Using messages precedence similarity to detect message injection in in-vehicle network

Mubark Jedh

Follow this and additional works at: https://lib.dr.iastate.edu/creativecomponents

Part of the Data Science Commons, and the Information Security Commons

Recommended Citation
https://lib.dr.iastate.edu/creativecomponents/651

This Creative Component is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Creative Components by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Using Messages Precedence Similarity to Detect Message Injection in In-vehicle Network

by

Mubark Jedh

A creative component submitted to the graduate faculty in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

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

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 dissertation/thesis. The Graduate College will ensure this dissertation/thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University
Ames, Iowa
2020
TABLE OF CONTENTS

LIST OF TABLES ................................................................. v
LIST OF FIGURES ............................................................... vi
ACKNOWLEDGMENTS ........................................................... viii
ABSTRACT ................................................................. ix

CHAPTER 1. INTRODUCTION .................................................. 1
  1.1 Car Network ......................................................... 1
  1.2 Problem ............................................................... 2
  1.3 Approach .............................................................. 2
  1.4 Threat Model .......................................................... 3
  1.5 Organization .......................................................... 3

CHAPTER 2. BACKGROUND .................................................. 4
  2.1 Electronic Control Units (ECUs) ................................. 4
     2.1.1 History .......................................................... 4
     2.1.2 Electronics Control Units ...................................... 4
  2.2 Control Area Network (CAN) Bus ............................... 6
     2.2.1 History .......................................................... 6
     2.2.2 CAN Bus Network ............................................... 6
     2.2.3 CAN Bus data frame .............................................. 7
     2.2.4 CAN Bus Protocol ............................................... 8
  2.3 CAN Bus Vulnerabilities ............................................ 8
  2.4 CAN Bus Attack Surface ............................................. 10
### 2.4.1 Direct Access

- Page 10

### 2.4.2 Indirect Access

- Page 10

### 3.1 Vulnerabilities and attack surfaces

- Page 12

### 3.2 Mitigation and Detection

- Page 13

### 4.1 Introduction

- Page 16

### 4.2 Used Data

- Page 16

### 4.3 Data Preparation

- Page 16

### 4.4 Data Preprocessing

- Page 17
  - 4.4.1 Data Normalisation
    - Page 18
  - 4.4.2 Training / Testing split
    - Page 19

### 5.1 Introduction

- Page 20

### 5.2 Similarity metrics

- Page 20
  - 5.2.1 Cosine Similarity
    - Page 20
  - 5.2.2 Levene Test
    - Page 20
  - 5.2.3 T-test
    - Page 21
  - 5.2.4 Approach
    - Page 21

### 5.3 Point Change Detection

- Page 22
  - 5.3.1 Approach
    - Page 23
  - 5.3.2 Closeness centrality
    - Page 23
  - 5.3.3 PyMC3 Bayesian Point Change Detection
    - Page 24
  - 5.3.4 Bayesian Inference and approach
    - Page 25

### 6.1 Introduction

- Page 27
6.2 Overview of Machine Learning ............................................. 27
6.3 Approach ................................................................. 30
  6.3.1 Building the Model .................................................. 31
  6.3.2 Keras Model .......................................................... 31
  6.3.3 Training the Model .................................................. 32
CHAPTER 7. Result and Discussion .......................................... 34
  7.1 Similarity Metrics Output ............................................ 34
  7.2 Evaluation of LSTM RNN ............................................. 36
  7.3 Speed Data Set ......................................................... 37
    7.3.1 Training With Different window sizes ......................... 39
  7.4 RPM data set ........................................................... 41
    7.4.1 Training with different window sizes ......................... 42
  7.5 Point Change Detection ............................................ 44
    7.5.1 PyMC3 Point Change ............................................ 45
CHAPTER 8. Conclusion and Future Work .................................. 48
LIST OF TABLES

<table>
<thead>
<tr>
<th>Page</th>
<th>Table Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>List of data set used in the research [1]</td>
</tr>
<tr>
<td>7.1</td>
<td>LSTM RNN model parameters</td>
</tr>
<tr>
<td>7.2</td>
<td>LSTM RNN accuracy prediction at different window size for Speed data set</td>
</tr>
<tr>
<td>7.3</td>
<td>LSTM RNN accuracy prediction at different window size for RPM data set</td>
</tr>
<tr>
<td>7.4</td>
<td>PyMC3 parameters output for PiId 115</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Electronic Control Units image adopted from [2]</td>
</tr>
<tr>
<td>2.2</td>
<td>Complexity of wiring ECUs with and without the CAN Bus protocol, image adopted from [1]</td>
</tr>
<tr>
<td>2.3</td>
<td>A simplified CAN Bus network connection</td>
</tr>
<tr>
<td>2.4</td>
<td>Structure of CAN bus standard 11-bits frame</td>
</tr>
<tr>
<td>4.1</td>
<td>Attack vector for speed, RPM, and normal Data</td>
</tr>
<tr>
<td>4.2</td>
<td>Data preprocessing steps using Normalisation</td>
</tr>
<tr>
<td>5.1</td>
<td>Cosine Similarity and attack vector for speed, RPM, and normal Data</td>
</tr>
<tr>
<td>5.2</td>
<td>A circle graph representation of RPM PIds. The number of connections represented the attributes between each PIds.</td>
</tr>
<tr>
<td>5.3</td>
<td>PyMC3 model architecture</td>
</tr>
<tr>
<td>6.1</td>
<td>Hierarchy and relationships between Artificial Intelligence, Machine Learning, and Deep Learning</td>
</tr>
<tr>
<td>6.2</td>
<td>Overview of simple Artificial Neural Network (ANN)</td>
</tr>
<tr>
<td>6.3</td>
<td>RNN processing sequential data, figure adopted from [3]</td>
</tr>
<tr>
<td>6.4</td>
<td>LSTM RNN architecture, image adopted from [4]</td>
</tr>
<tr>
<td>6.5</td>
<td>LSTM RNN model architecture used to detect an attack in-vehicle communication network.</td>
</tr>
<tr>
<td>7.1</td>
<td>Developed messages precedence graph for cosine similarity on Speed, RPM, and normal data set</td>
</tr>
</tbody>
</table>
7.2 Developed messages precedence graph for levene test on Speed, RPM, and normal data set ........................................ 36
7.3 Developed messages precedence graph for t-test on Speed, RPM, and normal data set ........................................ 37
7.4 The Epochs Accuracy for sequential LSTM RNN ......................... 38
7.5 LSTM RNN output prediction for Cosine Similarity, Levene Test, and T-Test test data set ........................................ 39
7.6 LSTM RNN prediction on Cosine Similarity, Leneve Test, and T-test vs Window Size for speed Data Set ......................... 40
7.7 LSTM RNN output prediction for Cosine Similarity, Levene Test, and T-Test for RPM data set ........................................ 42
7.8 LSTM RNN prediction on Cosine Similarity, Leneve Test, and T-test vs Window Size for RPM Data Set ......................... 44
7.9 Measured centrality value vs time in seconds ............................. 44
7.10 PyMC3 Trace plot For PId 115 ........................................... 46
7.11 Point change detection probability output ............................... 47
ACKNOWLEDGMENTS

I would like to express my gratitude to my research advisor Dr. Lofti Ben Othmane for the useful comments, aspiring guidance, and present help. Without his guidance and supervision, this thesis would not have been possible. Also, I would like to thank Dr. Dough Jacobson for his contribution and for serving as my committee member. I am sincerely grateful to them for sharing their truthful and illuminating views on the thesis manuscript.

Getting through my thesis required more than academic support. I want to express my gratitude and appreciation to Thelma Harding for her continues support and encouragement. Also, the GMAP for their financial support.

Most importantly, this could have happened without my family. I would like to thank my parents and my brother for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them. Thank you.
We developed an intrusion detection technique that detects an attack on CAN Bus network using Long Short-Term Memory Recurrent Neural Network (LSTM RNN). Modern cars contain a number of Electronic Control Units (ECUs) that are responsible for normal car functionality and safety. These ECUs communicate with each other by an in-vehicle bus communication network such as CAN Bus. The CAN Bus lacks encryption and authentication mechanism to ensure secure communication. The safety of the driver relies on the ECUs exchanging messages on the CAN Bus securely. We proposed an intrusion detection system (IDS) that secure in-vehicle communication network. We develop a technique that counts the number of precedence messages, then we use a similarity matrices to detect injection of messages into the CAN Bus. First, We develop message precedences similarity graphs that distinguish between an attack state and no attack state. Then, we constructed an LSTM RNN for anomalous detection. Then, we use similarity matrices graph to training the LSTM RNN at different size and time window. Also, we develop a method to detect changes in PIDs centrality due to the injection of messages on CAN Bus. The method was not successful in detecting attacks in-vehicle communication network.
CHAPTER 1. INTRODUCTION

Cars of the past didn’t come with any advance safety features we see today. Computerization revolutionized car safety e.g. auto collision avoidance, anti-lock brakes, traction control, and electron stability control. As a result, cars become a rolling computer. Modern cars bear a little resemblance to the cars of the past. The trending of powerful computing platforms increasingly growing with more automated safety features and self-driving options. Within the past 20 years, cars have evolved into a complex computer and network as of many as 80 Electronic Control Units (ECUs) networked together with up to 100MB of binary code [5]. These ECUs are connected together through in-vehicle communication network such as Control Area Network (CAN) Bus.

1.1 Car Network

Each car uses set of ECUs which are responsible for its operation and safety. These ECUs communicate through two types of network:

1. A fast and reliable for critical data communication (e.g., engine messages, transmission messages, and speed). [5]

2. A slow for less critical data communication (e.g., radio, Bluetooth, and Wi-Fi) [5]

The CAN Bus network that is used in modern cars has security vulnerabilities, which can be abused by an adversary to take control of the car. Vulnerabilities found on the network are within the protocol design. Therefore, distortion of the communication can lead to a severe accidents. The CAN Bus architecture is designed to be stable and flexible but lacks robust security; it allows an attacker to access critical features of the car through comprising ECUs. Increasing the number of ECUs for car safety and operation increase attack surface on the CAN Bus network and potential for a plethora of vulnerabilities.
1.2 Problem

The CAN Bus is reliable communication and error detection capabilities have enhanced car safety and functionality. Unfortunately, designed more than 30 years ago, the car manufactures failed to implement even the most basic security mechanism such as [1]:

- No authentication built into CAN messages to ensure the message the validity of the source.
- Packets are broadcasted to all components on the bus, any component has access to bus can read the message.
- No encryption mechanisms to ensure message confidentiality.
- No mechanism for detecting of message injection attacks.

The thesis addresses the question: Can we detect an injection attack in vehicle network without knowledge in the systemantics of data exchanged in the CAN BUS?

1.3 Approach

The goal of the research is to develop a technique for detecting injection messages into CAN Bus network without knowledge in the systemantics of data exchange in the CAN Bus network. Data used in this research was collected by Othmane et. al using OBD-II physical access [1]. The injection of messages was performed on tachometer (RPM) and speedometer (Speed) [1]. Three data were collected by a colleague, Speed, RPM, and normal data [1]. The Speed and RPM data were compared with the normal data to distinguish between an attack state and no attack state.

We used three methods for development of messages precedence graph and detection of messages injection using similarity metrics:

- Cosine Similarity
- Levene Test
• T Test

Long Short-Term Memory Recurrent Neural network (LSTM RNN), was used to distinguish between two state: normal driving behavior scenario and an attack scenario. LSTM RNN was training using 60% of the data set and 40% used for the testing the accuracy of the model. Bayesian point change detection was used to detect an injection of messages into a tachometer (RPM) and speedometer (Speed) based on centrality of ECUs messages ID.

1.4 Threat Model

We consider threat model to be an attacker that can exploit the vulnerabilities and access CAN Bus network to perform a malicious act. An attacker can access the CAN network physically through the access OBD-II, USB, and wirelessly through the interception of a radio frequency signal or on-broad Wi-Fi. If an attacker has physical access into the car through the OBD-II, then s/he can take over all the functionality of the vehicle including disabling safety features, shutting down the engine, disables electronic stability control, etc.

1.5 Organization

The remainder of this thesis organizes as follows. Chapter 2 provides background information on The Electronic Control Units (ECU) and CAN Bus. Chapter 3 discusses literature publication on CAN Bus vulnerabilities, mitigation, and detection mechanism. Chapter 4 describes data preparation step. Chapter 5 describes detection of messages using similarities matrices and Bayesian point change. Chapter 6 describes machine learning neural network and how it is applied to distinguish between normal driving behavior and an attack scenario. Chapter 7 discusses the results and limitations. Chapter 8 is conclusion and future work.
CHAPTER 2. BACKGROUND

This chapter provides background information on ECUs, CAN Bus vulnerabilities and Attack surfaces. Section 2.1 describes overview of Electronic Control Units (ECUs). Section 2.2 describes the CAN Bus architecture and data frame. Section 2.3 describes the vulnerabilities in the CAN Bus. Section 2.4 describes attack surfaces of a connected vehicle.

2.1 Electronic Control Units (ECUs)

2.1.1 History

Prior to the use of ECUs, components of the car engine, steering, brakes, and transmission work on the principles of mechanics. The mechanical components had inherent limitations and flaws. On 1970, the Environmental Protection Agency issued standards for vehicle emissions that all cars sold in the United States are required to meet. To comply with law car manufactures began utilizing ECUs to control vehicle functionality and emission. Unlike older cars, modern automobiles rely on ECUs for normal car operations. The first Electronic Control Unit (ECU) produced is the Engine Control Unit (ECU), which controls the engine functionality (ignition timing, fuel injection, and air flow ratio), producing an optimal performance of the engine and less emission. Later on, the control units began to be used on other parts of the cars e.g. transmission, brakes, change of speed, power steering, and gear shift.

2.1.2 Electronics Control Units

Car manufacturers incorporate safety features into the vehicles using ECUs. To coordinate those features, the automobile makers introduced the CAN Bus network. The modern car may have as of many as 100 ECUs networked together with up to 100MB of binary code [1]. The ECUs receive, process, and output data from and to other ECUs. The design and implementation of
ECUs vary across car manufactures, but their communication is standard since 2008. A typical car ECU comprises of hardware and software. The hardware components consist of an electronic circuit with embedded microcontrollers [6]. The functionality of ECU is achieved by software stored in the chip microcontroller. Also, a microprocessor for processing data coming from sensors or other ECUs. The microprocessor process input data from sensors performing complex calculation and lookup table by the ECU software to output data that adjust car functionality e.g. output data can be determining the ignition time, cylinder firing time, cylinder deactivation, and fuel injection. Furthermore, ECUs send messages to and from other ECUs to perform a task, together ECUs form the car network as shown in figure 2.1. Examples of ECUs found in cars including but limited to:

- Body control units (e.g., controls heated seat, interior, and exterior lighting, ... etc)
- Passenger door units (e.g., controls power door locks, power windows...etc)
- Airbags control units
- Electronic vehicle display system
- Brake control units
- Instrument clusters units (gauge indicator for speed and fuel)
- Adjustable pedal units
- Electric power steering units

An attacker can send messages to the ECUs to perform a malicious task. They can update or delete the software on the ECUs. ECUs are great devices for car functionality, but their communication inherently insecure and vulnerable to attacks.
2.2 Control Area Network (CAN) Bus

2.2.1 History

In older cars, ECUs are hard wired as depicted by figure 2.2. As more modules being added into the car hard wiring becomes complex, expensive, and add more weights into the car. To optimize ECUs wiring, Robert Bosch introduced the CAN Bus in 1986 [7]. As a result, car manufactures began quickly to adopt the CAN Bus into newer cars and becomes an international standard in 1993 known as ISO 11898. In 2008, CAN Bus protocol was mandated to be included in every car sold in the United States and 2001 for cars sold in Europe. CAN bus is the central nervous system of a car and ensures that ECUs communicate with each other without complexity in the wiring. In figure 2.1 we see the complexity of wiring without the use of CAN Bus protocol, but with CAN Bus protocol the communication is simpler and allows other features to be added into cars such as Wi-Fi, Bluetooth, and wireless accessibility. Figure 2.2 shows a simplified CAN Bus network where each ECU connected to the bus.

2.2.2 CAN Bus Network

Controlled Area Network (CAN) Bus is a network that connects ECUs in a vehicle, so they work together to run effectively and efficiently [8]. These ECUs communicate through the CAN bus, which transport messages back and forth. ECUs can broadcast messages on the Bus to each
2.2.3 CAN Bus data frame

CAN Data frame is used for carrying messages between ECUs. The messages are broadcast on the bus, and the identifier field determines the message priority. Each ECU listens on the bus and constantly filtering out messages they want and disregarded the ones they don’t want. Also, each ECU uses the identifier field to determine if the message intended for them or not. CAN data frame consists of many fields, but most importantly:

- An identifier field is of 11-bits or 29 bits identifies the message.
- Data length is of data up to 4-bit determines the length of the message.
Data field is of up to 64-bits or 8 bytes contains the actual messages.

Messages in CAN Bus referred to as data frames. There are 4 types of data frames [9]:

1. The general frame used for transfer and broadcasting data between the transmitter or ECU.
2. An error frame used for detecting errors on a bus by the ECU.
3. The remote frame used for request data from another ECU.
4. Overload frame used for delaying packets between ECU.

Each ECU broadcasts the data frame in response to a request or change in-car driving behavior scenario. Therefore, an attacker can leverage their attacks by sending different data frames to perform a malicious task.

### 2.2.4 CAN Bus Protocol

Figure 2.4 shows normal CAN messages that is composed of identifier field, Control field, Data filed, CRC field, End of Frame field, and small fields in between. The arbitration field can be either standard with 11-bits identifier field or extended with 29-bits identifier field. Figure 2.4 shows the 11 bits arbitration field which server as identifier and prioritization ID to the ECUs [1]. ECUs synchronizing and monitoring messages in the CAN Bus.

CAN uses a signal for non-destructive message transmission, which differentiates between recessive and dominant states. In figure 2.4 identifier field determines the state of the message, which achieved by sending 0 indicating dominant state and 1 indicate a recessive state. Hence, the smaller the identifier field the higher the priority. However, there is no authentication mechanism on the Bus, but the receiver of the message sends an ACK signal to the sender which is located before the End of Frame field.

### 2.3 CAN Bus Vulnerabilities

CAN Bus protocol is inherently insecure. It is created three decades ago with minimal consideration to security. Although CAN Bus is resilient, robust, easy wiring, high speed transmission,
and error detection capability, it is vulnerable to attack. Vulnerabilities in the CAN Bus including but limited into:

- Lack of authentication: Messages are constantly broadcast on the CAN Bus. The CAN Bus data frame has no source identifier field to identify if the source is a legitimate ECU or not. As a result, an adversary can send messages in the CAN Bus to perform a malicious task or action.

- Lack of encryption: The encryption technique was never applied to the protocol because of hardware and software computing resource constrains. Moreover, many researchers suggested a redesign of the ECU to implement encryption, but it is not feasible. Therefore, no prevention against eavesdropping and sniffing attack.

- A common point of entry: Lack of authentication provides access to the bus by an adversary. For instance, an adversary can connect to the bus in Figure 2.3 and have access to other connected modules in Figure 2.1.

- Denial of Service: As mention before in section 2.2.2 messages with smaller Identifier has the highest priority. An adversary can flood the bus with a smaller identifier field to prevent other ECU from communicating and makes their service unavailable. Another denial of service attack could be flooding the bus with an error data frame [10].
2.4 CAN Bus Attack Surface

There are several of access points to the CAN Bus network that could potentially provide a way to exploit ECUs on vehicle. An adversary can compromise an ECU and may acquire the ECU to send messages to flood the bus or access other ECU to act maliciously. Access to buses can be categorized into direct access and indirect access.

2.4.1 Direct Access

Cars automakers are mandated by law to have an accessible On-Broad Diagnostic (OBD-II) port, typically located under the steering wheel. The OBD-II port allows mechanic shops and governmental agencies to access the CAN Bus for diagnostic information and emission test of the car. An attacker can access the car CAN Bus through OBD-II to comprise and exploit the car. Another direct access through vehicle multimedia such as CD player and USB [11]. An intruder can encode a malicious software onto a CD to exploit entertainment system of a car. further, the entertainment system may provide access to other bus network or ECUs. Also, direct access through the portable USB ports or electric charger ports.

2.4.2 Indirect Access

Indirect access is remote access through short or long-distance wireless access. A short-distance access point includes Bluetooth, on broad Wi-Fi, Remote keyless entry, and Tire Pressure Monitoring Systems (TPMs). Typically to exploit ECU wirelessly, an adversary must be in close proximity with access point. Conversely, a long-distance wireless access includes Global Positioning System (GPS) and cell phone network to break into the telematics units that normally used to provide road site assistance.

Attack surfaces are constantly changing as more features being added into the car. For instance, vehicle apps used for telematics services e.g. road site assistance, vehicle to vehicle communication, and remote diagnostic provide a remote attack surface. Further, as more cars become computerized
vulnerabilities and attack surfaces continue to increase. However, an adversary must know the system beforehand for successful exploitation.
CHAPTER 3. RELATED WORK

This chapter provides an overview of current literature research on car security. It is divided into two parts. The first part is on vehicle vulnerabilities and attack surfaces, and the second part is mitigation and detection.

3.1 Vulnerabilities and attack surfaces

Attacks surface are increasing as car manufactures continue to add more features into the car. Several of vulnerabilities exist on the CAN network as mention before, Wolf et al. were the first to describe CAN Bus network vulnerabilities [12]. These vulnerabilities include unauthorized access into the CAN Bus, no encryption mechanism for message integrity, and on-broad wireless communication vulnerabilities[12]. Further, they outline ways to achieve CAN Bus communication security. However, their methodology is infeasible due to the computing constraining on the CAN Bus network and ECUs.

Hartzell et al. discuss several of vulnerabilities found in the CAN Bus network that can be leveraged by an adversary including common points of entry, lack of authentication, multi-cast messaging, lack of encryption, lack of addressing, and multi-system integration [11]. additionally, newer features added by the manufactures such as on-broad Wi-Fi and remote control functionality provide attack surface if not secure properly. furthermore, Koscher et al. demonstrate car telematics security weakness that can be leveraged by an adversary to completely take over vehicle critical features such as disabling the brakes [13].

Koscher et al. provide an experimental security analysis of the modern interconnected vehicle [13]. They analyze a broad range of external attack surfaces e.g. short and long-range wireless channels. They define several entertainment systems that provide an entry point for hackers e.g. CD player, iPod port, USB port, and Electric charge port [13]. Further, they define threat models
as technical capabilities and operational capabilities of an adversary about the target system. Hacking into a car required technical skills to reverse engineer messages into the car network. Therefore, car manufacture are addressing these vulnerabilities in design to secure cars against remote attack.

An emerging trend wireless attacks on CAN Bus network as modern vehicles being to have external wireless communication for software updates or GPS signal. Chris Valeseka and Charlie Miller hijacked a Jeep Grand Cherokee through Sprint Cellular network remotely [14]. They took over vehicle critical features e.g. disabling breaks, steering wheeling, and turn on windshield wipers while the car in motion [14]. Also, a team of security researchers from Keen security lab were able to remotely control Tesla Model S P85 and Model 75D wirelessly through vulnerabilities on the infotainment [15]. They trigger the indicator lights, move the seat, open the sunroof, and control the brakes system [15]. Also, Koscher et al. constructed an attack on the TPMS and the telematics ECU [13]. They change the tire pressure signals and broadcasts the message over the CAN Bus. In addition, they were successfully able to reverse engineer the remote key signal to unlock the doors [13]. Moreover, Othmane et al. were able to successfully inject a speed and RPM messages into a Ford Transit through the OBD-II system [1]. They change the RPM and speed reading while the car still parked.

### 3.2 Mitigation and Detection

An adversary can take over a vehicle by simply transmitting a packet with a legitimate ID. For instance, Othmane et al. broadcast injected speed and RPM messages into the CAN Bus, which became available for all ECUs to read and other ECUs to each on [1]. Therefore, several message authentication mechanisms were proposed in the literature. Hiroshi et al. proposed a 3-4 Pre-shared key. This allows ECUs with the keys to communicate with each other [16]. Messages with no keys get disregard by the ECU. However, this method can’t prevent a spoofing attack. An adversary can spoof messages with keys and reverse engineer them back on the CAN Bus. Herrewege et al. proposed CAN Auth, which is also based on the Pre-shared key [17]. Similarly, the authentication protocol is susceptibility to spoofing attacks. Groza et al. proposed a lightweight
broadcast authentication protocol for CAN Bus network [18]. Furthermore, Kurachi et al. proposed a CaCAN authentication mechanism by using an authentication code on the data frame to checks for legitimate messages [19]. They proposed an extended CAN Bus where a centralized authenticator destroys the unauthorized data frame [19]. Thus, it enables ECU to check the authenticity of the messages.

The approach described above requires modification of the current CAN Bus protocol which is computational constraint by the CAN Bus and ECU capabilities. Kurachi et al. proposed a hardware method in which ECU detects unauthorized messages in the CAN Bus [20]. Each ECU checks the header information on the data frame and sends an error message frame to overwrite the unauthorized frame. Therefore, this prevents spoofing attacks. To prevent eavesdropping, several encryption methods were proposed in the software and hardware level. In [21], Farag et al. proposed a CANTrack that implements an encryption key on the data payload field on the data frame. However, the encryption mechanism requires modification of data field on the data frame. Therefore, researchers are focusing on intrusion detection system that doesn’t require any modification to the CAN Bus protocol and are computationally feasible.

In attack detection, several anomalous detection methodologies were proposed in the literature. These methods are based on frequency, entropy, correlation, location, formality, protocol, and range [22]. Taylor et al. proposed an intrusion detection based on frequency and time anomalous [23]. They measured inter-packet travel time at the different sliding windows. The intrusion detection uses an average Hamming distance between successive packet data fields. However, the method is not reliable because of the sort of anomalous. They inserted and deleted packets from the data to create an anomalous messages.

Intrusion detection based on frequency and entropy is unreliable and untrustworthy. These methods can be effective only for certain threat models and result in high false positive. To overcome this problem, recent literature propose the use of machine learning algorithm. In [24], Deep Neural Network (DNN) based approach was proposed as enhanced security for the in-vehicle network. Features were extracted from the CAN Bus data frame message. The corresponding
feature represents the labeling for the DNN to learn on [24]. Features were passed on DNN to learn; thus, allows the DNN to distinguish between normal messages and an attack messages. However, the approach is susceptible to high false positive and false negative. The identifier field on the data frame is constantly changing based on the intend receiver of the message.

In [25], Long Short-Term Memory (LSTM) is employed to detect online hijacking on an autonomous vehicle. The approach was to train the model on normal self-driving car behavior, then an alarm is raised upon deviation from the predicted normal behavior. Also, the method showed that LSTM can be implemented to detect vehicle input time-series data from speed and acceleration. The method is great for detection a change in driving behavior; however, the method doesn’t not explain the source of anomalous deviation. Moreover, the data set used in the study generated on a treadmill. To achieve an accurate result, a real car data set must be used.
CHAPTER 4. Data Preparation

4.1 Introduction

This chapter describes data preparation and preprocessing step. The first section describes the data collected for this research. The data consists of three data sets: normal data, speed data, and RPM data. The data sets collected by Othmane et al. from the 2017 Ford Transit 500 [1]. The second section describes the data preparation steps. The third section describes the data processing before training the LSTM RNN on the data sets.

4.2 Used Data

Table 4.1 shows the number of CAN messages that were used in the research. The data were collected by Othmane et al. [1]. The Speed data set consists of inject speed PId 254 and RPM PId 115 along with other ECUs messages.

Table 4.1 List of data set used in the research [1]

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Number of messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal CAN Data No injection of messages</td>
<td>23,960</td>
</tr>
<tr>
<td>2</td>
<td>Speed CAN Data injection of speed messages</td>
<td>88,492</td>
</tr>
<tr>
<td>3</td>
<td>RPM CAN Data injection of RPM messages</td>
<td>30,308</td>
</tr>
</tbody>
</table>

4.3 Data Preparation

The tachometer and speedometer are related and correlated. Engine optimal performance is proportional to the RPM and speed of the vehicle. The RPM gauge on the dash broad displays crankshaft rotation per minute while the speed gauge displays the speed of the car proportional to
The RPM reading. As a result, change the correlation between the speed and RPM can be detected using Cosine Similarity, Levene test, and T-test. An adversary can inject RPM or speed signal into the CAN Bus, as a result, their correlation will no longer exist. The Three methods are utilized to distinguish between normal driving behavior scenario and an attack scenario.

### 4.4 Data Preprocessing

Data preprocessing is an important step in the neural network training process; it transformed data into understandable format and prepares data for further processing. In figure 4.2, we can see that if cosine similarity, Levene test, and T-test data were passed on to the neural network model without transformation, the model will not be able to distinguish between the two data sets (Normal Records and Speed Records). Furthermore, empirical evidence has shown that data
preprocessing and transformation have improved accuracy and shorting training time in neural network. neural network learn by mapping input to output from the training sets.

4.4.1 Data Normalisation

Normalisation is the scaling of data between 0 and 1 in which the data set reorganized to increase cohesion (group data together) and eliminate redundancy. Redundancies have an adverse effect during the training step in LSTM RNN. Using the previous assumption that ECUs data under attack should have a smaller correlation than normal data, therefore, the ECUs data is dependent on each other, and they should be in close proximity. Also, data that relate to one other should be group together. Normalisation rescales the data set for each sample by dividing the sum of all the values in the data set. Figure 4.2 shows that normal records with no attack have a higher correlation value than speed data. Thus, reduce the training time and eliminate vanish gradient problem during the training step.

Normalizer implementation using Scikit-learn library specifically sklearn.preprocessing module:

- Reshaping the data into 2D matrices
- Creating a Normalizer Scaler object
- Passing the data set into the object
- Transform data with NormalizerScaler.fit_transform() method
4.4.2 Training / Testing split

After normalization of the data, the data were split into a training set and testing set. The training set contains the known output, and the LSTM RNN model learns on this data to be generalized to the testing set. Also, a training set is used to identify the predictive model of neural network. Using Scikit-learn library, specifically `train_test_split` method, we split the data into ratio of 60% for the training and 40% for the testing.
CHAPTER 5. Detection of Message Injection Using Similarity Metrics

5.1 Introduction

This chapter provides an overview of methods used for detecting message injection. The first section 5.2 describes the similarity metrics cosine similarity, Levene test, and t-test used to detect messages on data and develop precedence messages graphs. The second section 5.3 describes Bayesian point change detection.

5.2 Similarity metrics

5.2.1 Cosine Similarity

Cosine similarity is a metric used to measure similarity between two vectors. It measures the angle between the two vectors and determines whether the two angles are equal. For instance, the cosine similarity of vectors of x and y is given by Eq 5.1. Cosine similarity is often used to measure document similarity mainly parallelism between two documents, where each vector is associated with the number of words on the document. Cosine similarity outputs a number between 0 and 1, the closer the number to 1 the more similar the two vectors are.

\[
\text{Cosine}(x, y) = \frac{(x \times y)}{(|x|)(|y|)}
\]

5.2.2 Levene Test

Levene test measures the variance of the two samples [26]. The Levene test assumes that the variance of the two samples is approximately equal. It uses an F-test which allows the interpretation of the output as greater than 0.5 is non-significant, otherwise are significant [26]. Variance represents the difference between the average squared deviation and the mean in the sample. Injection of RPM
and speed messages into the CAN Bus altered the variance of the data. The change in variance of messages exchange in the CAN Bus will be determined using the levene test.

### 5.2.3 T-test

A T-test is a statistical test that checks whether the mean of two samples is different [27]. It is an inferential statistic that allows inference about the sample. T-test measures the difference between the samples divided by the variance within the sample. The T-test assumes the mean of the two samples to be equal. The T-test uses the p-value to determine whether the difference in the mean is significant or not [27]. Injection of messages in the CAN Bus changes the mean of the population of the messages exchange in the Bus.

### 5.2.4 Approach

Similarities between the ECUs’ messages data was measured using the cosine similarity, levene test, and t-test. Car ECUs communicate together in a complex way by exchanging messages to adjust in gear shifting or speed. For instance, idle speed is controlled through several ECUs and sensors sending messages to the Engine Control Module (ECM). The ECM takes the input from sensors and perform calculations and table lookup using the stored software in the microcontroller chips. Then, the program outputs needed information such as the best ignition timing and adjust fuel injection. Therefore, input and output data from different ECUs are related. Alteration of this communication by injecting messages on the CAN Bus may result change on car behavoiur, which can be detected using cosine similarity, leven test, and t-test. Further, Othmane. at el, showed that speed and RPM values have a strong correlation, as a result, injection of fabricated speed and RPM messages disturbs their correlation [1]. Under no attack scenario, speed and RPM messages have a high correlation; conversely, they have low correlation under an attack scenario. Three correlated methods were used to create messages precedence similarity graphs that distinguish between normal driving behavior (no attack) and an attack scenario (under attack) as shown in figure ??.

In Figure 5.1 (a) The value of zero indicates that messages on the CAN network are
Figure 5.1 Cosine Similarity and attack vector for speed, RPM, and normal Data

highly related to each other, hence, no attack state. Conversely, the value of one indicates that the messages on the CAN network are not related. Figure 5.1(b) and (c) depicted the Cosine Similarity on the speed and RPM data with injected attack messages.

5.3 Point Change Detection

Graph analysis is a branch of data science that deals with a visual representation of the data in the form of a graph. Graphs can be used for fraud detection, social network, and detection of communities in the social media platform. For instance, a phone number can be used to represent a vertex in graph and detection of fragile when a phone number linked to more than one person on the graph. Graphs provide a better way of visual abstract of relationships and interaction. A simple network graph consists of vertices represent the objects and edges represent the entity between the vertices. Networkx Python package was used to measure the centrality between the
PIds [28]. Networkx is an open-source python package for studying and creating of structure in the network. Networks have long interested researchers in social data analysis e.g. understanding the relationship between communities. Networks are great tools to study complexity in data sciences from people’s relations to protein interaction and anything in between. In our case, Networkx was used to help us visualize the connection between the messages PIId IDs.

5.3.1 Approach

Figure 5.2 is a circle graph visualization representation of the connection between the nodes (PIds ID). The PIIds ID were represented as vertices on the graph, and edges are the connection between the PIIds as shown in figure 5.2. Node attributes were added as the number of connections between the nodes. The Figure 5.2 shows that there are nodes with many connections, which gives a clue as they are important on the graph. Networkx closeness centrality was used to measure and determine the most important messages PIId ID. The parameters for the networkx graph are:

- Vertices: CAN Bus PIIds ID whose network we are building
- Edges: represents the connection between the PIIds ID

5.3.2 Closeness centrality

The aims of centrality is to find the most important nodes in the network. We compute closeness centrality for each PIIds ID at time interval ranges from:

- 1-24 seconds for normal data.
- 1-34 seconds for RPM data.
- 1-87 seconds for speed data set.

Closeness centrality measures the average shortest path distance from node u to all other nodes[28]. The node with the highest centrality is the closed node in reference to all other nodes. For example, PIIds 110 and 339 were found to have the highest closeness centrality. Thus, means it
Figure 5.2  A circle graph representation of RPM PIds. The number of connections represented the attributes between each PIds.

takes a few steps to reach nodes (PIds) 110 and 339. Conversely, nodes (PIds) 211 has the lowest closeness centrality, which tends to take more steps to reach the nodes. Closeness centrality was calculated using formula 5.2:

\[ C(u) = \frac{(n - 1)}{\sum d(v, u)} \]  

Where \( d(v,u) \) is the shortest path between \( v \) and node \( u \), and \( n \) is the number of nodes that can reach node \( u \).

5.3.3 PyMC3 Bayesian Point Change Detection

Point change detection is a statistical method that determines abrupt change in time series data using probability. It is great for detecting anomalous in a time series set and detecting abrupt change in a sequence of data. We used PyMC3 an open-source python platform for probabilistic programming using Bayesian statistical modeling: it allows users to do Bayesian inference on time
series data. PyMC3 uses Theano as a backend or low-level programming library to perform vector multiplication and compute the gradient [29].

5.3.4 Bayesian Inference and approach

Bayesian inference is a way of updating information about prior parameters from the observation data or evidence and generative model (PyMC3 model) [29]. However, it is a computationally intensive method. Normal, RPM, and speed data set were divide into intervals as describe in section 5.3.2. For each time interval closeness centrality was calculated using formula 5.2. Figure 5.3 illustrates the approach used for Bayesian Inference for detecting point change detection. The unknown parameter is the parameters that the model tries to learn or update from the evidence we passed (calculated closeness centrality). Our unknown parameters are:

- Early mean is the mean before the change point occurred.
- Late mean is the mean after the change point occurred.
- Early standard deviation for the data before the change point occurred.
- Late standard deviation for the data after the change point occurred.
- The switch point is a point where the change occurred.
Figure 5.3 PyMC3 model architecture
CHAPTER 6. Detection of Message Injection Using Similarity Metrics and Long Short-Term Memory Neural Network

6.1 Introduction

This chapter describes the LSTM RNN approaches used for the detection of injection messages on the CAN Bus network. Section 6.2 describes Machine Learning, Deep Learning, and neural network model. Section 6.3 describes approach for using LSTM RNN for detecting injection of messages in CAN Bus network.

6.2 Overview of Machine Learning

Machine learning is a branch of artificial intelligence that allows a machine to learn from examples, experiences, and data. Figure 6.1 here shows subset representation in a hierarchy structure between Artificial Intelligence, Machine Learning, Deep Learning, and Long Short Term Memory (LSTM). Machine learning enables a machine to learn and think like a human. Therefore, machines make decisions based on the data rather than been explicitly programmed to carry a certain tasks. Machine learning technology power most aspects of modern society: from web searches to content filtering on social networks to recommendation on e-commerce websites. It is increasingly present in consumer products such as cameras and smartphones. Deep learning powers most of the developments in machine learning. Machine learning algorithms advent in many aspects of data sciences, however, they struggle in complex data such as image classification, voice recognition, and language translation.

Deep learning is an advanced machine learning algorithm based that mimics brain perception of data. It revolutionize many sciences fields thanks to increased computing power. It’s the main technology in Google search engine, Facebook faces recognition, language translation, and autonomous driving. In fact, in the automotive industry deep learning is incorporated into car safety features
for detecting near-miss collision. Artificial Neural Network (ANN) is embedded with several hidden layers that adjust themselves to the properties of the data they are learning. Figure 6.2 shows a basic artificial neural network feed-forward built with 3 inputs units. ANN consist of three interconnected layers: input layers, hidden layers, and output layers. Information follows from the input to the output layer. The first layer is the input layer which takes the input features and multiplies the inputs by the corresponding weights, and then applies a non-linear activation function, and pass them to a hidden layer. This process gets repeated for each input feature pass into the input layers. This type of learning process good for a supervised machine problem because its feedforward network. There are different variants of ANN, but Recurrent Neural Network (RNN) is the most widely used variants of ANN.

RNN is good for processing sequential data and prediction from these data. It forms a sequence of neural layers, input from the previous layer is combined with a new hidden state to form a new vector [3]. Then, the vector passes through the activation function to adjust it’s output between 0 and 1. This process repeats multiple times until the vector passes on to the output layers. RNN, however, suffers from vanish gradient descent and long-term dependence. To counter the vanishing
gradient problem and long-term dependence, Long Short-Term Memory (LSTM) was introduced. LSTM is a special type of Recurrent Neural Network that capable of learning long term memory dependence. These memory dependence will be forgetting during the validation step. Figure 6.3, shows the absence of memory dependence. LSTM was created as a solution and explicated design to solve the short-term memory problem [3].

Figure 6.3 shows RNN processing sequence of data one element at a time. New inputs or features coming into the network gets multiplied by the previous hidden state; previous hidden state gets updated each time new input is received, the previously hidden act as a memory for RNN [4]. The vector goes through a tanh activation function, which keeps the vector values between -1 and 1. This process continues until the final output of the network is calculated.

LSTM complex structure solves the problem of vanishing gradient, and proven to be very useful for long term dependence. The architecture allows LSTM to make decisions about the to store
and what to forget through different gates and cells. The core concepts of the LSTM structure as shown in Figure 6.4 are:

- Forget gate decides on the information to throw away from the new input and the previous hidden state.

- Input gate decides on the information to update from the new inputs to the previous hidden state.

- Output gate this outputs the information based on the information from the cell state and the input gate.

- Cell state allows information from the previous hidden state to be stored and remembered. The cell state gets modified by the forget gate.

- $\alpha$ is sigmoid activation function helps regulate the values between 0 and 1.

- The tanh is also an activation used to help regulate the values between -1 and 1.

6.3 Approach

We assumed that messages from the ECU in CAN Bus network related to each other during a car normal driving behavior scenario. In an attack scenario messages relationship disturbed. The three methods describe in section 5.2 was used to detect the changes in the messages behavior.
Based on this assumption, two data sets were used to train the LSTM RNN network with different Window Size and constant threshold. Prior to train, the data set was transformed, so that, the model can train faster and output better prediction. The trained model is then used to predict on test data set. The model outputs prediction values between 0 and 1. A threshold was used to detect an attack if the output is great than threshold and 0 otherwise. The following steps were taken to train an LSTM RNN:

1. Data preprocessing see section 5.2
2. Training / Testing split see section
3. Building the Model
4. Training the Model

6.3.1 Building the Model

The neural network was constructed using Keras [30], which is high-level neural networks APIs that runs on top of Google TensorFlow. Keras does not do its low-level operations such as vector multiplication, graphs, tensors, operations, and sessions; it relies on the low-level backend for that. Conversely, TensorFlow is an open-source library that is used for numerical computation and machine learning algorithms. TensorFlow has adopted Keras as their high-level APIs since TensorFlow 2.0.

6.3.2 Keras Model

The LSTM RNN was constructed using the sequential model, which is a linear stack of layers as shown in figure 6.5. LSTM RNN layers are very powerful in sequence prediction problems because they are able to store past information for a long time. This is an important in our case because learning the value of the Cosine Similarity, Levene test, and T-test are crucial in distinguishing between an attack state and no attack state. Additionally, LSTM expects input data to be in a
Figure 6.5 LSTM RNN model architecture used to detect an attack in-vehicle communication network.

Specific format usually a 3D array. The output shape of the first layer will be passed on to the next layer.

Figure 6.5 depicts Neural layers were added in the confirmation:

1. Input shape of (none, 1,1) matrices

2. 42 Units for the first layer which specify the dimensionality of the output from the first layer, with return sequence set to true to return the last output from the layer

3. Dropout layers to prevent overfitting

4. Another LSTM RNN layer with 12 units and return (none, 12)

5. Dense layer with only one unit as an output layer.

6. 20 percent dropout was specified to be dropped from the input layer.

7. Dense layer that specifies the output of 1 unit

6.3.3 Training the Model

The fact that our data is a label with output makes this a supervised learning. Initially, we use the training data set, put it through the neural network, and then get to predict the right label for
each record as the output. The LSTM RNN is trained on a training set that is made of a mix of speed (low correlation data set) and normal data (high correlation data set) to distinguish between an attack state and no attack state. The testing set was used for validation and prediction. Several trails have been carried out in order to find the best values for hyperparameter optimization:

- Dropout regularization technique to prevent overfitting in the model layers units
- Learning rate determines the learning rate for the model.
- Optimizer an argument required for training the model. network architecture specifying the number of hidden layers and the number of units in each layer
- Regularization penalty for weight size to prevent overfitting in the model units.
- Epochs is the number of iterations to run from the data set through the model.
- The batch size defines the number data that going to be fed into the model.

Figure 6.5 shows the final output for the hyperparameter optimization that was chosen to train the model. The output of the first layer will be 1, 42 units. Then, a 20 percent dropout from the output of the first layer. Subsequently, the data passed on the second hidden layer that consists of 12 LSTM units. The performance Model tested at different Window Sizes.
CHAPTER 7. Result and Discussion

This chapter discusses the result of identification of messages injecting using LSTM RNN and point change detection. Section 7.1 discusses the use of similarity metrics for the detection of an attack on car network. Section 7.2 describes LSTM RNN performance on the RPM and speed data. Section 7.3 discusses the LSTM RNN prediction on speed data. Section 7.4 discusses LSTM RNN prediction on RPM data and limitation. Section 7.5 discusses point change detection result on the detection of messages injection on car network.

7.1 Similarity Metrics Output

Methods described in section 5.2 were utilized to distinguish between normal driving behavior scenario and attack scenario. The value of zero in Figure 7.1 (a) indicates that messages on the CAN network are highly related to each other. Conversely, the values in figure 7.1 (b) one indicates that the messages on the CAN network are not related or correlated because of the injection of messages into the CAN Bus. Figure 7.1 (b) and (c) depicted the cosine similarity on the speed and RPM data with zeros. Before the injection of messages into the CAN network, the data were recorded. As the driver turns on the ignition, the ECUs turn on and begin to receive and send messages in the CAN Bus network, hence the presence of the zeros values on the attack data sets. Additionally, these values were shown to be on the plot along with injected messages.

Figure 7.1 shows precedence graph developed using cosine similarity to distinguish between no attack scenario and an attack scenario where an adversary injected speed and RPM messages. Values of 1 indicate that data are not related to each other, and hence an attack state. Conversely, values of 0 indicative of data are related to each other, and hence no presence of injected messages.

Also, levene test was employed to assess the variance between the two states. Levene test looks at dispersion within the data set; It assumes that highly related ECU messages came from the
same data set. Messages with low levene test assumed to be correlated while messages with high levene test not correlated. An Injection of speed and RPM messages tend to disperse the messages away from the mean. In consequence, we assume those values with high dispersion are indicative of an attack state. When the output applied to the attack vector, it showed zero for messages with a low levene test and one for values with high levene as shown in Figure 7.2 (a). Also, Figure 7.2 (b) and (c) depicts the similarity between the detection of attacks using the levene test figure 7.2 shows the levene at window size 100.

T-test was used to detect the injection of fabricating messages into the CAN Bus network. The method measure whether the average value differs significantly across precedence messages exchange in the CAN Bus e.g. if we observe a small p-value, we can say that the two data set are
(a) An attack vector for normal car driving behavior  (b) An attack vector for injected speed messages

(c) An attack vector for injected RPM messages

Figure 7.2 Developed messages precedence graph for levene test on Speed, RPM, and normal data set

similar. Figure 7.3 (a) depicts values of zero as a normal functionality of the car (No injection of messages). Conversely, values of 1 indicate an attack as shown in figure 7.3 (b) and (c).

The three methods described in section 5.2 were used to observe the correlation between the normal driving behavior of a car and during an attack scenario. Applying these methods will result in two different data set along with an attack vector. Then, data from each method was passed on to the LSTM RNN. However, levene and t-test were partially success in detection of attack using the LSTM RNN see a section 7.3 for more details.

7.2 Evaluation of LSTM RNN

Figure 7.4, we see that the LSTM RNN model has comparable performance for both the train and validation data sets. Also, the model reaches convergence and plateau at training epochs 65,
Figure 7.3 Developed messages precedence graph for t-test on Speed, RPM, and normal data set

which attribute to train and learn step. However, this behavior was only seen for cosine similarity because the model was able to map the values of the cosine similarity to the attack vector. Levene and t-test converged and plateau at early epochs; hence, the model can’t distinguish between normal driving behavior messages and an attack state (injection of messages). Table 7.1, describes the parameters used to training the LSTM RNN.

7.3 Speed Data Set

LSTM RNN is an effective method for the detection of anomalous and sequence classification in the data set given that the data are labeled. The LSTM RNN is trained on label data set normal and speed (attack data). The intuition for this approach is that the LSTM RNN model mimics and adjust its weights and different cell state to the features on the training set. The model learned to
distinguish between normal behavior driving and attack scenarios in 4.1 seconds as in 7.1. We fed the correlation measurements from the Data Preprocessing along with their attack vector (0 for no attack and 1 for an attack). In the LSTM RNN model, the output layer was a dense layer with sigmoid activation function which out values between 0 and 1, and hence that a threshold was used to map the output to 0 and 1.

Figure 7.5 depicts the output prediction for the three methods respectively. From figure 7.5 (b) and (c) we can see clearly that levene and t-test are prone to false positive and false negative more than cosine similarity because the methods are partially successful on differentiate between the two-state. Additionally, the model missed identify 14 attacks and identify 20 as an attack for the levene test data set. Similarly, the model missed identifying 9 attack states and identify 18 as an attack state where in actuality they are normal driving car behavior.
Figure 7.5 LSTM RNN output prediction for Cosine Similarity, Levene Test, and T-Test testing data set parameters were window 200 and threshold 0.80 respectively. The output accuracies were:

- 96.6% Cosine Similarity
- 82.5% Levene Test
- 86.7% T-test

7.3.1 Training With Different window sizes

The efficiency of the model was compared across different window sizes while keeping the threshold constant. Table 7.2 depicts the accuracy prediction for cosine similarity, levene test, and t-test respectively relative to the window size.
Table 7.2 LSTM RNN accuracy prediction at different window size for Speed data set

<table>
<thead>
<tr>
<th>Window Size</th>
<th>Cosine Similarity %</th>
<th>Levene Test%</th>
<th>T-test%</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>98.465</td>
<td>82.353</td>
<td>81.586</td>
</tr>
<tr>
<td>200</td>
<td>96.907</td>
<td>82.474</td>
<td>86.082</td>
</tr>
<tr>
<td>300</td>
<td>96.875</td>
<td>76.563</td>
<td>86.719</td>
</tr>
<tr>
<td>400</td>
<td>96.875</td>
<td>86.458</td>
<td>75.000</td>
</tr>
<tr>
<td>500</td>
<td>96.053</td>
<td>69.737</td>
<td>81.579</td>
</tr>
<tr>
<td>600</td>
<td>79.365</td>
<td>85.714</td>
<td>73.016</td>
</tr>
<tr>
<td>700</td>
<td>82.209</td>
<td>75.926</td>
<td>83.333</td>
</tr>
<tr>
<td>800</td>
<td>100.000</td>
<td>86.957</td>
<td>78.261</td>
</tr>
<tr>
<td>900</td>
<td>87.805</td>
<td>75.610</td>
<td>82.927</td>
</tr>
<tr>
<td>1000</td>
<td>78.378</td>
<td>67.568</td>
<td>78.378</td>
</tr>
</tbody>
</table>

Figure 7.6 LSTM RNN prediction on Cosine Similarity, Levene Test, and T-test vs Window Size for speed Data Set

Figure 7.6 and Table 7.2, we can see the similarity between window size and cosine similarity, levene test, and t-test respectively. Figure 7.6 shows increase in window size decrease in LSTM RNN accuracy prediction. However, at window size 800 the model prediction was 100%, which is due to overfitting on the LSTM RNN. During the training, the learning patterns from the training data don’t generalize to the test data feature, which leads to random guessing. At window size 100, the number of training data set was 516, and at window size 800 the training data set was 123. The relationship between the parameters on the model can different to number of units on the hidden layers and number of hidden layers, affecting the learning on LSTM RNN. For instance, when the amount of data set is 10% of the training data set at window size 100, this leads to overfitting and
results on the model memorization instead of learning. In our case, we applied hyperparameters optimization such as 20% dropout, regularization, batch normalization, and reduce the size of LSTM RNN to prevent overfitting. However, the fact that our training data less than the LSTM RNN parameters, produce overfitting.

Also, levene and t-test were oscillating as window size increase. Previously stated the two methods are partially successful in similarity between two different states. As a result, their accuracy were lower than the cosine similarity and tend to have more false negative and false positive on the data set.

From Figure 7.6 we can deduce that Cosine Similarity window size 100, 200, and 300, had much better accuracy and fewer false positive and false negative. Also, LSTM RNN was able to distinguish between an attack scenario and no attack (normal driving behavior) with high accuracy. The LSTM RNN with Cosine Similarity can be used to detect an attack on a car network without much knowledge of CAN Bus data. In addition, LSTM RNN with data from Levene and T-test can be used as the first line of attack detection on car network and cosine similarity as a validating step.

7.4 RPM data set

Similar to the method described in section 5.2, cosine similarity, levene test, and t-test data were fed into the LSTM RNN to distinguish between an attack state and normal car driving behavior. In Figure 7.7 (a), shows the LSTM RNN had 2 false positives for Cosine Similarity, contrary to the levene test and t-test. Figure 7.7 (b) and figure 7.7 (c) respectively, shows LSTM RNN identifies 13 attack states that are normal car driving behavior. Similarly, LSTM RNN was not able to distinguish between the normal data and attack data from the T-test. In contrast, the speed data set showed much more correlation between the two the data then the RPM.

Figure 7.7 depicted heatmap plots for LSTM RNN prediction Cosine Similarity, Levene, and T-test testing data respectively. Parameters were window 200 and threshold 0.80. The output accuracies were:
Figure 7.7 LSTM RNN output prediction for Cosine Similarity, Levene Test, and T-Test for RPM data set

- 96.970% Cosine Similarity
- 79.032% Levene Test
- 67.742% T-test

7.4.1 Training with different window sizes

The efficiency of the model is compared along different training set with different window sizes and constant threshold. Table 7.3 and Figure 7.8 depicts the accuracy prediction for cosine similarity, levene test, and t-test respectively.

The LSTM RNN fails to distinguish between normal driving behavior and attack scenarios (injection of messages). Figure 7.6, we can see the convergence of the LSTM RNN 100% prediction accuracy on the cosine similarity method after size window 700. Therefore, this mainly due to model overfitting, which was described in section 7.3.1.
Table 7.3 LSTM RNN accuracy prediction at different window size for RPM data set

<table>
<thead>
<tr>
<th>Window Size</th>
<th>Cosine Similarity %</th>
<th>Levene Test%</th>
<th>T-test%</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>93.860</td>
<td>73.684</td>
<td>84.211</td>
</tr>
<tr>
<td>200</td>
<td>96.970</td>
<td>79.032</td>
<td>67.742</td>
</tr>
<tr>
<td>300</td>
<td>95.082</td>
<td>54.098</td>
<td>45.902</td>
</tr>
<tr>
<td>400</td>
<td>51.111</td>
<td>44.444</td>
<td>57.778</td>
</tr>
<tr>
<td>500</td>
<td>91.429</td>
<td>68.571</td>
<td>74.286</td>
</tr>
<tr>
<td>600</td>
<td>93.103</td>
<td>44.828</td>
<td>51.724</td>
</tr>
<tr>
<td>700</td>
<td>100.000</td>
<td>48.000</td>
<td>64.000</td>
</tr>
<tr>
<td>800</td>
<td>100.000</td>
<td>38.095</td>
<td>76.190</td>
</tr>
<tr>
<td>900</td>
<td>100.000</td>
<td>33.333</td>
<td>55.556</td>
</tr>
<tr>
<td>1000</td>
<td>100.000</td>
<td>56.250</td>
<td>37.500</td>
</tr>
</tbody>
</table>

Figure 7.7 shows that the LSTM RNN missed labeling 20 normal driving behavior with no fabricated messages. In table 7.3, we see that the accuracy for window 700 to 1000 for levene test is less than 50%. Also, in figure 7.7 the accuracy continues to diverge downward. Thus, the reason for the lower accuracy showing graph is due to data labeling. The levene and t-test failed to distinguish the different between injection messages into the CAN Bus network and normal car driving behavior scenario. The method can be used to discover an attack state. In conclusion, the LSTM RNN failed to distinguish between the attack state and no attack state for the RPM data set, particularly Levene and T-test.

Our result shows that LSTM RNN was successful in detecting an injection of messages into the car network using the cosine similarity method. The model performs poorly on levene and t-test for speed and RPM data set with high false-positive and false-negative. A possible explanation for this result is the two methods can’t distinguish between the two data set normal driving car behavior leading to false negative and false positive in the testing stage. However, LSTM RNN performed better in the Cosine Similarity for the speed and RPM.

levene test and t-test are partially successful in detecting injection of messages in the car network using LSTM RNN due to dispersion on the data set. This dispersion could be from instantaneously change of RPM and Speed data to adjust in car driving behavior.
Figure 7.8 LSTM RNN prediction on Cosine Similarity, Leneve Test, and T-test vs Window Size for RPM Data Set

Figure 7.9 Measured centrality value vs time in seconds

7.5 Point Change Detection

Figure 7.9 shows that output from the centrality measurement. We assume that the injection of fabricated speed and RPM messages into the CAN bus network affect centrality. All nodes in a CAN Bus network listen and send messages at the same time. When messages are broadcasted in a CAN Bus network, they follow certain sequences behavior. Injection of fabricated messages PIDs 115 and 254 on the CAN network will abruptly change behavior of other ECUs messages. We hypothesize that deviation of this behavior can be used to deduce an attack on the car network.

Figure 7.9 depicts the calculated closeness centrality values for each PIDs on the CAN Bus at a period of 1 to 29 seconds. The List on the right is the PIDs that have an abrupt change in
their closeness centrality. Additionally, the PIDs change their closeness centrality in response to the injection of fabricated messages. The value of closeness centrality was measure at different times interval. Figure 7.9 shows a shift in the centrality at 4 seconds for PIDs 115 and 254. The change in closeness centrality for PIDs 254 is smaller than in 115. For instance, PIDs 117 at time 6 seconds changed from closeness centrality of 0.41 to 0.47. Similarly, PIDs 211 changed from 0.33 to 0.47. In contrast to the change in 115 from 0.47 to 0.78. Therefore, the change in centrality for injected messages was larger than the ECUs messages. Further, some ECUs message tend to follow the shift in centrality.

PIDs values with higher closeness centrality values don’t exhibit the change affect because of the priority of the message on the CAN Bus packet frame. Also, the changes in the closeness can be contributed to a change in-car driving behavior scenario. Although we cannot determine the causes of the change of the closeness centrality, we can detect a change in the closeness centrality values.

7.5.1 PyMC3 Point Change

WE used Bayesian point change detection from PyMC3 to estimate abrupt changes in the RPM time series. The technique models the changes by estimating the posterior probability for the change in mean and standard deviation before and after the change point occurred. Figure 7.10, the shows a trace plots for MCMC iteration sampling from the distribution. The plots on the right column shows MCMC algorithm converged (finished sampling data from the posterior distribution) Also, Top left represents the probability when the change point occurs. Here, $\mu_1$ and $\mu_2$ defined as stochastic variable early mean and late mean respectively. $\sigma_1$ and $\sigma_2$ represent the standard deviation before and after the change point occurs. The likelihood function is chosen to be Gamma distribution. The model randomly draws 10000 samples of parameters $\mu_1$ and $\mu_2$. Then, for each parameter, the model draws 100 random numbers from the Exponential distribution for the change in mean. The result of the trace plot is summarized in table ??.
Figure 7.10  PyMC3 Trace plot For Pld 115

Table 7.4  PyMC3 parameters output for Pld 115

<table>
<thead>
<tr>
<th>PyMC3 Parameters</th>
<th>Description</th>
<th>Mean Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>switchpoint</td>
<td>The change point where the change occurred</td>
<td>3</td>
</tr>
<tr>
<td>mu_1</td>
<td>Early mean before the change point</td>
<td>0.414</td>
</tr>
<tr>
<td>mu_2</td>
<td>late mean after the change point</td>
<td>0.832</td>
</tr>
<tr>
<td>sd_1</td>
<td>Early standard deviation before the change point</td>
<td>0.011</td>
</tr>
<tr>
<td>sd_2</td>
<td>Late standard deviation after the change point</td>
<td>0.031</td>
</tr>
</tbody>
</table>
Figure 7.11  Point change detection probability output

Figure 7.5.1 shows that the change point occurs at 4 seconds, however, the Bayesian model predicted the change between 2.5 and 3.5 with a mean of 3 seconds and 94% probability. The shaded region on the graph helps explain the probability of where the change point occurred. Figure 7.5.1 depicted the shaded region in Figure 11. Similar results were obtained for Pids 117, 211, 346, 113, and 344.

Our experimental findings revealed that not all PIDs on the CAN Bus network change their closeness in response to attack scenarios. The change in closeness can result from instantaneous on the car driving behavior adjusting to decrease or increase in speed. Further, high PIDs with priority values don’t get the effect with the injection of messages into the CAN Bus. We conclude that Bayesian point change detection can’t detect an attack on the CAN Bus network.
CHAPTER 8. Conclusion and Future Work

This thesis investigates the use of similarity metrics on message similarity graph to detect message injection in car network. The data sets used in this research were collected by Othmane et al. From 2011 Ford Transit by injecting speed PID 254 and RPM PID 115 [1]. The injection changed the tachometer and speedometer reading. Furthermore, we demonstrate that modification in CAN Bus network to improved security computational infeasible. We computed messages precedencies similarity graph using cosine similarity, levene test, and t-test to distinguish between no attack state and under attack state. Then, we demonstrate that LSTM RNN successfully detected the injection of messages into the car CAN Bus network using the cosine similarity method. However, the LSTM RNN model performs poorly on levene test and t-test for speed and RPM data set with high false positive and false negative. Therefore, we conclude that LSTM RNN with messages precedence similarity graphs from levene and t-test perform poorly with high false positive and false negative. The proposed model provides a real-time detection for injection attack on the CAN Bus network. Also, we demonstrate point change detection to observe an abrupt change in PIDs upon the injection of messages into the data. Our experimental findings revealed that not all PIDs on the CAN Bus network change their closeness centrality in response to attack scenarios. PIDs with high priority values don’t get the effect with an injection of messages into the CAN Bus. We conclude that Bayesian point change detection can’t detect an attack on the CAN Bus network.

This thesis considered only tachometer and speedometer attacks in which the LSTM RNN model applied. To prototype a device, LSTM RNN needs to be applied to other types of ECUs attack e.g. Engine control units, transmission control units, and Electronic stability control units.
Bibliography


