Security in CAN Bus

Rushabh Patel (131070005)
Siddhant Shah (131070010)
Abhijit Gupta (131070014)
Mehul Jain (131070051)

May 2017

ABSTRACT

The evolution of electronics in the previous century has changed the nature of vehicles dramatically. Nowadays, the control of different systems within a vehicle is carried out using tens of electronic control units (ECUs). These ECUs are clustered into networks having gateways in-between. Several standards are used for the communication within these networks and between them. The most widely used standard is the Controller Area Network (CAN) bus standard. In general, the design of such networks has been always concerned with reliability and safety. There was no much attention paid to the security of such networks. This is because there was no clear evidence if the security of such networks could be compromised. Practical experiments have demonstrated practical attacks on different systems inside vehicles. This included the ECUs controlling engine, brakes, lighting, climate control lighting and body controller. As a result, it is possible for an adversary to take-over the control of a vehicle and harm the passengers. This highlights a major risk for all cars; whether they have already been sold or they are still under development.

There are two types of security vulnerabilities that make these attacks possible. The first type is due to the inherent weaknesses of the used communication standards themselves. The second type is due to the deviation from security standards. The goal of this project is to investigate different methods by which the security of in-vehicle communication networks can be improved. The proposed methods consist of different levels of security enhancements ranging from prevention, protection and detection. The project focuses on achieving protection of the CAN bus against adversaries. We would like to introduce message encryption and source authentication protocol for the CAN bus. This can be achieved by designing a new lightweight authentication protocol. The main aspect of the protocol will be that it will be simple and practical so that it can be adopted directly in the automotive industry. The protocol does not need any extra hardware modifications and thus can be deployed inside cars that have already been sold. For intrusion detection of various kinds malicious attacks in the CAN bus, we researched and found various Machine Learning and Data Mining approaches which promises to detect malicious CAN packets. The methods used for anomaly detection include Long-Short Term Memory Neural Networks, Deep Neural Networks technique, Entropy based anomaly detection and finding associations between the different ECUs using Apriori algorithm.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The automotive industry has been affected dramatically by the advances of electronics during the last century. The introduction of electronics into cars has made them no purer mechanical systems. Almost every new car manufactured nowadays contains tens of electronic control units (ECUs). ECUs have been introduced initially in cars for the purpose of engine management. Later on, they have been used for controlling many systems inside cars like brakes, transmission, airbags, climate control, power windows, infotainment and telematics. Also, they have been used to add new capabilities to cars like parking assistance, lane departure warning, blind spot detection, etc.

Electronic control units themselves are not pure hardware components. Instead, they consist of both software and hardware. In most cases, the ECU consists of a set of electronic circuits that are controlled using a microcontroller running from tens to hundreds thousands lines of code. As a result, a considerable amount of production defects may arise from software. When such defects are discovered after cars have been sold, the ECUs need not to be physically replaced in order to fix those defects. Typically, reprogramming the software of the ECU can solve everything. If the defects are not critical, then car manufacturers do not need to recall their defected cars in order to fix them. Instead, software reflashing can be made during regular service that is made in authorized workshops.

1.1 New threats

Although it sounds good to have the ability to fix defective parts using software, it introduces many safety considerations. Consider an attacker who has physical access to an ECU and could insert malicious code into it. Since all car networks are linked together through gateways, it is possible for this malicious code to control several safety-critical parts of the car. This highlights a major risk which has been demonstrated in [2]. However, if we assume that the attacker may have physical access to the car then he may be able to replace complete ECUs not only replace the software flashed in them. That's why information security has not been introduced in the automotive industry except for specific systems like immobilizers.

During the development of communication networks inside cars, it was always assumed that the network is a trusted zone. May be this assumption came from the fact that cars had limited interfaces with the outside world. The OBD II port (used for diagnostics) was the main external interface

between the car network and the outside world. Recently, cars contain several wireless interfaces for many purposes. For example, cars now contain tire pressure monitoring systems TPMS, smartphone integration using Bluetooth, web connectivity, etc. Other new interfaces may be deployed for vehicle-to-vehicle and vehicle-to-infrastructure communications. Thus, cars are no more closed systems. Accordingly, the car network is no more a trusted zone. As a result, it is required to secure the internal network of cars from the threats that may arise from the new interfaces. This security may be needed either at the interfaces or within the network itself or both.

This situation is similar to the case of personal computers during the spread of internet usage. Before the internet, the spread of computer viruses was limited to using infected removable media (floppy discs by that time and compact discs later on). Computer users already had their own precautions to protect their computers from those threats. With the emergence of the internet, personal computer users found themselves susceptible to new types of threats that may spread on a global level. It took some time for security systems to become mature and provide and appropriate protection. Unfortunately, the automotive industry nowadays is facing the same problem again.

1.2 Purpose

The goal of the project is to attack the existing system of CAN bus in Cars and exploit its vulnerabilities and propose a preventive and detective measures of the same. Security is utmost important in the communication channel. As CAN bus is being widely used, attackers are intruding the system and exploiting it with the intention of theft, surveillance, kidnapping and terrorist attacks. Thus it is paramount to have the security layer in the CAN bus and avoid these intentions of the attackers by not intruding them. This project will give boost to the community for security in CAN Bus.

1.3 Motivation

Lack of security in the CAN Bus Protocol which is one of the most upcoming technologies of automated cars that motivated us to undertake this project. Controlled Area Network is becoming the backbone of communication amongst the components in Car, due to which many attacks are taking place inside the Car. Thus trying to add the security layer on widely used protocol in Car which will reduce these attacks to a great extent and enhance the security of the protocol.

1.4 Problem Statement

The project is based on attacking of CAN Bus and prevention and detection of attack on CAN Bus. Developing an authentication scheme that would add a logical layer of security over the protocol without affecting the timeliness of the messages would be our ultimate goal. Another domain that we delve into is detecting intrusion using Machine Learning Techniques such as Deep Neural Network, Long Short Term Memory and Data Mining techniques such as Apriori Algorithm and Entropy.

1.5 Scope

We have studied the working of the CAN protocol which is the widely used inside cars so that many different components such as Anti-braking system, Door lock, ECU, Dashboard etc. can communicate at real time with very little delay. After studying its working inside the CAN Bus, we had started exploiting its security with various attacks on it. Thus conveying the security loop holes in CAN Bus and confirming that it is vulnerable to attack. We also propose an encryption and authentication protocol as a preventive measure for CAN Bus and study its effects. We had detected malicious attacks using Deep Neural Network, Long Short Term Memory and other such models. as a state of an art solution.Thus prevention and detection of the anomalous data packets in CAN Bus system is the scope of our project using state of the art solutions.

1.6 Organization of the Project

The rest of the project is organized as follows. Chapter 2 dwells into the research the literature survey that we conducted while researching on our project. It made us aware of a large number of developments with an aim to secure CAN Bus.Chapter 3 discusses the importance of CAN Bus as Technology in in-vehicle networks,the vulnerabilities of CAN Bus system and types of attacks possible in CAN Bus system. Then, chapter 4 describes the threat model and identify security requirements and the work that has been previously done in securing the CAN Bus. Following that, we introduce different levels of security and implement an encryption and authentication protocol for in-vehicle networks. Thus covering our first goal of the project as prevention of CAN Bus system from getting exploited. In chapter 5, we had applied various Machine Learning Models and Data Mining Techniques to detect Anomaly Data Packets in the Data Stream of CAN Packets and analysing and comparing each Model's' results. Later in chapter 6, we presented the conclusion of our endeavour towards the security in CAN Bus and identify future work towards enhancing the security of in-vehicle networks and provide a conclusion for the thesis.

Chapter 2

Literature Work

In this Chapter we will discuss about the research work that has been done in the field of in-vehicular communication. Various techniques have been proposed to ensure security and reliable communication. Let us try to understand each of them.

2.1 EVITA Project

To the best of my knowledge, the largest project that aimed at securing in-vehicle communication networks is the EVITA project. EVITA stands for E-safety Vehicle Intrusion protected Applications. The project took place in the period from July 2008 to December 2011. The project was co-funded by the European Commission. The objectives of the project were to design, to verify, and to prototype an architecture for automotive on-board networks where security-relevant components are protected against tampering and sensitive data are protected against compromise. Thus, the goal of the project was to provide a basis for the secure deployment of electronic safety aids based on vehicle-to-vehicle and vehicle-to-infrastructure communications. The target was to complement other e-safety related projects that focus on protecting the communication of vehicles with the outside by focusing on on-board network protection.

This section discusses some of the work done within that project.The EVITA project has inferred the following set of security requirements and related functional requirements in order to satisfy the stated security objectives:

1. Integrity / authenticity of e-safety related data: Actions depending on critical information should be decided based on assurances about integrity and authenticity in terms of origin, content, and time. Forgery of, tampering with, or replay of such information should at least be detectable.

2. Integrity / authenticity of ECU / firmware installation / configuration: Any replacement or addition of an ECU and/or its firmware or configuration to the vehicle shall be authentic in terms of origin, content, and time. In particular, the upload of new security algorithms, security credentials, or authorizations should be protected.

3. Secure execution environment: Compromises to ECUs should not result in system wide attacks, primarily with regard to e-safety applications. Successful ECU attacks should have limited consequences on separate and/or more trusted zones of the platform.

4. Vehicular access control: Access to vehicular data and functions should be controlled (e.g. for diagnosis, resources, etc.)

5. Trusted on-board platform: The integrity and authenticity of operated software shall be ensured. An altered platform might be prevented from running in an untrusted configuration (e.g. via comparison with a trusted reference) if so required.

6. Secure in-vehicle data storage: Applications should be able to use functionality in order to ensure access control to as well as the integrity, freshness and confidentiality of data stored within a vehicle, especially for personal information and security credentials.

7. Confidentiality of certain on-board and external communication: The confidentiality of existing software / firmware as well as updates and security credentials shall be ensured. Some applications might additionally require that part of the traffic they receive or send internally or externally should remain confidential.

8. Privacy: A privacy policy shall be enforceable on personal data stored within a vehicle or contained in messages sent from a vehicle to the outside. For example, some applications should limit the ability to link sent messages

9. Interference of security functionality: The operation of security services must not negatively affect the availability of bus systems, CPUs, RAM, or of the radio medium.

2.2 The TESLA protocol and its variations

TESLA protocol was proposed as a new protocol for multicast authentication. The main idea behind it is to achieve asymmetric properties using delayed disclosure of keys. However, this leads to delayed authentication. This delayed authentication has two main drawbacks. First, the receiver need to have storage for some unauthenticated messages till their key is disclosed. This increases the effect of DoS attack where an attacker may good the receiver with many wrong messages. The second point is that this makes TESLA not suitable for real-time systems. The protocol was published later as RFC 21. In order to achieve immediate authentication, a modification to TESLA

protocol was proposed. In the proposal, messages do not need to be queued at the receiver waiting to be authorized. Instead, they are queued at the sender putting the hash code of each message in the preceding one. This solved the problem of DoS attack. Later on, TESLA was developed in order to be used in wireless sensor networks. The main aspects of such systems is the lack of processing capabilities, low memory for storing code, small RAM and running on battery power devices. TESLA was also modified to be used for Secure Real-time Transport Protocol (SRTP). Recently, another modification was made for TESLA introducing TESLA++ . TESLA++ was developed to be used in Vehicular Ad-hoc Networks (VANETs). It modifies TESLA in a way that makes it resilient to memory-based DoS attacks.

2.3 Security Module

In [3], Marko Wolf et al introduced the idea of using a security module for providing different cryptographic functionalities to vehicles. Both centralized and distributed approaches are discussed. In the centralized approach, a single security module is used for providing security to several ECUs within the car. On the other hand, the distributed approach is based on attaching a security module to each ECU that needs protection. From implementation point of view, both software and hardware can be used for realizing such security module. In the case of centralized approach, the hardware implementation is more suitable, while in the case of distributed approach, the software implementation is more practical.

2.4 Key Distribution Protocol for CAN

In [4], a key distribution protocol has been introduced for securing in-vehicle communications over CAN bus. Due to the embedded constraints, symmetric key cryptography was used. In each ECU, a hardware security module (HSM) was attached. The HSM implements some cryptographic primitives as well as securing the key storage. Moreover, it stores metadata for the keys (called user-flags). For example, a key may be tagged for signing at a node while it is tagged for verifying at another node. In exchange of shared keys is done through a logical entity called "Key Master"

(KM). Each node, has two keys to communicate with the KM; one for authentication and the other for transporting generated keys. To establish a secure communication channel between an ECU and other ECUs, the following steps are followed:

1. The ECU generates a pair of keys; one for generation and the other for verification.
2. The ECU sends the verification key encrypted to the KM.
3. The KM forwards the key encrypted to all other ECUs.

Those generated session keys are made valid for a limited time only. It is valid for one drive cycle for at most 48 hours. In order to be able to send messages larger than the standard CAN payload of 8 bytes, segmentation of data had to be used.

2.5 Conclusion

EVITA project has made a considerable progress in securing in-vehicular communications. To achieve this, the concept of adding a hardware security module (HSM) was introduced. The main disadvantage of using HSM is that it infers additional hardware cost to be added to the cost of manufactured vehicles.

Chapter 3

Overview of CAN Bus

A vehicle bus to allow micro-controllers, ECUs and devices to communicate directly with each other. It is designed for multiplex electrical wiring within automobiles. Control Area Network (CAN) is used in many in-vehicle system where minimizing the need of wiring system is needed where many components are communicating and exchanging information with each other in real rime.It replaces the traditional architecture of wiring with a new architecture consisting of only one bus, the CAN bus.

3.1 Introduction to CAN Bus

CAN is a multi-master serial bus standard for connecting Electronic Control Units [ECUs] also known as nodes. Two or more nodes are required on the CAN network to communicate. The complexity of the node can range from a simple I/O device up to an embedded computer with a CAN interface and sophisticated software. The node may also be a gateway allowing a standard computer to communicate over a USB or Ethernet port to the devices on a CAN network.

ISO 11898-2, also called high speed CAN, uses a linear bus terminated at each end with 120 Ω resistors.

ISO 11898-3, also called low speed or fault tolerant CAN, uses a linear bus, star bus or multiple star buses connected by a linear bus and is terminated at each node by a fraction of the overall termination resistance. The overall termination resistance should be about 100 Ω, but not less than 100 Ω.

3.1.1 CAN Bus Frame Format

The CAN message data frame format takes the following form as shown in figure 3.1. The length of the Data Field (which carries the actual payload of the message) varies from 1 byte to a maximum of 8 bytes. The cyclic redundancy check(CRC) Field stores the error detecting code of the contents in the Data Field.

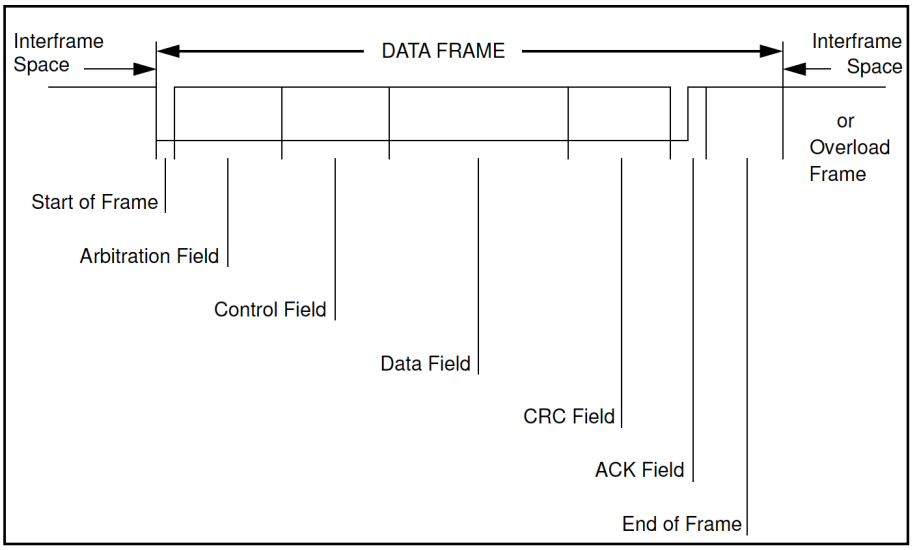

Figure 3.1: CAN Bus Frame Format [5]

Regarding the “Arbitration Field”, it takes one of two forms as follows.

1. Standard Format: The arbitration field consists of 11 bits representing the message identifier in addition to the RTR (Remote Transmission Request) bit. This is shown in figure 3.2.

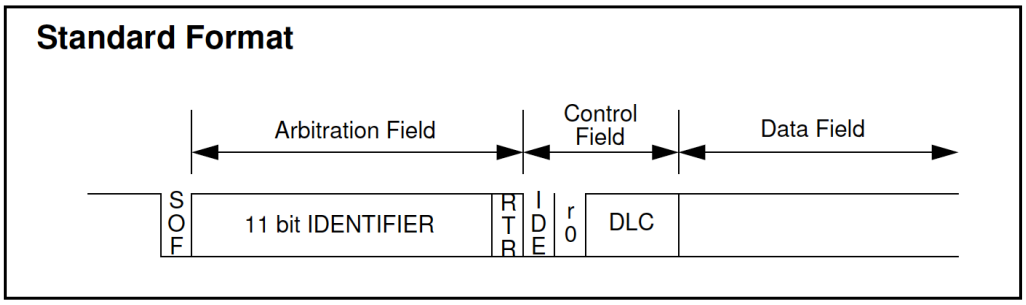

Figure 3.2: CAN Bus Standard Frame Format [6]

2. Extended Format: The arbitration field consists of 29 bits representing the identifier in addition to SRR (Substitute Remote Request), IDE(Identifier Extension) and RTR (Remote Transmission Request) bits. This is shown in figure 3.3.

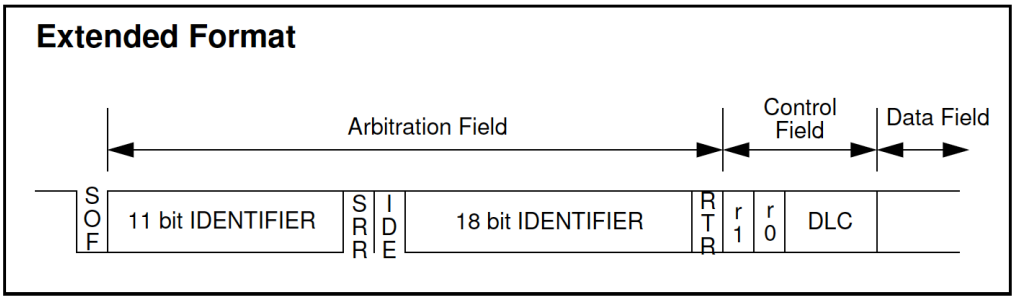

Figure 3.3: CAN Bus Extended Frame Format [6]

3.1.2 Arbitration working in CAN Bus

Many ECU nodes in the network try to acquire the CAN Bus network to send the data to other desired ECUs. For acquiring the CAN Bus, ECUs work together in arbitration mode such that only one ECU whose CAN ID‘s priority is higher than others competing CAN ID will win. Figure 3.4 shows the arbitration mode of 3 nodes competing to get the CAN Bus system.

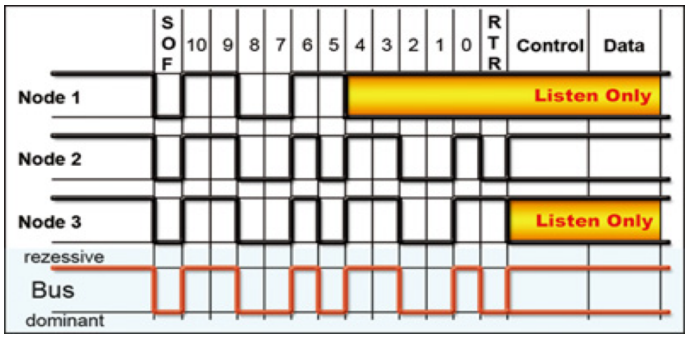

Figure 3.4: Arbitration in CAN Bus [3]

3.2 Reasons of Vulnerability

CAN Bus exists with a large number of ”hotspots” for hackers- a major driving force behind our project.

3.2.1 Inherent Weaknesses

It has been highlighted that some of the weaknesses of vehicles security arise from the widely used CAN bus itself such as:

1. Broadcast Nature: Since the bus has a broadcast nature, then any node connected to the bus can listen to all data exchanged.

2. Fragility to DoS: Based on the arbitration scheme of CAN, any node can put the bus in a dominant state preventing other nodes from sending any messages.

3. Absence of authentication: The CAN message itself does not contain any authentication information about its sender. Thus, it is possible for any attacker who connects to the bus to send messages using the identity of any trusted node.

Besides the direct weaknesses of CAN bus listed above, there are also access control weaknesses due to the current used protocols.

1. Reflashing: In order to protect an ECU from being reflashed by an unauthorized party, a challenge-response protocol is used. By design a service shop can do such authentication in order to upgrade the software of the ECU.

2. Testing: For diagnostics, there exists a service that allows the tester in the workshop to control some vehicle parts directly (skipping the user input). Such service is protected by a challenge-response pair.

The main drawback in using this protocol, is that for each ECU, this challenge-response pair is fixed. This is to avoid having the algorithm being run inside the ECU. With limited length of key/seed pairs, it could be possible to determine the key in about 7 days!

3.2.2 Weaknesses due to deviation from standards

In addition to the weaknesses mentioned above, there are also some weaknesses that arise from deviating from security standards and regulations. Disabling communications: It is specified that a car shall reject any command that disables communication as long as the car is moving. However, this rule is not always respected. Thus an attacker, can disable communications while the car is moving leading to severe accidents.

1. Reflashing while driving: It is also mandated that reflashing the code of an ECU is not allowed while the car is moving. However, it was possible to reflash some ECUs while the car was moving.

2. Reflashing the telematics unit: It has been found that the telematics unit of the car under test had a single seed-key pair that is used for all ECUs deployed in all cars. Moreover, it is also possible to reflash the ECU even if the wrong key is entered.

3. Unrestricted access to keys: In order not to run the challenge-response protocol in each ECU, manufacturers store the keys inside the ECUs themselves. The area in memory where those keys are stored should have restricted access. However, it was possible to read those keys using "memory read" service.

4. Same keys used for both Reflashing and DeviceControl: It has been also discovered that some ECUs use the same key for authenticating both DeviceControl and reflashing. As a result, when a key is known, the access to both services is open.

5. Reflashing gateways from low-speed network: Gateways are used to bridge different CAN networks. Networks are of two types; high speed and low speed. High speed are considered more trusted than low speed. As a result, standards specify that a gateway can be reflashed only from the high speed network. However, it has been discovered that it can be also reflashed from the low speed network.

With all these vulnerabilities, there is a need for several studies to secure in-vehicle networks.

3.3 Motivation behind attacks

There could be many reasons for which someone might want to perform any of the attacks discussed above.

1. Theft: The main motivation behind attacking cars is to steal them. As mentioned above, it could be possible for the adversary to unlock the door locks under certain trigger conditions. Moreover, it is possible to track the location of a car by sending its GPS location using 3G connection. Hence, a thieve could locate the victim's car and log into it, bypass the immobilizer, start the engine and take it away. This theft mechanism has been realized in several publications.

2. Surveillance: The microphone used for hands-free calling can be used for recording conversations that are being held inside the car. Those recordings can be sent either in real-time or in a later time through a compromised 3G connection. Moreover, the GPS location information of a car can be also tracked in real-time. As a result, if a car is compromised, it would be possible to track it and all the conversations that are being held inside.

3. Kidnapping: If the adversary can display false track information on the head unit, then he could direct the victim towards an alternative road where he get kidnapped. It is also possible to play threatening messages on the speakers of the car together with locking doors and doing any irritating effects like activating the horn, ashing the internal light, etc.

4. Terrorist attacks: Since the level of attacks can cover a large number of cars, it would be possible for terrorists to do disasters by gaining control over a large numbers of a street in a certain city. For example, the attacker may disable brakes of all controlled car simultaneously

causing major accidents. The attacker may also disable the wipers during a day with heavy rain or snow, thus causing many accidents.

5. Assassination: A famous way of assassination that was used tens of years ago is to disable the brakes of a car. Unfortunately, this type of attack is still valid if the attacker can control the brakes of the car. If the steering of the car is done by wire, then it would be also possible to disable the steering wheel causing severe accidents. The difference between such attacks and the older ones is that old ways always left a trace while it is possible to hide traces for modern attacks. This can be realized if the attack is held by downloading some code in the RAM and then reset the ECU after the accident to remove any trace.

3.4 Types of Attacks

This section covers the types of Attacks that were performed on CAN Bus system.

3.4.1 Denial of Service (DoS)

In computing, a denial-of-service attack (DoS attack) is a cyber-attack where the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet. Denial of service is typically accomplished by flooding the targeted machine or resource with superfluous requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled. Similarly in CAN BUS network system, DoS attack was performed by different ECUs to target ECU so that the target receiver ECU gets overwhelmed by flooding the CAN packets unsolicited, thus malfunctioning the ECU.

3.4.2 Replay Attack

A replay attack (also known as playback attack) is a form of network attack in which a valid data transmission is maliciously or fraudulently repeated or delayed. This is carried out either by the originator or by an adversary who intercepts the data and retransmits it, possibly as part of a masquerade attack by IP packet substitution.

Another way of describing such an attack is: “an attack on a security protocol using replay of messages from a different context into the intended (or original and expected) context, thereby fooling the honest participant(s) into thinking they have successfully completed the protocol run.”

In CAN BUS, compromised ECU or attacker getting an access to the CANBUS masquerades itself and conducted Replay Attack on the target ECU by sending the sniffed packet again in later time frame.This has given us the evidence that CAN module of ECU can be compromised.

3.4.3 Injection/Diagnostics Attack

Since CAN Bus network is visible to all ECUs or controllers who are on the network they can send CAN data message to any target ECU concerned or can read the data if it concerns itself.This makes

it vulnerable for the attacker to inject any CAN data message to break any ECU to malfunction. Thus the concept of trusted network in CAN Bus is not adhered.

The biggest problem with CAN message injection is that, while attackers can inject arbitrary messages onto the bus, the original sender of the message (i.e. the legitimate ECU) is still sending legitimate messages. For all non-diagnostic messages, ECUs typically broadcast messages at a fixed interval. Even if the data values have not changed, ECUs will continuously send messages. For example, there is not a message that suddenly 'appears' to make the headlamps turn on. Instead a message is sent at a constant rate that changes data values to affect the state of the headlamps.

3.5 Hardware Specifications

We have purchased these following modules so that we can sniff live CAN Bus Stream packets directly from the car. Cars like I10 and Honda City were used to collect CAN Bus Messages with 3 distinct CAN IDs found in I10 and 38 distinct CAN IDs found in Honda City. Figure 3.5 shows the connection of two CAN Bus shield along with Arduino UNO. The Hardware components used by us in this project are as follows:

1. Arduino UNO R3
   Environment for CAN Bus data.

2. MCP2515
   Stand-Alone CAN Controller With SPITM Interface.

3. MCP2551
   High-Speed CAN Transceiver.

4. OBD to DB9 Cable Wire
   Cable to interface Arduino with cars.

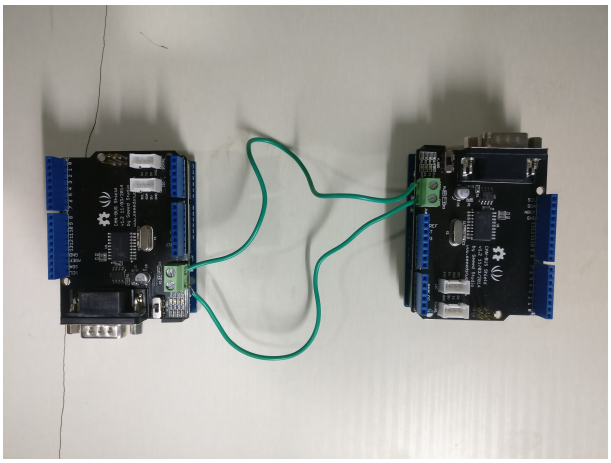

Figure 3.5: CAN BUS Module Mounted on Arduino Uno

Chapter 4

Prevention of anomalous packets on CAN Bus

This chapter showcases the security in CAN Bus from being compromised by attacker by preventing anomalous packets to malfunction the in-vehicle network system.

4.1 Security Levels and Cryptographic Protocol

4.1.1 Threat Model

In-vehicle networks consist of various ECUs that are interconnected using communication buses. There are many types of communications buses used like CAN, LIN, FlexRay and MOST. CAN is the most widely used type of buses. As discussed earlier, CAN messages do not contain any source or destination addresses. Also, they do not provide any means of encryption and authentication. Hence, any adversary node that succeeds to have access to the bus can listen to any transmitted message. Moreover, it can inject malicious messages into the network. In the following sections, we will discuss a multilevel approach for securing in-vehicle networks. Then, we propose a new encryption and authentication protocol that can be deployed inside in-vehicle networks. The protocol is designed mainly to be used inside CAN networks. However, it can be also modified to be used in other communication bus types.

4.1.2 Levels of Protection

In order to protect vehicles from various attacks, we recommend three levels of protection:

1. Level 1: Prevent ECUs from being compromised
   The first level of protection is to protect every ECU inside the vehicle from being compromised. There are many ways by which an ECU can be compromised. Here is a list of some of them:

   (a) Flash Bootloader
   An ECU can be compromised by replacing the flashed software by another malicious software using the flash bootloader. Typically, the flash bootloader is a piece of software that facilitates the update of the software flashed on an ECU. It is used by car

manufacturers to update the software of ECUs when necessary. Basically, accessing the bootloader is secured by challenge-response pair. To protect the ECU, the value of the key used in the authentication algorithm shall be unique for each ECU. Also, the length of the key used should be large enough in order to be protected against brute-force attacks. The value of the key stored in the flash memory of the microcontroller shall be secured against malicious reading. In order to protect the ECU from brute-force attacks, if the wrong response is entered for few times, the ECU shall be locked and cannot enter bootloader mode.

(b) Flasher
In this case, the adversary has direct physical access to the ECU. The protection against such attack can be done by censoring the flash memory during the production. By that way, the bootloader is the only entity that is granted the access to ash the ECU. It is worth noting that this type of attack is special because the attacker who can connect a flasher to the ECU can replace the microcontroller itself or even the whole ECU rather than replacing the software.

(c) Exploiting vulnerabilities
When the software of the ECU contains some vulnerabilities, they can be used by an attacker to insert his malicious code. Several types of vulnerabilities exist. Among these are the buffer overflow. The way of protection for such attacks is to perform static analysis to the software to ensure that the software is free from such vulnerabilities.

2. Level 2: Protect the vehicle's internal network against a compromised ECU
In the second level of protection, we assume that at least one ECU was compromised by an adversary. The compromised ECU can be one of two cases:

   (a) Case A: An ECU which is part of the vehicle's internal network is compromised (using any of the ways mentioned in Level 1 protection).

   (b) Case B: Any malicious node that is connected to the bus by the adversary.

   The goal of protection in this level is to prevent those compromised ECUs from harming the vehicle's internal communication network. The compromised ECU can harm the vehicle's network by doing any of the following actions:

   (a) Send a malicious message on behalf of another node: This type of attack arises from the nature of the CAN bus since it does not specify methods of source authentication. As a result, the compromised ECU may send false messages on behalf of other ECUs. This can be protected by using a source authentication protocol in the communication in the CAN bus.

   (b) Send a malicious message that it normally sends: Unfortunately, this type of attack cannot be protected after the ECU is already compromised. This is because whatever cryptographic methods used, the compromised software can still use this secured communication stack to send messages containing false data.

   (c) Perform a denial of service attack on the CAN bus: This can be protected by using something like the bus guardian of the FlexRay protocol.

3. Level 3: Detect the compromise of an ECU
   As a 3rd level of protection, it should be possible to detect the compromise of any node of the vehicle's internal network. The process of checking ECUs can either be done regularly by the vehicle or at least be done during regular service of the vehicle. The proposed way is to verify the checksum of the contents of each ECU. This can be the responsibility of a new Security ECU or the body controller. When the software of any ECU is updated, then the new checksum must be communicated to the Security ECU.

4.1.3 Security requirements

Levels 1 and 3 can be viewed as recommendations that can be adopted directly by vehicles' manufacturers. However, level 2 needs some research. We will focus on solving the problems mentioned above in Level 2 and later focus on Level 3. In this case, it is required to protect the vehicle's internal network from a compromised node. We will focus on the two cases A and B:

1. Case A: The threat model of this case is as follows. The CAN bus has some ECUs connected to it. One of these ECUs was previously compromised by an adversary. Figure 4.1 demonstrates this case.

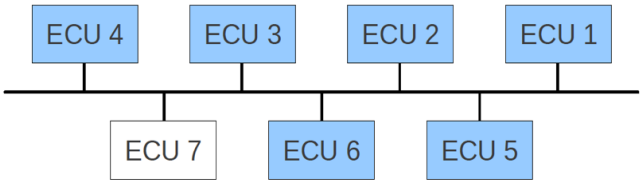

Figure 4.1: Case A: Existing ECU is compromised [18]

2. Case B: The threat model of this case is as follows. The CAN bus has some ECUs connected to it. All ECUs are communicating correctly. The adversary node is attached to the CAN bus at some point of time. According to the CAN bus specification, the attached node can listen to all exchanged CAN messages. Also, it has the ability to send any CAN message on behalf of any other node. Figure 4.2 demonstrates this case.

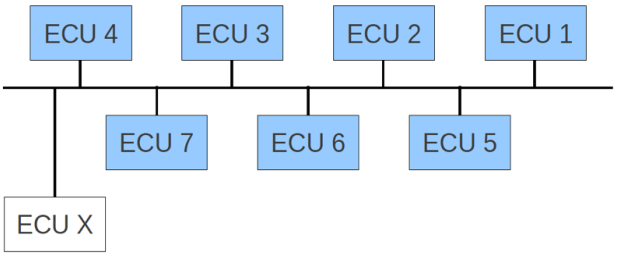

Figure 4.2: Case B: Adversary node connected to the bus [18]

For both cases, in order to protect the network against such attacks, message encryption and source authentication is required. However, the CAN bus protocol does not specify any means of encryption and authentication. As a result, it is required to implement a higher level encryption and authentication protocol that can be adopted in automotive CAN networks. The following requirements are needed when designing the encryption protocol.

1. Each ECU would be assigned the same 128-bit key.
2. Each time the ECU sends a message to another ECU, it encrypts it using this key.
3. On receiving this key, each ECU will decrypt the received packet and interpret the data.
4. The protocol shall add a small communication overhead. Since AES requires 16 bytes of data as block size, 2 packets of data need to be encrypted at once or we need to pad a single packet with 8 bytes of 0s. This is because each CAN message is 8 bytes in length.

The following requirements are needed when designing the authentication protocol.

1. The message shall contain an evidence that can be generated only by its trusted sender.
2. The receiver shall be able to verify that evidence.
3. The receiver shall not be able to re-transmit the message masquerading the trusted sender.
4. The protocol shall add a small communication overhead. The payload of any CAN message is already too small (8 bytes by maximum). For the current networks, those 8 bytes are highly utilized. Thus, the smaller the overhead used, the easier to deploy the authentication protocol in the currently designed networks with minimum reformatting of messages.
5. The protocol shall not require either heavy computation or high memory consumption. This is because the currently produced ECUs use microcontrollers with limited resources.

4.2 The Encryption Process

This section describes the encryption process used in our project.

4.2.1 Basics of Encryption

Encryption is the process of converting data to an unrecognizable or ”encrypted” form. It is commonly used to protect sensitive information so that only authorized parties can view it. This includes files and storage devices, as well as data transferred over wireless networks and the Internet. There are many different types of encryption algorithms, but some of the most common ones include AES (Advanced Encryption Standard), DES (Data Encryption Standard), Blowfish, RSA, and DSA (Digital Signature Algorithm). While most encryption methods are sufficient for securing your personal

data, if security is extremely important, it is best to use a modern algorithm like AES with 256-bit encryption. AES is based on a design principle known as a substitution-permutation network, a

combination of both substitution and permutation, and is fast in both software and hardware. Unlike its predecessor DES, AES does not use a Feistel network. AES is a variant of Rijndael which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, the Rijndael specification per se is specified with block and key sizes that may be any multiple of 32 bits, both with a minimum of 128 and a maximum of 256 bits. AES operates on a 4 by 4 column-major order

matrix of bytes, termed the state, although some versions of Rijndael have a larger block size and have additional columns in the state. Most AES calculations are done in a particular field. Another advantage of AES would be the variations in key size provided by NIST. Key size is usually a function of the number of devices on the network.

4.2.2 Strength of AES key

For symmetric algorithms (like AES or DES or RC4 – but not for RSA or ECDSA), a key is a sequence of bits, such that any sequence of bits of the same size is a potential key. So longer keys mean more possible keys. Exhaustive search is about trying all possible keys until a match is found.

It is an absolute limit to the strength of an algorithm: exhaustive search is always applicable, there is nothing an algorithm can do against it. The only defence against exhaustive search is to expand the space of possible keys, i.e. have longer keys. Fortunately, this is very effective: the largest exhaustive search experiment which was ever reported to have been performed was for a 64-bit key (a search for a 72-bit key is underway, but not completed yet). Traditionally we use ”80 bits” as the practical limit, but technology improvement may force us to increase that. Every additional bit doubles the hardness of exhaustive search. AES morphs with the key size: with a longer key, there

are more rounds. This is not about exhaustive key search; this is because a longer key is a promise of ”higher academic strength” (”academic” means ”we assume that the attacker can perform 300-bit exhaustive searches”). In an academic way, AES needs exhaustive search to be the best attack (i.e. attacks on the algorithm internal structure must be slower). A 256-bit key raises the bar: the non-exhaustive-search attacks must not be more effective work factor. Hence the extra rounds. In practice, the extra rounds do not buy you more security (just like the larger key does not make the algorithm really less breakable than the already unbreakable AES-128); what the extra rounds do is that they make encryption 40 percent slower.

4.2.3 Selection of the encryption function

After a careful evaluation of the encryption speeds of various ciphers and focussing only on block ciphers, AES proves to provide the best performance. The following graph, labelled as Fig 4.3, describes the theoretical measured values for each cipher suite.

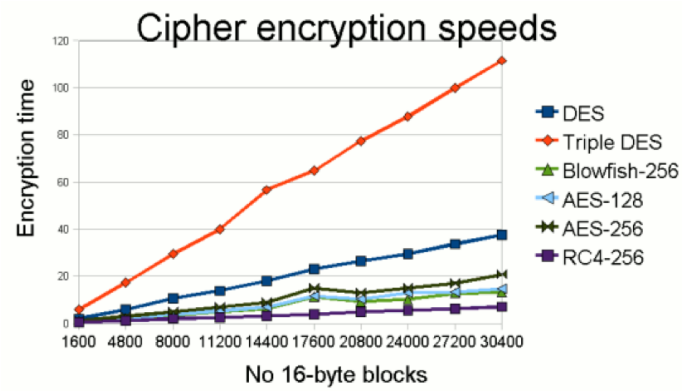

Figure 4.3: Comparison of cipher suites [1]

4.2.4 Strength of AES

Until May 2009, the only successful published attacks against the full AES were side-channel attacks on some specific implementations. The National Security Agency (NSA) reviewed all the AES finalists, including Rijndael, and stated that all of them were secure enough for U.S. Government non-classified data. In June 2003, the U.S. Government announced that AES could be used to protect classified information: The design and strength of all key lengths of the AES algorithm (i.e., 128, 192 and 256) are sufficient to protect classified information up to the SECRET level. TOP SECRET information will require use of either the 192 or 256 key lengths. The implementation of AES in products intended to protect national security systems and/or information must be reviewed and certified by NSA prior to their acquisition and use. AES has 10 rounds for 128-bit keys, 12

rounds for 192-bit keys, and 14 rounds for 256-bit keys. By 2006, the best known attacks were on 7 rounds for 128-bit keys, 8 rounds for 192-bit keys, and 9 rounds for 256-bit keys.

4.3 The Authentication Process

This section describes the authentication process used in our project.

4.3.1 Basics of authentication

Before discussing multicast authentication, we introduce a brief discussion about unicast authentication. Message source authentication between a sender and a single receiver can be achieved by adding a message authentication code (MAC) to each transmitted message. The sender calculates the MAC based on the message contents and a shared secret. On the other side, the receiver uses the same shared secret to verify the MAC. This can be achieved using symmetric cryptography. When

there is more than one receiver - the case of multicast communication another way shall be used. The shared secret cannot be shared between more than two entities. Therefore, asymmetric cryptography is needed. For each transmitted message, the sender creates a MAC using its private key. Various receivers use the sender public key to verify the MAC. However, asymmetric cryptography

is not recommended for our domain because it needs high computational capabilities. TESLA is

one of the most famous authentication protocols for multicast communications. However, if we try to adopt it in CAN networks we will and that we will need large communication overhead. This overhead is because each message shall contain the original data to be transmitted in addition to the MAC and a key. Also, the delay introduced by TESLA in unacceptable for in-vehicle networks as real-time systems.

4.3.2 Strength against replay attacks

Replaying a receiver request to the sender may allow the sender to do the channel setup based on an invalid nonce value. In this case, the receiver will not accept any of the sent parameters. Replaying sender responses is not possible because it depends initially on the value of the nonce and later on, it depends on the values of the magic numbers. In both cases, the receiver will reject the replayed value.

4.3.3 Strength of the HMAC key

The length of the HMAC Key is chosen to obtain HMAC security of 128 bits. The security of HMAC is divided into two parts; the security of the HMAC algorithm and the security of the HMAC value. The security of the HMAC algorithm is the minimum of the security of HMAC key and twice the length of the output of the used hash function. The latter parameter is 256 bits. Therefore, from this point of view, the length of HMAC key shall be set to 128 bits. On the other hand, the security of the HMAC value is bounded by the length of the HMAC output which is 256 bits. Therefore, it is sufficient for the length of the HMAC key to be 128 bits

4.3.4 Selection of hash function

As indicated above, HMAC requires the use of a cryptographic hash function. There are many available hash functions from which we can select the one to use for HMAC. The selection criteria depend on two main points:

1. The selected hash function shall have high pre-image resistance as well as a considerable collision resistance.

2. The selected hash function shall add minimum overhead from point of view of execution time and memory consumption (both RAM and ROM).

Initially, we will list the available hash functions and then select which one to use.

1. MD5: The first function to consider is MD5 which is deemed by RFC 1321. Its output consists of 128 bits. It is widely mainly used to check data integrity. However, many successful attacks have been discovered for both collision resistance (with a complexity of 220:96) and pre-image resistance (with a complexity of 2123:4).

2. SHA-1 and SHA-2: The Secure Hash Standard (SHS) specifies 5 hash functions that are approved by the National Institute of Standards and Technology (NIST). The functions are: SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512. The output length in bits of them is 160, 224, 256, 384 and 512 respectively.

3. SHA-3: In 2007, NIST announced a public competition to develop a new hash algorithm. The competition was NIST's response to advances in the cryptanalysis of hash algorithms. The winning algorithm will be named "SHA-3" for SHA 3.

Figure 4.4 summarizes the output size of different available hash functions together with the strength of each one of them. The values are obtained from [3].

Function	Output Size in bits	Collision Resistance Strength in bits	Preimage Resistance Strength in bits	2nd Preimage Resistance Strength in bits
MD5	128	20.96	123.4	N/A
SHA-1	160	< 80	160	105-160
SHA-224	224	112	224	201-224
SHA-256	256	128	256	201-256
SHA-384	384	192	384	384
SHA-512	512	256	512	394-512

Figure 4.4: Output size and Strength of Hash Functions

Regardless the chosen hash function to be used for HMAC, we need to truncate the output of HMAC into 16 bytes. According to NIST, the result of HMAC shall not be truncated to less than 8 bits. Since we chose 128-bits, then we conform with this recommendation. Also, NIST states that when truncation is to be done, then the leftmost bits shall be selected. Therefore, we choose to select the 128 leftmost bits of the HMAC output. Figure 4.5 represents the theoretical values of runtime of authentication functions.

Key Length / Hash Function	16 bits	32 bits	64 bits	80 bits	128 bits
MD5	150	152	154	155	158
SHA 1	286	288	290	291	294
SHA 224	543	544	546	546	549
SHA 256	543	544	546	547	550

Figure 4.5: Theoretical Values

Figure 4.6 shows the experimental results that we obtained.

Hash Function \ Key Length	16 bits	32 bits	64 bits	80 bits	128 bits
MD5	144	145	146	147	150
SHA 1	278	280	282	283	286
SHA 224	539	540	542	544	547
SHA 256	540	541	542	544	547

Figure 4.6: Experimental Values

4.3.5 Security of HMAC

The security strength of the HMAC algorithm is the minimum of the security strength of K and the value of 2L (i.e., security strength = min(security strength of K, 2L)). For example, if the security strength of K is 128 bits, and SHA-1 is used, then the security strength of the HMAC algorithm is 128 bits. The HMAC key K shall be generated with a security strength that meets or exceeds the desired security strength of the HMAC application, and the approved hash algorithm in the HMAC application shall have a message digest length of at least half of the desired security strength (in bits) of the HMAC application. For example, if the desired security strength of the HMAC application

is 256 bits, the HMAC key K shall be generated with a security strength of at least 256 bits, and an approved hash function with the message digest length of at least 256/2 (128) bits shall be used.

4.3.6 Security of HMAC values

The successful verification of a MacTag does not completely guarantee that the accompanying text is authentic; there is a slight chance that an adversary with no knowledge of the HMAC key, K, can present a (MacTag, text) pair that will pass the verification procedure. From the perspective of an adversary that does not know the HMAC key K (i.e., the adversary is not among the community of users that share the key), the security strength provided by a MacTag depends on its length. The length of a MacTag shall be sufficiently long to prevent false acceptance of forged data. For most applications, a length of 64 to 96 bits is sufficient. Shorter MacTags may also be acceptable if the rate of false acceptances does not create a significant impact for the application. For example, in video/audio stream applications, accepting one bad data package in 28 data packages may not create a huge impact for the application.

4.4 Software Architecture

4.4.1 Dynamic architecture

The real-time behaviour of the software is achieved using a simple rate-monotonic scheduler. The scheduler triggers the execution of different tasks according to their statically specified periods.

The scheduler is based on a hardware timer (from the microcontroller) providing a tick every 1 millisecond. The scheduling is chosen to be non-preemptive; that is when a task is running while another task becomes ready to run, then the execution of the first task continues first then the other task is run

4.4.2 Static Architecture

Initially, a simple CAN communication stack was implemented. Then, the protocol layer was added to it. Finally, the software architecture is as shown in Figure 4.7:

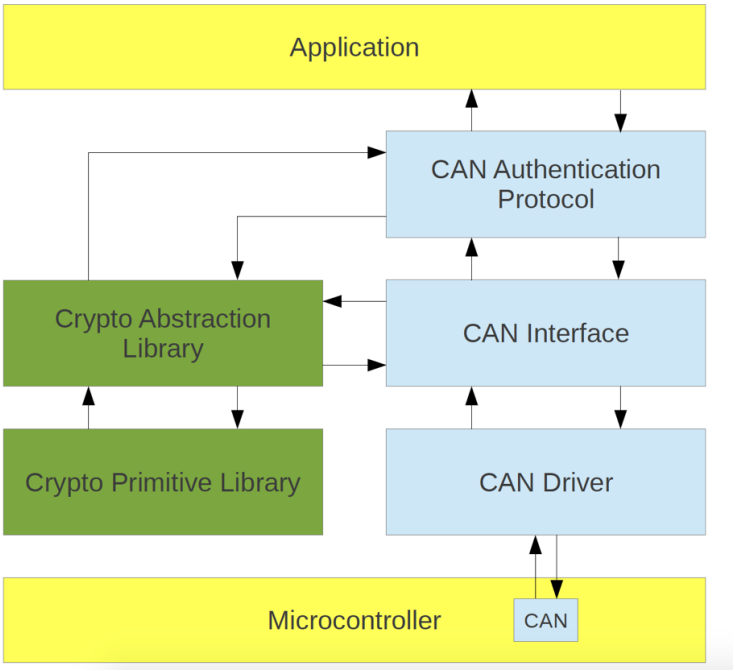

Figure 4.7: Static Architecture

As shown in the figure, the main components are:

1. CAN Driver: The CAN Driver is the component which interfaces with CAN peripheral of the microcontroller. It configures the baudrate of CAN and configures different message buffers. It provides APIs for setting and getting data to / from message buffers.

2. CAN Interface: The CAN Interface is the component that provide APIs to the upper layers in order to transmit/receive different CAN messages. It uses CryptoAbstraction Library to encrypt/decrypt messages.

3. CAN Authentication Protocol: The CAN Authentication Protocol is the main component that implements the proposed authentication protocol. It uses Crypto Abstraction Library to generate/verify magic number chain.

4. Crypto Abstraction Library: The Crypto Abstraction Library is the component responsible for abstracting the access to different cryptographic functions that are needed by the authentication protocol. It provides APIs responsible for performing various cryptographic functions such as encryption, decryption, one-way hash function transformation.

5. Crypto Primitive Library (CPL): This component implements cryptographic primitive functions like encryption, decryption, hash functions, HMAC, etc. It is actually, the OpenSSL library.

4.5 Optimizations in the project

This section dwells into the optimization that could be applied to our encryption and authentication schemes.

4.5.1 Encryption/Decryption Optimization

In general, the execution time of applying a stream cipher consists of two parts:

1. Generation of a pseudo-random key stream
2. Doing an XOR operation between the generated stream and the stream to be encrypted/decrypted.

Since the generated stream is the same for every encryption/decryption operation of the same key, then this byte stream can be cached in order to optimize the execution time. However, the cost of this optimization technique is that 10 bytes are needed from RAM in order to cache the generated pseudo-random stream. The obtained result for doing XOR operation only took 4 micro s.

4.5.2 Authentication Optimization

Regarding hashing, the execution time for SHA224 was 146 us. For HMAC, the execution time was initially long (before optimization). This step is needed to be performed at the receiver side for each received CAN message. According to the acceptance of how many messages that may be dropped, this time can be needed many times per message. In order to have a uniform load at the

sender side when refreshing the magic number chains, it is recommended that the sender generates the new chain while consuming the current chain. In other words, with every message that the sender transmits, it consumes an element of the current chain and at the same time, it generates a new element of the new chain. Using OpenSSL library (with optimization), the execution time in us of HMAC using different hash functions varying the key size is as shown in the table below. The execution times in micro-s for subsequent runs of HMAC has reduced times as shown in the Figure 4.8, in comparison with our previous calculations.

It is noticed that for the chosen key lengths, the length of the key does not affect much the execution time of HMAC. This is because for all key lengths that are below the input block size of the hash function, there is no much difference in calculations since the key is used directly without hashing. Changing the key length affects only the calculation of “ipad” and “opad”; which is done using a simple XOR operation. The initial results indicated that the time consumed for calculating HMAC

is too long. As a result, we needed to optimize the implementation of HMAC. In order to do that,

Hash Function \ Key Length	16 bits	32 bits	64 bits	80 bits	128 bits
MD5	86	86	86	86	87
SHA 1	154	154	154	154	154
SHA 224	285	286	285	285	285
SHA 256	285	285	286	285	285

Figure 4.8: Experimental Values for Optimized MAC

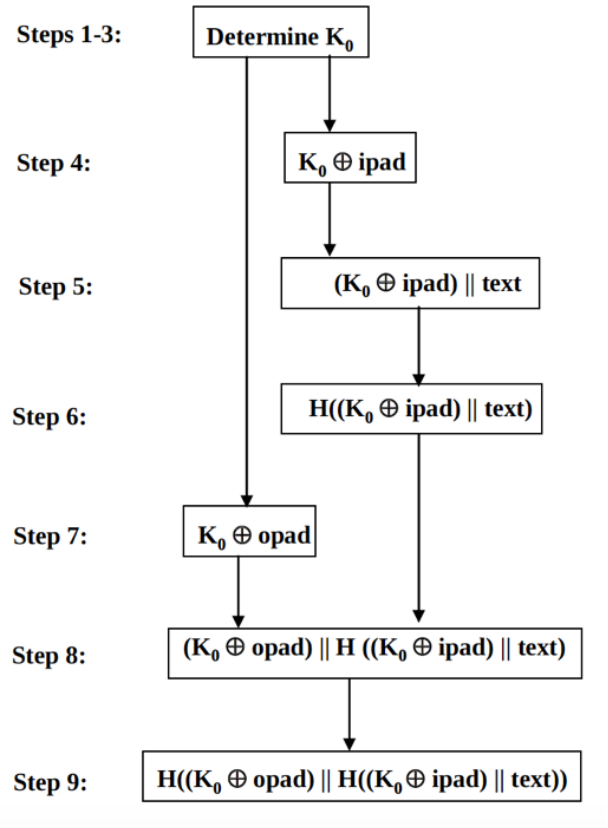

Figure 4.9: Steps of Optimized MAC

we first take an overview on the HMAC algorithm itself. The figure below describes the steps of the algorithm:

Defining the following:

1. K: The key to be used for HMAC.
2. B: The input block size of the used hash function.
3. L: The length of the output of the used hash function.
4. H: The hash function that is used to calculate the HMAC.
5. text: The input text to HMAC.

6. ipad: The value 0x36 repeated B times.
7. opad: The value 0x5C repeated B times.

Measurements showed that the most consuming part is the hash function. Generally, a single HMAC calculation requires from 2 to 3 hash operations. In our case, the length of the key is less than the block size of the used hash functions. Therefore, a single HMAC calculation requires only 2 hash operations. The time consumed in HMAC calculation can be optimized by doing the part of hash function on each of ipad and opad only once per used key.

4.6 Conclusion

The exposure of in-vehicle networks to the external world requires securing the communication inside these networks. However, security adds a cost represented by extra processing, memory, reformatting of messages and time delay - both at initialization and in runtime. At the same time, the small payload of CAN messages puts a limit on the strength of the security that could be added to the bus. Some protocols have been proposed before but their adoption was not easy because

they either needed modifications in the physical layer of the CAN or they added much overhead inside the CAN message that reduced the size of the useful payload inside the message. The main

goal of the work presented in the thesis is to add lightweight encryption and authentication to the CAN bus with the minimum impact on the currently deployed networks. The protocol shows good analysis results and proves that it is more practical than other published protocols and is low-cost as well. However, due to the small bandwidth available for exchanging authentication data, it is recommended to migrate in-vehicle communication networks to a higher bandwidth one. FlexRay with a payload up to 254 bytes per frame is a good alternative from security point of view. Also, One Pair Ethernet – as an emerging method - looks more promising.

Chapter 5

Detection of anomalies within CAN Bus data

In-vehicle intrusion detection systems are reactive security components which can monitor the vehicular networks in real-time and respond immediately if a potential threat has been detected. In the Internet world, typical intrusion detection systems like Snort (http://www.snort.org/) are based on signatures of known attacks. Signature-based detection requires regular updates of attack signatures but rewards the user with a high accuracy in detection and a low rate of false-positives. However, due to different reasons like the restriction to known attacks, the limited update possibilities in combination with the long lifetime of vehicles, the anomaly-based detection approach looks more promising for the automotive domain than the signature-based concept [14].

5.1 Data Generation

Our work is based on data obtained from Honda City car. We captured approximately 30 minutes of driving data, obtained with a USB-CAN bus interface connected to the in-car on-board diagnostics (OBDII) port. The bus transmits at 500 kbps. Thirty eight distinct IDs were observed (for convenience, sometimes we identify the IDs by the numerals 0 through 37 and not by their actual values). About 800 packets per second are transmitted while the car is running, with each ID transmitting at a consistent period varying from 10 ms to 1s; this amounted to approximately 1 million packets in our data set. Each ID uses only a subset of its 64 bits.

5.2 Synthesizing CAN Bus Anomalies

Unfortunately there is no publicly available repository of attack traffic, so we resort to synthesizing anomalies by modifying packet capture data. Creating actual attack effects on a car is time consuming and requires specialized knowledge, and the willingness to risk damage to the test vehicle and its surroundings [7]. Consequently most researchers resort to emulating or simulating effects characteristic of the attacks they are attempting to detect, e.g. switching message IDs to simulate a counterfeit transmitter [8], or inserting innocuous random packets into the live bus [9]. We have generated anomalies based on following techniques:

1. **Random Distribution:** There can be some unintentional tamper with an ECU which may lead to irrelevant data being sent on the CAN Bus. This kind of data is usually random due

to some fault in the ECU. Hence, this kind of anomaly is created with random distribution in the dataset.

2. **Bit-wise Probability Distribution:** The bits which usually do not change, would certainly would not be changed by the attacker. For each bit, the probability of it being 0 or 1 is calculated and with this probability distribution the 64 bits of each of the anomaly data packet is generated. This kind of anomaly data is usually seen when the attacker wants to manipulate control bits of the CAN packets.

3. **Byte-wise Probability Distribution:** Here, instead of each bit, each of the 8 bytes are generated in accordance with their respective probability in the original dataset. Bytes are generally individually altered to distort the continuous valued data and also the control data to some extent.

4. **Pairwise Probability Distribution:** From the data which we analysed, much of the continuous valued data is represented pairwise, eg. if the first byte of a packet which represents speed is say 0x02(2 in decimal) and the second byte is 0xD2(210 in decimal), then the actual speed of the car will be represented by 0x02D2(722 in decimal) or 7.22 km/hr. In such cases, it is probable that the attacker will modify data of a packet pairwise. Hence, we created anomaly which generates 2 bytes based on their respective probability distribution whereas the remaining bytes are kept the same. This type of anomaly is also generated bitwise by the probability distribution of individual bits with pairwise anomaly for 16 bits keeping the remaining 48 bits unchanged.

5. **Pairwise Inverted Probability Distribution:** This technique used for anomaly generation is similar to the one described previously, just the probability distribution function of each byte is inverted in this scheme. That is, the values of a byte which occur usually, along with the remaining bytes won't be kept around the same value because the attacker won't be able to attack the CAN Bus system since although the bytes are changed, the data cannot attack the system. The attacker would manipulate the pairwise continuous valued bytes to unsuitable values which would affect the system. Hence, anomaly data is generated by inverting the probability of the values of particular byte.

5.3 Apriori

Car is made up of number of ECUs that work by interacting with each other through CAN Bus. It has been observed that the packets in CAN Bus that come from some particular ECUs have sequential relation between them. From our analysis we have observed this relation among different IDs in CAN Bus. That is, if one ECU sends some data then it triggers some other ECU to send some data either in response to the data from the former ECU or to just be syncronized with that ECU. This happens because of the interaction taking place between them at different point in time. To exploit this property we have to find packets that occur in groups. Apriori calculates the probability of an item being present in a frequent itemset, given that another item or items is present. By using this concept we had tried to find the probability of occurrence of a packet coming from ID A given that packet from B has arrived at some point in time before A.

5.3.1 How the Model works

An association mining problem can be decomposed into two subproblems:

1. Find all combinations of items in a set of transactions that occur with a specified minimum frequency. These combinations are called frequent itemsets.

2. Calculate rules that express the probable co-occurrence of items within frequent itemsets.

Apriori calculates the probability of an item being present in a frequent itemset, given that another item or items is present.

Association rule mining is not recommended for finding associations involving rare events in problem domains with a large number of items. Apriori discovers patterns with frequency above the minimum support threshold. Therefore, in order to find associations involving rare events, the algorithm must run with very low minimum support values. However, doing so could potentially explode the number of enumerated itemsets, especially in cases with a large number of items. This could increase the execution time significantly. Classification or anomaly detection may be more suitable for discovering rare events when the data has a high number of attributes.

5.3.1.1 Association Rules

The Apriori algorithm calculates rules that express probabilistic relationships between items in frequent itemsets For example, a rule derived from frequent itemsets containing A, B, and C might state that if A and B are included in a transaction, then C is likely to also be included. An association rule states that an item or group of items implies the presence of another item with some probability. Unlike decision tree rules, which predict a target, association rules simply express correlation.

Association rules analysis is a technique to uncover how items are associated to each other. There are three common ways to measure association.

5.3.1.2 Antecedent and Consequent

The IF component of an association rule is known as the antecedent. The THEN component is known as the consequent. The antecedent and the consequent are disjoint; they have no items in common.

5.3.1.3 Measure 1: Support

Support is an indication of how frequently the itemset appears in the dataset. The support of X with respect to T is defined as the proportion of transactions t in the dataset which contains the itemset X.

$$supp(X) = \frac{|t \in T; X \subseteq t|}{|T|} \tag{5.1}$$

5.3.1.4 Measure 2: Confidence

Rules have an associated confidence, which is the conditional probability that the consequent will occur given the occurrence of the antecedent.The confidence value of a rule, X $\Rightarrow$ Y , with respect to a set of transactions T, is the proportion of the transactions that contains X which also contains Y.

Confidence is defined as:

$$conf(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)} \tag{5.2}$$

5.3.1.5 Measure 3: Lift

Lift is a measure of the performance of a targeting model (association rule) at predicting or classifying cases as having an enhanced response (with respect to the population as a whole), measured against a random choice targeting model.The lift of a rule is defined as:

$$lift(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X) x supp(Y)} \tag{5.3}$$

or the ratio of the observed support to that expected if X and Y were independent.

5.3.2 Application of model in CAN Bus

CAN Bus packet IDs were arranged in an ascending order of their occurrence. Each continuous chunk of 10 IDs was considered to be as 1 item set. So we had exactly 10 items in one itemset. Now we applied apriori algorithm on this given itemset. Based on appropriate value of support and confidence, a number of association rules were generated between them. These rules determines the dependency among packets coming from a particular ECU and their sequential relevance. This gave us association rules that there were strong correlation between some IDs and helped us to have greater insight of the CAN Bus data set.

5.3.3 Results

The results of this approach are just the association rules. We are not able to test these rules against ECUs to detect the ones which are compromised, due to the hardware and software limitations which are incurred for testing this approach. Also, it is not suggestive to generate sequential testing data as we do not have the specifications of the ECUs and their interrelation with each other along with the time at with they send data. Even if we assume some specifications, the system would not be thoroughly tested as it would be biased to the association rules themselves.

Antecedent	Consequent	Support	Confidence	Lift
158,328,39	13C	0.0350	0.9668508	1.0438899
17C,1ED,221	1A6	0.0416	0.8421053	1.3248981
1A4,1A6,295	17C	0.0576	0.8299712	1.0580969
1ED,221	1A6	0.0524	0.8264984	1.3003436
158,1AB,1ED,255	39	0.0868	0.7876588	1.2284136
158,1ED,255	39	0.1282	0.7788578	1.2146878
221,39	1A6	0.0442	0.7594502	1.1948555

Table 5.1: Association Rules found in CAN BUS ID

For example, table 5.1 shows few of the many association rules formulated from the sequence of data generated by different ECUs. From these, some of the ECUs send data very frequently with not much important information in it. One example is the ECU with ID 13C which send data continuously and very frequently. Due to this we have a lot of associations which include these IDs and we ignored them during the analysis.

An example of frequent itemset that can be derived from many of the association rules is the correlation between IDs 221 and 1A6. It is found that in majority of the association rules which has ID 1A6 as consequent, has one of its many antecedent as ID 221. If we consider just the association rules with high confidence then all of the rules follow this pattern. Another example which has similar correlation is the pair of ID 255 as antecedent and ID 39 as consequent. With these correlations between IDs we can filter the data and check if the strong association rules hold even with the actual data sent by different ECUs on the CAN Bus. These association rules have to be coded in each ECU to detect any compromised ECU if there is any.

5.4 Entropy

Anomaly-based systems require a definition of the normal behavior of the system and recognize every deviation from this behavior as an attack. How exactly can the normal behavior of a vehicular network be defined? This is where an entropy-based approach yields promising results: In this, we have taken an approach to measure the entropy of an automotive network and use the result as a specification of the behavior for an intrusion detection system. Generally speaking, entropy is a measure of how much coincidence a given data-set contains. The more coincidence it comprises, the higher the entropy it contains. The entropy of a (finite) sequence of values can be measured without knowing the semantic meaning of all parts of this data. This was the motivation reason for us to take entropy as a measure to check for the deviation of an anomaly data against our normal data. Entropy can be calculated by representing the sequence in a binary form and specifying the portions of the measurement to consider. Consequently, entropy allows an abstract representation of the randomness of this data. In the context of network and Internet systems, the concept of entropy-based intrusion detection has been considered in various publications [18]. However, in this area of application the big disadvantage of the entropy approach is typically its high rate of false positives[19]. This is due to the fact that traffic in standard computer networks can vary a lot and is usually not limited in a strict manner. Neither the type nor the protocol of a packet are generally restricted, arbitrary data content and numerous options are allowed. Moreover, the timing behavior of message can vary strongly and service guarantees are usually at a best-effort level. All these aspects show that the amount of randomness in standard computer networks is fairly high.

Instead, traffic in automotive networks is much more restricted. Every packet in a vehicular CAN network and its possible data content is specified before. The identifier of a CAN message, which determines the destination(s) of a packet, also specifies which kind of payload this message is allowed to comprise in terms of different signals and values.

The permitted value range is defined as well as the length of every signal and the packet function. For many messages the exact timing behavior and frequencies are defined as well. Moreover,the number of additional options in CAN offer is very limited. Because of the clear and more restricted specification of traffic in vehicular networks we conclude that the amount of randomness in a standard computer network is much higher than in a vehicular network, or – said differently – the entropy of an automotive network in general is lower. This is the reason why we consider an adaption of the entropy-based approach for intrusion detection to be well-suited and yield a low rate of false-positives in the automotive domain. Most attacks on vehicular networks, like the injection of new packets, the manipulation of the payload of messages, or the omission of packets, e.g., by disconnecting a special ECU or launching a man-in-the-middle attack, will influence the traffic on the bus system and with it the normal system behavior in the network. This change in the normal behavior is reflected by the entropy in the automotive network because every attack that influences the network traffic changes the randomness on the automotive bus system. Therefore, the attacks lead to an increased entropy in the vehicular system, which in general contains a comparatively low entropy as explained before. This raise in entropy can thus be detected by an automotive intrusion detection system and used as an indicator of an Attack.

5.4.1 How the model works

Anomaly detection can be defined as the process of analyzing a set of data aiming at identifying patterns that differ significantly from the expected normal behavior. These patterns are defined as anomalies, and often translate to relevant and actionable information about security and safety characteristics of a monitored environment. Entropy-based anomaly detection algorithms characterize the normal behavior of a set of data based on their level of statistical $entropy$. The entropy H of a dataset comprising i different symbols is defined according to equation 1:

$$H(X) = \sum_{x \epsilon C_x} P(x) \log \frac{1}{P(x)} \tag{5.4}$$

where p (i) represents the probability of occurrence of the i th symbol. In information theory, entropy represents the amount of information conveyed in the dataset, expressed in bits. As an example, a dataset composed by only one symbol has H = 0 independently of its length, meaning that it conveys 0 bits of information. On the other hand, a dataset containing n independent and identically distributed symbols has H = $log_2(n)$. H also represents the expected amount of information conveyed by each message belonging to the dataset. The value of H is also used to measure the randomness of an information source. The use of entropy as a mean to describe the normal behavior of an information source relies on the following underlying assumptions:

1. the entropy of messages generated by the information source exhibits stable statistical characteristics;

2. relevant anomalies (that is: anomalies that should be detected by the algorithm) introduce significant deviations in the statistical characteristics of the entropy.

5.4.2 Application of model in CAN Bus

In the previous section we explained why an entropy-based detection method for attacks in the automotive domain is a promising and worthwhile approach. This section investigates the different dimensions which are relevant for the general concept of an information-theoretic intrusion detection approach for in-vehicle networks.

5.4.2.1 Data Abstraction Level

The first dimension which influences the results of an information-theoretic anomaly detection approach for in-vehicle networks is the level of data abstraction. In general, various information-theoretic measures can be applied to the input data. However, the selection of the parts of the input data which are relevant and suitable for the detection of anomalies is a non-trivial task. The interpretation of the input-data depends on the selected abstraction level. In our context, different classifiers exist which describe how the input-data is selected and interpreted. We identified three major abstraction levels for the data selection:

1. Binary Level: This level describes the situation that the communication flow in the automotive network is monitored as a binary stream consisting of ones and zeros. The boundaries

between messages and fields are not considered and the corresponding delimiters are neglected. At binary level,two different types of classifiers are possible: The bitwise classifier considers every single bit in the data stream of the network as a separate event class. The x-bitwise classifier considers combinations of x bits of the data stream as an event class.

2. Signal Level: In comparison to protocols like IP, where arbitrary content can be transferred, the payload in automotive bus systems, like CAN, is strictly specified. Generally, CAN is a shared medium which allows every participant to listen on the bus and select relevant messages based on the identifier. Typically, the payload of a CAN message comprises different CAN signals each of which carries the data for a specific function, option or configuration. For instance, different signals with status information, configuration data and sensor values of the Antilock-Braking-System (ABS) could be summarized in the data field of one CAN message. At signal level, the classifier generates one event class for every signal value of a message. It requires exact information about the signals which can be derived from the specification of the vehicular network. For CAN, this is specified in the CAN-Matrix.

3. Protocol Level: The highest data abstraction level we consider is based on the protocol specification of the automotive network. For CAN, the protocol defines 12 different fields for data frames of the base format [1]. The classifier at protocol level considers the content per field in the data frame and generates an event class for every field value.However, not all fields of the CAN protocol are suitable for an investigation, e.g., fields like the CRC check. Currently, our investigations focus on two fields: The identifier and the data field. The CAN identifier is a unique value which identifies the data and determines the receiver(s) of the message. The data field contains the payload of a message and can take up to 8 bytes[20].

5.4.2.2 Information-Theoretic Measures

Relative Entropy: The relative entropy can be used to measure the distance between two data sets. As we will see in Sect. V, the distance between individual items of two data sets can be determined by using the definition below without using the sigma sign to sum up all values of the sets. Definition: For two probability distributions p(x) and q(x),which are defined over the same x ϵ C_x , the relative entropy is specified as

$$RelEnt(p/q) = \sum_{x \epsilon C_x} p(x) \log \frac{p(x)}{q(x)} \tag{5.5}$$

Relative Entropy was calculated for each pair of d record. The CAN data record consist of 8 bytes in which it has some values representing mode and some values representing value. So we concatenated each 2 pair of bytes and resulted into 4 bytes for each CAN Data record.This will ensure the relation between each bytes in higher dimension rather than understanding its individual byte.This was assisted by the analysis of CAN data set in which some bytes was not changing for considerable amount of records and some bytes was fluctuating between its extreme range. This gave us correlation between the bytes and compelled us to combine 2 bytes each from 8 bytes of record. The results were promising.

5.4.3 Results

The following result consist of CAN Message ID 158.

Total no of records: 46407

Average entropy: 0.0973206012162

In the Figure 5.1,the relative entropy found between pair of data records was less than 4 for most o the pure non-anomaly data.

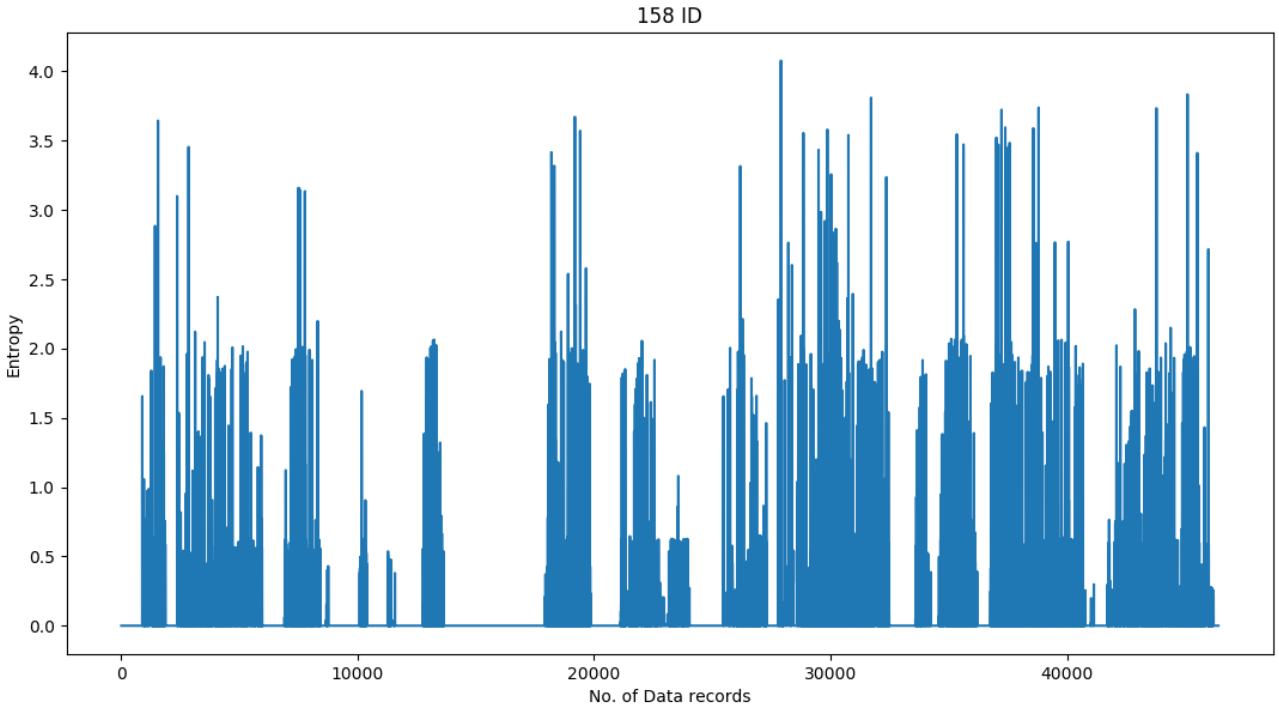

Figure 5.1: Relative Entropy

The following result consist of Relative Entropy Byte Wise Anomaly Data of CAN Message ID 158.

Total no of records: 13066

Average entropy:0.788268548194

In Figure 5.2, the majority of CAN Data message's relative entropy is above 4 which shows the randomness of the Anomaly.

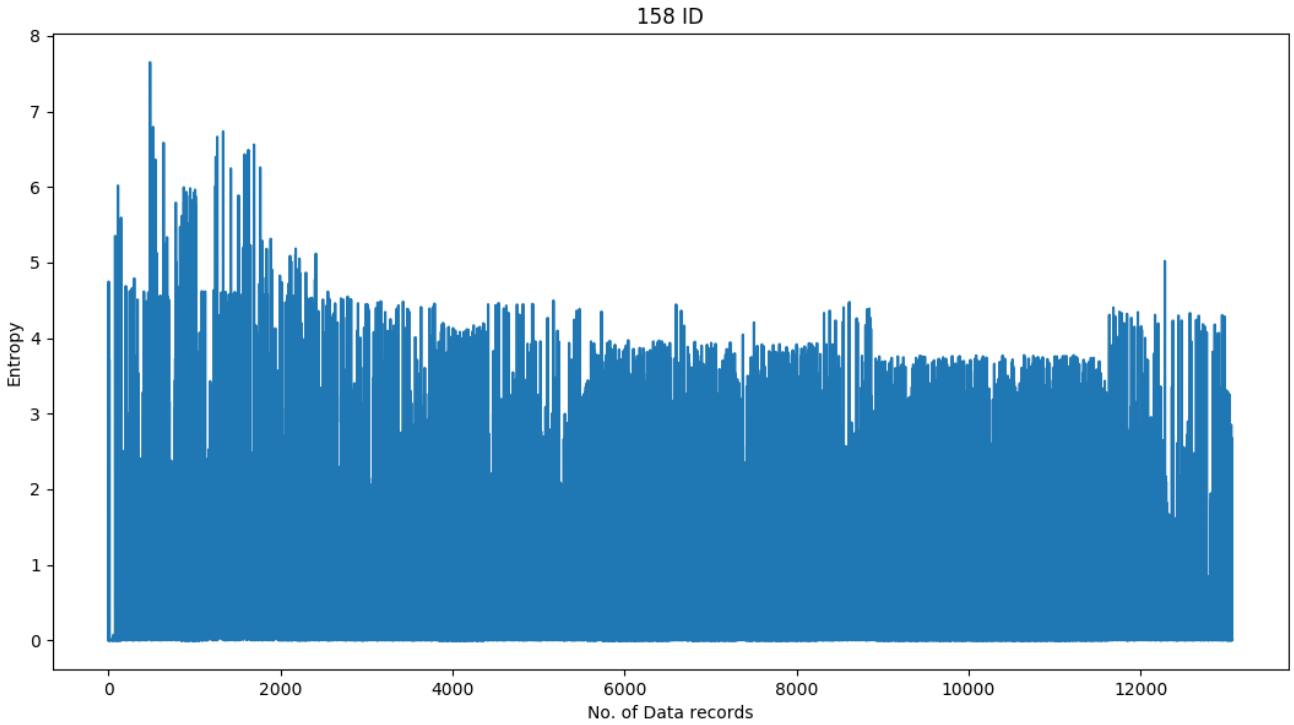

Figure 5.2: Relative Entropy for ID 158 Byte Wise Anomaly Data Message

The following result consist of Relative Entropy for Pair-Wise Inverted Probability Distribution Anomaly and Normal Data for ID 158.

Total no of records: 14232

Average entropy: 0.755837401269

In Figure 5.3, the relative Entropy for Pair-Wise Inverted Probability Distribution Anomaly for ID 158 along with Normal CAN Data Message at interval of 100 records shows the difference of their entropy overlay-ed on each other

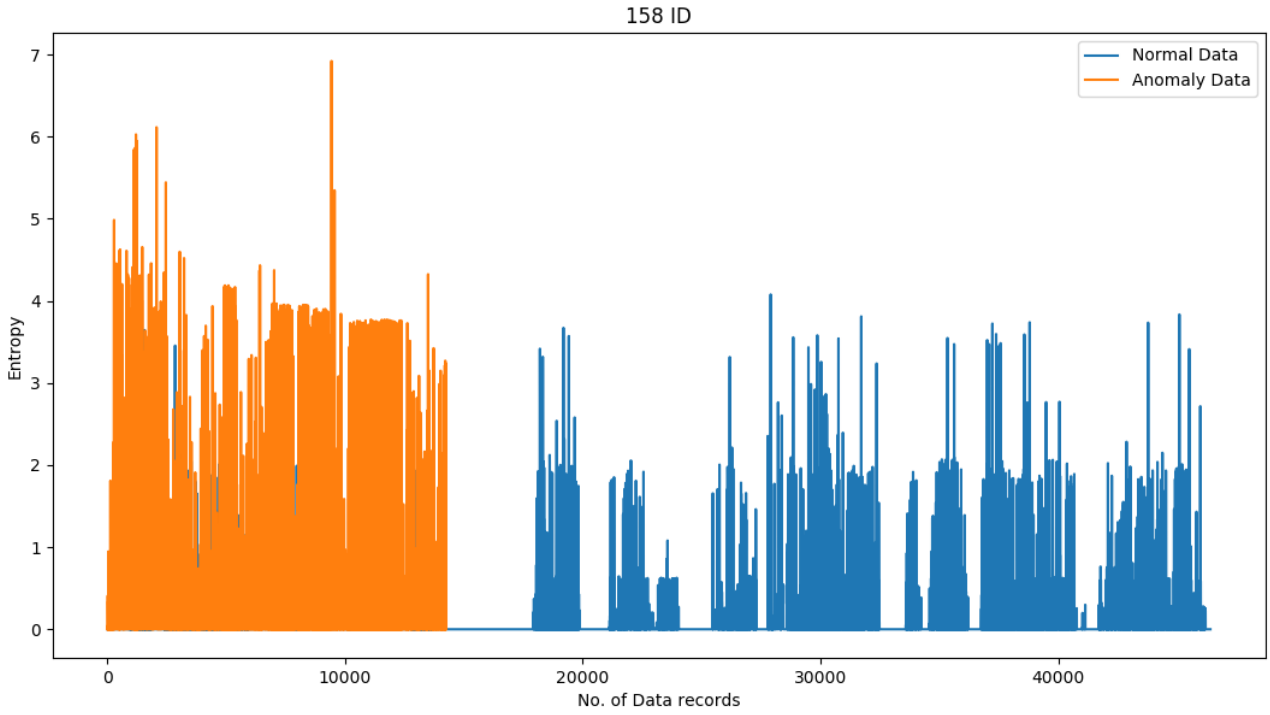

Figure 5.3: Relative Entropy Pair-Wise Inverted Probability Distribution Anomaly

Relative Entropy for DataSet which consist of Pair-Wise Inverted Probability Distribution Anomaly and Normal Data. Figure 5.4 shows that the Dataset synthesis consist of Anomaly Packets each at the interval of 100 in Normal Data of 20000 records.This resulted in higher entropy with nearby Normal data records as depicted through the Plot. High spikes was recorded at regular intervals of 100 data records which was strong enough to confidently detect anomaly within Normal data set.

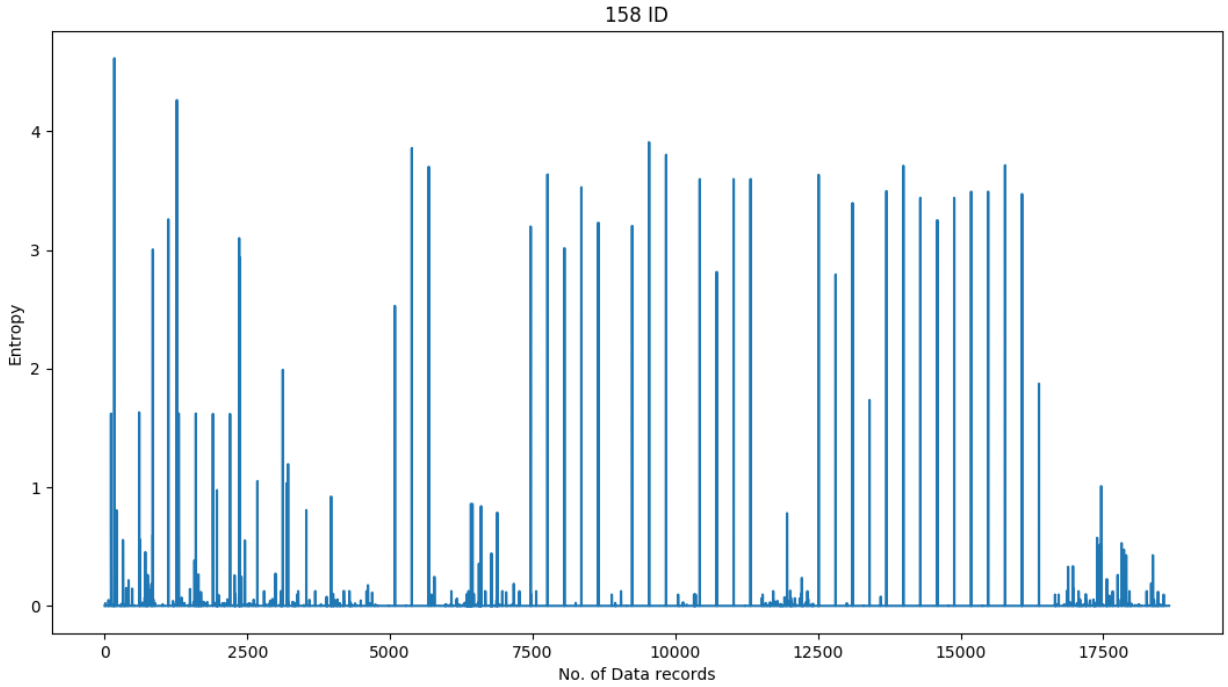

Figure 5.4: Relative Entropy of Anomaly and Normal CAN Data

5.5 Deep Neural Network

DNN has been extensively studied in a machine learning research field, and widely used for practical applications in computer vision and image processing, speech recognition, etc [13]. DNN is adopted here as it shows remarkable classification performance [14]. Our proposed method trains high-dimensional CAN packet data to figure out the underlying statistical properties of normal and hacking CAN packets and extract the corresponding features. Once the features are trained off-line, the proposed system monitors an exchanging packet in a vehicular network to decide whether a system is being attacked, or not. The system provides an instant response to the attack as DNN requires little computations in the decision.

5.5.1 How the model works

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

Neural networks help us cluster and classify. You can think of them as a clustering and classification layer on top of data you store and manage. They help to group unlabeled data according to similarities among the example inputs, and they classify data when they have a labeled dataset to train on. (To be more precise, neural networks extract features that are fed to other algorithms for clustering and classification; so you can think of deep neural networks as components of larger machine-learning applications involving algorithms for reinforcement learning, classification and regression.)

5.5.1.1 Elements of Neural Network

Deep learning is the name we use for “stacked neural networks”; that is, networks composed of several layers. The layers are made of nodes. A node is just a place where computation happens,

loosely patterned on a neuron in the human brain, which fires when it encounters sufficient stimuli. A node combines input from the data with a set of coefficients, or weights, that either amplify or dampen that input, thereby assigning significance to inputs for the task the algorithm is trying to learn. (For example, which input is most helpful is classifying data without error?) These input-weight products are summed and the sum is passed through a node’s so-called activation function, to determine whether and to what extent that signal progresses further through the network to affect the ultimate outcome, say, an act of classification.Here’s a diagram 5.5 of what one node might look like.
A node layer is a row of those neuron like switches that turn on or off as the input is fed through the net. Each layer’s output is simultaneously the subsequent layer’s input, starting from an initial input layer receiving your data.Figure 5.6 shows the Basic Deep Neural Network Cell Layers stack where input, hidden and output layer are connected.
Pairing adjustable weights with input features is how we assign significance to those features with regard to how the network classifies and clusters input.

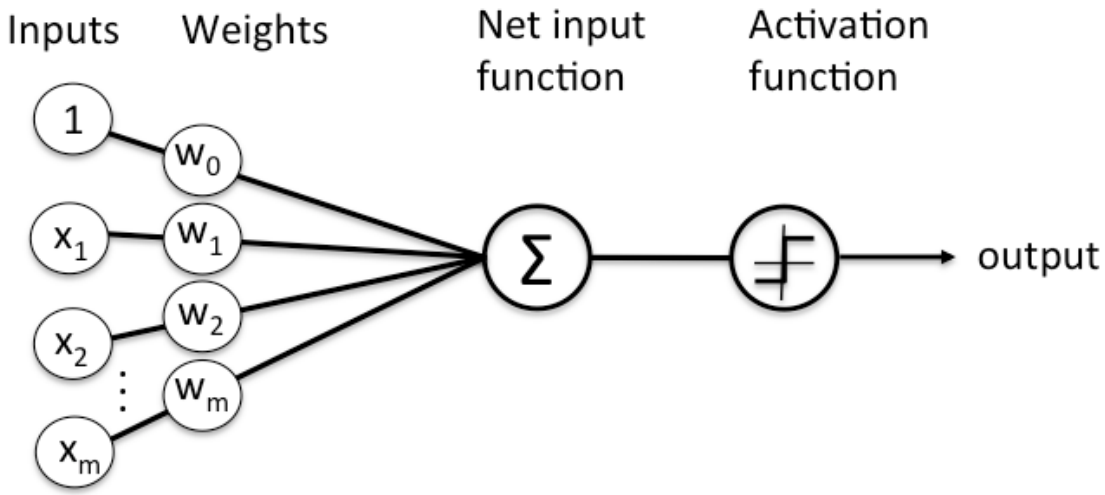

Figure 5.5: DNN Basic structure [17]

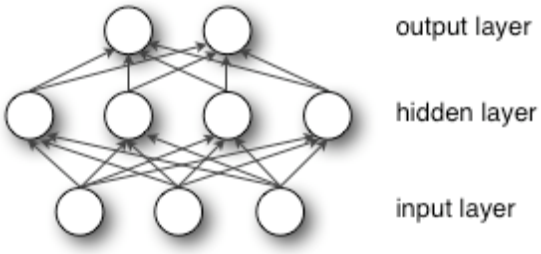

Figure 5.6: Basic DNN Cell Layers [16]

5.5.2 Application of model in CAN Bus

The proposed method extracts a CAN packet feature as an abstract representation of the system status. The feature design considers two key factor, i.e., the performance of an intrusion detection and the computational complexity during the extraction.

To this aim, we create features directly extracted from a bitstream in the network before decoding. Specifically, counting an occurrence of a bit-symbol in the binary bitstream is employed for forming a high dimensional feature. As the "DATA" contains 8 Bytes (=64 bits), the bits ranging from the 0-th position to the 63-th position are treated as a 64-dimensional feature. The counts of the bits are normalized into a probability. We may use all the bit patterns as a high dimensional feature. However, if necessary, the dimension can be reduced by a prior knowledge regarding a specific semantics in the corresponding syntax element. In the proposed method, the semantics is divided into mode information and value information. The mode information is defined as a state of a vehicle, e.g. adjusting a wheel, characterizing a CAN packet. The value information is constructed with bit-fields determining the value of the specific mode e.g. a wheel angle or a speed. There can be the other bits beside the mode information and the value information, and they will be not included into the training process.

Note the proposed method requires only little computational complexity during the feature extrac-

tion because it requires few shift and logical operators handing bits. Any features form the reconstructed original digital signal may yield a better performance, but the complexity is intractable to the real-time detection in the IDS.

5.5.2.1 Training

DNN provides an efficient mean to model nonlinear relations between inputs and outputs while other supervised machine learning algorithms are fitted into linear models.

Hence, we adopt DNN to the intrusion detection problem as our feature from CAN bitstreams have nonlinearity. The DNN can be augmented from the conventional multi-layered artificial neural network (ANN). However, the straightforward augmentation is inefficient when using the back-propagation learning, caused by the vanishing gradient problem and the slow convergent speed. To prevent the problem Erhan et al. propose the unsupervised pre-training scheme where the weight parameters in the hidden layers are trained using a Restricted Boltzmann Machine [15]. Motivated by this, we apply the pre-trained parameters as an initial value in the proposed technique. We show the learning mechanism of the proposed DNN applied to classifying the normal and hacking packets as follows.

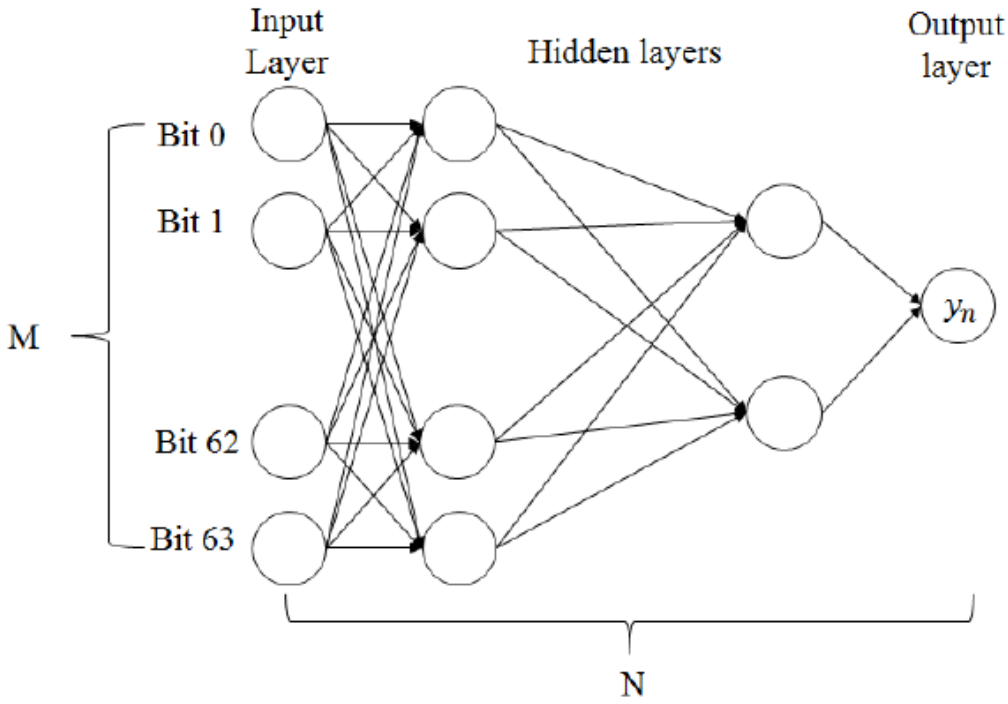

Figure 5.7: Proposed Architecture of deep neural network. [17]

Fig. 5.7 shows an input layer, multiple hidden layers, and an output layer. Each node in Fig. 5.7 calculates an output with using an activation function of an input value. Though advanced activation functions such as rectified linear unit (ReLU) [16] are developed in recent, we use a sigmoid activation for an ease of implementation. Considering a supervised learning, we have a training set $(b^{(1)}, y^{(1)})$,...,$(b^{(k)}, y^{(k)})$ of samples where b = b_0,b_1,...,$b_B \epsilon R^B$ is the set of the feature

including bits, and is the binary result determining a normal packet or a hacking packet. For this, a cost function as an mean squared error function between the prediction value and the output is defined as,

$$C(W; b, y) = \frac{1}{2}\|h_w(b) - y\|^2 \tag{5.6}$$

where is the set of an adaptation weight in an edge, connecting two nodes in the network, and $h_W(b)$ is a hypothesis providing an estimated output. For a batch training, we define the overall cost function to be

$$C(W) = \frac{1}{K}\sum_{k=1}^{K} C(W, b^k, y^k) + \frac{\gamma}{2}\sum_{n=1}^{N}\sum_{i=1}^{M_l}\sum_{j=1}^{M_{l+1}}(w_{ji}^n)^2 \tag{5.7}$$

where N is the depth of the neural network, M_l is the number of nodes in the l-th layer, and $w_{ji}^n \epsilon$ W is the weight of the connection between a node i in the n-1 th layer and a node j in the n th layer. In each node in a layer, the output is computed with the sigmoid function of the linear combination of input values and the weights. We aim to minimize the cost function in (5.4) to obtain the weighting parameters. The back propagation algorithm is known to be effective with a pre-trained parameters in the problem and, thus, we use a stochastic gradient method to train the network. Specifically, we obtain the partial derivative of the cost function $C_k(W)$ and use the term for the adaptation in each iteration,

$$w_{ji}^{(n)} = w_{ji}^{(n)} + \eta\frac{\partial C_k(W)}{\partial w_{ji}^{(n)}} \tag{5.8}$$

where η is an adaptation parameter.

5.5.2.2 Detection

In a detection step, we predict a class of a testing CAN packet with the trained deep neural network. The output is simply calculated with the trained weight parameters in and the extracted feature set b from the testing CAN packet. The optimum weight parameter set after training the DNN is stored in a profiling module because the learning requires some computational burden. However, the decision is very quick as compared to the learning process since it consists of a few multiplications to a forwarding direction. Once the input feature is given to the network, the output in the final node is determined as either 0 or 1, telling if the packet is normal or abnormal. We have also trained our system by converting each byte of our packet data into decimal value and then training our DNN model for 8 field input value.

5.5.3 Results

	TP	TN
Bit-wise	100	99.79
Byte-wise	99.96	98.56

Table 5.2: Random (Unusual CAN Messages) for DNN

	TP	TN
Bit-wise	99.33	99.35
Byte-wise	99.865	95.08

Table 5.3: Probability Distribution for DNN

Here in Table 5.2 and Table 5.3 show case True Positive(TP) and True Negative(TN) for both Bit-wise and Byte-wise format of CAN Messages in Random and Probability Distribution way of expressing anomaly. Random Anomaly has better result than Probability Distribution Anomaly because Random Anomaly is less likely to be expected in the CAN Bus system.Probability Distribution takes into consideration of probable bit or byte occurring in the CAN Message as perceived by the attacker.

	TP	TN
Bit-wise	72.59	73.48
Byte-wise	90.32	81.81

Table 5.4: Pairwise Probability Distribution for DNN

	TP	TN
Bit-wise	NA	NA
Byte-wise	83.26	91.54

Table 5.5: Pairwise Inverted Probability Distribution for DNN

Here in Table 5.4 and Table 5.5 show case True Positive(TP) and True Negative(TN) for both Bit-wise and Byte-wise format of CAN Messages in Pairwise Probability Distribution and Pairwise Inverted Probability Distribution way of expressing anomaly.Pairwise Inverted Probability Distribution in Bit-wise doesn't apply as the value is in Bit.Pairwise Inverted Probability Distribution in Byte-wise contains extreme minimum and maximum probable values in Byte position.The result is considerable as the boundary conditions were too strict for Deep Neural Network.Pairwise Probability Distribution in Byte-wise performed better than Bit-wise as the features in Byte-wise were more co-related in higher dimension (Byte-wise) than lower dimension (Bit-wise).

5.6 Long Short Term Memory

A variety of approaches have been applied to anomaly detection in sequences [10]. A common approach is to identify and compare patterns in n-grams, where an n-gram is a length-n subsequence. Anomalies can take the form of foreign symbols, rare n-grams, or foreign n-grams [11]. These methods are difficult to apply to CAN traffic because they assume a finite symbol dictionary, or use quantization methods to convert continuous values to a finite set of symbols. CAN symbols are 64-bit binary words. The number of possible words for a particular ID in a 64-bit address space is potentially infinite over the lifetime of the car, so we cannot rely on foreign symbols or n-grams to detect anomalies. Instead, Recurrent Neural Networks(RNNs) are a more natural fit for CAN bus data sequences. LSTM-based RNNs have recently been shown to be very effective for anomaly detection in standard time series test sets [12], electrocardiography, and aircraft telemetry. Each of these systems trains a RNN to predict the next symbol in the sequence. Anomalies are flagged when the error between real and predicted values is high. When the series is inherently unpredictable, the LSTM encoder-decoder can produce superior results to the LSTM predictor [12]. We based our detector on the original LSTM structure because it has been shown to be robust in a variety of domains.

5.6.1 How the model works

Unlike traditional RNNs, an LSTM network is well-suited to learn from experience to classify, process and predict time series when there are time lags of unknown size and bound between important events. Relative insensitivity to gap length gives an advantage to LSTM over alternative RNNs and hidden Markov models and other sequence learning methods in numerous applications.

Long-Short Time Memory architecture consists of a set of recurrently connected subnets.
LSTMs help preserve the error that can be backpropagated through time and layers. By maintaining a more constant error, they allow recurrent nets to continue to learn over many time steps (over 1000), thereby opening a channel to link causes and effects remotely.

LSTMs contain information outside the normal flow of the recurrent network in a gated cell. Information can be stored in, written to, or read from a cell, much like data in a computer's memory. The cell makes decisions about what to store, and when to allow reads, writes and erasures, via gates that open and close. Unlike the digital storage on computers, however, these gates are analog, implemented with element-wise multiplication by sigmoids, which are all in the range of 0-1. Analog has the advantage over digital of being differentiable, and therefore suitable for backpropagation. Those gates act on the signals they receive, and similar to the neural network's nodes, they block or pass on information based on its strength and import, which they filter with their own sets of weights. Those weights, like the weights that modulate input and hidden states, are adjusted via the recurrent networks learning process. That is, the cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, backpropagating error, and adjusting weights via gradient descent.
Figure 5.8 shows the Data flow in Long Short Term Memory Architecture. The phases included are input squashing,input gating, memorizing and forgetting, output squashing and output gating.

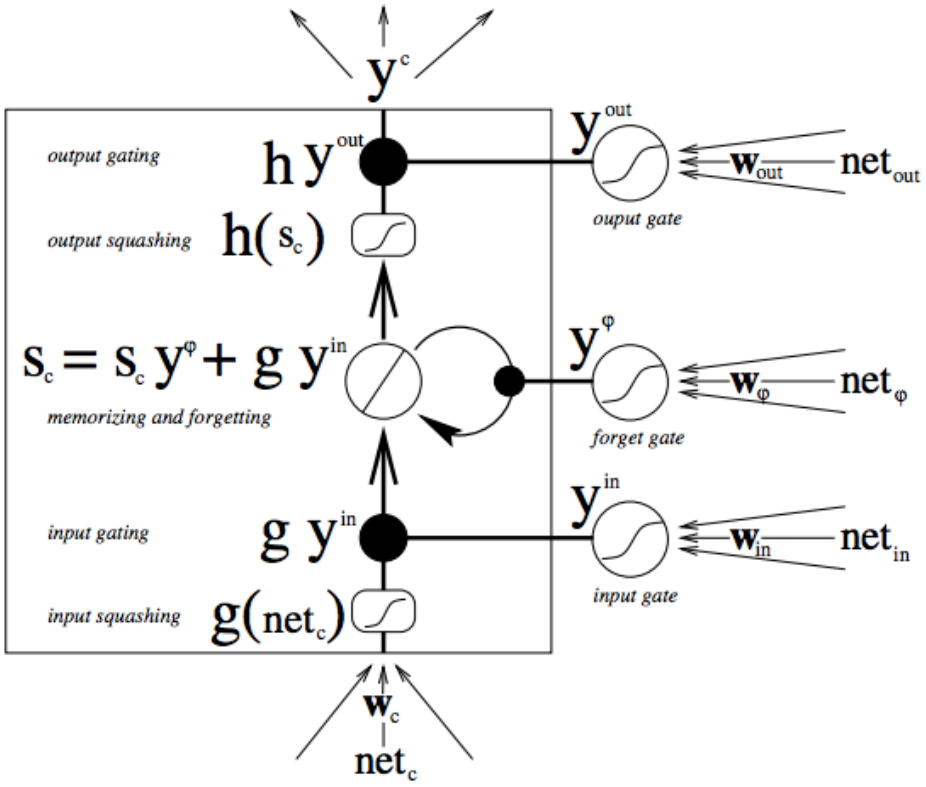

Figure 5.8: Data flow in LSTM Architecture [8]

The steps involved in LSTM shown in Figure 5.8 LSTM Architecture are:

1. Input gate activation
2. Forget gate activation
3. Cell input and cell state
4. Output gate activation
5. Cell output

LSTM Recurrent Neural Networks : Our anomaly detector is a RNN with LSTM units.The basic RNN is a neural network with feedback in Figure 5.8.The output h_t is a function of both input and x_t and the previous output h_{t-1}:

$$h_t = f(Wx_t + Uh_{t-1} + b) \tag{5.9}$$

where W, U are weight matrices, b is a bias term, and f is a nonlinear transformation (e.g. the sigmoid or tanh function). In this way RNNs make use of information from both the past and the present. However they are difficult to train. Neural networks are trained with backpropagation, a process of adjusting the weights according to the derivative of the output errors. The feedback loop in RNNs cause errors to shrink or grow exponentially, destroying the information in the backpropagation

signal. Hochreiter solved this gradient problem by introducing additional structure to the network in the form of explicit input and forget gates. These gates depend on both the input xt and previous step's output ht: (Fig. 5.9), and control how much of the internal state depends on the new input and the previous state.

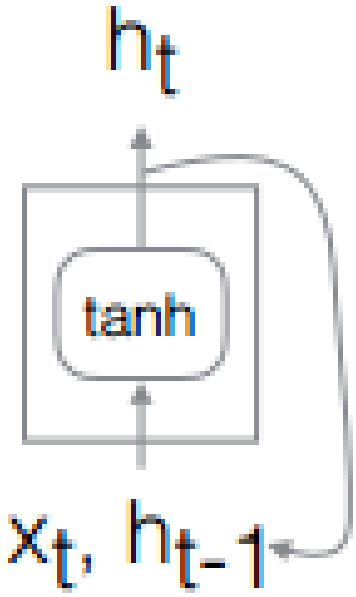

Figure 5.9: Basic RNN cell.The weight multiplications are omitted.[9]

$$i_t = \alpha(W_i x_t + U_i h_{t-1} + b_i) \tag{5.10}$$

$$f_t = \alpha(W_f x_t + U_f h_{t-1} + b_f) \tag{5.11}$$

Here W_i, U_i, W_f, U_f are learned weights, and b_i and b_f are learned bias terms. The sigmoid function α is typically used for the gate terms to scale their output between 0 and 1. The input and forget signals (i_t and f_t) determine the new cell state c_t as a linear combination of the previous internal state $c_t - 1$ and new candidate internal state $\bar{c}_t$:

$$\bar{c}_t = tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{5.12}$$

$$c_t = f_t c_{t-1} + i\bar{c}_t \tag{5.13}$$

where again W_c and U_c are weight matrices and b_c is the bias. The output is also controlled by a gate function with its own weights W_o, U_o and bias b_o:

$$o_t = \alpha(W_o x_t + U_o h_{t-1} + b_o) \tag{5.14}$$

$$h_t = o_t tanh(c_t) \tag{5.15}$$

The design and evaluation of LSTM variants is an active area of research; we chose the standard structure because it has been proven to work well over a range of problem domains.

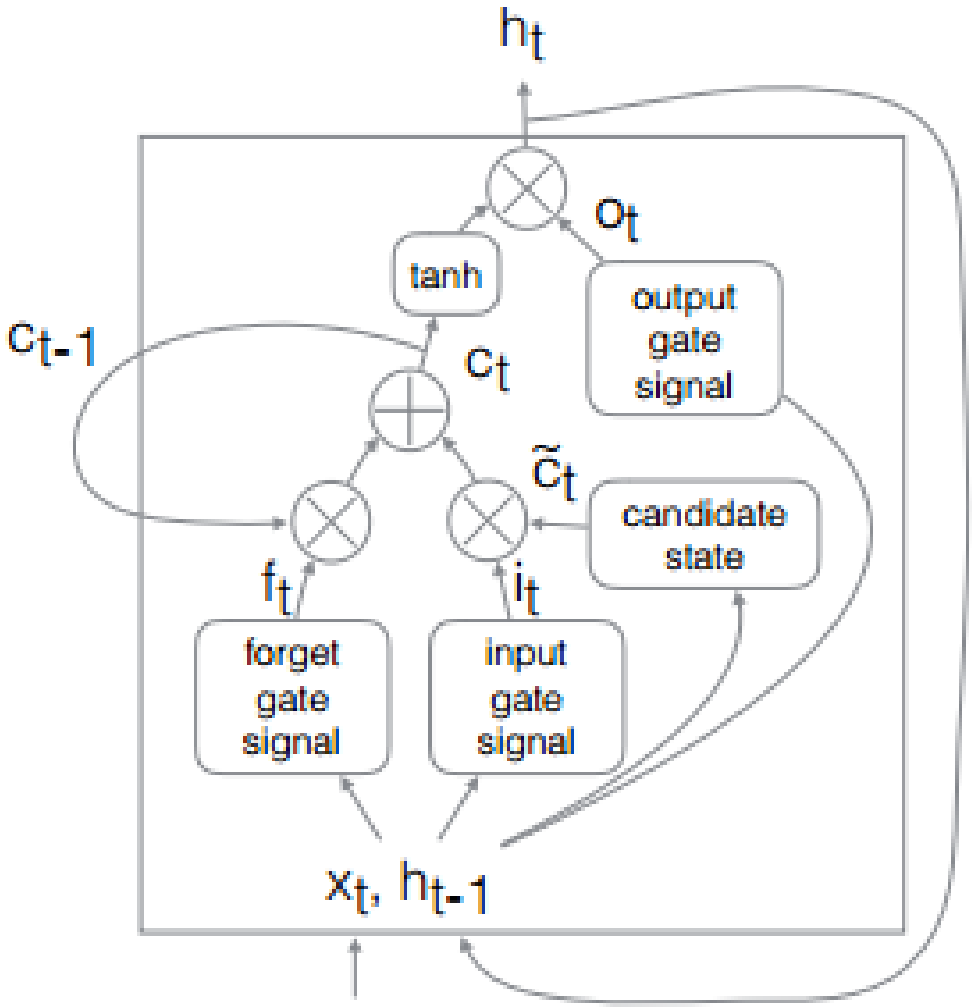

Figure 5.10: LSTM basic unit.[9]

LSTM adds additional connections designed to facilitate forgetting and remembering the past. Note the weight multiplications have been omitted for clarity in emphasizing the information flows. Figure 5.10 shows the overall LSTM basic unit in which each symbols representing is covered in equations mentioned before.

5.6.2 Application of the model in CAN Bus

The LSTM model was applied in two segments, for bit data with 64 bits of the data word as input and for byte data with 8 bytes of input.

Training on bit data:
Firstly, we used 64 bits of each packet as input each as a separate input feature. We experimented with adding linear embedding layers to project the binary inputs into a real-valued state space space. Our final network is composed of a linear segment, a recurrent segment, and a linear output segment (Figure 5.11). Given an input sequence x_1, x_2, x_3,...,x_N , where x_i is a 64-bit vector, the network is trained for each i with a target of the subsequent word $y_i = x_i + 1$. At each time step, the input x_i is the 64-bit data word from the packet. These bits are transformed by two non-recurrent hidden layers, each with 128 units and tanh activation functions. The output of the linear layers is fed into two recurrent LSTM layers, each with 512 units and tanh activation functions. The final layer is again linear, with 64 units using sigmoid activation functions to scale the final values to between 0 and 1. A separate network was trained for each CAN bus ID.

This network structure was selected by experimenting with different numbers of layers, hidden units, activation functions, and other parameters and evaluating with validation data. The training data was composed of subsequences 20 words long, presented in batches of size 128. Training was stopped when the validation performance failed to improve after five consecutive runs through the training data.

Training on byte data:
We used similar architectural layers but now giving input as 8 byte words in integer format at each time step. These bits are transformed by two dense hidden layers, each with 32 units and tanh activation functions. The output of the linear layers is fed into two recurrent LSTM layers, each with 64 units and tanh activation functions. The final layer is again linear, with 8 units using sigmoid activation functions to scale the final values to between 0 and 255 for each of the 8 bytes. A separate network was trained for each CAN bus ID. The last step is the calculation of loss for each of the output. We calculated the binary cross entropy loss of each packet for bit-wise data in the testing dataset. Binary cross-entropy between predictions and targets is calculated by

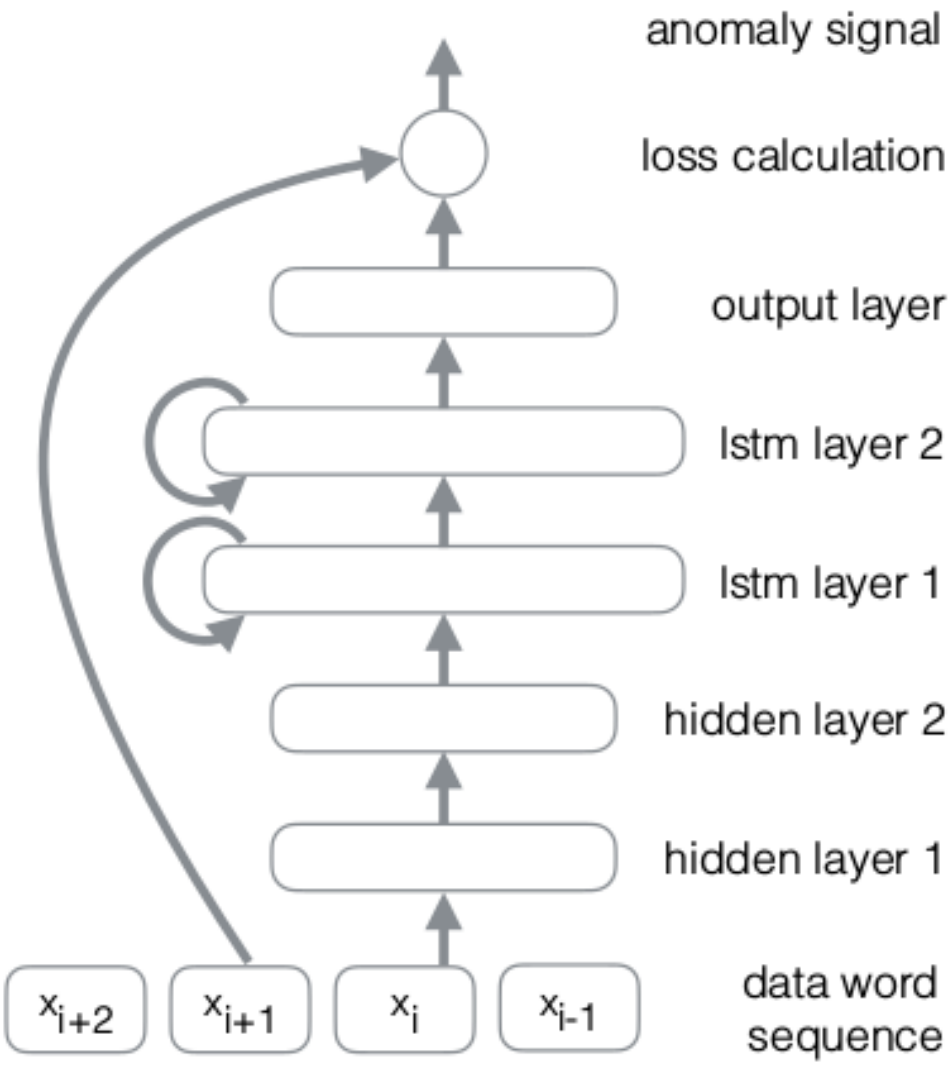

Figure 5.11: LSTM-based anomaly detector structure.[9]

5.6.3 Results

We trained the neural network with the described architecture. In order to test the model, we calculated the loss of each packet for the data in the testing dataset. The loss within the true data turned out to be less compared to the loss in the anomaly data for both the bit-wise as well as byte-wise methods. Hence, to classify the data as true or anomaly we approximately chose a loss function value which tries to separate the loss values most distinctively. For bit-wise loss function we chose 0.55 as the loss value which separates the two kinds of data from the dataset as shown in the figure 5.12. For byte-wise loss function a loss value of 0.425 was used for all the different anomalies on which the model was tested as shown in the figure 5.13.From the figures 5.12 and 5.13 it can be seen that the loss values of the true data and the anomaly data clearly separates out from each other for both the bit and byte representations. Although the loss of bit-wise structure of the two kinds of data is more prominent, the separation for the byte-wise structure is sufficient enough to classify the anomaly data.

Following Table shows the results of LSTM. The model performed very well in training bit data patterns compared to byte data patterns due to the inherent advantage of the large number of input features being 64 in bit data compared to 8 input attributes in byte data. Table 5.6 shows the result

of True Positive and True Negative for testing data which consist of completely unusual Random CAN Messages. The results were quite encouraging and promising.

	TP	TN
Bit-wise	97.86	98.41
Byte-wise	96.11	97.89

Table 5.6: Random (Unusual CAN Messages) for LSTM

Table 5.7 shows the results of testing data of Probability Distribution of both bit and byte CAN data.The Bit representation of CAN data showed more promising results than Byte data.

	TP	TN
Bit-wise	92.47	94.84
Byte-wise	88.25	77.87

Table 5.7: Probability Distribution for LSTM

Table 5.8 shows the results of testing data consisting of Pairwise Probability Distribution of both bit and byte CAN data. In this too, we got the similar results as compared to Probability Distribution of CAN messages.

	TP	TN
Bit-wise	94.34	94.29
Byte-wise	85.25	74.92

Table 5.8: Pairwise Probability Distribution for LSTM

Table 5.9 shows the results of True Positive and True Negative of Pairwise Inverted Probability Distribution considering only Byte representation of CAN data Packets.

	TP	TN
Bit-wise	NA	NA
Byte-wise	88.25	92.96

Table 5.9: Pairwise Inverted Probability Distribution for LSTM

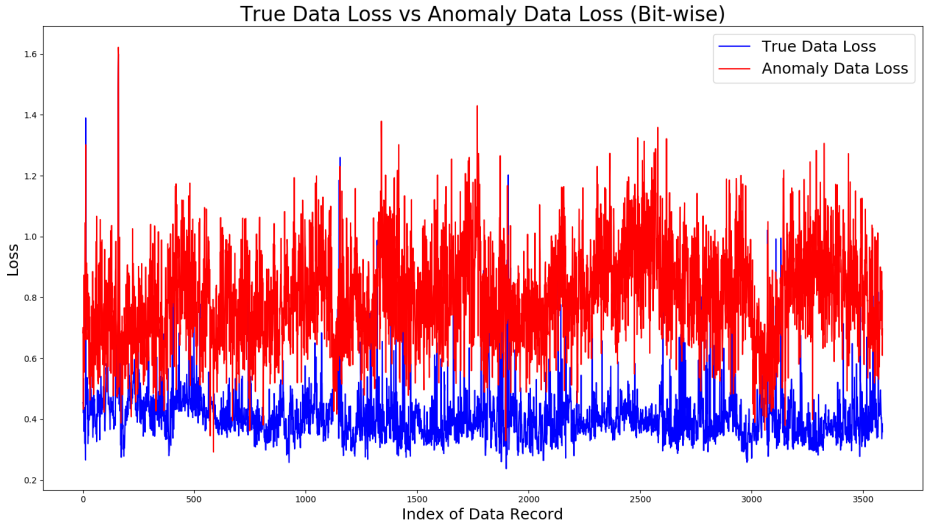

Figure 5.12: True Data Loss Vs Anomaly Data Loss(Bit-wise)

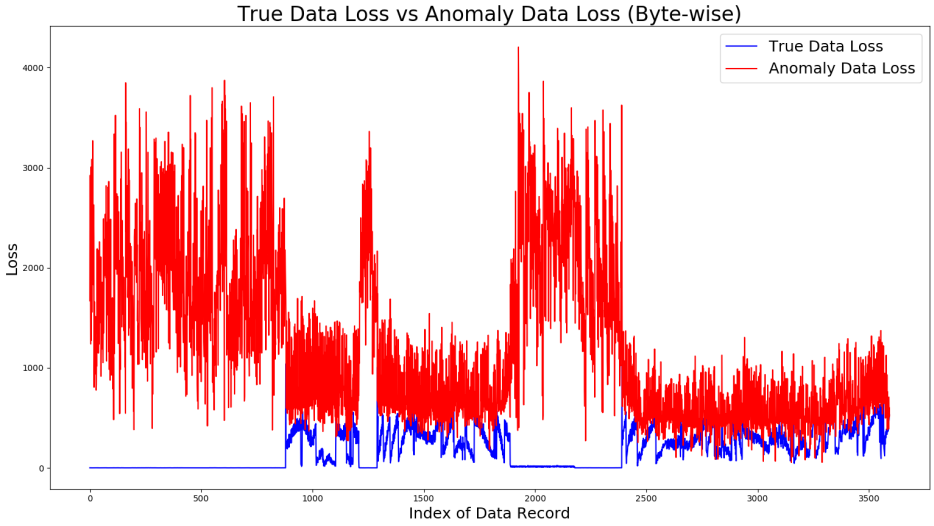

Figure 5.13: True Data Loss Vs Anomaly Data Loss(Byte-wise)

Chapter 6

Summary and Conclusion

We have understood the working of CAN Bus Technology by going through all the research and review paper. We have purchased the CAN BUS Shield consisting of MCP2515 CAN Controller and MCP2551 CAN Transceiver. We have started with the sending and receiving of the CAN Packets. Our main goal was to exploit the security of the CAN BUS and perform various attacks on it and implement the prevention and detection of the malicious attacks on CAN BUS Protocol.

A reliable, optimized and fast encryption and authentication algorithm has been implemented on CAN Bus. Although these algorithms increase the latency in the bus, in these times of security breaches , it is important to provide security over the bus. Even faster ciphers such as blowfish could be implemented over the bus, but these ciphers have the issues of large input block sizes.

We have demonstrated the detection of sequence data anomalies on the automotive CAN bus using LSTM neural networks. The LSTM approach has the advantages of not requiring knowledge of the specific protocol, and has shown promising performance in detecting a range of anomalies corresponding to known attacks on the CAN bus, and can detect a kind of attack invisible to previous detection methods. However more work needs to be done to achieve a practical level of false alarm rate while reliably detecting anomalies. One shortcoming of the current approach is that it treats each ID's data sequence as independent. It is more likely that there are data interdependencies between IDs, so that a detector that monitors all IDs simultaneously may achieve better overall performance.

Here we also proposed a novel intrusion detection approach using a deep neural network (DNN), training in vehicle network packets exchanged between electronic control units (ECU). High-dimensional in-vehicle network packets to extract features were trained and used for discriminating normal and hacking packets. The features were directly taken from a bitstream over the network for being high efficiency and low complexity features. Once the features were trained and stored in the profiling module, the proposed system examined an exchanging packet in the vehicular network to decide whether a system was being attacked, or not. The system provided a real-time response to the attack with a significantly high detection ratio from our experimental results.

We also proposed and evaluated an entropy-based algorithm for detecting anomalies in CAN messages generated by an unmodified licensed vehicle. In particular this includes extensive experimental evaluation based on several hours of CAN traffic captured during driving sessions on public

motorways, reflecting real road and traffic conditions.Thus the relative entropy of Anomalous data defer to great extent with the CAN data Message. Additionally we also used Apriori algorithm to find relation between packets arriving from different CAN IDs and find out their association rules.

In the age of connected vehicles, there could be a monitoring component for a fleet of cars. Such an approach could contribute to managing the false alarm rate by building knowledge of normal outliers for each make and model of car. The bigger picture also addresses an issue we avoided, the question of how to respond to an anomaly detection. For example, a high-confidence alert could automatically put the vehicle into a “safe mode” designed to allow the driver to safely stop the car while preventing further damage from being done. Alerts could also be transmitted to the manufacturer for further analysis. Regardless of how system designers incorporate an anomaly detector, if it can be made reliable it will become an important part of automotive cyber security and prevent various mishap that would make humans life safer on road.

References

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, Experimental security analysis of a modern automobile," in Security and Privacy (SP), 2010 IEEE Symposium on, Oakland, CA, USA, May 2010, pp. 447 462.

[2] R. N. Charette, This car runs on code," IEEE Spectrum, Feb 2009.

[3] Delivering online capabilities on the road," Alcatel-Lucent, Tech. Rep., 2010.

[4] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Sav- age, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, Comprehensive experimental analyses of automotive attack surfaces," in Proceedings of the 20th USENIX conference on Security, Berkeley, CA, USA, Aug 2011.

[5] C. Miller and C. Valasek, "Adventures in Automotive Networks and Control Units," IOActive Labs Research, Tech. Rep., Aug. 2013. [Online]. Available: http://blog.ioactive.com/2013/08/car-hacking-content.html

[6] A. G. Illera and J. V. Vidal, "Dude, WTF in my CAN!" presented at Black Hat Asia 2014. [Online]. Available: https://www.blackhat.com/asia-14/briefings.html

[7] E. Keogh, J. Lin, and A. Fu, "HOT SAX: efficiently finding the most unusual time series subsequence," in Proc. Fifth IEEE International Conference on Data Mining, Nov. 2005, pp. 226–233.

[8] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long Short Term Memory Networks for Anomaly Detection in Time Series," in Proc. 23rd European Symposium On Artificial Neural Networks, Computational Intelligence and Machine Learning, Bruges, Belgium, 2015, pp. 89–94.

[9] Adrian Taylor,Sylvain Leblanc,Nathalie Japkowicz "Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks" in 2016 IEEE International Conference on Data Science and Advanced Analytics

[10] M. Markovitz and A. Wool, "Field Classification, Modeling and Anomaly Detection in Unknown CAN Bus Networks," in escar Europe 2015, 2015. [Online]. Available: https://www.escar.info/history/escar-europe/escareurope- 2015-lectures-and-program-committee.html

[11] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” Neural Computation, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[12] T. Hoppe, S. Kiltz, and J. Dittmann, “Security threats to automotive can networkspractical examples and selected short-term countermeasures,” Reliability Engineering and System Safety, vol. 96, no. 1, pp. 11–25, 2011.

[13] C. Miller and C. Valasek, “Adventures in automotive networks and control units,” 2013.

[14] M. Muter and N. Asaj, “Entropy-based anomaly detection for in-vehicle networks,” IEEE Intelligent Vehicles Symposium, Proceedings, no. Iv, pp. 1110–1115, 2011.

[15] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” Advances in Neural Information Processing Systems 25 (NIPS 2012), 2012.

[16] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups,” Communications Magazine, IEEE, vol. 44, no. 1, pp. 74–82, 2012.

[17] Min-Ju Kang,Je-Won Kang ”A Novel Intrusion Detection Method Using Deep Neural Network for In-Vehicle Network Security” in Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2014R1A1A2056587).

[18] T. Hoppe and J. Dittman, “Sniffing/replay attacks on can buses: A simulated attack on the electric window lift classified using an adapted cert taxonomy,” in Proc. of the 2nd Workshop on Embedded Systems Security (WESS), Oct 2007

[19] M. Muter, A. Groll, and F. Freiling, “A structured approach to anomaly detection for in-vehicle networks,” in Proc. of the Sixth International Conference on Information Assurance and Security (IAS 2010), Aug 2010

[20] T. M. Cover and J. A. Thomas, Elements of Information Theory. New York, NY, USA: Wiley-Interscience, 1991.

[21] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” ACM Computing Surveys (CSUR), vol. 41, no. 3, p. 15, 2009.

[22] M. Sutton, A. Greene, and P. Amini, Fuzzing: brute force vulnerability discovery. Addison-Wesley Professional, 2007.

Acknowledgements

We would like to thank our project guide Dr. S. G. Bhirud for contributing his time and effort to help us during the course of this project, giving it the present shape. It would have been impossible to complete the project without his support, valuable suggestions, criticism, encouragement and guidance.

He has always been involved by discussing our topic at each phase to make sure that the experiment is designed and carried out in an appropriate manner and that our conclusions are appropriate, given their results. His constant support and interaction have been a driving force which has constantly motivated us to explore the different aspects of our project.

We are also grateful for all other teaching and non-teaching staff members of the Computer Engineering department for directly or indirectly helping us for the completion of this project and the resources provided.