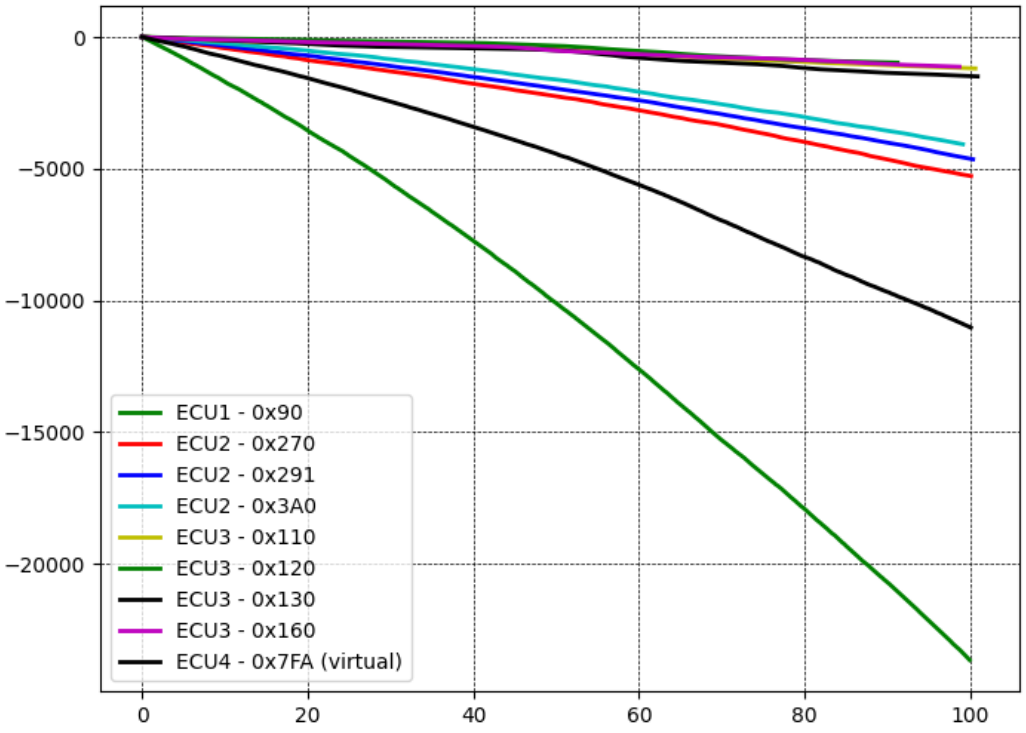


Attacker Identification Using Low-Level Characteristics of Automotive ECUs

Master's Thesis in Computer Science and Engineering

DEEPAK DESAI

BURKIN GÜNKE

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

Attacker Identification Using Low-Level Characteristics of Automotive ECUs

DEEPAK DESAI

BURKIN GÜNKE

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Attacker Identification Using Low-Level Characteristics of Automotive ECUs

DEEPAK DESAI and BURKIN GÜNKE

Supervisor: Magnus Almgren, Department of Computer Science and Engineering
Academical Advisor: Wissam Aoudi, Department of Computer Science and Engineering
Co-Supervisor & Industrial Advisor: Nasser Nowdehi, Volvo Car Corporation
Examiner: Tomas Olovsson, Department of Computer Science and Engineering

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Nine voltage profiles from four ECUs sending nine different message-IDs on the Boxcar.

Typeset in LaTeX
Gothenburg, Sweden 2020

Abstract

The Controller Area Network (CAN) is one of the most important In-Vehicle Network (IVN) protocols used for reliable communication between Electrical Control Units (ECUs). ECUs are responsible for critical in-vehicle operations such as transmission, brakes and active safety (e.g., airbag deployment) among others. However, the CAN protocol lacks basic security features such as message authentication and encryption, making it vulnerable to a variety of attacks such as message spoofing, replication, fabrication and denial of service.

In order to detect these attacks and proactively protect the ECUs, researchers have proposed intrusion detection systems for vehicles. Since the majority of the IVN traffic is highly regular, most of the proposed solutions aim at detecting anomalies in the vehicle by evaluating incoming in-vehicle messages against potential irregularities. Despite these efforts, there are not many works done on associating a malicious CAN message to its origin and thereby locating the source of an attack.

Recently attacker identification methods for IVNs have been introduced. The proposed solutions focus on the low-level characteristics of the ECUs such as voltage, clock-skew or clock-offset to fingerprint ECUs and to identify the attacker ECU. Given that these methods have recently been proposed in the literature, there is a need to investigate and verify the applicability and practicality of the proposed methods and identify the challenges of implementing them.

In this work, we study two of the most prominent automotive IDS solutions proposed in the literature recently; CASAD as an IDS and Viden as a fingerprinting-technique based on ECU voltage characteristics. We mainly focus on assessing the performance of Viden with respect to detection accuracy, viability, practicality and efficiency by implementing a proof-of-concept of the proposed method. We replicate the algorithm used by Viden, and as an extended objective of the thesis, we also investigate whether CASAD's detection engine can be extended to use the ECU voltage behaviour for distinguishing different ECUs from each other, thus detecting the source of an attack. Finally, we propose a unified system where CASAD detects the attack and Viden identifies the source of the attack.

Keywords: Vehicle-cybersecurity, Controller Area Network, CAN, Bus, Attacker Identification, Engine Control Unit, ECU, In-Vehicle Networks.

Acknowledgements

We would like to thank our supervisor Magnus Almgren and our academical advisor Wissam Aoudi, both from the Department of Computer Science and Engineering - Chalmers University of Technology, for their constant guidance and support. We would also like to the thank our supervisor at Volvo Cars, Nasser Nowdehi, for giving us an opportunity to conduct this thesis at Volvo Cars by providing the necessary resources and being consistently supportive throughout the thesis. We also express our sincere gratitude to the examiner, Tomas Olovsson, who also introduced us to the thesis and provided an opportunity to work as a team. We would also want to thank Sara Boström and Jonatan Nylund for being the opponents during the thesis presentation and having an intriguing discussion. Last but not least, we would like to thank our friends and family who have been a constant source of support and encouragement.

DEEPAK DESAI and BURKIN GÜNKE,
Gothenburg, October 2020

Contents

List of Figures

List of Tables

1
Introduction

With the advent of technology, modern vehicles have evolved from just transportation systems to offering infotainment systems, Internet-enabled services, and much more. In recent years, driver-less vehicles have been realized, all thanks to significant efforts and resources invested by researchers. As shown in Figure 1.1, a network of Electrical Control Units (ECUs), called the In-Vehicle Network (IVN), is responsible for controlling various functionalities like vehicle-manoeuvrability, fuel-efficiency, air-conditioning and entertainment-systems.

ECUs can communicate via various in-vehicle communication technologies such as Controller Area Network (CAN), Local Interconnect Network (LIN), Media Oriented Systems Transport (MOST), FlexRay, and Ethernet. CAN is a fast and reliable bus technology that supports single-channel communication and is used for critical real-time tasks such as transmission, brakes and airbag deployment. LIN was developed as a low-cost alternative for CAN where communication speed is not critical. FlexRay, which supports two-channel communication, was designed in order to provide more reliable and faster communication than CAN. In recent years, Automotive Ethernet has also been developed for vehicles to enable greater bandwidth usage primarily for diagnostic services and in-vehicle infotainment systems.

Among these protocols, CAN is the only message-based networking protocol with prioritization-features. In addition to this, several other advantages such as low cost, built-in error detection, robustness, speed, flexibility, among others, are associated with this protocol and are detailed in Section 2.1 [2]. CAN eliminates the need for excessive wiring by connecting electronic devices to a single multiplex wire enabling these devices to communicate with other nodes on the same bus. These features make the CAN-protocol the first-hand choice for implementation in various industrial applications, mainly automotive networks. Since its first implementation in 1991 [3], CAN has been by far the most widely used standard for in-vehicle communication.

CAN was designed with the assumption that it is going to be used for high-speed

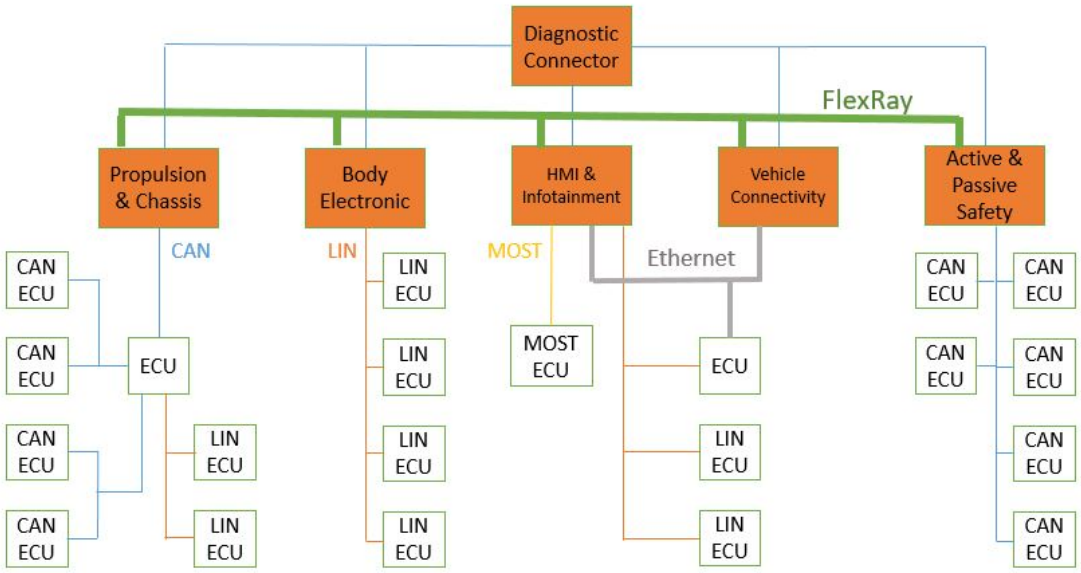


Figure 1.1: Typical in-vehicle network with a FlexRay backbone [1]

communication in closed networks, and cybersecurity was not considered as a prominent design aspect. The original design of CAN was a single serial bus connecting all the nodes into a single network. This provided absolutely no segmentation in the network as all the messages were broadcasted. However, as shown in Figure 1.1, this has changed over time, and nowadays the IVN is divided into multiple domains based on the functionality such as propulsion and chassis, body electronics, active and passive safety, among others, preventing ECUs from sending messages across domains. Despite such segmentation, it is still possible for an ECU to send messages to other safety-critical ECUs which exist in the same domain. For example, in the IVN shown in Figure 1.1, any ECU in the *"Propulsion & Chassis"* domain can communicate with any other ECU in the same domain.

In order to provide more functionalities and thus enhance the user experience, modern vehicles rule out the assumption of a closed operating environment by allowing external (e.g. the Internet) connections to some of the ECUs in order to provide navigation and entertainment services among others. Although such level of connectivity has added numerous new functionalities, it has also made the vehicles vulnerable to remote attacks [4, 5]. Most of the safety-critical in-vehicle ECUs are implemented on the CAN-bus where they can transmit messages to each other, and also listen to messages sent on the bus. As a result, if an adversary compromises one or more ECUs, (s)he gains access to the CAN-bus, and can then send spoofed

messages, claiming to have originated from legitimate ECUs. Depending on other safety-critical ECUs that belong to the same domain as the compromised ECU, the attack can vary in magnitude and consequence. Researchers have demonstrated various attack methods [6], [7], [8] which can be performed either remotely by compromising the ECU(s) communicating with external networks through cellular [9], or a multitude of attacks could also be exploited through USB, Bluetooth, and WiFi by standing close the vehicle. An attacker having physical access can also make use of the OBD-II (on-board diagnostics) connector that gives access to various vehicle subsystems, to monitor or inject CAN-messages.

The increase in cyberattacks on vehicles over the past decade [10], [11], has developed a curiosity in researchers, and various defence measures have been proposed as a result. These measures can be broadly classified as message authentication schemes and Intrusion Detection Systems (IDS(s)). Considering the limited size of a CAN-message and tight resource constraints from the automotive systems, all proposed message authentication schemes have failed to be deployed in real-world vehicles [1]. This makes it essential to be able to detect attacks in order to proactively protect the ECUs. As further described in Section 2.3.2, the proposed in-vehicle IDSs have been effective in passively monitoring IVN traffic, thus, being more feasible to adhere to the highly demanding resource constraints from IVNs. Owing to the long life-span of vehicles and less demand for updates, anomaly-based IDSs have been preferred over signature-based IDSs [12].

A majority of anomaly-based IDSs [13], [14], [15], [16], take into account the regularity of timing behaviour of IVN messages and detect anomalous deviations in the periodicity of these messages. Such changes usually indicate the presence of an intrusion in the IVN. Although these solutions are able to detect an attack, they fail to detect the source of the attack [17]. ECUs are natively associated with unique low-level characteristics such as voltage, clock skew and clock offset. Thus, some of the recent studies aim at exploiting one of these characteristics to detect the attacker ECU. Although these methods are successful in detecting the source of an attack (not the attack itself), there are major shortcomings of such approaches [12]. First, these methods are not effective when an attacker inflicts certain kinds of attacks, usually known as stealthy attacks, that do not cause a noticeable change in the IVN dynamics. Second, these methods are system-specific as they require knowledge about the underlying IVN traffic such as frame ID, message sequence, and transmission frequency. These factors make existing approaches in-comprehensive. In order to address these shortcomings, Nowdehi et al. propose a technique named CASAD to analyse and detect attacks by capturing deterministic behaviour of IVN dynamics by processing CAN-traffic at the payload level [12]. Although CASAD addresses the

issues with the other in-vehicle IDSs, it is not capable of detecting the source of an attack.

1.1 Aim

The existing attacker identification methods have been introduced recently, and as such, require assessment. That is, there is a need to study their *accuracy*, *practicality* and *efficiency*. In this thesis, we focus on Viden [17], one of the most prominent methods to fingerprint ECUs based on voltage as a low-level characteristic. We perform a systematic study and evaluation of this method to understand the voltage characteristics it exploits. As a secondary goal, we aim to make a thorough investigation of CASAD and implement it for two prominent reasons. First, to have a system which can detect attacks and then trigger Viden to identify the source. Second to better understand how CASAD works and see whether CASAD's payload analysis can be extended to analyse voltage for detecting the source of an attack.

1.2 Problem Formulation

The thesis aims to provide answers to the following questions.

1. Investigate the performance of Viden with respect to detection accuracy, viability, practicality, and efficiency. Can it be replicated?

2. Review the algorithm used by CASAD and replicate its implementation.

3. Propose a unified system by using CASAD for attack detection and Viden for source identification.

4. Experiment if it possible for CASAD's detection engine to analyse voltage measurements instead of payloads.

1.3 Limitations

The primary focus of this thesis is to investigate and implement Viden, which is challenging due to the technical complexity of this algorithm. Only if time allows, we also intend to extend CASAD's features to analyse voltages measurements as a secondary goal of the thesis.

1.4 Outline

The report is organized as follows: Chapter 1 gives a brief introduction to vehicle-cybersecurity, the recent attacks on vehicles and the corresponding defence mechanisms that have been proposed over time. Chapter 2 presents the relevant background information for the thesis by highlighting the significance of the CAN-protocol, the vulnerabilities it is prone to, and the types of attacks possible. The defence-mechanisms proposed in the form of Message Authentication Schemes and different types of IDSs for vehicles are also discussed in detail. The discussion on fingerprinting ECUs based on their voltage behaviour followed by the description on both Viden [17] and CASAD [12] are also included here. Chapter 3 explains the existing work in the area of IDSs based on low-level characteristics and ends with a detailed study on CASAD's detection engine, including its implementation and limitations. Chapter 4 presents a detailed study on Viden, the challenges we faced while implementing it and the corresponding roundabouts we took in order to achieve the replication of Viden. In Chapter 5, we explain in detail the experiments we performed in order to evaluate the proposed methods. In Chapter 6, we present and discuss the results. Finally, the conclusion of the thesis is given in Chapter 7.

2
Background

In this chapter, we describe in detail the CAN-protocol, its significance, and how CAN-messages are transmitted. Then we describe the vulnerabilities associated with the CAN-protocol and the type of attacks that can be carried out by exploiting these vulnerabilities. We then briefly describe the countermeasures that have been developed in the form of Message Authentication Schemes, different types of IDSs for vehicles and fingerprinting ECUs using voltage characteristics. We end this chapter by providing a brief introduction to Viden and CASAD.

2.1 The Controller Area Network Protocol

The Controller Area Network protocol (CAN) is one of the most widely used standards for in-vehicle communication, developed by *Robert Bosch GmbH* [3]. CAN is a message-based protocol which was designed to enable communication between electric components such as ECUs, microcontrollers, sensors and actuators through a single/dual wire bus, providing bitrates up to 1Mbit/s. With properties such as prioritization of messages, guaranteed latency times, error detection and signalling, configuration flexibility, i.a., some of which are detailed later, CAN finds applications in domains ranging from high-speed networks to low-cost multiplex wiring.

The CAN-bus can have one of two complementary logical values: *dominant* (0-bit) or *recessive* (1-bit). The ID-field of every transmitted CAN data frame indicates the priority of the data it is carrying, lower values signifying higher priority. The priority of messages prevents collisions within the CAN-bus traffic, and only the message with the highest priority is delivered to all nodes on the same bus [11]. Since CAN-messages are broadcast, all nodes can listen to incoming messages from any node on the same bus.

2.1.1 Significance of CAN

CAN is a Carrier-Sense Multiple Access protocol with Collision Detection and Arbitration on Message Priority *(CSMA/CD+AMP)* [18]. Each node on the bus waits

for a specific period of inactivity before attempting to transmit a message as referenced to by CSMA. Collision detection and arbitration on message priority together resolve collisions through bit-wise arbitration based on a pre-programmed priority for each message.

A CAN-bus can be manifested to transmit the following four different types of messages [3]:

- **Data-Frame**: This is the most common type of frame and contains the *data-field*, *CRC* and *ID-field*.
- **Remote-Frame**: This type of frame essentially has no data but is intended to solicit the transmission of data from another node by explicitly marking the *RTR-field*.
- **Error-Frame**: This is used when one of the nodes on the CAN-bus detects an error in the message and propels all other nodes to transmit an error message as well.
- **Overload-Frame**: This is used for completeness and to provide an extra delay between messages.

Usually, an *"Interspace Frame"* is used to separate the data-frames and remote frames from the preceding frames.

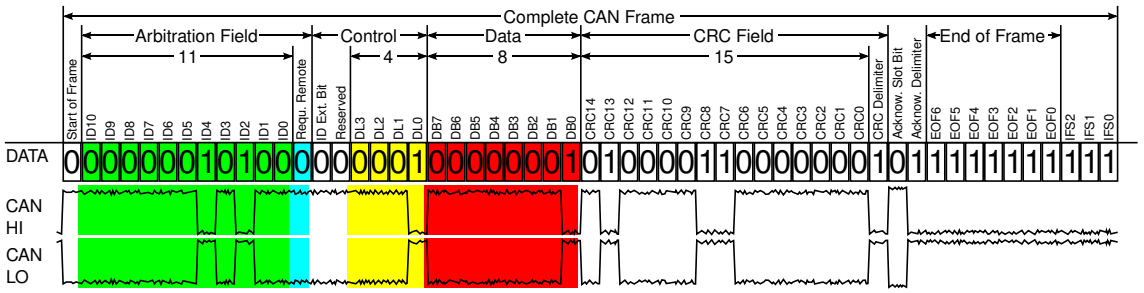


Figure 2.1: Structure of a CAN frame format [19].

Figure 2.1 shows the general structure of a CAN data-frame which is composed of different bit-fields, detailed as follows:

- **Start-of-Frame**: This marks the beginning of the data and remote-frames and consists of a single dominant bit.
- **Arbitration-Field**: It consists of two fields. First, the *Identifier* (ID) field (indicated by green colour in the figure) signifies the priority of the message. Second, the *Remote Transmission Request* (RTR) field (indicated by the blue colour in the figure) is a dominant value in data frames and a recessive value in remote-frames. For a *Standard CAN frame*, the ID field is 11-bit long, while an *Extended CAN frame* has a 29-bit ID field. The RTR field is always 1-bit long.

- **Control-Field**: It includes the *ID Ext. bit*, the *Reserved-bit* along with the *Data Length Code* (DLC) field (is 4-bit long and is indicated by the yellow colour in the figure). These fields combined occupy a space of six bits between the ID and Data-field.
- **Data-Field**: This field (indicated by red colour in the figure) consists of the data to be transferred within the frame and can contain up to 8 bytes (0-64 bits) of data.
- **CRC-Field**: This is a 16-bit long field which can be used to verify the integrity of the data-frame in transmission and an error-frame can be transmitted if this fails.
- **ACK-Field**: This is a two-bit value comprising of a 1-bit *"ACK-Slot"* and a 1-bit *"ACK-Delimiter"*, used to report acknowledgement when a message is received.
- **End of Frame**: A flag sequence consisting of seven recessive bits delimits each data and remote frame, marking the end of the frame.

2.1.2 Transmission of CAN-Messages

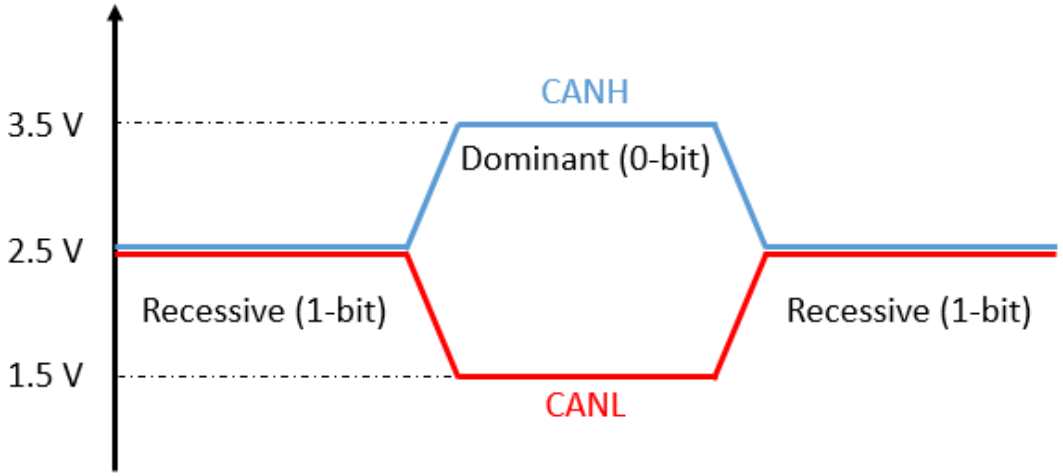


Figure 2.2: Output-voltage of CAN-Messages [17].

All the fields of the CAN-frame are composed of a sequence of either *dominant* (0) or *recessive* (1) bits. These bits are agreed upon by the CAN-transceivers to be of certain voltage levels on the two dedicated CAN wires *CANH* and *CANL* [17]. As shown in Figure 2.2, a 0-bit is corresponding to a value of approximately 3.5V and 1.5V on the CANH and CANL wires so that the differential-voltage is

approximately 2V. In order to send a 1-bit, the transceiver outputs 2.5V on both the wires making the difference to be approximately 0V. Thus, by measuring the differential voltage of CANH and CANL, it is possible to read the incoming stream of 0 and 1-bits, and hence, the message. CAN-transceivers correspond to different voltage-levels by simultaneously switching on/off two transistors. For a 0-bit, both transistors are turned on, resulting in a low-impedance state and the differential-voltage of approximately 2V flowing from V_{cc} to the ground. To send a 1-bit, on the other hand, both transistors are turned off, making it a high-impedance state. The following section describes how the CAN-communication characteristics have been exploited by adversaries.

2.2 CAN - Vulnerabilities and Attacks

Even though CAN is very popular, in the last couple of years a number of attacks in the wild have been seen [9], [10]. The lack of message/device authentication by the CAN-protocol is the main reason behind most of the vulnerabilities associated with it. In other words, the protocol can not associate a message with its sender and thus fails to distinguish between a legitimate and a malicious ECU in the network. Adding to the vulnerabilities is the absence of any encryption standard by the CAN-protocol, hindering integrity and confidentiality of the messages [20]. Implementation of a robust cryptographic algorithm is difficult due to the low computational power of the ECUs along with other constraints such as lack of backward-compatibility, cost-ineffective and insufficient implementation details, among others [1]. This makes it possible for any device on the same bus to listen and intercept messages.

Messages on a CAN-bus are transmitted in sequential order except when multiple devices transmit at the same time. In the latter case, the message with the highest priority continues while others retreat, e.g., the CAN-protocol allows messages to take priority over others in accessing the shared medium. An attacker can exploit this characteristic to continuously send high-priority messages and flood the bus and thereby exhaust bandwidth. This creates a situation where other nodes can hardly transmit any messages, thus making CAN vulnerable to Denial of Service (DoS) attacks.

Adversaries can compromise one or more ECUs on a CAN-bus in essentially two different ways. First, the adversary can compromise a target ECU to passively listen to communication on the bus. In this scenario, the attacker does not have any access to the memory of the weakly/partially compromised ECU [12]. A weakly compromised ECU cannot be used to inject arbitrary messages but can be prevented from

transmitting its own messages. On the other hand, an adversary can compromise the target ECU in such a way that, (s)he has complete knowledge about the specifications of the ECU, has full access to its memory and can use it to inject arbitrary messages on the bus (strongly/fully compromised ECUs). The adversary can use one or both types of compromised ECUs to inflict different types of attacks on the bus, described as follows in [12]:

1. **Suspension Attack:** In this type of attack, a weakly compromised ECU is prevented from further operation by suspending its actions (of transmitting messages). This, for instance, can be achieved by putting the ECU into programming-mode so that it is no longer able to transmit messages. This can also result in improper functioning of the receivers which depend on incoming data from the suspended ECU.
2. **Fabrication Attack:** The adversary is incapable of compromising the target ECU, but fully compromises another ECU on the same bus and uses it as a means to impersonate the target ECU. The compromised ECU injects fabricated messages into the bus at a higher frequency than the target ECU, thus abolishing the legitimate messages. These forged messages can go unnoticed since all of the injected messages use legitimate IDs belonging to the target ECU.
3. **Masquerade Attack:** This is a combination of both suspension and fabrication attack where the adversary weakly compromises the target ECU and further suspends it from transmitting any messages. Making use of another fully compromised ECU on the same bus, the attacker impersonates the target ECU and sends messages at the original frequency as the target ECU. As a result, there is no violation of arrival times for these messages. Since no conflicting signal values are arriving from the target ECU, the messages appear as if they originated from the actual source.
4. **Conquest Attack:** The authors of CASAD [12] introduced a truly stealthy attack known as the conquest attack where the attacker can avoid both the IDS and other security protection mechanisms. Unlike other attacks, the target ECU is directly conquered and fully compromised by the adversary. It is then reprogrammed to manipulate the payload of the sensitive message, thus causing the receiver ECU to function erroneously. In contrast to all other attack scenarios mentioned above, this attack does not change the normal behaviour of any of the ECUs; hence it is truly stealthy.

These attacks performed individually or in combination can result in the vehicle being completely controlled by the attacker and thereby posing serious safety-risks to the passengers and pedestrians. The defence-mechanisms, which are discussed in the following section, address one or more types of these attacks and the techniques

used to mitigate the vulnerabilities which are exploited by the attackers.

2.3 CAN - Defence Mechanisms

The vulnerabilities associated with the CAN-protocol, as discussed in Section 2.2 has motivated researchers to propose several methods and approaches to addressing multiple issues related to the protocol. However, as already mentioned, these methods are based on two essential lines of defence: Message Authentication Schemes and IDSs, which are detailed in the subsequent subsections. Despite such efforts, malicious attackers are always perfecting their methods to find loopholes and undermine the existing protection-mechanisms. Thus, it is not enough to guarantee state-of-the-art technology in implementing security at the production-level. Instead, a holistic approach is required where systems are continuously monitored for any suspicious behaviour where IDS being one such approach.

In addition to the two major lines of defence, researchers have proposed to address other vulnerabilities that can originate due to one or more features of the CAN-protocol. For example, as mentioned in Section 2.2, CAN is a priority-based communication-protocol which is a highly desirable feature for IVNs, but a vulnerability in itself, making the vehicle prone to DoS attacks. Si et al. showed that by using a meticulous configuration, a single attacker could block the entire CAN-network and make it unusable [21]. They implemented a priority-based DoS attack at the application-layer as it is responsible for generating and transferring packets to the CAN-transceiver. The attack-model consists of a queue, a packet-generator and a flow-controller, all three of which are used to transmit high-priority messages and thereby inflict a DoS attack [21]. In order to mitigate this problem, they propose to integrate a wireless interface (such as ZigBee, WiFi) and design a hybrid wired/wireless protocol to monitor link-qualities and accordingly schedule packet-transmissions. Through real testbed experiments, the authors demonstrated that the proposed protocol is robust to DoS attacks and increases the capacity of the communication system under normal operations. However, with the introduction of wireless communication into IVNs, the authors mention the possibilities of security issues such as privacy and packet-spoofing.

2.3.1 Message Authentication Schemes

Message Authentication Schemes are one of the most popular cryptographic-based software solutions that researchers have been exploiting in order to propose efficient defence-mechanisms for IVNs. The main idea is to ensure the authentication of the CAN data frame that is being exchanged between ECUs. Over the years, several

methods have been proposed in this direction and have essentially failed, owing to two major shortcomings. First, the CAN data-frame has a fixed size of 8 bytes and implementing any scheme in such a short space is a tedious task. However, as described by Lokman et al., in their work [11], this issue has been tackled using several countermeasures, especially from a cryptographic approach. For example, truncating a *Message Authentication Code* (MAC) across several CAN-frames, utilizing an out-of-band channel to authenticate the message, and using various of the CRC-fields in order to accommodate additional bytes. Second, as mentioned in Section 2.2, the high resource constraints of automotive networks in terms of computational power, capability and response-time make it extremely difficult and complex to implement these schemes.

Woo et al. exploit the vulnerabilities associated with the CAN-protocol and demonstrate a practical long-range wireless attack in a connected car environment using a real vehicle and a malicious Android-based smartphone application. Further, they propose a detailed five-phase security-protocol to combat such attacks symmetric keys for encryption and a MAC for authentication of the data-frames [6]. Many assumptions regarding the system have been made, for example, the gateway ECU (which has higher computational power) and other ECUs use a secure channel to pre-share long-term symmetric keys, and ECUs use message-filtering functionality and only accept messages if the sender-ID is on the list. Despite these assumptions, the authors claim that their proposed method is more efficient than other methods, with respect to authentication delay and communication load.

Cros et al. have recently proposed an authentication solution based on hashing-mechanisms that could be integrated into the CAN-protocol [22]. The authors propose to implement a hashing-process for each message on the CAN-bus, considering the fact that a large number of resources are needed to decrypt a message, which cannot be afforded in an embedded setting. They define a key for a group of messages and store it locally on each ECU associated with the message. Using the data-frame and a unique counter, they compute a secure hash of variable length and reuse parts of CAN extended header to store additional hash-bytes. By relying on regular key rotation, using unique counters and computing new keys dynamically, the authors claim that data can be securely transmitted inside a CAN-bus.

However, such methods have major shortcomings due to which their practical deployment have not been successful. The assumptions made by Woo et al. hinder the ability of the system to withstand various types of attacks. For example, in the proposed system, the ECUs use message-filtering functionality based on the ID of the sender, making it prone to spoofing-attacks. The approach by Cros et al.

depends on generating and storing hashes and other related data such as private keys, counters, among others, which would still result in loss of data space which is scarce in an IVN.

2.3.2 Intrusion Detection Systems for Vehicles

IDSs are one of the most commonly used solutions for monitoring systems and networks for malicious activity and policy violations. The malicious activity can come from within the network; through one of the internal components or from an external component outside the network. Over the past few decades, there has been a tremendous increase in the number of IDS implementations across domains, and vehicles are no exception to this. Based on the method of operation and the detection technique, as already mentioned in Chapter 1, IDSs can be broadly classified into two well-known variants, namely signature-based IDSs and anomaly-based IDSs. The former classifies intrusion by recognizing patterns such as those similar to malware, already known attacks, and malicious activities. The latter detects deviations from a predefined model of normal behaviour to discover anomalies. In addition to these, there are *Intrusion Detection and Prevention Systems* (IDPSs) which actively monitor the system, respond to the detected attacks and thereby making the system more secure by taking preventive actions. In recent years, researchers have developed vehicular-IDSs [12, 13, 14, 15], making use of both signature-based and anomaly-based techniques to identify intrusions.

However, Lokman et al. mention in their paper [11] that IDSs can also be classified based on their placement in the IVN. An IDS directly attached to each ECU in the vehicle is known as host-based IDS, while placing an IDS on the CAN-network and the central gateway of the IVN is known as network-based IDS. In their work, Lokman et al. [11] and Dupont et al. [23] meticulously describe and compare the different types of IDSs for the automotive CAN-bus system that have been introduced over the years. Recently, in-vehicle IDSs have evolved to identify intrusions based on different criteria. We begin by describing three different variants of IDSs, namely, frequency, entropy and machine learning-based IDSs in the following subsections.

Inserting extra CAN-packets into the system is one of the prominent methods to inflict an attack on the CAN-network. However, messages in a CAN-network are strictly ordered and arrive at a specific frequency. Researchers have proposed IDSs which utilize this characteristic to identify anomalies in the CAN network.

1. Song et al. proposed a method that uses predefined CAN-frame frequencies to define a normal status of the system based on regular message intervals. Injected messages would significantly deviate from the normal status and hence be classified as an attack by the IDS [24].

2. Taylor et al. proposed an algorithm that measures inter-packet timing over a sliding window and compares the average times to historical averages to indicate any anomalous behaviour. Data was captured from a real vehicle, a part of which was used to train the algorithm for historical average times [25].

The information-theoretic concept measures the coincidence occurred from a given data-set and utilizes the obtained result as an IDS specification profile. Entropy allows an abstract representation of randomness of data by specifying portions of the dataset against which coincidence has to be measured. Using the notion of entropy, researchers have proposed methods to detect anomalies in a CAN-bus.

1. In their work, Müter et al. exploit the characteristic of low randomness in an automotive network to design an IDS. [26]. As we know from Section 2.2, most attacks on a CAN-bus either inject new messages or change payloads. This will influence the IVN traffic, change the randomness of the bus, and increase the entropy of the network; all which will be identified as an attack by the IDS.
2. Marchetti et al. evaluated the method proposed in [26] by introducing a huge number of forged CAN packets. Their results show that high detection accuracy can be achieved only when a large number of packets are injected quickly. However, the method performed poorly for both low-volume attacks and for messages which have large entropy variations [27].

Machine learning based methods, in general, are broadly classified into supervised and unsupervised-learning. While the former requires fully labelled data for training, the latter does not, hence making it difficult to predict and generate attack-behaviour. This has resulted in researchers opting for supervised or semi-supervised learning based IDSs for vehicles. Deep Neural Networks (DNN) have found applications in a wide range of security aspects.

1. Kang et al. proposed a method to capture and process CAN packets to extract the underlying statistical features of the data. The first phase labels every packet by a binary value (attack/ordinary packet) and uses it to train features based on a DNN model. Using the trained parameters, the second phase extracts features of incoming packets and classifies them accordingly [13].
2. Taylor et al. converted raw CAN-packets into a string data type, which used as an input sequence to predict the next value using a supervised long short-term memory (LSTM) neural network method [28]. The authors use messages transmitted by each node on the CAN-bus to predict the subsequent data-field values and claim that the method can detect a range of known attacks and does not require knowledge of underlying protocols.

In order to further comprehend the reader's understanding of IDSs, we base our

presentation on the works of Lokman et al. [11] and Dunpont et al. [23] and briefly describe a few different IDSs that are of relevance to our thesis. We highlight specific examples of IDSs and the characteristics they use to make decisions in order to label attacks.

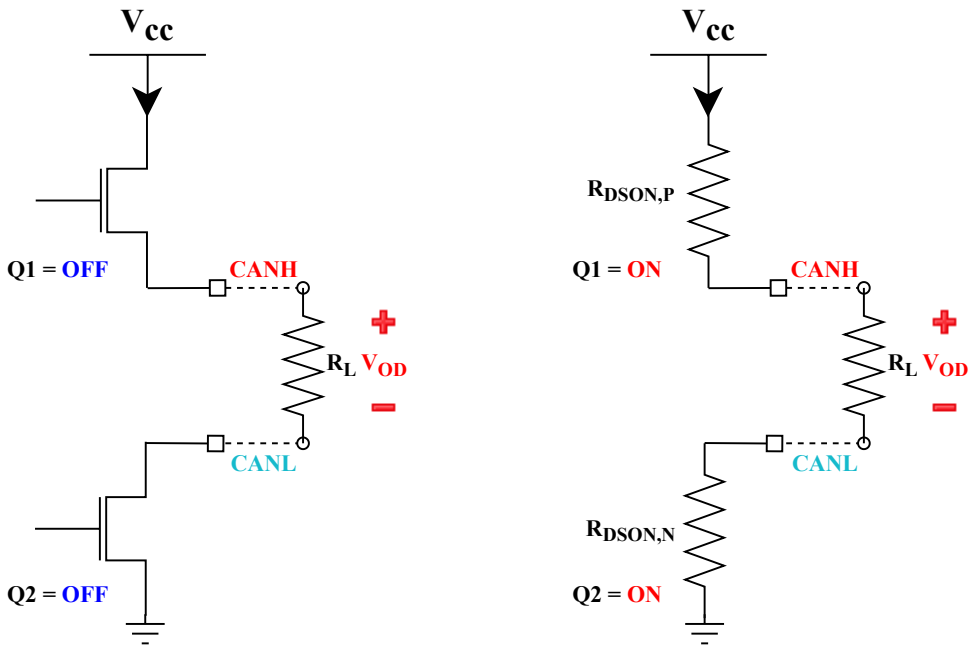
1. Owing to the drawbacks of conventional IDSs that fail to detect and distinguish spoofed messages from legitimate messages, Hamada et al. proposed an IDS to detect attacks at the central gateway of the IVN. They monitor and evaluate the sensor-based control data against any deviations from a pre-specified threshold in order to detect attacks. However, designed specifically only to detect spoofed messages, this method fails to identify any other modifications in the payload of the message, which is a major drawback.
2. Weber et al. introduced a two-stage hybrid anomaly detection technique which is a combination of specification-based system and a machine learning algorithm called Lightweight On-line Detector of Anomalies (LODA) [29]. The first stage corresponds to implementing static checks on a semi-formal network-specification. In the second stage, learning checks were used to observe the behaviour of temporal communication signals which are not specific.
3. The technique of using a distributed IDS was proposed by Wang et al. using a Hierarchical Temporal Memory (HTM), a neural network which integrates massive training and continuous learning at the same time [30]. This method incorporates online learning for streaming data and simultaneously enables the anomaly-detector to learn continuously from CAN-data.
4. Verendel et al. proposed a honeypot-based security-mechanism which would act as a decoy in simulating the IVN [31]. The honeypot was placed at the wireless gateway with an aim to trap the behaviour of attackers who gain access to the vehicle's wireless interface. The collected information was used to analyse the state of the system and enhance it for future versions.
5. Larson et al. [32] proposed a detailed framework consisting of five layers, namely detection, prevention, recovery, countermeasure and deflection in protecting vehicles. Nilsson et al. [33] extended the approach of the five layers by adding detailed features to each layer namely, logging messages for detection, reacting to the attacks through IDPS, message-authentication in preventing unauthorised access and using traceability for recovery.

The IDSs described above are influenced by a variety of approaches from different domains. For example, the work by Hamada et al. was influenced by the security incident on Jeep Cherokee in 2015, Weber et al. used the concept of using the same IDS at multiple locations in an IVN, and Wang et al. were inspired by the distributed protection approach in a cloud environment, and thus introduced the

Distributed Intrusion Detection System (DIDS) for vehicles. Additionally, Kleberger et al. [34] reviewed and discussed the potential vulnerabilities with IVNs. They also highlighted the challenges in implementing security measures and corresponding protection-mechanisms that have been proposed overcoming such challenges.

2.3.3 Fingerprinting ECUs using Voltage-Characteristics

As explained in Section 2.1.2, the CAN-protocol sends messages by using dominant and recessive bits. To understand how CAN-transceivers output the intended voltages and how these can be used to fingerprint ECUs, we must look at the gate level schematics of a CAN-transceiver.



(a) Recessive State (1-bit) **(b)** Dominant State (0-bit)

Figure 2.3: Gate-level schematics of a CAN-transceiver for the recessive and dominant state [35, 36].

In Figure 2.3, we see a typical ECU-connection to a CAN-bus [35, 36]. A voltage-regulator maintains the nominal supply-voltage (V_{cc}) of 5V. The input of the reg-

ulator comes from a power-supply, i.e., 12V/24V battery. According to the ISO 11898-2 standard, transceivers must operate with a common-mode range of -2V to 7V. Since CAN-transceivers measure differential voltage to determine the bus level, transceivers using 3.3V as the supply voltage can communicate with transceivers using 5V as their supply voltage.

The transceivers are designed to accommodate high levels of ground-shift and power-supply imbalance between typical CAN-nodes. This is also known as *"the common-mode voltage operating range"*. Without this design choice, the transceivers would be affected with failures which could cause intermittent reboots, lock-ups, bad data transfer, or in the worst-case physical damage to a transceiver.

Since the relevant information is contained in the differential voltage between CANH and CANL, the CAN-bus requires a high *"common-mode rejection-ratio"* (CMRR). CMRR of a differential-input indicates the capability of the input to reject input signals (noise) common to both input leads. Common-mode noise exists within the networks associated with CAN-applications due to noise from pulsing motor controllers, switch-mode power-supplies, or from fluorescent-lighting load. Different ECUs have different voltage-regulators with different common-mode rejection ratios. Because of this, there exist small but non-negligible differences in ECUs' nominal supply-voltage (V_{cc}) [37, 38]. Since there exists no perfect ground in the real world, we will have variations in the ground and supply-voltages [39].

The transceivers send bits by turning on/off two transistors, $Q1$ and $Q2$ in Figure 2.3, to generate a differential output voltage (V_{OD}) between CANH and CANL. To send a 0-bit, the transceivers turn on the transistors, which in such case can be considered as the resistors $R_{DSON,P/N}$. These resistors are designed to have the same values; however, due to manufacturing variations, the resistor values can have slight variations from each other [40]. Since the resistor's value is inversely-related to the gate-voltage (V_{cc}), which varies with each ECU, the resistor values will also have variations in between ECUs. These variations are more evident when transmitting a 0-bit because of the resistors being in use. As seen in Figure 2.3b, this is also referred to as a *dominant state*. The voltage measured during this bit is therefore referred to as a *dominant voltage*.

A dominant voltage depends on the V_{cc}, ground and the resistor-values in each ECU, and since these vary in between ECUs, the dominant voltages will also vary in CANH and CANL. Therefore, as mentioned, the ISO 11898-2 standard requires that a compatible transceiver must accommodate variations in the dominant voltages [41]. According to ISO 11898-4, CANH= +2.75 to +4.50, CANL= +0.50 to

+2.25 [42].

One of the most critical challenges in using low-level characteristics of ECUs for fingerprinting is that environmental conditions such as temperature and magnetic fields may affect such characteristics. However, since the CANH and CANL cables are physically close, they are equally exposed to the same noise sources. This ensures that the exposure to electromagnetic fields is nearly equal on each line, and thereby, cancelling the differential influence from magnetic field coupling by reversing the polarity in adjacent loops of twist in twisted-pair wiring. Even a change in temperature affects both cables equally due to the close physical distance. The transceivers adjust CANH and CANL in the opposite direction to still generate the intended differential-voltage [43].

In summary:
Different CAN transceivers' nominal supply-voltage, ground-voltage, and resistor-values can have variations, especially when transmitting a 0-bit. Thus, the CANH and CANL dominant voltages will vary in between ECUs. Transient changes in magnetic fields and temperature affect CANH and CANL equally, and the transceivers address this by deviating either cable in the opposite direction to generate the intended differential voltage.

These properties are exploited by Viden [17] to construct voltage profiles for ECUs, which we will discuss later. The recessive voltages when the nodes are sending 1-bits are ignored due to little voltage variation.

2.4 Voltage-based Attacker Identification - Viden

Cho and *Shin* introduced a concept, *Viden* [17], to accurately identify attackers by focusing on the output voltage of attack messages. They claim that their method fingerprints ECUs differently in comparison with other methods proposed by Murvay et al. in [16] and Patsakis et al. in [15], which also use voltage-measurements to fingerprint ECUs. Viden does not impose any restriction on a specific type of CAN-message or the speed of CAN-bus. At a sampling-rate of 50KSamples/s and through an online update of profiles, Viden provides adaptability to transient changes such as temperature and magnetic fields. Due to the relatively low false-positive rate, Viden is the prominent benchmark for this thesis. We detail its methodology and further describe the improvisations we have made in order to obtain the results.

Viden's system model comprises of both an IDS which detect an attack and a fingerprinting device which complements the IDS by identifying the source of the attack. The fingerprinting device is based on both timing and voltage schemes [17], [15], [16]. Viden's threat model takes into account attack-scenarios and assumes that no compromised devices are attached to a vehicle, meaning, it only considers devices that can be remotely compromised and not otherwise. The attack-model of our thesis covers and complements these attacks, and in fact, is even more complete than the attack model proposed in Viden. Viden does not consider conquest attacks but defines two impersonation techniques, namely, *arbitrary impersonation* and *targeted impersonation*. Arbitrary impersonation is misleading the fingerprinting device to think that some arbitrary ECU but the attacker itself has mounted the attack. Targeted impersonation corresponds to making the fingerprinting device believe that a targeted ECU has mounted the attack.

The authors of Viden consider three different types of adversaries based on their capabilities and the knowledge they possess about the vehicle:

1. **Naive Adversary**: Lacks knowledge about ECU fingerprinting but injects messages (with forged IDs) unwarily at arbitrary times.
2. **Timing-aware Adversary**: Knows how ECUs can be fingerprinted via timing-analysis, logs the CAN-traffic, and learns the timing-behaviour in order to imitate other ECUs. This type of adversary can perform both types of impersonations.
3. **Timing-voltage-aware Adversary**: Knows how ECUs can be fingerprinted via voltage and timing-measurements. (S)he can impersonate other ECUs and evade the fingerprinting device by changing own voltage-levels in order to fit in the output-voltage range of the target ECU.

However, Viden is still prone to attacks owing to a weakness detailed as follows. Fingerprinting ECUs based on time utilize the fact that most in-vehicle ECUs operate on local clocks with distinct skews. An attacker cannot change/modify the clock skew of an ECU since it is a physical property. Hence, Sagong et al. propose *Cloaking the Clock* [44], an attack which imitates the timing of the transmitted messages in order to match the clock skew of a targeted ECU and inflict into the network.

The authors implement the cloaking attack on the current state-of-the-art IDSs and a new IDS developed by themselves, which is based on the widely used Network Time Protocol (NTP). In order to measure and quantify the effectiveness of the cloaking attack, a new index called *Maximum Slackness Index* was defined. Lower values on this index reflect higher effectiveness in identifying the attack. The authors define two types of attackers, as detailed in Section 2.2, weak attackers (who can

inflict a suspension attack) and strong attackers (who can inflict both suspension and fabrication attacks). Their findings show that the NTP-based IDS is more effective in detecting masquerade attacks than other state-of-the-art IDSs. The approach presented in our thesis does not focus on modifying clock-values. Since we utilize CASAD as an IDS on top of which the fingerprinting-technique is built, our design as a whole is still efficient against such a cloaking attack.

2.5 CASAD: CAN-Aware Stealthy Attack Detection

Most of the attacks on IVNs either noticeably affect the system properties they operate in or change the overall behaviour of the CAN-traffic and are hardly stealthy. There exists a strong notion of considering a masquerade attack as a stealthy attack where the adversary injects messages from a compromised ECU while simultaneously muting the intended ECU. However, the Nowdehi et al. argue against this notion by defining a truly stealthy attack called the *"conquest attack"* as detailed in Section 2.2. The adversary reprograms the intended ECU while not affecting the IVN-characteristics, yet directly manipulating the payload of its messages. Hence, this type of attack is hard to detect and at present does not have any countermeasures.

Nowdehi et al. propose CASAD (CAN-Aware Stealthy-Attack Detection) [12], which learns the normal behaviour of the CAN-traffic in a vehicle based on the historical data related to the IVN-dynamics. CASAD is influenced by a recently proposed specification-agnostic method, PASAD [45], which detects attacks on industrial control systems. By directly processing CAN-traffic at the payload-level, and by using an exploratory time-series analysis-technique, CASAD captures the deterministic behaviour of the IVN-dynamics. The approach is the first to identify a mathematical representation of the normal behaviour of IVNs and continuously monitoring for any changes that indicate an attack. Since this approach does not require any knowledge about the underlying mechanism generating the CAN-traffic, it overcomes some of the significant challenges such as lack of backward compatibility, insufficient implementation details, among others related to practical deployment.

The authors of CASAD show that their approach can detect different types of attacks, including the conquest attack. The conquest attack can prove to be effective against the features offered by Advanced Driver Assistance Systems (ADAS) such as collision warning, lane departure warnings, among others which depend on the integrity of sensor values. This shows the far-reaching consequences a conquest attack

can have on the vehicle, and hence the passengers/pedestrians in the proximity of the vehicle. In our thesis, we use the original implementation of PASAD to replicate the working of CASAD. Following the methodology stated in the CASAD paper, we make necessary modifications to the original PASAD algorithm to ensure that it can analyse and understand CAN-traffic and detect different types of attacks.

3
Related Work

The parameters considered to evaluate the behaviour of the IVNs and detect attacks can broadly vary in between IDSs, as described in Section 2.3.2. The main focus of our thesis is the two prominent works in vehicle-cybersecurity, namely Viden [17] and CASAD [12]. Although we do independent experiments on both of these systems, we do not change any parameters for each of them. Instead, we propose to run them together as a unified system for both attack detection and source identification. Viden emphasizes the low-level characteristics of ECUs and hence, in this section, we narrow our approach to IDSs similar to Viden, which evaluate the state of the system using different low-level characteristics such as voltage, temperature, clock-skew and clock-offset. We also discuss the weaknesses of a few other solutions and how such weaknesses can be possibly addressed. CASAD's detection engine is capable of detecting attacks which other IDSs fail to detect. This being the primary motivation for our choice of CASAD as the IDS, we detail the way CASAD works towards the end of this section.

3.1 Clock-based IDS

In recent years, many studies and methods have been introduced by researchers to fingerprint ECUs based on their low-level characteristics. In [46], *Cho* and *Shin* propose a method to fingerprint ECUs based on their clock-offset and clock-skew values extracted from the message arrival times. This method, so-called *Clock based IDS* (CIDS), measures and exploits the intervals of periodic in-vehicle messages for fingerprinting ECUs. These fingerprints are then used to construct a model of ECUs' clock behaviours using mathematical techniques such as the *Recursive Least Square* (RLS) algorithm. Building on the thus constructed model, the method uses a *Cumulative Sum* (CUSUM) analysis (which derives cumulative sums of the deviations from a specific value to identify sudden changes) to assess the error. This analysis helps to detect intrusions by observing any abnormal shifts in periodic intervals of message arrival times. The model is evaluated on a CAN-bus prototype, and experimental results indicate that CIDS is capable of detecting various attacks at a low false-positive rate. CIDS fingerprinting is also useful in identifying the root

of the attack, i.e., determining which ECU mounted the attack. In comparison with this technique, our thesis is built on another low-level characteristic, namely voltage and differs significantly in identifying the source for different types of attack.

3.2 Temperature-based ECU-Fingerprinting

Tian et al. propose "*Temperature-Varied Fingerprinting*" (TVF) [47] for fingerprinting ECUs and making source identification efficient even in the presence of temperature variations. The authors observed that the clock-offset value of a specific ECU varies with environmental temperature of that ECU, however, remains linear in normal operating temperatures. By exploiting this fact, the authors develop temperature-varied clock-offset as a fingerprint for ECUs. A linear regression model is constructed for each ECU based on parameters, namely, temperature ranges (which can be obtained from vehicle manufacturers), periodic message-variation at different temperatures, and the average clock-offset value. This model corresponds to the fingerprint of each ECU at a specific temperature. Many such fingerprints at different temperatures are built and used as a baseline. TVF can detect an attack by observing and identifying abnormal clock-offset at certain temperatures. The temperature-varied fingerprints of each ECU then facilitate the source detection. The experimental results published by the authors show that TVF is able to detect masquerade attacks (detailed in Section 2.2) at various temperatures and also identify the source of the attack. In our thesis, we do not take into consideration any temperature variations.

3.3 Physical Fingerprinting of ECUs based on Machine Learning Algorithms

ECUs can be fingerprinted by exploiting characteristics such as material imperfections and deficiencies in the design. In [48], the author, Omid Avatefipour, proposes a fingerprinting method by constructing unique artefacts based on physical characteristics at both the channel-level (for example cables connected to the CAN-bus) and the digital device-level (for example ECU). In order to leverage an identification method, the framework uses the physical channel-dependent attributes to link a message to the transmitting ECU. If an adversary sends a malicious message from an external ECU either by impersonation or changing the cables that are connected to the CAN-bus, this method is able to clearly distinguish the message by comparing it with the existing patters for all the known ECUs.

In order to construct a unique profile for each ECU, the proposed method uses a

feature vector that makes use of eleven time and frequency-domain statistical signals collected across sixteen different channels from four identical ECUs transmitting the same message. This data is used to train a multi-layer neural network based classifier by performing a convolution of the impulse-response of the physical channel (connecting the ECU and the CAN-bus) and a message from that particular ECU. The resulting signal from the convolution itself acts as a unique artefact to which a specific ECU can be attributed. However, in our thesis, we do not use characteristics which are based on the physical attributes of ECUs/CAN-bus; instead, we fingerprint ECUs based on their voltage characteristics.

3.4 CASAD's Detection Engine

As explained in Section 2.5, CASAD is able to detect different types of attacks detailed in Section 2.2. As mentioned earlier, CASAD is inspired by PASAD, and hence we begin with first presenting a brief overview of PASAD. We then dive deeper into the working of CASAD and detail the modifications made to PASAD to accommodate to the requirements of IVNs.

3.4.1 PASAD: Process-Aware Stealthy Attack Detection

Process-Aware Stealthy Attack Detection (PASAD) [45], is an attack detection mechanism recently proposed to continuously monitor physical sensor-measurements from industrial control-systems and thereby detect any deviations from the normal historical behaviour of the system. A signal information representing the deterministic behaviour of the system is extracted from a time-series of these sensor-measurements. This method is a time-series analysis-technique known as *singular spectrum analysis*. The method consists of two phases, namely, an offline learning-phase to determine the normal behaviour of the system, and an online detection-phase to determine variations from the normal behaviour. Both of these phases operate on an input of time-series, τ measurements from sensors values, detailed as follows:

1. **Learning Phase:** A vector-space, referred to as a trajectory-space is created by embedding a subseries of the monitored signal which is used for training. Next, the training-vectors are projected to a signal-subspace which is identified using a mathematical procedure known as *isometry trick* shown by Aoudi et al. [45]. The projected vectors form a cluster in the signal subspace and thereby define the normal behavior, depending on the regularity in the underlying system behaviour. Formally, an initial subseries of τ of length N is unfolded into a *trajectory-matrix* $\mathbf{B}$ by forming K L-lagged vectors $\mathbf{b}_i$, where

L is the lag-parameter and $K = N$ - $L + 1$. Next the *singular value decomposition* of $\mathbf{B}$ is performed to obtain an orthonormal set of L eigenvectors of the covariance-matrix $\mathbf{BB}^T$. A matrix $\mathbf{U}$ is then formed whose columns are the $r < L$ leading eigenvectors, where r is the so-called *statistical dimension*. The training-vectors $\boldsymbol{b}_i$, $1 \leq i \leq K$, are then projected onto the signal-subspace spanned by the column-vectors of $\boldsymbol{U}$, with the centroid of the cluster they form being computed as $\tilde{\mathbf{c}} = \mathbf{U}^T\mathbf{c}$ where $\mathbf{c}$ is the sample mean of the training-vectors.

2. **Detection Phase:** A test-vector is composed at every iteration by incorporating the most recent sensor-value. By measuring the distance between the most recent test-vector and the centroid of the determined cluster, a *departure score* is iteratively computed for the most recent lagged vector $\mathbf{b}_j$, $j > K$. This is done by obtaining the squared Euclidean distance between the centroid $\tilde{c}$ and the most recent test vector $\mathbf{b}_j$ as

$$D_j = \|\tilde{c} \text{ - } \mathbf{U}^T\mathbf{b}_j\|^2 \tag{3.1}$$

An alarm is raised if the score crosses a prescribed threshold.

3.4.2 CAN-Aware Detection

PASAD, as it is, is not applicable to vehicular networks and hence two major changes have been made to the algorithm resulting in the development of CASAD.

1. **A mechanism to model CAN-traffic:** The payloads of consecutive messages from the CAN-traffic are extracted as bytes of data and are modelled as a time-series $\tau = \mathrm{b}_1, \mathrm{b}_2, \mathrm{b}_3, ..., \mathrm{b}_N, ...$ Despite the fact that the CAN data-field has a variable length and may contain up to 8 bytes of signal-data, yet they can be expressed a multiple of one byte. Hence the vectors are always processed byte-wise and are used to construct the trajectory-matrix and thus perform training. For example, suppose on a CAN-bus, the traffic is modeled as $\tau = \mathrm{b}_i, \mathrm{b}_{i+1}, \mathrm{b}_{i+2}, \mathrm{b}_{i+3}, \mathrm{b}_{i+4}$. If the payload of a message m_j consists of three bytes of data $\mathrm{b}_i, \mathrm{b}_{i+1}, \mathrm{b}_{i+2}$, then the following message m_{j+1} would have only two bytes of data $\mathrm{b}_{i+3}, \mathrm{b}_{i+4}$, as its complete payload.
2. **Compute departure scores using a modified procedure:** Nowdehi et al. confirmed that the use of Equation 3.1 yielded poorly scaled departure scores during the detection-phase. They attributed this result to the fact that monitored values belong to a bounded range unlike the continuous real-valued sensor-measurements as in PASAD. Hence, they proposed a modification in measuring the departure score for every test-vector, by computing the squared weighted Euclidean distance from the centroid. The ratio of each eigenvalue to the total sum of eigenvalues associated with the r eigenvectors determined

in the learning-phase account for the weights. Formally, the departure score in Equation 3.1 is modified as follows:

$$\tilde{D}_j = \|\mathbf{W}(\tilde{c} \text{ - } \mathbf{U}^T\mathbf{b}_j)\|^2 \tag{3.2}$$

W is the corresponding weight and an r-dimensional diagonal matrix whose i^{th} diagonal entry is defined as $e_i/\sum_i^r e_i$, where e_i is the eigenvalue corresponding to the i^{th} eigenvector $\mathbf{u}_i$ for $1 \leq i \leq r$. A prespecified threshold θ is also defined and whenever $\tilde{D}_j > \theta$, an alarm is triggered.

3.4.3 Limitations of CASAD

The authors of CASAD mention certain limitations of the proposed detection algorithm and also mention scenarios where CASAD might need complementary support in order to detect intrusions. As noted by Nowdehi et al. CASAD performs well when handling periodic CAN-messages (most CAN-messages have this behaviour) as CASAD performs spectral analysis of time-series. In case of emergency, event-driven or rare messages (e.g. Airbag deployment), e.g., CASAD might require additional support to blacklist/whitelist messages for more precise handling. Additionally, a conquest attack that makes slow normal-looking changes at the payload-level may not be promptly detected by CASAD. In fact, to our best knowledge, there is currently no proposed solution in the literature that is capable of detecting such stealthy attacks that are slowly performed. The linear increase in fuel-consumption, engine-speed, e.g. are examples where an adversary can manage to put a car in an unsafe state over a certain period of time.

4

Empirical Study of Viden

In this chapter, we describe in detail the working path on how we implemented each phase of Viden [17]. We received direct help from the authors of CASAD and hence had little/no problems with its implementation and hence do not describe it in detail. Instead, we refer to the original paper of CASAD [12] for more details on the implementation.

As explained in Chapter 2.4, the authors of Viden, Cho and Shin, claim to accurately identify attackers by uniquely profiling each ECU through their voltage behaviour when transmitting CAN-messages.

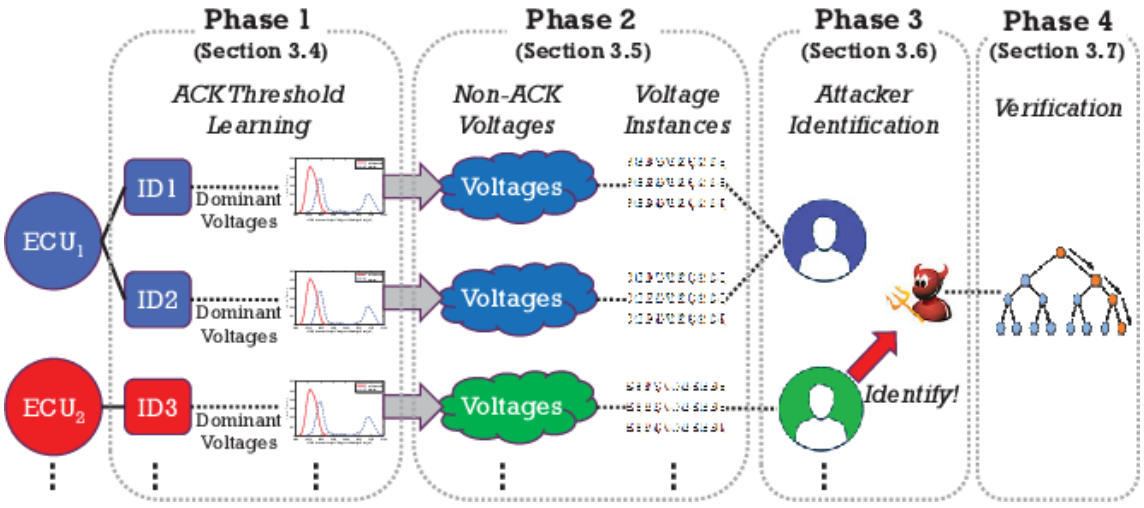


Figure 4.1: Four Phases of Operation in Viden [17].

As seen in Figure 4.1, Viden fingerprints ECUs and achieves attacker identification in four phases, detailed as follows:

- **Phase 1**: The CANH and CANL voltages are measured, and the recently acquired values are mapped with the ID of the message received from the ECUs

receive buffer. Next, for that message, Viden learns its ACK-threshold, which is essential in determining whether or not the measured voltages originated from the actual message transmitter. This step is run at initialization and repeated when an update is necessary.

- **Phase 2**: The learned ACK-threshold is exploited to select the voltages that are outputted solely by the message transmitter. These voltage-measurements are used to derive a voltage instance, which is a set of features that reflect the transmitter ECUs voltage output behaviour. Phase 2, 3 and 4 are run iteratively.
- **Phase 3**: The newly derived voltage instance is used to update the voltage profile of the message-transmitter. In case an attack is detected by an IDS, Viden identifies the attacker ECU by constructing a message profile for the attack-messages and mapping it to the existing profiles.
- **Phase 4**: This is a verification phase using multi-class classification on the obtained results from Phase 3.

To understand how profiling using low-level characteristics of ECUs works, we began with replicating Viden's four phases on an Arduino Uno board, described in Subsection 5.1.1. Implementing Viden is not a simple task, and we had to find workarounds for several issues. In the following sections, we explain how the voltage characteristics explained in Section 2.3.3 can be exploited to fingerprint ECUs. We also explain how we replicated each phase of Viden, the challenges that we encountered and the workarounds we took. Each section begins with a flow-chart displaying the technical and design-challenges we encountered in a topological order when replicating each phase of Viden. We then explain each challenge with its solution.

4.1 Phase 1 - Learning the ACK-threshold

In Phase 1, we collect all dominant voltages from the transmitter for different message-IDs. The message-payload does not affect Viden and, we therefore filled all messages with 0s.

To transmit a message, an ECU transmits a 0-bit after a certain idle-period to tell other ECUs that this is the start of a new message. During this stage, the voltage on the CANH increases from 2.5 (recessive) and exceeds 2.75 (dominant), and the voltage on CANL decreases from 2.5 (recessive) to below 2.25 (dominant). Towards the end of the message-transmission, all nodes except the transmitter output a 0-bit (dominant voltage) on the CAN-bus to acknowledge the message. This occurs before the transmitter has finished sending the message, as seen in Figure 4.3.

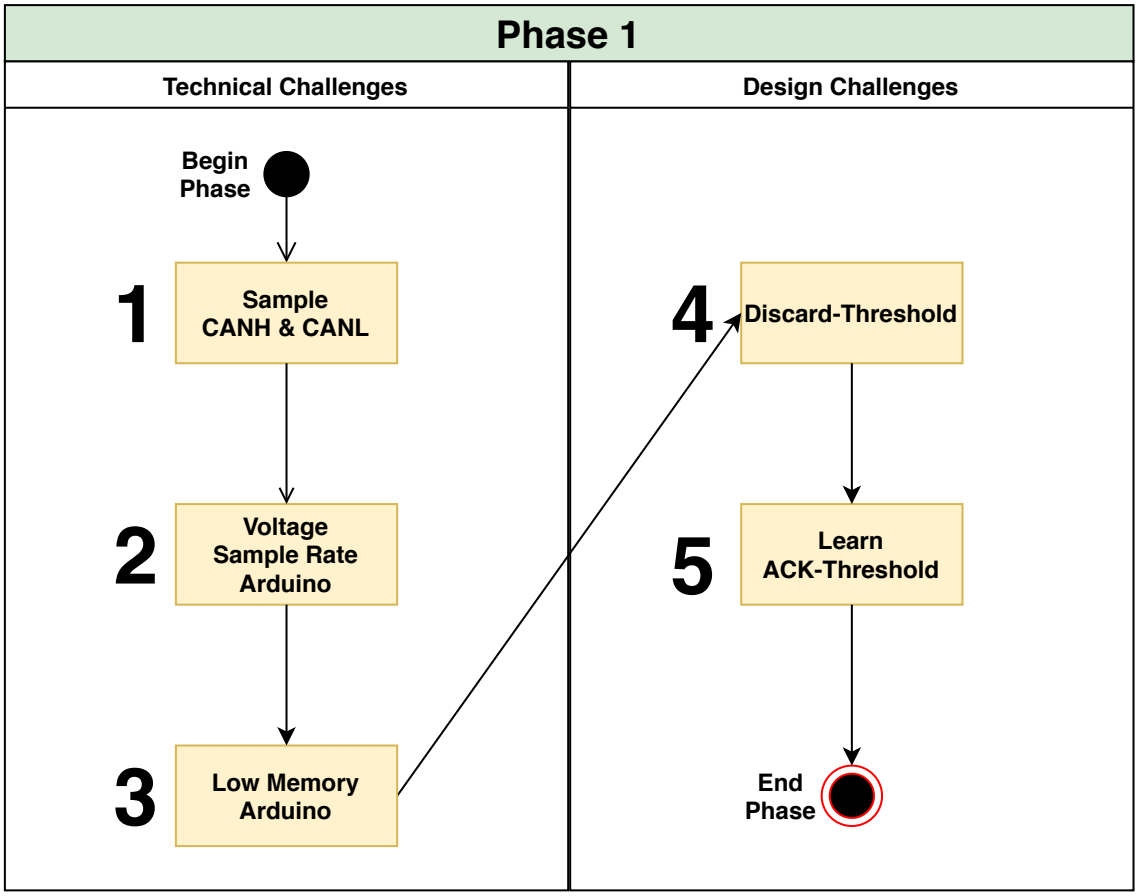


Figure 4.2: The technical and design-challenges we encountered when replicating Phase 1 of Viden.

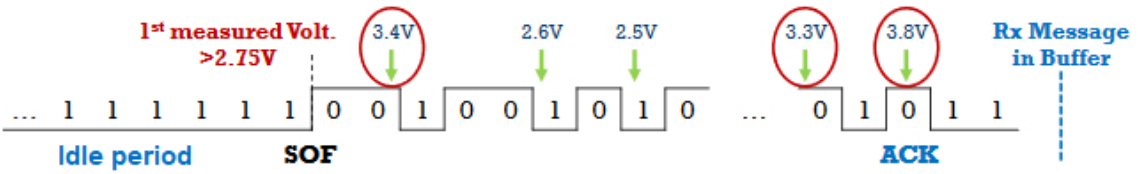


Figure 4.3: All ECUs except the transmitter send an acknowledgement to the transmitter before the message has finished transmitting. The marked voltages {3.4V, 3.3V, 3.8V} represent the collected dominant voltages [17].

Since the acknowledgement-message is sent before the transmitter finishes the transmission, the dominant voltages from the other ECUs get captured together with the other dominant voltages that were sent from the transmitter during the transmission. To accurately profile the transmitter, the voltage-measurement from the ACK-slot (*"ACK-voltage"*) needs to be discarded as it is not originating from the actual transmitter. This is done by exploiting two facts:

1. The probability of measuring an ACK-voltage is low because it consists of a single bit out of 128-bits of a CAN-message. Because of this, the most frequent measured voltages will most likely consist of voltages originating from the actual transmitter, *"non-ACK voltages"*.
2. Since the ECUs are connected in parallel, the voltage-measurement during this slot is higher than the other dominant voltages measured on CANH and lower than the other dominant voltages measured on CANL. However, if we only focused on the maximum voltages, we would sometimes get measurements from the non-ACK voltages, but with some probability, we would sometimes measure the ACK-voltages.

Cho and Shin, therefore, suggest that Viden should learn the ACK-threshold for the message IDs it wants to focus on. By learning the ACK-threshold for a specific message ID, Viden can discard voltages above that threshold and hopefully discard the voltage-measurement from the ACK-slot and therefore build a more accurate profile of the transmitter for a specific message ID.

4.1.1 Challenge 1: How can we measure the voltage-levels on CANH & CANL at the same time?

Since the CAN-bus uses two cables, CANH and CANL, we want the ADC-interrupts to read from two channels. Cho and Shin do not specify how this is done in their implementation. Reading from two channels in parallel is not an option on the Arduino Uno since it only has one ADC [49], we, therefore, chose to learn the ACK-threshold for one cable at a time. The disadvantage of this method is that we are not able to get the differential-voltage or compare CANH with CANL, which cooperate to generate the intended differential-voltage. Nevertheless, we have not noticed any adverse effects that this can cause in our implementation.

4.1.2 Challenge 2: Measuring voltage with Arduino - Sampling rate?

The default sampling rate using the ADC on the Arduino is 9600 samples/s, which is too slow to capture dominant voltages and to learn the ACK-threshold. We need to capture ACK-voltages to learn the ACK-threshold, and since the ACK-slot is

only 1-bit long, the probability of capturing it with a low sampling rate is low. Cho and Shin optimized their Arduinos, and they claim that their working solution had a sampling rate of 50KSamples/s. Similarly, we studied the Arduino-datasheet and optimized our Arduinos.

To transmit CAN-messages on the Arduinos, we used the CAN-library provided by the CAN-shield developers [50]. The voltage-levels on CANH and CANL were measured by using two analog-pins on the Arduino. The Arduino ATmega microcontroller features a single 10-bit "successive approximation ADC". *"Successive approximation ADC means that the ADC takes 13 clock cycles to complete the conversion (and 25 clock cycles for the first conversion)"* [51]. To avoid the first slow conversion, we triggered the ADC once at the bootup.

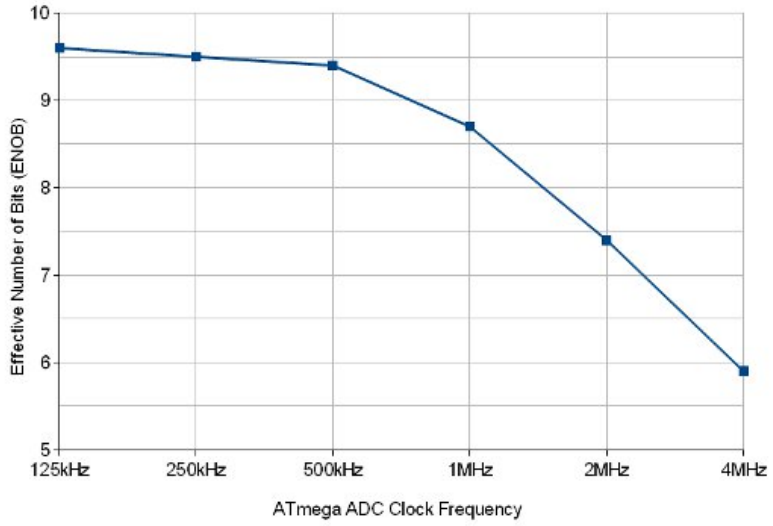


Figure 4.4: The ADC clock frequency can be increased by reducing the ADC-resolution [51].

Due to the ADC-clock being slower than other components of the microcontroller, the ADC's clock-signal is "computed" from the main clock. To get the maximum resolution of 10 bits, an input clock frequency between 50 kHz and 200 kHz is required. As seen in Figure 4.4, to get a higher sampling rate, the input clock frequency to the ADC can be increased with the cost of lowering the resolution.

As seen in Figure 4.5, the sampling rate can be increased through an interface to the ADC-module, which is controlled by a group of control-registers. The ADC-clock can be calculated by dividing the main clock frequency (16 MHz) with the *prescale*

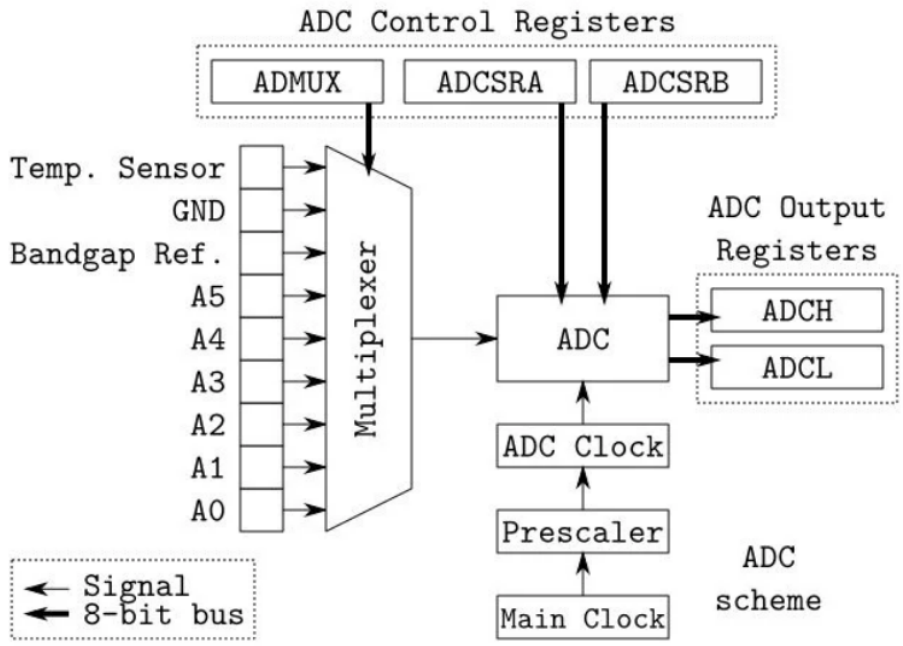


Figure 4.5: The ADC-scheme [51].

factor which by default is set to 128. Thus, we get 16MHz/128 = 125kHz on the ADC-clock. Since a conversion takes 13 ADC-clock cycles, the default sampling rate is about 9600 Hz (125Khz/13). As seen in Table 4.1, by setting the ADPS0-1-2 bits in the *"ADCSRA"* register, we can set the prescale factor to 16 and obtain an ADC clock frequency of 1.00 MHz, which provides a sampling rate of 76.9 kHz. The optimization results can be seen in Figure 4.6, where the time for 1000 ADC-conversions is measured.

The measured sampling rate is the best-case scenario. This is because, when different calculations are added to the ADC-measurements, the ADC sampling rate may drop and reach 50KSamples/s, which is the same sampling rate used by Viden. The ADC can be optimized even more to achieve this best-case scenario. One way to do so is to use the *"ADC Free-running mode"* instead of calling *"analogRead()"* each time to read the analog-pins. In this mode, the ADC throws an interrupt at the end of each conversion. By doing this, the ADC does not waste time waiting for the next sample, allowing to execute additional logic. It also improves the accuracy of the sampling by reducing jitter. To enable the free-running mode, we set the *ADATE-bit*, and to enable interrupts, we set the *ADIE-bit* in the *ADCSRA-register* [49].

Prescale	ADPS2	ADPS1	ADPS0	Clock frequency (MHz)	Sampling rate (KHz)
2	0	0	1	8.00	615
4	0	1	0	4.00	307
8	0	1	1	2.00	153
16	1	0	0	1.00	76.9
32	1	0	1	0.50	38.4
64	1	1	0	0.25	19.2
128	1	1	1	0.125	9.61

Table 4.1: The prescaling factor can be used to increase the sampling rate of the ADC [51].

```
Time per sample: 103.99 micro-seconds
Frequency: 9.62 KHz

Time per sample: 103.99 micro-seconds
Frequency: 9.62 KHz

Time per sample: 104.00 micro-seconds
Frequency: 9.62 KHz

Time per sample: 104.00 micro-seconds
Frequency: 9.62 KHz
```

(a) Default sampling rate with prescaler value 128 and with interrupts

```
Time per sample: 13.00 micro-seconds
Frequency: 76.90 KHz

Time per sample: 13.00 micro-seconds
Frequency: 76.92 KHz

Time per sample: 13.00 micro-seconds
Frequency: 76.95 KHz

Time per sample: 13.00 micro-seconds
Frequency: 76.92 KHz
```

(b) Optimized sampling rate using prescaler value 16 and with interrupts.

Figure 4.6: A comparison between the default sampling rate and our optimized sampling rate. We print to serial output after measuring the sample time for 1000 ADC conversions.

Now that we can sample voltages, we trigger the sampling of dominant voltages whenever the voltage exceeds 2.75V on CANH (first dominant voltage for CANH) or falls below 2.25V on CANL (first dominant voltage for CANL) after a certain idle-period. This represents *"start-of-frame"* of a new message [41]. For each message transmission, we store the dominant voltages in a buffer until the message is received in the receive buffer.

Since the SRAM on the Arduino has a size of 2KB, and we store float-values (4 bytes each), we can theoretically only store a maximum of 500 voltage-samples. However, in practice, this number is even less since we store other variables in the SRAM as well [52].

4.1.3 Challenge 3: Low memory on Arduino - Flush data to PC?

Due to the memory limitation on the Arduino's SRAM, we needed to flush the measured dominant voltages for each message through the serial port to a PC. Since the serial port requires interrupts to transmit messages, its interrupts collide with the ADC-interrupts, and since the serial communication is slower than the ADC conversions, it becomes a bottleneck in the process. Thus, we stop the ADC interrupts while using the serial port, which consequently leads to uncaptured dominant voltages on the CAN-bus. Increasing the number of CAN-messages for a particular message-ID suppresses this problem.

The ADC can be turned off by clearing the ADCSRA-register (setting it to zero). This is done after we collect all of the dominant voltages for a given message. We then use the serial port to transfer the voltage-measurements to a PC where a Python-script is listening for incoming messages on the serial port. The PC then stores the received measurements in a buffer (instead of a file) to avoid I/O operations that can be a bottleneck. When the Arduino finishes transmitting the data on the serial port, we restore the ADCSRA-register value from a backup we created at bootup. This loop continues until the PC receives M messages for each message-ID for N rounds.

Through experiments, we noticed that as long as we capture some ACK-voltages, we can calculate an accurate ACK-threshold. Initially, we used large sets such as 100K messages so that we could confidently determine whether we capture the ACK-voltages. However, as we started to understand each phase of Viden, we were able to improve our implementation and also reduce the number of messages required.

Through experiments, we noticed that each CAN-message would lead to 5-15 captured dominant voltages. To avoid flushing for each message which will cause I/O-bottleneck, we used $M = 30$ measurements and thereby increased the probability of capturing an ACK-voltage for each round. Through experiments, we also found the number of rounds required to learn the ACK-threshold accurately. While ten rounds could be large enough to capture ACK-voltages, we increased the number of rounds until we could confidently determine whether we captured the ACK-voltages. Each time we receive 30 measurements, we extract the maximum and the most frequently occurring voltage out of the 30 and store it in two buffers, $frequent\_samples$ and $max\_samples$. We do this for each message ID for $N = 100$ rounds, which is enough round to capture some ACK-voltages. The PC stops the reading after receiving $M{\cdot}N$ $(= 3000)$ dominant voltages from the serial port and flushes the samples into a text-file which could be used later. As explained in Section 4.1, the most frequent messages will most likely consist of non-ACK voltages, and the maximum voltages will contain some non-ACK voltages, and with some probability, some ACK-voltages as well.

As seen in Figure 4.7, a *"Kernel Density Plot"* (KDE) can be used to observe the probability density function of the measured voltages. We observed that the most frequent messages lay between approximately 3.20V and 3.23V, while the maximum voltages could sometimes vary. Cho and Shin, therefore, suggest that a *"discard-threshold"* should be used:

$$discard\_threshold = max(frequent\_samples) + \beta\sigma_{frequent\_samples} \tag{4.1}$$

where values lower than the threshold are discarded when extracting the highest voltage-samples. Here, $\sigma_{frequent\_samples}$ is the standard deviation of the set containing the most frequent samples, and β is a design parameter that determines how aggressive the ACK-voltages should be discarded.

4.1.4 Challenge 4: How strict should the ACK-voltages be discarded?

The value of β in Equation 4.1, is not known and not mentioned by Cho and Shin. In order to find an optimal value for β, we had to experiment with different values. We noticed through experiments that as long as the discard-threshold is higher than the most frequent values, it works. Thus, we used values 1-4 for β.

The values above the discard threshold define a new set, $max\_samples\_discarded$, which contains mostly ACK-voltages and with some probability a few high non-ACK voltages. By using this set, Cho and Shin determine:

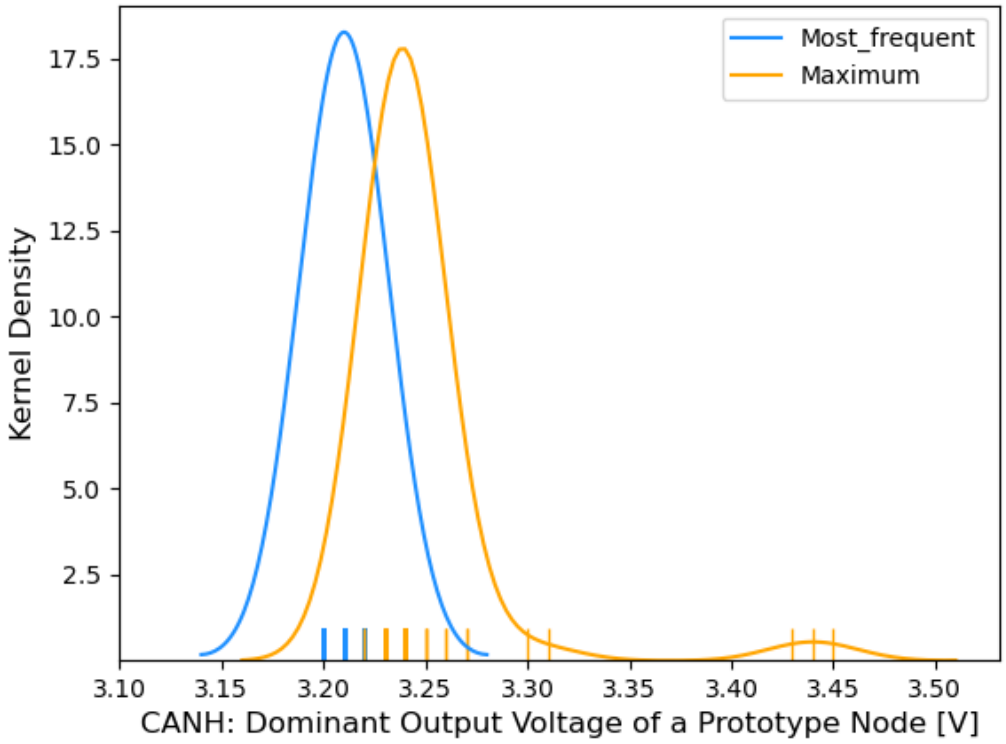


Figure 4.7: Dominant voltages from our CAN-bus prototype while sending message-ID *7FA* with data filled with 0s on a sample set of 3000 dominant voltage-measurements from CANH.

$$\Gamma_1 = median(max\_samples\_discarded) - 3MAD(max\_samples\_discarded)$$
$$\Gamma_2 = mean(max\_samples\_discarded) - 3\sigma_{max\_samples\_discarded}$$

where $MAD(x)$ denotes the median absolute deviation of x. The lower $3MAD$ and the 3σ-limit is used to avoid other side-lobes than the max-data side-lobe that can exist for some ECUs. The CANH ACK-threshold, Γ^H_{ACK}, is then derived to be $max(\Gamma_1, \Gamma_2)$. According to Cho and Shin, the maximum is taken to be conservative when discarding any non-ACK voltages.

When deriving the CANL ACK-threshold, Γ^L_{ACK}, a minimum set is used (instead of the maximum), and the upper limits are used (instead of the lower):

$$\Gamma_1 = median(min\_samples\_discarded) + 3MAD(min\_samples\_discarded)$$
$$\Gamma_2 = mean(min\_samples\_discarded) + 3\sigma_{min\_samples\_discarded}$$

Similarly, the CANL ACK-threshold is then derived to be $min(\Gamma_1, \Gamma_2)$. The final result for CANH and CANL can be seen in Figure 4.8 and Figure 4.9, where $\Gamma^H_{ACK} = 3.34$ and $\Gamma^L_{ACK} = 1.66$.

4.1.5 Challenge 5: Number of samples to learn the ACK-threshold?

Cho and Shin do not mention how many samples they collect to learn the ACK-threshold for a certain message-ID from a particular transmitter. Learning the ACK-threshold is crucial because even a single ACK-voltage can significantly distort the understanding of transmitter ECU's voltage-behaviour. In order to find an optimal value for the number of samples needed to learn the ACK-threshold, we had to experiment with different values. We noticed that as long as we capture some ACK-voltages, Viden can calculate an accurate threshold. While sometimes the ACK-threshold can stay constant from ten rounds to over 500 rounds, it can also suddenly update to a new value if the voltage distribution changes. We mostly ran for 100 rounds on our calculations. If we noticed that we had a sample set without ACK-voltages for a certain message-ID, we increased the number of rounds when learning the ACK-threshold for that message-ID.

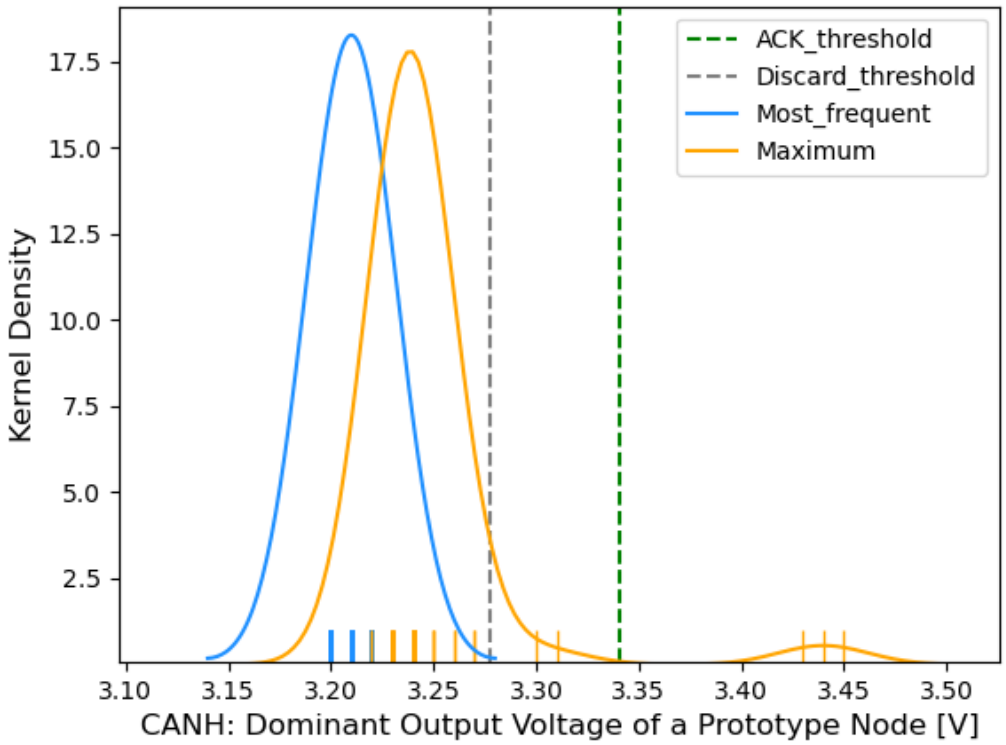


Figure 4.8: ACK-Threshold in our CAN-bus prototype for message-ID *7FA* and data filled with 0s on a sample set of 3000 voltage samples from CANH.

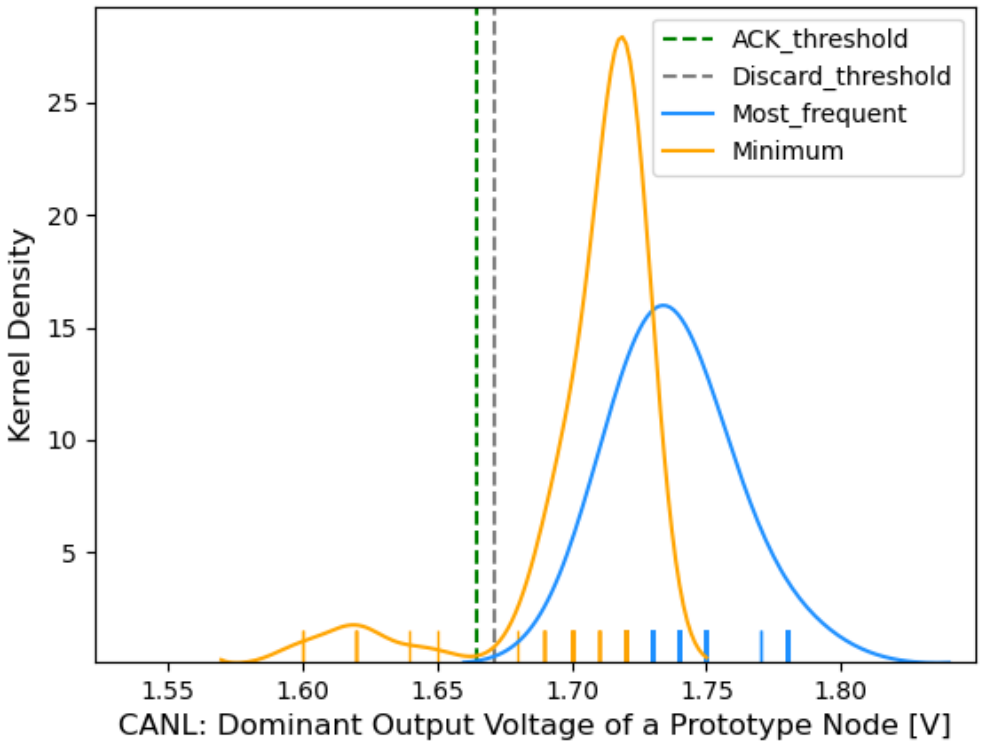


Figure 4.9: ACK-Threshold in our CAN-bus prototype for message ID 7FA and data filled with 0s on a sample set of 3000 voltage samples from CANL.

4.2 Phase 2 - Deriving a Voltage Instance

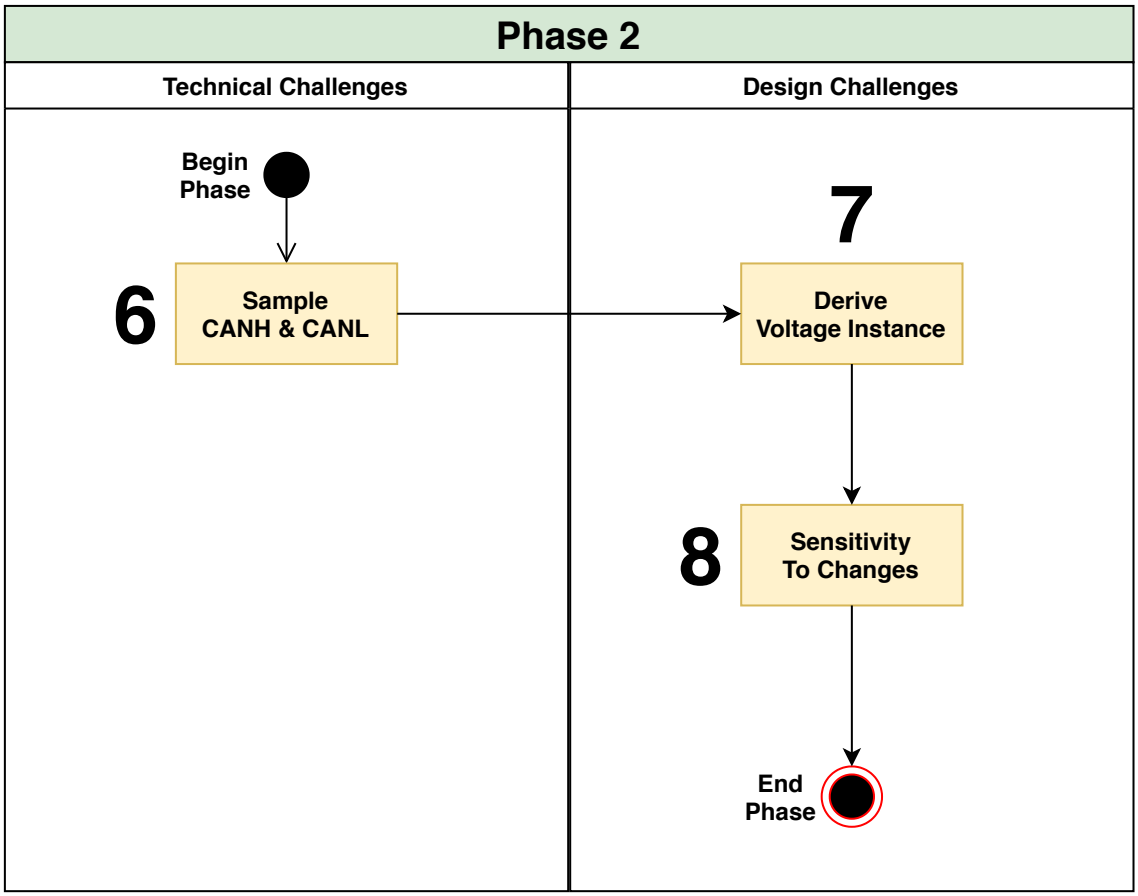


Figure 4.10: The technical and design-challenges we encountered when replicating Phase 2 of Viden.

Once we learn the ACK-thresholds for a given message-ID, we continue to collect dominant voltages on the Arduino, but this time we read from CANH and CANL iteratively. We flush the collected dominant voltages from both cables and use the learned ACK-thresholds for CANH and CANL to solely focus on the dominant voltages from the actual transmitter by discarding the ACK-voltages.

- CANH: ACK-threshold$_{CANH} > voltage > 2.75$
- CANL: ACK-threshold$_{CANL} < voltage < 2.25$

4.2.1 Challenge 6: How can we measure the voltage-levels on CANH & CANL at the same time?

As mentioned earlier, the ADC can not read from two channels in parallel. It is, however, required in this phase to construct and continuously update voltage profiles. We solve this by continuously flipping ADC-channels in between interrupts by modifying the *ADMUX-register* and hence reading from both the CANH and CANL cables iteratively. That said, it is unclear to us how Cho and Shin solved this challenge. The disadvantage of our method is that we lose dominant samples on the other wire that may be useful when profiling ECUs. We have not noticed any adverse effects that this can cause in our implementation.

Cho and Shin define:

$$K = \#\textit{New measurements from CANH and CANL} \tag{4.2}$$

After collecting K new measurements from CANH and CANL, a new voltage instance can be derived, which is defined as a set of six tracking points, as described below:

Most frequent values:
F1: Most frequent values of CANH
F2: Most frequent values of CANL

Dispersions: (4.3)
F3: 75th percentile of CANH
F4: 90th percentile of CANH
F5: 25th percentile of CANL
F6: 10th percentile of CANL

The voltage distributions of the voltage-outlets were different for different ECUs and could be used to uniquely profile the transmitter.

4.2.2 Challenge 7: How many samples should be collected to derive a voltage instance?

Cho and Shin do not mention the value of K (Equation 4.2) in their implementation. In order to find an optimal value for K, we had to experiment with different values. We noticed that around ten measurements could be used to generate an accurate voltage instance. However, increasing this number resulted in more linear

and stable profiles. Also, having too large values could cause the tracking points to be not adjusted often enough in the absence of transient changes resulting in non-linear profiles. We settled on values in between 30-40 for K.

By tracking the transmitter's voltage distribution, we understand the transmitter ECU's momentary voltage output behaviour. The rationale behind tracking of the suggested percentiles of CANH and CANL is 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 the same applies to the high percentiles of CANL measurements.

In Phase 2, Viden continuously samples dominant voltages from both cables. The collected samples are only stored in the PC's RAM and overwritten when the tracking points are adjusted. The tracked dispersions are updated whenever we acquire K dominant voltages from each CANH and CANL. Using the past kR measurements ($R = 10$ in Viden), we roughly estimate what percentile the current tracking point, Λ, represents. If the voltage instance varies in between rounds, an adjustment is made to correct and move each tracking point by using:

$$\Lambda \leftarrow \Lambda + \alpha(P^* - \frac{\#V < \Lambda}{\#V})^3 \tag{4.4}$$

where α is a design-parameter determining the sensitivity to changes, P^* is the percentile in decimals, and the fraction represents the percentage of the number of samples that falls below that percentile. The value of each tracking point, $\mathbf{F_x}$, will be close to constant in the best-case scenario, and little to no adjustment will be needed. However, if the ECU's voltage behaviour suddenly makes a change, it will be noticed in the next kR measurements' dispersion values, and the tracking point will be adjusted accordingly to the new position.

4.2.3 Challenge 8: Optimal value for α (sensitivity to changes)?

Cho and Shin do not mention the value of α, used in Equation 4.4, in their implementation. In order to find an optimal value for α, we had to experiment with different values. We noticed that when the ACK-threshold is accurate, the value of α is not important. We mostly experimented with values between $5 \cdot 10^{-5}$ and 5. Using too large values for α would cause the profile to over-adjust and thereby not stabilize, which most of the time, resulted in an exponential graph. We settled on $\alpha = 0.5$, which we noticed was sufficient enough when the tracking points needed adjustments.

The tracking points adjust faster if they are farther away from their desired positions. As a result, the four tracking points, {**F3**, **F4**, **F5**, **F6**}, as specified in Equation 4.3, continuously move if the transmitter's voltage distribution (i.e., output behaviour) shows changes, thus adapting to any changes on the CAN-bus. Cho and Shin explain that the percentiles could have been derived from the kR measurements directly. This, however, is too sensitive to transient changes, especially if kR is small. Therefore, in order to make Viden work under various circumstances, Cho and Shin suggest to track them instead as time progresses.

4.3 Phase 3 - Attacker Identification

Now that we have the tracking points, **F1-F6**, as specified in Equation 4.3, which only represent the momentary behaviour of a transmitter ECU, Cho and Shin suggest that in order to log the transmitter's usual behaviour, we exploit every newly derived voltage instance to construct a voltage profile of the message-transmitter. Cho and Shin claim that, although the voltage instances were derived per message-ID, if the messages originate from the same transmitter ECU, the voltage instances will be near-equivalent, thus leading to the construction of the same voltage profile each time.

In order to construct the profiles, we first update the *"cumulative voltage deviations"* (CVDs) of the features, **F1-F6**. A CVD represents how much the transmitter's dominant voltages deviated cumulatively from their ideal values. Thus, for feature $\mathbf{F_x}$, the CVD at step n, $CVD_x[n]$, is updated as:

$$CVD_x[n] = CVD_x[n-1] + \Delta[n](1 - \frac{v_x[n]}{v_x^*})$$

where $\Delta[n]$ is the elapsed time since step $n-1$, $v_x[n]$ the value of the feature $\mathbf{F_x}$ at step n, and v_x^* the desired value of v_x. Ideally, the most frequently measured values, as well as any percentiles of CANH and CANL dominant voltages should be equal to 3.5 V and 1.5V respectively. Therefore:

$$\{\mathbf{F1}, \mathbf{F3}, \mathbf{F5}\} : v* = 3.5V$$
$$\{\mathbf{F2}, \mathbf{F4}, \mathbf{F6}\} : v* = 1.5V$$

As explained in Section 2.3.3, since ECUs have different V_{cc}, ground and resistor values, they output different CANH and CANL dominant voltages. Because of this, their momentary voltage instances would be different as well, and hence, the trends

in their CVD changes would also be different from each other. So, for every obtained CVD of features **F1-F6**, Viden derives:

$$\Psi[n] = \sum_{x=1}^{6} CVD_x[n]$$

The rationale behind the summing of all CVDs is to exploit transient changes due to temperature and magnetic fields. By summing the CVDs, Viden suppresses any transient deviations that may have occurred when constructing the voltage profiles, and only focuses on the constant factors.

Since the values are rather constant, the accumulated sum: $\sum_{k=1}^{n} \Psi[k]$, becomes linear over time, and is referred to as $\Psi_{accum}[n]$. The sum of CVDs from a transmitter ECU can be seen in Figure 4.11. Moreover, as the sum of CVDs is distinct for different ECUs, the trends in how the accumulated sum changes as time progresses also become different for different ECUs. 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, which varies with each transmitter, it is defined as the *"voltage profile"*. Figure 4.11, shows an accurate voltage profile.

To determine the voltage profile, Υ, Viden uses an adaptive signal processing technique, the *Recursive Least Squares* (RLS) algorithm, which is an online approach to learning the regression parameter. Due to space limitations, we omit the details of RLS and refer the readers to read the book *"Adaptive filter theory"* by Simon Haykin [53] for further details.

When an IDS detects an attack, Viden filters out the voltage outputs obtained only from the detected attack messages, and then builds a voltage profile from only those. This voltage profile is then compared with the other existing profiles, thus, identifying the attacker ECU.

4.4 Phase 4 - Verification

Due to the *birthday paradox* [54], multiple ECUs can have near-equivalent profiles (profile collisions). To solve this Cho and Chin added Phase 4 which complements Phase 3 by using classifiers (*200-tree Random Forest classifier* [55]) on the voltage

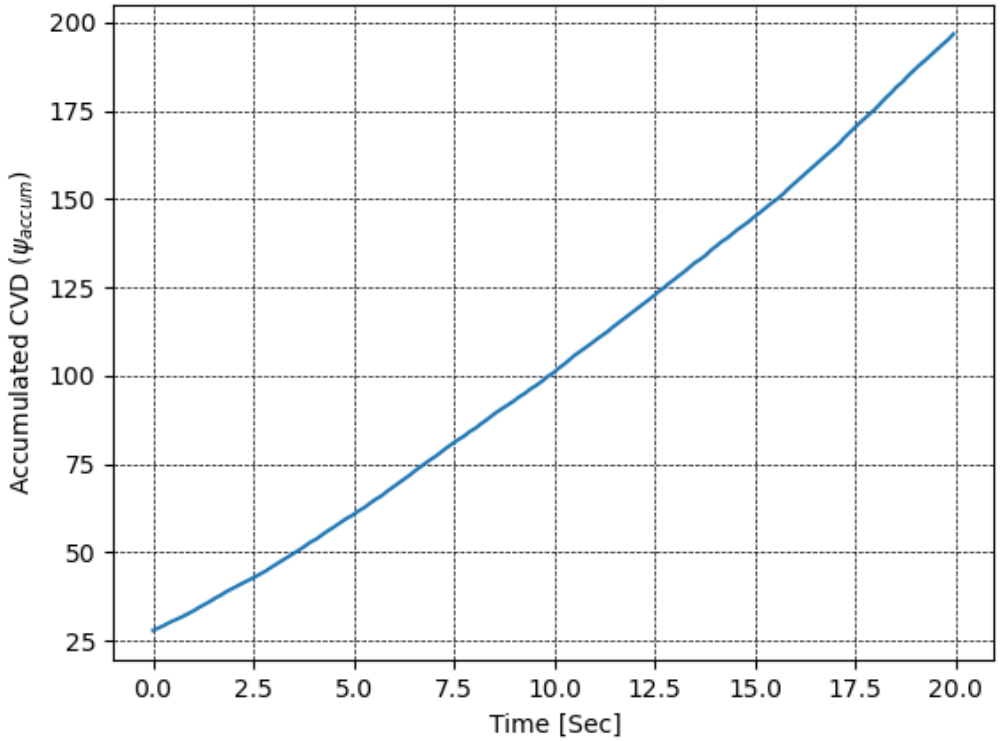


Figure 4.11: Accumulated sum plot displaying the linear voltage profile for an ECU.

instances, and thereby resolve the profile collisions. Phase 4 is intended to be used in real cars due to the increased number of ECUs on a single CAN-bus, which increases the probability for profile collisions. In our implementation we used the *Random Forest classifier* algorithm provided by *scikit-learn* Python-library [56].

As seen in Figure 4.12, a *training-set* is selected from a given set; in our case, 50% of all the voltage instances collected until an attack is detected. Then a decision tree is constructed for each sample, and a prediction result is output from each decision tree. A vote (averaging) is performed for each predicted result, and the most voted prediction result is output as the final prediction [57], which in our case is the transmitter the voltage instances belong to.

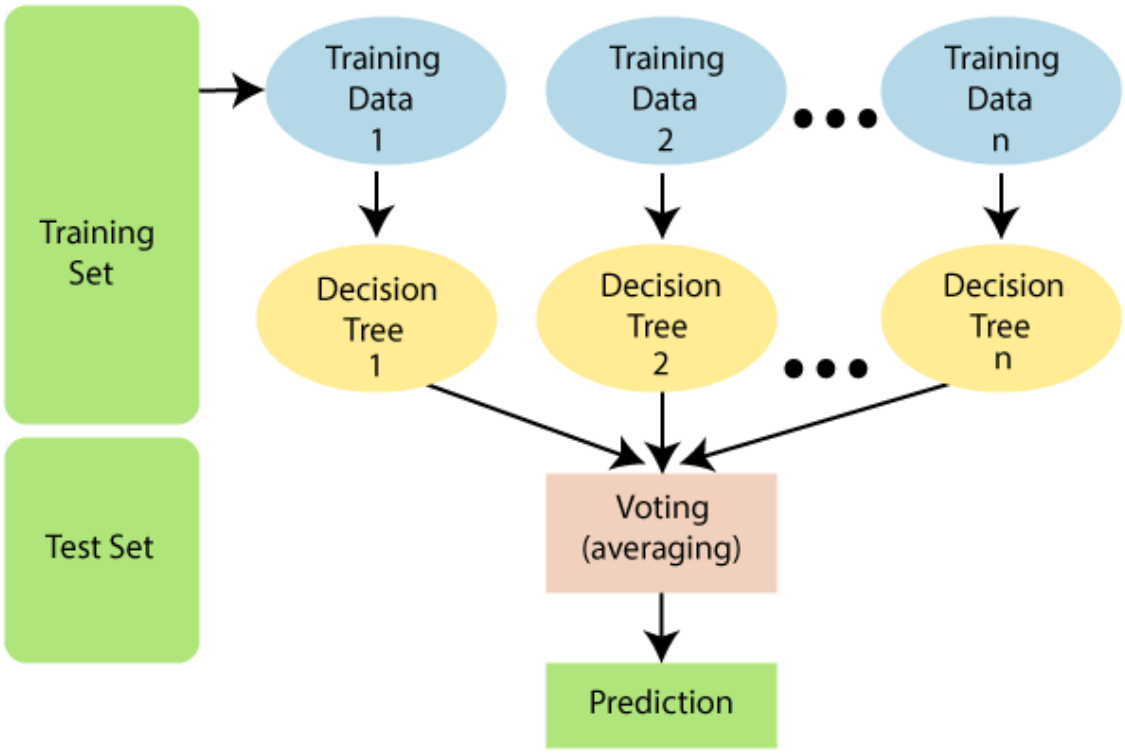


Figure 4.12: The Random Forest Algorithm [55].

5

Evaluation

In this chapter, we start with describing the evaluation environment for our experiments on Viden and CASAD. We explain in detail the experiments we performed on our Viden-implementation to evaluate its practicality, efficiency, and finally, identification accuracy when profiling ECUs. We also describe the experiments we performed on our CASAD-implementation to evaluate its attack detection capability on CAN-traffic. Finally, we conclude this chapter by proposing a unified system with CASAD as an IDS together with the Viden fingerprinting method.

As already discussed in Section 2.2, attacks on a CAN-bus can be classified into four major categories. The suspension attack is suspending an existing ECU from further action, while the fabrication attack corresponds to impersonating an existing ECU by sending in messages at a higher frequency than the target ECU. The third type is the masquerade attack which is a combination of both the suspension attack and the fabrication attack. The last attack is the conquest attack, which involves directly manipulating the payload of the message in order to make the receiver ECU work erroneously.

5.1 Evaluation Environments

We evaluated our implementations of Viden and CASAD in two separate evaluation environments, namely a CAN-prototype and on Boxcars. In order to evaluate the proof-of-concept of our Viden-implementation, we performed experiments on our CAN-prototype, which we explain in Subsection 5.1.1. The main reason for the initial implementation on the CAN-prototype was to evaluate experiments in a controlled environment. The Boxcars, which we explain in Subsection 5.1.2, was used to generate real CAN-traffic which was used to study CASAD's attack detection capabilities and also evaluate our Viden-prototype's source detection capabilities in a realistic environment.

5.1.1 CAN-bus Prototype

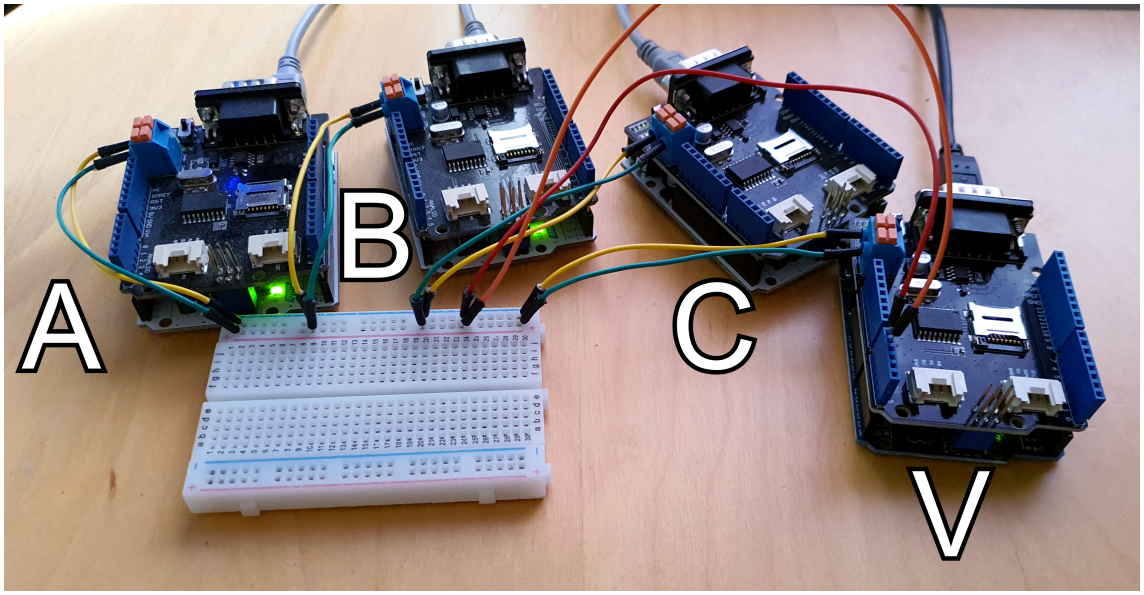


Figure 5.1: CAN-bus prototype.

We configured a CAN-bus prototype in which four nodes were connected, as seen in Figure 5.1. Each node consisted of an Arduino Uno board with a SeedStudio CAN shield V2.0 [58] connected in parallel with the CANH and CANL cables. All nodes were set up to operate at 500Kbps, which is typical for in-vehicle high-speed CAN-buses. Three prototype nodes A, B, and C were programmed to send/receive messages. The fourth node, V, was programmed to run Viden and construct profiles. This node read the voltage levels on the CANH and CANL cables through the analog-pins on the Arduino by using the microcontroller's *Analog-To-Digital Converter* (ADC).

5.1.2 Boxcar

A Boxcar at Volvo Cars was used to evaluate our implementations in an environment resembling an IVN in a vehicle. The Boxcar consisted of multiple ECUs connected in a network similar to Figure 1.1. The Boxcar comes equipped with additional network interfaces which allowed us to communicate with the ECUs that are otherwise unreachable on a released car. The ECUs were connected to vehicle components such as front-panel, lights, engine, among others, that was placed on a desk. The OBD-II port was directly accessible from the front-panel and was used to sniff the CAN-traffic and inject the attack-messages. A *Vehicle Connectivity Master* (VCM)

port was also directly accessible from the front-panel. The VCM-port was used to communicate with ECUs that were not accessible from the OBD-II port but were necessary for performing the attacks.

In order to communicate with the IVN, we used *CANoe*, a widely used software for ECU development and testing, developed by Vector Informatik GmbH. In addition to ECU development and testing, CANoe is a popular tool in the automotive industry for analysis, simulation, diagnostic and start-up of ECU networks. In our setup CANoe was running on a laptop along with a Vector VN1630A CAN-interface connected to the vehicle's OBD-II port. Using this setup, we were able to create two virtual ECUs, namely ε_1 and ε_2. These newly added virtual ECUs are practically considered to be a part of the CAN-bus capable of both actively influencing the vehicle's internal communication by injecting crafted messages and passively monitoring the traffic in real-time.

5.2 Viden

The experiments we perform to evaluate our Viden-implementation is described in Table 5.1. We demonstrate a proof-of-concept and evaluate the performance of our Viden-implementation with respect to detection accuracy, feasibility, practicality, and efficiency by performing experiments on our CAN-prototype, as explained in Section 5.1.1 (Experiment V.1). We then evaluate our Viden-implementation's performance against adversaries performing a fabrication attack (Experiment V.2). When our implementation passes these experiments, we evaluate whether our implementation works on Boxcars (Experiment V.3), as explained in Section 5.1.2, where we first profile the ECUs and then pinpoint the attacker ECU during a fabrication attack (Experiment V.4).

Section	Experiment	Environment
5.2.1	Experiment V.1: Voltage Profiles - Attack-Free	CAN-bus Prototype
5.2.2	Experiment V.2: Source Identification - Fabrication Attack	CAN-bus Prototype
5.2.3	Experiment V.3: Verification - Voltage Profiles	Boxcar
5.2.4	Experiment V.4: Verification - Source Identification	Boxcar

Table 5.1: Experiments with Viden.

Experiment V.1 is further described in Subsection 5.2.1, where we evaluate whether our Viden-implementation can uniquely fingerprint the Arduinos in an attack-free environment. Experiment V.2 is described in Subsection 5.2.2, where we evaluate whether our Viden-implementation can pinpoint the attacker in case of a fabrication attack. When our Viden-implementation passes these experiments, we verify the feasibility, practicality, accuracy and efficiency of our Viden-implementation on the Boxcar; first without attacks, in Experiment V.3, as described in Subsection 5.2.3, and then, in case of a fabrication attack, in Experiment V.4, as described in Subsection 5.2.4.

5.2.1 Experiment V.1: Voltage Profiles - Attack-Free

The CAN-bus prototype was programmed to inject messages aperiodically to prove that even in such cases, Viden is capable of fingerprinting the transmitters, as claimed by Cho and Shin. We refer to the Arduinos as ECUs during our evaluation since they simulate ECUs. The ECU running Viden was programmed to construct voltage profiles for the message-IDs *0x01*, *0x1D*, and *0xFE*, respectively sent by the ECUs *A*, *B*, and *C*.

As seen in Figure 5.2, our implementation constructed voltage profiles for each message-ID sent on the CAN-bus prototype to profile each transmitter. Although the transmitters were identical to each other, the linear trend in each transmitter's accumulated sum was different, resulting in unique profiles. This is due to variations in the transceivers' nominal supply voltage, ground voltage, and resistor values, which are especially noticeable when transmitting 0-bits, as explained in Section 2.3.3.

By using the Recursive Least Square algorithm, we were able to extract the linearity parameter (voltage profile) for each transmitter (*8.3*, *739.7*, and *2.6*, respectively). As shown in Figure 5.2, the voltage profiles for each message-ID were unique as they were sent by different transmitters, thus verifying the feasibility and accuracy of our Viden-implementation.

5.2.2 Experiment V.2: Source Identification - Fabrication Attack

To further evaluate our Viden implementation's source identification capabilities, we programmed *ECU C*, in Figure 5.3, to behave as a timing-aware adversary. In this setup, *ECU C* not only injected message *0xFE* as before, it but also injected attack messages with ID *0x01* aperiodically to spoof *ECU A*. The attack messages

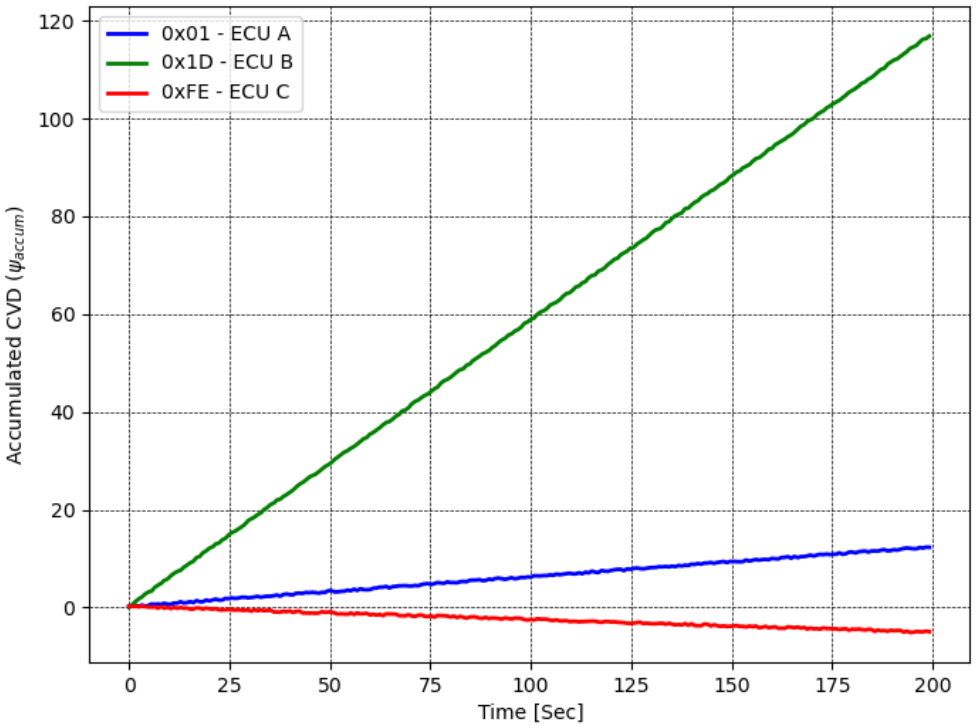


Figure 5.2: Three unique voltage profiles for three different message-IDs from three different ECUs.

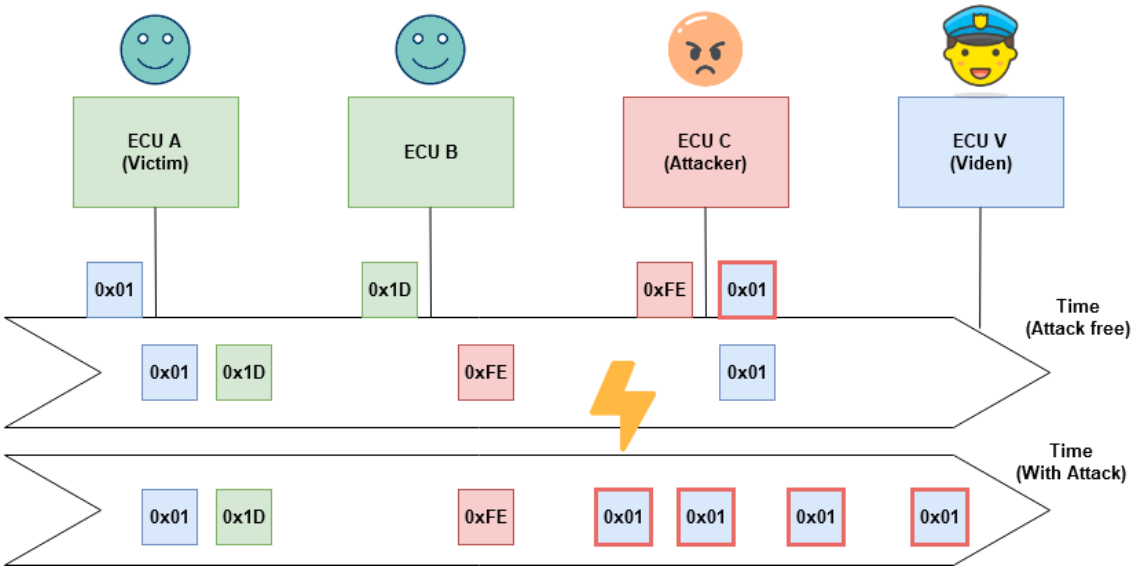


Figure 5.3: *ECU C* performing a fabrication attack on *ECU A*. The above time line represents the attack free CAN-bus and the bottom timeline represents CAN-bus under attack. When we reach the thunder, *ECU C* starts injecting messages with ID *0x01* aperiodically to impersonate *ECU A*.

were injected aperiodically to simulate an adversary performing arbitrary impersonation to avoid timing-based fingerprinting devices. *ECU A* and *ECU B* inject *0x01* and *0x1D* as before. Although we construct voltage profiles per message-ID, two different messages originating from the same transmitter will have similar profiles.

As shown in Figure 5.4, the voltage profile for *0x01* reflects the voltage output from both *ECU A* and *ECU C*. Since the message frequency from the attacker *ECU C* was much higher than the legitimate *0x01* messages from *ECU A*, the voltage profile for *0x01* changed and became an equivalent to the profile for message *0xFE*. Since this profile belongs to *ECU C*, our implementation determined that the messages *0xFE* and *0x01* are originating from the same transmitter, and thus identifying the attacker to be *ECU C*.

5.2.3 Experiment V.3: Verification - Voltage Profiles

To evaluate our Viden-prototype in a more realistic CAN-environment, we connected the Arduino-node running Viden to the OBD-II port on the Boxcar which gave Viden access to the CAN-bus. The Boxcar was set to run in *"driving-mode"* to keep a consistent state on the ECUs and avoid them from suspending after a certain idle-period. In this mode, we profiled mainly three ECUs that were active and injected

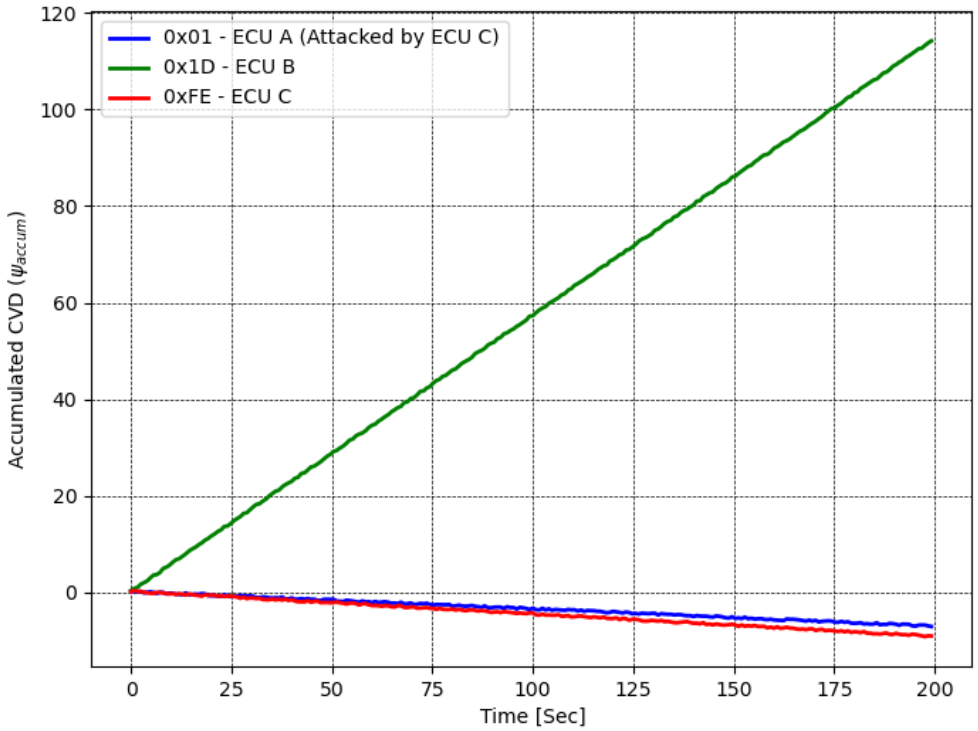


Figure 5.4: *ECU C* tried to impersonate *ECU A*. The message-ID *0x01* belonging to *ECU A* got a near-equivalent profile compared with *ECU C*. Since no voltage-profile collisions existed originally this proves that *ECU C* performed the attack.

multiple message-IDs into the bus. We also profiled the virtual ECU for one message ID. This profile is later used to pinpoint attack-messages back to the virtual ECU.

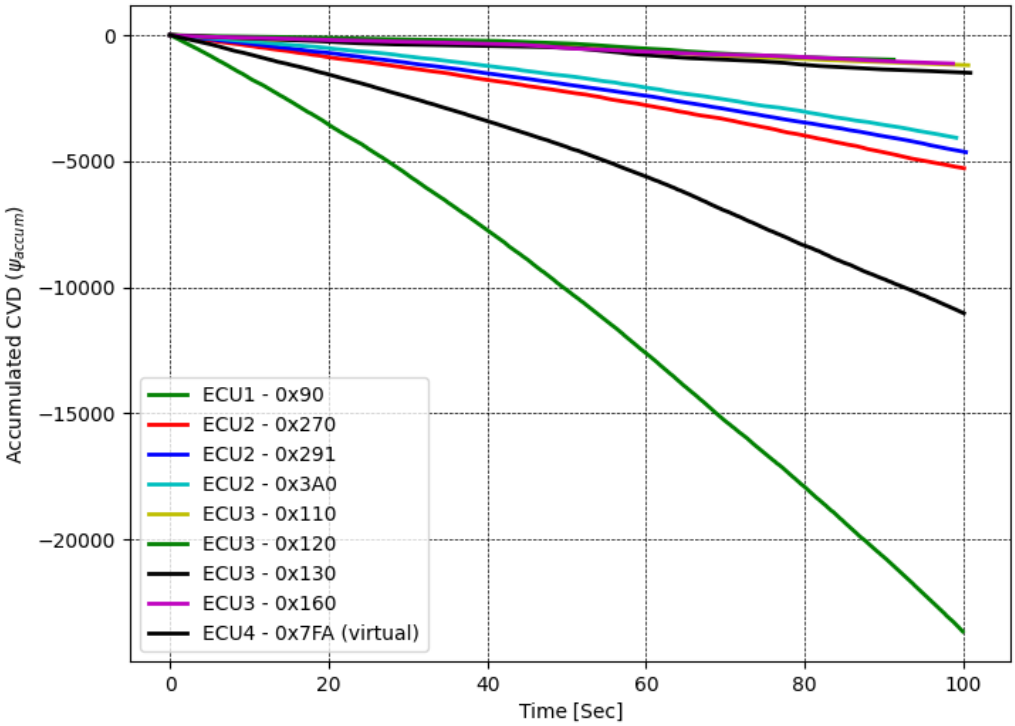


Figure 5.5: Nine voltage profiles from four ECUs sending nine different message-IDs on the Boxcar.

As seen in Figure 5.5, our Viden-prototype uniquely profiles four ECUs sending nine different message-IDs. We also see that message-IDs originating from the same ECU have identical voltage profiles.

5.2.4 Experiment V.4: Verification - Source Identification

To further evaluate our Viden-prototype's source identification capabilities, we programmed the virtual ECU to perform a fabrication attack similar to the scenario explained in Section 5.2.2. In this case, the virtual ECU, ε_1, tried to spoof *0x110* messages originating from *ECU1* in Figure 5.5.

As seen in Figure 5.6, the voltage profile for *0x110* now reflects the voltage-output

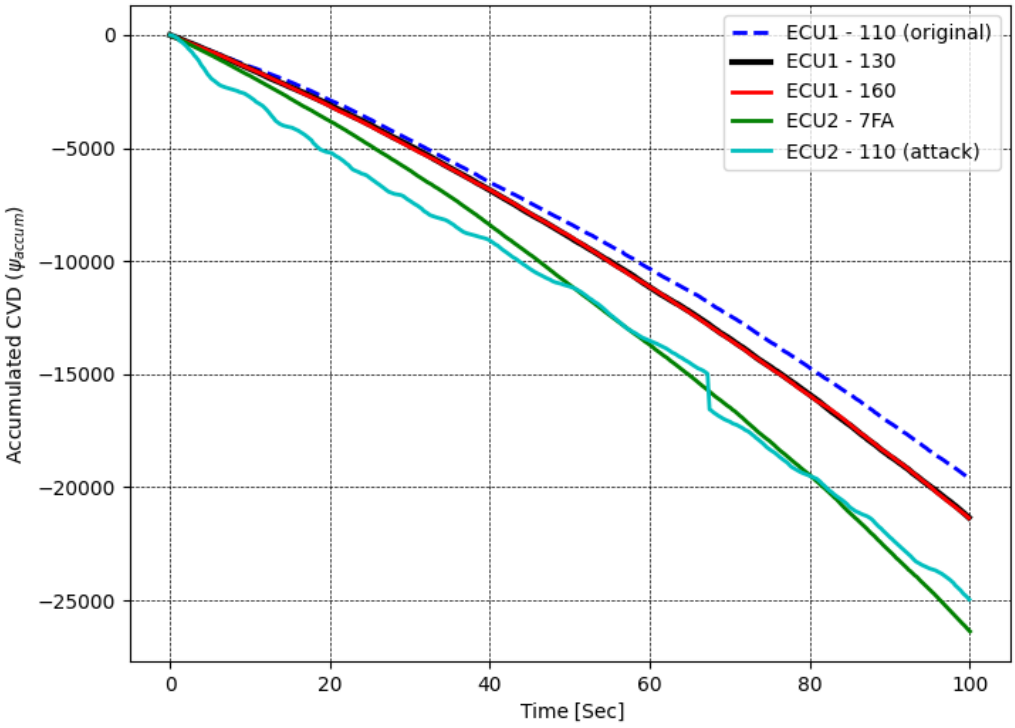


Figure 5.6: *ECU2* tried to impersonate *ECU1*. The message-ID *0x110* belonging to *ECU1* got a near-equivalent profile compared with *ECU2*. Since no voltage-profile collisions existed originally this proves that *ECU2* performed the attack. This was also verified using the classifiers in Phase 4 of Viden.

from the attacker ECU. *0x110* originally belonged to *ECU1* and had the blue dashed line as its voltage profile. Since the message frequency from the attacker *ECU2* was much higher than the legitimate *0x110* messages from *ECU1*, the voltage profile for *0x110* changed and became an equivalent to the profile for message *0x7FA*. Since this profile belongs to *ECU2*, our implementation determined that the messages *0x7FA* and *0x110* are originating from the same transmitter, and thus identifying the attacker to be *ECU2*.

The reason for the attack-profile's instability is due to a few voltage measurements captured from the legitimate ECU sending *0x110*. Therefore, Viden is continuously adjusting the voltage profile; however, since the output is not from a single ECU all the time, it creates some noise. Since Viden only creates one voltage profile per message-ID, as long as the attack messages have a higher frequency than the legitimate messages, the constructed profile will be mainly constructed for the attacker ECU's voltage output behaviour, and therefore more accurately pinpoint the attacker.

5.3 CASAD

We evaluate our implementation of CASAD on real CAN-traffic and investigate whether it efficiently detects any variation in the normal behaviour. Taking into account that the original implementation of CASAD was tested on real vehicles, for this thesis, we do not focus on building a testbed to evaluate our experiments. Instead, we use the Boxcar environment described in Subsection 5.1.2 to test and evaluate our implementation of CASAD. As mentioned in Section 3.4, we inflicted the attacks through one of the virtual ECUs on the CAN-bus of the Boxcar. The CAN-traffic captured from the Boxcars consists of a large payload which we down-sample according to our requirement. These CAN-messages are collected in a file and provided as an input to our implementation of CASAD in Python running on our laptop.

The experiments we performed with CASAD and their short description with corresponding outcomes are summarised in Table 5.2. In order to evaluate and analyse the behaviour of our implementation of CASAD, we performed two variants of each attack, namely:

1. Scenario 1: In this case, we start an attack, run it for a specific duration of time and terminate the attack in order to observe the behaviour of CASAD during and after the attack.
2. Scenario 2: We start an attack and let it run for a long duration of time which is sufficient to capture CASAD's behaviour towards the attack.

Section	Experiment	Description
5.3.1	Experiment C.1: Suspension Attack	Suspend an existing ECU from sending further messages.
5.3.2	Experiment C.2: Fabrication Attack	Impersonate an existing ECU by sending messages at a higher frequency.
5.3.3	Experiment C.3: Masquerade Attack	Suspend an existing ECU and impersonate it by sending messages.

Table 5.2: Experiments with CASAD - Attack types.

We now describe the experiments we performed for three of the primary attack-scenarios described in Section 2.2. The corresponding results for these three attack scenarios will further be discussed in Chapter 6. For each of the subsequent figures the following hold:

- The upper subplot refers to the time-series of raw CAN-bytes
- In the upper subplot, the part of the plot highlighted in blue corresponds to the training-phase for CASAD.
- The part highlighted in red corresponds to the time-range during which an attack was inflicted, while the black part corresponds to the time-ranges when no attack has been executed.
- The lower subplot corresponds to the variation in departure score for each of the scenarios.
- The alarm threshold (indicated by a red dashed line) determines the upper limit for departure scores, which, when exceeded, triggers an alarm.
- The alarm thresholds are determined experimentally, meaning after performing a series of similar experiments and observing the behaviour and the variation in departure score, the alarm thresholds are set accordingly. These alarm thresholds are different in each of the experiments and are influenced by the method proposed in [12].

5.3.1 Experiment C.1: Suspension Attack

The suspension attack was performed by using one of the virtual ECUs, ε_1, to monitor the CAN-traffic while using the VCM to suspend a real target-ECU on the Boxcar by putting it into *programming-mode*. This experiment was performed in two different ways. In the first type, we carried out the attack for a specific duration of time and put the ECU back online on the bus by terminating the programming-

session. In the other experiment, the attack begins at some point in time after the diagnostic-session is established and goes on until the end.

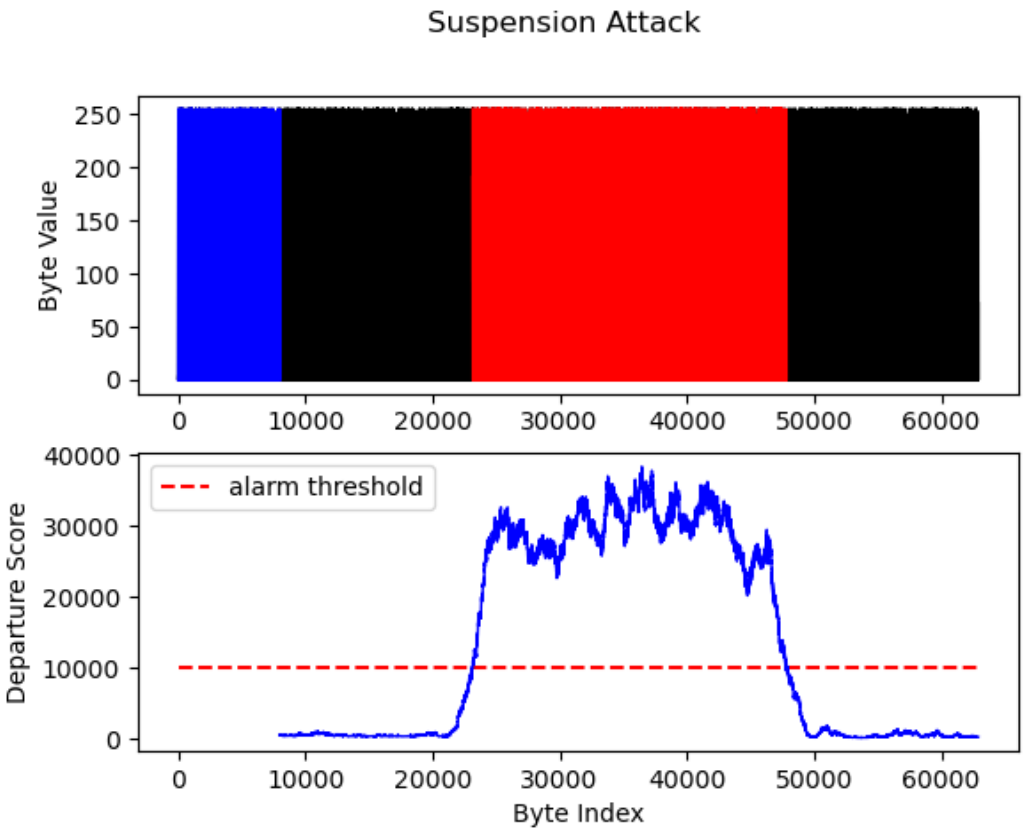


Figure 5.7: Experiment C.1.1: Suspension Attack - Scenario 1.

With an aim to understand the traffic behaviour in the presence of an attack and once it is terminated, we suspend the target ECU for a specific duration and bring it back online, and this corresponds to the first scenario. It is clearly evident from Figure 5.7 that CASAD was able to determine that the suspended ECU was back online after about 48 seconds since the initial diagnostic-session was established. We can observe in Figure 5.8 that once the diagnostic-session is established, the attack begins approximately after 18 seconds and goes on until the end. This corresponds to Scenario 2 with the alarm threshold for the departure score being set at 10,000.

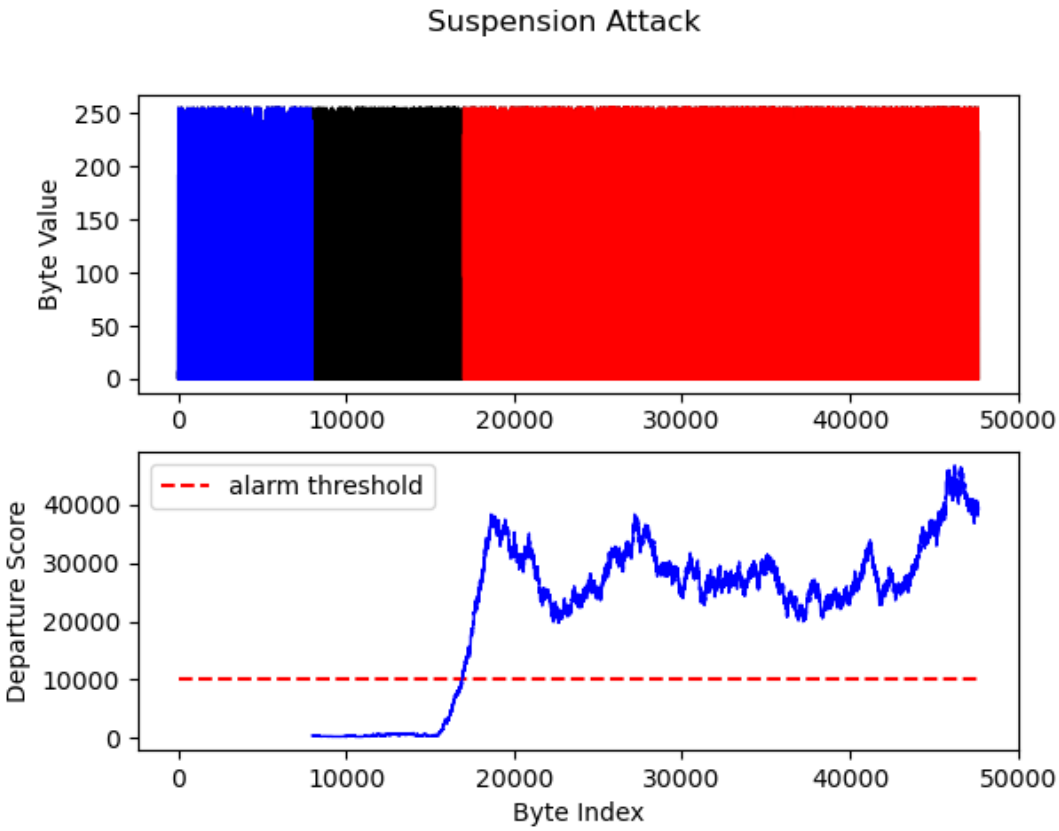


Figure 5.8: Experiment C.1.2: Suspension Attack - Scenario 2.

5.3.2 Experiment C.2: Fabrication Attack

In order to perform a fabrication attack, we assume that the virtual ECU, ε_2, is fully compromised by the adversary and is used to mount an attack on the engine-ECU. For this experiment, we used a proprietary signal-database internally used at Volvo Cars to identify the CAN-message responsible for transmitting the engine rotation speed to the dashboard of the vehicle. The aim was to fabricate the rotation speed of the engine by injecting crafted transmission signal values through ε_2. We performed two types of experiments, one with the attack starting at some point in time and continuing until the end. In the other one, the attack was executed only for a particular duration of time. The results for both of these experiments are shown in Figure 5.9 and Figure 5.10.

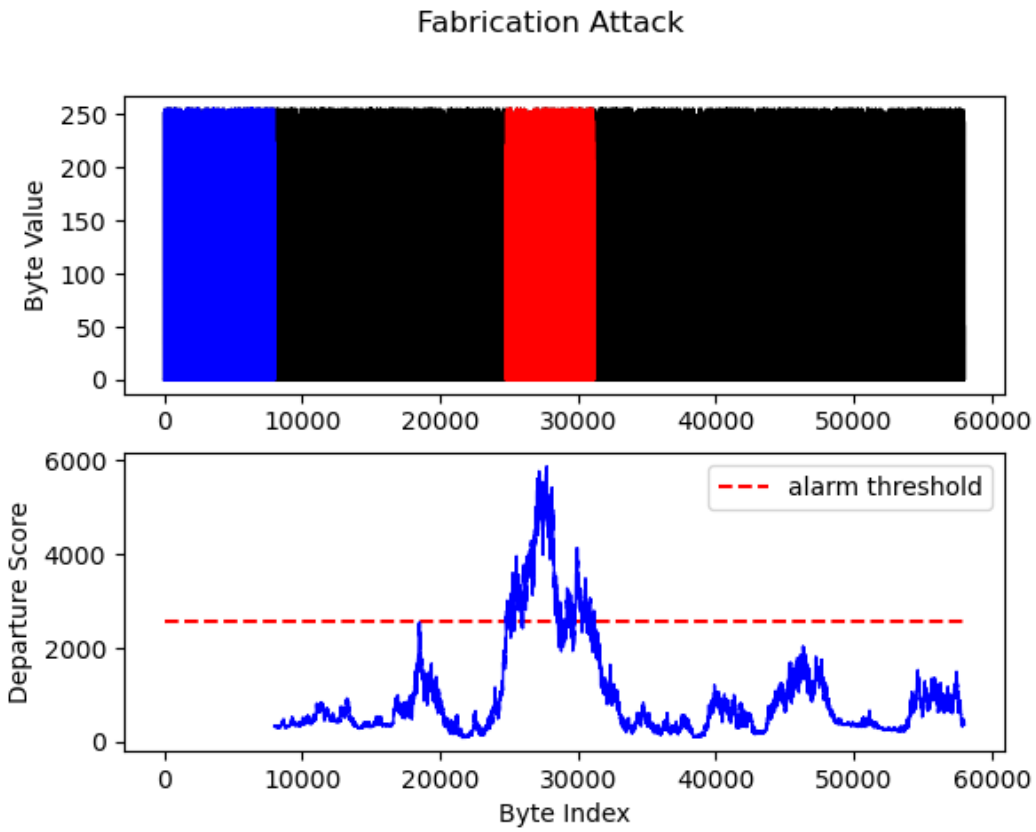


Figure 5.9: Experiment C.2.1: Fabrication Attack - Scenario 1.

The virtual ECU, ε_2, is used to mount a fabrication attack on the target-ECU, in our case, the ECU responsible for transmitting the *Revolutions Per Minute* (RPM) values to the dashboard of the vehicle. While the attack was ongoing, we were able to see the needle of the speedometer move around rapidly. Figure 5.9 shows the

fabrication attack starting at approximately 24 seconds into the experiment and continuing for around eight seconds before the needle on the speedometer comes to a halt. As shown in Figure 5.10, the attack starts quite late (approx. 44 seconds) and goes on until the end of the plot.

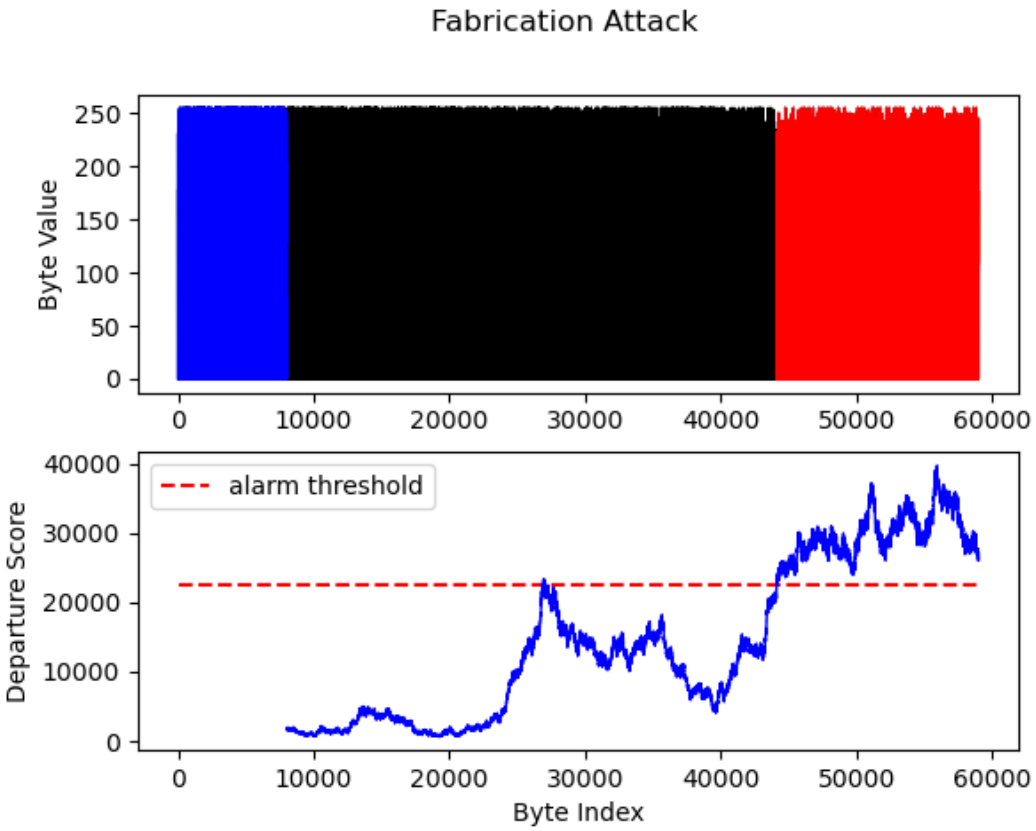


Figure 5.10: Experiment C.2.2: Fabrication Attack - Scenario 2.

5.3.3 Experiment C.3: Masquerade Attack

As mentioned in Section 2.2, a masquerade attack is a combination of both a suspension attack (performed on a weakly compromised ECU) and a fabrication attack (performed through a fully compromised ECU on the same bus). We assume the virtual ECU, ε_2, to be fully compromised by the adversary and perform the masquerade attack in the same way as the fabrication attack with the only addition that the legitimate ECU (in this case the engine-ECU which is responsible for transmitting RPM-values) is suspended. In the first step of the attack, the engine ECU was suspended through a diagnostic session, while ε_2 was programmed to transmit

maliciously altered RPM-messages with the same ID and frequency as the original RPM-message.

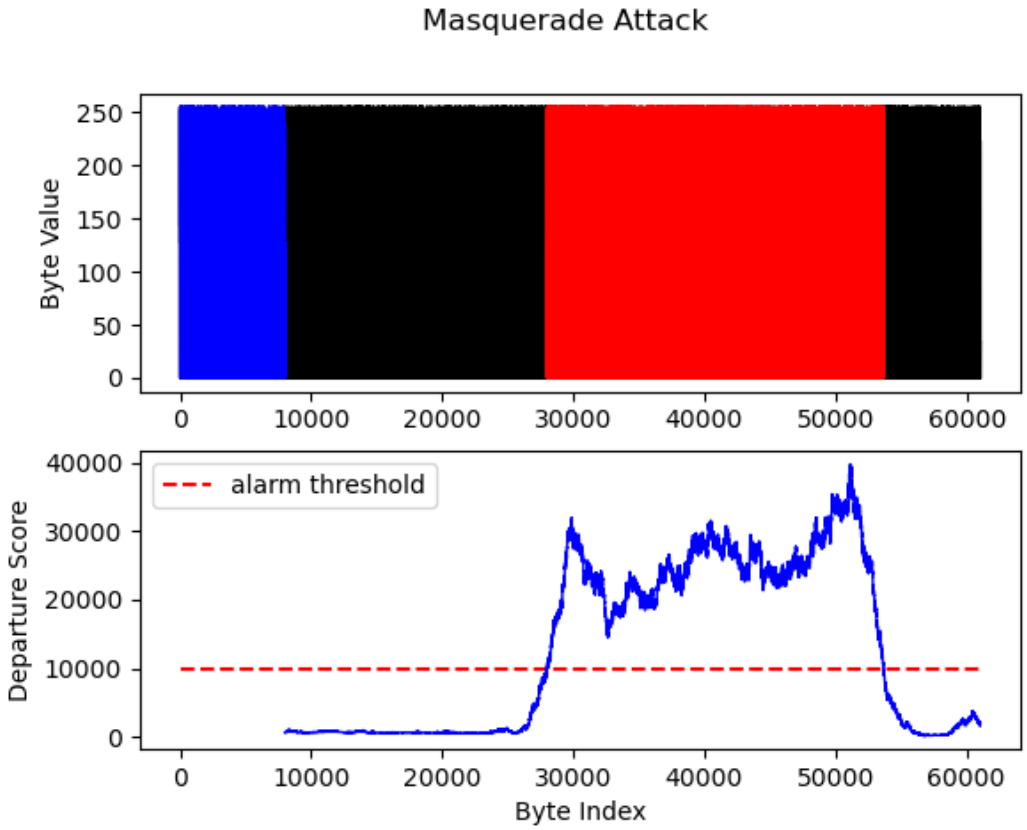


Figure 5.11: Experiment C.3.1: Masquerade Attack - Scenario 1.

Figure 5.11 corresponds to the first scenario where the masquerade attack begins approximately after 28 seconds of session establishment and goes on for about 26 seconds before all the ECUs in the same domain are reset manually. The alarm threshold is set at the departure score value of 10,000, and we observe that all the values above the threshold are identified by CASAD as an attack. Similarly, in the second scenario as shown in Figure 5.12, the threshold remains the same, and we can see that the attack begins at around 19 seconds and goes on until the end of the plot.

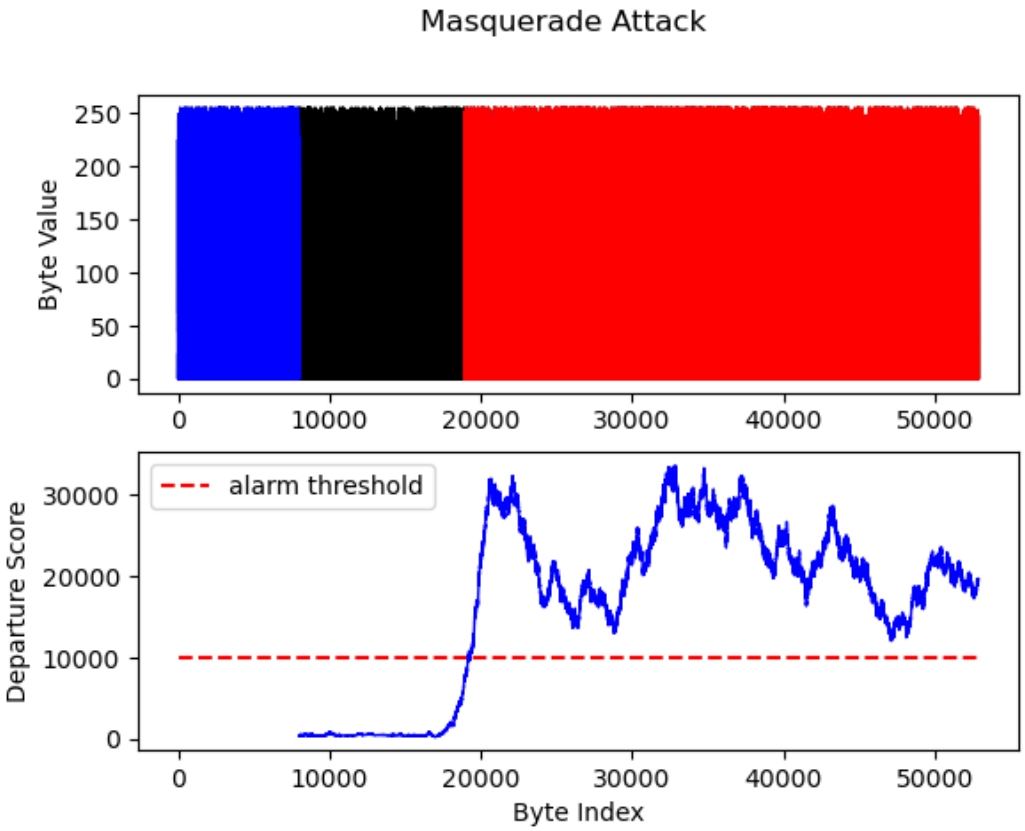


Figure 5.12: Experiment C.3.2: Masquerade Attack - Scenario 2.

5.4 Experiment U: Unified System - CASAD & Viden

We integrated CASAD & Viden to create a unified system where CASAD detects the attack and Viden pinpoints the attacker. As shown in figure 5.13, both CASAD and Viden operate on the same CAN-bus. While CASAD begins with learning the deterministic behaviour of the IVN traffic and monitors it for changes, Viden continuously constructs voltage profiles for all the messages being communicated on the bus. As mentioned in Subsection 3.4.2, CASAD monitors CAN-traffic at the payload level, and any (unusual) change in the payload of the message corresponds to an increase in the departure score. If this value is greater than the alarm threshold, CASAD classifies this change as an attack and triggers Viden. While CASAD is continuously monitoring the departure score, Viden runs Phase 2 and Phase 3 of its operations, to build and update voltage profiles, as discussed in sections 4.2 and 4.3.

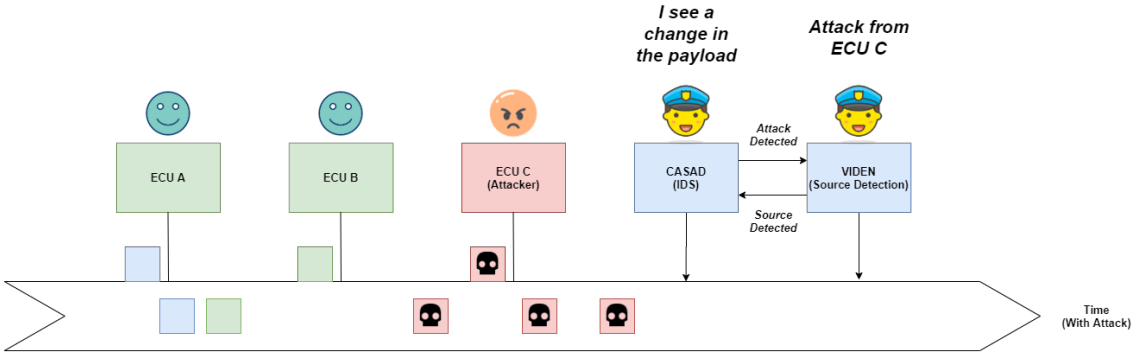


Figure 5.13: Experiment U: Unified System - CASAD as an IDS and Viden as a source detection mechanism.

When CASAD detects an attack, it sends an interrupt to Viden to look for the source of the attack. However, it does not specify any information about the message-ID causing the deviation in the departure score. Hence, all messages received after this are considered by Viden, and it starts to build a new voltage profile for the attack timeline, that is, an *attack profile*. Viden checks if any of the profiles deviated significantly from the previous known profiles by comparing the attack profile's RLS-values with its existing voltage profiles' RLS-values. Based on this, Viden can determine which ECUs have been spoofed and which ECUs are potential attackers.

In Phase 4, Viden investigates these profiles of potential attackers. The classifiers use 50% of the data received until the point of the attack as train-data. This phase

helps Viden to reduce the number of potential attackers and return a predicted attacker with a probability score. In rare cases, if the classifier is not sure of each ECUs' voltage output behaviour, it might return multiple potential attackers.

To evaluate our unified system, we performed a fabrication attack by using the fully compromised virtual ECU, ε_2, to mount an attack by impersonating one of the existing ECUs, in our case the engine-ECU responsible for the RPM-values. Since we run both systems individually and then trigger a message from CASAD to Viden, the results would remain the same as if we were running each system by itself. We further discuss this in Section 6.3.

5.5 Experiment X: Extending CASAD with voltage-analysis

We also saw a possibility for CASAD to fingerprint ECUs solely based on voltage-measurements. As seen in Figure 5.14, CASAD is able to fingerprint three ECUs transmitting two message-IDs each. Each colour represents a voltage profile for a message-ID, and as with Viden, two message-IDs transmitted from the same ECU results in similar profiles, thereby two adjacent colours.

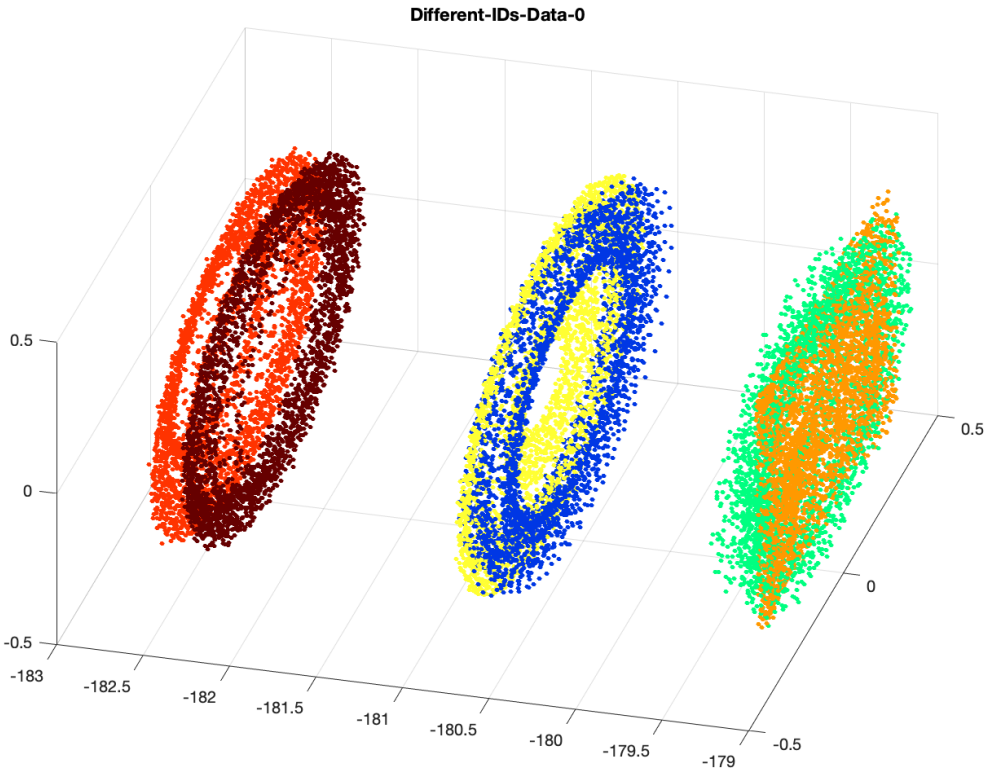


Figure 5.14: CASAD fingerprinting three ECUs using only voltage measurements.

6
Discussion

In this chapter, we discuss the results from the experiments we conducted using the methodology and evaluation presented in Chapter 5. We begin with describing the results for our experiments with Viden on the CAN-bus prototype and then detail the results from the Boxcars. As mentioned earlier, we did not build a testbed for CASAD but performed the experiments directly on Boxcars which are discussed here. Followed by this, the discussion of the unified system is presented. We end this chapter by highlighting the future work, ethical considerations and sustainability-aspects of our thesis.

6.1 Viden

We demonstrated that our Viden-prototype was able to successfully and uniquely fingerprint ECUs using voltage measurements. We demonstrated attacker identification capabilities in case of a fabrication attack in both the CAN-bus prototype (Experiment V1 & V2) and on the Boxcar (Experiment V3 & V4) in Section 5.2. Although the profiling worked well when Viden was able to learn the ACK-threshold for each message-ID successfully, we demonstrate here some of the difficulties we faced throughout our experiments and also discuss the workarounds we have taken. We also discuss how our implementation could be improved.

6.1.1 Arduino - Sampling-rate

In Phase 2, we used the learned ACK-threshold to solely focus on the voltage-output for a particular transmitter in order to create a voltage instance. We optimized our Arduinos by using the free-running mode because, in theory, a higher sampling rate would help us capture the ACK-voltages in Phase 1 for learning the ACK-threshold.

The authors of Viden wrote this in their paper when explaining their ADC-configuration "... (The ADC) had 10-bit resolution and was configured to sample voltages at its maximum rate of 50KSamples/sec.". When looking at Table 4.1 and Figure 4.5 in Section 4.1.2, we see that this is not possible. In order to achieve a sampling rate

of 50KSamples/sec, the ADC-clock frequency must increase to 1 MHz. As a result, the 10-bits ADC-resolution is not reliable, and only eight bits are effective.

We knew from the ATmeaga328 microcontroller datasheet, which the Arduino UNO is based on, that the default sampling rate for the ADC was 9600 KSamples/sec [49]. In order to achieve at least 50KSamples/sec, the authors of Viden must have optimized their Arduinos and used the prescale factor 16 in order to increase the ADC-clock frequency to 1 MHz and thus achieve 76.9KSamples/s. However, they were claiming that the 50KSamples/sec was the maximum rate.

In Phase 2, we also noticed the unpredictable behaviour of using the optimized free-running mode on the ADC to acquire ADC-conversions faster. As seen in Figure 6.1a, although, we iteratively read from CANH and CANL, the number of dominant voltages captured for CANH and CANL would be unequal.

When using large sample sizes such as 100K, the voltage distribution between CANH and CANL could be 70% to 30%. The reason for this, according to the ATmega datasheet is due to changing channels before a conversion is completed [49]. This affected the voltage instances since the voltage distribution of CANL could change significantly between voltage instances. Although this can be solved by stopping the CANH sampling and capturing CANL voltages until the distribution is 50-50, or by using various ADC-flags to avoid changing ADC-channels during conversion, we instead switched back to "single-conversion-mode" since the free-running mode was unstable when reading from two channels. By doing this, our sampling rate dropped to 49.6KSamples/sec, which is the sampling rate used by Cho and Shin. They may have made the same realization and used single conversions. We also realized that Viden do not require a high rate of voltage sampling as long as the sampling rate is sufficient enough to capture ACK-voltages.

As seen in Figure 6.1b, using *analogRead()* to read from CANH and CANL iteratively resulted in an equal number of measurements more often (than in Figure 6.1a), and therefore more accurate voltage instances. The reason for the one difference that could occur sometimes is due to the single/odd number of captured voltage measurements from a message.

6.1.2 Voltage profiles & free variables

When using the voltage instances from Phase 2 to construct voltage profiles, the voltage profiles would be linear in the best-case. However, if the ACK-threshold learned from Phase 1 is inaccurate, it results in ACK-voltages getting captured together with the non-ACK voltages. This would for Viden, seem like a sudden change

```
Connected to COM3.
counter_samples: 100
counter_CANH: 56
counter_CANL: 44
counter_samples: 100
counter_CANH: 46
counter_CANL: 54
counter_samples: 100
counter_CANH: 53
counter_CANL: 47
```

(a) Free-running mode.

```
Connected to COM3.
counter_samples: 100
counter_CANH: 50
counter_CANL: 50
counter_samples: 100
counter_CANH: 50
counter_CANL: 50
counter_samples: 100
counter_CANH: 49
counter_CANL: 51
```

(b) Single conversion mode.

Figure 6.1: A comparison between the free-running mode and the single-conversion-mode. We sample 100 measurements and print the distribution.

in the voltage-output of the particular transmitter and Viden would, therefore, adjust its tracking-points. However, since the ACK-voltage is not originating from the transmitter and not captured regularly, this would confuse Viden and result in non-linear profiles which can not be used to point out the source. The behaviour of Viden, when the ACK-threshold is inaccurate, can be seen in Figure 6.2.

When the ACK-thresholds are accurate, the voltage profiles would be linear as a result. This would require little to no adjustments on the tracking points, and the sensitivity to changes "alpha" variable would not have much of an effect.

Through experiments we noticed that small values on the sensitivity variable such as $5 \cdot 10^{-4}$ were sufficient to make small adjustments on the tracking points when needed in order to keep the profiles linear. In some cases, we needed larger values as big as five. This, however, was not optimal. The profiles, as a result, would reach linearity after a certain time, but a single change in the voltage behaviour afterwards could result in a big adjustment causing the profile to be unstable again until reaching linearity after a certain time. These profiles were also inaccurate and unreliable since they were not linear.

Phase 3 is also sensitive to time since it updates the tracking points based on how much the tracking points have changed (over time) since the last calculated voltage instance. Since our Viden-node flushes the measured voltages to a PC over the serial

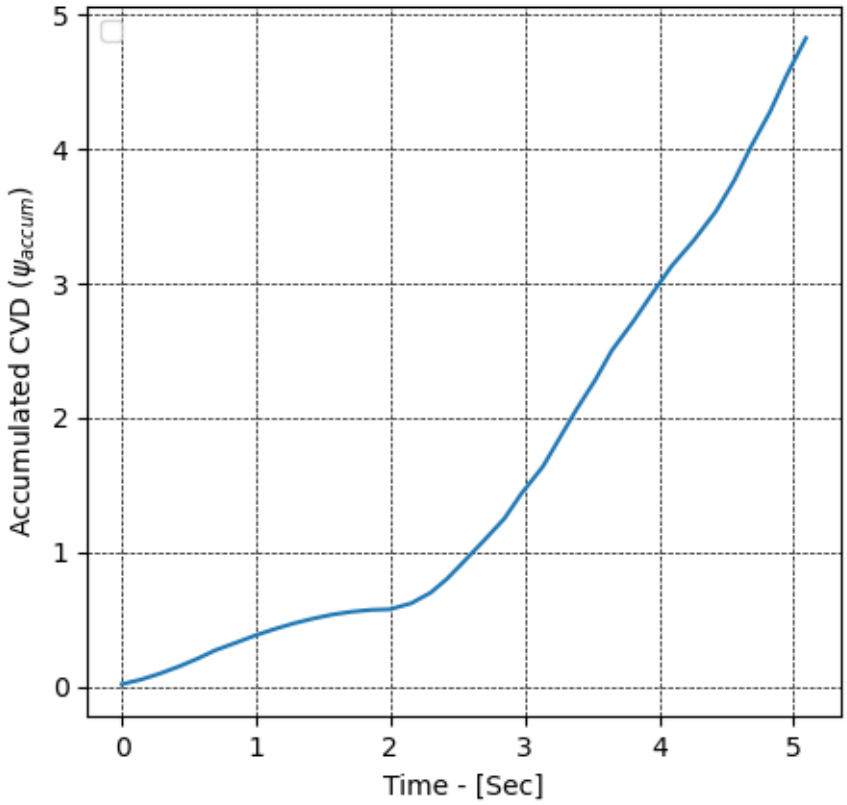


Figure 6.2: The behaviour of Viden when the ACK-threshold is inaccurate. The slope can switch from positive to negative over time.

port, the overload on the serial port could affect the timing-behaviour in theory. The reason for this implementation choice was that, initially, we were not able to run all calculation on the Arduino due to limited SRAM. However, after optimizing our implementation, we reduced the number of voltage measurements required to generate a voltage instance. As a result, the Arduino could fit all measurements in its SRAM, and our optimized implementation would be able to run solely on an Arduino. It would be interesting to test this as a future work. Cho and Shin do not mention how much of their Viden calculations are run on the Arduino compared to the laptop they used.

6.1.3 Fabrication attack

We chose to only present the results for the fabrication attack since the masquerade attack had identical results. However, the masquerade attack profile would contain less noise since the voltage profile was constructed using only measurements from one ECU. The suspension attack would result in the voltage profile disappearing.

6.1.4 Experiment V.3 & Experiment V.4 - Boxcars

When we evaluated our Viden-prototype on the Arduinos, we always got the same voltage profiles since the number of message-IDs on the bus were few. At the Boxcars, we noticed that the voltage profiles are constant when the number of message-IDs tracked are constant. We noticed that increasing the number of message-IDs to track would result in different voltage profiles compared to the voltage profiles without the added message-ID. This is because the number of CVDs accumulated over time would be reduced per message-ID since more IDs are tracked at the same time, and thus the slopes would decline/incline less. Thus, as long as no message-IDs are added to the *"tracking-list"* in close time proximity to the attack-profile, the attack profile can be compared with the existing profiles to pinpoint the attacker.

6.1.5 Limitations - Viden

If multiple ECUs are assigned to send messages with the same ID, only one profile is constructed since profiles are constructed per message ID. However, in reality, this is rare. Since the CAN-protocol has Arbitration on Message Priority as mentioned in Section 2.1, manufacturers avoid two ECUs sending the same ID due to priority collisions that would cause bit/checksum-errors. However, if we know the timing of each ECU transmitting a certain ID, we would be able to create two profiles separately. However, this was not considered in our implementation.

Another limitation is that Viden requires at least one voltage profile to identify an attacker ECU. An attacker could perform an attack at bootup and thus evade the attacker identification. According to Cho and Shin, this could be solved by storing voltage profiles and then updating them via profile adjustments. However, this was not tested in our implementation.

6.2 CASAD

As described earlier in Table 5.2, we performed three types of attacks and observed interesting results for each of the two variants for the different types of attacks. The following subsections correspond to the results and discussion of our experiments.

6.2.1 Experiment C.1: Suspension Attack

As we detailed in Section 3.4, CASAD processes CAN-traffic byte by byte and monitors it at the payload-level. It is interesting to know that in a suspension attack, the adversary only manages to suspend the target ECUs and does not manipulate the payloads of CAN-messages, yet CASAD successfully detects attacks as visible in both Figure 5.7 and Figure 5.8. The primary reason for this detection is the change in the sequence of bytes when the target ECU is suspended. CASAD would be unfamiliar with these changed subsequences from the learning-phase and hence would term this as an attack. This also ensures that CASAD would efficiently detect any change in frequency of transmitted messages as there would be a change in the sequence of bytes in the payload.

6.2.2 Experiment C.2: Fabrication Attack

For the first type of attack, the virtual ECU, ε_2, takes over the ECU responsible for transmitting RPM-values for eight seconds; a short duration to fluctuate the speedometer needle rapidly. The increase in departure score values corresponding to these eight seconds can be seen in Figure 5.9 with all the values above the threshold (at departure score of 2560) being identified as an attack by CASAD. However, in the second case as in Figure 5.10, the increase in departure score at the centre of the graph can be accredited to the fact that only the ECU had been unlocked (i.e., the adversary managed to get access to the ECU) but the attack was still not instigated. As and when the adversary managed to manipulate the payload (44 seconds onward) the departure score crossed the threshold, prompting CASAD to classify this as an attack as visible in the plot. The time-interval between accessing the ECU and starting the attack is significant in this case, such that the threshold

falls back to normal and only rises when the attack starts. This is one of the reasons that the alarm threshold in this scenario has been set higher than usual.

6.2.3 Experiment C.3: Masquerade Attack

We observe from the experiments for masquerade attacks that while the attack is in place, the departure score always stays above the alarm threshold, yet falls and rises several times owing to the variation in the payload of the message as inflicted by the adversary. The departure score directly depends on how and when ε_2 transmits maliciously altered RPM-values and thus corresponds to the deviations for the same. Technically, in our case, ε_2 being the fully compromised ECU takes the role of the partially compromised engine-ECU. ε_2 suspends the target-ECU and makes changes to the payload byte sequence of the intended message. Since CASAD analyses payloads, the alterations to the message payloads can be easily detected by CASAD as visible from Figure 5.11 and Figure 5.12. Moreover, ECUs having protection-mechanisms against receiving messages can withstand fabrication attacks but are still susceptible to masquerade attacks since it involves both suspending the target ECU and making changes in the payload byte sequence of the intended message. But, since CASAD process CAN-traffic at the payload level, it is capable of detecting both fabrication and masquerade attacks as confirmed from our experiments.

6.3 Experiment U: Unified System - CASAD & Viden

In the unified system we propose, we run both CASAD and Viden together yet independently. CASAD monitors the departure score, and Viden continuously constructs and updates voltage profiles. Once a trigger is transmitted from CASAD, Viden focuses on a specific timeline to construct attack profiles. In our experiments, the fabrication attack was successfully detected by the unified system as CASAD observed a change in the departure score and then notified Viden, which constructed attack profiles for the corresponding timeline. Once attack profiles were ready, Viden compared them with the existing profiles and was able to pinpoint that the virtual ECU, ε_2, was responsible for the attack.

In Section 5.4, we mentioned that although CASAD detects an attack it does not specify the message-ID responsible for the attack. If CASAD was able to indicate what message-ID initiated the attack, Viden would build an attack profile for only that message-ID. Hence, Viden would capture more attack related messages, leading to a more accurate attack profile. This would also result in a higher precision

of prediction from the classifier. In our experiments with the unified system, the classifier and hence the system predicted the attacker accurately. We were not able to construct a scenario where the classifier was unsure and made a wrong prediction. However, more experiments are needed to strengthen our claim for accurately predicting with a unified system.

6.4 Future Work

Our Viden-prototype was able to fingerprint the transmitters when successfully having learned the ACK-thresholds. An interesting next step in our work could be to translate all code to solely run on an Arduino and thus avoiding I/O-bottlenecks. We believe that this would not only reduce the load of the process but also improve the voltage profiles due to the accurate timing of the measurements.

It would also be interesting to test the stealthy conquest attack proposed by CASAD's authors [12] and see whether our CASAD-implementation can detect the attack and whether Viden can pinpoint the stealthy attacker. Another interesting addition to our proposed unified system would be to add a prevention mechanism, so that in case an attack is detected and the source is known, an action could be performed to block the attack.

As described in Section 5.5 (Experiment X), we also saw a possibility for CASAD to fingerprint ECUs solely based on voltage-measurements; however, we were not able to further investigate its capability due to time limitations. It would be interesting to further explore CASAD's capabilities as future work.

6.5 Ethical Considerations & Sustainability

With the large attack surfaces of modern vehicles, manufacturers have a responsibility to investigate and propose various defence measures that can detect intrusion, and also even prevent attacks. No matter how well an IDS detects the presence of an intrusion in a vehicle, if it is not capable of finding which ECU is mounting the attack and hence not capable of knowing which ECU to isolate/patch, the vehicle may remain insecure and unsafe in some circumstances. This vehicle may not only potentially cause a safety risk for the driver and the passengers but also put the people around at risk if the compromised component is safety-critical.

From a sustainability perspective, vehicles equipped with Intrusion Detection and

Prevention Systems that can isolate and patch compromised ECUs not only save lives but also reduce the impact on the environment by emissions and manufacturing costs because of fewer vehicles requiring service at auto-repair shops.

We believe that our proposed unified system together with a prevention mechanism which responds to the attacks would contribute to solving the mentioned issues.

7
Conclusion

Modern vehicles have evolved from being just a means of transportation to offering various new functionalities such as navigation and entertainment systems. Although these functionalities have provided a high level of connectivity, they have made the modern vehicle more vulnerable to attacks. However, state-of-the-art vehicle security solutions lack the ability to pinpoint the source of an attack which is essential to efficiently detect, isolate and patch the attack. Through this thesis, we propose a unified system for both attack detection and source identification, by using two of the most prominent works in vehicle-cybersecurity, CASAD and Viden.

We have investigated the feasibility of using voltage measurements to fingerprint ECUs. We have demonstrated that, by learning the ACK-threshold, our Viden-prototype was able to focus on voltage measurements solely outputted from the actual transmitter and exploit these measurements to construct voltage instances reflecting the voltage output behaviour. These voltage instances were then used to construct and continuously update voltage profiles. Through experimental evaluations on the CAN-prototype and the Boxcar, we verified the accuracy, viability, practicality, and efficiency of Viden. We also demonstrated that each voltage profile constructed uniquely represented each ECU.

We studied the algorithm of PASAD, proposed by Aoudi et al. for detecting attacks in an industrial setting. We demonstrated that our implementation of the version for IVNs, CASAD, is capable of identifying underlying changes in the payload of messages on the CAN-bus. Through evaluation on real CAN-traffic from Boxcars, we also demonstrated that CASAD is able to successfully detect the three most common types of attacks on IVNs. Finally, we also saw a possibility for CASAD to fingerprint ECUs solely based on voltage measurements.

Finally, we proposed a unified system where CASAD acts as an IDS and detects the attacks and then triggers Viden, which acts as a fingerprinting technique based on voltage as the low-level characteristic. We believe that the functionality our unified system provides is an important step towards securing IVNs.

Bibliography

[1] Nasser Nowdehi, Aljoscha Lautenbach, and Tomas Olovsson. "In-Vehicle CAN Message Authentication: An Evaluation Based on Industrial Criteria". In: *The Proceedings of 2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*. IEEE. 2017, pp. 1–7.

[2] Staff Writer. *5 Advantages of CAN Bus Protocol*. Online - Accessed 2020-03-23. 2019. URL: https://www.totalphase.com/blog/2019/08/5-advantages-of-can-bus-protocol/.

[3] Robert Bosch GmbH. *CAN Specification, Version 2.0*. Online - Accessed 2020-05-08. 1991. URL: https://www.kvaser.com/software/7330130980914/V1/can2spec.pdf.

[4] Andy Greenberg. "Hackers remotely kill a jeep on the highway—with me in it". In: *Wired* (2015).

[5] Andy Greenberg. "Tesla responds to Chinese hack with a major security upgrade". In: *Wired* (2016).

[6] Samuel Woo, Hyo Jin Jo, and Dong Hoon Lee. "A Practical Wireless Attack on the Connected Car and Security Protocol for In-Vehicle CAN". In: *IEEE Transactions on Intelligent Transportation Systems* 16.2 (Apr. 2015), pp. 993–1006. ISSN: 1524-9050. DOI: 10.1109/TITS.2014.2351612.

[7] Charlie Miller and Chris Valasek. "A survey of remote automotive attack surfaces". In: *Black Hat USA* 2014 (2014), pp. 1–94.

[8] Frederic Lambert. *Keen Lab hackers managed to take control of Tesla vehicles again*. Online - Accessed 2020-05-08. 2017. URL: https://electrek.co/2017/07/28/tesla-hack-keen-lab/.

[9] Charlie Miller and Chris Valasek. "Remote exploitation of an unaltered passenger vehicle". In: *Black Hat USA* 2015 (2015), pp. 1–91.

[10] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. "Comprehensive experimental analyses of automotive attack surfaces." In: *USENIX Security Symposium*. Vol. 4. San Francisco. 2011, pp. 447–462.

[11] Siti-Farhana Lokman, Abu Talib Othman, and Muhammad-Husaini Abu-Bakar. "Intrusion detection system for automotive Controller Area Network (CAN) bus system: a review". In: *EURASIP Journal on Wireless Communications and Networking* 2019.1 (2019), pp. 184–201.

[12] Nasser Nowdehi, Wissam Aoudi, Magnus Almgren, and Tomas Olovsson. "CASAD: CAN-Aware Stealthy-Attack Detection for In-Vehicle Networks". In: *arXiv preprint arXiv:1909.08407* (2019).

[13] Pierre Kleberger, Nasser Nowdehi, and Tomas Olovsson. "Towards designing secure in-vehicle network architectures using community detection algorithms". In: *2014 IEEE Vehicular Networking Conference (VNC)*. IEEE. 2014, pp. 69–76.

[14] Min-Joo Kang and Je-Won Kang. "Intrusion detection system using deep neural network for in-vehicle network security". In: *PloS one* 11.6 (2016).

[15] Wonsuk Choi, Hyo Jin Jo, Samuel Woo, Ji Young Chun, Jooyoung Park, and Dong Hoon Lee. "Identifying ECUs using inimitable characteristics of signals in controller area networks". In: *IEEE Transactions on Vehicular Technology* 67.6 (2018), pp. 4757–4770.

[16] Pal-Stefan Murvay and Bogdan Groza. "Source identification using signal characteristics in controller area networks". In: *IEEE Signal Processing Letters* 21.4 (2014), pp. 395–399.

[17] Kyong-Tak Cho and Kang G Shin. "Viden: Attacker identification on in-vehicle networks". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 1109–1123.

[18] *Introduction to the Controller Area Network (CAN)*. SLOA101B. Rev. 2016. Texas Instruments. Aug. 2002.

[19] enginerd4ni. *Trying to make sense of T4R OBD/CAN PID codes*. Online - Accessed 2020-05-08. URL: `https://www.toyota-4runner.org/engines-suspension-wheels-tires-audio-accessories/146434-trying-make-sense-t4r-obd-can-pid-codes.html`.

[20] Robert Buttigieg, Mario Farrugia, and Clyde Meli. "Security Issues in Controller Area Networks in Automobiles". In: *18th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*. IEEE. 2017.

[21] Wei Si, David Starobinski, and Moshe Laifenfeld. "Protocol-Compliant DoS Attacks on CAN: Demonstration and Mitigation". In: *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)*. 2016, pp. 1–7.

[22] Olivier Cros and Gabriel Chênevert. "Hashing-based authentication for CAN bus and application to Denial-of-Service protection". In: *2019 3rd Cyber Security in Networking Conference (CSNet)*. 2019, pp. 91–98.

[23] Guillaume Dupont, Sandro Hartog Jerry den Etalle, and Alexios Lekidis. “Network intrusion detection systems for in-vehicle network - Technical report”. In: *arXiv preprint arXiv:1905.11587* (2019).

[24] Hyun Min Song, Ha Rang Kim, and Huy Kang Kim. “Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network”. In: *2016 International Conference on Information Networking (ICOIN)*. 2016, pp. 63–68.

[25] Adrian Taylor, Nathalie Japkowicz, and Sylvain Leblanc. “Frequency-based anomaly detection for the automotive CAN bus”. In: *2015 World Congress on Industrial Control Systems Security (WCICSS)*. 2015, pp. 45–49.

[26] Michael Müter and Naim Asaj. “Entropy-based anomaly detection for in-vehicle networks”. In: *2011 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2011, pp. 1110–1115.

[27] Mirco Marchetti, Dario Stabili, Alessandro Guido, and Michele Colajanni. “Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms”. In: *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*. 2016, pp. 1–6.

[28] Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz. “Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks”. In: *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)* (2016), pp. 130–139.

[29] Marc Weber, Simon Klug, Bastian Zimmer, and Eric Sax. “Embedded Hybrid Anomaly Detection for Automotive CAN Communication”. In: *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*. Jan. 2018, pp. 1–10.

[30] Chundong Wang, Zhentang Zhao, Liangyi Gong, Likun Zhu, Zheli Liu, and Xiaochun Cheng. “A Distributed Anomaly Detection System for In-Vehicle Network Using HTM”. In: *IEEE Access* 6 (2018), pp. 9091–9098.

[31] Vilhelm Verendel, Dennis Nilsson, Ulf Larson, and Erland Jonsson. “An Approach to using Honeypots in In-Vehicle Networks”. In: *2008 IEEE 68th Vehicular Technology Conference*. 2008, pp. 1–5.

[32] Ulf Larson, Dennis Nilsson, and Erland Jonsson. “An approach to specification-based attack detection for in-vehicle networks”. In: *2008 IEEE Intelligent Vehicles Symposium*. 2008, pp. 220–225.

[33] Dennis Nilsson and Ulf Larson. “A Defense-in-Depth Approach to Securing the Wireless Vehicle Infrastructure”. In: *Journal of Networks*. Vol. 4. Sept. 2009. DOI: 10.4304/jnw.4.7.552-564.

[34] Pierre Kleberger, Tomas Olovsson, and Erland Jonsson. “Security aspects of the in-vehicle network in the connected car”. In: *2011 IEEE Intelligent Vehicles Symposium (IV)*. 2011, pp. 528–533.

[35] *SN65HVD233-HT 3.3-V CAN Transceiver*. SLLS933G. Rev. 2015. Texas Instruments. Jan. 2008.

[36] John Griffith. *Learn the inner workings of a CAN bus driver and how to debug your system*. Online - Accessed 2020-05-08. Jan. 2016. URL: https://e2e.ti.com/blogs_/b/industrial_strength/archive/2016/01/26/the-inner-workings-of-a-can-bus-driver.

[37] *Introduction to the Controller Area Network (CAN)*. SLOA101A. Rev. 2008. Texas Instruments. Aug. 2002.

[38] TiePie. *Common-mode rejection ratio*. Online - Accessed 2020-05-08. URL: https://www.tiepie.com/en/fut/cmrr.

[39] Najamuz Zaman. *Automotive electronics design fundamentals*. (Book). Springer International Publishing, 2015.

[40] Sparsh Mittal. “A Survey of Architectural Techniques for Managing Process Variation”. In: *ACM Comput. Surv.* 48.4 (Feb. 2016). ISSN: 0360-0300. DOI: 10.1145/2871167. URL: https://doi.org/10.1145/2871167.

[41] *ISO 11898-2:2016. Road vehicles — Controller area network (CAN) — Part 2: High-speed medium access unit*. Online - Accessed 2020-05-08. Dec. 2016. URL: https://www.iso.org/standard/67244.html.

[42] *ISO 11898-4:2004. Road vehicles — Controller area network (CAN) — Part 4: Time-triggered communication*. Online - Accessed 2020-05-08. Apr. 2019. URL: https://www.iso.org/standard/36306.html.

[43] Steve Corrigan. “Common-mode effects in CAN networks”. In: *CAN Newsletter 1/2012* (2012), pp. 52–54.

[44] Sang Uk Sagong, Xuhang Ying, Andrew Clark, Linda Bushnell, and Radha Poovendran. “Cloaking the clock: emulating clock skew in controller area networks”. In: *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*. IEEE Press. 2018, pp. 32–42.

[45] Wissam Aoudi, Mikel Iturbe, and Magnus Almgren. “Truth will out: Departure-based process-level detection of stealthy attacks on control systems”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2018, pp. 817–831.

[46] Kyong-Tak Cho and Kang G. Shin. “Fingerprinting Electronic Control Units for Vehicle Intrusion Detection”. In: *Proceedings of the 25th USENIX Conference on Security Symposium*. SEC’16. Austin, TX, USA: USENIX Association, 2016, pp. 911–927. ISBN: 9781931971324.

[47] Miaoqing Tian, Ruobing Jiang, Chaoqun Xing, Haipeng Qu, Qian Lu, and Xiaoyun Zhou. “Exploiting Temperature-Varied ECU Fingerprints for Source

Identification in In-vehicle Network Intrusion Detection". In: *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*. IEEE Press. 2019, pp. 1–8.

[48] Omid Avatefipour. "Physical-Fingerprinting of Electronic Control Unit (ECU) Based on Machine Learning Algorithm for In-Vehicle Network Communication Protocol "CAN-BUS"". In: (2017).

[49] Atmel. "ATmega328P - Datasheet". In: (2015). URL: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

[50] *Github: CAN Bus Shield - MCP2515 & MCP2551*. Online - Accessed 2020-05-08. 2020. URL: https://github.com/Seeed-Studio/CAN_BUS_Shield.

[51] Caffeinomane. *Girino - Fast Arduino Oscilloscope*. Online - Accessed 2020-05-08. 2012. URL: https://www.instructables.com/id/Girino-Fast-Arduino-Oscilloscope/.

[52] Bill Earl. "Memories of an Arduino". In: *adafruit learning system* (2018). URL: https://cdn-learn.adafruit.com/downloads/pdf/memories-of-an-arduino.pdf.

[53] Simon Haykin. *Adaptive Filter Theory (3rd Ed.)* USA: Prentice-Hall, Inc., 1996. ISBN: 013322760X.

[54] Better Explained. *Understanding the Birthday Paradox*. Online - Accessed 2020-09-22. URL: https://betterexplained.com/articles/understanding-the-birthday-paradox/.

[55] javaTpoint. *Random Forest Algorithm*. Online - Accessed 2020-09-08. URL: https://www.javatpoint.com/machine-learning-random-forest-algorithm.

[56] Scikit-learn. *sklearn - RandomForestClassifier*. Online - Accessed 2020-09-08. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[57] Avinash Navlani. *Understanding Random Forests Classifiers in Python*. Online - Accessed 2020-09-08. 2018. URL: https://www.datacamp.com/community/tutorials/random-forests-classifier-python.

[58] Seeedstudio. *CAN-BUS Shield V2.0*. Online - Accessed 2020-05-08. 2020. URL: http://wiki.seeedstudio.com/CAN-BUS_Shield_V2.0/.