN, it can be up to $2^{16} = 65536$ bits long, i.e., $131ms$ in a 500Kbps bus [79]. This lets some messages be sent multiple times between consecutive synchronizations. So, if the time interval is long, `CIDS` would still be able to extract clock skews from messages which are sent multiple times, whereas, if the period is short, `CIDS` may not be feasible. However, the fact that TTCAN and FlexRay have high implementation cost, whereas for CAN-FD it is minimal, makes CAN-FD a favorite candidate for next-generation in-vehicle networks [24, 84]. This means that `CIDS` can be applicable to not only current but also future in-vehicle networks.

3.7 Conclusion

New security breaches in vehicles have made vehicle security one of the most critical issues. To defend against vehicle attacks, several security mechanisms have been proposed in the literature. They can cope with some attacks but cannot cover other safety-critical attacks, such as the masquerade attack. To remedy this problem, we have proposed a new IDS called `CIDS`, which extracts clock skews from message intervals, fingerprints the transmitter ECUs, and models their clock behaviors using RLS. Then, based on the thus-constructed model, `CIDS` detects intrusions via CUSUM analysis. Based on our experiments on a CAN bus prototype and on real vehicles, `CIDS` is shown to be capable of detecting various types of in-vehicle network intrusions. `CIDS` can address all attacks that existing IDSs can and cannot handle as well as facilitates attacker identification. Thus, it has potential for significantly enhancing vehicle security and safety.

CHAPTER IV

Viden: Attacker Identification on In-Vehicle Networks

4.1 Introduction

Numerous defense schemes have been proposed to detect and/or prevent various vehicle cyber attacks [33, 45, 65, 66, 71, 83, 86]. Although these countermeasures have reinforced cyber security of vehicles somewhat, they all fail to meet one important need: *attacker identification*. That is, although state-of-the-art countermeasures are capable of determining whether or not there is an intrusion in the in-vehicle network, they cannot determine *which* ECU is actually mounting the attack. In Chapter III, despite its main objective being intrusion detection, we mentioned that `CIDS` can also achieve attacker identification, but only in a very restricted scenario: when attack messages are injected *periodically*. An accurate attacker identification is imperative as it provides a swift pathway for forensic, isolation, security patch, etc. No matter how well an IDS detects the presence of an intrusion in a vehicle, if we still do not know which ECU is mounting the attack and hence which ECU to isolate/patch, the vehicle remains insecure and unsafe. It is much better and more economical to isolate/patch the attacker ECU, than blindly treating *all* ECUs as (possible) attackers.

To meet this essential need for attacker identification — that existing solutions have not yet been able to satisfactorily meet — we propose a novel scheme, called `Viden` (Voltage-based attacker identification), which fingerprints message transmitter

ECUs on CAN via voltage measurements and thus facilitates attacker identification. The rationale behind using voltage for fingerprinting ECUs is the existence of small inherent discrepancies in different ECUs' voltage outputs when they inject messages. To capture this and then use it to fingerprint the transmitter ECUs, `Viden` first monitors the output voltages from the two dedicated wires on the CAN bus: CAN-High (CANH) and CAN-Low (CANL). All ECUs' transceivers are connected to, and use these for their message transmissions and receptions. Through the acquired voltage measurements for each message ID, `Viden` first learns the *ACK threshold*, the key information `Viden` uses to discard the measurements of voltages outputted by ECUs while acknowledging the receipt of, but not transmitting, the message. `Viden` utilizes the thus-derived ACK threshold to learn the voltage output behavior of each in-vehicle ECU by constructing new features called *voltage instances*. Then, it transforms those instances to the transmitter ECU's voltage profile (i.e., *fingerprint*) via Recursive Least Square (RLS) algorithm, an adaptive signal processing technique. As a result, `Viden` utilizes the derived voltage profiles for an accurate attacker identification. Through experimental evaluations on a CAN bus prototype and on two real vehicles, we show that the constructed voltage profiles are distinct for different ECUs, thus validating `Viden`'s capability of identifying the attacker ECU.

While there have been proposals to fingerprint ECUs with timing [33] or voltage (like `Viden`) measurements [35, 64], their practicality and efficiency in identifying the attacker ECU remain limited to only certain attack scenarios, mainly because they were designed for intrusion detection, not attacker identification. In other words, there are many scenarios in which existence of an attack is detected but the attacker cannot be identified correctly. Thus, we design, implement, and evaluate `Viden` by focusing on attacker identification via a distinct way of fingerprinting ECUs from the existing schemes. As a result, `Viden` is efficient and easy to deploy on any ECU, thanks to its *adaptability* and *practicality*.

Adaptability. Existing voltage-based fingerprinting uses supervised batch learning that generates a norm model by learning from a pre-defined training data set [35, 64]. So, until the training data set and hence models/fingerprints are updated again, the norm models remain unchanged. Such an approach, however, cannot adapt the norm models to unexpected changes (e.g., changes in temperature) inside/outside the vehicle. More importantly, adversaries who intentionally generate changes can evade these existing fingerprinting schemes. `Viden` takes a very different approach from them in that it models and updates the voltage-based fingerprints by applying adaptive signal processing (i.e., *online* (not batch) learning) to its new set of features: voltage instances. This enables `Viden` to correctly modify the fingerprints and hence adapt to inevitable but unpredictable changes in vehicles that can either occur naturally (due to the mother nature) or be intentionally triggered by an intelligent adversary. Such adaptability is essential for vehicle security.

Practicality. Unlike the existing voltage-based fingerprinting schemes, the unique approach taken by `Viden` eliminates the requirement/assumption of using a specific CAN message type or CAN bus speed, thus facilitating its deployment. Moreover, it does not require any knowledge of which message fields the voltages are measured on, i.e., *message-field-agnostic*. This enables `Viden` to achieve its goal even with a low voltage sampling rate, thus lowering cost. Furthermore, even though it is message-field-agnostic, since `Viden` filters out undesired samples using its derived ACK thresholds, there is no need to impose restrictions on which fields of the message should be sampled to run `Viden`. All of these salient features enable `Viden` to run without re-designing current CAN controllers and make `Viden` very practical and cost-efficient, which is very important for the cost-conscious automotive industry.

We have implemented and evaluated `Viden` on a CAN bus prototype and on two real vehicles. Our evaluation results show that `Viden` can identify the attacker ECU with a low false identification rate of 0.2%, thanks to its unique fingerprinting that makes it adaptive to handle various attack scenarios.

This work makes the following main contributions:

1. Proposal of a new scheme which retains only the voltage measurements output by the transmitter ECU (Section 4.3.4);
2. Design of `Viden` which constructs voltage profiles, i.e., fingerprints, by modeling the norm voltage output behaviors of in-vehicle ECUs and exploits them for accurate identification of the attacker ECU (Sections 4.3.5–4.3.8);
3. Implementation and demonstration of `Viden` on a CAN bus prototype and on two real vehicles (Section 4.4).

4.2 Background

4.2.1 CAN Message Transmission

In-vehicle ECUs broadcast their retrieved sensor data via a CAN frame/message. Instead of carrying the address of the transmitter/receiver, as shown in Fig. 2.1, it contains a unique identifier (ID), which represents its priority. Starting from a 0-bit followed by a sequence of dominant (0) or recessive (1) bits, all fields within the CAN frame are sent on the bus by the "transmitter ECU" except for the Acknowledgment (ACK) slot. The ACK slot is, in fact, used by *all* ECUs *at the same time* — except for the transmitter ECU — that have correctly received the preceded fields of the ACK slot, regardless of whether they are interested in their content or not. If correctly received, those ECUs send a 0-bit in the ACK slot. Thus, multiple ECUs acknowledge the message simultaneously, even before the transmitter finishes sending its message on the bus.

To send either a 0- or 1-bit, CAN transceivers (are agreed to) output certain voltage levels on the two dedicated CAN wires: CANH and CANL. As shown in Fig. 4.1, to issue a 0-bit on the CAN bus, CAN transceivers (are agreed to) output approximately 3.5V on

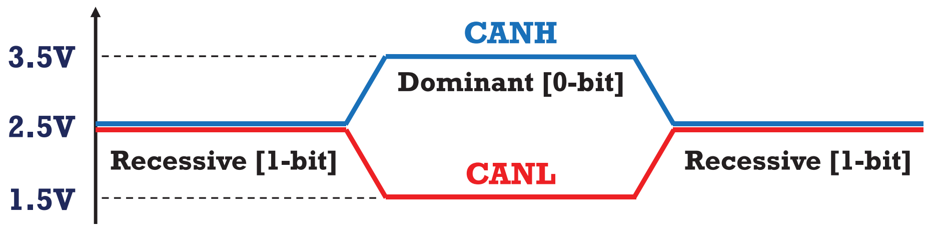
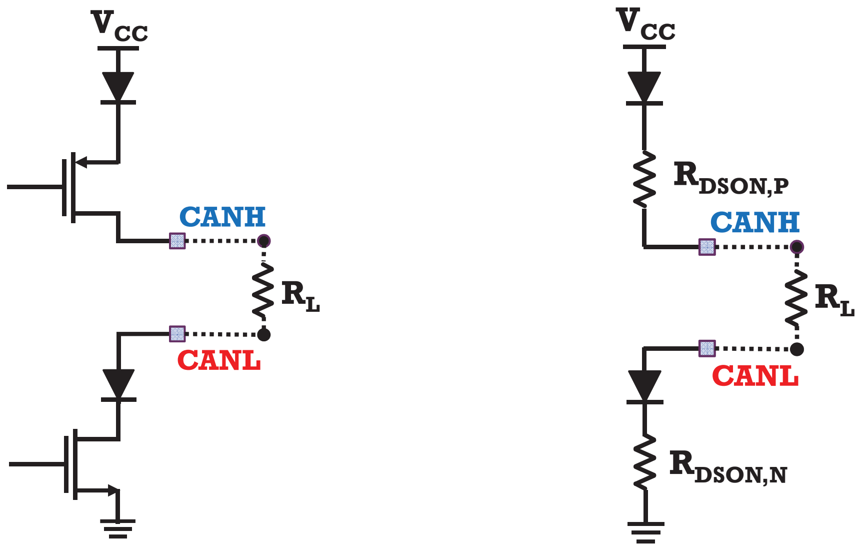


Figure 4.1: CAN output voltages when sending a message.



(a) Transceiver schematic. (b) When sending a 0-bit.

Figure 4.2: Output schematics of a CAN transceiver.

CANH and 1.5V on CANL so that the differential voltage becomes approximately 2V. On the other hand, when sending a 1-bit, the transceivers output approximately 2.5V on both CANH and CANL, yielding a differential voltage of approximately 0V [8, 9]. So, by measuring the differential voltage of CANH and CANL, receiver ECUs read the streams of 0 and 1 bits, and thus receive the message. From this perspective, CAN is a *differential bus*.

CAN transceivers output the intended voltages by simultaneously switching on/off their transistors. Fig. 4.2a shows an equivalent schematic of a CAN transceiver [16, 18]. Note that CAN transceivers of multiple ECUs are connected to the CAN bus in parallel, thus sharing the same load resistance R_L, which is normally set to 60Ω [20]. The high- and low-side output circuits consist of a series diode and a P- and N-channel transistor, respectively.

For the transceiver to send a 1-bit, both the high and low side transistors are switched *off* and are thus in a high impedance state. This results in negligible current flowing from V_{CC} to ground, yielding negligible differential voltage on CANH and CANL. On the other hand, when sending a 0-bit, both transistors are turned *on* and are thus in a low impedance state. When the transistors are on, they can be equivalently described as resistors with drain-to-source on-state resistance R_{DSON} as shown in Fig. 4.2b, where current flows from V_{CC} to ground through R_L and thus creates a differential voltage of (approximately) 2V between CANH and CANL. This way, the CAN transceivers are capable of outputting either 0 or 2V of differential voltage on the two CAN wires.

4.2.2 Related Work

Researchers have attempted to fingerprint ECUs in various ways, mostly for the purpose of intrusion detection.

In Chapter III, we proposed CIDS which detects intrusions by fingerprinting ECUs on CAN. While its main objective was to detect intrusions, we also mentioned that the thus-derived fingerprints may also be used for attacker identification, but only when attack messages are injected *periodically*. In other words, if the attacker transmits messages *aperiodically*, then CIDS cannot identify the attacker ECU, i.e., the adversary can evade CIDS as far as attacker identification is concerned. Viden takes an entirely different approach: looking at attack messages from the perspective of ECUs' *output voltages* on CAN. This allows Viden to accurately identify the attacker ECU irrespective of how and when the attacker injects its messages, which is crucial for attacker identification.

Instead of fingerprinting ECUs based on message timings, as in Viden, some researchers also proposed to fingerprint them via voltage measurements. The authors of [64] used the Mean Squared Error (MSE) of voltage measurements as fingerprints of ECUs. However, they were shown to be valid only for the voltages measured during the transmission of CAN message IDs, and more importantly when voltages were measured on a low-speed (10Kbps) CAN bus; this is far from contemporary vehicles that usually operate on a 500Kbps CAN bus.

To overcome these difficulties, researchers proposed to extract other time and frequency domain features of voltage measurements (e.g., RMS amplitude) and use them as inputs for classification; more specifically, supervised (batch) learning algorithms (e.g., SVM) [35]. This way, they were able to fingerprint ECUs with enhanced accuracy and was successful on high-speed CAN buses. However, this solution was neither practical nor attractive for attacker identification for the following reasons. First, it required not only a high sampling rate (2.5 GSamples/sec), but also the use of the *extended* CAN frame format with 29-bit IDs, which is seldom used (due to its bandwidth waste) in contemporary vehicles; most vehicles use the *standard* format with 11-bit IDs. Moreover, since the modeling was done via batch learning, unpredictable changes in the CAN bus (e.g., temperature, battery level) and adversary's behaviors can lead to false identifications. These will be detailed later when we discuss the details of `Viden`.

In contrast, `Viden` fingerprints ECUs very differently and hence achieves effective attacker identification (1) through online update of fingerprints via adaptive signal processing to provide adaptability; (2) at a low sampling rate (50 KSamples/sec); and more importantly, (3) without imposing restrictions on the type of CAN message or the speed of CAN bus to be used. As a result, the deployment of `Viden` in legacy and new vehicles will be much easier.

4.3 Viden

Attacker identification is essential for expedited forensic, isolation, and security patches, all of which are the key requirements for vehicle safety. To meet this need, we propose a novel fingerprinting scheme, `Viden`, that exploits small inherent discrepancies in different ECUs' voltage outputs. Before delving into the inner workings of `Viden`, we first describe the system and threat models.

4.3.1 System and Threat Models

4.3.1.1 System Model

The vehicle's CAN bus under consideration is assumed to have been equipped with an IDS as well as a timing- (e.g., `CIDS`) and voltage-based (e.g., schemes in [35, 64] or `Viden`) fingerprinting device; the latter complements the former via attacker identification. We discern a fingerprinting device from an IDS based on the fact that the IDS detects the *presence* of an attack whereas the fingerprinting device identifies the *source* of the (detected) attack. An attack can be mounted by the adversary who has control of a physically/remotely compromised ECU. In our system model, however, we consider such an ECU to have been *remotely* compromised and thus controlled by the adversary as in [31,58]. We do not consider a compromised device which was attached to the in-vehicle network (e.g., device plugged in the OBD-II port), as it requires physical access and its identification has been addressed elsewhere [35, 72]. So, the compromised ECU we consider is one of those originally installed on the vehicle's CAN bus.

4.3.1.2 Threat Model

By injecting fabricated attack messages through his compromised ECU, the attacker can control the vehicle maneuver. We consider the attacker to be smarter than this: beyond just controlling the vehicle, the attacker's goal is to also hide the identity of the ECU injecting the attack messages. That is, while the deployed IDS may detect the presence of an attack, the adversary tries to *evade* the fingerprinting device, i.e., prevent it from determining the source of the attack. For evasion, the adversary can perform two different impersonations when injecting his attack messages:

- *Arbitrary impersonation*: The attacker misleads the fingerprinting device to think that some *arbitrary* ECU other than himself is the attacker.

- *Targeted impersonation*: The attacker acts smarter by impersonating a *targeted* ECU for evasion, i.e., make the fingerprinting device believe that the targeted ECU is the attacker.

Depending on the adversary's capabilities and knowledge of different defense schemes (available in the market or in literature) as well as their operation, his approach to evading a fingerprinting device would be different. Specifically, based on whether the adversary is aware of the fact that an in-vehicle ECU can be fingerprinted via timing and/or voltage measurements, his best effort in achieving his goal would be different. Thus, we consider three different types of adversaries: *naive*, *timing-aware*, and *timing-voltage-aware* adversaries.

While all attackers are capable of injecting and sniffing messages on the CAN bus, a *naive* adversary does not have any knowledge of how ECUs can be fingerprinted (either via timing or voltage), due possibly to lack of his technical expertise or curiosity. Thus, the naive adversary injects his attack messages imprudently at arbitrary times with forged message IDs (for impersonation).

An intelligent adversary, however, might know how ECUs can be fingerprinted via timing analysis. Thus, the adversary uses his knowledge to evade any (possibly-installed) fingerprinting scheme as much as possible as follows. The adversary logs CAN traffic, learns the timing behavior of other ECUs, and exploits the learned information in injecting attack messages at the appropriate (learned) times so as to imitate other ECUs' timing behavior. This way, the adversary can perform arbitrary/targeted impersonation and thus attempt to evade the fingerprinting device. We refer to this adversary as a *timing-aware* adversary.

The adversary might also have knowledge of how ECUs can be fingerprinted via voltage and timing measurements. Hence, when injecting attack messages, such an adversary may try to exploit his knowledge in impersonating other ECU(s) and thus evade any fingerprinting device as much as possible. We call this adversary a

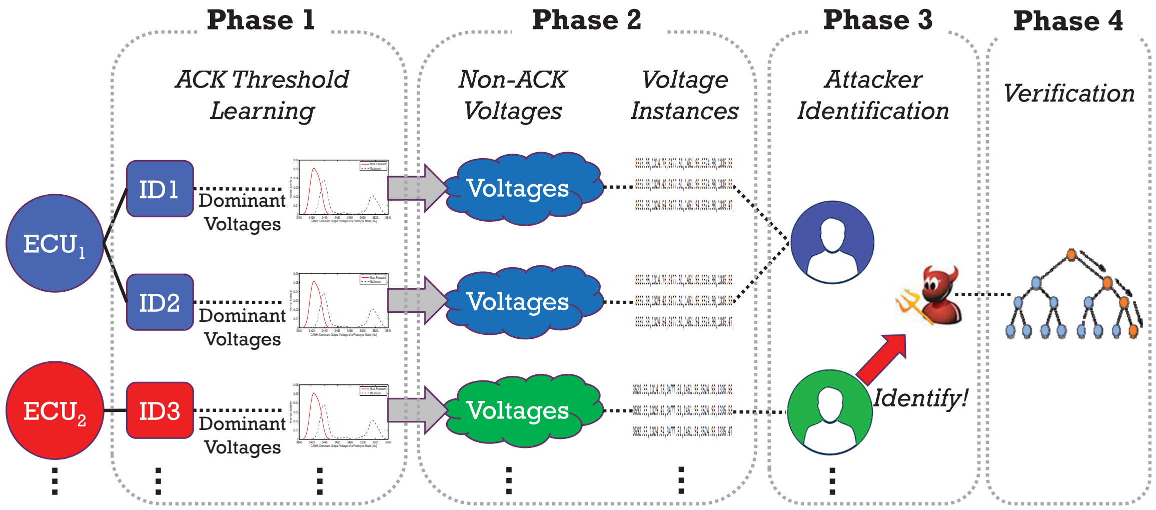


Figure 4.3: An overview of `Viden`.

timing-voltage-aware adversary. We consider such an adversary to be capable of changing his voltage outputs via running battery draining processes, changing the supply voltage level, or by heating up or cooling down the ECU. Although he can change them to a certain level, we consider him to be incapable of *precisely* controlling their instantaneous values. This is reasonable as precise control of voltages would require, for example, control of even the ambient temperature. By changing his ECU's voltage outputs to a certain level in which the targeted ECU is outputting, a timing-voltage-aware adversary can perform a targeted impersonation. Similarly, he can arbitrarily change the output levels for arbitrary impersonation. Since changing his voltage levels (either before or during message injections) does not necessarily imply that he is attacking the CAN bus, in this chapter, we differentiate "impersonation" from an actual attack of message injections. In addition, the adversary might even know when the voltage-based fingerprints are updated (if not updated in real time) and thus use that as a reference in determining when to perform arbitrary/targeted impersonation. Note, however, that he must "play" within the setting boundaries of the given CAN bus. For example, the attacker cannot control/tune the values of resistors within the CAN bus in order to control the voltage levels, as this requires physical access.

4.3.2 High-Level Overview of `Viden`

As shown in Fig. 5.1, `Viden` fingerprints ECUs via voltage measurements and achieves attacker identification in four phases.

Phase 1: `Viden` measures the CANH & CANL voltages and maps the recently acquired values to the ID of the message it has just received through the ECU's receive buffer. Then, for that message ID, `Viden` learns its ACK threshold. This threshold helps `Viden` determine whether or not the measured voltage originates from the actual message transmitter. Phase 1 is run in the initialization step of `Viden` and when an update is necessary.

Phase 2: Exploiting the learned ACK threshold, `Viden` selects voltages that are outputted solely by the message transmitter. Then, `Viden` uses them to derive a *voltage instance*, which is a set of features that reflect the transmitter ECU's voltage output behavior. Phase 2 and onwards are run iteratively.

Phase 3: `Viden` uses every newly derived voltage instance to update the voltage profile of the message transmitter. When an attack is detected by the IDS, `Viden` constructs a voltage profile for the attack messages and maps that profile to one of those `Viden` has, thus identifying the attacker ECU.

Phase 4: The results from Phase 3 are verified further via multi-class classification, only when necessary.

For a given message ID, only one ECU is assigned for its transmission in most cases. Thus, for now we consider the relationship between the numbers of ECUs, IDs, and voltage profiles to be 1, $N(\geq 1)$, and 1, respectively. We will discuss further in Section 4.5 on how `Viden` deals with cases where the relationship between the numbers of ECUs and IDs might be N and 1, respectively.

4.3.3 CANH and CANL Voltage Outputs

Before presenting the details of `Viden`, we first discuss which voltage characteristics of ECUs it exploits for attacker identification.

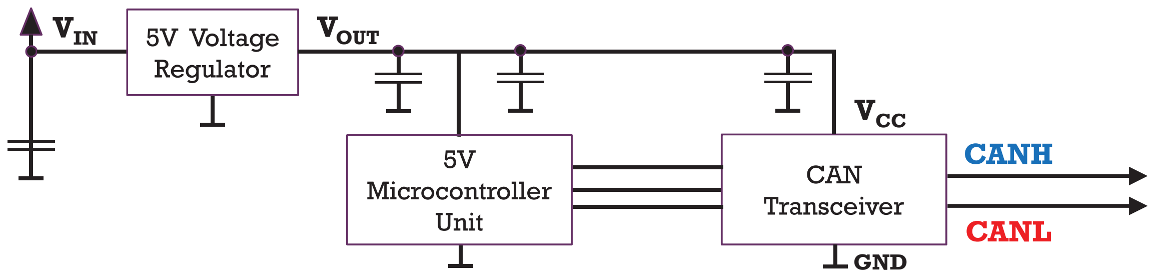


Figure 4.4: CAN typical application schematic.

Variations in supply and ground voltages. Fig. 4.4 shows a typical ECU connection to CAN [16, 18]. In order to output the desired voltage levels on CANH and CANL, transceivers are powered with the nominal supply voltage (V_{CC}) of 5V, which is provided and maintained by a voltage regulator. The input of the regulator, V_{IN}, comes from a power supply, i.e., a 12V/24V battery powering all the ECUs [52]. Not only the voltage regulator but also the connected bypass capacitors help stabilize the V_{CC} level. However, the output voltage of an ECU's regulator varies *independently* and *differently* from other ECUs' regulators, as their supply characteristics are different (e.g., different regulators' common-mode rejection ratios). Thus, there are inherent, small but non-negligible differences in ECUs' V_{CC}. There exist variations in not only V_{CC} but also in the ground voltage since there does not exist a perfect ground [20].

For these reasons, CAN transceivers are built to operate over a range of voltages (e.g., TI TCAN10xx devices are designed to handle 10% supply variations [18]). This guarantees transceivers with different V_{CC} and/or ground to communicate messages correctly.

Variations in on-state resistance. When transceivers send a 0-bit, their two transistors are turned on so that the flowing current generates the required differential voltage between CANH and CANL. In such a case, transistors in the transceivers are considered as resistors $R_{DSON,P/N}$ (see Fig. 4.2b). Although transceivers are designed to have the same $R_{DSON,P/N}$ values, process/manufacturing variations/imperfections cause transistors' $R_{DSON,P/N}$ values to be slightly different from each other [60].

Fig. 4.5 shows a typical circuit diagram of a CAN transceiver. Transistors' R_{DSON}

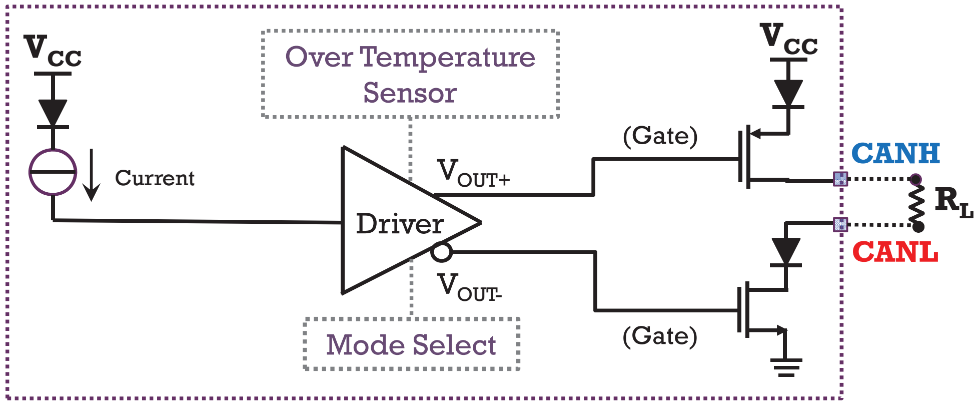


Figure 4.5: Transistors' gate voltages are fed by the driver.

values are inversely related to their gate voltages, which are supplied by a driver, i.e., a fully differential amplifier [14, 80]. Interestingly, since the driver input is affected by V_{CC}, which also varies with ECU, the transistors' gate voltages are also affected by V_{CC}. Therefore, variations in V_{CC} lead to variations in transistors' actual R_{DSON} values. In summary,

𝕍1. *There exist differences/variations in CAN transceivers' nominal supply voltage, ground voltage, and $R_{DSON,P/N}$ values, especially during the transmission of a 0-bit.*

When transmitting a 1-bit, the two transistors are simply turned off and thus there is little voltage variation between nodes. Hence, we do not consider any voltage measurements when the transmitter was sending a 1-bit. Instead, we only consider those measured when it was sending a 0-bit, and refer to those as *dominant voltages*.

Variations in dominant voltages. From Fig. 4.2b, when transceiver i is transmitting a 0-bit, the current, $I_{(i)}$ flowing from its $V_{CC(i)}$ to its ground can be derived as $I_{(i)} = \frac{V_{CC(i)} - V_{G(i)} - 2V_D}{R_{DSON,P(i)} + R_{DSON,N(i)} + R_L}$, where $V_{G(i)}$ denotes its ground voltage, and V_D the diodes' forward bias (assuming they are equivalent). To simplify the analysis, we omit other factors such as leakage current or variations in diodes. We can thus derive the CANH and CANL dominant voltages, $V_{CANH(i)}$ and $V_{CANL(i)}$, from transceiver i as:

$$\begin{aligned} V_{CANH(i)} &= V_{CC(i)} - V_D - I_{(i)} R_{DSON,P(i)}, \\ V_{CANL(i)} &= V_{G(i)} + V_D + I_{(i)} R_{DSON,N(i)}. \end{aligned} \tag{4.1}$$

From Eq. (4.1), one can see that

$\mathbb{V}2$. *Variations in V_{CC}, ground, and $R_{DSON,P/N}$ result in different ECUs with different CANH and CANL dominant voltages.*

For this reason, the ISO11898-2 specifies that a compliant transceiver must accommodate dominant voltages of CANH=2.75∼4.5V and CANL=0.5∼2.25V [8]. Hence, we refer to any voltage values meeting this requirement as *dominant voltages*.

Transient changes in on-state resistances. In Fig. 4.5, when V_{OUT+} of the driver increases, V_{OUT-} concurrently decreases as they are differential outputs. So, for both transistors, the absolute differences between their gate and source voltages simultaneously decrease. This results in both $R_{DSON,P}$ and $R_{DSON,N}$ to increase, i.e., change in the *same* direction [14,80]. Even when a change in the ECU temperature affects $R_{DSON,P}$ & $R_{DSON,N}$, they change in the same direction. So, for a given V_{CC} and ground voltage, the opposite signs of $I_{(i)}R_{DSON,P/N(i)}$ in (4.1) indicate that

$\mathbb{V}3$. *Transient changes in the ECU temperature and driver's input/output affect $R_{DSON,P/N}$, and thus make V_{CANH} and V_{CANL} temporarily deviate in the "opposite" direction.*

Since regulated V_{CC} and ground voltage remain constant, and are not affected by transient changes in $R_{DSON,P/N}$,

$\mathbb{V}4$. *Transient changes in V_{CC} and ground are significantly smaller than those in V_{CANH} and V_{CANL}, i.e., their values remain relatively constant.*

$\mathbb{V}1$–$\mathbb{V}4$ indicate that CANH and CANL dominant voltages of each ECU are different from each other. `Viden` exploits this fact in constructing different voltage profiles for (fingerprinting) ECUs.

4.3.4 Phase 1: ACK Threshold Learning

`Viden` is designed to run with a low voltage sampling rate so that it can be easily installed as a low-cost software application, which requires no changes in the CAN

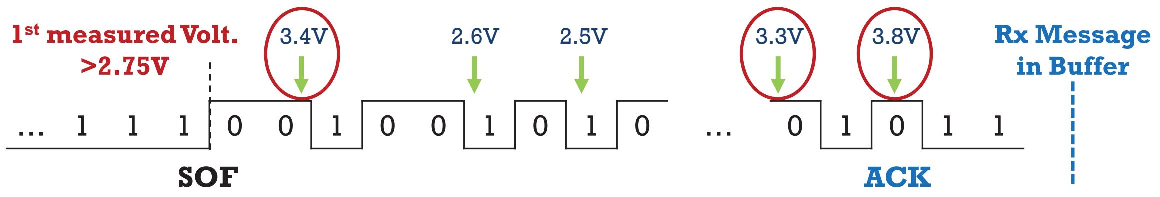


Figure 4.6: `Viden` measuring CANH voltages.

protocol; the high rate of voltage sampling would only be required for the CAN protocol to receive messages as it is designed to be. Such a feature, however, renders `Viden` incapable of determining at which slot the voltage values were measured; all it knows is the value. Thus, `Viden` goes through a phase of learning the *ACK threshold*, which determines whether or not the measured voltage was outputted by the message transmitter.

Measuring dominant voltages. `Viden`'s measurement is triggered whenever a CANH voltage exceeds 2.75V after a certain idle period. This is because the first measured voltage exceeding 2.75V represents the case of some transmitter transmitting a 0-bit on the bus [8]. Since `Viden` is only interested in dominant voltages, it discards any measurements that are lower than 2.75V on CANH and higher than 2.25V on CANL. The measurement continues until some message is shown to have been received into `Viden`'s receive message buffer, i.e., an indication that the transmitter has finished sending a message. By reading the ID value of that received message, `Viden` knows which message ID the acquired dominant voltages correspond to.

Non-ACK voltages. `Viden` continues collection of more dominant voltages for the acquired ID (whenever the message is received) until it learns its CANH and CANL ACK thresholds. When collecting and exploiting voltage measurements, one needs to be cautious of the fact "During the ACK slot of a transmitted message, if received, all other nodes but its transmitter output a 0-bit on the CAN bus" [2]. Thus, even though `Viden` samples at least a few dominant voltages while receiving a certain message, *not all* represent the outputs from the actual message transmitter. Fig. 4.6 shows an example of `Viden`'s five voltage measurements of {3.4V, 2.6V, 2.5V, 3.3V, 3.8V} from the CANH line during the reception of a message, where 3.8V was measured during the ACK slot. Of them, `Viden` discards measurements $\{2.6V, 2.5V\}$ as they do not meet the criteria of dominant voltages. If `Viden` had considered the remaining 3 measurements as if they were output by the message transmitter, it would have been incorrect since 3.8V was from all ECUs but the message transmitter in the ACK slot. Therefore, to accurately fingerprint the transmitter ECU, `Viden` derives the ACK threshold which distinguishes a non-ACK voltage measurement from an ACK voltage measurement. We refer to *non-ACK voltages* as dominant voltages measured from slots other than the ACK slot, and *ACK voltages* as those measured from the ACK slot. The threshold is derived by exploiting the following two facts of the ACK voltage.

- $\mathbb{K}1$. *Low probability*: Since ACK is only 1 bit long, when measuring dominant voltages during a message reception, most of them would be outputted from the message transmitter.
- $\mathbb{K}2$. *Different voltage level for ACK*: During an ACK slot, all ECUs but the transmitter acknowledge their message reception. Since those responders are connected in parallel and turned on concurrently, when receiving the ACK, the measured voltages are much higher on CANH and much lower on CANL than those when receiving non-ACK bits.

`Viden` exploits these facts to collect *M* dominant voltages from both CANH and CANL for *N* rounds for a given message ID. So, based on $\mathbb{K}1$, the *most frequently* measured voltage value (of the *M* values) will most likely represent the non-ACK voltage. During the *N* rounds, we refer to the set of *N* most frequently measured values as the *most frequent set*, S_{freq}. On the other hand, if we were to determine the *maximum* and the *minimum* of the *M* values from CANH and CANL, respectively, then they would represent ACK as well as non-ACK voltages. This is because even a single dominant voltage value collected (without awareness) from the ACK slot would become the maximum/minimum of the *M* values due to $\mathbb{K}2$. Here, the set of *N* maximum/minimum values measured from CANH/CANL is

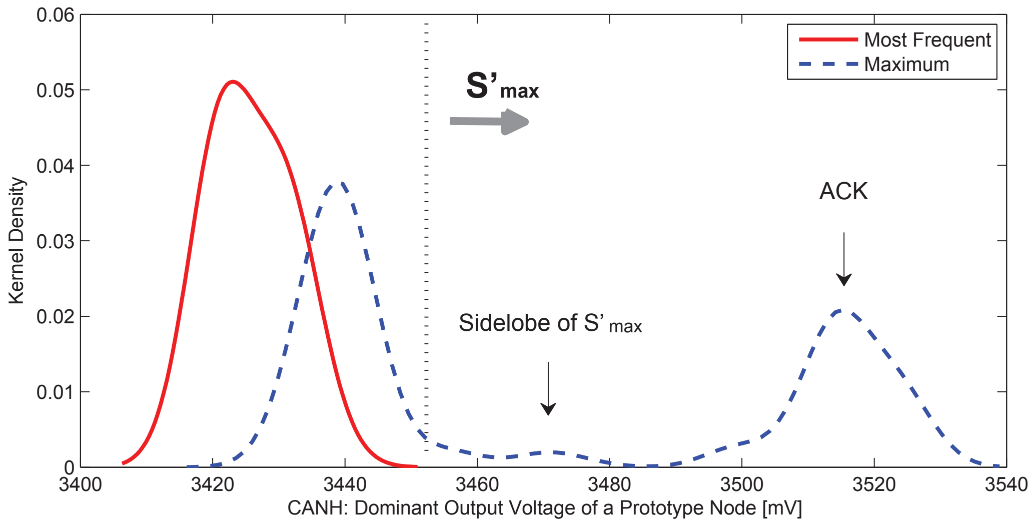


Figure 4.7: ACK threshold in a CAN bus prototype.

defined as the *maximum/minimum set*, denoted as $S_{max/min}$. For each message ID, `Viden` exploits sets S_{freq} and $S_{max/min}$ to derive the ACK threshold that differentiates a non-ACK voltage from an ACK voltage.

Derivation of ACK threshold. Fig. 4.7 shows the kernel density plots of the most frequent and the maximum sets of the measured dominant voltages from the CANH line. The measurements were made while running `Viden` on our CAN bus prototype, which will be detailed in Section 4.4. One can see that only for the maximum set, there exists a side lobe, whereas the most frequent set resembles a Gaussian distribution. Note that during the N rounds of M measurements each, the most frequent and the maximum values can be different. Thus, from the maximum set, `Viden` first discards values lower than $\max(S_{freq}) + B\sigma_{S_{freq}}$, where $\sigma_{S_{freq}}$ is the standard deviation of set S_{freq}, and B a design parameter determining how aggressive one wants to be in discarding ACK voltages. Note that such a value also represents the rightmost end-point of the most frequent set's kernel density (e.g., dotted vertical line in Fig. 4.7). Then, the usual side lobe of the maximum set (S_{max}) becomes the main lobe of a refined maximum set, S'_{max}. From S'_{max}, `Viden` determines $\Gamma_1 = median(S'_{max}) - 3MAD(S'_{max})$ and $\Gamma_2 = \mu_{S'_{max}} - 3\sigma_{S'_{max}}$, where $MAD(x)$ denotes the median absolute deviation of x, and μ_x its mean. The CANH ACK threshold of the given message ID (or its transmitter), Γ^H_{ACK}, is then derived to be $\max(\Gamma_1, \Gamma_2)$. We take the maximum of the two to be conservative in discarding any non-ACK voltages. Moreover, not only the lower 3σ limit but also the lower 3-MAD limit is used since the refined maximum set S'_{max} may still contain its own (new) side lobe as shown in Fig. 4.7, i.e., an outlier for S'_{max}. Using these processes, the ACK threshold of the example in Fig. 4.7 is determined to be $\Gamma^H_{ACK} = 3.499$V — a point where the two lobes in the maximum set are separated. Depending on the transmitter ECU, the ACK threshold can be different as the set of responders is different. Thus, the ACK learning is performed for all message IDs of interest.

When deriving the CANL ACK threshold, Γ^L_{ACK}, the minimum (instead of the maximum) and the upper (instead of the lower) limits are used. Later through evaluation, we show that the proposed scheme can correctly determine the ACK thresholds even in real vehicles.

4.3.5 Phase 2: Deriving a Voltage Instance

Once ACK thresholds, Γ^H_{ACK} and Γ^L_{ACK}, of the given message ID are learned, from that point and on, `Viden` continuously collects dominant voltages, but discards those from CANH that are lower than 2.75V or higher than Γ^H_{ACK}, and those from CANL that are higher than 2.25V or lower than Γ^L_{ACK}. This way, `Viden` selects and processes only *non-ACK voltages*. Whenever `Viden` obtains κ new measurements of CANH and CANL non-ACK voltages, `Viden` derives a new *voltage instance* which is defined as the set of 6 tracking points, $\mathbf{F}_1$–$\mathbf{F}_6$.

$\mathbf{F}_1$–$\mathbf{F}_2$: Most frequent values. Similarly to Phase 1, `Viden` determines the most frequently measured CANH and CANL voltages (from κ values), which are denoted as F_1 and F_2, respectively. Since `Viden` knows the ACK thresholds, the main differences from Phase 1 are that only non-ACK voltages as well as κ ($< M$) of them are used in deriving the most frequent values. This way, `Viden` keeps track of the *median* of the transmitter's

Algorithm 2 Dispersion Update

```
1: function UPDATEDISPERSION($V, \Lambda, P^*$)
2:     return $\Lambda \leftarrow \Lambda + \alpha(P^* - \frac{\#(V<\Lambda)}{\#V})^3$          ▷ Adjust tracking position
3: end function
4: if #measured CANH and CANL voltages both $\geq \kappa$ then
5:     $V_H, V_L \leftarrow$ {past $\kappa R$ CANH, CANL measurements}
6:     $F_3 \leftarrow$ UPDATEDISPERSION($V_H, F_3, 0.75$)
7:     $F_4 \leftarrow$ UPDATEDISPERSION($V_L, F_4, 0.25$)
8:     $F_5 \leftarrow$ UPDATEDISPERSION($V_H, F_5, 0.9$)
9:     $F_6 \leftarrow$ UPDATEDISPERSION($V_L, F_6, 0.1$)
10: end if
```

dominant voltages.

F_3–F_6: Dispersions. `Viden` also keeps track of the *dispersions* of CANH and CANL dominant voltages. As the transmitter's voltage output behavior can change over time, `Viden` continuously updates 4 different *tracking points*, F_3–F_6, which reflect (1) F_3: 75th, (2) F_5: 90th percentile of the transmitter's CANH outputs, (3) F_4: 25th, and (4) F_6: 10th percentile of CANL outputs. By tracking the transmitter's voltage distribution, `Viden` understands its *momentary* voltage output behavior. Thus, voltage instances represent those momentary behaviors. Since even a single ACK voltage can significantly distort `Viden`'s understanding of transmitters' behaviors, it is important to learn the ACK threshold. The reasons for `Viden`'s tracking of different percentiles of CANH and CANL are that the low percentiles of CANH would contain voltages measured when the transmitter switches from sending a 1-bit to sending a 0-bit, and vice versa. The same applies for the high percentiles of CANL measurements. Although other percentiles can be tracked as well, to minimize `Viden`'s overhead, we only track F_3–F_6.

Algorithm 2 describes how the tracked dispersions are updated whenever `Viden` acquires κ dominant voltages from each of CANH and CANL. Using the past κR measurements, as in line 2, `Viden` roughly estimates what percentile the current tracking point, Λ, represents. In `Viden`, we set $R = 10$. Then, to correct and thus move the tracking point Λ to the desired position — where it represents the P^* percentile — an adjustment is made as in line 2, where α is a design parameter determining the sensitivity to changes.

With the adjustment function proportional to $(P^* - \frac{\#(V<\Lambda)}{\#V})^3$, the tracking points move faster if they are far away from their desired positions. As a result, the four tracking points move if the transmitter's voltage distribution (i.e., output behavior) shows changes, thus adapting to any changes on the CAN bus. Instead of tracking, it is also possible to directly derive the percentiles from the κR values. `Viden`, however, does not follow this since it is too sensitive to transient changes, especially when κR is small, i.e., insufficient samples in deriving the percentiles. Thus, in order to make `Viden` work under various circumstances, we *track* them instead.

4.3.6 Phase 3: Attacker Identification

A voltage instance (F_1–F_6) represents the momentary voltage output behavior of the message transmitter. So, to log its usual behavior, `Viden` exploits every newly derived voltage instance to construct/update the *voltage profile* of the message transmitter. Although the voltage instances are derived "per message ID", if messages originate from the same transmitter/ECU, their instances are near-equivalent, thus leading to construction of the same voltage profile. We will later show through evaluations that there exists only one voltage profile for a given transmitter/ECU, thus enabling its fingerprinting. By exploiting a newly derived voltage instance, `Viden` first updates the *cumulative voltage deviations* (CVDs) of features F_1–F_6. We define a CVD to represent how much the transmitter's dominant voltages deviated cumulatively from their ideal values. Thus, for feature F_x, the CVD at step n, $CVD_x[n]$, is updated as:

$$CVD_x[n] = CVD_x[n-1] + \Delta[n]\,(1 - \nu_x[n]/\nu_x^*)\,, \tag{4.2}$$

where $\Delta[n]$ is the elapsed time since step $n-1$, $\nu_x[n]$ the value of feature F_x at step n, and ν_x^* the desired value of ν_x. Ideally, the most frequently measured as well as any percentiles of the CANH and CANL dominant voltages should be equal to 3.5V and 1.5V, respectively, i.e., no variations in their output voltages. Therefore, for features $\{F_1, F_3, F_5\}$, which represent CANH values, we set $v^*_{\{1,3,5\}} = 3.5V$ and similarly we set $v^*_{\{2,4,6\}} = 1.5V$.

Suppressing transient changes. As ECUs have different V_{CC}, ground, and R_{DSON} values, they output different CANH and CANL dominant voltages. Their momentary voltage instances would, therefore, be different, and hence the trends in their CVD changes would also be different from each other. So, for every obtained CVD of features F_1–F_6, `Viden` derives $\Psi[n] = \sum_{x=1}^{6} CVD_x[n]$. The reason for `Viden`'s summing of all the CVDs is to exploit $\mathbb{V}3$. Recall from Section 4.3.3 that $\mathbb{V}3$ gives us transient deviations in CANH and CANL output voltages are opposite in direction. So, via CVD summation, `Viden` *suppresses* any transient deviations that have occurred (due to changes in driver, temperature, etc.) when constructing and/or updating the voltage profiles. Note that since CAN is a differential bus, F_2, F_4, F_6 suppress F_1, F_3, F_5, respectively.

Voltage profile. Suppression of transient changes yields a value, Ψ, that (mostly) represents the *consistent* factors in the voltage instances: V_{CC}, ground voltages, and the usual voltage drops across the transistors. As stated in $\mathbb{V}4$, since these values are rather constant, the accumulated sum of Ψ, $\Psi_{accum}[n] = \sum_{k=1}^{n} \Psi[k]$ becomes linear in time. Moreover, from $\mathbb{V}1$–$\mathbb{V}2$, as Ψ values are distinct for different ECUs, the *trends* in how Ψ_{accum} changes also become different, i.e., the slopes in a Ψ_{accum}–time graph are different. Therefore, `Viden` formulates a linear parameter identification problem as $\Psi_{accum}[n] = \Upsilon[n]t[n] + e[n]$, where at step n, $\Upsilon[n]$ is the regression parameter, $t[n]$ the elapsed time, and $e[n]$ the identification error. As the regression parameter Υ represents the slope of the linear model and varies with the transmitter, we define this as the *voltage profile*. This way of formulating the problem and constructing the profiles facilitates `Viden`'s online update of fingerprints, which is key to `Viden`'s adaptability. To determine the voltage profile Υ, i.e., fingerprint ECUs, we use an adaptive signal processing technique, the Recursive Least Squares (RLS) [43], which is an online approach in learning the regression parameter. Note, however, that the choice of algorithm does not affect Viden's performance. In RLS, we use kiloseconds (=10^3 secs) as the unit for t. Due to space limitation, we omit details of RLS, and refer the readers to [43] for its details. We will later show, via experimental evaluations, that the thus-derived profile Υ is constant over time and also distinct for different ECUs, thus allowing Viden to correctly fingerprint them.

Identifying the attacker. When an adversary mounts an attack, the underlying IDS can determine whether the message is malicious or not, so Viden can filter out the voltage outputs obtained only from the (detected) attack messages and build a voltage profile from only those. We refer to such a voltage profile as an *intrusion voltage profile*. Viden then looks up the voltage profiles it had built until the detection of the attack and searches for the one that is similar to the intrusion voltage profile.[1] This way, Viden identifies the attacker ECU.

The performance of Viden will, of course, depend on how well the IDS detects the intrusion; this dependency needs to be investigated when an IDS and Viden are integrated as a whole system. Note, however, that the mostly periodic nature of in-vehicle messages makes correct detection of intrusions not as difficult as pinpointing the attacker ECU. Researchers and car-makers are now well aware of how to detect intrusions, but not how to accurately identify the attacker ECU.

The only case where the identified ECU would have an unknown/unlearned profile is when it was physically attached to the vehicle by an adversary. However, since this requires physical access and its identification has been addressed elsewhere [35, 72], we do not discuss its detection any further in this chapter.

4.3.7 Phase 4: Verification

By the birthday paradox, two different ECUs may naturally have near-equivalent voltage profiles, i.e., voltage profile collision, thus confusing Viden in identifying

[1]The initial set of "ground truth" voltage profiles can be verified via timing-based fingerprinting schemes [33, 68].

the attacker ECU. Note, however, that `Viden` has at least narrowed its search scope significantly. An adversary may also attempt to mimic some other ECU's voltage output behavior, i.e., targeted impersonation. In such a case where further verification besides the voltage profiles is required, in Phase 4 of `Viden`, machine classifiers are run with the (momentary) *voltage instances* as their inputs, i.e., F_1–F_6 as their features. This way, an analysis of attacks from a different vantage point — not only its trend (Phase 3) but also its momentary behavior — is performed, thus resolving ambiguities in attacker identification. We, however, stress that while the adaptability achieved from Phase 3 is an essential attribute for an accurate attacker identification, Phase 4 cannot totally replace it, i.e., only complements Phase 3. We will later show through evaluations that by using voltage instances as machine classifiers' input, `Viden` can resolve issues such as voltage profile collision and an adversary's targeted impersonation.

4.3.8 Voltage Profile Adjustment

For attacker identification, it is important to not only have the correct fingerprint of an ECU but also that fingerprint to be still valid when examining a voltage measurement obtained during the (detected) attack. If it was updated much earlier than when the attack was detected, any changes occurred between those two time instants would not be reflected in the latest model, thus leading to false identifications. We refer to this as a *model-exam discrepancy*. Since `Viden` continuously updates the voltage profiles in real time, such a model-exam discrepancy is minimized/nullified. Since attacker identification is performed *upon* detection of an intrusion, as long as `Viden` keeps the fingerprints up-to-date until an intrusion is detected (by an IDS), `Viden` can locate the source of the attack. Even when there are abrupt changes in the temperature of an ECU, `Viden` suppresses those transient changes, and adapts its model accordingly for an accurate attacker identification.

One corner case in which the performance of `Viden` might suffer from the model-exam discrepancy would be when the vehicle has not been turned on for a long time. During that period, various features (e.g., power supply level, ambient temperature) which affect the output voltages might have changed. In such a case, since the old voltage profiles may not correctly reflect the current status, `Viden` may have to reconstruct (instead of update) them. In fact, a timing-voltage-aware adversary may attempt to exploit such a fact and attack the CAN bus as soon as the vehicle is turned on, making `Viden` incapable of handling the attacks. However, even in such a case, as ECUs use the same power source, i.e., battery, and thus all voltage profiles change in the same direction and with the same magnitude, `Viden` re-adjusts and reuses the old ones as a starting point for voltage profile *update* rather than reconstructing it from scratch when the vehicle is turned on. Specifically, `Viden` first determines how much of *common* changes occurred in ECUs' V_{CC} by deriving the differences between the previous and current mean values of $(F_3+F_4+F_5+F_6)/2$ — an estimated value of V_{CC} based on Eq. (4.1). `Viden` then adds the thus-derived differences to v^* (in Eq. (4.2)) based on the fact that if common changes in V_{CC} incur, CANH and CANL output values increase simultaneously [16, 18]. This way, `Viden` correctly adjusts/updates its voltage profile(s) and thus identifies such a type of timing-voltage-aware adversary; we will later evaluate this via real vehicle experiments. Note, however, that if voltage-based fingerprinting was done solely via batch learning (as in [35, 64]), it cannot make such an adjustment, suffer from high model-exam discrepancy, thus allowing a timing-voltage-aware adversary to evade it.

4.3.9 Security of `Viden`

Once an intrusion is detected, via voltage measurements, `Viden` can identify the attacker ECU.

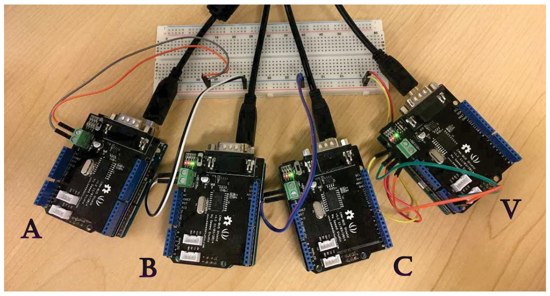
A *naive* adversary would be capable of controlling the vehicle via continuous message injections. However, since he has no knowledge of how ECUs might be fingerprinted, he would inject them imprudently. In such a case, he cannot evade `Viden`.

A *timing-aware* adversary who knows that ECUs can be fingerprinted via timing analysis, will attempt to exploit this knowledge in not only controlling the vehicle but also evading the fingerprinting device. For example, the adversary may know that CIDS (Chapter III) can identify the attacker ECU only if the attack messages were injected periodically. Hence, he may perform an arbitrary impersonation by injecting messages *aperiodically*, thus fooling CIDS. Note, however, that CIDS would still detect the presence of the attack. In addition, based on his knowledge that CIDS's fingerprints are basically clock skews, he may attempt to imitate the targeted ECU's clock behavior, i.e., targeted impersonation. However, with Viden also installed in the vehicle, since it identifies the attacker ECU via voltage measurements, i.e., irrespective of message timings, a timing-aware adversary can evade CIDS, but not Viden.

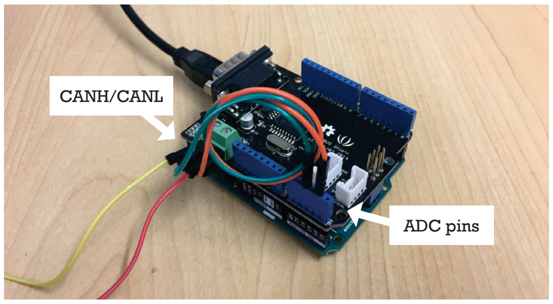
A *timing-voltage-aware* adversary may also try to evade Viden using his knowledge of how voltage-based fingerprinting devices run. In order to achieve this, when or before the adversary injects the attack messages, he may attempt to change the voltage output levels by changing the supply voltage (e.g., run processes which drain battery) or by heating up or cooling down the ECUs so that the transistors' internal resistance values change. He could even attempt to start attacking the CAN bus only when the vehicle is turned on after staying off for a long time as discussed in Section 4.3.8. However, since Viden performs an online update of voltage-based fingerprints and also adjusts them if necessary, thus minimizing/nullifying model-exam discrepancy, it would be difficult for the timing-voltage-aware adversary to evade Viden. Moreover, since Viden analyzes voltage outputs from two different perspectives — momentary behavior (Phase 4) and its trend (Phase 3) — a timing-voltage-aware adversary incapable of precisely controlling the instantaneous voltage outputs cannot evade Viden.

4.4 Evaluation

We now evaluate the practicability and efficiency of Viden in achieving an effective and accurate attacker identification on a CAN bus prototype and two *real* vehicles. When

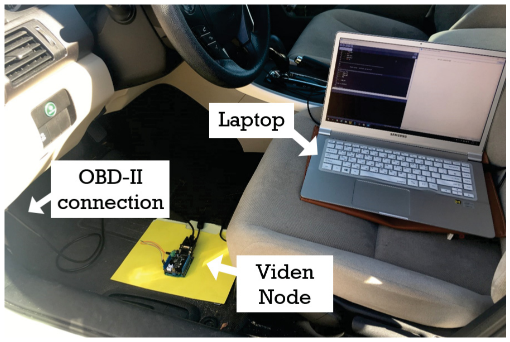


(a) CAN bus prototype.



(b) Prototype node.

(c) 2015 Chevrolet Trax.



(d) Connection to the vehicle.

Figure 4.8: Experiments were conducted on a CAN bus prototype and on two real vehicles.

running `Viden` for both evaluation settings, in Phase 1, $M = 30$ dominant voltages were obtained for each message ID for $N = 50$ rounds. From Phase 2, voltage instances were outputted whenever $\kappa = 15$ non-ACK voltages from both CANH and CANL were acquired.

4.4.1 Evaluation Setups

CAN bus prototype. Different from previous CAN bus prototype settings (in Chapter II and Chapter III), as shown in Fig. 4.8a, this time, we configured the CAN prototype to have four nodes. Three prototype nodes $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$ were programmed to inject messages 0x01, 0x07, and 0x15 at random message intervals within [20ms, 200ms]. The fourth node $\mathbb{V}$ was programmed to run `Viden` and construct voltage profiles for messages 0x01, 0x07, and 0x15 (i.e., transmitters $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$), respectively. The reason for injecting the messages *aperiodically* is to show that even in such cases, `Viden` is capable of fingerprinting the transmitters. For node $\mathbb{V}$ that runs `Viden`, as shown in Fig. 4.8b, its CANH and CANL lines were not only connected to the bus but also to the

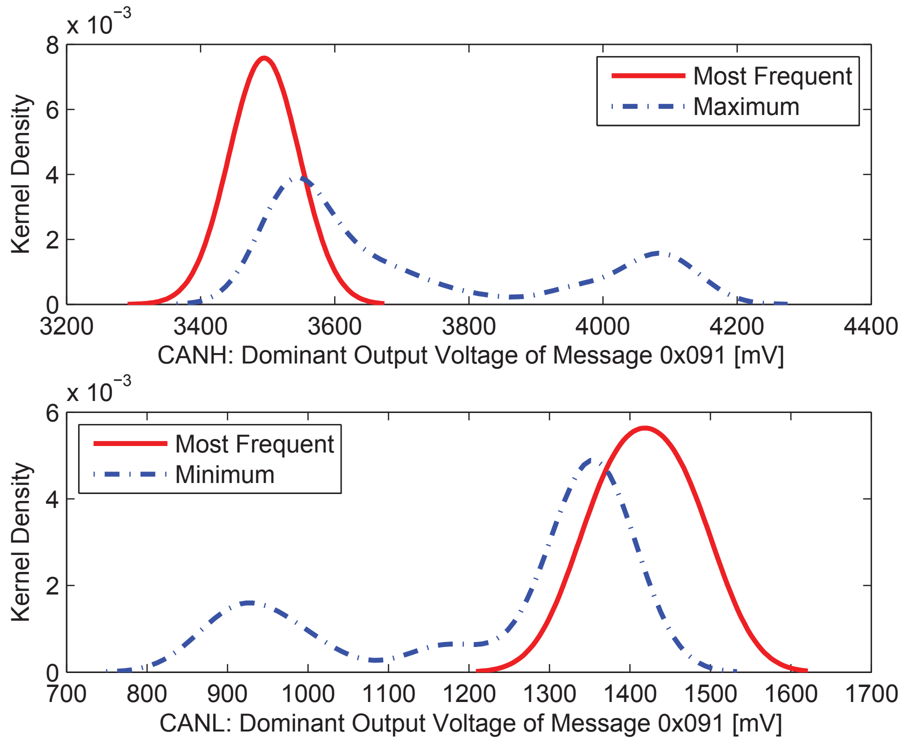


Figure 4.9: Deriving the ACK thresholds for message 0x091 in the 2013 Honda Accord.

microcontroller's Analog-to-Digital Converter (ADC), which had 10-bit resolution and was configured to sample voltages at its maximum rate of 50 KSamples/sec. This way, $\mathbb{V}$ acquired measurements of dominant voltages on the bus when nodes $\mathbb{A}$–$\mathbb{C}$ were sending their messages. The CAN bus prototype was set up to operate at 500Kbps, which is typical for in-vehicle high-speed CAN buses. In such settings, `Viden` required only 2–3 messages to output a voltage instance and update the profiles.

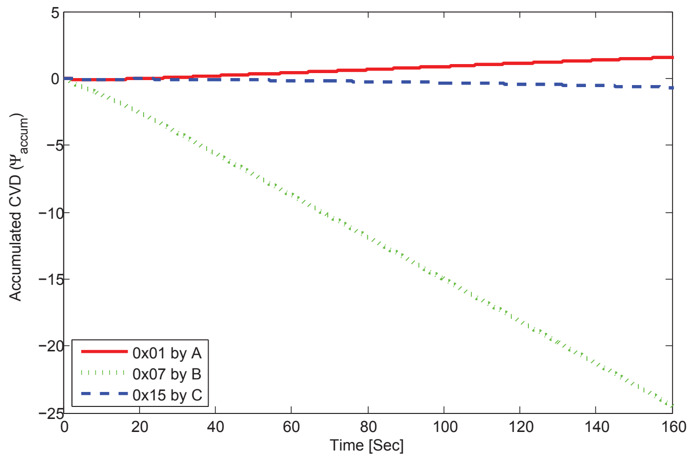
Real vehicles. Two cars, the 2013 Honda Accord (Fig. 2.17a) and a 2015 Chevrolet Trax (Fig. 4.8c), were also used for our experimental evaluation of `Viden`. Through the OBD-II port, the `Viden` node ($\mathbb{V}$) was connected to the in-vehicle CAN bus, both running at 500Kbps. From a laptop and through the `Viden` node, as shown in Fig. 4.8d, we were able to read messages from the 2013 Honda Accord's and the 2015 Chevrolet Trax's CAN buses. While `Viden` was receiving messages from the two vehicles, it sampled their CANH and CANL voltages and then derived their ECUs' voltage instances and profiles.

4.4.2 ACK Threshold Learning In a Real Vehicle

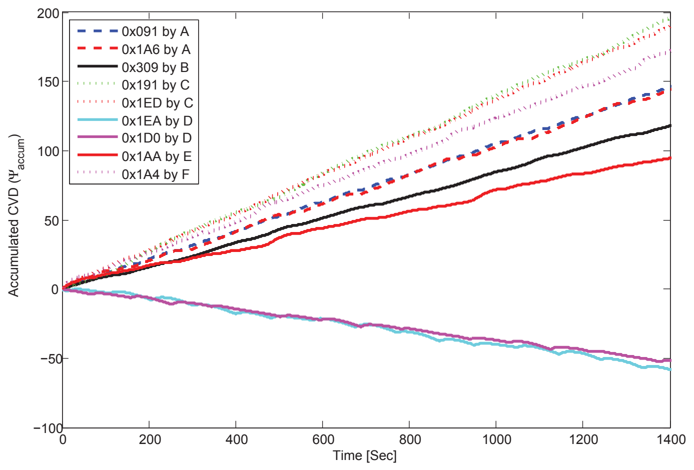
`Viden` first learns the thresholds which determine whether the measured voltages are from the ACK slot or not, before outputting voltage instances and profiles. This was achieved by determining most frequent and maximum/minimum sets, and exploiting the side lobe which only exists in the latter. In other words, the existence of such a side lobe (as shown in Fig. 4.7 in a CAN prototype), which represents the distribution of ACK voltages, is critical in learning the ACK threshold. Thus, to show that the proposed ACK learning is feasible even in real vehicles, through `Viden`, we obtained both most frequent and maximum/minimum sets for message ID=0x091, which was sent every 10ms by some ECU in the 2013 Honda Accord.

Fig. 4.9 (upper) shows the kernel density plots of the most frequent and maximum sets of the CANH dominant voltages while receiving ID=0x091, and Fig. 4.9 (lower) the kernel density of those obtained from the CANL line. One can see that as in the CAN prototype result (Fig. 4.7), side lobes exist in both the CANH and CANL lines. Thus, the proposed ACK learning mechanism in `Viden` derived the refined maximum and minimum sets, S'_{max} and S'_{min}, correctly and thus derived the ACK thresholds of message 0x091 to be 3.844V for CANH outputs and 1.114V for CANL outputs.

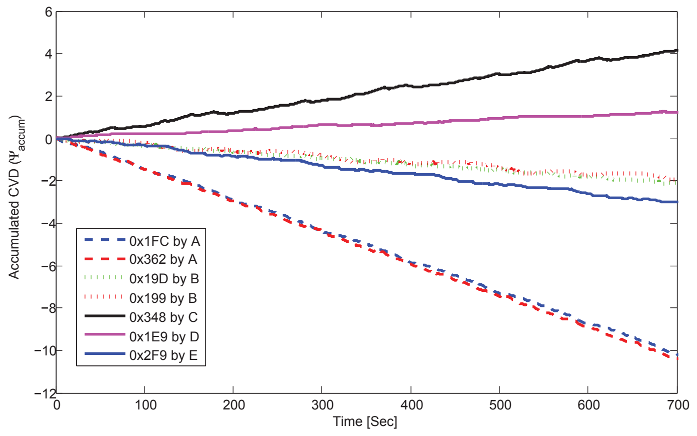
One interesting observation is how high and low CANH and CANL ACK voltage levels are. In our evaluation of `Viden` on the CAN prototype, since we had only 3 nodes acknowledging to the message, the median of the CANH ACK voltages was 3.514V as shown in Fig. 4.7. On the other hand, in our experiment on the 2013 Honda Accord, the median of the CANH ACK voltage showed a high voltage level, 4.049V — much higher than the one obtained from the CAN prototype. For the CANL ACK voltage, the median was 0.953V. Such a result is due to the fact that there were much more ECUs (compared to 3 in the prototype) inside the vehicle which ACKed message 0x091.



(a) CAN bus prototype.



(b) 2013 Honda Accord.



(c) 2015 Chevrolet Trax.

Figure 4.10: Voltage profiles obtained from the CAN bus prototype and the two real vehicles.

4.4.3 Voltage Profiles as Fingerprints

We first evaluate the accuracy and validity of using voltage profiles to fingerprint the transmitter ECUs.

CAN bus prototype. Fig. 4.10a shows the voltage profiles of all the three messages sent on the prototype bus. Although the three CAN prototypes nodes were built with the same hardware, the corresponding message IDs showed different trends in how their Ψ_{accum} changed over time, since the three ECUs differ in their supply and transistor characteristics. Based on the RLS implemented in `Viden`, we were able to find that nodes $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$ had different voltage profiles (Υ) being equal to 10.1, -154.3, and -4.9, respectively. In other words, voltage profiles of 0x01, 0x07, and 0x15 were shown to be different from each other as they were sent by different ECUs, thus verifying the feasibility and accuracy of `Viden`.

Real vehicles. In the CAN prototype, we knew which ECU is sending which message(s), but it is difficult to know this in a real vehicle. In order to obtain the ground truth on the message source(s), we exploit the schemes in [33, 68], which analyzed timing patterns in CAN for fingerprinting the ECUs. Note, however, that these are used only for obtaining the ground truth, since those cannot identify the attacker ECU if messages are injected at random times.

Through the connected `Viden` node, we not only logged the CAN traffic of the 2013 Honda Accord but also measured the dominant voltages from its CAN bus. The measurements were made on a stationary vehicle, but while continuously changing their operations (e.g., pressing brake pedal, turning the steering wheel) to generate some transient changes. We later show that outputs in `Viden` is *not* affected by whether the car is being driven or stationary. By logging the CAN traffic and exploiting the schemes in [33, 68], we were able to verify that messages {0x091, 0x1A6} were sent from some ECU $\mathbb{A}$, {0x309} from $\mathbb{B}$, {0x191, 0x1ED} from $\mathbb{C}$, {0x1EA, 0x1D0} from $\mathbb{D}$, {0x1AA} from $\mathbb{E}$, and {0x1A4} from $\mathbb{F}$. Fig. 4.10b shows the messages' voltage profiles. The profiles (Υ) derived by `Viden` are shown to be equivalent *only* for those messages sent from the same ECU; ECU $\mathbb{A}$ sending {0x091, 0x1A6} had $\Upsilon_{\mathbb{A}} = 102.6$, $\mathbb{B}$ sending {0x309} had $\Upsilon_{\mathbb{B}} = 85.0$, $\mathbb{C}$ sending {0x191, 0x1ED} had $\Upsilon_{\mathbb{C}} = 137.0$, $\mathbb{D}$ sending {0x1EA, 0x1D0} had $\Upsilon_{\mathbb{D}} = -39.2$, $\mathbb{E}$ sending {0x1AA} had $\Upsilon_{\mathbb{E}} = 67.5$, while $\mathbb{F}$ sending {0x1A4} had $\Upsilon_{\mathbb{F}} = 120.8$. This result again shows that voltage profiles for *different* ECUs are different and can thus be used as their fingerprints.

To further verify that `Viden`'s capability of fingerprinting is not restricted to a specific vehicle model, `Viden` was also run on a 2015 Chevrolet Trax. Again, by exploiting the schemes in [33, 68], we obtained the ground truths of messages {0x1FC, 0x362} sent from some ECU $\mathbb{A}$, {0x19D, 0x199} from $\mathbb{B}$, {0x348} from $\mathbb{C}$, {0x1E9} from $\mathbb{D}$, and {0x2F9} from $\mathbb{E}$. Fig. 4.10c shows the result of `Viden` determining that {0x1FC, 0x362} have a voltage profile of $\Upsilon_{\mathbb{A}} = -14.7$, {0x19D, 0x199} have $\Upsilon_{\mathbb{B}} = -2.8$, {0x348} has $\Upsilon_{\mathbb{C}} = 5.9$, {0x1E9} has $\Upsilon_{\mathbb{D}} = 1.8$, and {0x2F9} has $\Upsilon_{\mathbb{E}} = -4.4$. Thus, using voltage measurements, `Viden` correctly fingerprinted their transmitters. This again confirms the diversity of voltage profiles (of different ECUs), thus facilitating `Viden`'s fingerprinting of in-vehicle ECUs. Moreover, these results show that `Viden`'s fingerprinting is not limited to a specific vehicle model, and can thus be applied to other vehicle models.

4.4.4 Voltage profiles while driving

We further validate that `Viden`'s derived voltage profiles do not depend on whether and how the vehicle is driven. We first obtained the voltage instances of 0x191 from the 2013 Honda Accord's CAN bus. At this time of measurement, the vehicle was stationary. Later on that day, instances of 0x191 was once again obtained, but this time while driving the vehicle for approximately 10 mins; the same data which we used in Section 4.4.7.1.

Fig. 4.11 shows the voltage profiles of 0x191 obtained while the vehicle was stationary and driven. One can see that the two voltage profiles are equivalent, even though they were measured under a different condition. These are due to the fact that the voltage outputs are much more dependent on hardware components' characteristics than their momentary

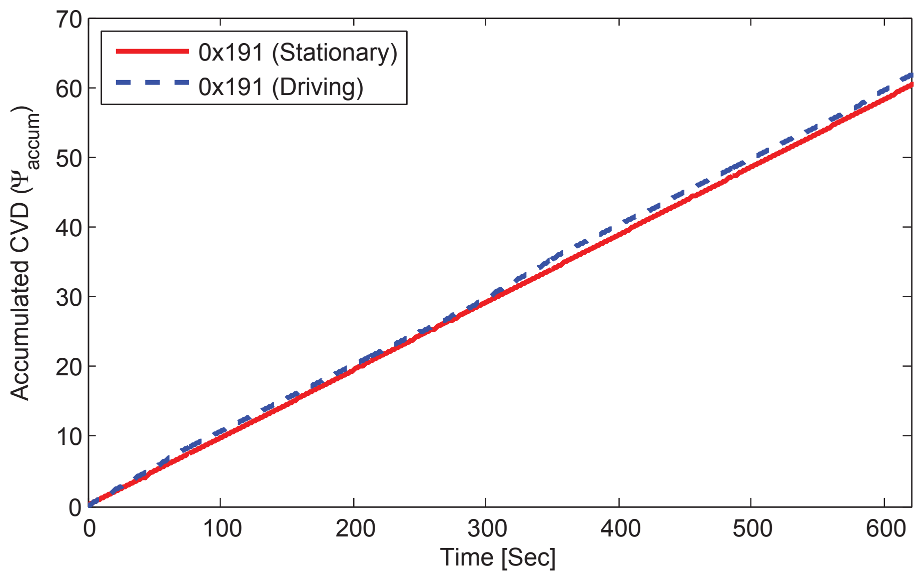
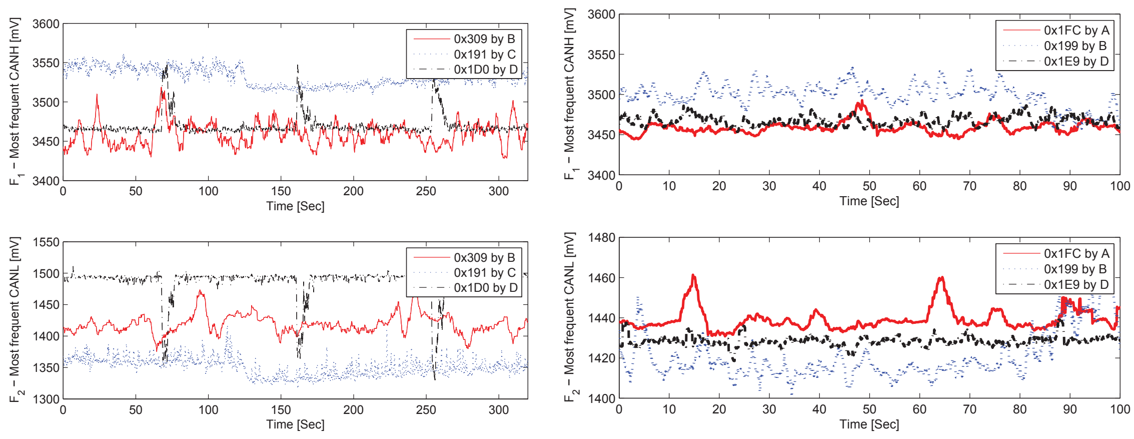


Figure 4.11: Voltage profiles of message 0x191 when the vehicle was stationary and driven.



(a) 2013 Honda Accord. (b) 2015 Chevrolet Trax.

Figure 4.12: Features F_1 and F_2 of `Viden` in the two real vehicles.

conditions. Moreover, transient deviations incurred from changes in momentary conditions would have been suppressed thanks to how `Viden` derives its voltage profiles; summing CVDs of CANH and CANL.

4.4.5 Voltage Outputs in Real Vehicles

We provided 4 characteristics, $\mathbb{V}1$–$\mathbb{V}4$, which were imperative for `Viden` in fingerprinting ECUs. We evaluate whether $\mathbb{V}1$–$\mathbb{V}3$ actually hold in real vehicles. Note that

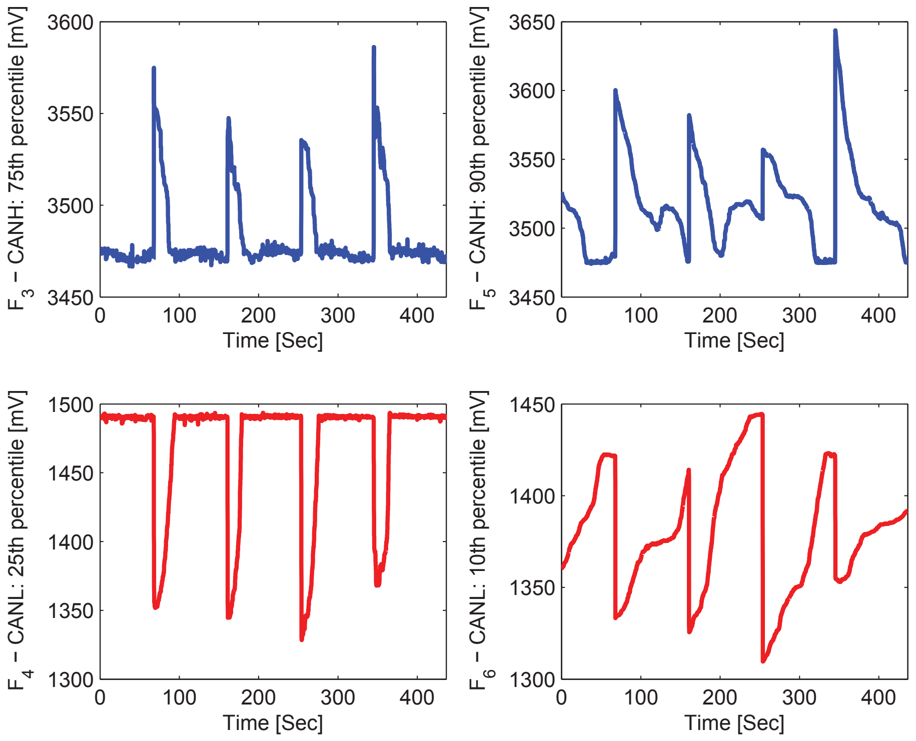
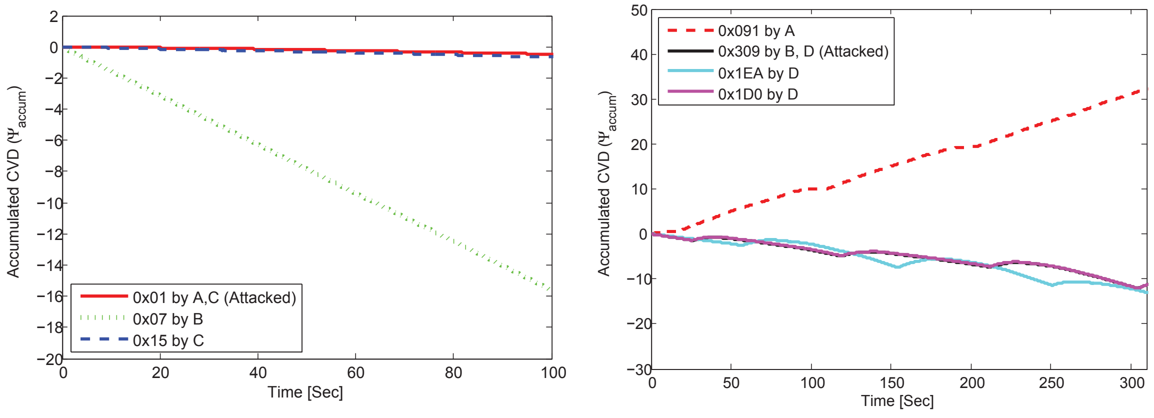


Figure 4.13: Changes of message 0x1D0 in the Honda Accord.

Fig. 4.10 verifies $\mathbb{V}4$, corroborating that the voltage profiles of ECUs were constant over time, i.e., linear.

Different outputs. According to $\mathbb{V}1$–$\mathbb{V}2$, ECUs output different dominant voltages. Fig. 4.12a plots features F_1–F_2 (i.e., the most frequently measured CANH and CANL values) outputted by `Viden` for messages 0x309 (sent by $\mathbb{B}$), 0x191 (sent by $\mathbb{C}$), and 0x1D0 (sent by $\mathbb{D}$) in the Honda Accord. Although the transceivers of all those messages are to output the agreed-on CANH=3.5V and CANL=1.5V when sending a 0-bit, they outputted values deviating from them. More importantly, their output levels were clearly discriminable. Even though ECU $\mathbb{B}$, which sent 0x309, was shown to output similar CANH dominant voltages to ECU $\mathbb{D}$, it outputted totally different voltages on CANL. Similarly, Fig. 4.12b plots F_1–F_2 values of 0x1FC (sent from $\mathbb{A}$), 0x199 (sent from $\mathbb{B}$), and 0x1E9 (sent from $\mathbb{D}$) outputted by `Viden` in the 2015 Chevrolet Trax. Again, we can see that the transmitters of those messages did not output the desired levels, but outputted discernible



(a) CAN bus prototype.

(b) Real vehicle.

Figure 4.14: `Viden` identifying a timing-aware adversary.

levels. These results confirm that $\mathbb{V}1$–$\mathbb{V}2$ hold even in real vehicles, thus facilitating `Viden`'s fingerprinting.

Transient changes. $\mathbb{V}3$ states that transient changes in CANH and CANL voltages are opposite in direction. Fig. 4.13 shows the 4 tracked percentiles, F_3–F_6, of message 0x1D0 in the 2013 Honda Accord. F_3–F_6 values are shown to temporarily deviate from and later return to their usual values. Since F_3 and F_5 are inverses of F_4 and F_6, respectively, vertically reversed shapes of the former resemble those of the latter. Thus, summing them suppressed their transient deviations when deriving the voltage profiles. Note, however, that since the tracked values in `Viden` depend on the time of sampling and its accuracy, the summation did not completely remove the deviations, but it sufficed for fingerprinting.

4.4.6 Against a Timing-Aware Adversary

We evaluated `Viden`'s performance of attacker identification in the CAN bus prototype and in a real vehicle against a timing-aware adversary. We did not evaluate its performance against a naive adversary since the timing-aware adversary subsumes his capabilities.

CAN bus prototype. In the CAN bus prototype, we further programmed node $\mathbb{C}$ to be the timing-aware adversary who injects not only 0x15 but also attack messages with ID=0x01 at a random interval of 10–20ms; injecting messages aperiodically to perform arbitrary impersonation and thus evade timing-based fingerprinting devices. Note that 0x01 is also being sent from the legitimate node $\mathbb{A}$ at a random interval of 20–200ms. Fig. 4.14a shows the determined voltage profiles for all three messages during the mounted attack. Even though the voltage profile for 0x01 now reflects both the voltage outputs from $\mathbb{A}$ and $\mathbb{C}$, since the injection frequency from the attacker $\mathbb{C}$ was much higher, the voltage profile for 0x01 changed to a profile equivalent to the one shown in 0x15 (sent by $\mathbb{C}$). As a result, `Viden` determined that the transmitters of 0x01 and 0x15 are the same, thus identifying the source of the attack to be ECU $\mathbb{C}$. Note that even when the injection frequency is lower, the attacker ECU can be identified by observing the intrusion voltage profile.

Real vehicle. We also evaluated `Viden`'s performance against a timing-aware adversary in a real vehicle setting. We focus on the results obtained from the 2013 Honda Accord for the purpose of more in-depth discussion. We consider a scenario in which a timing-aware adversary controlling the Honda Accord ECU $\mathbb{D}$ attacks ECU $\mathbb{B}$ and also impersonates ECU $\mathbb{A}$, i.e., targeted impersonation. Thus, from the vehicle, `Viden` acquired voltage instances and profiles of the monitored messages: 0x091 sent from $\mathbb{A}$, 0x309 from $\mathbb{B}$, and {0x1EA, 0x1D0} from $\mathbb{D}$. To generate the scenario of $\mathbb{D}$ impersonating $\mathbb{A}$ (while attacking $\mathbb{B}$), $\mathbb{V}$ was further programmed to record only every 4-th message of 0x091 (sent by $\mathbb{A}$ every 15ms), and every 3rd message of 0x1D0 (sent by $\mathbb{D}$ every 20ms) as its ID to be 0x309. This was to emulate a scenario where the attacker $\mathbb{D}$ injects its attack messages with forged ID=0x309 at a *similar* frequency to $\mathbb{A}$, thus attempting to imitate its timing behavior for impersonation.

Fig. 4.14b plots the voltage profiles of {0x091, 0x1EA, 0x1D0} and the intrusion voltage profile of 0x309. Although the adversary attempted to impersonate ECU A, one can see that since `Viden` fingerprints the transmitter regardless of message timings, the intrusion voltage profile of 0x309 matched the profiles of {0x1EA, 0x1D0}. As a result, `Viden` concluded the attacker to be $\mathbb{D}$.

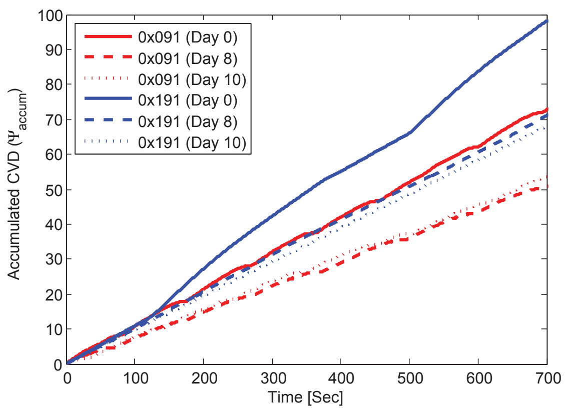
4.4.7 Against a Timing-Voltage-Aware Adversary

Based on his knowledge of voltage-based fingerprinting devices, a timing-voltage-aware adversary could attempt to evade `Viden` in two ways. First, the adversary might perform arbitrary impersonation by attacking the vehicle only when voltage-based fingerprints have not been updated for a long period of time, i.e., a high model-exam discrepancy. Next, the adversary might also perform targeted impersonation by changing its voltage output levels so as to imitate some specific ECUs' voltage output behavior.

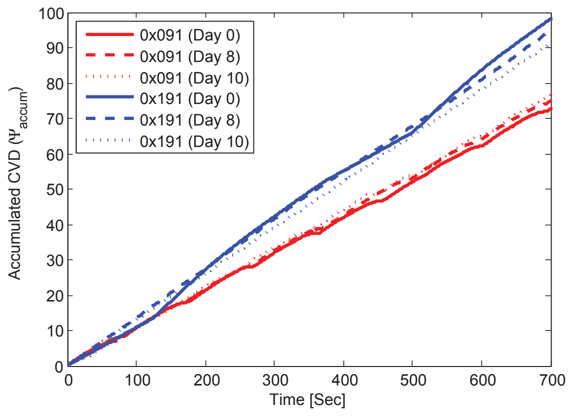
4.4.7.1 Arbitrary impersonation

In most cases of a timing-voltage-aware adversary performing arbitrary impersonation, `Viden` accordingly/adaptively updates the voltage profiles and can thus correctly identify the attacker. One corner case, however, in detecting the adversary would be when he performs arbitrary impersonation by attacking the vehicle only after a long idle period. To verify `Viden`'s reaction to such an adversary, we evaluated the following scenario. We first obtained the voltage profiles of 0x091 and 0x191 from the 2013 Honda Accord while driving the vehicle for approximately 10 mins. After 8 and 10 days had elapsed, we again obtained their profiles; the average temperatures during the three days were 14.4°C, 7.7°C, and 12.2°C, respectively. In between the three update dates, the vehicle was driven 700 miles and 40 miles to generate (on purpose) the considered scenario where the voltage profiles might be outdated, thus becoming a chance for the timing-voltage-aware adversary to perform arbitrary impersonation.

Fig. 4.15a shows the acquired voltage profiles corresponding to messages 0x091 and 0x191 on the three different dates. The initial profiles obtained were found different from those obtained on the 8-th and 10-th elapsed days, whereas the latter two were equivalent. One interesting observation, however, was that the voltage profiles of both message IDs were decreased by the *same* amount. The changes we observed were due to a slight shift in all ECUs' V_{CC} — most probably due to the change in the battery state after the long 700

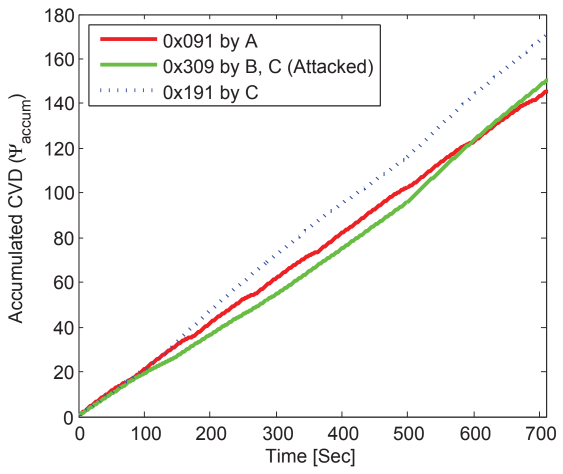


(a) Before adjustment.

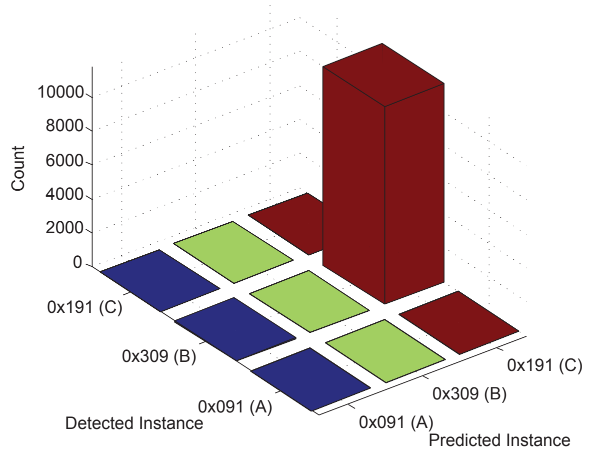


(b) After adjustment.

Figure 4.15: Adjusting voltage profiles of {0x091, 0x191}.



(a) Targeted impersonation.



(b) Predicting the attacker ECU.

Figure 4.16: Efficacy of `Viden`'s Phase 4 execution.

miles driving. In such a case, as we discussed in Section 4.3.8, `Viden` adjusts its voltage profiles. Once such an adjustment was made, we obtained the results shown in Fig. 4.15b, where all voltage profiles were properly aligned. This result shows that `Viden` is capable of handling cases where a timing-voltage-aware adversary performs arbitrary impersonation right after a vehicle's long idle period.

4.4.7.2 Targeted impersonation

Under scenarios where a timing-aware adversary performs arbitrary/targeted impersonation or where a timing-voltage-aware adversary performs an arbitrary impersonation, `Viden` can correctly identify him solely based on voltage profiles, i.e., within Phase 3. It could be much more challenging for `Viden` (requiring to run Phase 4) when a timing-voltage-aware adversary performs a *targeted* impersonation, i.e., trying to imitate some specific ECU's voltage outputs. Since the adversary creates a situation of at least two ECUs having similar voltage profiles (i.e., not unique), targeted impersonation would be more difficult for `Viden` to handle than arbitrary impersonation. We evaluated how `Viden` performs against such an adversary via (1) vehicle experiments and (2) simulations based on vehicle data.

Experiment-based evaluation. In the real vehicle setting, to generate a case which reflects a timing-voltage-aware adversary performing targeted impersonation, we considered the following scenario in the Honda Accord: adversary's ECU $\mathbb{C}$, which usually sends 0x191, injects attack messages with ID=0x309, thus attacking its original sender $\mathbb{B}$ and at the same time imitating $\mathbb{A}$'s voltage output behavior. In generating such a scenario, evaluation settings were similar to the previous ones, except that `Viden` recorded every 10n-th message of 0x191 as its ID to be 0x309. This was to generate the voltage profile of 0x309 to be similar to $\mathbb{A}$'s as in Fig. 4.16a; $\mathbb{C}$ impersonates $\mathbb{A}$. To introduce ambiguity in the decision, we do not use the intrusion voltage profile in this evaluation. In such a case, if only voltage profiles are exploited, `Viden` might consider ECU $\mathbb{A}$ to be the attacker. However, `Viden` deals with this in Phase 4 by using voltage instances as machine classifier's input. In this evaluation, we used a 200-tree Random Forest classifier with 50% of the acquired data until detecting the intrusion as its training set.

Fig. 4.16b shows the number of *misclassified* voltage instances by the Random Forest classifier. It shows that `Viden` misclassified a large number of 0x309's voltage instances as those of 0x191. That is, even when the voltage profiles of 0x091 and 0x309 were similar, since `Viden` observed the measurements in a momentary manner and the adversary was incapable of precisely matching them, the attack source was correctly identified as $\mathbb{C}$, i.e., transmitter of 0x191, not $\mathbb{A}$. This validates that by using voltage instances as machine classifiers' inputs, `Viden` can prevent targeted impersonation by a timing-voltage-aware adversary.

By the Birthday paradox, at least two ECUs may *naturally* have similar voltage profiles. However, since `Viden` was feasible to distinguish them via machine classifiers, profile collision can be mitigated.

Simulation-based evaluation. In addition to the scenario shown in Fig. 4.16a, which we evaluated via real vehicle experiments, there could be different ways in which a timing-voltage-aware adversary might perform targeted impersonation. For example, the adversary might heat up or cool down his ECU to match some other ECUs' voltage profiles, even before he starts injecting attack messages. Thus, we conducted a more in-depth evaluation as follows. Based on the 35-min data of voltage instances output by the Honda Accord's 6 ECUs and those output by the Chevrolet Trax's 5 ECUs, two attack datasets were constructed to each contain 1000 different "targeted impersonation" attempts by a timing-voltage-aware adversary. We refer to Honda Accord's ECUs as $\mathbb{A}$–$\mathbb{F}$ and Chevrolet Trax's ECUs as $\mathbb{G}$–$\mathbb{K}$. The first dataset was based on only voltage instances of $\mathbb{A}$–$\mathbb{F}$ whereas the second was based on data from both vehicles, assuming that $\mathbb{A}$–$\mathbb{K}$ lie in the same vehicle. Such an assumption was made to evaluate how `Viden` performs when the number of ECUs increases. Each impersonation attempt was constructed by (1) randomly choosing one ECU to be the adversary and another to be the victim, then (2) randomly choosing the times when the adversary starts to change his voltage outputs and (later) when to start attacking the victim, and finally (3) steadily shifting the adversary's voltage instance values (when it starts impersonation) so that his voltage profile matches the victim's, i.e., profile collision, before mounting an attack. Note, however, that such a shift does not make their instantaneous instances to be equivalent. As we discussed

	𝔸	𝔹	ℂ	𝔻	𝔼	𝔽
𝔸	100	0	0	0	0	0
𝔹	0	99.3	0	0	0.7	0
ℂ	0	0	100	0	0	0
𝔻	0	0	0	100	0	0
𝔼	0	0	0	0	100	0
𝔽	0	0	0	0	0	100

(a) ”Honda Accord” attack dataset.

	𝔸	𝔹	ℂ	𝔻	𝔼	𝔽	𝔾	ℍ	𝕀	𝕁	𝕂
𝔸	100	0	0	0	0	0	0	0	0	0	0
𝔹	1.5	98.5	0	0	0	0	0	0	0	0	0
ℂ	0	0	100	0	0	0	0	0	0	0	0
𝔻	0	0	0	100	0	0	0	0	0	0	0
𝔼	0	0	0	0	100	0	0	0	0	0	0
𝔽	0	0	0	0	0	100	0	0	0	0	0
𝔾	0	0	0	0	0	0	100	0	0	0	0
ℍ	0	0	0	0	0	0	0	100	0	0	0
𝕀	0	0	0	0	0	0	0	0	100	0	0
𝕁	0	0	0	0	0	0	0	0	0	100	0
𝕂	0	0	0	0	0	0	0	0	3.2	0	96.8

(b) ”Honda Accord + Chevrolet Trax” attack dataset.

Table 4.1: Confusion matrix of `Viden` [Unit: %].

in Section 4.3.1, although an adversary may match the target’s profile, it would be very difficult for him to precisely follow the target’s instantaneous behaviors (e.g., transient changes due to temperature). This way, we were able to emulate a scenario where the adversary first imitates some specific ECU’s voltage output behavior and then injects attack messages.

Table 4.1a shows the confusion matrix of `Viden` when identifying the attacker of the 1000 targeted impersonation attempts in the first attack dataset. For identification, `Viden` not only used voltage profiles but also a 200-tree Random Forest with voltage instances as its input. Again, half of the data until an attack was detected was used as the training set. Thanks to `Viden`’s analysis of the adversary’s impersonation attempts from two different viewpoints — ECU’s usual voltage output behavior via voltage profiles and its momentary behavior via voltage instances — `Viden` was able to identify the attacker with only a 0.2% false identification rate. Even when `Viden` was evaluated based on our second attack dataset, which had 11 ECUs, `Viden` identified the attacker with a 0.3% false identification rate where the confusion matrix is shown in Table 4.1b. Albeit the increased number of ECUs, `Viden`'s false identification rate increased only by 0.1%, thus corroborating its effectiveness. Note that such false rates reflect `Viden`'s capability and robustness against the most skillful adversary who is aware of timing and voltage, i.e., the timing-voltage-aware adversary. Thus, `Viden`'s false rate against *all* types of the considered adversaries — including the naive and timing-aware adversaries — would be much lower.

One can also interpret such good performance of `Viden` equivalent to its effectiveness in mitigating (naturally occurred) profile collision.

4.5 Discussion

Number of ECUs on CAN. As of 2017, the average vehicle is reported to have approximately 25 ECUs, while luxury cars have approximately 50 [5], but *not all* of them on CAN; some are installed on LIN, MOST, etc. Moreover, to accommodate a large (increasing) number of ECUs on bandwidth-limited CAN, each vehicle is equipped with *multiple* CAN buses [40]. Accordingly, network architectures of various modern vehicles (Audi A8, Honda Accord, Jeep Cherokee, Infiniti Q50, etc.) are shown to have 3$\sim$20 ECUs *per* CAN bus [57]; a similar figure to which we considered in our evaluations. Hence, if `Viden` was installed on each CAN bus in a vehicle, profile collision within that bus is much less likely to occur than the case when all ECUs are (considered to be) installed on one single CAN bus. Even in such a case with profile collisions, `Viden` can still handle it via the execution of its Phase 4.

Multiple ECUs per ID. `Viden` may underperform when multiple ECUs are assigned to send messages with the same ID, albeit unusual/rare. For example, although message

ID=0x040 is scheduled to be sent, in turn, by ECUs $\mathbb{A}$–$\mathbb{D}$, `Viden` would construct only one voltage profile for 0x040. However, if such scheduling information is known in advance (e.g., every 4*n*-th message of 0x040 is sent by $\mathbb{D}$), which is in fact defined by the car-makers, then `Viden` could construct voltage profiles accordingly, thus solving the problem.

Intrusion Detection. Timing-based IDSs exploit the *periodic* nature of CAN messages and thus suffice to detect attacks on periodic messages, but fail to detect attacks on *aperiodic* ones. Since `Viden` determines the transmitter ECU based on voltages, similarly to [35, 64], it can complement those IDSs in detecting intrusions. However, since most in-vehicle messages are periodic [33] and thus most intrusions are detectable, `Viden`'s potential is maximized when it is used for attacker identification.

Attacker from Another In-vehicle Network. If the attack originates from a different in-vehicle network (e.g., FlexRay, MOST, LIN) inside the vehicle other than CAN, the corresponding gateway ECU will be the one that injects attack messages into CAN. `Viden` will, therefore, identify that gateway ECU as the attacker, since `Viden` is designed just for CAN. In such a case, the best both `Viden` and the gateway ECU can do is to look up the message routing table (describing which messages/signals to forward to/from), and identify the "compromised network". Handling such a scenario is important in integrating `Viden` in real vehicles.

Limitations. For `Viden` to identify the attacker ECU, it requires at least one voltage profile to use. For the example shown in Fig. 4.14b, `Viden` referred to the voltage profiles of {0x1EA, 0x1D0} to determine that the attacker ECU was $\mathbb{D}$. Since most ECUs are designated to transmit at least one message ID, one can identify the attacker ECU with `Viden`. However, if the compromised ECU does not send any messages, `Viden`'s attacker identification can be inaccurate. In such a case, the best `Viden` can do would be obtaining the voltage profile of those ECUs during the manufacturing stage and updating them via voltage profile adjustments.

4.6 Conclusion

State-of-the-art vehicle security solutions lack a key feature of identifying the attacker ECU on the in-vehicle network, which is essential for efficient forensic, isolation, security patching, etc. To meet this need, we have proposed `Viden`, which fingerprints ECUs based on voltage measurements. Via the ACK learning phase, `Viden` obtained correct measurements of voltages only from the message transmitters, and exploited them for constructing and updating correct voltage profiles/fingerprints. Using these profiles, we showed via evaluations on a CAN bus prototype and two real vehicles that `Viden` can identify the attacker ECU with a low false identification rate of 0.2%. Considering the fact that vehicles are safety-critical, `Viden` is an important first step toward securing the vehicles and protecting drivers and passengers.

CHAPTER V

Who Killed My Parked Car?

5.1 Introduction

While various ways of attacking/controlling vehicles via security vulnerabilities have been proposed and demonstrated, all of them are shown to be feasible and effective, *only when the vehicle is running*. That is, attacks are commonly believed feasible and hence their defenses necessary only while the ignition is *on*. This has led to a lack of understanding of, and attention to, what an adversary can do while the vehicle's ignition is *off*.

In this chapter, we show such a common belief does not hold by proposing and demonstrating two new attacks—*Battery-Drain* (BD) and *Denial-of-Body-control* (DoB)—through which an adversary can disable parked vehicles with the ignition off. Such attacks are critical and difficult to diagnose because their resulting symptoms are the same as those of typical/usual mechanical/electrical failures (e.g., depleted car battery, malfunctioning key-fob). This will likely delude their forensics, which may, in turn, trigger unnecessary repairs, extend their service outage, or even blame wrong parties (vehicle OEMs/suppliers) for the problem.

By mounting the BD attack, an adversary first *gains* access to the in-vehicle network then *controls* various standby functions of the car, and finally *drains* the battery to a level where the car (doors) cannot be started (opened). We will henceforth use the word "immobilize" as an action that prevents the driver from starting or operating his vehicle. As the ignition is off, one might think that no matter what message(s) the attacker injects, none of the in-vehicle ECUs would receive, and respond to, the injected messages. Surprisingly, however, our analyses of various in-vehicle network standards and their protocol implementations reveal the feasibility of controlling ECU functions via message injections even when the ignition is *off*. Ironically, the main reason for this feasibility is the “wake-up functions”—which are intended to enhance the driver’s experience/convenience—let the adversary wake up ECUs (of a parked vehicle) and then control them. That is, the wake-up functions that were originally designed for a good cause become an *attack vector*. Wake-up functions are standardized, implemented, and thus provided in various in-vehicle networks so that car manufacturers can provide remote standby (not powertrain control) functions, such as remote diagnostics, door control, and anti-theft.

Thus, exploiting such a standardized standby function, the attacker (i) wakes up ECUs by injecting a wake-up message, (ii) controls the awakened ECUs by sending certain messages (e.g., those that control lights, unlock/lock the door, change power mode), and therefore, (iii) achieves his goal of draining the vehicle’s battery. In order to control such functions, the attacker must know which messages (more specifically, with which message IDs) to inject, usually requiring some (painstaking) reverse-engineering with fuzzing [31, 32]. However, for the purpose of BD attack, we propose a driver-context-based scheme, which significantly lowers the technical barrier for the adversary to reverse-engineer the required control messages, i.e., figuring out which message IDs to use, thus helping the attacker succeed in BD attack.

Through the proposed *Denial-of-Body-control* (DoB) attack, in addition to simply waking up ECUs (as was also done in the BD attack), an adversary can force all awakened ECUs to enter the “bus-off” (i.e., shut-down) state. The attacker does this to exploit the fact that, depending on their software configuration, some ECUs do *not* recover from a bus-off; a policy specified in the ISO standard [7]. We find through evaluations of real vehicles that a DoB attack can, in fact, lead to a case where important ECUs, such as a Remote Control Module (RCM)—which is an integral part of remote key and security functionalities—do not recover from a bus-off, i.e., remain shut down. As a consequence, the communication between the key-fob and the RCM is cut off, thus making the driver unable to enter or start his vehicle.

Only a few ECUs of a parked vehicle can be awakened and thus be controlled. This prevents the adversary from controlling certain ECUs. For example, an adversary cannot control the turned-off powertrain ECU (as if the owner could have) when the ignition is off. Interestingly, however, even under such restrictions, the two proposed attacks are still feasible, mainly because they are achieved via controlling ECUs that are asleep, but not completely turned off. According to our in-depth evaluation of which ECUs can be awakened, these two appear to be the most critical attacks that are feasible, given the restricted set of controllable ECUs. It is also important to note that they are very practical and very different from the attacks known to date for the following reasons.

- One common irony of the two proposed attacks is that the wake-up function of in-vehicle networks, which was originally for enhanced user convenience, is exploited as an *attack vector*.
- The ease of mounting the attacks stems from the fact that the wake-up signal was defined to be very *simple*. A simple (agreed-on) wake-up message facilitates extended battery operation time, which is defined as how long the battery can last to provide enough power for the driver to start the car. From a security viewpoint, however, this battery-saving feature makes it easier for the attacker to wake up ECUs and then drain the vehicle's battery.
- The two proposed attacks can be achieved without leaving any traces, except an immobilized vehicle. In fact, their resulting symptoms—drained car battery and inoperable key-fob—are identical to those that may occur in our daily lives due to mechanical/electrical failures (not attacks), thus making their forensic

reconstructions difficult.

- The simplicity of the wake-up signal makes message encryption or use of Message Authentication Codes (MAC) (some state-of-the-art defenses) unable to prevent an attacker from waking up ECUs.

Through extensive experimental evaluations of 11 real vehicles—i.e., 2008–2017 model-year (compact and mid-size) sedans, coupe, crossover, PHEV (Plug-in Hybrid Electric Vehicle), SUVs, truck, and an electric vehicle—we show that all (except one 2008 model-year) of our test vehicles are equipped with the wake-up functions, rendering both BD and DoB attacks feasible. Moreover, we show that by mounting a BD attack, the adversary can speed up the average battery consumption by at least 12.57x, drain the car battery within a few hours, and therefore immobilize the vehicle. We also demonstrate the proposed DoB attack on a real vehicle and show that the attacker can shut down an ECU indefinitely and thus prevent the driver from starting or even entering the car.

In summary, this work makes the following contributions:

1. Analysis and application of in-vehicle network protocols, standards, and implementations on 11 different real vehicles, showing the feasibility of waking up ECUs via message injections;
2. Discovery of two new and practically important attacks—BD and DoB attacks—through which an adversary can immobilize vehicles while the ignition is off; and
3. Demonstration of the new attacks on a real vehicle.

5.2 Background

5.2.1 Terminal Control

The car battery powers in-vehicle ECUs not only when the ignition is on but also when it is off. When and how much battery/power an ECU drains depends on the *terminal* it is connected to. In the DIN 72552 standard [4], terminals are defined as follows. Connected ECUs attached to

- Terminal 15: switched on with ignition on and (totally) off when ignition is off
- Terminal 30: permanently powered on but usually runs in sleep mode, while the vehicle is parked and locked (i.e., ignition is off).

This definition of differentiation in terminal control is to provide different functionalities in vehicles when the ignition is on/off. As an example, consider the Passive Keyless Entry and Start (PKES) system, which allows users to open and start their cars while keeping their car keys in their pockets [42] and is equipped in most contemporary vehicles. In order to provide the keyless entry feature, the ECU running PKES will have to be connected to terminal 30 and be permanently powered on, continuously sensing whether the owner's/driver's key fob is within a certain range. Meanwhile, if the power modes of such permanently powered on ECUs are not properly managed, they will quickly drain the car battery. Thus, to minimize their power consumption, as described in the DIN 72552 standard, they operate in *sleep* mode in which only their transceiver (not their microcontroller) is powered on. This is for the ECU's transceiver to still be capable of detecting and therefore responding to any wake-up signals. For this reason, transceivers have a separate power supply [44]. This way, while the ignition is off, ECUs asleep switch to, and operate in *normal* mode, i.e., wake-up, only when required, thus preventing fast drain of the battery and typically keeping cars continuously parked/idle for 25$\sim$40 days without losing battery power. The battery doesn't get charged by the alternator until the vehicle engine runs back again with the ignition on.

5.2.2 Operation Modes of ECUs

The need of ECUs asleep to be awakened is increasing for enhanced user/driver experience/convenience. For example, vehicle OEMs are introducing new useful functions, such as PKES, overnight remote diagnostics, remote temperature & door

Bus	Data Rate	Industry Standard	Wakeup
CAN	500 kBit/s	ISO 11898-1 ISO 11898-2/5	Global
	125 kBit/s	ISO 11898-1 ISO 11898-3	Global
	33.3 kBit/s	GM LAN	Global
LIN	19.2 kBit/s	LIN	Global
SAE J2602	10.4 kBit/s	SAE J2602	Global
FlexRay	2.5/5/10 MBit/s	FlexRay	Global
MOST	0.1/1 GBit/s	MOST	-

Table 5.1: In-vehicle networks' wake-up capability [53].

control, and anti-theft while the vehicle is parked and turned off. We refer to such functions that are executable/executed while the ignition is off as *standby functions*. Safety-related controls such as powertrain/engine controls are *not* standby features since the ECUs controlling them are usually configured to be completely turned off when parked with the ignition off. To meet such an increasing need, those ECUs asleep are configured to be awakened via local or bus wake-up.

A local wake-up is triggered when a switch attached to the ECU (e.g., a receiver for the remote key) is turned on. This drives a logic state change on its WAKE pin and thus re-activates the whole ECU. Another mechanism in which an ECU wakes up is whenever it sees a specific (i.e., wake-up) message/signal on the bus. Upon detection of a wake-up signal by the ECU's transceiver, which remains ON even while the ignition is off, it turns on the power supply of the ECU, wakes up the whole ECU, and thus enters *normal* operational mode. We will later elaborate on what the wake-up signals are when we discuss how the proposed attacks work. If no additional wake-up signal is received within a certain (preset) time period, the ECU goes back to sleep.

Table 5.1 summarizes in-vehicle network standards/protocols that define such a wake-up function/capability of providing a pathway for driver-friendly standby features. Note that it is standardized, implemented, and used not only in CAN but also in all other in-vehicle networks, except for MOST.

5.2.3 Related Work

Exploiting a remotely compromised ECUs, researchers have shown how various vehicle maneuvers can be (maliciously) controlled by injecting packets into the in-vehicle network [26, 50, 56]. Although such attacks were effective, they were mounted and thus considered malicious only when the vehicle is running. To the best of our knowledge, there has been no study on what the adversary can do while the ignition is *off*. We fill this void by discovering and applying two new attacks, BD and DoB, that are practical, effective and important especially when the ignition is off. They work very differently from existing attacks since the type of ECUs that an adversary can control (when the ignition is off) are far more limited than when the vehicle is running. For example, the adversary may control the door control ECU, but not the powertrain ECUs. For this reason, in the BD attack, the adversary targets those ECUs *controllable* while the ignition is off, such as body control, door, and trunk instead of steering, brake, or engine—those that would have been the main targets in previously known attacks. In fact, for a parked vehicle with the ignition off, control of those ECUs equipped with standby features can be much more "critical" than others. The steering under malicious control is clearly safety-critical when the vehicle is running, but not so when the vehicle is completely turned off; illumination of exterior/interior lights via a door unlock/lock might be a more important problem in such a case!

5.3 Threat Model

As in previously known attacks [31, 32, 50, 58], we consider the adversary capable of remotely (but *not physically*) compromising an in-vehicle ECU via numerous attack surfaces and means, and can thus gain its control; physically compromising an ECU requires physical access and is thus out of our scope. That is, the adversary can 1) compromise a (third-party) OBD-II dongle/device in advance, and gain remote control of the vehicle once the driver plugs it into his car [41] or 2) compromise an in-vehicle ECU (e.g., telematic unit) remotely so as to access the in-vehicle network [31,58]. Since such an adversary would have access to the vehicle's CAN bus, we call such an adversary a *CAN attacker*. Through the compromised ECU, the CAN adversary can inject any message with forged ID, DLC (Data Length Code), and data—which we refer to as *attack messages*—on the bus. Also, since CAN is a broadcast bus, the adversary can sniff messages on CAN.

While previous studies have mainly focused on threats made while the vehicle is running, we consider a different, unexplored threat: the adversary's objective is to *immobilize* the victim's vehicle while the ignition is *off* and make its forensic investigation/reconstruction as difficult as possible. Immobilizing the vehicle prevents the driver from starting or driving the car. In order to make forensics difficult, the adversary also aims to immobilize the car before the driver tries to enter/start it without leaving any trace, except the crippled vehicle. The adversary may even delude forensics by making the symptoms resulting from the attacks resemble typical mechanical/electronic failures, thus misleading the owner/mechanics to wrong causes of the symptom/failure(s). This will likely result in replacing wrong parts (wasting money unnecessarily), extending service outage, or blaming wrong suppliers/OEMs. Although the adversary may immobilize a vehicle by simply flooding the in-vehicle network when the driver tries to start the car, thus preventing any other ECUs from receiving/processing commands, we do not consider such an adversary since its symptom is very different from usual/typical mechanical/electronic failures (thus exposing itself for easy detection and removal).

As the attacks under consideration focus on parked vehicles with the ignition off, the adversary must "play" within the given boundaries of its CAN bus. In parked vehicles, only those ECUs that contribute/relate to *standby* features are asleep, while others are completely turned off. This imposes a certain restriction on the CAN attacker: he can only control ECUs that remain asleep (i.e., not completely off), once the ignition goes off. That is, the attacker's capability is limited to controlling *only* those ECUs equipped with standby

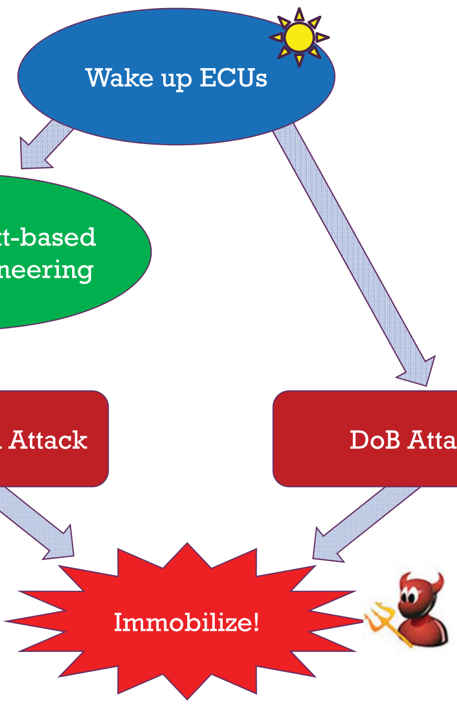


Figure 5.1: An overview of the proposed attacks.

features, e.g., door and lighting ECUs. So, we consider the CAN attacker to be incapable of, for example, driving/crashing a car since the powertrain/engine ECUs are *completely turned off* when its ignition is turned off, due to its irrelevance to standby features.

5.4 Immobilizing a Vehicle

We now introduce two new attacks through which a CAN adversary can compromise the vehicle's availability while it is parked with ignition off.

5.4.1 Overview of the Proposed Attacks

Through a compromised in-vehicle ECU, the CAN adversary has access to the CAN bus irrespective of whether the ignition is on or off. However, especially when the ignition is off and thus all ECUs are asleep or turned off (until the ignition is turned back on again), an attack message injected by the adversary may not be delivered to those ECUs. That is, no matter what messages the adversary may inject, these ECUs may not respond.

Fig. 5.1 presents an overview of how the CAN adversary immobilizes a vehicle in such a case. In order to control/attack ECUs on the bus, s/he first wakes them up and then immobilizes/cripples the vehicle via a BD or DoB attack. In the case of BD attack (Section 5.4.3), the adversary will attempt to exploit the awakened ECUs for controlling certain functionalities of the vehicle (e.g., illuminating exterior/interior lights) and thus draining its battery. In order to figure out which message ID to use for such a control, the attacker goes through a message reverse-engineering process based on *driver context*, which will be detailed in Section 5.4.3.3. That way of reverse-engineering messages allows the attacker to succeed in BD attack much easier than via conventional reverse-engineering such as fuzzing. In case of DoB attack (Section 5.4.4), the attacker need not go through the reverse-engineering process, because the DoB attack does not control ECUs but shuts them down by exploiting their error handling and recovery mechanisms. By mounting a BD and/or a DoB attack, the adversary immobilizes the vehicle.

To immobilize a vehicle, the two proposed attacks rely on standby feature ECUs, which can normally be awakened in any car, and the natural design of the CAN standard. Therefore, they are critical, yet feasible on parked vehicles with ignition off despite the restriction in number/type of ECUs that the adversary can control.

5.4.2 Waking Up ECUs

When the ignition is off, some ECUs are configured to run in sleep mode and continuously monitor if there is any incoming local or bus wake-up signal. Taking this into consideration, using the compromised ECU, the CAN adversary attempts to wake up all other ECUs by delivering a *bus wake-up* signal to them. Then, what type of *bus wake-up signal* should the adversary use in order to wake up the ECUs?

Standardized wake-up. The remote wake-up behavior of a CAN ECU was first introduced and specified in the ISO 11898-5 standard which defines the bus wake-up behavior as *"One or multiple consecutive dominant (0-bit) bus levels for at least* t_{Filter}, *each of them separated by a recessive (1-bit) bus level, trigger a bus wake-up."* While t_{Filter} is defined to be within [500ns, 5µs], its actual value depends on the transceiver being used.

In a CAN bus with bit rates up to 200kBit/s, i.e., bit width longer than 5µs, *any* 0-bit within a CAN frame/message triggers a wake-up. For bit rates up to 500kBit/s, the wake-up condition is always met for any normal CAN data message since its 1) RTR, 2) IDE, and 3) r0/FDF bit are all defined to be dominant (0s) as shown in Fig. 2.1. That is, in a 500kBit/s bus, since those three bits—each with width 2µs—are sent consecutively, the resulting duration of dominant bus level becomes at least 6µs, thus always guaranteeing/satisfying the wake-up trigger condition.

Note that the ISO standard specifies such dominant bus levels to be separated by a recessive bus level. This is easy to achieve since CAN always 1) stuffs a recessive 1-bit after 5 consecutive 0 bits, i.e., bit-stuffing [2, 32], 2) has certain fields fixed with a 1-bit (e.g., CRC delimiter, ACK delimiter as shown in Fig. 2.1), and 3) the user can determine what value(s) to fill in such fields as ID, DLC, and DATA. The same also applies to the extended CAN format which has a 29-bit ID. However, in this chapter, we only consider the basic/standard CAN data format since the extended format is seldom used (due to its bandwidth waste) in contemporary vehicles; most vehicles use the basic/standard format with 11-bit IDs.

The reason for OEMs' agreement on a standardized and simple wake-up signal was to not only guarantee a 100ms link acquisition time [44] but also to allow for a low-power design (e.g., RC-circuit) of wake-up detection, i.e., an energy-efficient sleep mode [44]. That is, the simple design/definition of a wake-up signal was to prolong the battery operation time.

Similarly to CAN, other in-vehicle networks such as FlexRay and LIN define the wake-up signals to be simple for energy-efficiency. FlexRay specifies the signal to be long high/low [44] and LIN specifies it to be a dominant bus level within the interval [250µs, 5ms] [6]. The wake-up signal in those networks also wakes up all ECUs asleep on the bus; see Table 5.1. An adversary can, therefore, easily wake up ECUs while the ignition is off, in not only CAN bus but also other in-vehicle networks. However, in this chapter, we focus on CAN for a more in-depth treatment of the attacks.

CAN adversary waking up the bus. As the wake-up signal is simple and standardized, all the CAN adversary needs to do is inject a fabricated *wake-up message/signal* into the bus, which s/he has access to. As mentioned earlier, due to its simple definition, a wake-up message with *any* content (i.e., ID, DLC, and DATA which are controllable by a remote attacker) will wake up ECUs. Note, however, that only those ECUs which are asleep (i.e., not completely off) will be awakened. This is ironic/paradoxic since wake-up signals are made simple for energy-efficiency, i.e., to minimize battery consumption, but such a simple design helps the attacker wake up ECUs, thus making the vehicle *less* energy-efficient.

Power source. One remaining requirement for a CAN adversary to achieve this is that his (compromised) ECU has to remain powered on. The attacker achieves this fairly easily thanks to two interesting facts.

First, the ECUs which a CAN adversary would (or can) compromise and thus use are most likely to be continuously powered on, or (at least) have a separate power source/supply during their operation. A typical example ECU that an attacker would target (to compromise) is the telematic unit due to its wide variety of external/wireless connectivities. The practicability of the telematic unit being compromised has already been demonstrated in [31, 50, 56]. Interestingly, the telematic unit—which is regarded as one of the most vulnerable ECUs [31–33, 41]—is usually completely (or at least periodically) powered on so as to respond to external events (e.g., requests for remote diagnosis, remote door/temperature control, anti-theft) even after the ignition key has been taken out [1]. Moreover, a telematic unit is usually equipped with an alternative power supply so that it can operate even when the car battery or electrical system is faulty [54]. Similarly, an OBD-II device/dongle, which is also a good target for an adversary to compromise (as demonstrated in [31, 41]), can also be completely powered on (by the attacker) since it is also normally equipped with an external power source (e.g., battery). In summary, ECUs which a CAN adversary will likely compromise via their exposed attack vectors are the ones which are completely/always powered on either by the car battery or their own power supplies.

Second, although the operational mode of a (compromised) ECU is preset to run in sleep mode when the ignition is off, it does not restrict the adversary to change such a setting. Two most common CAN controllers—Microchip MCP2515 and NXP SJA1000—both allow modification of their operation mode (e.g., normal, sleep, listen-only) through software commands [10, 15]. For ECUs with the Microchip MCP2515 CAN controller, the Serial Peripheral Interface (SPI) remains active even when the MCP2515 is in sleep mode, thus allowing access to all registers. Thus, through the SPI, it is also possible for the user/adversary to read/write the CAN controller registers, including the operational mode register [10]. Such user-level features for configuring the CAN controller allow attackers to easily switch from sleep to normal mode via software commands.

As a result, a CAN adversary can inject wake-up messages to the CAN bus while the ignition is off. The transceivers of ECUs asleep observe a wake-up signal on the bus, switch on the ECUs' power supply (usually via an interrupt), and boot up the microcontroller. Hence, the ECUs return to normal operational mode. Since CAN is a broadcast bus, even a single injected message from the CAN adversary wakes up all ECUs asleep.

5.4.3 Battery-Drain (BD) Attack

We now give a detailed account of how an adversary can drain the battery of a vehicle via message injections, i.e., mount a BD attack. Here, we only give the details of how the BD attack can be mounted. Section 5.5 will detail their consequences via in-depth evaluations on real vehicles.

5.4.3.1 Attack 1 — Waking up ECUs

While the ignition is off, the CAN adversary can first attempt to wake up ECUs via message injections. Once the ECUs asleep wake up, they switch to, and run in normal operational mode. Note, however, that an awakened ECU goes back to sleep after a certain period of time (configured by the OEM). Hence, by waking up ECUs as much and as frequently as possible, the adversary can *continuously* force those ECUs to run in normal mode, although they should remain asleep. If ECUs are configured to stay in normal mode (after waking up) for a duration of T_{wakeup}, the frequency of wake-up messages from the attacker has to be at least $\frac{1}{T_{wakeup}}$. Since the adversary can read/sniff messages on the CAN bus, s/he can easily infer T_{wakeup} from the monitored traffic.

With continuous injections of wake-up messages, ECUs (that would usually be asleep) will be forced to stay up in normal mode and thus draw much more current from the battery. In contrast to the ECUs asleep, which normally consume less than 100µA of the battery current, normal-mode ECUs consume several mA. Therefore, by simply waking up ECUs—the simplest BD attack—the CAN adversary can significantly increase the battery current consumption and can thus reduce the battery operation time. We will later in Section 5.5 detail the amount of current drained by simply waking up ECUs, and how that affects the vehicle battery operation time.

5.4.3.2 Attack 2 — Controlling ECUs

An interesting consequence of waking up ECUs is not only the increased battery current consumption but also the pathway it provides for the attacker to *control* ECUs. After waking up, since ECUs previously asleep now run in normal operational mode—the same as when the ignition is *on*—the CAN adversary becomes capable of controlling them. We refer to “controlling an ECU” as executing the ECU’s function(s) via message injections. For example, an attacker may inject an attack message with ID=0x11, which is usually sent by some other ECU. If message 0x11 is processed and used by the brake ECU in engaging/disengaging the brake, an injection of 0x11 will control that ECU's brake function and thus the corresponding vehicle maneuver.

However, the criticality levels of some (malicious) controls would be different from when the vehicle's ignition is on and moving, compared to the case when the vehicle is parked with its ignition off, i.e., different from existing attacks. For example, ensuring that the brakes do not unwillingly engage/disengage in a moving vehicle is safety-critical. It might not be critical when the ignition is off and the vehicle is parked. From the *battery/energy consumption* perspective, the controls that would be considered *malicious* are different.

Controls that increase battery consumption. Since the activation of interior/exterior lights is one of the highest battery-consuming functionalities, the adversary can attempt to control them to increase the battery current drain. Instead of attempting to *directly* control the interior/exterior lights (via light-control messages), we exploit the following vehicle functions which *indirectly* illuminate various lights inside and outside the vehicle.

C1. Changing the vehicle's power mode;

C2. Repeatedly unlocking and locking doors; and

C3. Opening the trunk.

The reason why we exploit such indirect functions is that their control messages are far easier to reverse-engineer (i.e., figure out the meaning/purpose of messages) than the direct (light-)control messages if done based on *driver context*.[1] This fact counters the common belief that reverse-engineering CAN messages is a non-trivial painstaking process [32]. We will later in Section 5.4.3.3 detail how the message reverse-engineering process can be eased, especially for BD attack.

C1. Changing the vehicle's power mode. Depending on the ignition switch position, vehicles run in different power modes such as off, accessory mode, run, and start. An adversary exploits this fact and can first reverse-engineer the control message (via driver

[1]To control vehicle functions, we must know which message ID(s) to use/inject. However, since that information is OEM-proprietary, we must reverse-engineer them.

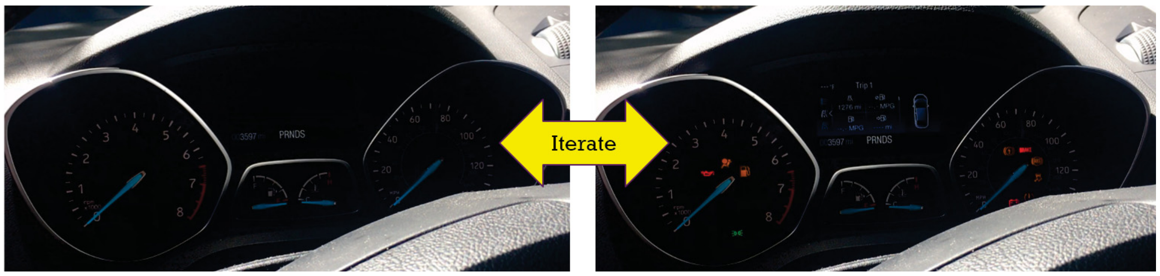


Figure 5.2: Consequence of controlling the power mode of a parked vehicle.

context) which determines/reflects the vehicle's power mode, wakes up ECUs on the bus, and attempts to change the power mode using the reverse-engineered message/ID.

Fig. 5.2 shows why changing the power mode of a vehicle can be considered as an indirect way of illuminating lights and thus increasing the battery consumption, i.e., mount a BD attack. It shows what happened when we tried to alter the power mode of one of our test vehicles via message injections. Here, every procedure and consequence was executed and happened while the ignition was off. We will later in Section 5.5 provide the evaluation settings and general methodologies in accessing the CAN bus and injecting messages. Note that it was fairly easy to reverse-engineer (or figure out) the power-mode control message, especially when using the context—a main reason why we attempted to mount the BD attack via power-mode control instead of (direct) light control. When we controlled the power mode of a parked vehicle, as shown in Fig. 5.2, various indicators on the dashboard were (temporarily) illuminated. Similarly, albeit not shown in Fig. 5.2, the infotainment system was also booted up. When we *periodically* injected the reverse-engineered "power-mode control" message to the bus, the indicators on the dashboard continuously flickered and the infotainment system was intermittently turned on. Although the vehicle ignition was off, a (fabricated) power-mode control message was injected successfully since the vehicles' power-mode master is the Body Control Module (BCM)—the ECU which has to be at least asleep but never completely off due to its important role in providing various standby functions, e.g., Remote Keyless Entry (RKE), Passive Anti-Theft System.

In summary, if a CAN adversary not only wakes up ECUs but also reverse-engineers and injects power-control messages, s/he can continuously illuminate various lights/indicators and further increase the overall battery consumption: an enhancement of BD attack.

C2. Unlocking and locking the door. In addition to the previous two types of BD attack — waking up ECUs and controlling the power mode—a CAN adversary can also attempt to repeatedly unlock and lock the vehicle (while parked). The reason why a CAN adversary would mount a BD attack in this way is not only 1) the unlock/lock messages are easy to reverse-engineer but also 2) it activates various light functions. When the driver (or the adversary) unlocks the car, *welcome lights* of the vehicle illuminate for an enhanced visibility for the driver. Note that the numbers and types of the welcome lights that illuminate may vary with the vehicle manufacturer/year/model, and also depending on whether the lighting control system is set as "automatic," which is the default setting for most drivers [31].

During daytime, lights such as marker lights, interior lights, exterior footlights (on side mirrors), and coming-home lights will/might illuminate when the driver unlocks the vehicle. At night time, the vehicle will turn on its headlights. By reverse-engineering and injecting "door lock control" messages to the bus and then exploiting these *driver-friendly lighting controls*, an adversary can continuously illuminate all the welcome lights and thus significantly increase the average battery consumption; another enhancement of BD attack. Similarly to the attack case C1 where the power mode was controlled, the door control module is another ECU which has to provide a standby function and must thus be not completely off (i.e., must be asleep instead). This allows the attacker to control the door locks even when the ignition is off.

C3. Opening the trunk. The adversary can also attempt to open the trunk of a car similarly to the attack cases C1 and C2. Again, the trunk control module/ECU is another ECU that would have to be asleep, not completely turned off. Therefore, by reverse-engineering the trunk control message, an adversary can (remotely) open the trunk. When the trunk is opened, for enhanced visibility for the driver, (almost all) vehicles are configured to illuminate its interior map, dome, and trunk lights. Again, thanks to such user-friendly lightings, the adversary can illuminate more lights inside/outside the vehicle and thus further increase the battery consumption, i.e., reduce the battery operation time.

In contrast to C1 and C2, the attacker is only required to inject a single trunk-control message into the bus if the lights remain on while the trunk is open (as some vehicles do). Even if the lights automatically go off after some time, the attacker can re-inject the trunk-control message to re-illuminate them. Note, however, that opening the trunk could be a bit (visibly) intrusive, which is a limitation of this approach, although for some vehicles we observed that it is not. However, if the adversary can deplete the battery before the driver/passenger notices and thus attempts to close it, such an intrusive approach will still succeed in immobilizing the vehicle. When mounted overnight, since the driver/passenger would notice this only when s/he attempts to start the car in the following morning, the attacker could be given approximately half a day or even a few days (e.g., weekends) in succeeding it!

5.4.3.3 Driver-Context-Based Reverse-Engineering

In controlling an ECU (as in C1–C3), since the message IDs that have to be used are different for different vehicle manufacturers and models, adversaries would have to reverse-engineer messages for each vehicle that it targets. This fact generally becomes a (high) technical barrier for the adversary, especially when mounting state-of-the-art attacks on different vehicles. However, for the purpose of BD attack, the message reverse-engineering can be achieved very differently, i.e., not via fuzzing. Specifically, by reverse-engineering messages based on the *driver-context*, it becomes much easier for the adversary to figure out which messages to use in succeeding the BD attack on different/various vehicles.

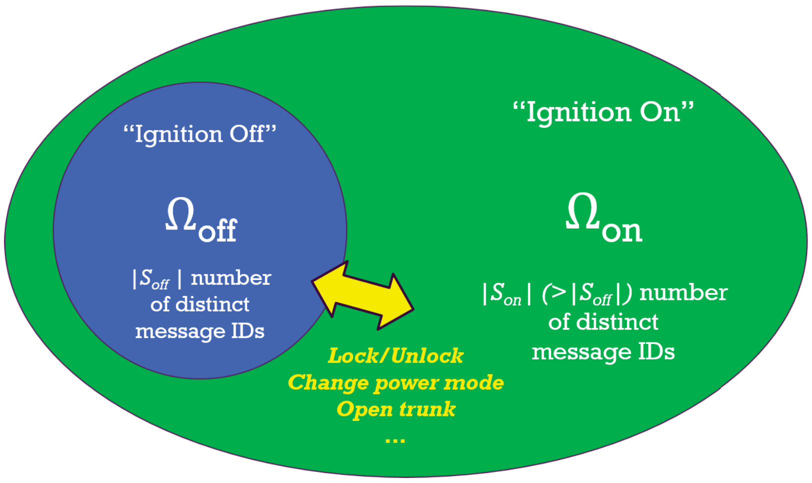

"Ignition Off"
Ω_{off}
$|S_{off}|$ number of distinct message IDs
"Ignition On"
Ω_{on}
$|S_{on}|$ ($>|S_{off}|$) number of distinct message IDs
Lock/Unlock
Change power mode
Open trunk
...

Figure 5.3: Driver-context-based reverse-engineering.

The proposed driver-context-based reverse-engineering works as follows. First, when the ignition is off, the CAN adversary continuously wakes up ECUs and records/logs the CAN traffic as Ω_{off} as illustrated in Fig 5.3. If the adversary knows when the driver usually starts the car (e.g., 9am in the morning), s/he can start such a process just before that time. We define the sets of distinct IDs sent while the ignition is on and off as S_{on} and S_{off}, respectively. Then, once ECUs are awakened via a wake-up message from the adversary, $|S_{off}|$ distinct message IDs would be observed on the bus and that number would be lower than $|S_{on}|$, since the number of ECUs running in normal operation mode (and thus periodically sending messages) would be less.

Next, the adversary continuously logs the CAN traffic as Ω_{off} and marks the *bit positions* of messages' ($\in S_{off}$) data fields that continuously change as Δ_{off}. One example of Δ_{off} can be the last byte of the data field where OEMs usually put their 1-byte checksum [32, 58]. When the CAN adversary finds that the ignition is (now) turned on, onwards, s/he records/logs the traffic (of an "ignition on" vehicle) as Ω_{on}. The CAN adversary can acknowledge this by observing a suddenly-increased number of distinct message IDs—from $|S_{off}|$ to $|S_{on}|$—on the bus. Since $S_{off} \subset S_{on}$, the CAN adversary analyzes how the data values/fields, excluding Δ_{off}, of those message IDs ($\in S_{off}$) changed during the period of just before to right after the ignition being started.

An interesting yet important fact about S_{off}'s data values (excluding those in Δ_{off}) is that they reflect the driver's *actions* before driving the vehicle. Imagine a person who tries to start and drive his/her car. S/he would first unlock the vehicle and then change its power mode, i.e., turn on the ignition. Perhaps, s/he might even open the trunk to put items there before starting the vehicle. Such a routine of accessing and starting the vehicle, which we define as the *driver context*, is embedded/reflected in Ω_{off}, i.e., the CAN traffic obtained while the ignition was off. Specifically, when the driver makes some action (e.g., unlock the door) while the ignition is off, the data value of an ID ($\in S_{off}$) would change. Note that such a change would not incur in any of the data fields in Δ_{off}, i.e., the bit positions which their values normally continuously change, since the driver's action and thus the corresponding change in the data values are "temporary". So, by observing the temporary data changes in S_{off} incurred right before the ignition is turned on, the CAN adversary can easily figure out which messages relate to those driver-context-related controls. Interestingly (and luckily for the CAN adversary), as discussed in Section 5.4.3, such driver-context-related controls lead to illuminating various indicators/lights. As a result, especially for a CAN adversary attempting to mount a BD attack, reverse-engineering the required control messages (e.g., door lock/unlock, changing power modes) becomes far more easier! The same can be applied when the driver stops and leaves the vehicle, since s/he would again change the power mode, perhaps open the trunk, and of course, lock the vehicle.

Given below is an example of how we reverse-engineered one of our test vehicle's door-control messages. When describing the procedures, we do not use the actual ID value nor the bit positions, since they are proprietary to the OEMs. When comparing two sets of Ω_{off}, one before pressing a remote key fob and another during it, we found that the data fields of message ID=0x01 had changed from [00 $\underline{10}$ 00 00 FF 00 AB CD] temporarily to [00 $\underline{30}$ 00 00 FF 00 BC EF]. We verified in advance that the last 2 bytes of message 0x01 continuously change, even without any actions taken on the vehicle. That is, we verified that the last two data bytes of 0x01 belong to Δ_{off}. As a result, we were able to easily figure out that the second byte of message 0x01 controls our test vehicle's lock and unlock functions. We will later show through evaluations that such an approach was indeed successful and thus let us easily unlock & lock the car, illuminate the welcome lights, and therefore, mount the BD attack.

5.4.4 Denial-of-Body-control (DoB) Attack

In addition to the BD attack, the CAN adversary can mount a Denial-of-Body-control (DoB) attack in order to immobilize a vehicle.

Error handling is built in the CAN protocol and is important for its fault-tolerance. If an ECU experiences or incurs continuous errors while transmitting or receiving a message, the CAN protocol specifies that its Transmit Error Counter (TEC) or Receive Error Counters (REC) should be increased, respectively [2]. If its TEC exceeds a pre-defined threshold of 255 due to continuous errors, the ECU is forced to enter the bus-off state and shut down. Exploiting such a standardized CAN feature, in Chapter II, we proposed the *bus-off attack*, which enforces other healthy/uncompromised ECUs to shut down. The proposed DoB attack is mounted in a similar way to the bus-off attack, except that it further exploits the following fact specified in the ISO 11898-1 standard [7] and thus immobilizes the vehicle. The standard specifies that *"a node can start the recovery from the bus-off state only upon a user's request,"* where the user's request depends on the ECU's software configuration. The proposed DoB attack thus exploits such a definition of bus-off recovery as follows.

While the ignition is off, the CAN adversary wakes up ECUs so as to make them responsive to his injected messages. Then, the adversary switches its bit rate (e.g., from 500 kBits/s to 250 kBits/s). According to the CAN error-handling scheme, this makes *all* awakened ECUs on the bus continuously experience and incur errors, and therefore enter the bus-off state, i.e., shut-down. This way, the adversary not only mounts the bus-off attack on a targeted ECU (as demonstrated in Chapter II) but also on all ECUs on the bus. Instead of changing the bit rates, changing internal/net resistances or capacitances can be an alternative method in achieving this. As a result, per bus-off recovery specification, depending on the ECUs' software configurations, some ECUs would recover from the bus-off state, whereas some will *not*.

Depending on the car manufacturer and year/model, ECUs such as BCM or RCM, which is the security ECU that authenticates each message to and from the remote key fob, can in fact be configured/defined not to recover from the bus-off state, mainly for safety, since the bus-off is a serious problem, or for anti-theft purposes. Hence, if the CAN adversary were to mount the DoB attack on such a vehicle, then s/he can indefinitely shut down the BCM or RCM, and thus cut off the communication between the (driver's) remote key fob and the vehicle. Contemporary/newer vehicles are mostly equipped with the PKES system, which allows users to open and start their cars while having their key fobs in their pockets [42] and is installed either in BCM or RCM. For the vehicle to be opened/started, PKES must verify that the legitimate key fob is in the vehicle's vicinity. Therefore, shutting down BCM/RCM (and thus PKES) would mean that the vehicle will *not* be able to receive and authenticate any remote key signals (sent by the key fob), thus preventing opening or starting the vehicle, i.e., the CAN adversary immobilizes the vehicle via a DoB attack.

Once the attacker succeeds in mounting DoB attack, there is no need for the attacker to mount the attack, again; some ECUs that have entered bus-off will never boot up again, anyway. This allows the attacker to not only succeed in mounting the attack in a very short period of time but also do not leave any trace, thus making its forensic difficult. We will later show through evaluations that such a configuration of BCM/RCM not recovering from bus-off actually exists in real vehicles and thus makes the driver unable to open the door/trunk and start the vehicle even with his/her legitimate, perfectly-functioning key fob.

5.5 Evaluation

We now evaluate the feasibility and criticality of the two proposed attacks on various real vehicles.

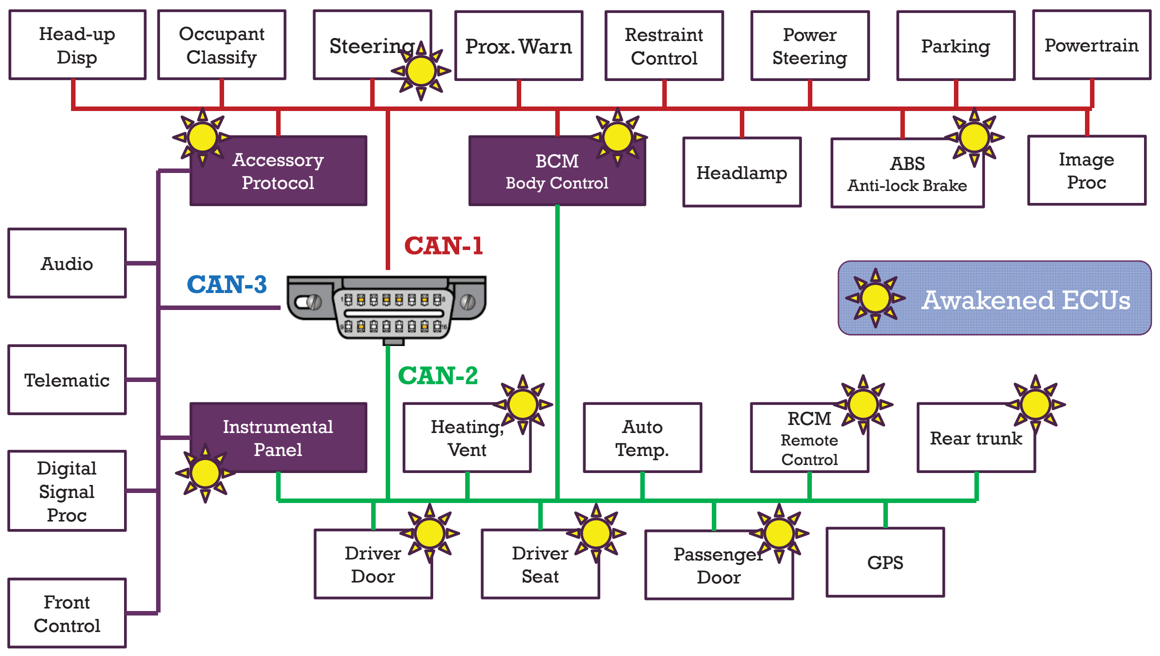


Figure 5.4: Vehicle architecture of one of our test vehicles.

5.5.1 Waking Up ECUs

To verify whether ECUs in real vehicles can indeed be awakened via simple wake-up messages while the ignition is off, we connected a Vector CAN device to our test vehicles' OBD-II port. We then injected wake-up messages/signals to the CAN bus. The wake-up message we used had its ID, DLC, and DATA fields—that a user/adversary can control at the application layer—all filled with 1s. This was to verify that messages with the minimum number of 0s can also function as a valid wake-up message.

In-depth analysis on a test vehicle. Fig. 5.4 shows the in-vehicle network architecture of one of our test vehicles—a 2017 year model—and the ECUs that responded to the wake-up messages. Note that this network architecture of the test vehicle is not unique for the OEM of our test vehicle but is valid for the vehicles built by other OEMs except for only slight variations in the network architecture [57]. We verified which ECUs were awakened by logging the CAN traffic that contains the message IDs observed on the bus, and by mapping those IDs to the corresponding transmitter ECUs using the test vehicle OEM's CAN Data Base Container (DBC). The CAN DBC describes the properties of the

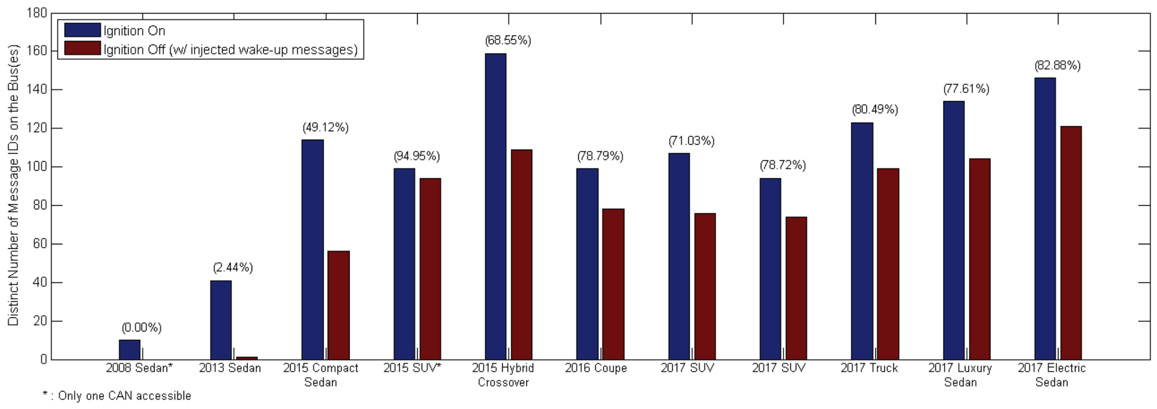


Figure 5.5: Verifying wake-up messages in 11 different vehicles.

CAN network, the ECUs connected to the bus, and the CAN messages and signals. For the purpose of this research, the CAN DBC file was provided by the test vehicle's OEM.[2]

When the wake-up message/signal was injected, one can see from Fig. 5.4 that not all ECUs were awakened. This would be most probably due to different ECUs being attached to different terminals (with different terminal control policies) in order to minimize battery/power consumption and, at the same time, provide various standby functions. In CAN-1 which connects ECUs performing safety-critical functions, only 4 of 13 of them were awakened. On the other hand, in the CAN-2 bus where ECUs responsible for vehicle body control were connected, almost all ECUs but two were awakened. Considering the fact that contemporary/newer vehicles provide various standby "body control" functions such as keyless entry, hands free (foot) trunk opening, and anti-theft, more ECUs being awakened in CAN-2 than CAN-1 would be the norm.

Verifying wake-up functionality in various vehicles. Using the OBD-II device, we also verified how different vehicles (OEMs/years/models) react to the injection of a wake-up message. To confirm that the wake-up functionality exists in different cars, we chose various types of test vehicles: (compact and mid-size) sedans, coupe, crossover, PHEV, SUVs, truck, and an electric vehicle with model-years 2008–2017.

Fig. 5.5 shows the number of distinct messages observed on the CAN bus 1) when the

[2]Due to its proprietary information, such DBC files are not shared without permission from the OEMs and their suppliers.

ignition was on, and 2) when we woke up ECUs on the bus by injecting a wake-up message while the ignition was off. Since the feasibility of wake-up stems from how the in-vehicle network standard is specified and implemented, instead of OEMs' design decisions, we have chosen not to identify/reveal the particular make and model used in our evaluation. Note, however, that the 11 test vehicles (shown in Fig. 5.5) we examined are from different OEMs and also represent different models. For some vehicles, since their OBD-II pinout was configured to not provide full access to all of their buses, we only show those that were awakened in the accessible bus(es).

When waking up ECUs in some old cars (most with low-level trims), far less distinct message IDs and lower percentages of them (compared to the case with the ignition on) were observed on the bus than other newer cars. Since the number of ECUs is proportional to that of distinct message IDs—although it is not linearly proportional—we can infer that there were less awakened ECUs transmitting them in older cars; not all messages can be sent by a single ECU due to the overhead. This would most probably be due to the fact that the older cars do not require/provide any (or not many) standby functions (e.g., PKES). For example, no standby functions (e.g., keyless entry, hands-free trunk opening) were provided in the 2008 and 2013 model-year test cars; this is the reason why none and only one ECU was awakened, when a wake-up message was injected on the bus.

On the other hand, when a wake-up message was injected on the buses of nine 2015–2017 model-year test vehicles, we observed that 49.12–94.95% (75.44% on average) of the distinct message IDs sent while the ignition was on, were *also* sent when ECUs were awakened while the ignition was off. Such a high number/portion of ECUs being awakened by a wake-up message and thus sending more message IDs on the bus is because they had numerous standby functions installed for enhanced driver's experience and convenience (e.g., PKES/RKE, hands-free trunk opening, anti-theft)—a trend that is expected to grow. These results corroborate the fact that vehicle ECUs are indeed equipped with the wake-up functionality (adhering to the standard) and can thus be exploited by the

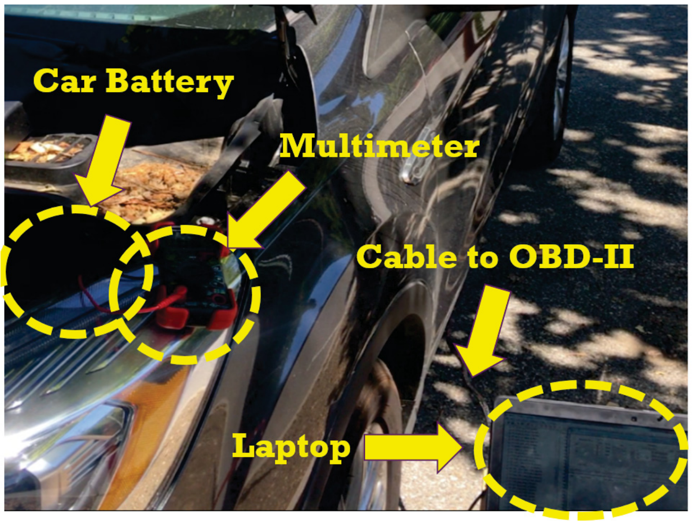


Figure 5.6: Setup for measuring battery consumption.

CAN adversary as an attack vector.

5.5.2 BD Attack

After verifying that the ECUs in our test vehicles can be awakened via basically any wake-up message (even with all 1s in ID, DLC, and DATA fields), we mounted 4 different types of BD attack on one of our test vehicles: 1) simple wake-up, 2) power mode control, 3) repetitive door unlock & lock, and 4) opening the trunk. We were able to reverse-engineer the control messages for those functionalities via the proposed driver-context-based reverse-engineering.

In order to measure the drained/discharged current from the car battery, we disconnected the negative cable from the negative battery terminal and connected our multimeter in series to the battery, i.e., one probe to the negative cable and the other to the battery terminal. We conducted the battery current draw test from the negative side to prevent accidental shorting and while the vehicle was parked with its ignition off. Then, as shown in Fig. 5.6, we injected (iterative) sequence of control messages to the vehicle through the OBD-II port.

Current drain. Table 5.2 summarizes the average amount of current measured to be

Attacks	Discharged Current [mA]	Amplification Factor	Max. Battery Operation Time [Days]
None	12.2	Baseline	30.7
+ Wake-up	42.0	x3.44	8.92
+ Change Power Mode	74.5	x6.11	5.02
+ Lock & Unlock Door	101.1	x8.29	3.70
+ Open Trunk	153.3	x12.57	2.44

Table 5.2: Maximum battery operation time under different BD attacks.

drawn from the vehicle battery when each attack was mounted additionally. When the ignition was off and no ECU was awakened, 12.2mA was consumed. As expected, this value was below the parasitic drain threshold, which is about 30mA. However, when we just woke up the ECUs, the discharged current exceeded that threshold and drained 42mA. Exceeding the parasitic drain threshold even slightly is considered to be a serious/critical problem. As we controlled more functions that indirectly illuminated exterior/interior lights, the average battery consumption increased further to 74.5, 101.1, and 153.3mA.

More worse cases. It is important to note that, depending on the vehicle model, there would be (much) more of such controllable functions, thus allowing the attacker to drain the battery more quickly and easily. However, we only show 4 controls as examples for the proposed BD attack since they are already very critical. Moreover, if the attacks are launched at night time when the exterior brightness is low, the car headlights will always come on (as one type of a welcome light) when the car is unlocked. As a result, the current drain will increase sharply, i.e., drain the battery very quickly. However, in order to show the *minimum drain/discharge*, i.e., the *worst* possible case for the adversary, we conducted all our experiments outdoor during day time.

Expected battery operation time. In order to better understand how the increased battery current consumption relates to how quickly the attacker can immobilize the car, we determine the *battery operation time* as follows. Consider a 45Ah battery, which is the standard car battery capacity, with a State-of-Charge (SoC) of 70%—the average battery SoC of a passenger vehicle—when parked. Then, since the minimum SoC for a cold start is considered to be 50% in the *worst* case [44], i.e., worst-case for the adversary, our test vehicle's battery can theoretically remain idle/parked for up to $\frac{45Ah\times(0.7-0.5)}{12.2mA} = 737.7$ hours ≈ 30.7 days; something that is normally expected. Note that this is the theoretical maximum battery operation time since we considered the worst possible case for its derivation.

One can see from Table 5.2 the reduction of the maximum battery operation time while mounting different types of BD attacks. Theoretically, with the 4 control functions, it only takes a maximum of 2.44 days for an adversary to immobilize the vehicle via BD attack, i.e., can be achieved over the weekends. It is important to note, however, that it could take much shorter especially when the temperature is low and the battery is aged. More interestingly, in reality, the *actual* battery operation time is known to be much *shorter* than the theoretical/ideal value, i.e., much shorter than the times shown in Table 5.2. According to the Peukert's Law, because of intrinsic losses and the Coulombic efficiency being always less than 100%, the actual battery operation time is much lower than the theoretic value in which the latter assumes the battery to be ideal [38]. Since the intrinsic losses in the battery escalate as load increases, the battery capacity is known to drop sharply as the drained/discharged current increases. This implies that as the adversary controls more functions, he can not only increase the battery consumption but also decrease the available battery capacity at the same time! So, he can significantly reduce the battery operation time via a BD attack, thus crippling the vehicle quickly, perhaps overnight.

5.5.3 DoB Attack

Through experiments on one of our test vehicles, we also verified the consequences of the proposed DoB attack. Using the connected OBD-II device, we mounted a DoB attack as described in Section 5.4.4.

After launching the DoB attack on one of our parked test vehicles, taking only a few seconds, we confirmed from the CAN traffic that all ECUs on the bus were continuously incurring and/or experiencing errors, causing all the ECUs to enter the bus-off state. After

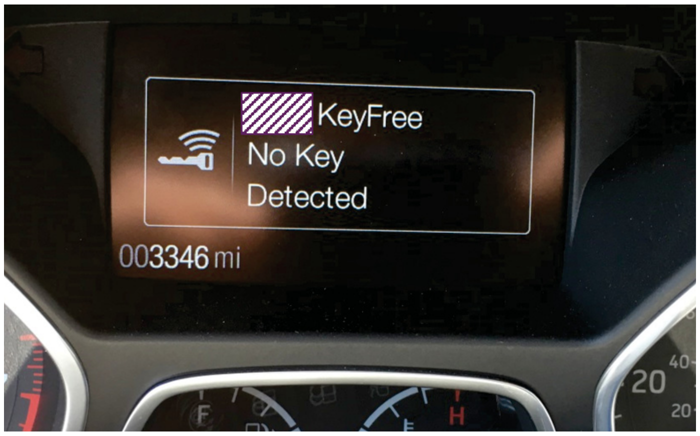


Figure 5.7: Consequence of a DoB attack.

mounting the DoB attack, we observed most, but *not all* of the ECUs recovered from the bus-off state as configured. We observed that the number of distinct message IDs sent on the bus was actually reduced by 6 after the DoB attack. By mapping those missing IDs to the actual transmitter ECU using the DBC file, we found that the RCM (Remote Control Module) did *not* recover from the bus-off, i.e., remained shut down, most probably due to its distinct recovery policy configuration (perhaps for anti-theft/engine-immobilizer purposes). Since the RCM was indefinitely off, the key fob was not authenticated and thus could not establish a connection to the vehicle. As shown in Fig. 5.7, the vehicle could not detect that the key was in its vicinity; the key was in fact placed right in front of the dashboard. This consequence of the proposed DoB attack was reproducible on our test vehicle.

Of course, remote key fobs are now equipped with RFID chips that can be used for authentication, connection establishment, and thus starting the vehicle in case of a dead key fob battery. However, since the communication between the key fob's RFID and the vehicle was also configured to be governed by the RCM, the RFID-based (emergency) start did not work either. More interestingly, after mounting the DoB attack, when we tried to open the doors or trunk to enter the car, we could not, because the RCM was not functioning and thus failed to authenticate the key fob. The only way to get in was actually using the back-up physical key hidden in the key fob. Note, however, that the car did not start anyway (as shown in Fig. 5.7) even though we were able to enter it!

As discussed earlier, this consequence comes from the fact that OEMs (or their ECUs) may have different bus-off recovery configurations. In our test vehicle, the setting of an RCM to not recover from the bus-off "favors" the attacker in mounting a critical DoB attack. We found the only way to restore the vehicle back to its original state after a DoB attack was to disconnect the battery, wait for a few minutes, and re-connect the battery. Such a process resets the *states* stored in each ECU and thus lets them run in their original/intended states. However, imagining a victim confronting the symptoms of DoB attack, i.e., the key fob neither working nor being detected, he might first try to change the key fob battery. Obviously, since that won't work, he would consider the car battery completely dead and therefore, would probably have the car towed to the service station for a battery replacement, wasting money and time unnecessarily!

5.6 Discussion

Countermeasures. As the proposed BD attack and DoB attack are mounted with the ignition off, the design principles of their countermeasures have to be very different from the state-of-the-art defenses, which are mostly concerned with attacks on "running" vehicles. For example, as of the current CAN standard, since the wake-up itself can be achieved with any CAN message having a 010 bit-sequence, adding MAC or message encryption cannot prevent the adversary from waking up ECUs; a message with MAC/encryption will still have such a sequence.

The cornerstone of BD attack and DoB attack is to wake up ECUs asleep on the bus while the ignition is off. So, as their feasible and effective countermeasure, one can think of continuously running an Intrusion Detection System (IDS) even when the ignition is off in order to capture any abnormal wake-up messages; wake-up messages usually should not be seen very frequently. However, since the operation of an IDS would increase the current drawn from the battery, such an approach may defeat the very purpose of reducing battery consumption. Like other ECUs asleep, the IDS ECU can also be configured to sleep most of time and wake up only when it sees a wake-up message. As a countermeasure against both types of attack, the wake-up pattern of an IDS can then be modeled and used to detect any abnormal wake-up requests on the bus without continuously running it. Similarly, the IDS can be configured to wake up periodically, check the battery SoC—if there was any significant drain recently—and react accordingly. Moreover, especially for the DoB attack, how to recover from the bus-off state has to be re-examined in order to prevent the consequences of the DoB attack, as we had demonstrated.

Enhanced wake-up functionality. Partial deactivation of subnets within a given network—or partial networking—has been discussed and planned to be installed by car manufacturers. This is to reduce energy consumption and thus CO_2 emissions [53]. In such a setting, only the pre-defined wake-up messages that pass the wake-up masks/filters of selective ECUs can wake up those ECUs during operation. However, since that message is "pre-defined" and can easily be learned by observing the CAN traffic and its sudden change in the number of message IDs (as in Section 5.4.3.3), the wake-up message itself can still be learned and used by an adversary. In fact, the introduction of partial networking will increase the number of ECUs to sleep rather than being completely turned off, and thus provide a larger attack space for mounting the proposed attacks.

Limitations. We assumed that the CAN adversary has access to the in-vehicle network via a compromised ECU to mount the proposed attacks. However, we must not overlook the fact that the adversary might not even need a compromised ECU to mount the proposed attacks. To further enhance the driver's convenience, companies such as Volvo [21], Lexus [22], and Tesla [23] let car owners unlock/lock their cars by using their smartphone apps with an eventual goal to totally replace key fobs with smartphone software. This means the existence of another (new) attack vector for mounting BD attack: compromising those apps! As more of such vehicle-related technologies evolve and get deployed, there may be more (intelligent) ways of mounting the proposed attacks.

Although we succeeded in mounting BD attack and DoB attack on our 2017 model-year test vehicle, not many ECUs were awakened when a wake-up message was injected in *older* vehicles, because they had less standby functions than *newer* models, and thus had less ECUs asleep while the ignition was off. The proposed attacks will likely be easier and more effective to be mounted on newer models as we observed in waking up more ECUs in newer vehicles (Fig. 5.5). For the DoB attack, however, since its success/feasibility will totally depend on how the OEMs configured their "bus-off recovery" for different ECUs, it might not be as feasible as BD attack. The BD attack will still be feasible unless the standard wake-up procedure is changed or standby functions are not installed.

5.7 Conclusion

In this chapter, we have discovered two new important vulnerabilities in vehicle availability: *Battery-Drain* (BD) and *Denial-of-Body-control* (DoB) attacks. They are counter-intuitive in that attacks are commonly believed to be possible and effective only while the ignition is on. Specifically, an attacker is shown to be able to wake up ECUs on the bus, even while the ignition is off, mount the proposed attacks, and then immobilize a parked vehicle with its ignition off. Through extensive experiments on real vehicles, such attacks are shown to be easy to mount and very critical to vehicle availability. Ironically, the adversary exploits, as attack vectors, the in-vehicle network features originally designed for either energy-efficiency (e.g., simple wake-up signals) or enhanced user/driver experience/convenience (e.g., standby functions). There could still remain different types of unknown and unintuitive vehicle vulnerabilities. It is therefore important to analyze and understand what consequences existing built-in/standardized functionalities can lead to. This calls for concerted efforts from both academia and industry on this possibility and countermeasures thereof in order to build secure vehicles.

CHAPTER VI

Conclusions and Future Work

The paradigm shift of vehicles being equipped with more and more electronics and external connectivities has led to new security breaches in vehicles, thus making automotive security one of the most critical issues. In this dissertation, through analyses of in-vehicle networks and state-of-the-art countermeasures, we have shown that there still remain several undiscovered attacks/vulnerabilities on in-vehicle networks and a large gap between what state-of-the-art defenses should achieve and can achieve. As a result, we discovered three new attacks/vulnerabilities and proposed two new defense schemes that outperform existing ones. We summarize the contributions of this dissertation and present some future research directions.

6.1 Dissertation Contributions

This dissertation looks at the vehicle security problem from two different perspectives: as an attacker and as a defender.

New Unveiled Attacks

In Chapter II, we unveiled a new vulnerability, called the *bus-off attack*, of in-vehicle networks. Since the proposed bus-off attack exploits in-vehicle networks' error handling scheme in disconnecting an uncompromised ECU and/or even shutting down the entire in-vehicle network, its consequence/symptom was shown to be very similar to an erroneous bus, i.e., difficult to distinguish the attack from a functional error. Moreover, since it has the capability of nullifying state-of-the-art solutions and is easy to launch, the bus-off attack must be thwarted with high priority, thus making it even more important to design and deploy its countermeasures.

In Chapter V, we discovered two new practical and important cyber attacks — called *Battery-Drain* (BD) and *Denial-of-Body-control* (DoB)—on real vehicles, invalidating the conventional belief that vehicle cyber attacks are feasible and thus their defenses are required only when the vehicle's ignition is on. The former can drain the vehicle battery while the latter can prevent the vehicle owner/driver from starting or even opening/entering his car. These attacks were shown to delude forensics since their symptoms were very similar to typical mechanical/electronic failures, thus triggering unnecessary repairs, extending service outage, or even blaming wrong parties (vehicle OEMs/suppliers) for the problem.

One common ironic fact of the three proposed attacks is that in-vehicle network features, which were initially designed and built for either driver's safety (e.g., error handling) or enhanced experience/convenience (e.g., wake-up function), were exploited as attack vectors. Hence, similarly to the three proposed attacks, there could still remain different types of attacks/vulnerabilities that are critical but undiscovered and unintuitive. It is therefore important to further analyze and properly understand — *from a security viewpoint* — what consequences existing built-in/standardized functionalities can yield.

New Defense Schemes

Based on our analyses of various types of vehicle cyber attacks, we were able to find that state-of-the-art defense schemes lack certain features in meeting the need of a *secure vehicle*. To address such issues, in Chapter III, we first proposed a new anomaly-based IDS called `CIDS`, which can cope with various attacks including those that existing solutions cannot, e.g., masquerade attack. Based on our experiments on a CAN bus prototype and on real vehicles, `CIDS` was shown to be capable of detecting various types of in-vehicle network intrusions.

Another key feature that state-of-the-art vehicle security solutions lacked was identifying the attacker ECU on the in-vehicle network. Without an accurate attacker identification, it is impossible to achieve an efficient forensic, isolation, and security patching. To meet this need, in Chapter IV, we proposed `Viden`, which fingerprints ECUs based on voltage measurements. Via the ACK learning phase, `Viden` obtained correct measurements of voltages only from the message transmitters, and exploited them for constructing and updating correct voltage profiles/fingerprints. Using these profiles, we showed via evaluations on a CAN bus prototype and two real vehicles that `Viden` can identify the attacker ECU with a low false identification rate of 0.2%.

Considering the fact that vehicles are safety-critical, `CIDS` (for detection) and `Viden` (for identification) are important steps toward securing the vehicles and protecting drivers and passengers. The fact that these schemes can be simply deployed in real vehicles as software — i.e., does not require any changes in the protocol or hardware — makes them efficient, low-cost, and practical.

6.2 Future Research Directions

Here, we discuss future research directions that can be built on top of this dissertation.

6.2.1 Secure In-vehicle Gateway

Researchers have proposed several message authentication methods in order to secure in-vehicle network communications [81, 83, 86]. Although they provide message authentication between ECUs connected to the same network, i.e., homogeneous network, they all lack consideration of the current in-vehicle network architecture in which more than one type of networks are usually used, i.e., heterogeneous network. Accordingly, for messages that have to be transmitted and exchanged between two different networks, such as between CAN and Ethernet or a high-speed CAN and a low-speed CAN, previously proposed schemes cannot provide proper authentication. Such exchange of messages between heterogeneous networks is expected to be more prevalent, considering the trend that the new emerging x-by-wire functionalities are realized using those messages.

Messages exchanged between at least two different networks, either having the same protocol but different speeds or having different protocols, are governed by an Internal Gateway (I-GW), which is connected to those multiple heterogeneous networks. To provide proper message exchange between those networks, I-GW possess and performs the following functionalities:

- Routing: transfers messages between different networks;
- Filtering: filters out messages so that only the pre-determined ones are delivered;
- ID modification: modifies the ID value of the message to conform to the ID assignment on the target network;
- Repacking: packs several short messages into one long message;
- Splitting: splits the payload data into segments so that only the required ones are delivered to the target network;
- Translation: converts the message from the source to target network's protocol format; and
- Scheduling: schedules message delivery based on its given priority on the target network.

In contrast to the basic network functionalities of ECUs, since I-GW has to perform much more, the approach of message authentication between heterogeneous networks should be approached differently from the case of homogeneous networks, i.e., the existing authentication schemes won't work. Moreover, since there is also a possibility of the I-GW itself being compromised, messages passing through that I-GW cannot be authenticated, indicating the limitations of existing authentication schemes. Thus, an enhanced level of authentication has to be applied at the I-GW. Therefore, it is important to investigate how to design a new authentication scheme that provides message authentication in not only homogeneous but also heterogeneous in-vehicle networks.

6.2.2 Security Analysis of Other CAN-based Systems

There are hundreds of millions of CAN nodes currently in use around the world. As mentioned before, CAN is widely used in various *cyber-physical* systems such as passenger cars, buses, drones, factory automation, work machines, and submarines. Such uses of CAN are expected to further grow in the future due to its fault-tolerance, efficiency, and low-cost. CAN provides high immunity to electrical interference and ability to self-diagnose and repair data errors.

Despite such an expected growth of CAN usage, we are yet to fully understand the security implications of CAN in those cyber-physical systems. One might simply think that defense schemes for vehicle CAN — including those that have been discussed in this dissertation — can be used in other cyber-physical systems such as drones and submarines; assuming those solutions would be as effective as they were in vehicles. However, we must be cautious about such an approach due to the fact that those systems would most likely have different network architectures, remote endpoints, and system requirements due to differences in provided functionalities and deployment cost. Such different configurations and conditions may provide more room in implementing a higher cost but more effective solution, or they might impose more restrictions on the design and deployment of defense schemes, than those that we had for analysis of CAN security in vehicles. Therefore, even though the same protocol might be used, it is imperative to first assess the security of those systems individually based on their *unique* functionalities, roles, and system configurations, and then build defense schemes based on such an assessment.

6.2.3 Post Detection and Identification

Through `CIDS` (Chapter III) and `Viden` (Chapter IV), we can not only detect attacks but also pinpoint the attacker ECU. However, there still remain unanswered research questions after detection: 1) what reactions/counterplans should we make? 2) can/should we simply alert the driver (e.g., check engine light)? 3) can/should we simply enforce the vehicle to enter "limp home" mode? The answers to these questions are non-trivial due to the following facts.

First, since ECUs have different ASILs (Automotive Safety Integrity Levels), post-detection reactions should be different. That is, since some ECUs would be more safety-critical than others (e.g., brake ECU definitely considered safety-critical whereas heating/ventilation ECU might not be), if those safety-critical ones were compromised, simply alerting the driver would be insufficient. Similarly, if the compromised ECU turns out to be not as safety-critical as others (e.g., heating/ventilation ECU), enforcing that ECU to not transmit/receive messages to/from the bus but still letting it independently run with its pre-set parameters (and thus with reduced functionality) might be an appropriate counterplan. Enforcing the car to enter limp-home mode due to a security (or even a functional) problem in the heating/ventilation ECU might be unacceptable to the driver/passenger(s). Second, the effectiveness of such alerts or post-detection reactions will, in fact, be dependent on various *physical* factors. Specifically, depending on the vehicle speed, location, its surrounding, and, more importantly, the driver's awareness (e.g., attentive vs. non-attentive), an alert itself might not be sufficient. Therefore, in order to (properly) determine which type of post-detection reaction to make, there should be an in-depth understanding/analysis of 1) the criticality of different vehicle cyber attacks and 2) how to take into account the physical factors that might affect the effectiveness of the counterplan(s). Without a proper design of post-detection/identification counterplans, the driver will still remain unsafe.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Application-Driven Power Management Keys In-Car Telematics. https://web.archive.org/web/20171105225654/https://www.eetimes.com/.

[2] CAN Specification Version 2.0. http://web.archive.org/web/20170926054355/https://www.nxp.com/.

[3] CAN with Flexible Data-Rate Specification Version 1.0. http://web.archive.org/web/20170927184813/https://vector.com/.

[4] DIN 72552 Standard. https://web.archive.org/web/20171102195050/https://www.din.de/en.

[5] Future Advances in Body Electronics. http://web.archive.org/web/20170926054355/https://www.nxp.com/.

[6] Introduction to the Local Interconnect Network (LIN) Bus. https://web.archive.org/web/20160714052136/http://www.ni.com/white-paper/.

[7] ISO 11898-1. Road Vehicles interchange of digital information controller area network (CAN) for high-speed communication. http://web.archive.org/web/20170609150140/https://www.iso.org/.

[8] ISO 11898-2. Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit. http://web.archive.org/web/20170609150140/https://www.iso.org/.

[9] Microchip AN228 - CAN Physical Layer Discussion. http://web.archive.org/web/20171029071459/http://www.microchip.com/.

[10] Microchip MCP2515 Datasheet. http://web.archive.org/web/20171029071459/http://www.microchip.com/.

[11] Microchip TB078, PLL Jitter and Its Effects in the CAN Protocol. http://web.archive.org/web/20171029071459/http://www.microchip.com/.

[12] NHTSA V2V Communications. http://web.archive.org/web/20170824070953/https://www.safercar.gov/v2v/.

[13] On-Board Diagnostic System. http://web.archive.org/web/20170917021231/http://obdii.com/.

[14] Optimizing MOSFET Characteristics by Adjusting Gate Drive Amplitude. `http://web.archive.org/web/20171031194856/http://www.ti.com/`.

[15] Philips/NXP SJA1000 Stand-alone CAN controller datasheet. `http://web.archive.org/web/20170926054355/https://www.nxp.com/`.

[16] SN65HVD1040-Q1 EMC-Optimized Can Transceiver Datasheet. `http://web.archive.org/web/20171031194856/http://www.ti.com/`.

[17] Symptoms of Improper CAN Termination. `http://web.archive.org/web/20170823002636/http://digital.ni.com/`.

[18] TCAN1051 Fault Protected CAN Transceiver with CAN FD Datasheet. `http://web.archive.org/web/20171031194856/http://www.ti.com/`.

[19] This Car Runs on Code. `http://web.archive.org/web/20171102001517/https://spectrum.ieee.org/`.

[20] Vector: Common High Speed Physical Layer Problems. `http://web.archive.org/web/20170927184813/https://vector.com/`.

[21] Volvo wants your phone to be the only car key you ever need. `https://web.archive.org/web/20170703130519/https://www.theverge.com/`, 2016.

[22] Lexus 2018 Technology. `http://www.herbchamberslexusofhingham.com`, 2017.

[23] Tesla Model 3 has no key: so don't forget your phone. `https://web.archive.org/web/20171115054847/https://www.cnet.com/`, 2017.

[24] Infineon: CAN FD Success Goes at Expense of FlexRay. `http://web.archive.org/web/20171006031518/https://www.eetimes.com/`, Feb. 2015.

[25] Hackers Remotely Kill a Jeep on the Highway - With Me in It. `http://web.archive.org/web/20171104002232/https://www.wired.com/`, Jul. 2015.

[26] Tesla Responds to Chinese Hack With a Major Security Upgrade. `http://web.archive.org/web/20171104002232/https://www.wired.com/`, Sep. 2016.

[27] F. Bai, H. Krishnan, V. Sadekar, G. Holland, and T. ElBatt. Towards characterizing and classifying communication-based automotive applications from a wireless networking perspective. In *Proceedings of IEEE Workshop on Automotive Networking and Applications*, 2006.

[28] M. Basseville and I. V. Nikiforov. *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[29] R. R. Brooks, S. Sander, J. Deng, and J. Taiber. Automobile security concerns. *IEEE Vehicular Technology Magazine*, 4(2):52–64, June 2009.

[30] Bureau of Transportation Statistics. Retail New Passenger Car Sales. *U.S. Department of Transportation*, Updated Apr. 2015.

[31] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 6–6, Berkeley, CA, USA, 2011. USENIX Association.

[32] 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*, CCS '16, pages 1044–1055, New York, NY, USA, 2016. ACM.

[33] K.-T. Cho and K. G. Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 911–927, Austin, TX, 2016. USENIX Association.

[34] K.-T. Cho, K. G. Shin, and T. Park. CPS approach to checking norm operation of a brake-by-wire system. In *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, ICCPS '15, pages 41–50, New York, NY, USA, 2015. ACM.

[35] W. Choi, H. J. Jo, S. Woo, J. Y. Chun, J. Park, and D. H. Lee. Identifying ecus using inimitable characteristics of signals in controller area networks. *CoRR*, abs/1607.00497, 2016.

[36] J. Daily. Analysis of critical speed yaw scuffs using spiral curves. In *SAE Technical Paper*. SAE International, 04 2012.

[37] R. I. Davis, S. Kollmann, V. Pollex, and F. Slomka. Controller area network (can) schedulability analysis with fifo queues. In *2011 23rd Euromicro Conference on Real-Time Systems*, pages 45–56, July 2011.

[38] D. Doerffel and S. A. Sharkh. A critical review of using the Peukert equation for determining the remaining capacity of lead-acid and lithium-ion batteries. *Journal of Power Sources*, 155:395–400, 2006.

[39] T. ElBatt, S. K. Goel, G. Holland, H. Krishnan, and J. Parikh. Cooperative collision warning using dedicated short range wireless communications. In *Proceedings of the 3rd International Workshop on Vehicular Ad Hoc Networks*, VANET '06, pages 1–9, New York, NY, USA, 2006. ACM.

[40] I. Foster and K. Koscher. Exploring controller area networks. In *USENIX ;Login: magazine*. USENIX Association, 2015.

[41] I. Foster, A. Prudhomme, K. Koscher, and S. Savage. Fast and vulnerable: A story of telematic failures. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, Washington, D.C., 2015. USENIX Association.

[42] A. Francillon, B. Danev, and S. Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *In Proceedings Of The 18th Annual Network and Distributed System Security Symposium (NDSS)*, 2011.

[43] S. Haykin. *Adaptive Filter Theory (3rd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[44] T. Hogenmuller and H. Zinner. Tutorial for Wake Up Schemes and Requirements for Automotive Communication Networks. `http://grouper.ieee.org/groups/802/3/RTPGE/`.

[45] T. Hoppe, S. Kiltz, and J. Dittmann. *Security Threats to Automotive CAN Networks – Practical Examples and Selected Short-Term Countermeasures*, pages 235–248. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[46] T. Hu. *Deterministic and flexible communication for real-time embedded systems*. PhD thesis, Politecnico di Torino, 2015.

[47] S. Jana and S. K. Kasera. On fast and accurate detection of unauthorized wireless access points using clock skews. *IEEE Transactions on Mobile Computing*, 9(3):449–462, March 2010.

[48] D. A. Khan, R. J. Bril, and N. Navet. Integrating hardware limitations in can schedulability analysis. In *2010 IEEE International Workshop on Factory Communication Systems Proceedings*, pages 207–210, May 2010.

[49] T. Kohno, A. Broido, and K. Claffy. Remote physical device fingerprinting. In *2005 IEEE Symposium on Security and Privacy*, pages 211–225, May 2005.

[50] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462, May 2010.

[51] G. Leen and D. Heffernan. Ttcan: a new time-triggered controller area network. *Microprocessors and Microsystems*, 26(2):77 – 94, 2002.

[52] J. Lepkowski, B. Wolfe, and W. Lepkowski. Emi/esd solutions for the can network. In *Proceedings. 2005 IEEE Networking, Sensing and Control, 2005.*, pages 413–418, March 2005.

[53] T. Liebetrau, U. Kelling, T. Otter, and M. Hell. Energy Saving in Automotive E/E Architectures. `https://www.infineon.com/`, Dec. 2012.

[54] J. Mikulski. *Transport Systems Telematics: 10th Conference, TST 2010, Katowice - Ustron, Poland, October 20-23, 2010. Selected Papers*. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2010.

[55] P. Milbredt, M. Horauer, and A. Steininger. An investigation of the clique problem in flexray. In *2008 International Symposium on Industrial Embedded Systems*, pages 200–207, June 2008.

[56] C. Miller and C. Valasek. Adventures in automotive networks and control units. *Defcon 21*, 2013.

[57] C. Miller and C. Valasek. A survey of remote automotive attack surfaces. *Black Hat USA*, 2014.

[58] C. Miller and C. Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015.

[59] D. Mills. Network time protocol (version 3) specification, implementation, 1992.

[60] S. Mittal. A survey of architectural techniques for managing process variation. *ACM Comput. Surv.*, 48(4):54:1–54:29, Feb. 2016.

[61] S. Mohalik, A. C. Rajeev, M. G. Dixit, S. Ramesh, P. V. Suman, P. K. Pandya, and S. Jiang. Model checking based analysis of end-to-end latency in embedded, real-time systems with clock drifts. In *2008 45th ACM/IEEE Design Automation Conference*, pages 296–299, June 2008.

[62] D. Montgomery. *Introduction to statistical quality control*. Wiley, New York, NY [u.a.], 3. ed edition, 1997.

[63] S. B. Moon, P. Skelly, and D. Towsley. Estimation and removal of clock skew from network delay measurements. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 227–234 vol.1, Mar 1999.

[64] P. S. Murvay and B. Groza. Source identification using signal characteristics in controller area networks. *IEEE Signal Processing Letters*, 21(4):395–399, April 2014.

[65] M. Muter and N. Asaj. Entropy-based anomaly detection for in-vehicle networks. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 1110–1115, June 2011.

[66] M. Muter, A. Groll, and F. C. Freiling. A structured approach to anomaly detection for in-vehicle networks. In *2010 Sixth International Conference on Information Assurance and Security*, pages 92–98, Aug 2010.

[67] A. Mutter and F. Hartwich. Advantages of CAN FD Error detection mechanisms compared to Classical CAN. In *CAN in Automation iCC*, 2015.

[68] M. D. Natale, H. Zeng, P. Giusto, and A. Ghosal. *Understanding and Using the Controller Area Network Communication Protocol: Theory and Practice*. Springer Publishing Company, Incorporated, 2012.

[69] D. Nilsson and U.Larson. A Roadmap for Securing Vehicles against Cyber Attacks. In *NITRD National Workshop on High-Confidence Automotive Cyber-Physical Systems*, Apr. 2008.

[70] D. K. Nilsson and U. E. Larson. Simulated attacks on can buses: Vehicle virus. In *Proceedings of the Fifth IASTED International Conference on Communication Systems and Networks*, AsiaCSN '08, pages 66–72, Anaheim, CA, USA, 2008. ACTA Press.

[71] D. K. Nilsson, U. E. Larson, and E. Jonsson. Efficient in-vehicle delayed data authentication based on compound message authentication codes. In *2008 IEEE 68th Vehicular Technology Conference*, pages 1–5, Sept 2008.

[72] A. Palanca. *A Stealth, Selective, Link-layer Denial-of-Service Attack Against Automotive Networks*. PhD thesis, Politecnico Milano, 2016.

[73] A. Pásztor and D. Veitch. Pc based precision timing without gps. In *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '02, pages 1–10, New York, NY, USA, 2002. ACM.

[74] V. Paxson. On calibrating measurements of packet transit times. *SIGMETRICS Perform. Eval. Rev.*, 26(1):11–21, June 1998.

[75] O. Pfeiffer, A. Ayre, and C. Keydel. *Embedded Networking with CAN and CANopen*. Copperhill Media Corporation, USA, 1st edition, 2008.

[76] I. Rouf, R. Miller, H. Mustafa, T. Taylor, S. Oh, W. Xu, M. Gruteser, W. Trappe, and I. Seskar. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[77] J. Rufino, P. Verissimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcasts in can. In *Digest of Papers. Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing (Cat. No.98CB36224)*, pages 150–159, June 1998.

[78] R. Ruth, W. Bartlett, and J. Daily. Accuracy of event data in the 2010 and 2011 toyota camry during steady state and braking conditions. *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, 5(1):358–372, apr 2012.

[79] C. Ryan, D. Heffernan, and G. Leen. Clock synchronisation on multiple ttcan network channels. *Microprocessors and Microsystems*, 28(3):135 – 146, 2004.

[80] M. C. Schneider. *Mosfet Modeling for Circuit Analysis And Design*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2007.

[81] H. Schweppe, Y. Roudier, B. Weyl, L. Apvrille, and D. Scheuermann. Car2x communication: Securing the last meter. *WIVEC*, 2011.

[82] Y. Shoukry, P. Martin, P. Tabuada, and M. Srivastava. Non-invasive spoofing attacks for anti-lock braking systems. In *Proceedings of the 15th International Conference on Cryptographic Hardware and Embedded Systems*, CHES'13, pages 55–72, Berlin, Heidelberg, 2013. Springer-Verlag.

[83] C. Szilagyi and P. Koopman. Low cost multicast network authentication for embedded control systems. In *Proceedings of the 5th Workshop on Embedded Systems Security*, 2010.

[84] S. C. Talbot and S. Ren. Comparision of fieldbus systems can, ttcan, flexray and lin in passenger vehicles. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems Workshops*, ICDCSW '09, pages 26–31, Washington, DC, USA, 2009. IEEE Computer Society.

[85] E. Tran. *Multi-Bit Error Vulnerabilities in the Controller Area Network Protocol*. PhD thesis, Carnegie Mellon University, 1999.

[86] A. Van Herrewege, D. Singele, and I. Verbauwhede. Canauth - a simple, backward compatible broadcast authentication protocol for can bus. In *ECRYPT Workshop on Lightweight Cryptography*, page 7, 01 2011.

[87] D. Veitch, S. Babu, and A. Pàsztor. Robust synchronization of software clocks across the internet. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, pages 219–232, New York, NY, USA, 2004. ACM.

[88] W. Woodall and B. Adams. The statistical design of cusum charts. *Quality Engineering*, 5(4):559–570, 1993.

[89] F. Yang. A bus off case of can error passive transmitter. *EDN Technical Paper*, Dec. 2008.

[90] S. Zander and S. J. Murdoch. An improved clock-skew measurement technique for revealing hidden services. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 211–225, Berkeley, CA, USA, 2008. USENIX Association.