2018

Detection of injection attacks on in-vehicle network using data analytics

Sri Lalitha Dakshayani Dhulipala
Iowa State University

Detection of injection attacks on in-vehicle network using data analytics

by

Sri Lalitha Dakshayani Dhulipala

(Srilalithadaksh Dhulipala)

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering (Computing and Networking Systems)

Program of Study Committee:
Lotfi Ben Othmane, Co-major Professor
Manimaran Govindarasu, Co-major Professor
Doug Jacobson

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2018

DEDICATION

I would like to dedicate this thesis to my parents Satyanarayana and Padmavathi Dhulipala and my sister Sarvani without whose support I would not have been able to complete this work.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, I thank Dr. Lotfi Ben Othmane for his guidance, patience and support throughout this research and the writing of this thesis. His insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education. I would also like to thank my committee members for their efforts and contributions to this work: Dr. Manimaran Govindarasu and Dr. Doug Jacobson. I would additionally like to thank my team mate Sudharrshan Janarthanan for his help during the experimentation phase of the project.

I thank Dr. Nicholas Multari, Professor Ashfaq Khokhar, and Professor Bharat Bhargava for supporting the work financially. My work was mainly funded by the Pacific Northwest National Laboratories through the Iowa State University Information Assurance Center. Equipment were funded by the Qatar National Research Fund and The ECpE Department Excellence fund.

ABSTRACT

We investigate the possibility of detection of injection attacks using data analytics techniques in this thesis. The automotive industry is innovating the modern vehicles towards connectivity by interfacing them with various external entities. These entities are exposing the automobile to cyber attacks instead of ensuring its safety. Therefore it is important to consider the security aspect while developing these interfaces. Firstly, we try understand the automobile network architecture and the possible security threats associated with it. Next, we examine the various possible cyber-attacks on automobiles described in the literature. We experiment and analyze the attack scenarios by performing injection attacks on a vehicle. We collect the data during the injection attacks and apply multiple data analysis techniques. These techniques build a model based on data during normal operation. The observations from the data collected during injection attacks is fit into these techniques. The data points that do not fit the model are termed as attack points. Finally we examine and analyze the results and their accuracy in detecting injection attacks.

CHAPTER 1. INTRODUCTION

Automobiles have become an integral part of the modern living. The technology has advanced from animal driven carriages to self-driving autonomous cars. The modern automobile is an increasingly complex network of computer systems. Cars are no longer analog, mechanical contraptions. Today, even the most fundamental vehicular functions have become computerized.

Modern automobiles connect to external entities and offer access to the in-vehicle network. The Controlled Area Network or the CAN network is the in- vehicle network which communicates with the external world through Cellular network, Bluetooth , Wi-Fi etc. Due to the absence of security mechanisms in the CAN network, the automobile is prone to attacks through these external entities connected to the system.

An attacker can gain access to the network through any of the above external entities and replicate messages on the network. The vehicles cannot differentiate between legitimate in-vehicle messages and malicious ones. The impact of these attacks can vary from harmless honking to fatal accidents. With the advances in technology moving towards autonomous vehicles, it is important to secure the automobile network from such attacks. Also the wide spread of tools and knowledge to perform cyber-attacks on connected vehicles is posing a threat to the safety of the automobile users.

1.1 Problem

The CAN network implemented in these automobiles has inherent security flaws. It lacks encryption and device authentication which gives easy access to the intruders. Also, the broadcast nature of the bus readily provides the data from safety critical systems as well on the network. The data from theses systems can be monitored, modified and malicious data can also be injected.

Therefore, it is important to have a mechanism to detect the attacks on the CAN network. Through this thesis the following question would be addressed:

Can we detect injection attacks on an automobile network ?

1.2 Threat model

Automobiles have several functional units which directly or indirectly provide easy access to the network. These access points are called attack vectors. A collection of attack vectors is called as an attack surface. Broadly the attack surfaces can be classified into three categories:

- Direct physical access to CAN
- Over the air - GPS, Bluetooth, Wi-Fi, Cellular and Vehicle to Vehicle networks
- ECU - Firmware update

Each of these attack surfaces has multiple attack vectors through which the intruder can inject data on to the network. The scope of this thesis is limited to the direct physical access. The physical access is obtained through the OBD - II port.

1.3 Approach

The goal of this project is to detect injection attacks on CAN network by emulating the attack scenarios applying Data analysis techniques to the attack data and detecting an attack. The attacks are performed on the tachometer and speedometer but can be replicated to several other units as well.

To begin with, the CAN data is monitored to examine how the message passing takes places. Then the data is collected from the tachometer and speedometer. The data is modified and it is injected onto the network. Then the data is again monitored and recorded during the injection attack. Finally, the data collected during the attack scenario is compared with the normal functioning data using data analysis techniques to detect an attack situation.

Five data analysis techniques were applied to the CAN data. They are broadly divided into three categories:

- Correlation techniques
  - Pearson Correlation
  - K Means Clustering
- Causality
  - Hidden Markov Model
- Time Series Analysis
  - Moving Average Technique

1.4 Hardware

All the automobiles manufactured after 2008 must contain CAN bus by federal law. So this experiment can be replicated on any automobile which contains CAN bus. We have used a 2017 Ford Transit model to perform the experiment. The data collection and injection was performed with the help of Raspberry Pi3 and Pi CAN 2 modules.

1.5 Organization

This thesis is organized into nine chapters. Chapter 2 is Background which discusses the architecture of CAN bus. Chapter 3 is Related Work which discusses the previous work in the literature. Chapter 4 is dedicated to the Experimentation setup and Methodology. The next three chapter discuss a data analysis technique, how is it applied to our data set and the results obtained. Chapter 5 discusses the Data Analysis techniques used. Chapter 8 is Discussion of results and the outcomes. Chapter 9 is the conclusion.

CHAPTER 2. BACKGROUND

This chapter provides the fundamental concepts of the CAN network. The automobile network architecture section describes how various components in an automobile are connected and how they function in tandem. The second section describes the CAN network architecture. The next section discusses On-Board Diagnostic standard. The last two sections discuss the limitations of CAN bus and the attack surfaces in an automobile.

2.1 Automobile Network Architecture

2.1.1 ECUs

The automobile network comprises of multiple dedicated units called as the Electronic Control Units (ECUs). These units are the nodes connected to the network to enable exchange of data between them. Since it is physically impossible to connect each unit with every other unit, they are connected in sub-networks which integrates into a common master network. Each unit broadcasts the information onto this bus and the unit which needs to respond, receives the information by an identifier. Some, typical ECUs present in every automobile are

- Transmission Control Unit
- Engine Control Unit
- Brake Control Module
- Speed Control Unit
- Telematics control Unit
- Battery Management system

Previously, these units were connected using a master-slave network called LIN bus [3]. The LIN bus was low speed at 20 KBps. It was not fault tolerant. Nowadays, the safety critical systems like Brake control, engine control etc are connected on a High-Speed bus and LIN bus is being used for non-critical subsystems in the units like mirror adjustment, window roll down etc.

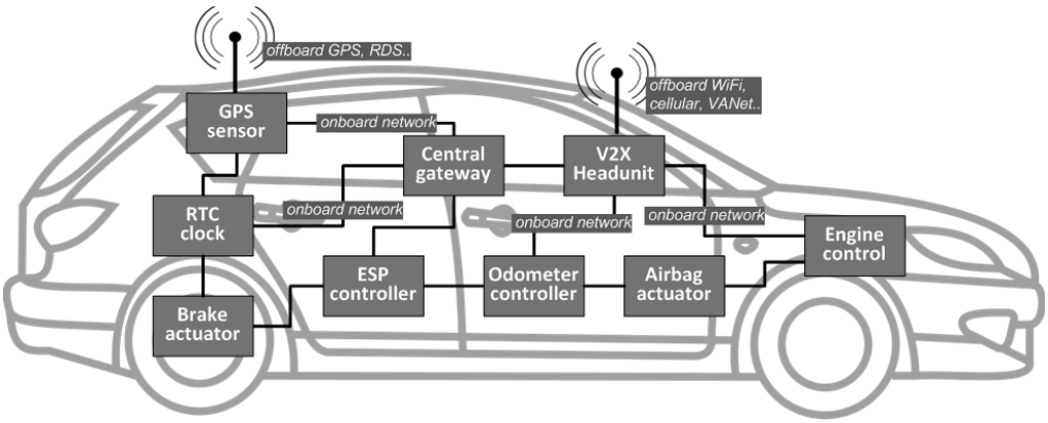

Figure 2.1 ECUs in an automobile [1]

2.2 CAN Bus

2.2.1 History

CAN or Controlled area network is a networking bus primarily used to communicate between multiple micro-controllers. It allows embedded devices to communicate with each other without a host computer. The modern-day automobiles as discussed earlier are not mere mechanical devices, but a complex system of control units connected by the CAN bus.

Robert Bosch began the development of the CAN bus in 1983. The protocol was officially released in 1986 at the Society of Automotive Engineers (SAE) conference in Detroit, Michigan. CAN bus is one of five protocols used in the on-board diagnostics (OBD)-II vehicle diagnostics standard. The OBD-II standard has been mandatory for all cars and light trucks sold in the

United States since 1996. The EOBD standard has been mandatory for all petrol vehicles sold in the European Union since 2001 and all diesel vehicles since 2004.

2.2.2 Message frame format

A CAN network can be configured to work with two different message formats: the standard or base frame format, and the extended frame format (described by CAN 2.0 B). These message formats are also known as frames. The extended format has an 18-bit identifier extension in addition to the 11-bit identifier field found in the 11- bit format. The frame format for a 11-bit CAN message is shown in Figure 2.2 below:

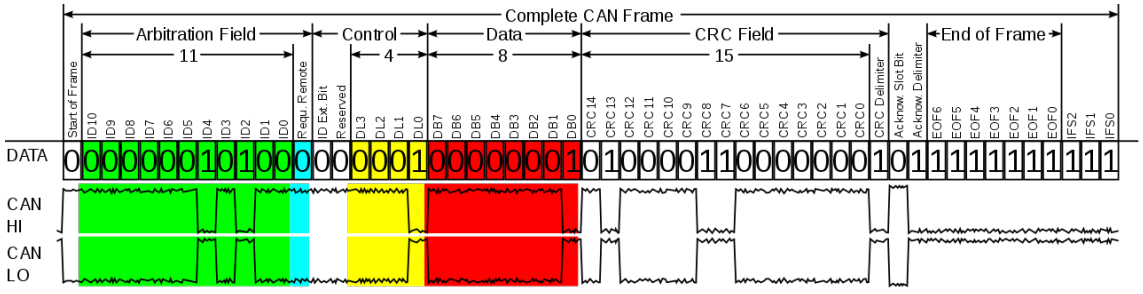

Figure 2.2 CAN Bus Frame Format

The CAN arbitration ID is an 11-bit field and the most important field. This ID is used to identify different devices on the CAN bus and to prioritize messages. These bits allow the attacker to filter the unwanted packets, replicate and spoof a legitimate packet.

The next important field, the CAN data field, can be 0 to 64 bits (8 bytes) in length. It holds the bits that direct the unit to perform the function. The functionality of the units can be understood by sniffing the data and varying the bits on this field.

The above figure is the general data frame. There a four types of frame formats:

- General frame: a frame used to exchange data between the nodes.
- Remote frame: a frame requesting the transmission of a specific identifier

- Error frame: a frame transmitted by any node detecting an error
- Overload frame: a frame to inject a delay between data or remote frame.

CAN is a multi-master serial bus standard for connecting Electronic Control Units. CAN works on CSMA/CD (Carrier Sense Multiple Access/Collision Detection) principle, where a single carrier is shared among number of ECUs. All nodes (ECUs) are connected to each other through a two-wire bus. One wire represents the CAN high or CANH and the other is CAN low or CANL. All nodes are connected to each other through a two wire bus. The two wires have an impedance of 120 ohms.

2.3 OBD

OBD stands for On Board Diagnostics. This is a functionality implemented in the automobiles which gives them the ability to report malfunctions and self-diagnose the errors. The OBD system helps the owner/mechanic to understand which unit or subsystem is causing errors. Currently, several hand-held, plug and play devices are available in the market which can be connected to the automobile to obtain real time monitoring of several ECUs.

2.3.1 History

The first On Board Computer System appeared in 1968 when Volkswagen's fuel injected Type-3 model. Later for almost two decades, several manufacturers implemented OBD without any standardization. In 1988, SAE introduced a standardized diagnostic port connector and a set of protocols for OBD. California Air Resources Board (CARB) made it mandatory for all vehicles sold in California to have some OBD capability. This was OBD - I. In 1994, CARB standardized the current iteration of OBD, known as OBD-II, and mandated that it be included on all new cars sold in California. By 1996, OBD-II was mandated nationwide in the United States.

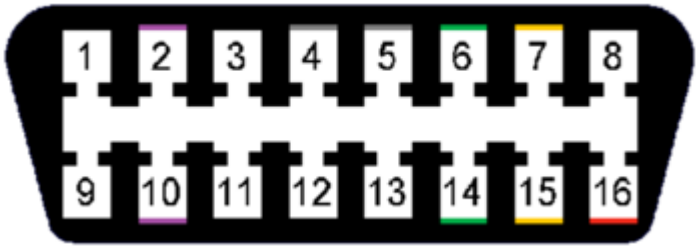

Figure 2.3 OBD pin-out diagram

2.3.2 OBD-II

The OBD-II standard specifies the type of diagnostic connector and its pin-out, signalling protocols and the format of messages. It also provides a list of vehicle parameters called **Diagnostic Trouble Codes (DTCs)** to monitor and diagnose malfunctions. OBD-II port is a 16-pin port usually located near the steering wheel. This port gives direct access to the CAN network. The table 2.1 below provides the pin specification:

Table 2.1 OBD-II Pin Specification

Pin	Function	Pin	Function
1	Manufacture Specific	9	Manufacture Specific
2	J1850 Bus (+)	10	J1850 Bus (-)
3	Manufacturer Specific	11	Manufacturer Specific
4	Ground	12	Manufacturer Specific
5	Ground	13	Manufacturer Specific
6	CAN High	14	CAN Low
7	K-Line (ISO 9141-2)	15	L-Line (ISO 9141-2)
8	Manufacturer Specific	16	12V Battery Power

Out of these 16 pins, pin 6 and pin 14 are the most important. Pin 6 carries CAN High at 3.75 V and Pin 14 carries CAN Low at 1.25 V. This creates a potential difference of 2.5V which facilitates the communication. this differential voltage also makes CAN noise tolerant and minimizes the interference.

2.3.3 PIDs

PID stands for Parameter ID. PIDs are codes sent to the ECUs to request data from the vehicle. These IDs are predefined and each PID corresponds to a particular unit's parameter. For example, in our target vehicle the PID 115 corresponds to tachometer reading and the PID 254 corresponds to speedometer reading. To obtain the data from a unit, the external hardware sends a query PID to the network. The network sends the data as a response to the query.

2.4 Attack Surfaces in Automobiles

An attack surface can be defined as the collection of access points or attack vectors through which an intruder can gain access to the system. In the context of automobile, the attack surfaces are the points through which the attacker can gain access to the CAN network. Modern day automobiles have several functional units which directly or indirectly provide easy access to the network. Broadly they can be classified into three categories:

- Direct access to CAN Direct access is connection to the physical wires of CAN. They can be obtained by connecting to the OBD II port or through the TPMS (Tire Pressure Monitoring System) module or by breaking open the infotainment system or other similar units. OBD-II port is essentially an unprotected backdoor into a vehicles most sensitive embedded systems.
- Over the air CAN network connects to the external environment through various communication systems. Most of the modern cars are equipped with GPS, Bluetooth, Wi-Fi and Cellular network modules. These units also provide easy access to CAN network.
- ECU - Firmware ECUs in automobiles are like the CPUs of a computer. They contain the firmware which controls all the subsystems (break control, transmission control, timing control etc) in the automobile. The ECUs firmware can be compromised as well.

2.5 Vulnerabilities in CAN network

The CAN network was developed to be a fast and robust way of communication between different nodes. But it did not include any kind of security features. Due to the inherent flaws in the implementation of CAN bus, it is easy to break into the network. the following are major security flaws of the CAN network:

- **Lack of Segmentation and Boundary Defense:** While the CAN bus utilizes many different Electronic Control Units (ECUs) to oversee different vehicle functions, all sections of the bus are connected back to a common ECU at some point. Thus an intruder can gain access to the Anti-Breaking System through the infotainment system.

- **Lack of Device Authentication:** All data is available to all other vehicular components on the CAN bus once it is broadcast onto the bus, irrespective of whether they require that information or not. Other controllers on the CAN bus constantly listen for specific messages. However, the system does not have a mechanism to prevent unauthorized devices from joining the CAN bus and broadcasting messages out to any listening controllers.

- **Un-encrypted Traffic:** Encryption was never taken into consideration when CAN was being developed. The goal of CAN was to be fast and robust and implementation of encryption would bloat CAN messages and clog the bus.

- **Availability Check:** The CAN network checks for the availability of the carrier before every message is sent. If the carrier is busy, all the ECUs are halted for a while and the carrier is rechecked. This mechanism can be used by attackers to carryout Denial Of Service attacks by flooding the bus with High priority messages from critical units thus blocking the less critical ECUs from sending their messages.

CHAPTER 3. STATE OF THE ART

This chapter describes the existing research in the field of automobile security. The first section describes the exploits performed on CAN bus. There are two ways to address this security issue. The first method is through prevention of attacks and the second method is to identify the attack scenario and mitigate the damage. Second section describes the previously developed attack prevention techniques and the next section describes the attack detection and mitigation scenarios.

3.1 Automobile Attacks

Early research by Wolf et.al [4] identified the potential threats when the ECUs were being interfaced with systems like bluetooth, GSM, GPS modules to receive updates. They have identified the importance of automotive security. Zhao [5] identified the potential attack vector in an automobile system as the automotive technology rapidly advances towards V2V and V2I connectivity. Being a crucial part of this connected ecosystem, it is important to secure automobiles to facilitate safe and efficient travel. Though there were several publications in the academia [6] [7], the first experimental proof was demonstrates by Koscher et. al [8]. They demonstrated several attacks like unlocking doors, controlling the lights, horn, wipers etc. They achieved these attacks my fuzzing the and reprogramming the ECUs.

Hoppe and others [9] demonstrated attacks on airbag control, gateway ECU and electric window lift. A team of researchers from University of Washington and University of California, San Diego [10] experimented with a plethora of attack vectors including CD players, Bluetooth, radio etc. But the attack that caught the world's eye was the Jeep Cherokee attack at 2013 DEF CON [11] demonstrated By Chris Valesek and Charlie Miller. They have successfully taken over the vehicles control remotely by accessing the vehicle's steering, brakes and transmission through a vulnerability in the infotainment system through Sprint Cellular network. They could maneuver the car into

a ditch by disabling the brakes. They previously demonstrated several attacks by altering the firmware on Toyota Prius and Ford Escape.

The famous Chrysler hack was also due to a vulnerability in the UConnect system [12]. In 2016, a group of researchers from Keen Security labs [13] took over both the infotainment and instrument cluster screens and remotely unlock the doors of Tesla X. They were also able to open the trunk, fold a side mirror, and activate the brakes while the vehicle was in motion. Researchers were also able to remotely open the sunroof, move the power seats, and activate the signal lamps.

3.2 Attack Prevention Methods

Attack prevention methods essentially require the architecture to be more secure. Several authentication protocols were proposed on top of the existing CAN which will provide an additional layer of security.

- **CAN Auth:** Attack prevention methods essentially require the architecture to be more secure. Several authentication protocols were proposed on top of the existing CAN which will provide an additional layer of security. Herrewege et al. [14] proposed CAN Auth which requires a pre-shared key and the authentication data is transmitted out of band providing 15 bytes for authentication It successfully prevent replay attacks with the help of a counter value. This protocol is compatible with the existing CAN protocol.

- **LCAP:** LCAP stands for Light Weight CAN Authentication Protocol. This was proposed by Hazen and Fahmy [15] A magic number is generated using a one-way hash function. This magic number would be selected by the sender and only the intended receiver van verify it. This protocol reduced the communication and computational complexity that usually comes with cryptography. However, the predefined key and the session keys are exchanged between each sender and each receiver to initiate the protocol.

- **LiBrA CAN**: LiBrA CAN stands for Light weight Broadcast Authentication CAN protocol. This is similar to LCAP which exchange a pair of keys between the sender and receiver.

However instead of assigning a key to every sensor or ECU, they assigned keys by grouping theses devices. They proposed a master oriented authentication scheme, Multi-master oriented authentication scheme and a Distributed authentication scheme.

However, all the above protocols can not be implemented through over the air updates. They need to be implemented by physically accessing the ECUs. Except CANAuth which is backward compatible, these protocols cannot work hand in hand with the existing CAN protocol architecture.

3.3 Attack Detection and Mitigation

The Attack detection and Mitigation strategy requires replication of attacks and then applying a detection methodology which would detect anomalous behavior. The detection algorithm would either correct the attack or alert the driver that the vehicle is under attack. Some of the detection and mitigation strategies are:

- **Sensor based detection:** Muter et, al. [16] proposed anomaly detection using sensors. These sensors are designed based on the network protocol specifications, redundant data sources in the vehicles and the defined cooperative networking behavior of the devices. They do not produce false positives. Since they are based on unambiguous and reliable information.

  Limitation: There is a high possibility of false negatives in this case. The errors caused by hardware will also be classified under attack.Also, the source of anomaly cannot be detected (hardware error or malicious attack) using sensors. Moreover, the attacks caused by messages that are fully compliant to the networks normal behavior will be left undetected.

- **Entropy based detection:** Muter et, al. [17] also proposed an anomaly detection technique based on the entropy of the CAN messages. This technique relies on a parameter called as Coincidence. Coincidence factor is calculated by examining the relative entropy between two data sets. One data is collected when there is no attack. The second set is real time data. Generally, the legitimate data had lower coincidence factor value than under attack. When

the bus is flooded with a similar type of messages, the entropy decreases and the coincidence factor grows steadily. This steady growth is used to detect the anomalous behavior.

Limitation: It is hard to determine the threshold of the coincidence after which the scenario will be termed as an attack. The attacks where the attacked value has very small deviation from the legitimate value on the bus would go unnoticed. Since both the values would give the same value of coincidence factor.

- **Artificial Injection** This is another commonly used technique for detection. In this technique, the legitimate data is collected from the automobile. Then, instead of replicating the attack on the vehicle, the malicious data is inserted randomly into the legitimate data set and a detection algorithm is applied. This technique is simpler and doesn't have the hassle of finding the right hardware for attacks. Also, the automobile is not at risk.

  Limitation: Although this is a simpler method to inject data, this method is not reliable. Artificial injection doesn't account for the response and feedback from the automobile. During an injection attack, the frequency at which the malicious messages to be broadcast on the bus is of a great significance. The response also depends on the target vehicle. In reality, automobile can ignore malicious messages as stray messages if the frequency is too low. At the same time, the systems can be bricked if the the flooding frequency is too high. So these factors are crucial while deciding a mitigation strategy.

CHAPTER 4. EXPERIMENTAL SETUP

4.1 Introduction

This chapter describes the experimental setup required to perform injection attacks on the automobile. The hardware section describes various boards which can be used to interface with CAN network. The Software section describes the packages used for packet sniffing and injection on to the CAN network. This chapter also includes other boards and which can be used replicate the attacks.

4.2 Hardware

The attack surface selected in this project is the OBD-II port. The OBD-II port is usually located under the steering wheel of the automobile. It is requires to be within 2 feet (0.61 m) of the steering wheel. (Some exceptions might exist but it would still be in the reach of the driver). This port give direct physical access to the CAN network. We use a CAN to USB cable to connect to a computer.

4.2.1 Raspberry Pi 3

Raspberry is a fully functional computer built on a single circuit board. Raspberry Pi 3 features a 64 bit ARM Cortex A53 quad core processor and 1GB RAM. It runs Raspbian, a Debian Based Linux operating system. It also has on-board Wi-Fi, Bluetooth and USB boot capabilities. Raspberry Pi 3 by itself is insufficient to communicate with the CAN network. In combination with a CAN controller, it can communicate with the automobile network. We achieve the ability to communicate with CAN by interfacing Raspberry Pi 3 with PiCAN 2 board.

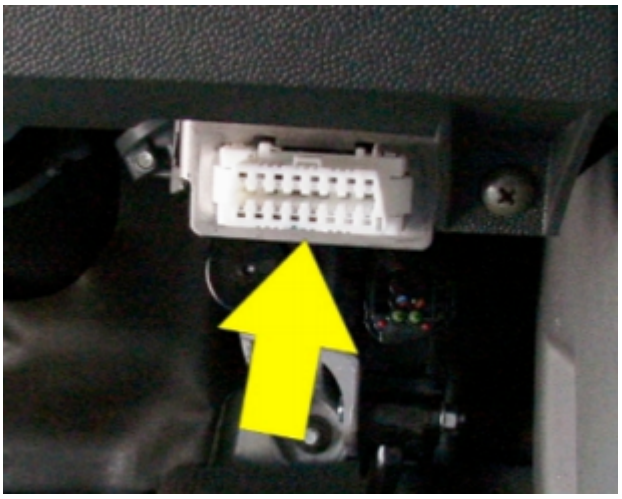

Figure 4.1 OBD port in 2005 Nissan Sentra

4.2.2 PiCAN2

PiCAN board provides CAN communication capabilities when mounted on the Raspberry Pi. It contains MCP2515 CAN controller in combination with MCP2551 CAN transceiver. The controller fulfills the communication functions prescribed by the CAN protocol. The transceiver connects the CAN controller to the physical transmission medium. In this case it connects to the CAN H and CAN L pins. PiCAN connects to the CAN network through a DB-9 connector. It can also be accessed through a 3 way screw terminal. It supports the Socket CAN driver which is discussed in Section 4.3.1.

4.2.3 Human Interface

A laptop is necessary to sniff the packets and to associate the PID (Parameter ID) with the corresponding ECU. A MacBook Pro was used in this experiment. It contains a 2.5GHz Core i5 Intel processor with 4 GB RAM and 500 GB storage. Any laptop with an Ethernet and USB ports is be sufficient. The laptop is used only for graphical interface to view the stream of packets. All the computation and processing are performed on the Raspberry Pi. Alternatively, a Linux system with Ubuntu OS or a Linux VM can be used with a CAN to USB interface like CANtact.

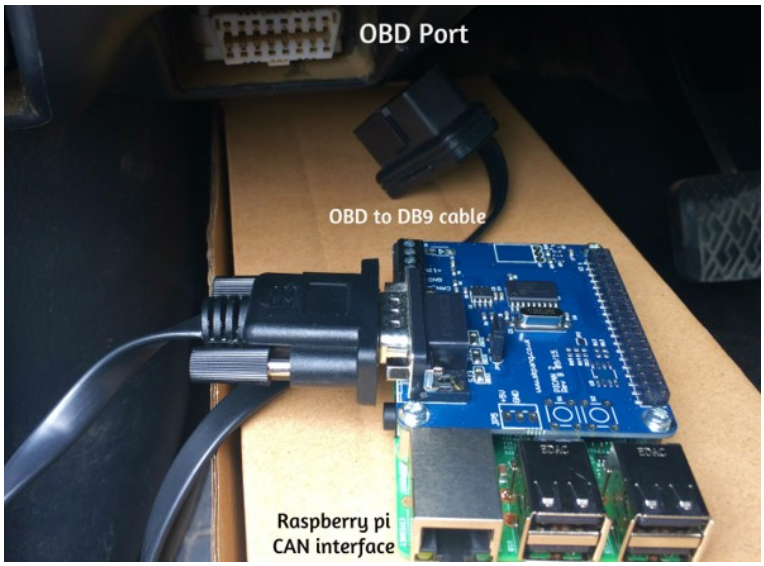

Figure 4.2 Raspberry PI, PiCAN board connection to OBD-II port

4.2.4 Cables

4.2.4.1 OBD to DB9

The PiCAN does not have an OBD port, instead it has a DB9 serial port. Therefore, an OBD to DB9 cable is required. This cable simply puts the CAN H signal (OBD-II pin 6) onto DB9 pin 3, and the CAN L signal (OBD-II pin 14) onto DB9 pin 5. This can be obtained from Spark fun.

4.2.4.2 USB Cable

The USB cable connects the Raspberry Pi to the laptop. This is primarily used to power the board from the laptop. We could use a power cable as well.

4.2.4.3 Ethernet Cable

The Ethernet cable connects between Raspberry Pi and the laptop. This facilitates the network between the Raspberry Pi and the laptop through VNC viewer.

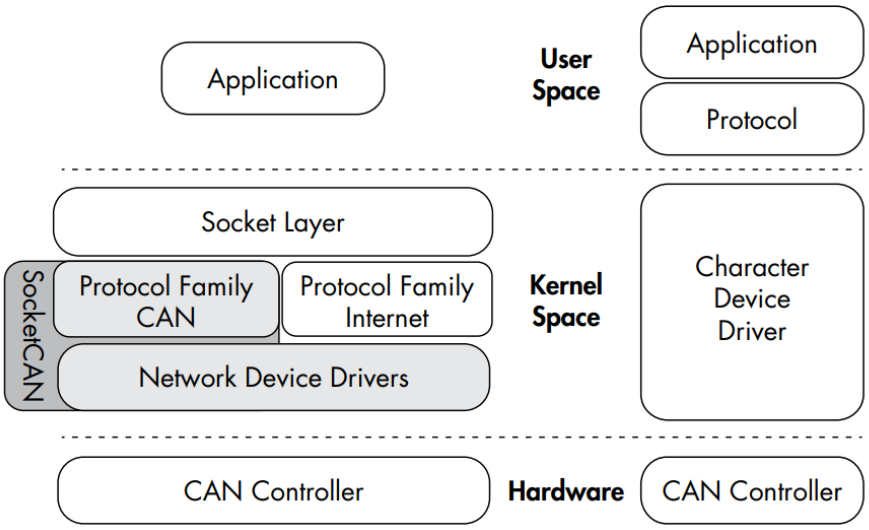

Figure 4.3 SocketCAN Architecture [2]

4.3 Software

4.3.1 SocketCAN

SocketCAN [18] is an open source Linux driver package specially designed by Volkswagen Research. It ties into the Linux networking stack. It supports built-in CAN chips, external USBs, Serial and Virtual CAN devices. SocketCAN applications can use standard C socket calls with a custom network protocol family, PF-CAN. This functionality allows the kernel to handle CAN device drivers and to interface with existing networking hardware to provide a common interface and user-space utilities. By creating its own CAN protocol family, SocketCAN can integrate with the existing network device drivers, thus enabling applications to treat a CAN bus interface as if its a generic network interface.

4.3.2 canutils

The can-utils package provides several applications and tools to interact with the CAN network devices, CAN-specific protocols, and the ability to set up a virtual CAN environment. These array

of tools and utilities can be used to sniff the CAN network, read and send packets onto the CAN network. It is available for free on Git-Hub [19].

- **candump**

  This command dumps all the CAN packets on the network to the console. The user can set flags in order to filter out the unwanted packets. It can also be used to log the packets into a file. This functionality is very useful in reverse engineering the CAN packets by replaying them from log files. The command is given along with the network name (can0).

  Usage: candump can0

- **canplayer**

  This command replays the CAN log files that are stored using the candump command. The .log file can be replayed by using the following command:

  Usage: canplayer -i speed.log *–I* is a flag which specifies the recorded log file (speed.log) to be played infinitely till the user stops.

- **cansniffer**

  This command displays the CAN packets and also highlights the bits which vary with time. This functionality can be used to identify repetitive packets and filter out the unwanted packets. Thus displaying the packets which change in real time.

  Usage: cansniffer can0

- **cansend**

  This command is sends a message on the CAN network. If used repeatedly, this command floods the bus with messages.

  Usage: cansend can0 115#4E0001FF0000FFFF In the above command, can0 is the name of the network, the digits before the pound sign (115) correspond to the PID and the digits after the pound sign correspond to the data.

4.4 Target Vehicle

he target vehicle is a 2017 model Ford Transit 500. We have tested several vehicles including Toyota Corolla, Chevrolet Sonic, Hyundai Veloster. But only Ford Transit delivered reliable results. The other vehicles gave erratic output. This was because the way each network is designed is manufacturer specific. The scope of this project is limited to only Ford Transit.

4.5 Other Methods

We experimented with several boards before finalizing the Raspberry Pi 3, PiCAN with Socket CAN. They are listed below:

4.5.1 Hardware

- ELM327 chipset is similar to MCP2515 and the cheapest available chipset. It has the ability to communicate with OBD over serial connections like USB, Bluetooth and Wi-Fi. However, it is primarily used for diagnostics and cannot interface with Socket CAN or CAN utils.
- Arduino also has the capability to communicate with CAN when a CAN interfaced with a CAN shield. Regardless of the shield, Arduino scripts have to be written in order to sniff/play the packets. It also requires another SD card shield to store the logs.
- Sparkfun OBD-II UART hosts STN1110 an OBD to UART interpreter. This board has both ELM327 as well as MCP 2515. It can be used to access both CAN and OBD protocols. The limitations of this device is that it needs an addition FTDI interface / an Arduino board to perform injection. Additionally, the pins need to be soldered.
- There are also several testers available in the market which are used by mechanics to determine the faults in the automobiles. The testers rely on Diagnostic trouble codes or DTCs. They are powerful in monitoring the traffic and analyzing the traffic. However their functionality is limited to sniffing the packets. The user cannot send packets on to the bus continuously.

4.5.2 Software

- Wireshark is a network sniffing tool. It can be run like can-utils on Linux with Socket CAN to log the data. However, Wiresharks decoder handles only the basic CAN header and doesnt know how to deal with ISO-TP or UDS packets.

- Kayak is a Java based GUI for analyzing CAN traffic. It has very advanced features like record and replay, GPS tracking etc. It needs a Linux interface and hence operates over SocketCAN.

- Caring Caribou is a security scanner for automobiles. It is still under development and very nascent. It can be used to brute force diagnostic services.

4.5.3 Simulator

- We also used ECUsim 2000, depicted in figure 4.4 which is an OBD-II simulator from Scan Tool OBD solutions. Although it cannot be used for injection attacks, it is useful in studying the working of an ECU. It can generate faults and MIL lights, and it also includes fault knobs for changing common vehicle parameters, such as speed.

4.6 Reverse Engineering CAN packets

4.6.1 Connecting to the CAN BUS

The Raspberry Pi connects to PiCAN circuit as shown in 4.2. This can be done by plugging the hardware into the OBD II port. Canutils can be cloned from GitHub and must be pre-installed on the Raspberry Pi. Once the hardware is in place, the network can be set by the following commands:

sudo ip link set can0 type can bitrate 500000

sudo ip link set up can0

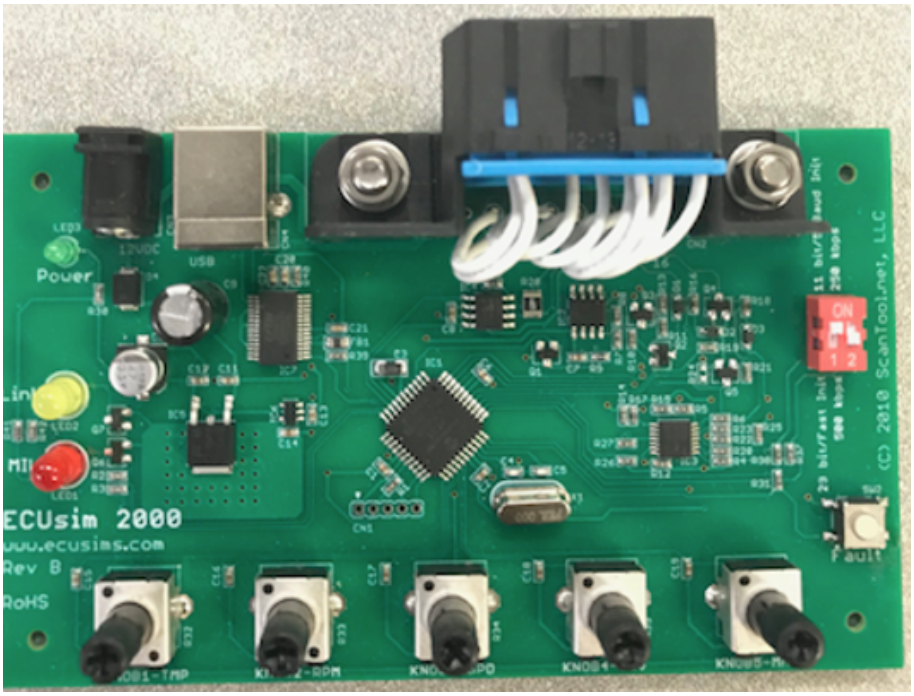

Figure 4.4 ECUsim 2000 simulator

4.6.2 Monitoring CAN bus and Data collection

The first step is to monitor the traffic and record the packets. A similar stream of packets like the following 4.5 can be observed on the console upon issuing candump command.

candump can0

4.6.3 Filtering CAN messages

CAN message filtering can be achieved using cansniffer command as discussed previously. Only the messages whose value changes in real time are displayed and all the static packets are filtered out. This can be efficiently used to identify the PID that needs to be flooded. Once there are a small number of active PIDs, the record-replay technique can be used. However, this technique has a disadvantage. The PIDs which act as counters/time-keepers will appear along with the PID corresponding to the action as well.

```
(1522004165.988068) can0 264#0003948C0C00FD83
(1522004165.988292) can0 04A#00D8D8D8FED8D8D8
(1522004165.988807) can0 010#0DFB00D4009380A1
(1522004165.992318) can0 050#8000880000000000
(1522004165.992561) can0 111#B000080000006000
(1522004165.992804) can0 113#0000020052160000
(1522004165.993044) can0 115#4E0002010000013E
(1522004165.993276) can0 117#F9358090805F8090
(1522004165.993593) can0 118#2208000060000000
(1522004165.993768) can0 119#4305000000000000
(1522004165.994017) can0 121#0000000040000000
(1522004165.994259) can0 122#7D7D3C00F4000000
(1522004165.994507) can0 123#0000300000000000
(1522004165.994753) can0 140#0000000080001227
(1522004165.994989) can0 229#003B801300001B26
(1522004165.995234) can0 25C#F860000002081B26
(1522004165.995472) can0 14A#6BFC28000005FFFF
(1522004165.995716) can0 170#801F80338033801F
(1522004165.995958) can0 211#00A0000520000407
(1522004165.996205) can0 253#01FF020C7DFA0000
(1522004165.996440) can0 254#918FD364239E0000
(1522004165.996687) can0 299#0600000000000000
(1522004165.996933) can0 301#B500000000000000
(1522004165.998447) can0 091#00007EF47F090001
```

Figure 4.5 Data stream output of candump command

The second method to reverse engineer the PIDs is using record and replay technique. For example, to find out the PID corresponding to the tachometer reading (Engine RPM), the candump command is given. Once the logging begins, the engine RPM is increased to a certain value say, 3000 rpm. Then the recorded log file is played back to check if the meter reading changes. Now the recording is divided into half and the first half is played. If the meter reading changes, the PID is in the first half. Else, it is present in the second half of the recording. This process of elimination is further continued till a single PID is achieved. The following flowchart 4.6 represents the steps required to reverse engineer the PIDs.

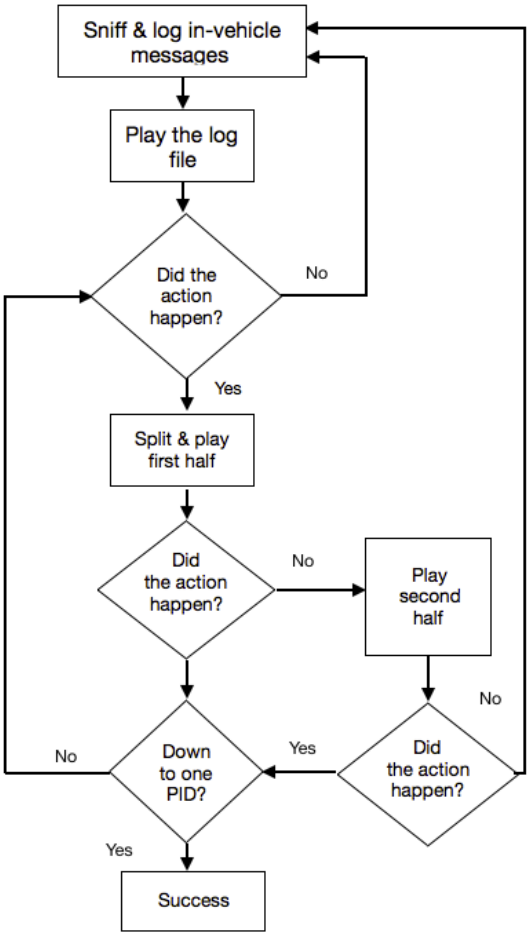

Figure 4.6 Finding PID using Record-Replay Technique

Using the above process of elimination, the PID for tachometer (RPM reading) is found to be 115 and speedometer reading is found to be 154.

4.7 Injection attack on CAN network

The goal of the experiment is not just monitoring the CAN network but also injecting packets onto the network. This can be achieved by performing similar steps performed to single down the PID. The attack part is simple and can be done via any one of the attack surfaces discussed earlier. The easiest and the most accessible is through the OBD port.

4.7.1 Modifying the files

The first step in injection attacks is to modify the log files. A sample line from the log file is given below. The first column is the time stamp in milliseconds, second column is the name of the CAN network (named as can0 in this case). The three digits before the pound sign (115 - RPM) represent the PID and the digits following the pound sign are the CAN message data bits. All of them are hexadecimal.

(1524351875.676969) can0 115#4E0001FF00000145

Out of the 16 digits, not all of them are responsible for the change in the RPM. The easiest way to determine the digits is by selecting two digits at a time and making them 00 (minimum) or FF (maximum) and sending them using cansend. In our case, the last four digits determined the value on the tachometer as well as the speedometer. The injected RPM value is 'FFFF' which in decimal translates to 16,384.

Eg: cansend can0 115#4E0001FF0000FFFF

This command needs to be sent continuously on to the network to observe a noticeable change in the meter reading (since the message rate is in the order of milliseconds). A Python script can loop cansend command at the required frequency. Alternatively, the log file can be altered and customized as well. The data packets can be altered simply by replacing all the packets which belong to a certain PID to a fixed value. This altered log file can be replayed using canplayer command. This will give the same result as looping cansend command.

When running canplayer, the -v option (verbose) displays each line of the .log file on the screen in real time, as it is being played back. The -I option is used to specify the filename of the input file. The CAN .log file can either be played back to the vehicle while the engine is running or while the engine is off (as long as the ignition is in the ON position). Regardless of whether the vehicle is parked or is in motion, it is still possible to take control of the CAN bus.

canplayer -I filename.log

The log file should be of the same format as the original. One interesting aspect of the log files is the timestamp. There is no mechanism in the automobile which checks if the time stamp is valid

or not. The time stamp represents the number of seconds elapsed since January 1, 1970 till now. However, the interval at which the time increments one line to next is important.

4.7.1.1 Data pre-processing

The data on the CAN bus is hexadecimal and is a mixture of data from all the units. So the first step is to filter RPM data out from the total CAN data and convert it into decimal value. As discussed earlier, only the last four digits in the data are responsible for the display on the meters.

- **RPM pre-processing**

  The RPM values when converted from hexadecimal to decimal vary between 0 to 65,535 (0 to FFFF). But the actual value of RPM varies from 0 to 16,383.75. The conversion can be calculated using the following formula:

$$rpm_{actual} = \frac{(rpm_{dec}) * 256}{1000} \tag{4.1}$$

  where rpm_{dec} is the decimal equivalent of the last four hexadecimal digits in the data field and rpm_{actual} is the actual tachometer value.

  The same procedure and formula applies to the speed values as well.

- **Speed pre-processing**

  The Speed values when converted from hexadecimal to decimal vary between 0 to 65,535 (0 to FFFF). The ECUs read the values in kilometers per hour (kmph) and the displays in the US show miles per hour (mph). So we multiply the decimal value with *0.62137119* to obtain the values in mph. Experiments showed that the resulting value's decimal point was off by two digits. Therefore, the obtained value is divided by 100 to calculate the approximate value of speed. The formula for calculation is shown below:

$$speed_{actual} = \frac{(speed_{dec}) * 0.62137119}{100} \tag{4.2}$$

where $speed_{dec}$ is the decimal equivalent of the last four hexadecimal digits in the data field and $speed_{actual}$ is the actual speedometer value.

CHAPTER 5. DETECTION OF ATTACKS USING DATA ANALYSIS TECHNIQUES

5.1 Introduction

This chapter will discuss the various data analysis techniques based on correlation applied on the CAN data to detect an attack scenario. The techniques are categorized into three sets of methods based on the approach. The first two methods are based correlation metric between the data points. The second technique is based on the Causality. The last two methods are based on Time-Series analysis of data.

5.2 Detection using Correlation

Correlation coefficients are used in statistics to measure how strong a relationship is between two variables. Pearson Correlation and K Means clustering were used in this category. Pearsons correlation (also called Pearsons R) is a correlation coefficient commonly used in linear regression. K Means is a clustering technique which groups data points based on the euclidean distance correlation.

5.2.1 Pearson Correlation

Pearson Correlation is a method to quantify how two variables are linearly related. The full name is the Pearson Product Moment Correlation (PPMC). It shows the linear relationship between two sets of data. Two letters are used to represent the Pearson correlation: Greek letter rho () for a population and the letter r for a sample.

$$\rho = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} \tag{5.1}$$

Correlation coefficient formulas are used to find how strong a relationship is between data. The value of the coefficient can be between -1 and +1.

- +1 indicates a strong positive relationship.
- -1 indicates a strong negative relationship.
- A result of zero indicates no relationship at all.

5.2.1.1 Approach

The speed and engine RPM are directly related. The engine RPM increases as we accelerate the vehicle. It decreases when we break or stop accelerating. So, engine RPM cannot increase on its own beyond a small threshold (when the vehicle is powered on). At the same time, the speed cannot be dropping if the engine RPM is increasing. So, the tachometer as well as the speedometer values increase when gas is applied. Therefore the speed and RPM values must show a strong positive correlation. But when data packets are injected to alter the values of the units, this relationship no longer exists. Therefore their correlation value under attack would be smaller than the previous case. The value of correlation will determine the attack scenario

- A correlation coefficient of 1 means that for every positive increase in one variable, there is a positive increase of a fixed proportion in the other.
- A correlation coefficient of -1 means that for every positive increase in one variable, there is a negative decrease of a fixed proportion in the other.
- Zero means that for every increase, there is no positive or negative increase. The variables are not related.
- The absolute value of the correlation coefficient gives us the relationship strength. The larger the number, the stronger the relationship. For example, 0.75, which has a stronger relationship than 0.65.

5.2.2 K-Means

K-means clustering is an algorithm of classification for unsupervised learning, for unlabelled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups of data points that are related in the data. The variable K represents the number of groups. Data points are clustered based on feature similarity. The results of the K-means clustering algorithm are:

- The centroids of the K clusters, which can be used to label new data
- Labels for the training data (each data point is assigned to a single cluster)

The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Rather than defining groups before looking at the data, clustering allows to find and analyze the groups that have formed organically. Each centroid of a cluster is a collection of feature values which define the resulting groups. Examining the centroid feature weights can be used to qualitatively interpret what kind of group each cluster represents.

5.2.2.1 Algorithm

The -means clustering algorithm uses iterative refinement to produce a final result. The algorithm inputs are the number of clusters and the data set. The data set is a collection of features for each data point. The algorithms starts with initial estimates for the centroids, which can either be randomly generated or randomly selected from the data set. The algorithm then iterates between two steps:

- Data assignment:

  In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance. Each centroid defines one of the clusters. More formally, if c_i is the collection of centroids in set C, then each data point x is assigned to a cluster based on

$$\arg\min_{c_i \in C} dist(c_i, x_i)^2 \tag{5.2}$$

where dist(.) is the standard L_2 Euclidean distance. Hence the new centroid is selected by minimizing the euclidean distance between a data point and its centroid. Let the set of data point assignments for each i^{th} cluster centroid be S_i.

- Centroid update:

  In this step, the centroids are updated. The new centroid is calculated by the mean of all data points assigned to that centroid's cluster.

$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i \tag{5.3}$$

  The algorithm iterates until no data points change clusters and the sum of the distances is minimized if run indefinitely. Or the maximum number of iterations can be assigned to stop it.

5.2.2.2 Parameter K

The algorithm described above finds the clusters and data set labels for a particular pre-chosen K. To find the number of clusters in the data, the user needs to run the K-means clustering algorithm for a range of K values and compare the results. In general, there is no method for determining exact value of K, but an accurate estimate can be obtained using the following techniques.

Monitoring the distribution of data points across groups provides insight into how the algorithm is splitting the data for each K. This method would be used in our approach. One of the metrics that is commonly used to compare results across different values of K is the mean distance between data points and their cluster centroid. Since increasing the number of clusters will always reduce the distance to data points, increasing K will always decrease this metric, to the extreme of reaching zero when K is the same as the number of data points. Thus, this metric cannot be used as the sole target. Instead, mean distance to the centroid as a function of K is plotted and the "elbow point," where the rate of decrease sharply shifts, can be used to roughly determine K. Various other

techniques exist for validating K, including cross-validation, information criteria, the information theoretic jump method etc.

5.2.2.3 Approach

The CAN data is unlabelled data. There is no difference between the data already on the bus and the data that is externally injected. There are many algorithms like nearest neighbours and support vector machines which can classify data into groups. But most of these algorithms need to be trained by manually labelling a part of the data as 'attack' or 'normal'. But, K - Means algorithm can classify unlabelled data. This method is called unsupervised learning.

Essentially, the data needs to be classified into two classes as either 'attack' or 'normal' . The algorithm is applied on tachometer and the speedometer data individually. Since the there are two classes, the value of the parameter k is set to be 2. This algorithm is guaranteed to converge to a result. The algorithm sometimes gives the local optimum centroid instead of the best possible outcome. Therefore, the algorithm is rum multiple times to obtain the best possible outcome.

5.3 Detection Using Causality

This section discusses the second category of Data Analysis techniques, the Causality techniques. Hidden Markov Model is the data analytics method selected for this analysis. In the previous section, the anomalous state was detected using the correlation between data points. Correlation technique cannot determine the source of the anomaly. Causality technique, in particular Hidden Markov Model can determine the source of the anomaly.

5.3.1 Hidden Markov model

Hidden Markov model(HMM) is a statistical method to model Time series data. It is based on the Markov property i.e, the probability distribution of future states will depend only upon the present state and not the events preceding it. In HMM, a sequence of emissions are observed but the sequence of states the model went through are unknown sequence of states. Each of these

hidden states will be associated with a set of observations. To create a HMM model, we generate two set of probabilities, Transition probabilities and Emission probabilities.

- **Transition probability** Transition probability controls how a new state, lets say S(t), is chosen from a current state S(t-1).
- **Emission probability** Emission probability is probability that a specific set of observations will be generated given current hidden state S(t).

During model generation we try to estimate these probabilities using the given data set. The different problems solved using HMM models include the following. Given the model, what is the probability of an observation sequence, what is the most likely sequence of hidden states corresponding to an observation sequence and the model that maximizes the probability of an observation.

5.3.1.1 Approach

We consider the change in vehicle's parameters as a series of states. For example, the vehicle is at rest and its speed is zero. Gradually, the speed uniformly increases to 25 miles per hour in 5 seconds. In this scenario, each 5mph increment per second is a state. Therefore, state S_1 is speed 0 mph, S_2 is speed 5 mph, S_3 is speed 10 mph, S_4 is 15 mph, S_5 is 20 mph and s_6 is 25 mph. Now, if the next state S_7 is 150 mph, probability of going to this new state is small. Hence this state can be termed unusual state. By comparing the probability of the new unusual state with the probability of previous state we detect anomalous behavior.

Using the above hypothesis as the foundation, our approach to detect injection attack using HMM has three main steps:

- Model Generation
- Training the model
- Anomaly Detection

5.3.1.2 Model Generation

We generate the model based on a sequence of observations. The observations are the RPM and speed values. We generate the models separately for RPM and speed. So each RPM or speed value is an observation. We used HMM tool box from MATLAB which has many built-in functions which help in generating the HMM model. We have used the following functions:

- **hmmestimate** This function estimates the transition and emission probabilities for a given set of observations. We used this function to generate transition and emission probabilities.
- **hmmviterbi** This function is based on Viterbi algorithm. It generates the most probable path or transition of states given a set of states and their emission and transition probabilities.
- **hmmtrain** This function calculates the maximum likelihood estimate for emission and transition probabilities.

We first calculated the transition and emission probabilities using *hmestimate*. If the data is too sparse, a speed or RPM gradient should be used instead of actual values to avoid false positives.

5.3.1.3 Training the model

The model can be trained using the function *hmmtrain*. This function takes observations, transition and emission probabilities as inputs. By default, the function uses Baum Welch algorithm to train the model. The algorithm can be changed to Viterbi algorithm as well. We have used Baum Welch algorithm.

5.3.1.4 Anomaly Detection

We use conditional probability, also called as posterior probability do detect anomalous states in the sequence. We use sliding window technique to detect anomalies. The sliding window moves with addition of each new state. The posterior probability of the sequence in the window is determined. This will result in a set of probabilities for each observation. These probabilities are compared to

a threshold value. When the posterior probability exceeds the threshold, an anomaly is detected. We used *hmmdecode* to perform anomaly detection.

5.4 Time Series Analysis

CAN data is time varying data. The data points are not random. Rather they are recorded at consecutive intervals of time. This kind of data is called as Time Series data. The data analysis techniques which are used to estimate the trends in a time-series dataset are called time-series analysis techniques. Time series analysis is predominantly used in financial institutions for forecasting purposes. This section will discuss the Moving averages method applied on the CAN data.

5.4.1 Simple moving average

Moving averages are calculated by creating windows or subsets in the complete data set. The window size is pre-fixed which will be decided based on a certain threshold. The average value of the data points in the subset is calculated and attributed to the first element in the data set. The window is shifted forward by an element and the average of the data points in that subset is calculated again and attributed to the second point. This process is continued till the end of the data set.

$$Y_{SM} = \frac{X_1 + X_2 + ... + X_{N-1} + X_N}{N} \tag{5.4}$$

where,

- Y_{SM} is the new data point
- X_1 to X_N are the values in the window
- N is the number of periods or the window size

The above formula gives the new or forecasted value using simple moving average. This method takes the unweighted mean of the data points.

5.4.2 Exponential Moving Average

Exponential moving average is a weighted moving average algorithm. It assigns exponential weights to the elements to compute the new value. Exponential moving average assigns higher weights to recent values than the past values. Hence it is more sensitive to the changes in the values.

$$S_t = \begin{cases} Y_1, & \text{t} = 1 \\ \alpha \cdot Y_t + (1 - \alpha) \cdot S_{t-1} & \text{t} > 1 \end{cases} \tag{5.5}$$

where,

- S_t represents the value of the exponential moving average at time t.
- α represents the smoothing factor. This will decide the extent to which the weights decrease over time.
- Y_t is the actual value at time t.

The smoothing factor is related to the window size 'w' as shown below:

$$\alpha = \frac{2}{W + 1} \tag{5.6}$$

5.4.2.1 Approach

The CAN data is a time series data. This data will consists of some irregularities. For example, the speed values cannot increment linearly. There could be minor peaks which are not due to injection. In this case, the model must not predict those peaks as attack values. At this point the moving average method can be used to detect attacks on automobile network. Moving average smoothens the curve. We can adjust the window size to match the required smoothness. This is the required threshold. An attack can be identified by looking at the values which are do not fall in the threshold range. They are represented by the spikes or sudden jumps in the values.

CHAPTER 6. RESULTS AND DISCUSSION

This chapter discusses results, the merits and demerits of each of data analytics techniques discussed in the previous three chapters and the impacts of the injection attacks on the host vehicle.

6.1 Data Analysis Techniques - Results

6.1.1 Correlation Techniques

6.1.1.1 Pearson Correlation - Results

6.1.1.2 Result

Initially, Pearson correlation was applied to the legitimate tachometer and the speedometer data to check whether they were related. The normal CAN data showed a high correlation coefficient value6.1 of 0.86. Which meant that the rpm values increased as the speed increased. The figure depicts a correlation matrix where var1 is the speed and var2 is RPM.

When the correlation was applied to the attack data, it showed zero value for speed as shown in Figure 6.3. Similarly, the value of correlation when the RPM was injected is -0.06 as shown in the Figure 6.2.

As discussed earlier, zero value means that the two variables are unrelated. The correlation value can be an initial parameter to determine if there is an attack. The results show that this could also be used to detect the correlation between the normal functioning and the attack scenario. Whenever the correlation value drops to a near zero value, it means that there is an anomaly in the data. Although we cannot determine the cause of the anomaly, we can detect the change in the behaviour.

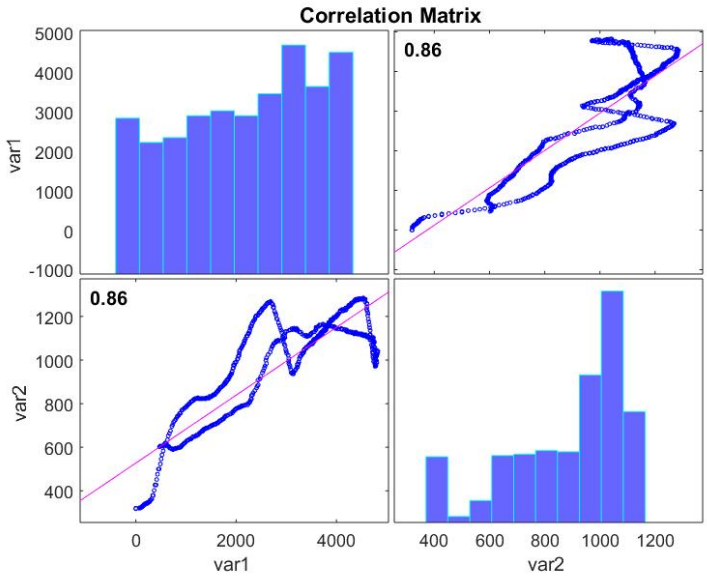

Figure 6.1 Speed RPM Correlation Matrix - Normal Scenario

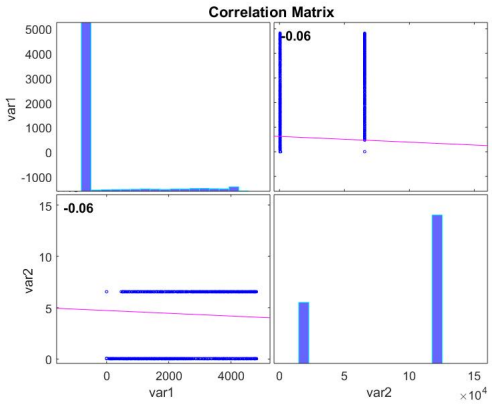

Figure 6.2 RPM attack scenario

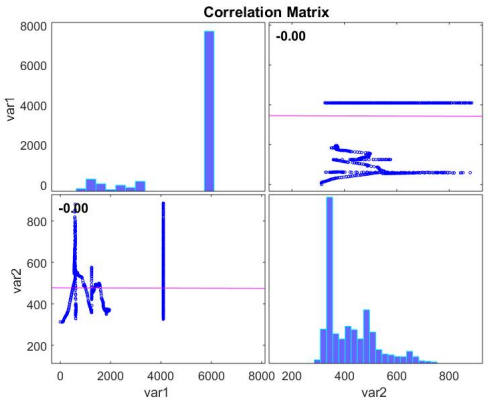

Figure 6.3 Speed attack scenario

The Pearson correlation coefficient is the most simple way to detect unusual behavior in speed and RPM data. This method can be used as the first line of several detection mechanisms in the vehicle. The accelerator is responsible for the change in the speed engine as well as RPM of the vehicle. At any given point of time, these two parameters must exhibit a strong correlation. Pearson correlation coefficient detects unusual behavior in the tachometer and speedometer readings. However, this method is only partially successful in detecting an attack scenario. The coefficient will show no correlation even if one of the units fail or malfunction. Therefore hardware malfunction cannot be differentiated from an attack.

6.1.1.3 K Means - Results

Figure 6.4 is the result of clustering the tachometer - RPM attack data. The X and Y axes represent the time and RPM values respectively. The blue line is the attack values and the red line represents the legitimate data on the bus. The grey points on both the lines are the cluster centers. The accuracy of classification on RPM tachometer data is 86.6%. It can be noticed that both the clusters are far apart and almost look like parallel lines. The injected RPM value is 'FFFF' which in decimal translates to 16,384. The legitimate values are around 1000 to 2000. Hence the variations in the RPM are not visible due the the large variation between attack and normal values.

Figure 6.5 is the result of clustering the speedometer attack data. The X and Y axes represent the time and speed values respectively. The red line is the attack values and the blue line represents the legitimate data on the bus. The grey points on both the lines are the cluster centers. The algorithm when applied on the speedometer data gave an average accuracy of about 87.9%. The injected value of speed was FFFF and the vehicle speed at the maximum point was around 20 mph. But we have observed during the experiment that any value above 4B70 (Hex equivalent of 120 mph, the max speed) is treated as 4B70. Hence, in comparison with RPM, the difference between attack value and normal values of speed is much lesser.

The accuracy of clustering can be increased by increasing the number of clusters. For example, at k = 100, the algorithm gave was 96% accurate. However, the number of clusters in this case

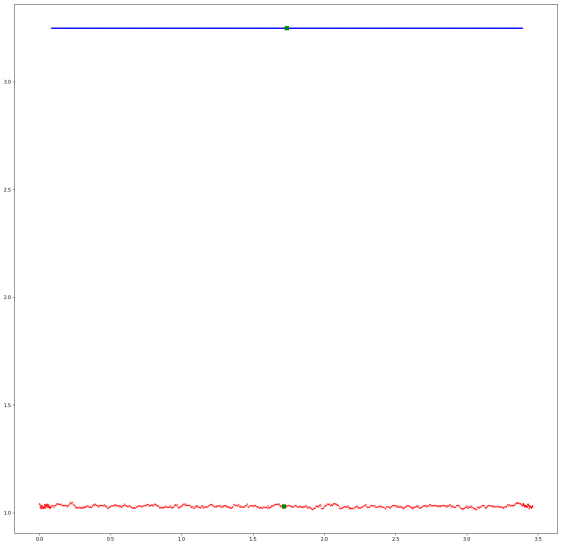

Figure 6.4 K-Means: RPM attack

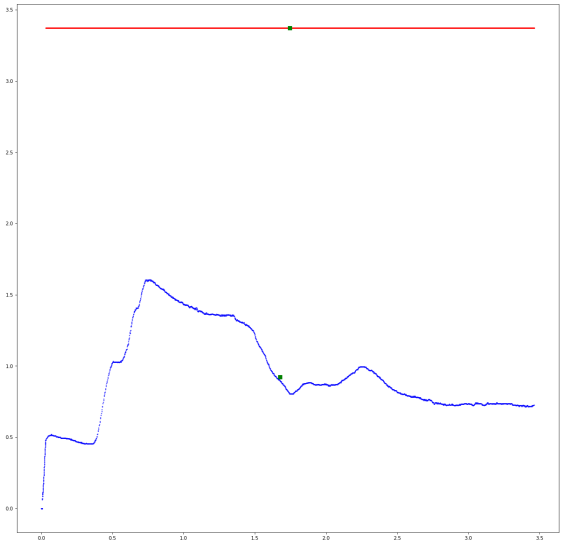

Figure 6.5 K-Means: Speed attack

is fixed at 2. Therefore, the data would over-fit if the number of clusters is increased. There are other methods like k-means++ which can be used to optimize the result.

Similar to Pearson Correlation, K Means can also detect anomalous behavior. We have achieved an accuracy of 87% on an average. The accuracy can be improved by implementing various optimization techniques. K means ++ will give better clustering accuracy. The results show that the model is successful in detecting a constant injected value. Further the experiment must be repeated several times with a range of random injected values to devise a reliable model. This scenario will be quite closer to an actual attack scenario.

6.1.2 Causality Techniques

6.1.2.1 Hidden Markov Model - Results

The Hidden Markov Model successfully predicts the anomalous states in the CAN data. We found five kinds of observations in the RPM and speed data. The observations are:

- Gradual increase

- Gradual decrease
- Constant value
- Sudden increase
- Sudden decrease

Out of the five cases, the first three cases were classified as normal behavior and the last two were abnormal behavior. Table 6.1 depicts the posterior probabilities. Table 6.2 depicts the posterior probabilities under tachometer attack. We can observe a profound difference between the normal and attack probabilities. These extreme values of probabilities (0 and 1) are used to differentiate attack from normal behavior. The extreme values are caused by the sudden increase or sudden decrease in the RPM value. The model could differentiate between a gradual increase caused by natural driving conditions and the sudden increase caused by the attack.

Table 6.1 Sample Posterior probabilities for normal functioning

0.38493275	0.243745946	0.260995802	0.340430998	0.309478055	0.309868661
0.409115024	0.525114533	0.480015556	0.490378124	0.51006003	0.500813405
0.199940304	0.229978461	0.25750628	0.166326769	0.177539471	0.186297938

Table 6.2 Sample Posterior probabilities for RPM attack

0	0.04462013	**0**	**0**	**0**	0.102719277
0.174251212	0.06227673	**1**	**1**	0.255753217	0.083630311
0	0.069276806	**0**	**0**	**0**	0.076115937

The HMM is generated by examining the probabilities of transitions between a series of states. The conditional probabilities of state transition helps differentiate between normal data and anomalous data. This model calculates the likelihood of speed or RPM to drop or increase drastically based on previous outcomes.

In this project, HMM was applied separately to speed and RPM sensors. It was successful in detecting the attack values. Further, this model can be applied to multiple observations simultaneously. For example, speed is correlated to RPM. Both of them could be under attack. In that

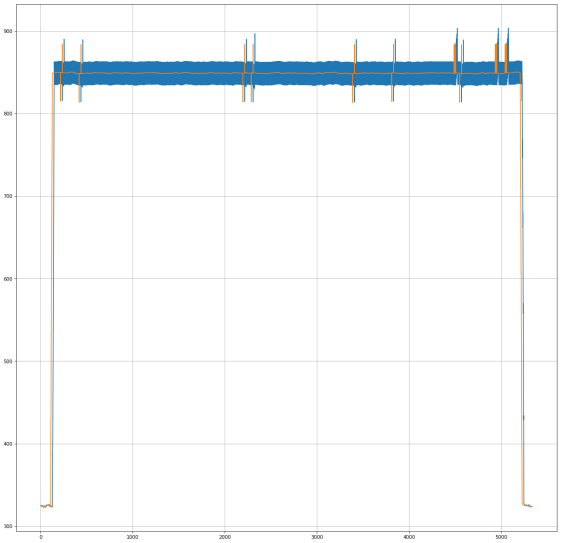

Figure 6.6 Time-series: RPM attack

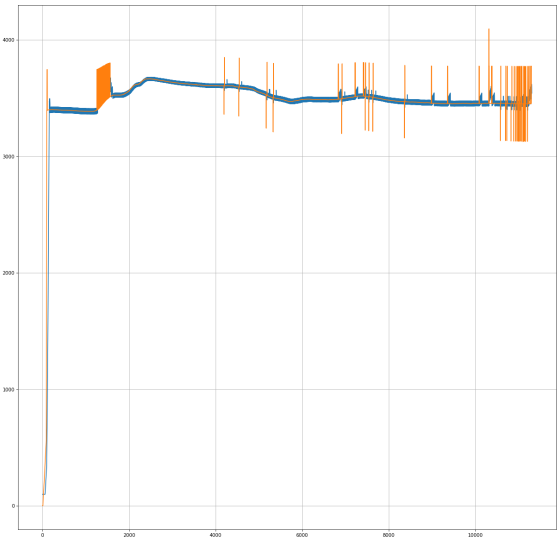

Figure 6.7 Time-series: Speed attack

scenario, both of them will have erratic peaks and falls. Along with these anomalous states, they will not have a positive correlation. HMM can be implemented on these multiple observations as well to detect injection attacks.

6.1.3 Time series Analysis

6.1.3.1 Moving Averages - Results

Figure 6.6 is the output of time series analysis on RPM attack data and figure 6.7 is the output from speed attack data. The X and Y axes represent the time and average value respectively. The orange band is the rolling mean or unweighted moving average and the blue band represents the exponential moving average trend. The window size is 20 time units for both. The value of α is calculated from the window size using the equation 7.3.

The peaks correspond to the injected value in both the cases. They correctly depict the trend in which the packets are injected. The packets were uniformly injected during the RPM attack. But in the case of speed injection, we waited till the car achieved a minimum speed. Hence there is a cluster of spikes after a small time period.

Also, a slight lag is observed between the peaks of exponential moving average and simple moving average. This lag in simple moving average could be due to the absence of weights. The exponential moving average assigns higher weights to the recent values thus reducing the lag factor. Overall, this method is successful in detecting the trends in which the packets were injected.

The results of time series analysis gives us the trends in the change of a parameter value. While driving a vehicle, the speed or RPM are never constant. They change gradually in a continuous fashion. But in an attack scenario, the speed or rpm change abruptly in discrete intervals. This method smoothens the gradual changes. Hence time series analysis helps us to differentiate between small gradients in speed and RPM and an attack scenario. Also, the computational complexity of this method is low and it can be used as a technique to alert the driver of the vehicle in case of an attack. But there might be a lag in predictions, which needs to be improved by adjusting the weighting factors.

6.2 Impacts of the Results

The goal of this project is to detect injection attacks on in-vehicle network. We tried to solve the problem of detecting the abnormal states into a data analytics problem. All these data analysis techniques are compatible with older and newer vehicles unlike the other detection methods. These techniques not only safeguard the vehicle from injection attacks, but also can be employed to analyze the driving patterns of the driver and assist the driver.

The data analytics methods demonstrate that all these algorithms are successful in detecting anomalous behavior at various levels of accuracy. There is always scope of improvement in terms of the accuracy of detection. These techniques can be implemented in combination to achieve better accuracy and reliability. Our approach is not limited to the techniques discussed here. It can be extended to various other data analysis techniques like Hidden Markov and Support Vector Machines or Hidden Markov PCA etc.

The absence of an anomaly detection mechanisms in the current in-vehicle network poses a threat to passenger safety. It also majorly impacts the automotive industry in general. Various

cars were hacked which demonstrated the severity of these security flaws. In such cases, the manufacturers not only damage their brand image but also have to pay hefty amounts in terms of collateral damage and legal formalities. The current technology does not support easy over-the-air updates. The Chrysler attack [12] prompted Fiat Chrysler to send USB sticks with security patches to 1.4 million car owners. Yet, the manufacturer cannot ensure that the patch has been installed on every Fiat Chrysler.

The absence of these security features has a major impact on autonomous vehicles. Recently, [13] a team hacked Tesla model X and Tesla model S. They disabled the breaks by accessing the CAN network wirelessly through the infotainment system. Human drivers have the ability to mitigate the damage by manually overriding in case of an attack. But the current autonomous vehicles cannot act on the fly. Although the manufacturers regularly release security updates, the security features need to keep up with with the advancing technology.

CHAPTER 7. CONCLUSION AND FUTURE WORK

This thesis presented the problem of injection attacks on automotive networks. We first looked at the various attacks performed by researchers. We also studied the automobile architecture, CAN bus architecture. Most importantly, we studied the loopholes in the CAN bus architecture which provides room for attacks. We demonstrated injection attacks on our target vehicle (Ford Transit) by injecting malicious speed and RPM data packets. The injection attack could manipulate the tachometer and speedometer readings even when the vehicle is in motion. This attack demonstrates how easily an attacker can access the CAN data. The anomaly detection approach based on several data analysis techniques was tested and the experimental results demonstrate the effectiveness of the approach. The techniques are simple to implement and should be incorporated into the existing intrusion detection systems easily.

The scope of this thesis limited our attack surfaces to OBD-II port. Similar injection attacks can be replicated through various other attack surfaces like Bluetooth, Infotainment systems, Wi-fi, Cellular networks, and even Key fobs. The attacks can also be targeted on several other critical systems like gas pedal, breaks, automatic cruise control and gear rods. It will also be interesting to replicate these attacks on autonomous vehicles to investigate the effectiveness of their security features. This project considered a single attack happening at any given point of time. But it is possible to launch attacks on multiple safety critical systems concurrently. To prototype a system which can detect injection attacks in real time, the data analytics should be optimized to detect multiple simultaneous attacks. It is also important to minimize the computational complexity. Although it is impossible to track and detect every security threat an automobile faces, the security features must keep up with the advancements in automotive technology.

BIBLIOGRAPHY

[1] Othmane, L.B., Fernando, R., Ranchal, R., Bhargava, B.: Likelihood of threats to connected vehicles

[2] Smith, C.: The Car Hacker's Handbook: A Guide for the Penetration Tester. 1st edn. No Starch Press, San Francisco, CA, USA (2016)

[3] Nouvel, F., Gouret, W., Mazrio, P., Zein, G.: Automotive network architecture for ecus communications (04 2009)

[4] Marko, W., Weimerskirch, A., Thomas, W.: State of the art: Embedding security in vehicles. (06 2007)

[5] Zhao, Y.: Telematics: safe and fun driving. IEEE Intelligent Systems **17**(1) (Jan 2002) 10–14

[6] Abbott-McCune, S., Shay, L.A.: Intrusion prevention system of automotive network can bus. In: 2016 IEEE International Carnahan Conference on Security Technology (ICCST). (Oct 2016) 1–8

[7] Zhang, Y., Ge, B., Li, X., Shi, B., Li, B.: Controlling a car through obd injection. In: 2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud). (June 2016) 26–29

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

[9] Hoppe, T., Kiltz, S., Dittmann, J.: Security threats to automotive can networks – practical examples and selected short-term countermeasures. In Harrison, M.D., Sujan, M.A., eds.:

Computer Safety, Reliability, and Security, Berlin, Heidelberg, Springer Berlin Heidelberg (2008) 235–248

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

[11] Miller, C., Valasek, C.: Adventures in automotive networks and control units (2013)

[12] Brandom, R.: The scariest thing about the chrysler hack is how hard it was to patch (Jul 2015)

[13] Golson, J.: Car hackers demonstrate wireless attack on tesla model s (Sep 2016)

[14] Groza, B., Murvay, S., van Herrewege, A., Verbauwhede, I.: Libra-can: A lightweight broadcast authentication protocol for controller area networks. In Pieprzyk, J., Sadeghi, A.R., Manulis, M., eds.: Cryptology and Network Security, Berlin, Heidelberg, Springer Berlin Heidelberg (2012) 185–200

[15] Hazem, A., Fahmy, H.A. In: LCAP - A Lightweight CAN Authentication Protocol for Securing In-Vehicle Networks. (2012)

[16] Mter, M., Groll, A., Freiling, F.C.: A structured approach to anomaly detection for in-vehicle networks. In: 2010 Sixth International Conference on Information Assurance and Security. (Aug 2010) 92–98

[17] Mter, M., Asaj, N.: Entropy-based anomaly detection for in-vehicle networks. In: 2011 IEEE Intelligent Vehicles Symposium (IV). (June 2011) 1110–1115

[18] SocketCAN

[19] linux can, G.: linux-can/can-utils (Apr 2018)