Intrusion detection on the automotive CAN bus

Schappin, C.N.I.W.

Award date:
2017

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Security Group

Intrusion Detection on the Automotive CAN bus

Master's Thesis

C.N.I.W. (Colin) Schappin

Supervisors:
dr. N. (Nicola) Zannone, Security Group, Eindhoven University of Technology
ir. H. (Henri) Hambartsumyan, Cyber Risk Services, Deloitte

May 2017

"The question of whether machines can think is about as relevant as the question of whether submarines can swim."

- Edsger Dijkstra

Abstract

In this thesis we investigate the possibilities for intrusion detection on the Controller Area Network (CAN). Modern developments in automotive innovation mainly focus on adding new forms of external interfaces to a vehicle with the goal of increasing comfort or safety. However, these interfaces often (indirectly) expose the vehicle's internal networks to the outer world. This gives rise to many new automotive security threats. Hence, during the development of these interfaces it is very important to address security. First, we investigate how automotive attacks are executed and what this looks like on the CAN bus. Then, we analyse and compare various automotive intrusion detection systems that have been proposed in literature with the goal of detecting these kinds of attacks. We select three anomaly detection algorithms that seem most promising in performing intrusion detection on the CAN bus: the support vector machine, isolation forest and robust covariance estimator. These algorithms are so-called one-class classifiers, they build a model based on data belonging to one class, in our case this is CAN data during normal operation. New observations are classified as an attack when they do not fit the built model. We simulate the identified attacks on a CAN dataset and compare the anomaly detection performance of these three algorithms. We conclude that the robust covariance estimator gives the best results in this setting and thus would be the most appropriate to use in an automotive intrusion detection system.

Acknowledgements

This thesis marks the end of my master's degree and with that the end of my career as a student at the Eindhoven University of Technology. Looking back at my time in Eindhoven there were times I had a lot of fun and there were very busy and stressful times. Now I can say all this hard work paid off.

First of all, I would like to thank Nicola Zannone from the TU/e, who has provided me with lots of comments and guidance that pointed me in the right direction. I also would like to thank Henri Hambartsumyan from Deloitte for all his guidance and useful discussions. Both supervisors were very helpful and always made time for me when I needed it. Nicola and Henri looked at my work from different perspectives which was extremely helpful for me. I am also grateful to Reinder Bril for accepting a place in my exam committee.

I would like to thank all my colleagues at Deloitte. I had a very good time with all of you and a lot of fun playing games at the office. Moreover, the many interesting conversations we had both in general and related to my thesis. You are an amazing group of very skilful people. Specifically, I would like to thank James and Jilles for their help with putting together and testing numerous devices.

A very big thanks goes to my parents, sister and friends for their support and welcome distractions in my free time. A special thanks goes to my girlfriend Bea for her encouragement and unconditional support. I can imagine that most of the time they had no idea what I was working on.

Regards,

Colin

Contents

List of Figures

List of Tables

List of Abbreviations

ALDL . . . . . Assembly Line Diagnostic Link
AUC . . . . . Area Under the Curve
BST . . . . . Binary Search Tree
C2C-CC . . . . . Car 2 Car Communication Consortium
CAN . . . . . Controller Area Network
CIDS . . . . . Clock-based Intrusion Detection System
CUSUM . . . . Cumulative Sum Method
DAB+ . . . . . Digital Audio Broadcasting Plus
DoS . . . . . Denial of Service
ECU . . . . . Electronic Control Unit
EOBD . . . . . European On Board Diagnostics
FPR . . . . . False-Positive Rate
GPS . . . . . Global Positioning System
ICSim . . . . . Instrument Cluster Simulator
IDS . . . . . Intrusion Detection System
IP . . . . . Internet Protocol
IPS . . . . . Intrusion Prevention System
ITS . . . . . Intelligent Transport System
LIN . . . . . Local Interconnect Network
LLC . . . . . Logical Link Control
MAC . . . . . Medium Access Control
MANET . . . . Mobile Ad Hoc Network
MCD . . . . . Minimum Covariance Determinant
MOST . . . . . Media Oriented Systems Transport
MVE . . . . . Minimum Volume Ellipsoid
OBD-II . . . . . On-Board Diagnostics version II
OCC . . . . . One-Class Classification
PCB . . . . . Printed Circuit Board
RDS . . . . . Radio Data System
RF . . . . . Radio Frequency
RFID . . . . . Radio Frequency Identification
RKE . . . . . Remote Keyless Entry

ROC . . . . . Receiver Operating Characteristic
SOME/IP . . . Scalable Service-Oriented Middleware for IP
SVM . . . . . Support Vector Machine
TCP/IP . . . . . Transmission Control Protocol for IP
TMC . . . . . Traffic Message Channel
TPMS . . . . . Tire Pressure Monitoring System
TPR . . . . . True-Positive Rate
TU/e . . . . . Eindhoven University of Technology
V2I . . . . . Vehicle-to-Infrastructure
V2V . . . . . Vehicle-to-Vehicle
V2X . . . . . Vehicle-to-Everything
VANET . . . . . Vehicular Ad Hoc Network

Chapter 1

Introduction

In recent years automotive innovation focused for a large part on communication technology, adding many forms of external interfaces to modern vehicles. Connections are being made to other vehicles and even to the Internet. Often these external interfaces (indirectly) expose the vehicle's internal networks. One of the technologies that is heavily under development is vehicle-to-vehicle (V2V) communication. The predictions of the amount of connected cars being driven in 2020 range from 125 to 250 million vehicles [42], this is around one fourth of the 900 million commercial vehicles that were in use in 2014 [72]. Securing these types of vehicles is a critical step in their development and deployment, since any vulnerability can put lives at stake.

Modern cars contain multiple types of networks, e.g. CAN, FlexRay, LIN and MOST. Among the others, CAN connects Electronic Control Units (ECUs) controlling all kinds of systems, e.g. brakes, lights and air bags, within a vehicle. Inside a modern vehicle there are up to 100 ECUs, all with their own operating system and program logic to control the systems they are operating, resulting in a total amount of almost 100 million lines of code [9]. This amount is much more than that of many technologies we use today, as can be seen in Figure 1.1, and can result in numerous security critical software flaws.

When CAN was introduced, in 1986, a car was still an isolated system, it had no connections to the outside world. Because of this, CAN was not designed with security in mind. The additional features present on a modern car give possibilities for many different attacks that were not possible before, e.g. DoS or injection attacks [33]. These attacks can result in damaging on-board systems, the vehicle itself or even its passengers. Thus, it is very important to research and secure these systems before they are widely used by society.

In 2013, two security researchers, Charlie Miller and Chris Valasek, demonstrated attacks on a Ford Escape and a Toyota Prius [27]. Sitting in the back seat of the vehicles with their laptops connected to the car, via a cable to the car's diagnostics port, they were able to control many aspects of the cars. They could blast the horn, slam the brakes at high speed, kill the power steering, and more. The victim of these attacks was Andy Greenberg, a technology journalist. While Ford said to take the hackers very seriously, Toyota was not very impressed. They stated that real car hacking happens from a remote wireless device. They claimed to have tested for remote attacks and that their vehicles are robust and secure.

In 2015 Greenberg again was the victim of Miller and Valasek's attacks [29], but this time the researchers did not need to be inside the vehicle, they could execute their attacks from anywhere, over the Internet. In this attack they were able to engage or disable the brakes, hijack the steering wheel when the vehicle is in reverse and even kill the engine at lower speeds. They could also track a car's GPS coordinates and measure its speed. They gained remote access to the vehicle via Uconnect [21], a system that controls in vehicle entertainment, navigation, offers the option the make phone calls and even has a Wi-Fi hotspot. Uconnect was reachable via its cellular connection. Once inside that system, the researchers could get access to the ECU controlling the entertainment system, from here they were able to send commands to the CAN bus. Two years earlier Toyota was not impressed by their local attack, now remote car hacking had become reality.

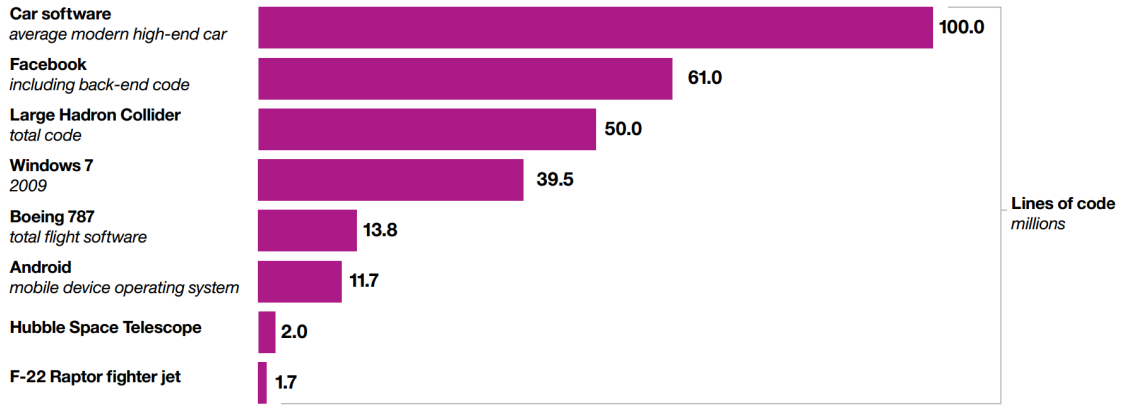

Figure 1.1: Lines of code in a modern car compared to other technologies [59]

Chrysler did realise this attack exposed a huge security risk and recalled 1.4 million vehicles [22].

A year later, in 2016, Miller and Valasek were back again to present new attacks [30]. They continued their research presented a year earlier and managed to execute much more destructive actions this time. They were able to accelerate, slam the brakes and turn the steering wheel at any speed. On the contrary to a year ago, this time they had to be physically connected to the car again. In the response statement of affected companies the emphasis was, surprisingly, again on the fact that the attack could not have been performed remotely. It should be clear by now that the car manufacturers' focus should be on the security of systems regardless of whether a remote attack is currently possible or not. The aim should be that when a new remote attack is found, an attacker cannot do (life threatening) damage because of the security measures inside the car. That is the reason why the attackers did not spend time looking for new remote exploits, since they already proved it is possible. It is only a matter of time before someone will discover another wireless vulnerability and also their new attacks can be executed remotely.

These two researchers are just one example of the active research into cyber security of automotive vehicles. Some researchers focus on the analysis of external connections, how these can be used to gain access to a car's internal network [10], [53]. Other researchers focus on the part of the attack that follows after getting access, creating network packets that are able to control the car [33].

The Vault 7 documents leaked by whistle blower website WikiLeaks on 7 March 2017 contained a number of documents indicating the CIA also has been investigating car hacking [77]. The purpose of this investigation was not specified in the leaked documents, nor by the CIA itself after the leak. The documents indicate that one of the systems they were looking into is QNX [4], an operating system for in-vehicle entertainment and navigation. If compromised, this could the used for tracking and eavesdropping. In the documents no indication was found that prove the CIA actually succeeded in their attempts.

At this point in the development of many new features, like V2V communication, it is important to, next to the research on attacks, also look at automotive security from the defensive side. Defensive measures can be divided into two categories, they are either preventive or detective. Examples of preventive measures are a firewall, preventing unauthorised messages from the entertainment network to enter the power train network, or encryption, ensuring integrity of the traffic on the internal network. A detective measure could be deployed on the internal network to detect messages that managed to bypass the firewall. An Intrusion Detection System (IDS) is an example of a detective measure, its aim is to detect attacks on a network. One way of achieving this is by identifying anomalies in the network traffic. When an intrusion is detected an alarm is raised by the system. A large number of anomaly detection techniques have been proposed, but only a few have been designed specifically for CAN. Methods designed for other environments cannot be applied to CAN directly, they often have to be adapted to fit the specific characteristics

of CAN.

One thing to keep in mind when dealing with vehicles operated by human beings, is what to do with a message when it is flagged as an anomaly. Obviously, a message cannot always be blocked, e.g., when an anomaly is detected in a message controlling the brakes. One option that has been proposed is introducing a so called limp mode [28]. When an attack is detected, the car is put into limp mode by shutting down the network and all higher-level functions like power steering and lane assist. The car can be reset by restarting it. Another method that is used is simply signalling to the operator of the vehicle that an anomaly is detected, with a light or display [11].

In this thesis we will investigate the detection of a cyber-attack on a car's internal network. In particular, we will focus on the CAN bus. We will investigate network traffic on the CAN bus and analyse and compare IDSs designed specifically for CAN as well as general solutions. The end result will be an implementation of an intrusion detection system to detect cyber-attacks on a vehicle.

1.1 Related Work

A number of anomaly detection methods for automotive vehicles have been proposed in literature. The majority is designed for CAN, but applications for other networks have been designed as well. These methods are based on characteristics of in-vehicle computers and networks and each one has its own strengths and weaknesses. Next to CAN-specific IDSs there are also generic solutions. In this thesis both these solutions are analysed and compared.

One of the proposed devices is developed by Miller and Valasek [28]. The device detects abnormal messages in the simplest way possible, it looks at diagnostic messages that are being sent while driving and at normal messages that are being sent at a much higher rate than normal. These features are enough to detect all automotive attacks they executed on cars themselves with a very low number of false positives. When an anomaly is detected the car is put into limp mode. This device can only detect attacks that use one of these two methods, new types of attacks cannot be detected.

Miller and Valasek's method is not the only anomaly detection method for CAN based on frequencies of packets. Taylor, Japkowicz and Leblanc introduce a method that compares the inter-packet arrival times to historical averages [74]. Their method is able to detect packets inserted manually into a very simple dataset, but no realistic attack was imitated. Their dataset was collected from a modern car while driving for five minutes and no user controls, e.g. lights and windows, were used. Their method is limited as it only considers a subset of all recurring packets and no non-recurring packets.

A very interesting intrusion detection system for CAN is CIDS [11]. CIDS stands for Clock-based Intrusion Detection System, it is based on a property of any computer clock called clock skew. Clock skew is the characteristic that a clock tends to slowly drift away from a true clock over time. A normal computer resets its internal clock periodically via the internet, but this does not happen inside a car. Every ECU has a distinct skew and offset from a true clock and it turns out this can be used for fingerprinting. Thus, if a message that is typically sent by ECU A now is spoofed by an attacker that compromised ECU B, this can be detected, since the clock skews of these ECUs are different. This method of impersonating ECUs is exactly what attackers currently use in their attacks. CIDS is able to detect all attacks currently known in literature and has the possibility to detect more sophisticated attacks that will be developed in the future. However, we came up with an attack that tricks the entire system and enables an attacker to insert any malicious CAN packet.

CAN is not the only internal network for which an IDS has been proposed. Herold, Posselt, Hanka and Carle designed an IDS for the Scalable Service-Oriented Middleware for IP (SOME/IP) [32]. This protocol is an addition that runs on top of TCP/IP to facilitate the needs of an automotive network. One of the advantages of using this protocol is the ability to use standard components, thus giving the option to make a cheap in-vehicle network. The anomaly detection algorithm aims to detect malformed packets, protocol violations, system-specific

violations and timing issues, and achieves this using complex event processing.

As can be seen, each of the IDSs that have already been described in literature have their own strengths and flaws. Hence, the need exists to analyse and compare these and other methods of detecting automotive attacks. As will be clear by now, automotive security is still a young field, with a number of attack options known right now and it is too early to settle for one type of detection system. Luckily, so far all known attacks were published by researchers and right now there is no record of civilians being victims of malicious attacks. With the current development of many new communication features in vehicles, now is the time to improve modern vehicle's security and detection capabilities, before these features are widely introduced.

1.2 Problem Statement

The question that will be answered in this thesis is:

> *How can cyber-attacks on modern automotive vehicles be detected at CAN bus level using anomaly detection techniques?*

This question can be refined into two sub questions:

1. *What does a cyber-attack look like at CAN bus level?*
2. *Which anomaly detection algorithm gives the best results at CAN bus level?*

An automotive attack typically consists of a number of phases: exploit an external interface to gain access to the car's internals, compromise an internal component to gain access to the internal network and control the vehicle's functions by modifying network traffic. To give a complete overview of an automotive cyber attack, the external interfaces and exploit options will be described. The step of compromising an internal component, however, is out of scope for this thesis, as this is not visible to the IDS we are going to implement. Lastly, CAN traffic modification to gain control of a vehicle's functions will be described in detail, e.g. what attacks are possible, what attacks can be detected and what features of this data can be useful for detection. When we have this information we can answer the first sub question.

To answer the second sub question, a number of anomaly detection algorithms have to be studied. They will be chosen based on a number of different characteristics, e.g. their theoretical performance, how computationally and memory storage expensive they are and the amount of information they give about a potential anomaly. The chosen algorithms can then be trained using the features selected earlier. To validate and compare the constructed algorithms a number of different attacks will be simulated on a CAN data set by inserting, modifying or deleting packets. To acquire CAN data we will test a number of devices on different car models. The performance of the different algorithms can be compared by looking at their detection rate, false positive rate and other metrics. The best scoring algorithm will be chosen based on these metrics.

1.3 Thesis Outline

The steps described to answer the research sub-questions and ultimately the main question will be taken in the next chapters.

Chapter 2 gives background information about a modern automobile's external interfaces and internal networks, highlighting the ones relevant to this thesis.

Chapter 3 describes automotive attacks. All steps of an attack are explained from gaining access to the vehicle to actually controlling systems. A detailed analysis will be made of what an attacks look like on CAN bus level. At the end of this chapter the first sub question can be answered.

Chapter 4 provides information about intrusion detection systems, both in general as well as specific to CAN. We will select three anomaly detection algorithms that seem the most promising for application on CAN.

Chapter 5 gives a detailed explanation about the selected algorithms and the class of anomaly detection algorithms they belong to, one-class classifiers.

Chapter 6 describes the methodology that is used for the research given in this thesis. The required hardware and software will be described, as well as how to collect data, train the selected algorithms and how to compare their results.

Chapter 7 presents the results of the experiments described in the methodology. The results are analysed, the different anomaly detection algorithms are compared and the limitations are described. At the end of this chapter the second sub question can be answered.

Chapter 8 concludes this thesis by giving an overview of all the steps taken to answer the sub questions and the main research question will be answered. This chapter is concluded by discussing improvements for problems we identified and future work.

Chapter 2

Preliminaries

A modern car is a complex system with numerous components and connections to the outside world. All components are controlled by so-called Electronic Control Units (ECUs). To facilitate communication between ECUs there are various internal networks in place. Some ECUs are connected to interfaces that are able to connect to external systems. A car can be connected to other vehicles, to the infrastructure around it and even to the Internet.

In order to obtain a better understanding of modern cars, this chapter explains many different aspects starting with the internal networks connecting all ECUs. One of these networks is the Controller Area Network (CAN), we focus on CAN since this network will be used for anomaly detection in this thesis. Next, the external interfaces that are present on a modern car, used to facilitate connections to the outside world, are explained. On-Board Diagnostics (OBD-II) will be explained more detailed since this interface will be used for data collection. Lastly, a number of new technologies that are based on the techniques behind the interfaces are explained, some of which are still heavily under development.

2.1 Internal Networks

To connect all ECUs, a modern automobile contains a number of different types of internal networks, each with different characteristics, designed for different purposes. There is a gateway in place to facilitate communication between networks. An overview of a number of networks typically located in a car can be seen in Figure 2.1.

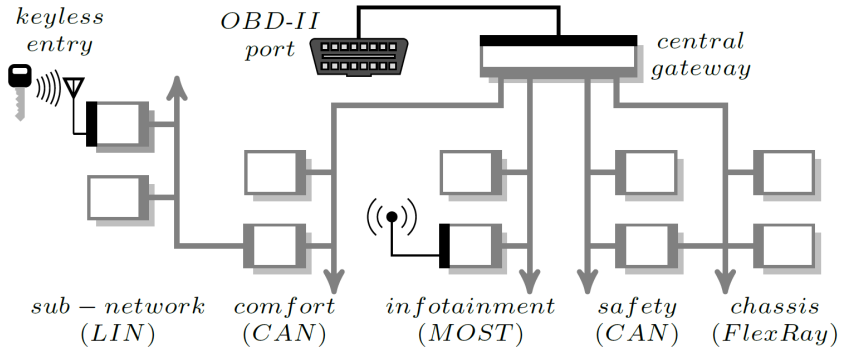

Figure 2.1: Networks inside a car [66]

The Controller Area Network (CAN) [7] is a very cost effective network. It only needs one simple twisted wire pair and offers many useful features like a high bandwidth and advanced error handling. The anomaly detection algorithms presented in this work will use CAN traffic. CAN

will be explained in detail in the next section.

The Local Interconnect Network (LIN) [39] is a serial network protocol that is used for components for which CAN is still too expensive to implement. These components, e.g. electronically controlled windows, do not require CAN's advanced features. Since nodes controlling these kinds of functionality are often on the CAN network, LIN is implemented as a sub-network of CAN in many vehicles.

The Media Oriented Systems Transport (MOST) [49] is a high-speed network used for multimedia appliances within the automotive industry. MOST has different characteristics than CAN and LIN since its purpose is to allow multimedia systems to communicate with minimum effort.

Another high-speed network is FlexRay [23] and, just as MOST, it is faster than CAN. However, FlexRay was not designed for multimedia, but for new technologies that increase comfort and safety, for which CAN does not offer the required performance. FlexRay was designed to be more reliable and faster than CAN, but is also more expensive.

2.2 Controller Area Network (CAN)

CAN is a communication system used within vehicles. CAN was developed by Robert Bosch GmbH and officially released in 1986. The most recent version published by Bosch is version 2.0 [7] and in 1994 the ISO 11898 standard [1] was published, making CAN an international standard. Nowadays, CAN is used in a wide variety of systems, e.g. medical instruments, agricultural machines and industrial control systems. For modern automotive vehicles the CAN bus has become the de facto standard.

CAN is a bus network which means that all devices on the network are connected to a single line, or bus. Before CAN was introduced a separate cable was required between every component that needed to communicate to each other. A new technology was required, simply because there was almost no space left inside the car and connectors were reaching their maximum size. Only needing one simple twisted wire pair, CAN reduced the size of the physical network significantly. In Figure 2.2 a schematic representation of CAN's improvement on the physical network can be seen.

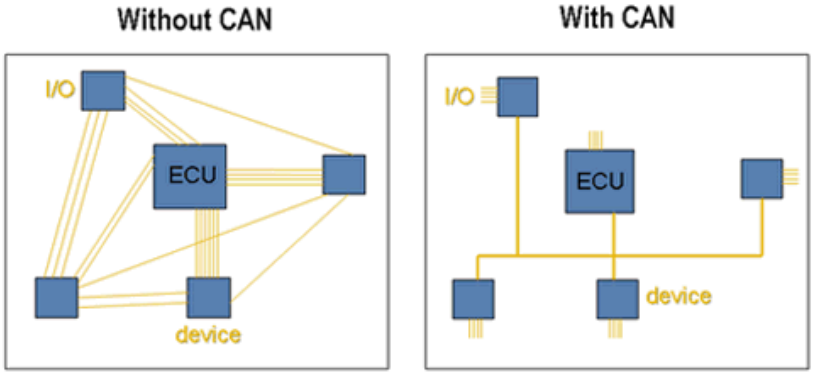

Figure 2.2: CAN's physical network improvements [34]

On a physical level the twisted wire pair consists of a CAN High wire and a CAN Low wire. When a bit with value 1 is sent, both wire carry 2.5 V. When a bit with value 0 is sent, the CAN High wire increases its voltage by 1.25 V, resulting in 3.75 V on CAN High. At the same time the CAN Low wire decreases its voltage by 1.25 V, resulting in 1.25 V on CAN Low. This way the average voltage on the wire is always 2.5 V, making CAN very resilient against electronic and magnetic fields.

The Bosch CAN specification version 2.0 consists of Parts A and B. Part A explains the standard CAN format and originally was the entire first version. In version 2.0 Part B was added,

in which different terminology was used for certain aspects and it includes the extended CAN format. In the extended format the identifier can have a longer value. One system implemented following part A and another system implemented following part B can communicate with each other as long as the extended format is not used. The layout of a CAN data frame can be seen in Figure 2.3.

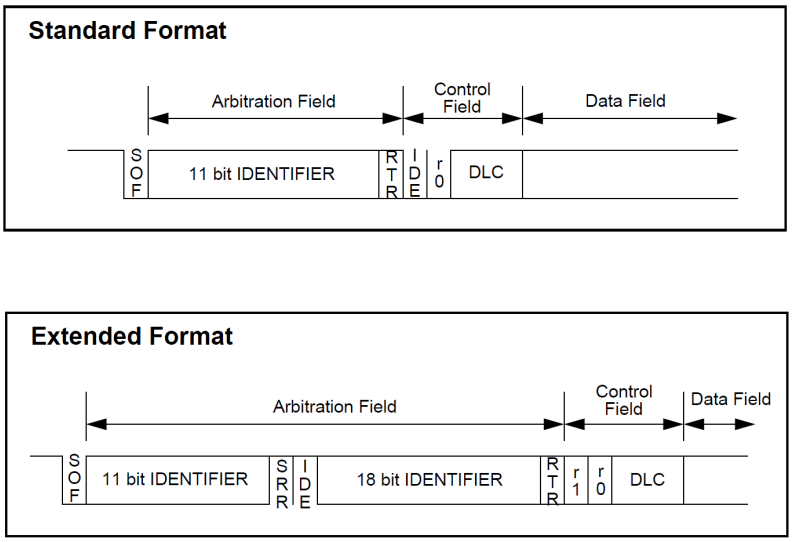

Figure 2.3: CAN data frames, standard and extended [7]

The control fields are used to define the message structure and for error detection. The data field can contain up to eight bytes of data, the length is given in the Data Length Code (DLC). A characteristic that introduces an interesting challenge for anomaly detection is that there are no source and destination fields in a CAN packet. Routing of a message is regulated by the ID field. A CAN packet is broadcasted to the entire network, but whether an ECU processes the packet depends on its input filter and the ID in the packet.

There are more advantages to using CAN next to decreasing the size of the physical network and versatile routing. CAN offers high-speed communication, up to $1\,\mathrm{Mbit\,s^{-1}}$. Due to the fact that CAN uses a simple twisted wire pair the costs are very low. It has a very efficient method of arbitration and it is extremely reliable due to many forms of error detection and correction. A number of these features will be explained in the next sections.

Message Filtering

The message filtering mechanism makes CAN very versatile. When a CAN message is placed on the bus, all ECUs will receive it. Whether the ECU processes the message or not depends on its input filter. This filtering can happen either at hardware or at software level. If filtering is done by hardware, there are preset programmable identifiers that are accepted. If none of these are matched, the message is rejected by the hardware.

In the case of software filtering, every ECU has its own acceptance mask and acceptance code. Typically these are eight bits long, only the eight most significant bits of the identifier are used for filtering. The acceptance mask specifies if the corresponding bit in the identifier should match the bit value in the acceptance code. A match is required when the acceptance mask bit value is equal to 0.

This way a message ID can be formed in such a way that the packet is processed by all ECUs (message broadcast), by a specific subset of ECUs (message multicast) or by only one specific ECU (peer-to-peer message).

Message Arbitration

CAN regulates arbitration in a very predictable and efficient manner. When a node on the network wants to transmit a packet, it first checks whether or not the bus is free. If it is free, transmission will start, otherwise it will wait until the bus becomes free. After transmitting a bit, the sender will check if the value that was sent is present on the bus. Important to note is that the 0 bit is dominant, meaning that if both a 0 and a 1 bit are transmitted on the bus at the same time by two different senders, the 0 bit will be placed on the bus. So it can happen that two nodes start transmitting at the same time, this will cause no problem as long as they are transmitting the same bit values. When the two nodes transmit a different bit value, the dominant bit will be placed on the bus, resulting in an error for the sender of the recessive bit. This sender will stop transmitting, wait until the bus becomes free again and retransmit the entire packet. The result is that the sender of the dominant bit is not aware there was a collision and will keep transmitting without delay, making CAN very efficient. This of course can also happen for more than two senders and will work exactly the same. Due to the fact that 0 is the dominant bit, messages with a lower ID will have priority over higher IDs.

An example of message arbitration with three senders is given in Figure 2.4. In this example we assume that no other ECU will attempt to send a message. Here it can be seen that all senders start their transmission without any problems. When sending the seventh bit sender A loses (A sends a 1, the others send a 0) and at the tenth bit sender B loses (B sends a 1 and C sends a 0). As a result the message with the lowest ID (310) is sent entirely by sender C. When senders A and B see that the bus is free again, they start retransmitting their entire packet. When sending the seventh bit sender A loses again (A sends a 1, B sends a 0) and sender B can transmit its entire packet. When sender A sees that the bus is free again it transmits its entire packet again and this time the entire transmission can be finished.

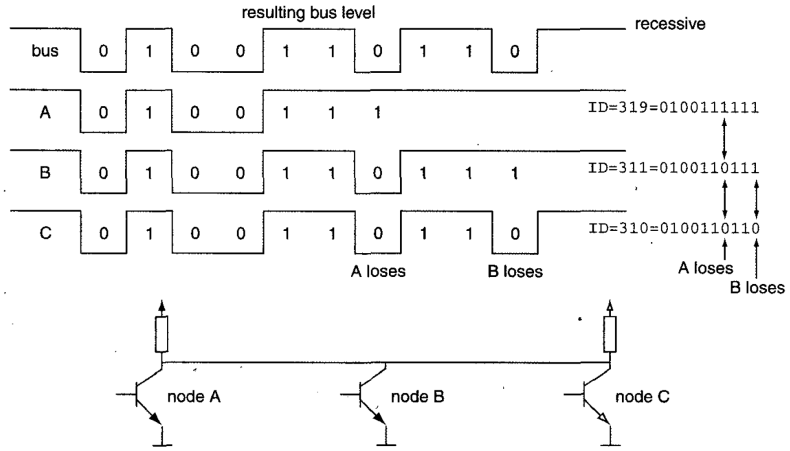

Figure 2.4: CAN message arbitration [20]

Error Detection and Correction

As mentioned before, when an ECU transmits a bit, it also listens to the bus whether the bit that was sent is also on the bus, when this is not the case this is called a **bit error**. When a module has a bit error, transmission is stopped. The module will keep listening to the bus until it becomes free. When the bus is free, the module will retransmit the entire package that was being sent when the bit error occurred.

When five bits with the same value are transmitted after each other, a bit with opposite value is transmitted, this is called bit stuffing. So after five 0 bits a 1 bit needs to be sent. When a module reads six bits with the same value from the bus this is called a **bit stuffing error**. Next to error detection, bit stuffing is also used for resynchronisation.

Every CAN packet contains a 15-bit **Cyclic Redundancy Check** (CRC), this value is the result of a CRC calculation by the sender. This value is also calculated by the receivers of the packet and compared to the value inside the data packet. When the two values are different an error has occurred.

In Figure 2.3 the layout of a CAN data frame is given. When a CAN packet is received it is checked by the receiver whether all parts have the correct size. If this is not the case, this is called a **form error**.

When an error is detected by any node on the network, regardless of whether the packet was intended for this node, an active error frame is sent. An active error frame consists of six dominant bits and causes a bit stuffing error for the sender of the erroneous message and thus forces this sender to stop transmitting.

Architecture

CAN's architecture can be explained using two layers of the ISO/OSI model. The OSI model is a reference model for network architectures [80], it serves as a framework for the definition of protocols. It is made up of seven layers: Physical, Data Link, Network, Transport, Session, Presentation and Application Layer. CAN only implements functionalities of the Physical and Data Link layer.

Obviously, there is a Physical Layer. The Data Link Layer consists of two sublayers in the CAN specification, the Logical Link Control (LLC) and the Medium Access Control (MAC). Figure 2.5 gives the functionalities of these layers, the functionalities relevant to this thesis will be explained.

The Physical Layer defines how signals are transmitted over the physical wire. This layer handles Bit Encoding/Decoding and Bit Timing, these are concerned with the actual handling of the physical bits on the wire. This layer also handles Synchronization to ensure all communicating ECUs read the same bits at the same time. Bit stuffing is one feature that is used for this.

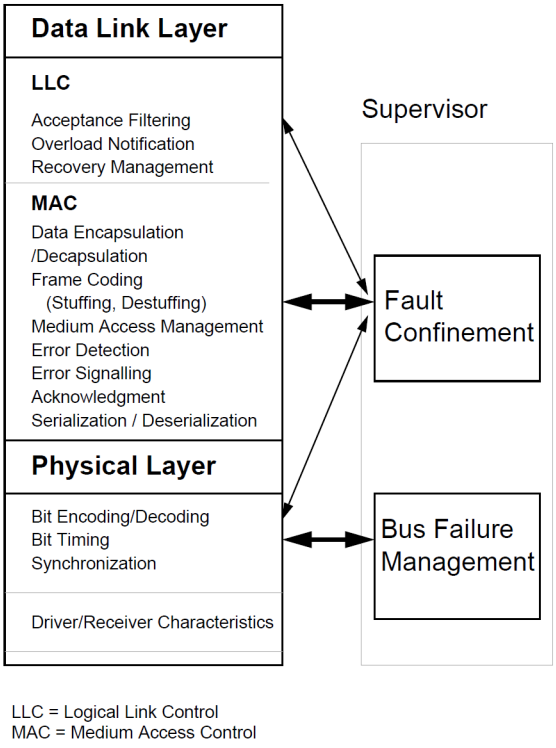

Figure 2.5: CAN architecture according to the OSI model [7]

The MAC sublayer receives messages from and transmits messages to the LLC sublayer. This

layer handles Message Framing to ensure all CAN packets are well-formed and have the correct size. Arbitration is handled by this layer as explained before. This layer also handles the Acknowledgement of the receipt of a message, there is an acknowledge slot in every packet for this. All Error Detection methods that were explained before and Error Signalling are handled by the MAC sublayer as well.

The LLC sublayer handles Messages Filtering, how this works is explained at the beginning of this section. The fact that acceptance filtering happens at the LLC sublayer and error detection happens at the MAC sublayer can lead to the situation where an ECU that is not an intended receiver of a packet raises an error, e.g. a form error.

2.3 External Interfaces

Some ECUs require functionality or data given by an interface that connects to the outside world. A modern car has many of these external interfaces. Functionality offered by these interfaces range from comfort to safety. An overview of all external interfaces can be seen in Figure 2.6. The coloured interfaces roughly group ECUs based on their function.

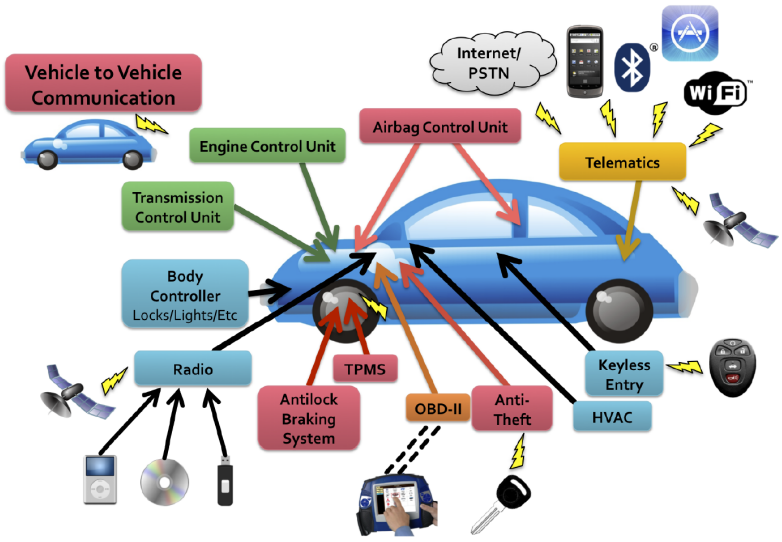

Figure 2.6: External interfaces on a car [10]

The interfaces can be grouped into three categories, as proposed in [10]: physical, short-range wireless and long-range wireless interfaces. In this section the interfaces will be explained from a technical point of view. In Section 3.2 the interfaces will be looked at from an attacker's point of view and their option for intrusion will be discussed.

2.3.1 Physical Interfaces

Inside a modern automotive vehicle there are multiple physical interfaces, some of them are directly connected to an internal network. OBD-II is the most well known connector, detailed information will be given in the next section. In this thesis OBD-II will be used for data collection.

The entertainment system inside a car provides the other physical access points, through discs and USB drives. Next to entertainment systems, navigations systems are sometimes updated using a CD.

On-Board Diagnostics (OBD-II)

One of the first automotive diagnostics interfaces was introduced in the late 1970's and was called Assembly Line Diagnostic Link (ALDL). Different diagnostics interfaces were created after ALDL, often created for a specific region following local standards and legislation. In the 1990's OBD-II was introduced. Also for this interface different implementations exist, with differences between countries. The European equivalent of OBD-II is the European On Board Diagnostics (EOBD) [15]. It is required that the port is reachable by the driver, it should be placed within two feet of the steering wheel.

In this thesis the OBD-II port will be used to connect to the CAN bus to read the traffic. This data will be logged to create a dataset containing normal driving behaviour. The port has the form of a 16-pin connector, the diagram is shown in Figure 2.7 and the layout of the connector can be seen in Table 2.1. The coloured pins in the figure correspond to the coloured cells in the table, white pins have vendor specific functionality. Two pins with the same colour belong to the same protocol. As can be seen in the table, CAN is connected to the OBD-II connector on pins 6 and 14, CAN High and CAN Low. The functionality of CAN High and CAN Low was explained in Section 2.2.

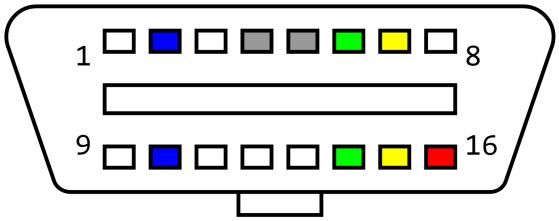

Figure 2.7: OBD-II connector diagram [78]. The colours correspond to the cells in Table 2.1.

Pin	Signal	Description
1		Vendor specific
2	SAE J1850 positive	Diagnostics
3		Vendor specific
4	CGND	Chassis ground
5	SGND	Signal ground
6	SAE J2284 bus	CAN High
7	ISO 9141-2 K-LINE	Tx/Rx
8, 9		Vendor specific
10	SAE J1850 negative	Diagnostics
11, 12, 13		Vendor specific
14	SAE J2284 bus	CAN Low
15	ISO 9141-2 L-LINE	Tx/Rx
16	+12 V	Battery power

Table 2.1: OBD-II connector pinout [60]

The other networks that were mentioned before do not appear in the table for two reasons. LIN is often implemented as a sub network of CAN and can be accessed from the OBD-II port via CAN but not directly, this is also illustrated in Figure 2.1. FlexRay is a relatively new network and thus if a manufacturer wants to implement this network one of the vendor specific pins can be used. A network like MOST, that focuses on multimedia, does not have to be connected to a diagnostics port.

In many attacks that have been demonstrated by security researchers access to the car's internal network was gained via the OBD-II port. However, as discussed before, cyber-attacks on vehicles that are executed while being connected to the OBD-II port are not seen as a risk by car manufacturers, as they require an attacker to be present inside a car. These attacks are nevertheless very useful for finding vulnerabilities inside the car, since it is only a matter of time before a new remote attack is found. Therefore, finding a vulnerability via the OBD-II port gives car manufacturers the opportunity to fix it before a new remote attack is disclosed. This is the reason that in our attack analysis we investigate attacks regardless of the way the attacker was connected to the internal network.

2.3.2 Short-Range Wireless Interfaces

Short-range wireless interfaces are used by a car to communicate with objects in its near surroundings. The range of these kinds of interfaces is typically about 10 metres, but for some this can get as large as a few hundred metres.

Many car models support Bluetooth communication, primarily for handsfree calling or multimedia purposes. Bluetooth is an open technology, managed by the Bluetooth Special Interest Group [6]. When a manufacturer wants to market a device as a Bluetooth device it needs to comply to the demands of the SIG. Bluetooth communication happens at a frequency of 2.4 GHz [5].

Radio frequency (RF) technology is used for a number of applications in a car. Remote Keyless Entry (RKE) is based on RF. This is a feature that is present in almost all modern vehicles. It is used to remotely open the doors and trunk by pressing a button on the key fob. Some cars have a proximity system that allows the user to open the car when the key fob is within a certain distance of the car. Some higher level models offer functions like starting the engine. Another application of RF technology is an RF Identification (RFID) chip inside a car's key fob to prevent unauthorized operating of a vehicle. Since 1998 these immobilisers have been mandatory by law in many European countries. The purpose of this device is to make sure the correct key is present in the car's ignition lock or near its steering wheel when the car being is operated.

In the United States of America a Tire Pressure Monitoring System (TPMS) is mandatory on all cars manufactured or imported after 2008 [51]. Similar legislation was introduced in the European Union [16]. The system monitors tire pressure and reports real-time information to the driver of the vehicle. The driver is informed using a gauge or warning light.

A new development is the addition of a Wi-Fi hotspot to a car. Wi-Fi itself is a technology widely applied nowadays for many different applications. Wi-Fi is based on the standard IEEE 802.11 (Wi-Fi) [17]. The range of Wi-Fi can be extended to a few kilometres, but this requires special antennas and amplifiers. As an attacker would have to modify the router inside the car for this we consider Wi-Fi to be a short-range interface in this thesis.

2.3.3 Long-Range Wireless Interfaces

Long-range channels have a range of up to almost an infinite distance. This type of interfaces is divided into two categories in [10], broadcast channels and addressable channels.

Broadcast channels include all types of wireless channels that are widely broadcasted by the transmitter and can be tuned into by a car. Examples of these channels are Global Positioning System (GPS), Satellite and Digital Audio Broadcasting Plus (DAB+), and the Radio Data System (RDS) and Traffic Message Channel (TMC). The range of these signals can be as large as 10 km.

Addressable channels are systems that have a cellular data connection. Many automotive companies have built their own proprietary systems. Supported features these systems aim to

provide include safety, diagnostics, anti-theft, entertainment and convenience. The range of these systems in linked to the cellular tower coverage, two entities that are on opposite sides of the planet can communicate as long as they both have a cellular data connection.

2.4 Applications

There are a number of modern technologies that are connected to external interfaces or make use of internal systems. For a potential attacker these technologies can be very interesting since they (in)directly expose the vehicles internal network. In this section two of these technologies will be explained, emergency assistance and vehicle-to-vehicle (V2V) communication. Exploitation of these technologies, to gain access to a car's internal network, will be explained in Chapter 3.

2.4.1 Emergency Assistance

On new Ford models an option called emergency assistance is available [13], other car manufacturers provide similar options. This feature is developed to help rescue people in case of an accident. When an air bag is activated the emergency services are called automatically and important information, like GPS data, is communicated to the rescue team. If the driver is conscious, the call can be cancelled or he can talk to the emergency services. In the case that the driver is unconscious, this system can alarm a rescue team and could possibly save the victims life.

To be able to execute these functionalities, this system needs to have access to a number of interfaces, as well as to numerous internal systems to gather data. Via the air bag controller the system can monitor when an accident happens. The cellular connection is needed to connect to the emergency central. The GPS interface is required to send location information. Possibly other information is sent, e.g. fuel level to indicate if there is a leak in the fuel tank. Next to the possibility of saving lives, this system adds new entry points for an attacker. Once compromised, it can be easier for the attacker to gain access to various internal systems.

2.4.2 Vehicle-to-Vehicle (V2V) Communication

A recent development, which is already present in a number of high-end vehicles and is expected to be used by mainstream consumers within a few years, is Vehicle-to-Vehicle (V2V) communication. V2V makes use of a Vehicular Ad Hoc Network (VANET), which is the automotive variation of Mobile Ad Hoc Network (MANET). This technology is based on the IEEE 802.11p [35] standard, which is a modified version of IEEE 802.11 (Wi-Fi) [17]. Vehicle-to-Infrastructure (V2I) offers the same functionality, but instead of between two cars, a car can communicate with its surroundings. Vehicle-to-Everything (V2X) is a broader term and means communication between a car and any other entity.

V2X is an important building block of a modern Intelligent Transport System (ITS). An ITS offers communication between vehicles without the need for a central infrastructure. Vehicles will communicate directly with other cars that can be out of range of sight and when using a multi-hop network even communication with vehicles that are out of range of radio is possible.

The CAR 2 CAR Communication Consortium (C2C-CC) has the goal to develop standards for interfaces and protocols that facilitate V2X communication in Europe [8]. After these standards have been developed, cars produced by different manufacturers can communicate with each other, as well as with the infrastructure.

There are many possible applications for V2X, most of them are focussed on safety. When a car brakes, a message can be sent to all cars in its proximity. An indication to brake can then be shown to the driver, or in the case of a smart vehicle it can brake automatically. Since this also works when the initial car is out of sight for the driver, e.g., because of weather conditions or other traffic, accidents can be prevented.

A possible solution for the ever increasing amount of cars on the road is called platooning. Platooning allows cars to drive behind each other with a very close proximity, resulting in a 'train'

of cars. The cars can exchange acceleration, braking and steering information to avoid collisions. Using platooning the existing road infrastructure can be used with a much higher car density and thus a higher efficiency.

A feature that is not focussed on immediate collision prevention is the exchange of very accurate location information. Cars can report their location, the location of vehicles around them and the location of obstacles on the road. The other way around the infrastructure can give information to vehicles, e.g., about road works. This information gives drivers the possibility to choose an alternative route well on time, before traffic jams occur.

There are also a number of challenges concerned with these technologies. First of all the existing communication protocols, like MANET and IEEE 802.11 are being adapted to fit the situation. There are many similarities, but there are a number of details specific to the use within cars and infrastructure. The nodes on the network can move with a very high speed, e.g., when driving on the highway. Next to the speed, the high number of nodes is also a difference. An upside of this situation is that the nodes follow more predictable routes than the devices MANET was designed for, since they have to follow roads. Lastly, the protocols should be very reliable and have minimal delay, since lives can be at stake in this situation. While these challenges could introduce security risks, these issues are out of scope for this work. Our goal is to detect attacks on cars by investigating CAN traffic.

Chapter 3

Automotive Attacks

Miller and Valasek's attacks described in Chapter 1 show that there are many possibilities for attacks against automotive vehicles. The results of these attacks range from seemingly innocent actions like turning on the air conditioner, to harmful actions like killing the engine or turning the steering wheel of a driving car. To be able to execute these types of attacks an attacker needs to execute a number of steps that range from getting access to the car's internal networks through an entry point to controlling systems inside the car.

The Intrusion Detection System (IDS) proposed in this work only investigates network traffic, and thus only focusses on the last step of an automotive attack. In order to test the IDS, we require data sets containing attacks. For this, we are going to simulate a number of different attacks on a data set recorded from a car under normal driving conditions. Therefore, we need to know the characteristics of such attack. Once we know this we can also answer the first subquestion that was described in Chapter 1: *What does a cyber-attack look like at CAN bus level?* Nevertheless, we will explain the entire process of executing an automotive attack in order to give a clear overview of all steps necessary to get to this point where CAN traffic can be manipulated.

3.1 Terminology and Concepts

A complete attack on a car consists of a number of steps. It starts with compromising an external interface, this is the entry point to the car's internals for the attacker. As explained in Section 2.3 there are many different interfaces with various characteristics. These interfaces are connected to an Electronic Control Unit (ECU) which is connected to other ECUs via various internal networks, these have been described in Section 2.1. When an ECU is compromised it can be used to send data on an internal network to other ECUs and influence their behaviour. In this chapter the Controller Area Network (CAN) is the only network we focus on as the IDS we are going to develop aims to detect attacks by solely investigating CAN traffic.

3.2 Entry Points

The first step of an automotive attack is to exploit an external interface to gain access to the internal systems of the vehicle. In Chapter 2 a number of external interfaces have been described, in this section we will describe how these interfaces can be leveraged to gain access to a car's internal network. The interfaces present on modern cars have different ranges, from physical access like OBD-II to remote access over an almost infinite distance via a cellular connection. All these interfaces have different risks when looking at automotive cyber-attacks. For an attack via the OBD-II port the attacker needs to be present inside the car for the duration of the attack, or the attacker needs to be able to remotely communicate to the malicious OBD-II device. For an attack via the Wi-Fi hotspot, the attacker has to follow the victim and stay close during the entire duration of the attack. An attack via the cellular connection however, can be executed as

Interface	Category	Typical range
OBD-II	Physical	0 m
USB, CD	Physical	0 m
Immobiliser	Short-range wireless	1 m
Bluetooth	Short-range wireless	10 m
TPMS	Short-range wireless	10 m
RKE	Short-range wireless	20 m
Wi-Fi hotspot	Short-range wireless	100 m
V2X	Short-range wireless	300 m
Radio	Long-range wireless	10 km
GPS	Long-range wireless	$>$ 10 km
Cellular	Long-range wireless	$>$ 10 km

Table 3.1: Overview of attack entry points

long as the attacker and victim have a working internet connection. Interfaces with a longer range generally have a convenient aspect for an attacker as it is easier for the attacker to preserve the connection during the full attack. An overview of all entry points with their range is given in Table 3.1. The range shown in this table is the distance under normal operating circumstances, the attacker can use directed antennas and amplifiers to increase the range.

3.2.1 Physical Access

Inside a modern automotive vehicle there are multiple physical interfaces, some of which are directly connected to the internal network. OBD-II is the most well known connector and is used by many security researchers to find and execute automotive attacks.

The entertainment system inside a car provides the other physical access points, through discs and USB drives. Often the entertainment system is connected to the CAN bus. If an attacker can put an exploit on a CD, or puts an exploit in a corrupted song on a USB drive, and can convince a victim to play this in his car, the media player can be compromised. Often the entertainment system is connected to the CAN bus and the attacker can get access to the car's internal network. Next to entertainment systems, navigations systems are sometimes updated using a CD; here the same vulnerabilities apply.

Physical access points are a very useful tool for security researchers to gain access to a car's internal network. For malicious attackers however, these access points are less attractive as they require the attacker to either be very close to the car or to the driver.

3.2.2 Short-Range Wireless Access

Wireless access is more convenient for the attacker since no cable is needed and the victim does not have to be convinced to use an infected CD or USB stick. Short-range wireless access does require the attacker to be in close proximity of the vehicle however and he needs to stay there during the entire duration of the attack. The range of these kinds of access points is typically about 10 metres. For some access points this can get as large as 300 metres, although sometimes this does require using enhancement techniques.

Typically Bluetooth has a range of ten metres, but it has been shown that the range can be

extended by using antennas and amplifiers. Bluetooth is typically used to control an entertainment system and as previously seen for the physical access points these systems are often connected to the car's internals.

A TPMS poses a number of concerns [62]. Both security and privacy related problems have been discovered. Sensor messages could be spoofed due to vulnerabilities in the communication protocol, causing false messages to the driver. Tracking a vehicle is possible from about 40 metres, which is a serious privacy risk.

Vulnerabilities in RKE and RFID, two RF based features, can give attackers the option to illegitimately open doors of a car or start the engine [25].

A Wi-Fi hotspot in a car usually has a larger range than conventional Wi-Fi networks used indoors, since there are no walls and floors to weaken the signal. Since conventional Wi-Fi has had many applications over the last years and is not purely focused on automotive application, there already has been a lot of research into vulnerabilities present in these protocols. However, even if the Wi-Fi protocol would be completely secure, there could still be errors in the implementation that can be exploited by an attacker.

V2X communication is based on the same technology as Wi-Fi hence the range is the same. This technology communicates information about a car to its surroundings. For example, when a car brakes suddenly, a signal can be sent to the car behind it to break too, to avoid a collision. This information is often retrieved from internal sensors and controllers and thus makes V2X an interesting access point for an attacker. When one car is compromised, malicious messages can be sent, causing traffic accidents instead of preventing them.

In this category the Wi-Fi hotspot and V2X communication would probably be the best entry points from an attacker's point of view, since they are usually connected to many internal systems. TPMS can be used for tracking.

3.2.3 Long-Range Wireless Access

When using short-range wireless access to connect to the victim's car it can be a problem for the attacker to stay close enough to the vehicle to execute an entire attack when a car is moving. Long-range wireless channels have range of up to almost an infinite distance. This type of access channels is divided into two categories by [10], broadcast channels and addressable channels.

Depending on the source of the broadcast channel, the range can be as large as 10 km in the case of radio towers. For satellite-based systems like GPS, the range is larger as the entire planet can be covered. These systems typically are processed by the entertainment or navigation system and as previously seen, an attacker can often get access to the CAN bus from these systems.

The range of addressable channels is very large, due to the already available high cellular coverage. Many systems already have a cellular connection and this amount will probably only grow. These connections offer the most options to an attacker, due to their versatile functionality and the high number of access points to the internal network.

3.3 Analysis of Existing Attacks

After an attacker has gained access to a car's internal network, e.g. via any of the entry points described above, an attack on the internal network can be launched. The goal of any attack is to influence the behaviour of an ECU by modifying the traffic on the CAN bus. This can be done by suspending an ECU from sending messages, modifying the messages an ECU sends or fabricating new messages. These modifications to the traffic can prohibit an ECU from executing its normal behaviour or change the behaviour to the attacker's wishes. ECUs critical to a vehicle's functioning can completely shut down when it receives messages that are not exactly what is expected.

The attacks by Miller and Valasek described in Chapter 1 make use of a number of different techniques. One of them is using diagnostics messages. These messages are used by mechanics when trying to find a problem in a car or solve a fault. They can be used to control the steering, brakes and engine. In recent model vehicles however, there are checks implemented to prevent

diagnostics messages from being sent while driving normally, e.g. when the speed is above a certain threshold, or when the engine is running.

One attack they executed to control the speedometer, was simply sending multiple forged messages between two consecutive appearances of a real message. The forged message can indicate a different velocity than the real value. If the forged message is sent often enough, the legitimate speedometer messages will become a small percentage of all messages and the real velocity will not be visible on the dashboard.

To control the steering wheel of a car, Miller and Valasek want the ECU controlling the steering module to only read their forged messages and ignore the real messages [48]. The manufacturer's attempt to protect the ECU against attacks is the addition of a counter value to the packets that has to be increased by one with each new message. When the difference between two consecutive counter values is not equal to one, the ECU ignores the second message and a flag is set. After that, when the next message arrives and the ECU reads a wrong counter value again, the ECU turns itself off. On the other hand, when the counter value in the next message is correct the flag is reset. Hence, when an attacker replays a message, the flag is set and the replayed message is ignored. Miller and Valasek trick the ECU by sending their forged message right before the real message and give the forged message the same (correct) counter value x as the real message would have. Since the forged message arrives earlier, the ECU checks the counter value x, confirms it is correct and reads the forged message. When the real message is received, the ECU expects the message to have counter value $x + 1$. However, this message has counter value x and the message is ignored and the flag is set. The next message is again a forged message and it has the increased counter value $x + 1$, thus the ECU checks the counter value and confirms it is correct, reads the forged message and the flag is reset. The following real message, however, also has counter value $x + 1$ and thus the ECU ignores the message and sets the flag. By repeating this, in the end, all forged messages are read, all real messages are ignored and the flag has constantly been set and reset without ever reaching the maximum amount of conflicts.

In another attack they were able to have an ECU only receive their forged messages. In the case where ECU A sends messages that are being processed by ECU C, they were able to suspend ECU A from sending messages. For this they reprogrammed ECU A with new firmware, halfway through reprogramming however, they stopped. This results in ECU A being partially programmed and thus it cannot function properly. ECU B was compromised as well and used this ECU to send forged messages.

One attack in [33] targets electronic windows, it observes the CAN network for a message that signals the concerning ECU to open the window. Immediately after this message, a forced message is sent, signalling to close the window. As a result the window stays closed. The same principle is used to prevent a warning light on the dashboard to light up when modified CAN messages caused a problem and signal a user that something is wrong. When a "warning light on"-message is sent, a "warning light off"-message is sent immediately after. Resulting in the light not turning on or glowing very faintly.

3.4 Attack Taxonomy

Based on the attacks described above a model can be described containing two attacker types and three different kinds of basic attacks. All attacks described above are based on these three basic attacks. This model can be used to simulate attacks on a CAN dataset that we will use to test our anomaly detection algorithm. This will give us realistic data sets reflecting attacks that are currently known in literature. As the IDS we are going to develop will only investigate recurring messages, we will only describe attacks that affects this type of messages.

3.4.1 Attacker types

Cho and Shin consider two types of attackers for the verification of their IDS [11]: *weak* and *strong* attackers. A weak attacker is able to compromise an ECU in such a way that the transmission of

messages can be stopped or paused, e.g., by putting the ECU in listen only mode. A strong attacker has access to an ECU's memory and has full control over the ECU. Hence, a strong attacker is, next to the capabilities of a weak attacker, also able to fabricate and transmit messages. Instead of weak and strong attackers, the terms weakly and fully compromised ECU, respectively, will be used interchangeably. Which type of attacker an adversary can become depends on multiple factors, e.g. hardware and software of the concerning ECU and the skill of the adversary. Based on this model, three attack scenarios are described in [11]: fabrication, suspension and masquerade attacks.

3.4.2 Fabrication Attack

A fabrication attack is carried out by fabricating messages with a forged ID and data and inserting these on the network. Since messages have to be forged, this attack has to be executed by a strong attacker. If ECU A normally sends a message with ID 0xA1, fully compromised ECU B can fabricate a message with ID 0xA1 and custom data. As a result, receiving ECU C gets conflicting messages, resulting in either giving ECU A control over its actions or making ECU C inoperable. The timing with which the fabricated messages are sent can be divided into three categories. One or more fabricated messages are inserted with a higher frequency than the original message, here the time the original message is sent is not taken into account. The fabricated messages can also be inserted either immediately before, or immediately after the occurrence of the original message.

The steps of simulating a fabrication attack on a CAN dataset are as follows:

1. Select an ID, e.g. 0xA1, that periodically shows up in the network traffic
2. After time t_{fab} take either one of the following three steps
   - 2.1 Insert one or more fabricated messages with ID 0xA1 with an increased frequency, resulting in equal interval times between fabricated messages, regardless of the appearance of the original messages
   - 2.2 Right before every occurrence of the original message send one fabricated message with ID 0xA1
   - 2.3 Immediately after every occurrence of the original message send one fabricated message with ID 0xA1

3.4.3 Suspension Attack

A suspension attack is simply executed by stopping ECU A from sending its messages. Thus, only a weak attacker is required to execute this attack. This can have two results, first of all the functionality normally controlled by ECU A will not be executed. Additionally, there are ECUs that rely on incoming data to function normally. So if ECU B requires information from ECU A and a suspension attack is executed on ECU A, ECU B will not function properly.

The steps of simulating a suspension attack on a CAN dataset are the following:

1. Select an ID, e.g. 0xA1, that periodically shows up in the network traffic
2. After time t_{sus} delete the messages carrying ID 0xA1

3.4.4 Masquerade Attack

The most complicated and also the strongest attack is the masquerade attack. For this attack two ECUs are required, ECU A should be fully compromised and ECU B should be weakly compromised. The goal of the attack is controlling the actions of one ECU without giving away the fact that this ECU has been compromised. To carry out the attack ECU A starts by observing the messages sent by ECU B, e.g. a message with ID 0xA1 is sent every 20 ms. After time t_{mas} the attacker makes ECU B stop sending messages and ECU A starts sending messages with ID

0xA1 at the same frequency. When observing the traffic on the CAN bus nothing seems to have changed, but these messages are now being sent by ECU A instead of B, giving the attacker control over the data in the messages. This attack can be executed when the desired ECU, i.e. the ECU sending the messages that the attacker wants to modify, in the example this is ECU B, can only be compromised as a weak attacker.

When simulating this attack on a CAN dataset, the fabricated message should not be inserted at exactly the same time as the original message would have been sent. A very small time offset should be added to account for differences in skew and network propagation delay. Cho and Shin's measurements show that a clock's skew can range from 30 µs to 460 µs [11].

The steps of simulating a masquerade attack on a CAN dataset are the following:

1. Select an ID, e.g. 0xA1, that periodically shows up in the network traffic
2. After time t_{mas} block the messages carrying ID 0xA1
3. From time t_{mas} insert a fabricated message at the same frequency as the original message

3.5 Conclusion

In Chapter 1 we described the research questions we want to answer in this thesis. Using the information given so far, we are able to answer the first sub question: *What does a cyber-attack look like at CAN bus level?*. In Chapter 2 we explained the various internal networks in a modern car and for CAN we provided a more detailed explanation. In this chapter we analysed the characteristics of an attack on an automobile and we ended our analysis with the effects that can be observed on the CAN bus. Combining all these aspects, we can conclude that a cyber-attack, when observed at CAN bus level, looks like a modification of CAN packets, i.e. CAN packets are inserted, deleted or modified. The way in which this happens can be divided into three groups, which we identified as the fabrication, suspension and masquerade attack.

Chapter 4

Intrusion Detection Systems

To detect the various types of attacks possible on CAN, a number of Intrusion Detection Systems (IDSs) have been proposed in literature. The aim of an IDS is to detect and possibly block attacks on a system or network. Typically, the detection method used by these systems can be divided into two categories; they are either signature-based or anomaly-based. The general idea of these two types of IDSs will be analysed and compared. Next to general algorithms, a number of systems have been designed for CAN specifically. After analysing all differences we conclude this chapter by explaining which type of intrusion detection algorithm is best fitted to implement in this thesis.

4.1 Terminology and Concepts

An intrusion is any unauthorised action that is executed on a system or network. The aim of an IDS is to detect intrusions. A common distinction that is made in how an IDS functions is between host-based and network-based intrusion detection. A host-based IDS runs on a host or individual device and monitors incoming and outgoing traffic of that device. It alerts when suspicious traffic is observed. Besides traffic, it also monitors files on the system. When critical system files are modified or deleted an alert is given. A network-based IDS monitors traffic on the network between all hosts. An alert is given when abnormal behaviour is observed. We will focus on network-based IDSs in this chapter, since that is also the kind of defence mechanism that will be proposed in this thesis.

An IDS typically only has detection capabilities. When an intrusion is found an alert is given, but no other actions are taken. In a car this could be visualised in a display or notification light. There are IDSs however, that also have prevention capabilities, so called Intrusion Prevention Systems (IPSs). When an intrusion is detected the IPS will take action to prevent the intrusion from continuing. This could be dropping all network packets from the same source or all packets that have similar characteristics. When dealing with a critical environment like a car however, messages that are flagged as anomalous cannot just be blocked, e.g., when the message concerns the brakes. A possible action that has been proposed is the so called limp mode [28]. When an attack is detected, the car is put into limp mode by shutting down the network and all higher-level functions like power steering and lane assist. When in limp mode, a car should be reset by restarting it.

4.2 Signature-Based IDS

Intrusion detection systems are frequently based on signatures. A signature is a unique identifier of an entity, e.g., a file signature uniquely identifies the contents of a file. A signature-based IDS for networks operates in the same way as many antivirus programs, which uses signatures to detect malware. A signature can be created by calculating a file's hash value. A hash is a numerical value that is the result of applying a so-called hash function to a string of text, a file in this case.

The calculated hash is then compared to a large database of hashes of known malware. If there is a match, the file is flagged as malware. For an antivirus program to keep detecting new malware, the signature database needs to be updated frequently.

Signature-based IDS for a network generally comes with a set of rules, or signatures, that define the characteristics of an attack. When behaviour on the network matches a signature, an alarm is raised and typically the message is blocked. Just as for an antivirus program, the false-positive rate, i.e. the amount of normal traffic that is flagged as an intrusion, is low. However, new attacks can obviously not be detected by a signature-based approach.

4.2.1 Systems for CAN

Miller and Valasek developed an intrusion detection device for CAN that uses two methods [47]. One of these methods is signature-based. Their method aims to detect intrusions by looking for diagnostic messages that are being sent while driving. Normally diagnostics messages are being used by mechanics to find and repair electronic problems in a car. Attackers can use these types of messages to control systems in a car, e.g., steering. It is obvious that this type of message should not appear in CAN traffic while driving. A diagnostics message that appears during normal driving could be seen as a rule the IDS uses to detect intrusions. Next to this signature-based method, their device also has an anomaly-based method, this type of detection will be explained in Section 4.3.1. This method monitors network traffic for normal CAN messages that are being sent with a higher frequency than normally.

Their device also has intrusion prevention capabilities. When an attack is detected the car is put into limp mode. This means that all higher level functionalities, such as power steering and ABS, are turned off. This way the driver can get the car off the road safely and restart the car to turn all functionalities on again.

As with many signature-based systems, this method has a very low number of false positives. Miller and Valasek claim their device is capable of detecting all attacks they are able to execute themselves as well as attacks currently known in literature.

4.2.2 Shortcomings

Miller and Valasek's claim about the performance of their device is based on the assumption that an attack consists of either diagnostics messages or normal messages with an increased frequency. As we saw in Section 3.4, more types of attacks have been identified. Their assumption does not include suspension and masquerade attacks.

Their system is the only IDS for CAN we investigated that also has intrusion prevention capabilities. However, it should be noted that this feature can also be abused by an attacker and result in a Denial of Service (DoS) to a car.

Next to flaws specific to this IDS, a signature-based IDS will never be able to detect new attacks that will be proposed in the future. The fact that a number of IDSs have been proposed in literature that are already capable of detecting many known attacks has lead us to decide to not implement a signature-based IDS in this thesis, as it would not be able to detect more than is currently already possible.

4.3 Anomaly-Based IDS

Where a signature-based IDS looks for known characteristics of malicious packets in network traffic to detect an attack, an anomaly-based IDS looks for network traffic that deviates from a known model. This model describes the characteristics of network traffic under normal circumstances. When behaviour on the network does not follow this model an alarm is raised. As opposed to a signature-based IDS, anomaly-based systems are able detect unknown attacks, but tend to have a higher false-positive rate, i.e., the amount of normal traffic that is flagged as an intrusion is higher.

An anomaly-based IDS typically consists of two phases, a training phase and a detection phase. In the training phase the system uses a training dataset to assess the normal situation and infers a model that describes normal behaviour. It is of course important that the training set does not contain traces of an attack, as the attack activity would also be regarded as normal behaviour by the trained system.

The type of learning can be divided into two categories, supervised and unsupervised. The difference between these two types is the information given in the training dataset. For supervised learning each entry in the training set it is known if the entry is normal or anomalous and it is labelled as such. For unsupervised learning this is not known, here the algorithm decides which entries are anomalous and which are not, one of the techniques that can be used for this is clustering. Usually an IDS that uses supervised learning gives better results. A labelled training set however, is much costlier to construct and is sometimes even impossible to gather.

After the training phase the IDS can be deployed in the system to defend. The system will monitor all traffic on the network and flag traffic that does not follow the established model as anomalies. Often the anomalous message is blocked as well.

Some anomaly detection algorithms incorporate a feedback loop after an anomaly is found. When an anomaly is found, this is presented to the user together with information about the anomaly. Then, the user can indicate if the anomaly was a false-positive, or that it indeed was an anomaly, i.e., a true-positive. The amount of information that is presented to the user can differ greatly between algorithms. Some algorithms can only say that a computed value is above a threshold. In this case no extra information about the cause can be given to the user and it can be very cumbersome to decide if the action was legitimate or not. Other algorithms can provide more information about a possible cause for the anomaly. The information provided typically depends on the characteristics of the algorithm.

4.3.1 Systems for CAN

A number of anomaly-based IDSs specific to CAN have been proposed. Four methods will be explained here:

1. The anomaly-based method of Miller and Valasek which is based on frequencies of CAN packets
2. A method that is based on entropy of the data field within CAN packets
3. One method uses a unique property, clock skew, of the internal clock of ECUs for fingerprinting to decide if a CAN packet was sent by its legitimate sender
4. Another method based on frequencies of CAN packets that uses a different detection mechanism

Miller & Valasek

Next to the signature-based method explained in Section 4.2.1, which looks for diagnostics messages, Miller and Valasek's device also utilises a method to investigate normal messages [47]. This method checks if messages are being sent at a much higher rate than normal. The method first learns the frequencies of packets per unique ID. While monitoring the CAN bus the actual frequencies are compared to these historical averages. If the measured value is much higher than the historical average a message is flagged as an anomaly. As said in the previous section, Miller and Valasek claim their device is capable of detecting all attacks they are able to execute themselves as well as attacks currently known in literature.

Entropy-Based

Marchetti et al. introduced an anomaly detection method based on entropy of the data field in CAN packets [41]. As explained in Section 2.2 a CAN packet consists of an ID, a data field and

a number of control fields. Entropy is a measurement from the field of information theory and represents the amount of information that is contained in a data set. Entropy can also be used to calculate the amount of randomness in a data set. The method of Marchetti et al. is one of the few IDSs for CAN that utilises information from the data field for anomaly detection, most methods only use timing information.

The meaning of the contents of the data field in a CAN packet is manufacturer-specific and can even differ between two consecutive versions of a single car model, therefore reverse engineering the content of the data field is not a viable option for a general automotive IDS. Using the entropy of this data field however, this method can still infer some information from the data field without requiring any knowledge about the actual content.

The entropy $\mathcal{H}$ of a dataset containing i different symbols is defined by

$$\mathcal{H} = \sum_i p(i) \log_2 \left[\frac{1}{p(i)} \right]$$

where $p(i)$ is the probability that the i_{th} symbol occurs. A dataset containing n independent symbols with a uniform distribution has an entropy of $\mathcal{H} = \log_2(n)$.

The algorithm can assess a CAN dataset in two different ways. Firstly, the set is divided into non-overlapping windows of 0.1, 0.5 and 1 seconds. For each window the entropy is calculated. Secondly, the set is assessed by looking at messages for each unique ID separately. Since each ID appears in the CAN traffic with a different frequency, time windows cannot be used here. The set is assessed in windows containing a fixed amount of messages.

In the training phase the entropy of each window, or set of messages with a unique ID, is calculated. The entire dataset has average entropy μ_e and standard deviation σ_e. For the entire set the following boundaries are calculated:

$$[\mu_e - k\sigma_e, \mu_e + k\sigma_e]$$

The value of k is decided by calculating $\mathcal{H}$ for all observations in a test set and raising k until all observations fall within this window, e.g., until the number of false-positives is equal to zero. Initially $k = 1$.

Once the values for k are decided new observations can be classified. For a new observation the entropy $\mathcal{H}$ is calculated and if this value is inside $[\mu_e - k\sigma_e, \mu_e + k\sigma_e]$ this observation is a normal observation and if it is outside the boundaries this observation is flagged as an anomaly.

Performance analysis was executed on a real CAN data set, captured from an unmodified vehicle during several hours of driving. Two different attacks were simulated, both based on injecting forged messages, i.e., fabrication attack. For one attack scenario a previously observed message was replayed. Since the same payload is sent often the measured entropy will decrease. The other attack scenario consisted of forging a message with a previously observed ID, but random data, this scenario imitates a fuzzing attack. Here the sent messages have different, random payloads, so the measured entropy will increase. Both scenarios were executed with different frequencies, a message was injected every 1, 0.5, 0.1, 0.05, 0.01 and 0.005 seconds.

When assessing all IDs together, the first attack scenario can be detected when messages are inserted every 0.01 and 0.005 seconds. The second attack scenario can be detected when messages are inserted every 0.005 seconds. For both scenarios all three window sizes of 0.1, 0.5 and 1 seconds show identical performance.

When each unique ID is assessed separately both scenarios at all attack rates can be detected for 40 to 42 out of 45 IDs. This means that for most IDs an injection of one packet per second can be detected, as opposed to 100 packets per second when all IDs are being assessed together.

Clock-Based

The Clock-based Intrusion Detection System (CIDS) was proposed by Cho and Shin [11]. CIDS is based on a property of any computer clock called clock skew. Clock skew is the characteristic that a clock tends to slowly drift away from a true clock over time. A normal computer resets its

internal clock periodically over the internet, but this does not happen inside a car. Every ECU has a distinct skew from a true clock and it turns out this can be used for fingerprinting.

We define clock skew as the difference between a clock's frequency and the frequency of a true clock, measured in microseconds per second ($\mu s\, s^{-1}$) or parts per million (ppm). Offset is the difference between the the time a clock reports and the true time, measured in milliseconds (ms). Figure 4.1 shows that the accumulated clock offset grows linearly over time, i.e. clock skew is constant, and that all ECUs A, B, C and D are clearly distinguishable from each other.

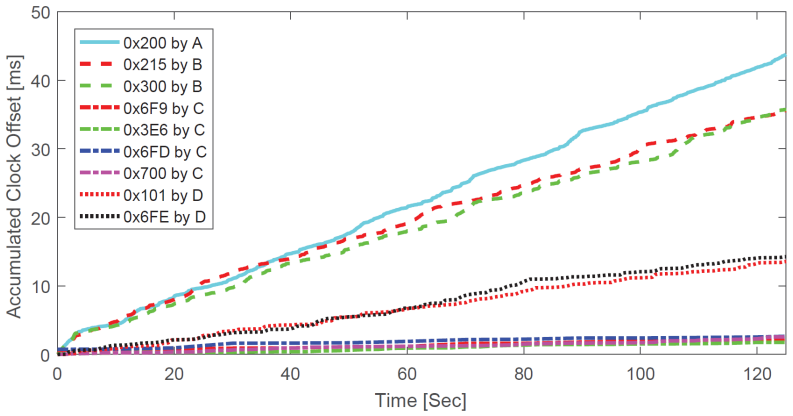

Figure 4.1: Accumulated clock offsets in a 2010 Dodge Ram Pickup [11]

The algorithm has two ways of detecting intrusions: per-message detection and message-pairwise detection. The main detection method is per-message detection and message-pairwise detection is used to reduce the amount of false-positives and -negatives. Both methods' training and detection phases are different, but they are based on the same concepts. Consider ECU A sending a message every T ms and ECU R receiving this message. Since ECU R has no other information about time, we will consider its clock as the true clock. ECU A will start sending messages from $t = 0$. O_i is its clock offset when sending the i-th message. The ideal times for sending messages for A would be T, $2T$, $3T$, ...

The timestamp when ECU R would receive A's i-th message is $iT + O_i + d_i + n_i$, where d_i is the network delay and n_i is noise in R's timestamp quantisation [79]. Thus, the interval between the message $i-1$ and message i is equal to $T_{rx,i} = T + \Delta O_i + \Delta d_i + \Delta n_i$ where ΔX_i denotes the difference between X_{i-1} and X_i and with $O_0 = 0$. Using this, the expected interval $\mu_{T_{rx}} = \mathrm{E}[T_{rx,i}]$ can be expressed as

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

The last step holds since the change in O_i between two messages is negligible, n_i is a zero-mean Gaussian noise term and since the lengths of the data field in periodic CAN packets are constant over time we can consider $\mathrm{E}[\Delta d_i] = 0$.

Based on the arrival time of the first message, i.e. $0T + O_0 + d_0 + n_0 = d_0 + n_0$, and the average intervals $\mu_{T_{rx}}$ the estimated arrival time of the i-th message is $i\mu_{T_{rx}} + d_0 + n_0$ and the actual measured arrival time is $iT + O_i + d_i + n_i$. It turns out that we can estimate the average clock offset $\mathrm{E}[O_i]$ using the average distance between the estimated and measured times becomes

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

Per-message detection uses this method to estimate the average clock offset. In the training phase a linear regression model is described per message ID based on the estimated clock offset.

This regression model describes the normal situation. In the detection phase for new messages CIDS estimates the clock skew and constructs a norm model. To detect sudden persistent, but small, differences, e.g. a change in clock skew caused by an impersonating ECU, CIDS uses the Cumulative Sum Method (CUSUM). Since it is cumulative it is able to pickup small persistent changes. If the norm model shows values deviating from the regression model the message is regarded anomalous.

Just as per-message detection, message-pairwise detection is also based on linear regression models. Therefore, the CUSUM method is applied here as well. Message-pairwise detection uses the correlation between the average clock offset of two periodic messages sent by the same ECU. Since these messages come from the same source, their offsets are likely equivalent and thus correlation should be high. Since message-pairwise detection is only applicable to messages coming from the same source, it is used as an additional feature in CIDS. Per-message detection can be used for all messages.

After either of these two methods identifies a message as being potentially anomalous, a verification step is executed to assess if a potential anomaly is a false-positive. This verification step checks whether the CAN bus was busy right before this message was sent. This check was implemented to ignore false-positives that could have been caused by CAN's arbitration. As explained in Section 2.2, when two ECUs attempt to transmit a message at the same time an arbitration scheme is used to decide the order in which the messages are sent: the message with the lowest ID gets the highest priority. When a message loses arbitration, this message will be resent when the bus becomes free, thus making the actual arrival time larger than the value that CIDS predicted based on clock skew. Without the verification step the message that lost arbitration would be flagged as being anomalous. In an attempt to lower the number of false-positives that were caused by arbitration, the verification step checks if the CAN bus was busy right before the potentially anomalous message was sent. If the bus was busy, the message is considered being normal traffic and is not flagged as an anomaly.

CIDS is the only IDS described here that has fingerprinting capabilities. Since it has a record of all skews belonging to each ID, it can determine the source of a fabricated message. When a message has a different time offset than predicted, the skew can be matched to another ID and thus to another ECU. To indicate which ECU sent the message, it has to be known which ECU sends which message. This can give an indication about the compromised systems inside a car.

CIDS is able to detect most attacks currently known in literature, i.e. fabrication, suspension and masquerade attacks, and has the possibility to detect more sophisticated attacks that will be developed in the future. While this method was designed to detect masquerades attack especially, which cause a small difference in arrival times based on clock skew, fabrication and suspension attacks can also be detected since these cause an even larger difference in arrival times. Fabrication and suspension attacks can be detected in all of their experiments without any false-positives. When CIDS uses per-message detection all masquerade attacks can be detected with a false-positive rate of 0.055 % in their experiments. When message-pairwise detection is used as well, all masquerade attacks can be detected without false-positives.

Frequency-Based

Taylor, Japkowicz and Leblanc proposed an anomaly detection method for CAN based on frequencies of packets [74]. This method compares collections of statistics about ongoing traffic to historical averages. They were able to detect packets inserted manually into a simple dataset. Their datasets were collected from a 2011 Ford Explorer while driving for five minutes, without operating user controls, e.g. lights and windows.

A collection of statistics about data traffic is called a flow, which is commonly used in network traffic analysis. For a conventional network a flow comprises all traffic of a complete communication between two endpoints. The concept of flow cannot be applied to CAN bus data directly, since there are significant differences with conventional networks. For CAN the source and destination of packets are not known and all traffic is broadcast so there is no end of transaction. A modification proposed by Valdes and Cheung for traffic analysis in industrial control systems is used [75].

Instead of a complete communication, a sliding window is used, inside which statistics about the traffic is collected. Among other information, a flow contains packet ID, the average time and variance of the time difference between successive packets. The window size was set to one second and is moved over the data in half second increments, thus generating statistics twice per second. Some messages however, are broadcasted only once per second or even less, others are not even periodical. The authors decided to only include packets that have an interval between broadcasts of 50ms or shorter and claim this is 90 % of all traffic. In one of the datasets we use this type of packets account for approximately 87 % of all packets.

The attacks that were simulated for testing are fabrication attacks, as explained in Section 3.4. Steps 2.2 and 2.3 of the fabrication attack are not used in their simulation, packets are only inserted at varying frequencies and for different periods of time. To account for packet collisions when inserting packets, existing packets' timestamps are modified. The contents of the data field are copied from an earlier message with the same ID. The other kind of fabrication attack, suspension attack and masquerade attack as explained in Section 3.4 were not simulated.

The best detection results were achieved by a one-class support vector machine. An attack where only three packets were inserted could be detected with an acceptable false-positive rate, when the window size was reduced to 0.2 seconds. From their experiments it also became clear that all information was taken from the mean interval time, hence all other information contained in a network flow did not increase performance. They initially also used information about the data field in the packet, but as this did not have any increase in performance we still consider the entropy-based method explained above as the only method that effectively uses information from the packets' data field.

A one-class support vector machine is an unsupervised learning algorithm, which is trained on a dataset only containing data describing normal behaviour. This is the reason it is called a one-class algorithm, only positive samples are given in the training set, no negative samples are supplied. The algorithm attempts to construct the best fitting model for the training set. New data is either classified as similar to the training set, i.e. a normal observation, or different from the training set, i.e. an anomaly.

4.3.2 Shortcomings

We will describe the shortcomings of each of the anomaly-based IDSs given above. Based on these findings a decision will be made about the type of anomaly detection method we will implement for our IDS.

Miller & Valasek

Miller and Valasek's device uses two methods for detection, the signature-based method explain in Section 4.2.1 and the anomaly-based method explain in Section 4.3.1. As already explained for the signature-based method, Miller and Valasek's assumption that an attack only consists of inserting either diagnostics messages or normal messages with an increased frequency, i.e. a fabrication attack, is incorrect. This assumption does not account for suspension and masquerade attacks, the two other types of attacks we identified in Section 3.4. Based on this fact and the very simple nature of their device we conclude that we also will not conduct any further research into the anomaly-based method of their approach.

Entropy-Based

The method of Marchetti et al. is able to classify a data set in two different ways. Either dividing it into timed windows and assessing all IDs together, or for each unique ID separately. Analysing CAN traffic in windows containing all IDs is very limited, as only insertion of messages with a very high frequency of hundreds of packets per second can be detected.

While assessing each unique ID independently gave much better results, it is required to run a number of instances of the algorithm in parallel, one for each ID. Another shortcoming is that

ten IDs have such a large deviation in entropies under normal circumstances that attacks are not distinguishable from normal traffic. For these IDs the k value in the boundaries is larger than 15. Although their method is the only one that utilises information other than time, the results are not very promising and thus we decided to not conduct any further research into these types of algorithms.

Clock-Based

We identified a critical weakness in CIDS that allows an attacker to execute any attack without being detected. The vulnerability is the verification step that happens after a message is flagged as being potentially anomalous. This check was designed to reduce the number of false-positives caused by lost arbitration. Our attack tricks this verification check by faking lost arbitration: send a message right before the attack message that causes the attack message to lose arbitration. This way the attack message will be delayed and thus will be flagged as a potential anomaly. In the verification step however, the message will not be considered an anomaly, because the bus was busy right before this message was sent.

Next to this significant vulnerability, there are a number of smaller flaws that are more difficult to execute for an attacker. One of the factors that influences clock skew is operating temperature. If the temperature of an ECU can be changed, the skew can be changed to match the skew of the ECU that the attacker compromised. Additionally, ECU clocks have been identified to have a skew between 30 and 460 µs [11]. A typical ECU has a clock speed of around 40 mHz [52]. This means that the clock skew will wrap around the clock. If the attacker times the sending of malicious packets well, this will not be detected by the algorithm. Another point is that CIDS is based on measurable values. If we can measure clock offset and skew, an attacker also could also do this. Once an attacker can measure skew of ECUs and impersonate an ECU, including its skew, CIDS will fail.

The unique property that CIDS uses for anomaly detection, clock skew, seemed very promising at first and its fingerprinting capabilities are an interesting addition. The flaws we detected however, have lead to our conclusion that we will not conduct any further research in the use of clock skew for anomaly detection.

Frequency-Based

The frequency-based method by Taylor et al. has a number of shortcomings. First of all, since their method is based on frequency of recurring packets, non-periodic packets cannot be checked. Moreover, all periodic packets that have an interval time of more than 50 ms are not investigated by them either. An attacker can still attack these messages, and thus the systems they control, unnoticed. Not investigating non-period might be less of a problem, since all critical systems like steering and brakes are controlled by periodic messages. Not investigating slower period messages however, is a significant shortcoming. Their claim that they investigate 90 % of all traffic seems to be realistic, as in one of our datasets this type of traffic accounts for approximately 87 %.

For evaluation of their algorithms they only simulate one type of attack on a very simple dataset. The circumstances under which the data was captured did not replicate normal driving behaviour. For the attack simulation they inserted packets at different frequencies for different periods of time, e.g., a fabrication attack. The lack of other types of attack that are currently known in literature make it hard to review their approach for real life applications.

4.4 Conclusion

A signature-based IDS does not have the capabilities to detect new, unknown attacks. Therefore, in this work we have chosen to implement an anomaly-based detection algorithm. We found shortcomings for all anomaly-based methods that we have described. For a number of methods these shortcomings are caused by the fundamentals of the proposed algorithms. For the method by Taylor et al. we found most shortcomings in the design decisions of the algorithm and the execution

of the experiments. Hence, we believe their method has the largest room for improvement and also comes with the largest potential. They concluded in their research that the algorithm giving the best results is a one-class support vector machine. This algorithm falls under a group of unsupervised learning algorithms called one-class classifiers. We will explore a number of different algorithms in this group. When implementing these algorithms we will improve the design of the method by Taylor et al. in areas such as the amount of packets that are being evaluated by the algorithm and better attack simulation for evaluation of the algorithm.

Chapter 5

One-Class Classification Algorithms

In this thesis we will compare three algorithms and decide which one is best suited to be used in an IDS. One of the algorithms is the one-class support vector machine, as it gave the best results for Taylor et al. [74]. The one-class support vector machine is part of a group of unsupervised learning classifiers, called one-class classification algorithms. The other two algorithms that have been investigated, namely the isolation forest classifier and robust covariance estimator, also belong to this group. These algorithms will be implemented and their performance will be compared.

5.1 Terminology and Concepts

Broadly speaking classification is a multi-class problem. A training set contains entries representing multiple classes and a classifier constructs a model describing all classes. In the classification phase, for each new observation the classifier decides to which class it belongs. In a one-class problem the training set only contains observations describing one class and the classifier constructs a model that captures this class. In the classification phase, for each new observation the classifier decides if it falls inside this model or that it is an anomaly.

A situation where one-class classification algorithms are useful is monitoring machine operations. Imagine monitoring an assembly line for errors in the production process. Data describing the normal situation can be gathered easily. Data describing errors however, cannot be collected easily and cost-effectively. Simulating this data would not give a complete view of all possible errors. For our research, the type of data we have available is very similar. We can collect CAN network traffic from cars under normal driving conditions, but traffic containing attacks is much more difficult to obtain. Hence, a binary classifier cannot be used. A one-class classifier is trained on the available data and thus constructs a model describing the behaviour of the production line under normal circumstances. In the monitoring phase, the classifier decides for each new observation whether or not it fits in the class describing normal behaviour or that it is an anomaly.

Other names used for one-class classification are outlier detection, since anomalies are usually more isolated from the normal data in the training set, or novelty detection, as the algorithm aims to detect new observations about which nothing is known.

5.2 One-Class Support Vector Machine

Schölkopf et al. introduced a one-class classification method based on support vectors, the one-class Support Vector Machine (SVM) [67]. In order to understand their method we first explain the concepts behind a conventional linear SVM applied to a binary classification problem [14]. Afterwards the differences with a one-class problem can be investigated.

A linear SVM aims to find a hyperplane $\overrightarrow{w}$ that separates the classes of observations. A hyperplane is a subspace of one dimension less than the space it resides in. The linear SVM then optimises $\overrightarrow{w}$, i.e., it maximises the distance between $\overrightarrow{w}$ and the closest observations of each class. In a binary classification problem a classifier is trained on a labelled dataset containing observations belonging to two classes. Hence, in a binary classification problem with two-dimensional data a hyperplane is a one-dimensional line.

Consider training data

$$x_1, ..., x_l \in \chi$$

with $l \in \mathbb{N}$ observations from some set $\chi \subseteq \mathbb{R}^N$. The goal is to train an SVM such that we have a binary classifier that returns 1 for observations from χ that are on one side of $\overrightarrow{w}$ and 0 for the others.

Recall that when χ is a two-dimensional set the hyperplane is a one-dimensional line. Intuitively, we will rarely encounter a linearly separable data set. To handle linearly inseparable data sets, all observations x_i are projected to a higher dimension, called the feature space F. In Figure 5.1 it can be seen that a linearly inseparable dataset can become separable in a higher dimension. For this projection step we need a function ϕ that maps elements from χ to elements in F. The SVM then aims to find a separating hyperplane in F.

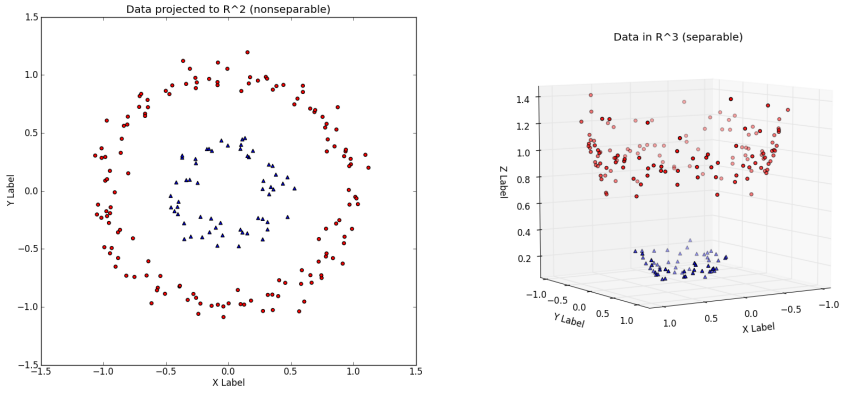

Figure 5.1: Projecting linearly inseparable observations to a higher dimension [37]

Suppose we have a function ϕ that can transform dataset χ to $\chi' = \phi(\chi)$. An example of what ϕ could look like is $\phi([x_1, x_2]) = [x_1, x_2, x_1^2 + x_2^2]$. Then, we can train a linear SVM on χ'. To classify an observation $y \in \mathbb{R}^N$ it should first be transformed into $y' = \phi(y)$ before it can be classified by the trained classifier.

When the dimension of F increases it becomes increasingly more computationally expensive to compute the result of transformation function ϕ. However, it can be shown that during the training phase of the SVM only pair-wise dot products are calculated, i.e., $(x_i \cdot x_j)$ with $x_i, x_j \in \mathbb{R}^N$. This is very useful because it turns out that there exists a function that is able to implicitly compute the dot product between two vectors $v, w \in \mathbb{R}^N$ in a higher dimensional $\mathbb{R}^M$ without making the explicit transformation of v and w to $v', w' \in \mathbb{R}^M$. This function is called a *kernel function*. Hence, all dot product computations can be replaced by applying kernel function k, i.e.,

$$k(x_i, x_j) = (\phi(x_i) \cdot \phi(x_j))$$

Implicitly, computing dot products in higher dimensions using a kernel function is called the *kernel trick* [36]. A number of kernel functions are commonly used with a SVM, e.g. linear, polynomial, radial basis function and sigmoid [26]. Using these concepts a linear SVM can construct and optimise a hyperplane in a theoretically infinite dimension and use this to solve binary classification problems.

For our one-class problem, Schölkopf's method [67] aims to create a function which returns +1 in a region representing the observations in the training set and -1 elsewhere. To achieve this the same concepts apply as for the described binary problem but now the distance between the origin and the hyperplane is maximised (in the feature space F). The following quadratic function describes the minimisation of this distance:

$$\min_{w,\xi,\rho} \frac{1}{2}\|w\|^2 + \frac{1}{\nu l}\sum_i \xi_i - \rho$$

subject to

$$(w \cdot \phi(x_i)) \geq \rho - \xi_i$$

Non-zero slack variables $\xi_i \geq 0$ are introduced to allow some training observations to be outside the constructed model. w is the normal vector that is positioned perpendicular to the hyperplane and $\frac{\rho}{\|w\|}$ gives the distance between the origin and the hyperplane along w. Moreover, $\nu \in (0, 1]$ controls the trade-off between ξ_i and $\|w\|$. ν is given as an input parameter to the SVM and the importance of this parameter is the reason that this algorithm is sometimes called ν-SVM. Expecting that w and ρ solve this problem, for most $x_i \in \chi$ the following function will be positive:

$$f(x) = \text{sgn}((w \cdot \phi(x)) - \rho)$$

We define $\text{sgn}(x)$ to be a function that returns 1 when $x \geq 0$ and -1 otherwise. Just as for the binary classification problem, the kernel trick can be used to avoid explicit transformation of observations to a higher dimension. Then, Schölkopf uses so called Lagrange multipliers, which are helper variables that are used to simplify optimisation problems, to solve the problem:

$$f(x) = \text{sgn}(\sum_i \alpha_i k(x_i, x) - \rho)$$

Thus, this method creates a hyperplane that separates all training observations from the origin and has a maximal distance from the origin in feature space F. Then, decision function $f(x)$ returns +1 for observations located on one side of the hyperplane and -1 for observations on the other side.

5.3 Isolation Forest

Liu et al. introduced a one-class classification algorithm that uses isolation forests [40]. Their method is based on the assumption that it is easier to isolate anomalies from the rest of the observations than to construct a model describing normal behaviour. To isolate an observation they recursively randomly partition the data set until the observation is the only data point in a partition. Recursive partitioning can be represented by a tree structure. A forest of random trees can collectively give a measure of normality for an observation.

An isolation tree is a proper binary tree, i.e., each node has exactly zero or two children. Let T be a node. Then T is either a leaf and has no children, or T is an internal node and has two children T_l, T_r. To decide which nodes are under which children, T is associated with a test. A test consists of an attribute q and a test value p such that the nodes with $p < q$ are under T_l and the rest is under T_r.

In a random tree the data is partitioned recursively until all observations are isolated. The amount of partitions required to isolate an observation is equal to the path length from the root to a leaf of the tree. The path length is noticeably shorter for anomalies when the dataset is partitioned randomly. One reason for this is that the number of anomalies is usually smaller than the number of normal observations, this results in a smaller number of partitions, i.e. short paths in a tree, necessary to isolate the observations. The other reason is that anomalies usually have attribute values that are easily distinguishable from normal observations and thus are more likely

to be separated in early partitioning. Figure 5.2 shows that anomalies are more susceptible to isolation. In this figure normal observation x_i requires twelve random partitions to be isolated, anomaly x_0 on the other hand only requires four partitions to be isolated.

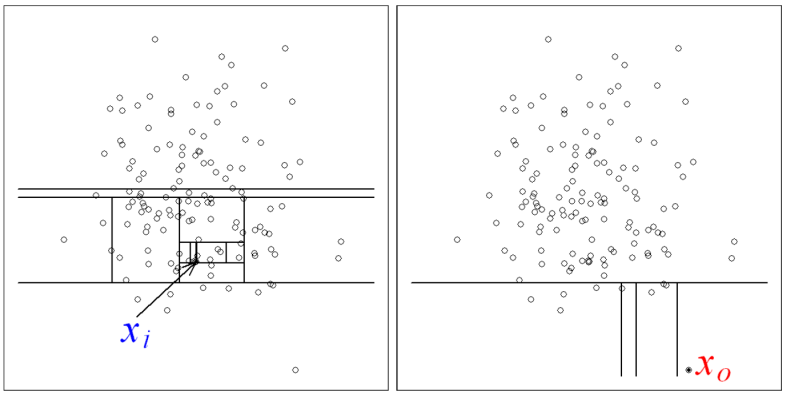

Figure 5.2: Isolating a normal observation (x_i) vs. isolating an anomaly (x_0) [40]

To explain how the isolation forest classifier works we will first describe a number of important building blocks. When these are clear, the process in terms of a training and classification phases can be given. Given a set $X = \{x_1, ..., x_n\}$ containing n observations an isolation tree can be built by recursively diving X. For division an attribute q and test value p are randomly selected and X is split based on this test. Division is executed recursively, until either: the tree reaches a height limit, $|X| = 1$, or all data in X have the same value. The height limit will be approximately equal to the average tree height and the idea behind this is that we are only interested in observations with a shorter than average path length. Assuming all values in X are distinct and the tree is fully grown, each observations is isolated to a leaf. Then the number of leaves is n, the number of internal nodes is $n - 1$ resulting in a total amount of $2n - 1$ nodes. Path length $h(x)$ of an observation x is calculated by measuring the number of edges the observation traverses from the root node to a leaf.

Each test when constructing the isolation trees is selected randomly, so each tree has different sets of partitions. To calculate the expected path length, the average of path lengths is taken over a number of trees. It is shown that the average path length converges well before using 100 trees.

For a random forest to be used as an anomaly detection algorithm it needs to be able to generate comparable anomaly scores. The value of $h(x)$ cannot be used as an anomaly score directly, since while the maximum height of a tree grows in the order of n, the average height grows in the order of $\log n$. In terms of structure, random trees can be compared to Binary Search Trees (BSTs). The estimation of $h(x)$ for a leaf is comparable to an unsuccessful search in a BST. Therefore, the average path length of an unsuccessful search in a BST is comparable to the average of $h(x)$ and thus it can be used to normalise $h(x)$. For a dataset containing n observations the average path length of an unsuccessful search in a BST can be calculated, as given in [61], by

$$c(n) = 2H(n-1) - (2(n-1)/n)$$

Here $H(i) \approx \ln(i) + 0.5772156649$ (Euler's constant) is the harmonic number. Hence, the anomaly score s of an observation x is defined as

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}$$

where $E(h(x))$ is the average of all $h(x)$ values from a collection of isolation trees.

Using the building blocks explained above the process of training an isolation forest classifier and classifying observations can be described. In the training phase isolation trees will be constructed randomly as explained above. The algorithm has two parameters, namely sub-sampling

size ψ and number of trees t. Based on the sub-sampling size the tree height limit l can be calculated as $l = ceiling(\log_2 \psi)$. Sub-sampling increases performance of the classifier as it controls the data size, such that anomalies can be better isolated. Since each isolation tree is based on a different sub-sample it can be specialised.

When the classifier is constructed anomaly scores can be calculated as explained before, using $s(x, \psi) = 2^{-\frac{E(h(x))}{c(n)}}$. The following assessment can be made about an observation based on anomaly score s. If for an observation s is close to 1, then that observations is definitely an anomaly. If for an observation s is much smaller than 0.5, then that observation is most probably normal behaviour. If for all observations s is about 0.5, then the entire set does not contain any distinct anomalies.

5.4 Robust Covariance Estimator

Rousseeuw and Van Driessen proposed an algorithm based on the Minimum Covariance Determinant (MCD), which is a robust estimator for covariance [65]. Here robust means that the estimator is not influenced by the presence of outliers in the training set. This algorithm was developed to measure a production process at Philips and detect deformations and abnormalities. The algorithm assumes the training set contains regular data that follows a known distribution, e.g., the Gaussian distribution. It aims to fit a robust covariance estimate to the data and then it uses the Mahalanobis distance to decide if a new observation is a normal observation or an outlier.

Given a distribution that has been fit on the training data set, the Mahalanobis distance is the distance between an observation to the mode of a distribution. The mode of a distribution is the value that appears most often. The Mahalanobis distance of an observation x can be calculated by

$$D(x) = \sqrt{(x - \mu)' \Sigma^{-1} (x - \mu)}$$

where μ is the mean and Σ covariance matrix of the distribution. In the remainder of this section the determinant of matrix Σ will be denoted by $\det(\Sigma)$. Thus, the greater the Mahalanobis distance for an observation, the more likely that the observation is an outlier.

Algorithms that use the usual covariance maximum likelihood estimate are very sensitive to outliers in the training set. When a number of outliers are close to each other they can appear to be normal, this effect is called masking. A number of outliers grouped together will change the value of the mode and thus will result in a lower Mahalanobis distance for these outliers. A robust estimator is not influenced by this effect. It will result in a high Mahalanobis distance for all outliers, even if they are grouped together.

The method Rousseeuw and Van Driessen proposed in [65] is actually an improvement of an older MCD method also developed by Rousseeuw [63, 64], which is called Minimum Volume Ellipsoid (MVE). Even though it was a promising algorithm with good performance, this method was too computationally expensive for practical use, as it could only handle a few hundred observations in a few dimensions. Rather than finding the exact solution, which the older version attempted, the optimised MCD version aims to find an approximation of the solution.

Consider training data $x_1, ..., x_n \in \chi$ where each observation x_i has p variables. The first step of the algorithm to take a subset of the data $H_1 \subseteq \chi$. This can be done in two ways:

1. Select a random h-subset $H_1 \subseteq \chi$. By default the value h is $\frac{n+p+1}{2}$.

2. Select a random $(p + 1)$-subset $H_0 \subseteq \chi$ and compute

$$\mu_0 = \frac{1}{h} \sum_{i \in H_0} x_i \text{ and } \Sigma_0 = \frac{1}{h} \sum_{i \in H_0} (x_i - \mu_0)(x_i - \mu_0)'$$

(If $\det(\Sigma_0) = 0$, then continue by adding random $x \in \chi$ to H_0 until $\det(\Sigma_0) > 0$.)

For χ compute distances

$$D_0(i) = \sqrt{(x_i - \mu_0)' \Sigma_0^{-1} (x_i - \mu_0)} \text{ for } i = 1, ..., n$$

and sort these distances. This yields the permutation π for which $D_0(\pi_1) \leq ... \leq D_0(\pi_n)$ and put $H_1 = \{\pi_1, ..., \pi_h\}$.

Now that we have subset H_1 we do the following step iteratively: Given h-subset H_k compute the distances $D_k(i)$ for $i = 1, ..., n$ and sort these distances. This yields the permutation π for which $D_k(\pi_1) \leq ... \leq D_k(\pi_n)$. Put $H_{k+1} = \{\pi_1, ..., \pi_h\}$. Compute average μ_{k+1} and covariance matrix Σ_{k+1}. This will result in

$$\det(\Sigma_{k+1}) \leq \det(\Sigma_k)$$

where these determinants are equal to each other if and only if $\mu_{k+1} = \mu_k$ and $\Sigma_{k+1} = \Sigma_k$. This step is repeated until $\det(\Sigma_{k+1}) = \det(\Sigma_k)$ or $\det(\Sigma_{k+1}) = 0$. The result of these iteration steps is a distribution describing the normal observations and can be visualised by an ellipse. This is our model of the normal behaviour. Figure 5.3 shows the resulting model of these steps and also shows the difference with a non-robust estimator. The classical model has a completely different shape caused by the small group of outliers on the left.

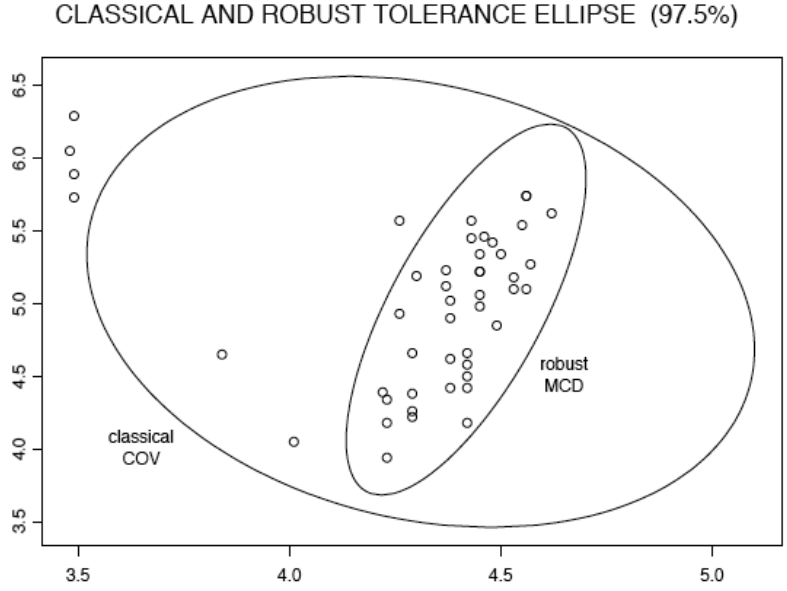

Figure 5.3: Model constructed by the classical and robust covariance estimators [69]

For new observations it is calculated if they are located inside our outside of the ellipses by using the Mahalanobis distance. As explained, the Mahalanobis distance is calculated using the mean and covariance matrix of the distribution, which is the model we created.

Chapter 6

Setting

In this chapter we describe the setting of the experiment, the methodology that will be followed to measure the performance of three anomaly detection algorithms selected in the previous chapter and how these results can be compared to each other. After the analysis of the results we can answer the second sub question that was described in Chapter 1: *Which anomaly detection algorithm gives the best results at CAN bus level?*

The experiments consist of a number of steps. First of all, we collect CAN traffic data. We attempt to record CAN traffic data from a real car. For this we try a number of devices that connect to the OBD-II port in a car, as explained in Section 2.3.1. We also collect CAN traffic logs from other sources. The collected data can be investigated to find features that our algorithms can use for anomaly detection and will be divided into a training set and a test set. The algorithms identified in Chapter 5 will be implemented and trained on the CAN training set. To test the algorithms the attacks described in Section 3.4 will be simulated on the remaining CAN test set. The performance of the three algorithms are compared based on a number of metrics. We chose Python [24] as programming language to implement the attack simulation and data preprocessing and its Scikit Learn packet [58] for the implementation of the anomaly detection algorithms.

6.1 Data Collection

Our experiments will be conducted on CAN traffic data. We attempt to collect data from a number of different sources. One of these sources is an unmodified, road legal car while it its being operated under normal driving conditions. To connect our devices to the car we use to the OBD-II port. As explained in Chapter 2 this port is connected to the CAN bus. We experiment with different devices that are able to log CAN data through the OBD-II port.

Next to logging CAN data ourselves, we acquire CAN traffic from two other sources. One CAN data set is available on the website of the Crash Reconstruction Research Consortium of the University of Tulsa [55]. Additionally, we use a CAN simulator, called Instrument Cluster Simulator (ICSim), to generate data [56].

In Section 2.2 the structure of a CAN packet was explained. The values that are important for identifying automotive attacks are the ID, data field and the time stamp. One strict requirement is that we need the time stamp in microseconds (µs), as the algorithms will only investigate timing information. Collecting this data proved to be a lot harder than expected.

6.1.1 CANBus Triple

The first device we used was the CANBus Triple [38]. This device is developed by Derek Kuschel and is a ready made device to read and write CAN messages via the OBD-II port. It is based on an ATmega32u4 microprocessor [44] and uses the MCP2515 CAN controller [45] to read from and write data to the CAN bus.

Unfortunately, this device did not give us the required results. In our experiments the device gave the same value for the ID of all packets and no timestamps were given.

6.1.2 Arduino Based Device

Our next attempt was to construct a device ourselves. The device we built is based on an Arduino Uno [3] and is extended with the CAN-BUS shield made by SparkFun [70]. The Arduino Uno is a micro controller based on an ATmega328P [43]. An Arduino shield is a board that can be placed on top of an Arduino to extend its functionality. It can add an interface, e.g. a connector for an OBD-II cable, or add extra storage, e.g. by adding a Micro SD card reader. The CAN bus shield uses the Microchip MCP2515 CAN controller [45] with the MCP2551 CAN transceiver [46], just as the CANBus Triple. The shield allowed us to use an OBD-II cable to connect the device to a car. To read the CAN data and log it, we wrote a sketch based on a CAN library provided by SparkFun [71].

With this device we managed to log CAN data including the correct ID values and timestamps in microseconds. However, later analysis showed that the data is not useful for our purpose as the Arduino was too slow. The log seems to contain all CAN packets, including timestamp, but after a few messages the interval time was directly dependent on previous data field length, indicating that the packets arrive faster than the Arduino can process them and puts new packets in a queue. As the messages are being queued and the Arduino calculates the timestamps, it does not provide any useful information.

We also tried an Arduino Due [2], which is a faster Arduino board. The problem here was that the Uno and the Due are not compatible, we could not use our shield and sketch. The problem was that the Arduino Due is based on a 32-bit ARM core microcontroller, while the Arduino Uno is based on an AVR microcontroller.

6.1.3 CAN Badger

The third device was Code White's CAN Badger [12], which we had to assemble ourselves completely. A Printed Circuit Board (PCB) design was provided that we could send to a PCB manufacturer and all components had to be ordered and soldered on the PCB separately. The CAN Badger also uses the MCP2551 CAN transceiver [46] and we used the LPC1l769 LPCXpresso micro controller [54] to control the board. The board has two connectors for an OBD-II cable that we can use to connect it to a car with the OBD-II cable we also used for the Arduino. Code White provided firmware for the micro controller.

The CAN Badger provides a number of operating modes. A USB mode in which it can be controlled via a serial monitor. This mode provides the most options, e.g. protocol analysis, logging and even a man-in-the-middle attack option. In standalone mode the CAN Badger is not connected to a computer (only to the car) and is operated by a button. Via the button and an indicator light the preferred mode can be selected by pressing the button a pre-specified number of presses.

Unfortunately, this device did not deliver good results either. The device connected to the car and presented the available functionalities, but was not able to log CAN traffic. We suspect this problem was caused by the particular SD card module we connected to the board, but we were not able to fix this issue.

6.1.4 University of Tulsa

The Crash Reconstruction Research Consortium of the University of Tulsa has a CAN data set available on its website. This dataset was recorded from a Dodge RAM Pickup. They used a Dearborn Group Gryphon S3 [18] to record the data and exported it to a .csv file using DG Technologies Hercules software [19].

The data was recorded under normal driving conditions. The car drove away from a normal residential driveway and pulled up on the street. After three right hand turns the car was backed up into a parking space. The length of the dataset is 185 s.

However, from a certain moment, half of these ID values disappears and the interval times increase. One possibility is that the engine of the vehicle was turned off that this time and that it was just left on contact. There was a very short period of time with similar behaviour at the beginning of the file as well. We removed the beginning and the end of the dataset that showed this behaviour. The leftover dataset was approximately two and a half minutes long.

6.1.5 CAN Simulator

ICSim is a CAN simulator for SocketCAN [56], which which is a collection of libraries that implement CAN protocols for Linux. ICSim simulates a car's instrument cluster, e.g. speedometer, its controls, e.g. throttle and steering, and the corresponding CAN traffic. By default only the bits corresponding to these basic functionalities in the CAN packets consist of data, the others are zero. In a real car there is a lot more information to communicate and to simulate this in ICSim a "difficulty" level can be specified when setting up the simulator. Difficulty level 2 corresponds to the most realistic CAN traffic.

We made a capture of the data when the difficulty level is set to 2 for a duration of approximately five minutes. We used all user controls, we opened and closed the doors, pushed the throttle and used the indicators lights. Upon inspection this data seems to be realistic as it contains many recurring packets with various frequencies.

6.1.6 Summary

Data collection proved to be a lot harder than we anticipated in the beginning. One of the reasons that made data collection so hard for us is our requirement to collect CAN traffic with timestamps in microseconds. Unfortunately, after many experiments, we were not able to record data with this requirement from a car ourselves. We did find two datasets that are usable for our experiments, the CAN dataset of the University of Tulsa and the dataset simulated using ICSim. We will refer to these datasets as the real CAN dataset and synthetic CAN dataset, respectively. These datasets are approximately two and a half and five minutes long, respectively.

6.2 Attack Simulation

To assess the attack detection performance of the algorithms we need CAN datasets containing attacks. We will simulate attacks on the CAN traffic we collected. Since our datasets are not very big we will split them in such a way that we use 80 % as training set and 20 % as test set, on which the attacks will be simulated. A smaller training set could lead to over fitting and a smaller test set could lead to a very small amount of packets for IDs with a large interval time.

We simulate the three types of attacks that have been described in Section 3.4. Eight different attacks are simulated in total: two fabrication attacks, two suspension attacks and four masquerade attacks. For each attack an ID and a start time will be selected, the values are given in Table 6.1. The IDs are randomly selected and the start times are chosen to be approximately halfway through the dataset. The concept of fast, medium and slow packets will be explained in Section 6.3.1.

6.2.1 Fabrication Attack

In a fabrication attack an attacker inserts additional messages with the ID of an existing message. As described in Section 3.4, three versions of this attack have been identified. Since inserting a malicious message right before or right after a legitimate message are two very similar attacks, we will only simulate one of these.

On our dataset we insert a fabricated message in the CAN log, right before legitimate messages. We inserted the message 200 µs before legitimate messages. In this attack we insert one packet

CAN Dataset	ID type	Attack ID	Start time
Real	Fast	0x200	150 000 000 µs
Real	Medium	0x520	150 000 000 µs
Real	Slow	0x5B0	150 000 000 µs
Synthetic	Fast	0x21E	275 000 000 µs
Synthetic	Medium	0x37C	275 000 000 µs
Synthetic	Slow	0x5A1	275 000 000 µs

Table 6.1: Attack simulation IDs and start times

in total. The other option is to insert messages with an increased frequency, independent of the timestamps of the original messages. We chose to increase the frequency ten times. In this attack we insert 25 packets in total. The content of the data field does not matter, since it is not used in our analysis.

6.2.2 Suspension Attack

In a suspension attack an attacker prohibits an ECU from sending messages. To simulate this attack we will simply delete messages from the CAN log. We execute the attack twice, on one packet and on 25 packets.

6.2.3 Masquerade Attack

In a masquerade attack an attacker substitutes a legitimate message with a malicious message. When observing network traffic this attack is very hard to detect. The only change is a tiny difference in the timestamp caused by the difference in clock skew. From the research done in [11] it follows that a clock's skew is typically a value between 30 and 460 $\mu s\, s^{-1}$, i.e., every second the clock's offset from a true clock increases by this amount of microseconds. In our simulation we take these two values as clock skew, to investigate the detectability of both the minimal and maximal clock skew. From this we can calculate the offset at each second, which is a cumulative value based on the clock skew. We will add the offset to the timestamps of existing packets. We simulate four different versions of this attack, we use the small and large skew twice, on one packet and on 25 packets. The content of the data field does not matter, since it is not used in our analysis.

6.3 Algorithm Execution

Using the concepts given above we can compare the three algorithms. Firstly, the training and test sets are preprocessed, resulting in feature vectors for fast, medium and slow messages. Per algorithm, three classifiers will be trained on the feature vectors, one for each type of message. Then the classifiers will predict to which class each observation of the test datasets belong. The test sets contain the three types of attacks.

6.3.1 Data Preprocessing

The collection of CAN logs containing normal and attack traffic has to be preprocessed before it can be analysed by our algorithms. These logs contain a timestamp, ID and data field for each packet. We describe how to construct the input for our anomaly detection algorithms from these CAN logs.

As seen in Chapter 4 all but one analysed automotive IDS based their detection method on timing information. The entropy-based approach is the only algorithm actually utilising information from the data field. From our analysis and comparison of these algorithms we concluded that the only information we can use are the timestamps in combination with the ID.

However, a separate tuple containing a timestamp and an ID does not contain much information. Therefore, we need a way to extend the information given in the CAN packets. We do this by using so called flows. This concept was already briefly mentioned in Section 4.3.2. The idea is that by combining metrics more information is given than is readily available when observing packets separately. Just as Taylor et al. [74] we will use the method of Valdes and Cheung [75], however, with a few modifications. As opposed to the method of Taylor et al. we will use all recurring packets by diving all IDs into three categories.

Taylor et al. only investigated packets with a maximal interval time of 50 ms. In their experiments they found that when the window size is decreased to 0.2 s, a fabrication attack of just three packets could be detected. This means that the slowest packet appears four times in one window. There are also packets that have an interval time shorter than 50 ms and occur more often in one window. In our analysis we want to investigates all packets. To do this we divide all packets in three groups, fast, medium and slow packets. Fast packets occur with a maximal interval time of 50 ms, medium packets occur with an interval time between 50 ms and 250 ms and slow packets occur with an interval greater than 250 ms. Based on the findings of Taylor et al. the window size each category is four times the largest interval time, hence the window size of fast packets is 0.2 s, for medium packets this is 1 s and for slow packets this is 8 s. The amounts of distinct IDs that belong to the three types of packets, i.e. fast, medium and slow, are given in Table 6.2.

CAN Dataset	All IDs	Fast IDs	Medium IDs	Slow IDs
Real	51	20	14	17
Synthetic	37	24	6	7

Table 6.2: Types of CAN packets in the datasets

Taylor et al. initially calculated many metrics for each window, including information from the data field. However, in their experiments removing certain metrics did not decrease performance. They came to the conclusion that all information used by the algorithms was taken from the mean interval time. That is why we will take the mean interval time as the only metric for our experiments.

The input that our algorithms will use is a feature vector that is calculated for each window. The feature vectors contain one metric per ID, namely the mean interval time μ. For example, if one window contains messages of four IDs, 0x200, 0x4D1, 0xA0A and 0xD1E, the feature vector FV for that window will look like follows:

$$FV = \begin{bmatrix} \mu_{0x200} \\ \mu_{0x4D1} \\ \mu_{0xA0A} \\ \mu_{0xD1E} \end{bmatrix}$$

6.3.2 Parameter Optimisation

All algorithms have a number of input parameters. By tuning these parameters the anomaly detection performance of the algorithms can be influenced. If the user does not specify any value for a parameter, the default value is used. Per algorithm all parameters are described and their default value is given. All parameters and the corresponding default values are described as they

are documented for Python's Scikit Learn packet [58]. We identify the most important parameters per algorithm and for these parameters we explain how we will choose the optimal value for our experiments.

One-Class Support Vector Machine

Table 6.3 describes the input parameters of the one-class support vector machine and their default values. The most important parameters for this algorithm are kernel and nu. The kernel parameter is so important, because of the importance of the kernel trick in the algorithm. The importance of the nu parameter, that controls the fraction of anomalies in the training set, is shown by the fact that the algorithm is sometimes called nu-svm.

The kernels we will use are the radial base function, linear, sigmoid and polynomial. For nu we take values between 0.5 and 0.0001. Since we assume our training set does not contain any anomalies, we want to investigate the effect of making nu smaller, starting at the default value and decreasing it. The largest value for nu that results in no false-positives on the training set is used for classification.

Name	Description	Default
kernel	Kernel function that the algorithm uses	Radial base function kernel
degree	Degree of the polynomial kernel. Ignored when other kernels are used	3
gamma	Coefficient for radial base function, polynomial and sigmoid kernel. Ignored when other kernels are used	1 / [number of features each point in the dataset contains]
coef0	Coefficient for polynomial and sigmoid kernel. Ignored when other kernels are used	0.0
tol	Tolerance of stopping criterion	0.001
nu	Fraction of anomalies in training data	0.5
shrinking	Specify if the shrinking heuristic is used	True
cache_size	Size of the kernel cache in MB	200
verbose	Specify if verbose output should be given	False
max_iter	Maximum number of iterations	No limit
random_state	Seed for the random number generator	Random seed

Table 6.3: Input parameters of the one-class support vector machine

Isolation Forest

Table 6.4 describes the input parameters of the isolation forest and their default values. The most important parameter for this algorithm is contamination, it has the same function as nu for the one-class support vector machine. Here also, we will take values between 0.5 and 0.0001. Selecting the best contamination value is the same as for the nu parameter for the one-class support vector machine. We start at 0.5 and decrease it. The largest value that results in no false-positives on the training set is used for classification.

Name	Description	Default
n_estimators	Number of base estimators in ensemble	100
max_samples	Number of samples that is taken from the training data to train each base estimator	min(256, [total number of samples in the dataset])
contamination	Fraction of anomalies in training data	0.1
max_features	Number of features that is taken from the training data to train each base estimator	1.0
bootstrap	Specify if sampling with replacement is performed on the random training subsets	False
n_jobs	Number of jobs to run in parallel	1
random_state	Seed for the random number generator	Random seed
verbose	Specify if verbose output should be given	0

Table 6.4: Input parameters of the isolation forest

Robust Covariance Estimator

Table 6.5 describes the input parameters of the robust covariance estimator and their default values. The most important parameter for this algorithm is contamination, just as for the isolation forest. Here also, we will take values between 0.5 and 0.0001. Selecting the best contamination value is the same as for the isolation forest. We start at 0.5 and decrease it. The largest value that results in no false-positives on the training set is used for classification.

Name	Description	Default
store_precision	Specify if the estimated precision is stored	True
assume_centered	Specify if the robust location and covariance estimates are computed and then, without centering the data, the covariance estimate is recomputed. If False, this recomputation step is not done	False
support_fraction	Amount of observations to use in the estimation	([total number of samples in the dataset] + [number of features each point in the dataset contains] + 1) / 2
contamination	Fraction of outliers in the dataset	0.1
random_state	Seed for the random number generator	Random seed

Table 6.5: Input parameters of the robust covariance estimator

6.4 Algorithm Comparison

To compare the performance of the different algorithms we need a way to quantify the correctly classified windows in the test sets containing attacks. First we investigate the windows an algorithm predicted as being anomalous. If a window actually contains an anomaly this is called

a True Positive (TP), however if this window contains normal behaviour this is a False Positive (FP). Similarly for the windows an algorithm classified as normal behaviour. If a window actually contains normal behaviour this is called a True Negative (TN), and if this window contains an anomaly this is a False Negative (FN). The total number of Positive (P) windows, i.e. that contain an anomaly, is equal to TP + FN. Similarly, the total number of Negative (N) windows, i.e. that contain normal behaviour, is equal to FP + TN. With these values there are a number of comparable metrics that can be computed.

First, the True-Positive Rate (TPR), which is the fraction of the total number of anomalies that the algorithm found that actually is anomalous and can be calculated by

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

The False-Positive Rates (FPR) is the fraction of the total number of anomalies that the algorithm found that actually is normal behaviour and can be calculated by

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$

Both rates have a value between zero and one. By combining these two metrics the Receiver Operating Characteristic (ROC) curve [73] can be created, which has the FPR on the x-axis and the TPR on the y-axis and gives the trade-off between these two values.

To draw this curve it is important to understand how an anomaly detection algorithm determines which label, e.g. normal or anomalous, to give to an observation. This is done by calculating a so called decision score. The decision score is a measure of how normal, or how anomalous, an observation is. The algorithm sets a threshold and all observations with a decision score on one side of the threshold are labelled as normal behaviour and all observations with a decision score on the other side are labelled as an anomaly. To create the ROC curve many values for this threshold are taken, between the minimal and maximal value of the decision score. For each threshold value the TPR and FPR are calculated and a curve is drawn through all these points, resulting in the ROC curve. A visual explanation of the ROC curve is given in Figure 6.1.

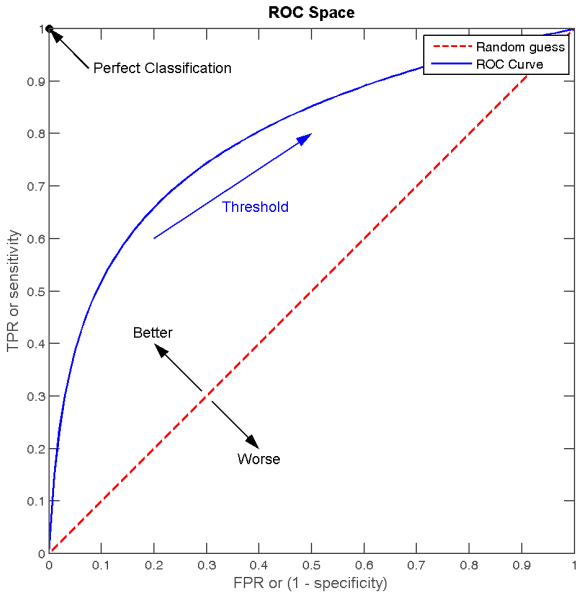

Figure 6.1: Receiver Operating Characteristic (ROC) curve [76]

This figure shows that to indicate high performance, the ROC curve should approach the top-left corner as close as possible with a steep slope, as this would indicate a high number of detected anomalies, i.e. a high TPR, with a low number of false positives, i.e. a low FPR.

To compare two ROC curves to each other we calculate the Area Under the Curve (AUC) [31]. This is a value between zero and one. As better performance is displayed by a ROC approaching the top-left corner, the closer the AUC is to 1 the better the classifier performs.

Given are the set of all m anomalies and the set of all n normal observations, iterators i and j, respectively, and the corresponding decision scores p_i and p_j. The AUC can then be calculated by

$$AUC = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \mathbf{1}_{p_i > p_j}$$

Here $\mathbf{1}_{p_i > p_j}$ is an indicator function [50] that returns 1 if and only if $p_i > p_j$. What this AUC function does is form all pairs of a normal observation and an anomaly. It then calculates the number of pairs where the normal observation has a higher score than the anomaly and divides this by the total amount of pairs. If all normal observations are correctly classified, i.e. have a high decision score, and also all anomalies are correctly classified, i.e. have a low decision score, this will result in an AUC of one, hence perfect detection. Similarly, this function will give zero for a detection where all observations are wrongly classified.

Chapter 7

Results and Analysis

In this chapter we present the results of the experiments described in the previous chapter. As explained, the comparable results are Area Under the Curve (AUC) values which are the area under the Receiver Operating Characteristic (ROC) curve values. Next to these results, we also give the corresponding values for the input parameters we identified as being important in Section 6.3.2. Since we have trained three different classifiers per algorithm, corresponding to fast, medium and slow messages, three different AUC values will be given per algorithm for each attack.

An analysis will be made to interpret the results and find differences in performance between the three algorithms. After this we can answer the second sub question that was described in Chapter 1: *Which anomaly detection algorithm gives the best results at CAN bus level?* In the analysis we also identify a number of limitations that we will describe.

7.1 Results

This section presents the results of the experiments described in Section 6.2. The results are divided per attack, with all variations of one attack in one table. This gives a clear overview of the differences in performance of each algorithm for each kind of attack. The tables give the AUC values, as explained in Section 6.4. An error occurred in the experiments with synthetic data, this is shown as "error" in the tables below. This will be explained in the result analysis.

7.1.1 Fabrication Attack

As explained in Section 6.2 two different kinds of fabrication attacks have been simulated. Either one message was inserted right before the occurrence of a real message, this will be denoted as "fabrication 1 before", or 25 messages were inserted at ten times the original frequency, denoted by "fabrication 25 frequency times 10".

Real CAN Dataset

Table 7.1 gives the detection results of all algorithms on the real CAN dataset containing simulations of two different kinds of fabrication attacks. The AUC values in this table are fairly high, this means the algorithms show good detection results for this attack. Half of the results are equal to 1. Recall that an AUC value of 1 means perfect classification, i.e., no false-positives and all true-positives.

Algorithm	Attack	Fast AUC	Medium AUC	Slow AUC
One-Class SVM	Fabrication 1 before	0.75	1.00	1.00
Isolation Forest	Fabrication 1 before	0.85	0.96	1.00
Robust Covariance Estimator	Fabrication 1 before	0.85	1.00	1.00
One-Class SVM	Fabrication 25 frequency times 10	0.84	1.00	1.00
Isolation Forest	Fabrication 25 frequency times 10	0.92	0.93	1.00
Robust Covariance Estimator	Fabrication 25 frequency times 10	0.90	1.00	1.00

Table 7.1: Fabrication attack detection results on the real CAN dataset

Synthetic CAN Dataset

Table 7.2 gives the detection results of all algorithms on the synthetic CAN dataset containing simulations of two different kinds of fabrication attacks. This table shows different results than for the real CAN dataset. Errors occurred during the execution of the robust covariance estimator. The isolation forest gives an AUC of approximately 0.5 twice, this corresponds to the detection results of a random guess. A few experiments gave an AUC of 1.

Algorithm	Attack	Fast AUC	Medium AUC	Slow AUC
One-Class SVM	Fabrication 1 before	0.95	0.97	1.00
Isolation Forest	Fabrication 1 before	0.49	0.92	0.81
Robust Covariance Estimator	Fabrication 1 before	error	error	error
One-Class SVM	Fabrication 25 frequency times 10	0.94	1.00	1.00
Isolation Forest	Fabrication 25 frequency times 10	0.55	0.94	0.88
Robust Covariance Estimator	Fabrication 25 frequency times 10	error	error	error

Table 7.2: Fabrication attack detection results on the synthetic CAN dataset

7.1.2 Suspension Attack

The suspension attack simply consists of the deletion of messages. This attack was simulated as explained in Section 6.2. We varied the amount of messages that were deleted in our experiments. The cases where 1 or 25 messages have been deleted will be denoted by “suspension 1” or “suspension 25”, respectively.

Real CAN Dataset

Table 7.3 gives the detection results of all algorithms on the real CAN dataset containing simulations of two different kinds of suspension attacks. This table shows very different detection results for the various experiments. Recall that an AUC of 0.5 corresponds to the detection rate of a random guess. An AUC of 0 means that for all false-positive rates the true-positive rate is 0, hence the attack was not detected. An AUC of 1 corresponds to perfect classification, i.e., no false-positives and all true-positives. All of these values are shown in the table.

Algorithm	Attack	Fast AUC	Medium AUC	Slow AUC
One-Class SVM	Suspension 1	0.50	0.00	0.00
Isolation Forest	Suspension 1	0.95	0.90	0.60
Robust Covariance Estimator	Suspension 1	0.90	1.00	1.00
One-Class SVM	Suspension 25	1.00	1.00	0.80
Isolation Forest	Suspension 25	0.90	0.97	0.80
Robust Covariance Estimator	Suspension 25	1.00	1.00	0.90

Table 7.3: Suspension attack detection results on the real CAN dataset

Synthetic CAN Dataset

Table 7.4 gives the detection results of all algorithms on the synthetic CAN dataset containing simulations of two different kinds of suspension attacks. For these experiments errors occurred again for during the execution of the robust covariance estimator. Besides that, this table shows a fairly similar result as for the real CAN dataset.

Algorithm	Attack	Fast AUC	Medium AUC	Slow AUC
One-Class SVM	Suspension 1	0.20	0.00	0.00
Isolation Forest	Suspension 1	0.56	0.76	0.92
Robust Covariance Estimator	Suspension 1	error	error	error
One-Class SVM	Suspension 25	1.00	1.00	1.00
Isolation Forest	Suspension 25	0.65	0.90	0.63
Robust Covariance Estimator	Suspension 25	error	error	error

Table 7.4: Suspension attack detection results on the synthetic CAN dataset

7.1.3 Masquerade Attack

The last attack we simulated was the masquerade attack. As explained in Section 6.2 four versions of this attack were simulated. Firstly, the distinction was made between a small and large difference in clock skew. Additionally, the amount of messages that were affected was varied. A simulation on 1 or 25 messages with a small skew difference will be denoted as “masquerade 1 small skew” or “masquerade 25 small skew”, respectively. Similarly, for a large skew this will be “masquerade 1 large skew” or “masquerade 1 large skew”, respectively.

Real CAN Dataset

Table 7.5 gives the detection results of all algorithms on the real CAN dataset containing simulations of four different kinds of masquerade attacks. Most values shown in this table are between 0.5 and 1. As explained, 0.5 corresponds to the detection rate of a random guess and 1 is perfect classification, i.e., no false-positives and all true-positives. Not a single experiment resulted in an AUC of 1.

Algorithm	Attack	Fast AUC	Medium AUC	Slow AUC
One-Class SVM	Masquerade 1 small skew	0.89	0.86	0.60
Isolation Forest	Masquerade 1 small skew	0.88	0.31	0.60
Robust Covariance Estimator	Masquerade 1 small skew	0.66	0.50	0.80
One-Class SVM	Masquerade 1 large skew	0.89	0.86	0.60
Isolation Forest	Masquerade 1 large skew	0.80	0.25	0.60
Robust Covariance Estimator	Masquerade 1 large skew	0.68	0.50	0.70
One-Class SVM	Masquerade 25 small skew	0.89	0.35	0.70
Isolation Forest	Masquerade 25 small skew	0.76	0.64	0.60
Robust Covariance Estimator	Masquerade 25 small skew	0.67	0.62	0.30
One-Class SVM	Masquerade 25 large skew	0.86	0.12	0.70
Isolation Forest	Masquerade 25 large skew	0.81	0.93	0.50
Robust Covariance Estimator	Masquerade 25 large skew	0.69	0.82	0.70

Table 7.5: Masquerade attack detection results on the real CAN dataset

Synthetic CAN Dataset

Table 7.6 gives the detection results of all algorithms on the synthetic CAN dataset containing simulations of four different kinds of masquerade attacks. For these experiments errors occurred again for during the execution of the robust covariance estimator. The rest of the table shows slightly lower results than was given for the real CAN dataset.

7.1.4 Parameter Values

In Section 6.3.2 we identified the important parameters for each algorithm. For the one-class support vector machine these are the kernel and nu parameters. For the isolation forest and robust covariance estimator this is the contamination parameter, which has the same function as the nu parameter: specifying the fraction of anomalies in the training set.

For the one-class support vector machine we found that the linear kernel had the best performance. For the nu parameter we found that, to our surprise, the effect on the performance is very small. Different values of this parameters only produced very small differences in AUC values. That is why we decided to set nu = 0.01 for all experiments, this means that the algorithm considers 1 % of the training data to be an anomaly.

For the contamination parameter for the isolation forest and robust covariance estimator we also found that its effect was very small. Thus, we used the default value as given in Section 6.3.2.

Algorithm	Attack	Fast AUC	Medium AUC	Slow AUC
One-Class SVM	Masquerade 1 small skew	0.63	0.35	0.27
Isolation Forest	Masquerade 1 small skew	0.64	0.47	0.81
Robust Covariance Estimator	Masquerade 1 small skew	error	error	error
One-Class SVM	Masquerade 1 large skew	0.68	0.35	0.27
Isolation Forest	Masquerade 1 large skew	0.67	0.46	0.77
Robust Covariance Estimator	Masquerade 1 large skew	error	error	error
One-Class SVM	Masquerade 25 small skew	0.37	0.31	0.33
Isolation Forest	Masquerade 25 small skew	0.44	0.26	0.56
Robust Covariance Estimator	Masquerade 25 small skew	error	error	error
One-Class SVM	Masquerade 25 large skew	0.36	0.28	0.24
Isolation Forest	Masquerade 25 large skew	0.49	0.29	0.54
Robust Covariance Estimator	Masquerade 25 large skew	error	error	error

Table 7.6: Masquerade attack detection results on the synthetic CAN dataset

7.2 Result Analysis

Experiments have been conducted on two datasets, the real and synthetic CAN dataset. The results of the experiments were presented in the previous section. However, a number of observations has led to the decision to only analyse the results obtained for the real CAN dataset.

First, the robust covariance estimator did not work at all for the synthetic CAN dataset, we denoted this problem as "error" in the tables given above. Those experiments returned the following error message:

> ... ValueError: Singular covariance matrix. Please check that the covariance matrix corresponding to the dataset is full rank and that MinCovDet is used with Gaussian-distributed data (or at least data drawn from a unimodal, symmetric distribution. ...

A singular, i.e. not invertible, covariance matrix has a determinant that is equal to zero. A covariance matrix is singular when all covariances are equal to each other. When this is the case, factor analysis does not work. Hence, this error means that some variables in our data are so similar to each other, that other variables become redundant. This could mean that the synthetic CAN dataset does not have the same amount of variation as real CAN data, as this error did not arise for the other dataset.

Additionally, the performance of the isolation forest algorithm decreased drastically for a small amount of experiments on the synthetic CAN dataset. This decrease in performance only happened for fast synthetic messages. This observation and the error messages could be an indication that the simulated CAN data is not similar enough to real CAN data, hence our decision to only analyse the results of the experiments that were conducted on the real CAN dataset.

We can conclude that all attacks have been detected. Two experiments resulted in an AUC value of 0, which means the attack was not detected, but other algorithms did detect this attack. However, just detecting an attack is not enough, the amount of false-positives decide the usability of an algorithm, which will be analysed now.

To analyse the performance of the algorithms for the three different kinds of attacks we plot the detection results of all algorithms in one scatter plot per attack. The AUC values for fast,

medium and slow packets are indicated by a different symbol and a red cross indicates the average value of all AUC values per algorithm. This way we can easily inspect the best and worse case results together with the average performance of all algorithms in one figure. All variations of one type of attack are shown together in one figure.

Figure 7.1 shows the fabrication attack detection results. In this figure we see that all algorithms perform comparably well. Moreover, this is the attack for which all algorithms show the best detection results. This means that the insertion of one or more packets decreases the interval time enough to be detectable by the algorithms. We do see that the fast packets have a slightly lower AUC. This could be caused by the fact that these interval times are already small in the normal situation and thus the relative change is smaller than for the slower messages. The best performing algorithm in detecting fabrication attacks is the robust covariance estimator, but it should be noted that the other two algorithms perform only slightly worse.

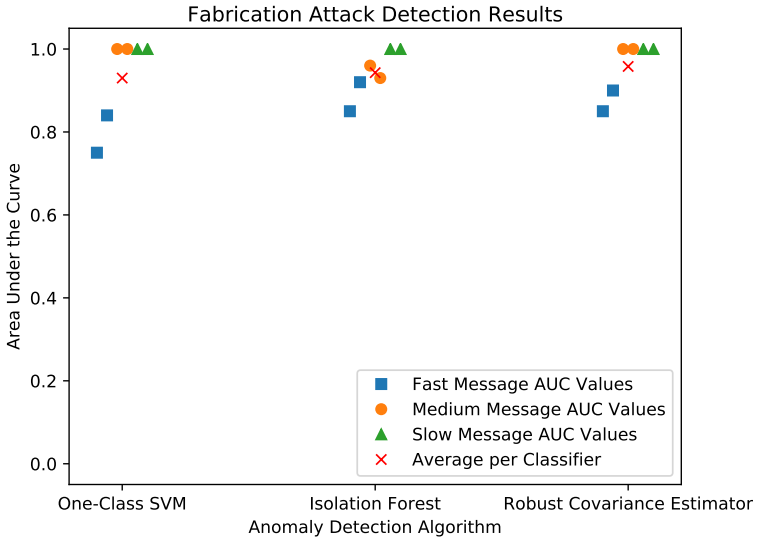

Figure 7.1: Fabrication attack detection results

Figure 7.2 shows the suspension attack detection results. Surprisingly, this figure shows more variation in the results than was shown in Figure 7.1. Apparently, the deletion of packets is harder to detect than the insertion of packets. The support vector machine even shows AUC values of 0 and 0.5 for the attack where one packet was deleted. An AUC value of 0 means that for all false-positive rates the true-positive rate is 0, hence the attack was not detected. One explanation could be that this is caused by the method of using windows during preprocessing, as described in Section 6.3.1, as the deleted packet could be at the beginning or the end of a window. The result of this would be that the mean interval does not change. However, this most likely is not an issue as the robust covariance estimator still shows good detection results here. Another explanation could be that the small variations caused by deleting one packet results in a dataset that strongly follows the Gaussian distribution. In this case, the support vector machine could not fully utilise the characteristics of Gaussian data. This is caused by the multi functionality of the algorithm, it has many parameters that can be tweaked to fit different situations, these were explained in Section 6.3.2. The kernel that is used has the biggest impact on anomaly detection performance and can be optimised by many different parameters. Hence, if all parameters are optimised for this situation, the algorithm can be able to detect this attack with a very high AUC. However, with the parameters used in these experiments the support vector machine performs poorly for this single attack, but has a good overall performance. Unfortunately, it would be impossible to build a model that would be suitable for all types of attacks.

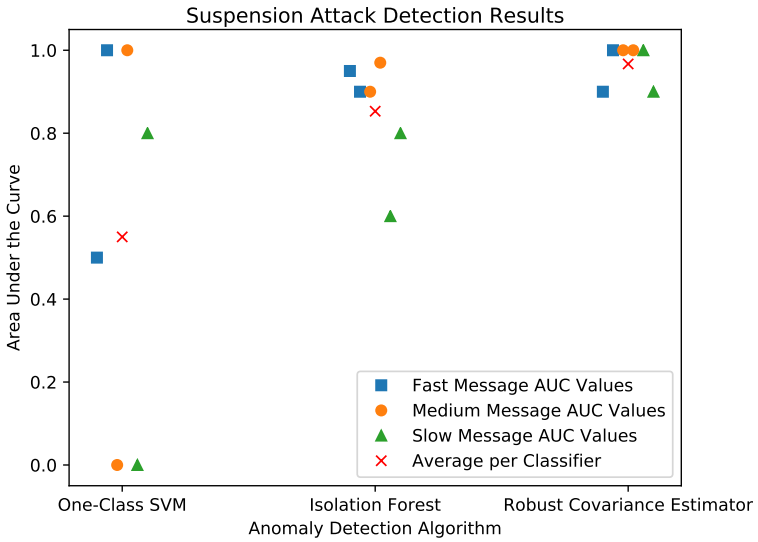

Figure 7.2: Suspension attack detection results

Where the support vector machine shows much worse results, the robust covariance estimator shows even better results than for the fabrication attacks. If the second explanation of the support vector machine's bad performance is correct, i.e. the dataset resulting from a suspension attack strongly follows the Gaussian distribution, this could be the cause of the robust covariance estimator's good performance. This algorithm's model is a Gaussian distribution that aims to capture the dataset as closely as possible.

The isolation forest's main strong point is that it performs well on high dimensional data. Since the feature vectors, that serve as input to the algorithms contain a dimension for each ID, the amount of dimensions corresponds to the numbers given in Table 6.2. The fact that these numbers are fairly high could explain the fairly constant good performance of the isolation forest.

For the suspension attack it is very clear that the robust covariance estimator has the best performance of the three algorithms and this time the differences are a lot larger than for the fabrication attack.

Figure 7.3 shows the masquerade attack detection results. As this is the most sophisticated attack it comes as no surprise that the performance in noticeably worse than for the previous two attacks. All algorithms have a lot of variation in their results and the average AUC values are lower than for the other two attacks.

We do see that for all four attacks the fast messages have good results, while the other types of messages have a lot of variation. One reason for this could be that the interval times of fast messages are small, so a clock offset caused by skew is a relatively large difference.

The support vector machine has the highest average AUC value, so we conclude it has the best performance in detecting masquerade attacks. However, similarly to the fabrication attack, it should be noted that the detection results of the three algorithms are very close.

After comparing the performance of all algorithms in detecting the three different kinds of simulated attacks we can come to a conclusion. The support vector machine gives both very good and very bad results, depending on the type of the attack and the amount of messages affected by the attack. The isolation forest gives reasonable results in all cases. There is no case where it has the worst performance, but it also never has the best performance. Very good results were given by the robust covariance estimator. It clearly outperforms the two other algorithms for the suspension attack, performed slightly better for the fabrication attack and barely any worse for the masquerade attack.

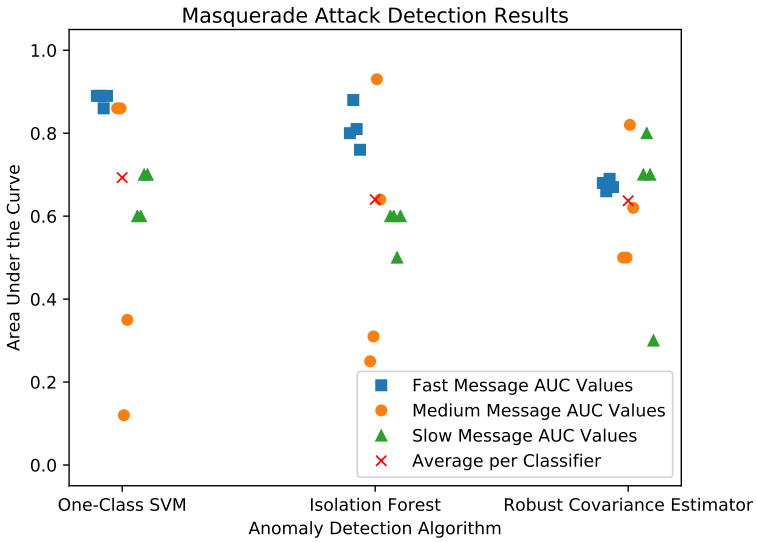

Figure 7.3: Masquerade attack detection results

7.3 Limitations

During the collection of CAN data we encountered a number of problems (see Section 6.1 for a description of the devices and methods we used). The main reason the devices we tested failed was that we required the timestamps of the packets to be logged in microseconds. As a consequence, we decided to use the CAN dataset made available by the University of Tulsa for the experiments. This dataset is fairly small and only contains two and a half minutes of CAN traffic captured during normal operation of a car. Due to the limited amount of data used for the experiments, our findings cannot be generalised. In particular, the algorithms could give different results on other datasets and the robust covariance estimator might not be superior in all cases.

From our analysis it became clear that the masquerade attack is very difficult to detect reliably, i.e., with a low false-positive rate. The goal of a masquerade attack is to cover up the fact that ECUs have been compromised and the visible results of this are very small changes in timestamps. As our algorithms' detection capabilities are based on timing information, this type of attack is very hard to detect.

Our algorithms considered all recurring messages. This was already an improvement on past research, where only fast recurring messages were investigated [74]. However, non-recurring messages were not investigated. As a result, any attack executed on this type of messages will not be detected by our algorithms. Furthermore, we also encountered some problems during our experiments regarding recurring messages. The real CAN dataset we used for our analysis contained different behaviour at the beginning and the end of the log, which caused the algorithms to not work properly.

The algorithms tested by us also do not have any fingerprinting capabilities. Per window it is assessed if an attack has occurred and that is the only information the algorithms can give about possible anomalies: a time frame. No information about the ID or contents of the message under attack can be determined.

The tested algorithms only have detection capabilities, they do not have any prevention options. When an anomaly is detected, the algorithms are able to give a notification about this, but no further actions can be taken to prevent the attack from continuing. The issue of prevention capabilities in a safety critical system as a car has been discussed in Chapter 4. The analysed IDSs had different methods for this, e.g. putting the car in limp mode when an attack is detected, but the problem has no trivial solution.

Chapter 8

Conclusion

In this thesis we researched the effectiveness of anomaly detection techniques on a car's internal networks, specifically the Controller Area Network (CAN). The main research question of this thesis was: *How can cyber-attacks on modern automotive vehicles be detected at CAN bus level using anomaly detection techniques?* We divided this question into two sub questions. The first sub question is: *What does a cyber-attack look like at CAN bus level?* After we answered that question we could conduct a number of experiments to answer the second sub question: *Which anomaly detection algorithm gives the best results at CAN bus level?*

We started by describing preliminary concepts that are crucial for the understanding of the research questions, the methodology and interpreting the results. First, we described different internal networks present in a modern car and gave a detailed explanation of CAN. In addition, various external interfaces and modern applications were investigated.

Once these aspects were clear we gave a detailed explanation of many possibilities for executing an attack on a modern car. Starting with various entry points and ending with the manipulating CAN traffic to control internal systems. To identify general attack forms, existing attacks were analysed. With this information we could answer the first sub question, we described three general attacks that are observable on the CAN bus: the fabrication, suspension and masquerade attack.

Various existing intrusion detection systems were analysed and compared. The research that was identified as most promising uses a one-class support vector machine to detect attacks on CAN traffic using network flows based on packet frequencies. The one-class support vector machine is a so-called one-class classification algorithm, meaning that it is able to learn a model from data containing only normal behaviour and use this model to detect anomalies. We identified two more promising one-class classification algorithms: the isolation forest and robust covariance estimator.

We devised a number of experiments to compare the anomaly detection performance of these three algorithms. We attempted to collect CAN traffic data from a car using a number of devices. Unfortunately, we did not manage to do this ourselves. None of our methods were able to log the traffic with timestamps in microseconds. Since the algorithms base their detection on timing information this was a crucial requirement. In the end we used one dataset available online containing real CAN data and one dataset we created using a CAN simulator. On these datasets, the three attacks were simulated. The three algorithms were used to classify the datasets containing attacks and the results were compared. We came to the conclusion that the robust covariance estimator gave the best results in detecting attacks on CAN traffic data. With this we answered the second sub question. During our experiments we also saw opportunities for improvements and future work.

Having answered both sub questions, we can formulate an answer to the main research question. Using a one-class classification algorithm a model can be built that describes the normal operating circumstances of the CAN bus. This model can then be used to classify new data and identify anomalies, i.e. attacks, occurring in the traffic. In our experiments the robust covariance estimator was the one-class classification algorithm that gave the best anomaly detection results in this setting.

8.1 Improvements

During our analysis we noticed a number of areas in which our algorithms could be improved. These improvements are mainly aimed at optimising the algorithms' resource usage.

The part of the algorithms that consumed the most time was preprocessing the CAN data. As described in Section 6.3.1 during preprocessing the CAN log is translated to feature vectors for each type of message. These feature vectors are the input for the algorithms. For a CAN log containing approximately five minutes of data, preprocessing took nine minutes. For our research this was no problem, as this was done once and training the algorithms and classifying new data can be done almost instantly. For larger datasets however, this could be too time consuming. Luckily, we identified two aspects where the performance can be improved by introducing parallelisation. First, as the fast, medium and slow messages are evaluated independently this could be executed in parallel. Additionally, calculating metrics of separate windows can also be executed in parallel. Moreover, windows have overlap, so once a window is processed, half of the calculated metrics can be used in the next window. This eliminates half of all the calculations.

Another aspect that could be improved is that for our analysis for each dataset we trained three different classifiers, one for each type of message. This is done to be able to analyse all recurring messages. As a result, to execute an algorithm on a dataset, all three classifiers have to be used to classify all recurring messages. For our analysis this was not a problem, but this can be improved by combining the three different classifiers into one classifier. The Python Scikit Learn packet that we used has methods available for this, which they call ensembles [68].

8.2 Future Work

Although our experiments indicate that the robust covariance estimator has potential for the detection of anomalies in CAN traffic, more experiments are needed to validate our findings. In particular, the approaches should be tested on other CAN datasets, for instance, from vehicles of different brands and models or encompassing more varied behaviour.

In our experiments we executed all anomaly detection algorithms offline. To turn the concepts researched in this work into an Intrusion Detection System (IDS) that can be deployed in a car, to detect attacks on live traffic, a number of changes have to be made.

First of all, the improvements explained in the previous section are required, to make preprocessing fast enough to ensure all incoming traffic can be analysed in real time. Additionally, the algorithms have to be adapted to accept a stream of incoming data, instead of analysing a log file containing traffic. An example of how this can be done is by using a library called Streamparse which offers the functionality to execute Python code on a real-time stream of data [57].

In that setting, traffic on the CAN bus is only observed, i.e., the data does not have to be processed by the IDS before it can be delivered to the receiving ECUs. This is easily implementable as on CAN all packets are already broadcast.

When anomaly detection in a car is possible, this can be combined with prevention. Hence, instead of only giving a notification when an anomaly is detected, also taking an action to prevent (further) execution of an attack. An example of an action that could be taken is putting the car in limp mode as proposed by Miller and Valasek [47].

A different area for future work was discovered during the analysis of the results, where we concluded that the synthetic CAN data was not realistic enough, due to a number of peculiarities in the results. As data collection proved to be very difficult, a realistic CAN simulator would be very helpful in generating large amount of train and validation data. Hence, there exists a need for CAN simulation tools able to generate more realistic data.

Bibliography

[1] ISO 11898:1993(E). Road Vehicles – Interchange of Digital Information – Controller Area Network (CAN) for High-Speed Communication. Standard, International Organization for Standardization, Geneva, CH, 1993. 8

[2] Arduino. Arduino Due. https://www.arduino.cc/en/Main/ArduinoBoardDue. Accessed 28-03-2017. 40

[3] Arduino. Arduino Uno. https://www.arduino.cc/en/Main/ArduinoBoardUno. Accessed 08-03-2017. 40

[4] Blackberry. QNX Operating System. http://www.qnx.com/content/qnx/en.html. Accessed 03-04-2017. 2

[5] Bluetooth Special Interest Group. Bluetooth Basic Rate/Enhanced Data Rate. https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/br-edr. Accessed 09-12-2016. 14

[6] Bluetooth Special Interest Group. Bluetooth Technology Website. https://www.bluetooth.com/about-us. Accessed 14-12-2016. 14

[7] Bosch. CAN Specification Version 2.0, 1991. 7, 8, 9, 11

[8] CAR 2 CAR Communication Consortium. CAR 2 CAR Communication Consortium: Mission & Objectives. https://www.car-2-car.org/. Accessed: 25-02-1017. 15

[9] R. N. Charette. This Car Runs on Code. http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code, 2009. Accessed: 24-10-2016. 1

[10] S. Checkoway, D. Mccoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *20th USENIX Security Symposium (USENIX Security 11)*. USENIX Association, 2011. 2, 12, 14, 19

[11] K. Cho and K. G. Shin. Fingerprinting Electronic Control Units for Vehicle Intrusion Detection. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 911–927. USENIX Association, 2016. 3, 20, 21, 22, 26, 27, 30, 42

[12] Code White. CANBadger. https://github.com/Gutenshit/CANBadger. Accessed 08-03-2017. 40

[13] Ford Motor Company. Ford Sync met Emergency Assistance. http://www.ford.nl/OverFord/Technologie/Veiligheid/Ford-SYNC-met-Emergency-Assistance. Accessed 14-12-2016. 15

[14] C. Cortes and V. Vapnik. Support–Vector Networks. *Machine Learning*, 20(3):273–297, 1995. 33

[15] Council of European Union. Council Directive 98/69/EC relating to measures to be taken against air pollution by emissions from motor vehicles, 1998. OJ L350/1. 13

[16] Council of European Union. Council Regulation 661/2009/EC concerning type-approval requirements for the general safety of motor vehicles, their trailers and systems, components and separate technical units intended therefor, 2009. OJ L200/1. 14

[17] B. P. Crow, I. Widjaja, L. G. Kim, and P. T. Sakai. IEEE 802.11 Wireless Local Area Networks. *IEEE Communications Magazine*, 35(9):116–126, 1997. 14, 15

[18] Dearborn Group. Gryphon S3. `http://www.dgtech.com/product/gryphon-s3/gryphon-s3.php`. Accessed 28-03-2017. 40

[19] Dearborn Group. Hercules Software. `http://www.dgtech.com/product/hercules/hercules.php`. Accessed 28-03-2017. 40

[20] M. Farsi, K. Ratcliff, and M. Barbosa. An Overview of Controller Area Network. *Computing & Control Engineering Journal*, 10:113–120, 1999. 10

[21] FCA Group LLC. Uconnect Access for Chrysler, Dodge, FIAT , Jeep & Ram. `http://www.driveuconnect.com/`. Accessed 18-11-2016. 1

[22] Fiat Chrysler Automobile. Statement: Software Update. `http://media.fcanorthamerica.com/newsrelease.do?&id=16849&mid=1`, 2015. Accessed: 18-11-2016. 2

[23] FlexRay Consortium. FlexRay Communications System Protocol Specification, 2010. 8

[24] Python Software Foundation. Python. `https://www.python.org/about`. Accessed 28-03-2017. 39

[25] A. Francillon, B. Danev, and S. Capkun. Relay Attacks on Passive Keyless Entry and Start Systems in Modern Cars. In *18th Annual Network and Distributed System Security Symposium. The Internet Society*, 2011. 19

[26] Gorden, G. Support Vector Machines and Kernel Methods. `https://www.cs.cmu.edu/~ggordon/SVMs/new-svms-and-kernels.pdf`, 2004. Accessed: 15-04-2017. 34

[27] A. Greenberg. Hackers Reveal Nasty New Car Attacks – With Me Behind The Wheel. `http://www.forbes.com/sites/andygreenberg/2013/07/24/hackers-reveal-nasty-new-car-attacks-with-me-behind-the-wheel-video/#4cd8eb795bf2`, 2013. Accessed: 13-10-2016. 1

[28] A. Greenberg. Hackers Could Take Control of Your Car. This Device Can Stop Them. `https://www.wired.com/2014/07/car-hacker/`, 2014. Accessed: 14-10-2016. 3, 23

[29] A. Greenberg. Hackers Remotely Kill a Jeep on the Highway – With Me in It. `https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/`, 2015. Accessed: 13-10-2016. 1

[30] A. Greenberg. The Jeep Hackers Are Back to Prove Car Hacking Can Get Much Worse. `https://www.wired.com/2016/08/jeep-hackers-return-high-speed-steering-acceleration-hacks/`, 2016. Accessed: 13-10-2016. 2

[31] J. A. Hanley and B. J. McNeil. The Meaning and Use of the Area Under a Receiver Operating Characteristic (ROC) Curve. *Radiology*, 143(1):29–36, 1982. 47

[32] N. Herold, S. A. Posselt, O. Hanka, and G. Carle. Anomaly Detection for SOME/IP Using Complex Event Processing. In *NOMS 2016 – 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 1221–1226, 2016. 3

[33] T. Hoppe, S. Kiltz, and J. Dittmann. Security Threats to Automotive CAN Networks – Practical Examples and Selected Short-Term Countermeasures. *Reliability Engineering & System Safety*, 96(1):11–25, 2011. 1, 2, 20

[34] National Instruments. Controller Area Network (CAN) Overview. http://www.ni.com/white-paper/2732/en/, 2014. Accessed: 24-10-2016. 8

[35] D. Jiang and L. Delgrossi. IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments. In *VTC Spring 2008 – IEEE Vehicular Technology Conference*, pages 2036–2040, 2008. 15

[36] M. I. Jordan and R. Thibaux. The Kernel Trick. http://www.cs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf, 2004. Accessed: 02-03-2017. 34

[37] E. Kim. Everything You Wanted to Know about the Kernel Trick (But Were Too Afraid to Ask). http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html, 2013. Accessed: 25-02-2017. 34

[38] D. Kuschel. CANBus Triple. https://canb.us. Accessed 08-03-2017. 39

[39] LIN Consortium. LIN Specification Package Revision 2.2A LIN Specification Package Revision 2.2A, 2010. 8

[40] F. T. Liu, K. M. Ting, and Z. H. Zhou. Isolation-Based Anomaly Detection. *ACM Transactions on Knowledge Discovery from Data*, 6(1):3:1–3:39, March 2012. 35, 36

[41] M. Marchetti, D. Stabili, A. Guido, and M. 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*, pages 1–6, 2016. 25

[42] N. McCarthy. Connected Cars By The Numbers. http://www.forbes.com/sites/niallmccarthy/2015/01/27/connected-cars-by-the-numbers-infographic/#3b74d38829ce, 2015. Accessed: 11-11-2016. 1

[43] Microchip. ATmega328P. http://www.microchip.com/wwwproducts/en/atmega328p. Accessed 28-03-2017. 40

[44] Microchip. ATmega32U4. http://www.microchip.com/wwwproducts/en/atmega32u4. Accessed 28-03-2017. 39

[45] Microchip. MCP2515. http://www.microchip.com/wwwproducts/en/en010406. Accessed 28-03-2017. 39, 40

[46] Microchip. MCP2551. http://www.microchip.com/wwwproducts/en/en010405. Accessed 28-03-2017. 40

[47] C. Miller and C. Valasek. A Survey of Remote Automotive Attack Surfaces. *Blackhat USA*, pages 1–90, 2014. 24, 25, 58

[48] C. Miller and C. Valasek. Advanced CAN Injection Techniques for Vehicle Networks. *Blackhat USA*, 2016. 20

[49] Most Cooperation. MOST Specification Rev. 3.0 Errata 2, 2010. 8

[50] Murison, B. Indicator Functions. https://turing.une.edu.au/~stat354/notes/node16.html. Accessed: 20-04-2017. 47

[51] National Highway Traffic Safety Administration. Federal Motor Vehicle Safety Standards; Tire Pressure Monitoring Systems; Controls and Displays. https://one.nhtsa.gov/cars/rules/rulings/TPMSfinalrule.6/TPMSfinalrule.6.html, 2005. 14

[52] Nice, K. How Car Computers Work. http://auto.howstuffworks.com/under-the-hood/trends-innovations/car-computer1.htm. Accessed: 12-04-2017. 30

[53] P. Nisch. Security Issues in Modern Automotive Systems, 2011. 2

[54] NXP Semiconductors. LPCXpresso board for LPC1769. http://www.nxp.com/products/software-and-tools/software-development-tools/software-tools/lpcxpresso-boards/lpcxpresso-board-for-lpc1769-with-cmsis-dap-probe:OM13085. Accessed: 15-04-2017. 40

[55] The University of Tulsa. Crash Reconstruction Research Consortium. http://tucrrc.utulsa.edu. Accessed 28-03-2017. 39

[56] OpenGarages. Instrument Cluster Simulator for SocketCAN. https://github.com/zombieCraig/ICSim. Accessed 28-03-2017. 39, 41

[57] Parsely. Streamparse. https://github.com/Parsely/streamparse. Accessed: 13-04-2017. 58

[58] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 39, 44

[59] C. Poulin. Driving Security: Cyber Assurance for Next-Generation Vehicles, 2014. 2

[60] M. du Preez. OBD II Diagnostic Interface Pinout Diagram. http://pinoutguide.com/CarElectronics/car_obd2_pinout.shtml. Accessed 11-11-2016. 13

[61] Preis, B. R. *Data Structures and Algorithms with Object-Oriented Design Patterns in Java.* Wiley, 1999. 36

[62] I. Rouf, R. Miller, H. Mustafa, T. Taylor, S. Oh, W. Xu, M. Gruteser, W. Trappe, and I. Seskar. Security and Privacy Vulnerabilities of In-Car Wireless Networks: A Tire Pressure Monitoring System Case Study. In *19th USENIX Security Symposium (USENIX Security 10*. USENIX Association, 2010. 19

[63] P. J. Rousseeuw. Least Median of Squares Regression. *Journal of the American Statistical Association*, pages 871–880, 1984. 37

[64] P. J. Rousseeuw. Multivariate Estimation With High Breakdown Point. *Mathematical Statistics and Applications*, 8:283–297, 1985. 37

[65] P. J. Rousseeuw and K. Van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, August 1999. 37

[66] F. Sagstetter, M. Lukasiewycz, S. Steinhorst, M. Wolf, A. Bouard, W. R. Harris, S. Jha, T. Peyrin, A. Poschmann, and S. Chakraborty. Security Challenges in Automotive Hardware/Software Architecture Design. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 458–463. EDA Consortium, 2013. 7

[67] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt. Support Vector Method for Novelty Detection. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 582–588. MIT Press, 1999. 33, 35

[68] Scikit-learn. Ensemble Methods. `http://scikit-learn.org/stable/modules/ensemble.html`. Accessed: 20-04-2017. 58

[69] Shore, J. Classical and Robust Tolerance Ellipse. `https://tr8dr.files.wordpress.com/2010/09/screen-shot-2010-09-24-at-6-23-30-pm.png`. Accessed: 18-04-2017. 38

[70] SparkFun. CAN-BUS Shield. `https://www.sparkfun.com/products/13262`. Accessed 08-03-2017. 40

[71] SparkFun. SparkFun CAN-Bus Arduino Library. `https://github.com/sparkfun/SparkFun_CAN-Bus_Arduino_Library`. Accessed 28-03-2017. 40

[72] Statista. Number of Vehicles in Use Worldwide 2014. `https://www.statista.com/statistics/281134/number-of-vehicles-in-use-worldwide/`. Accessed: 18-11-2016. 1

[73] J. A. Swets, R. M. Dawes, and J. Monahan. Better Decisions Through Science. *Scientific American*, 283:82–87, 2000. 46

[74] A. Taylor, N. Japkowicz, and S. Leblanc. Frequency-Based Anomaly Detection for the Automotive CAN bus. In *2015 World Congress on Industrial Control Systems Security*, pages 45–49, 2015. 3, 28, 33, 43, 56

[75] A. Valdes and S. Cheung. Communication Pattern Anomaly Detection in Process Control Systems. In *2009 IEEE Conference on Technologies for Homeland Security*, pages 22–29, 2009. 28, 43

[76] Walz, K. ROC Curve. `https://upload.wikimedia.org/wikipedia/commons/archive/3/36/20101011161259!ROC_space-2.png`, 2010. Accessed: 04-04-2017. 46

[77] WikiLeaks. Vault 7: Embedded Development Brach Meeting Notes. `https://wikileaks.org/ciav7p1/cms/page_13763790.html`, October 2014. Accessed 03-04-2017. 2

[78] Xoneca. OBD connector shape. `https://commons.wikimedia.org/wiki/File:OBD_connector_shape.svg`, 2014. Accessed: 11-11-2016. 13

[79] Zander, S. and Murdoch, S. J. An Improved Clock-skew Measurement Technique for Revealing Hidden Services. In *17th USENIX Security Symposium (USENIX Security 8)*, pages 211–226. USENIX Association, 2008. 27

[80] H. Zimmermann. OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980. 11