MASTER

Model Based Framework for In-vehicle Network Simulation and Security Threat Demonstration

Anita Shanmugalingham, Sasiraaju

Award date:
2015

Technische Universiteit Eindhoven University of Technology

Department of Mathematics and Computer Science

Model Based Framework for In-vehicle Network Simulation and Security Threat Demonstration

Masters - Automotive Technology
Graduation Thesis

Author:
Sasiraaju Anita Shanmugalingham
[0977023]

Supervisors:
dr. Alexios Lekidis[2]
dr.ir.Ion Barosan[1]

Graduation Professor:
Prof.dr.M.G.J.V.D Brand[1]

[1]Eindhoven University of Technology
Department of Mathematics and Computer Science
Software Engineering and Technology Group
Eindhoven, The Netherlands

[2]Security Matters
Eindhoven, The Netherlands

Eindhoven, Monday 20th August, 2018

Technische Universiteit
Eindhoven
University of Technology

Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct[i].

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

16.08.2018

Name

SASIRAAJU ANITA SHANMUGALINGHAM

ID-number

0977023

Signature

A.S. Raju

Submit the signed declaration to the student administration of your department.

[i] See: http://www.tue.nl/en/university/about-the-university/integrity/scientific-integrity/
The Netherlands Code of Conduct for Academic Practice of the VSNU can be found here also.
More information about scientific integrity is published on the websites of TU/e and VSNU

"Learning never exhausts the mind"
-Leonardo DA Vinci

Acknowledgements

This master thesis marks the end of my master's degree in Automotive Technology at the Eindhoven University of Technology. Many people have contributed in a variety of ways to make this thesis successful. Without the help of them, it was impossible to finish this project.

At first, I want to thank my graduation supervisor Prof.Dr.ir. Ion Barosan who gave this opportunity to work in this new field of study. He supported and motivated throughout my master's by giving his knowledge and support. Without his guidance, I could not complete this Master thesis and as well as my degree. I would also like to thank Dr. Elisa Constante who interviewed me and gave the opportunity to work in Security Matters.

My heartiest thanks to Dr. Alexis Lekidis who was my thesis supervisor in Security Matters. From the start to the end of this thesis, his constant advice, thoughts helps me to think in various angles in this project. With his guidance, I have learned a lot of things during this project. I would also thank Guillaume Dupont who helped me in programming during this project at various stages.

My sincere thanks to the A-Team members at our University, who provided the vehicle for my thesis. I would also like to thank all my friends in Eindhoven who gave me good memories during this study period.

Finally, I would like to thank my father Shanmugalingham, my mother Anita and my sister Varshu for their unconditional love and support at all my stages of my life. Without my parents, I would never have enjoyed so many opportunities in my life.

Once again, Thank you all for their love and support.

Abstract

The introduction of autonomous driving features, real-time communication, and the Internet of Things (IoT) in the vehicle industry helps to increase the comfort and safety of the passengers. However, manufacturers are achieving this by adding new external interfaces and new networks to make this happen which leads to increase of cybersecurity threats in the automotive industry. There are many types of research going on to prevent the cybersecurity threats in the vehicle. As a first step, to know the behavior of the vehicle during cybersecurity attacks, a simulation model is created. In this thesis, the primary focus is to create a simulation model of Toyota Prius in-vehicle network and replicate the network behavior in the simulation environment. From that, various attacks are performed in the model to know the behavior of the vehicle when there is a threat.

First, a detail investigation was carried around to know the vehicle E/E architectures used by the manufacturers and different protocols used for the communication between the E/E systems. Then, we analyzed different external interfaces used by the automotive attackers to gain access to the vehicle. The study mentioned above helps more during the practical phase of this thesis (i.e.) while reverse engineering the CAN network traffic of the vehicle and creating the model in the simulation environment.

After this, a comprehensive study was carried out to find the perfect simulation environment to create a simulation model of vehicle E/E architecture and replicate the network traffic. Another important thing is to perform the attack in the simulation model. To match these requirements, OMNET++ a discrete event simulator is chosen. To create a vehicle model in the simulation model, they are many inputs needed such as vehicle architecture, ECU functions, CAN traffic of the vehicle, the purpose of each identifier in the CAN network and associated ECUs. To gather this data a reverse engineering process is carried out in the chosen vehicle (i.e.) 2010 Toyota Prius.

A detailed study is conducted to know the basics of the vehicle, vehicle architecture, and connection between the ECUs, which is a crucial part of the reverse engineering. The CAN traffic of the real vehicle is captured using appropriate tools (CANtact). From the captured traffic the purpose of the identifier and payload information is decoded. Using this information, the input data required for the simulation is obtained. Then, the simulation model is created and replicate the real network traffic. Next step is to perform the various attack in the simulation to know the real vehicle behavior in the case of threats. For these four different types of attacks performed in the model such as Denial of Service, Frame Injection, Masquerade and Black-hole attacks are performed. By achieving this, we can reduce the risk of testing the behavior of threats in the real-car which will protect from the unpredicted behavior of the vehicle and improve the architecture of the vehicle network during the development phase itself.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ACK ..... Acknowledgement

ADAS ..... Advanced Driving Assistance System

ABS ..... Anti-lock Brake System

AVB ..... Audio Video Bridge

CAN ..... Control Area Network

CAPL ..... Communication Access Programming Language

CRC ..... Cyclic Redundancy Code

CSMA/CA ..... Carrier Sense Multiple Access with Collision Avoidance

DLC ..... Data Length Code

DoS ..... Denial of Service

E/E ..... Electric/Electronic

ECU ..... Electronic Control Unit

ECM ..... Engine Control Management

EV ..... Electric Vehicle

EOF ..... End of Frame

FCA ..... Fiat Chrysler Automotive

GPS ..... Global Positioning System

GM ..... General Motors

GUI ..... Graphical User Interface

HSD ..... Hybrid Synergy Drive

ICE ..... Internal Combustion Engine

IDE ..... Integrated Development Environment

IoT ..... Internet of Things

ISO ..... International Organization for Standards

JC ..... Junction Connector

LIN ..... Local Interconnect Network

MOST ..... Media Oriented System Transport

MG ..... Motor/Generator

OBD-II ..... On-Board Diagnostics version II

OEM ..... Original Equipment Manufacturer

OMNET++ .... Objective Modular Network Testbed in C++

OTA ..... Over-the-Air

PATS ..... Passive Anti-Theft System

PHEV ..... Plug-in Hybrid Electric Vehicle

RADAR ..... Radio Detection And Ranging

RFID ..... Radio Frequency Identification

RKE ..... Remote Keyless Entry

RPM ..... Rotation per Minute

RTaW ..... Real Time at Work

RTR ..... Remote Transmission Request

SAE ..... Society of Automotive Engineers

SD ..... Secure Digital

SOF ..... Start-of-Frame

SRR ..... Substitute Remote Request

TPMS ..... Tire Pressure Monitoring System

UTP ..... Unshielded Twisted Pair

USB ..... Universal Serial Bus

V2V ..... Vehicle to Vehicle

V2X ..... Vehicle to Everything

VGA ..... Video Graphics Array

Chapter 1

Introduction

The vehicle industry faces numerous cybersecurity threats due to the growing usage of internet-based communication such as IoT (Internet of Things), network and software in the vehicle. Cars contain over 100 million lines of codes in their ECU (Electronic Control Units) for advanced functionalities, which is 95 times more than the commercial aircraft [41]. Connectivity in cars is also rapidly increasing. By 2020, 75% of modern cars in the world will have network connectivity [22]. Many researchers have been working on autonomous vehicle projects, a vehicle to vehicle communication (V2V) and vehicle to infrastructure communication (V2X) systems to make the travel safely and effortless[35]. The increased dependence on technology as mentioned above has opened the door for cybersecurity threats. It is, however, not clear yet how much importance is given to cybersecurity threats by vehicle manufacturers[62].

Concerns about cybersecurity issues have been raised after the two major attacks. The first cybersecurity attack occurred in General Motors (GM) vehicles On-Star system on 2010 [2]. This attack leads to taking complete control of the car besides the steering wheel by using an on-board computer, due to the exploit present in the On-Star entertainment system. Several million GM vehicles were affected by this attack. The company has released the patches in the form of over the air (OTA) update to the On-Star Systems. However, to protect the vehicle completely from the vulnerability GM took about five years [2]. Another major cybersecurity attack happened in Jeep vehicle on 2015 [13], [6]. This attack leads the attacker to take complete control of the car remotely, due to the vulnerability present in the U-connect system. This attack caused Fiat-Chrysler Automotive (FCA) to recall 1.4 million vehicles in 2015. FCA released the patches to the affected vehicles by USB drives [13]. Other notable attacks are Volkswagen key fob attack on 2013 and Tesla hack on 2016 [3], [25]. The vulnerability in the Volkswagen vehicle is due to the transponders in the key. Due to the vulnerability, the attacker can open and close the vehicle without the key. The vulnerability is due to lack of encryption methods between the transponder in the key and receiver in the car. As result of this lack of security, an attacker can steal a car. The whole hardware of the lock systems should be replaced to rectify this issue [3]. The vulnerability in the Tesla is due to a software issue; this leads the attacker to brake the car unintendedly, open/close the doors and windows. The issue resolved by issuing updates to the software by OTA method [25]. Given such an increasing threat landscape, vehicle manufacturers started to take automotive security more seriously and to give more importance to the prevention of such type of attacks.

Cybersecurity threats in the vehicle classified into two as per SAE J3061 standards [48]; they are cybersecurity critical threats and safety-critical threats. According to this standard, cybersecurity threats will cause financial and privacy loss, whereas, safety-critical threats will cause the loss of control in the car which will lead to damage to people and vehicles. The threats are due to

both hardware and software vulnerabilities. If the problem is due to software issues only (like GM and FCA case), then the vulnerability could be resolved by issuing patches to the vehicle systems. However, if the problem is due to the hardware (like in the Volkswagen case), the respective component should be changed completely. This landscape analysis suggests that to reduce risks due to cyber-attacks car manufactures should take cyber-security into account since the early phase of car design and development.

Several elements in the vehicle can cause Cyber-security issues. For example In-car connectivity (like On-Board-Diagnostic (OBD) - II connector, Bluetooth, Wi-Fi and other infotainment entry points) and car to X connectivity (like Radio communication, GPS, 3G and 4G telecommunication networks) [34]. To understand security vulnerabilities and attack points, the researchers should take into account the car architecture, various protocols used in the network and the other related cyber-physical functions. In this project, we will mainly focus on security issues within the in-car environment. Hence, internal architecture, communication protocols, and connectivity will be analyzed and discussed briefly in the *Chapter 2*.

1.1 Problem Statement

Currently, the threat landscape for vehicles is increasing. One of the main reasons why a vehicle is vulnerable to cyber-threat is the fact that, within the in-vehicle network, there is no authentication of messages during communication between ECUs. This opens up a wide attack surface on the vehicle system. Once an attacker is within the network, e.g., via the exploitation of vehicle-to-infrastructure communication channels, CAN bus messages on the bus can be easily intercepted and corrupted compared to other messages, e.g., to send false messages during the driving. This is possible because ECUs do not check the authenticity of the injected messages. In CAN bus the message broadcasted by a sender will be received by every other node on the bus, then the receiver will accept the message based on the identifier in the data frame. However, the identifier can be easily changed to all receivers by a malicious actor.

To compromise the safety of critical ECUs, an attacker needs to send messages in several parts of the in-vehicle network. To this end, the attacker needs to compromise the gateway of the network and to know the exact format of the message sending to the ECU. As a defense technique, automotive manufacturers are trying to isolate the critical safety functions of an ECU from the other ECUs.

In this project, we will focus on enhancing in-vehicle networks security by simulation models. By considering the traffic of messages in the in-vehicle network, a message pattern can be created for a specific (or a group of) ECUs in the vehicle architecture. For example, if we consider an ECU important for Engine functions, we want to analyze the traffic pattern of the incoming and outgoing messages in the specific ECU. The pattern must be saved and analyzed in all sort of conditions such as in normal and critical driving. Instead of taking one specific ECU, we can also take a cluster of the bus such as drivetrain bus in the architecture, record traffic of messages in the cluster and learn a pattern. By using this pattern, we can detect the cyber-attacks by assuming they will create anomalies in the pattern. Since the attack in the vehicle only takes place by injecting messages or tampering the normal message payload in the network, the abnormality of the message traffic can be detected if we know the pattern of the network. However, performing attacks directly on the vehicle will cause abnormal behavior, which will lead to harmful actions such as crashes or another unexpected outcome. To make this contained, simulating the in-vehicle network in the simulation model is the safe way. However, the robustness of the simulation model is checked by comparing the simulation and real vehicle traffic. This will ensure how much the simulation will be realistic to perform attacks on the simulation model which helps to predict the vehicle behavior during further research purpose.

Based on the problem statement, the following research questions are formulated and answered in this thesis.

1. What are the different in-vehicle network protocols and architecture used in the automotive system?
2. What are the in-vehicle network entry points for vulnerabilities?
3. What (and where) traffic, can we monitor and which are the ideal ECUs or bus monitoring points and detection capabilities?
4. How to replicate the in-vehicle network behavior in the simulation environment?

   This question can be refined into three other sub-question, which are:

   - What is the suitable environment for creating the in-vehicle network model and simulate the network traffic?
   - What are the extra model based components added in the simulation environment to replicate the real in-vehicle communication behavior?
   - How realistic will be the traffic of the real-vehicle and simulation model in the environment?
5. What are the outcomes of the attack scenarios in the simulation and what are the possible effects in the real environments based on the results obtained?

1.2 State of the art

In modern cars, vulnerabilities are higher due to the implementation of a large number of ECUs in the vehicle for different functionalities. Due to the introduction of connected cars, the communication between the vehicles and infrastructure is enormous. The attackers are trying to exploit these vulnerabilities present in the outside communication of the vehicles remotely from multiple entries present in the vehicle, i.e. TPMS, GPS, RFID key fobs and others mentioned in the *Section 2.4 in Chapter 2*. Due to the insecure communication between the Wi-Fi and cellular network inside the vehicle, the attackers can find the vulnerability in the outside communication unit and exploit it. This leads the attacker to control the car remotely [12]. The vulnerabilities in the modern cars as, by example, lack of encryption mechanism in the key fobs, can lead the attacker to gain access to inside the vehicle, allowing the attacker to exploit the in-vehicle access points such as OBD - II connector and USB/CD.

Today automotive companies facing several security problems due to the vulnerabilities present in the vehicle. However, industries are reacting to the problem by conducting the bounty program to find attack points in the vehicles before the release of the car, whereas, various suppliers in the automotive industry are developing an Intrusion Detection and Protection Systems to protect the vehicle from the hacking. For example, HARMANs ECU shield [49] developed by its subsidiary company TowerSec provides security monitoring in internal vehicle network continuously detects and log the intrusion locally in the system. The system also sends the real-time data to the cloud server via the internet connection from the smartphone or Telematics Unit present in the vehicle. In addition to this, some automotive researchers are also working in the field to protect the vehicle from the vulnerabilities[13][25].

Charlie Miller and Chris Valasek are performed their first hacking experiment in Ford and Toyota vehicles to know the vulnerabilities [5]. From the data and experience gained from the experiment, they developed an anti-hacking device. This device helps to find out abnormal message

traffic on the CAN bus. If there is an abnormality in the CAN bus, then the device will put the vehicle in *Limp Mode* and warns the driver to take precautionary actions. The *Limp Mode* will deactivate all the functions except the necessary function such as steering, engine controls, and brakes for a predefined time to make the car halt to some safe place. According to the researchers, since an attacker will always send unusual or diagnostic packets on the CAN bus to perform an attack, the device can detect it. Unfortunately, during their experiment of capturing traffic in CAN bus for 22 minutes, no diagnostic packets were seen. An attack can also be identified by observing the frequency of packets or diagnostic messages. However, the frequency of packets and type of messages flowing in the CAN bus will be different from vehicle model to model and manufacturers. In addition to that, compromising CAN bus will not happen only by increasing frequency of normal or diagnosing messages. Also, putting the vehicle in the limb mode will be dangerous while driving in the highway or massive city traffic conditions, because driver need to have some other additional functionalities to halt the vehicle in safe place.

Automotive manufacturers are considering to deploy autonomous vehicle on the roads for more safety and convenience to the user. To do this, they need to provide sophisticated in-vehicle network communication to communicate with other critical components *(such as RADARS, Sensors, and ECU's)* in high bandwidth and timing that exceed capacities of existing systems. To make this possible, automotive manufacturers face significant paradigm change in in-vehicle networks, by deploying new protocols and new vehicle Electric/Electronic(E/E) architecture. However, to design and evaluate the new protocols, architecture design concepts, the suitable toolchain is required to know the performance of the system. In this article [50], the researchers used the event-based simulation OMNET++ toolchain to evaluate the network architecture and protocols. In their work, they performed the simulation model for real-time Ethernet and Fieldbus technologies such as CAN & FlexRay. However, their primary focus is integrating the Fieldbus technology with the Ethernet. To achieve this, the researchers created an open-source toolchain for in-vehicle networks on OMNET++, which is based on the Eclipse IDE (Integrated Development Environment). As a first step, they created a model suite called CoRE4INET[37] for Ethernet technologies and FiCo4OMNeT[38] for Fieldbus technologies. To integrate the both Ethernet and Fieldbus, they developed the Signals and Gateway suites. By developing this module, they can create an in-vehicle network architecture and able to test their performance. However, the open-source suits provided by them cannot be able to record the in-vehicle traffic performance in the real time, when the user chooses to use only one suite such as FiC04OMNeT. In addition to this, the simulation models cannot be able to replicate the dynamic behavior of the vehicle such as while driving the car.

As discussed above, automotive manufacturers are considering Ethernet as the backbone for next-generation vehicles. So the researchers are focusing on developing the simulation model based on the Ethernet. Another research [24] focuses on finding out the latency of the CAN message sent from CAN bus to Ethernet AVB gateway. The research was done to prove that the efficiency of the Ethernet-based architecture will guarantee a high bandwidth for the critical real-time systems in the next - generation vehicles. To evaluate this, they perform created a simulation model in the OMNET++ environment to evaluate the performance of the system.

However, in the point of securing the in-vehicle network, both existing vehicles and next-generation vehicles should be protected. As discussed in this articles [24][50], they are creating a simulation model and evaluate the networks for the next generation vehicle. As we do not know exact timeframe of deploying these types of networks in the real-vehicle, until then, the existing architecture using Fieldbus *such as CAN, FlexRay, and LIN)* will be used. To secure this network protocols from cyber-attacks, the developing the simulation model in a simulation environment to know the vehicle behavior during the attacks in quite important. In this thesis, this issues is addressed briefly as explained in the above section 1.1.

1.3 Thesis Outline

The report is organized as follows.

Chapter 2 describes background information about the internal automotive networks and external interfaces and explains about the different attack scenarios to gain access to the modern vehicle. It provides information about the simulation software used in this project to simulate the attack in the in-vehicle network system.

Chapter 3 provides information about the various software tested for simulation in-vehicle network and more details about the chosen software for this thesis to carry out the in-vehicle simulation.

Chapter 4 describes the methodology for simulating the in-vehicle networks in the OMNETPP++ software. The chapter gives background knowledge of each component used in the simulation environment. It also provides information about the experiments carried out in the Toyota Prius to gain real-time CAN data. From this data, The simulation environment (OMNETPP) can replicate some behavior to perform attack scenarios.

Chapter 5 presents the results of the experiments such as attack scenarios carried out in the simulation environment(OMNETPP). From this results, the car behavior can be predicted if the attack replicated in the real environment.

Chapter 6 concludes this thesis and explains the answers for the research questions asked above in this chapter.

Chapter 2

Background

2.1 Vehicle Network Architecture

In a modern vehicle, the manufacturers are using more than 20 Electronic Control Units (ECUs), 50 sensors and actuators and more than one serial bus communication for different vehicle functions. These combined aspects of different infrastructure in vehicles are known as automotive Electric/Electronic (E/E) architecture. As discussed in Chapter 1, the number of ECU's and signal transmitting in the system is increased dramatically over the years. In coming years, it will further increase to adopt new systems (such as Advanced Driving Assistance System (ADAS) for Autonomous Driving). The Figure 2.1 shows the evolution of the E/E architecture of Mercedes E-Class series over the years. As shown in the Figure 2.1 the number of signals is increased from 0 to 10000 over the period of 40 years [26].

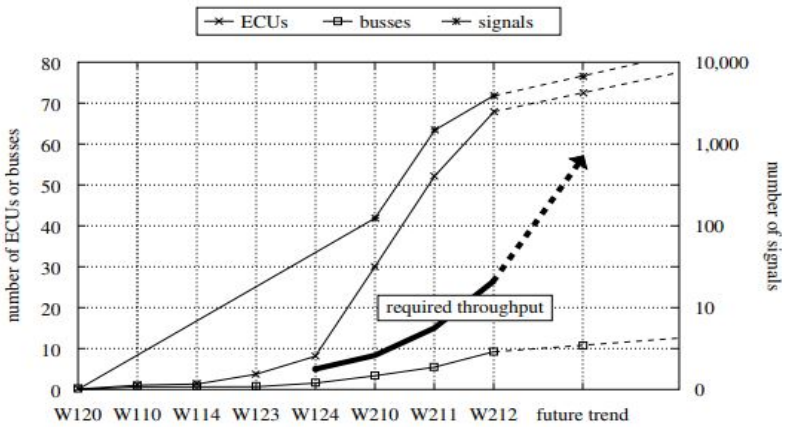


Figure 2.1: The Evolution of E/E Architecture Trend of Mercedes E-Class [26].

In a vehicle the communication between different ECUs, sensors and actuators can be achieved using serial bus system, according to Original Equipment Manufacturer (OEM) requirements and functions. Commonly used serial bus communication protocols to interconnect ECU's are Controller Area Network (CAN) [19], Local Interconnect Network (LIN) [46], FlexRay [8], Media Oriented System Transport (MOST) [9], and Automotive Ethernet [7]. Each of this protocols are discussed briefly in the section 2.2. By introducing E/E architecture, the possibility of reusing sensor information and other ECUs data are easily shared within the vehicle systems. By doing

so, the scalability of the system application is increased, and wiring harness of the vehicle will be reduced, Since the wiring harness cost will be reduced, which is the 3rd costliest part in manufacturing vehicle.

The vehicle E/E architecture will be different for each OEMs and vehicle models. From the point of view of topology of the in-vehicle network, vehicle E/E architecture network classified into two types: i) simple and ii) advanced in-vehicle network. The simple network architecture will be using one or two serial buses to communicate between the ECUs in the vehicle [51]. In a simple in-vehicle network, CAN and LIN protocols are used to interconnect the ECU's with the help of one centralized Gateway as shown in figure 2.2.

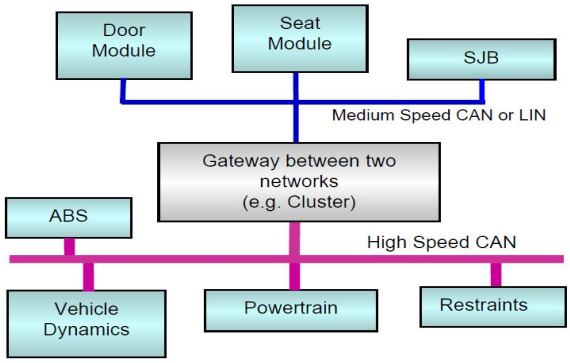


Figure 2.2: Schematic diagram of Simple in-vehicle network [51].

In Figure 2.2 several important ECU's (such as Anti-lock Brake System(ABS), Vehicle Dynamics, Powertrain) are connected to the High speed CAN bus, whereas basic functions (such as Door module, seat module) connects to the LIN bus. Then, the two buses are interconnected to the gateway. Which ECU's are interconnected to the respective buses is based on functionality and importance. For example, Powertrain ECU is a critical ECU which sends information to the other ECU's in real time for smooth function of the vehicle, therefore, it is connected to the High speed CAN bus. However, Door module ECU is a noncritical ECU used only in few scenarios, therefore, it is connected to the LIN bus.

In an advanced in-vehicle network, several protocols, as shown in 2.2 will be used to adapt various features such as Lane Warning system, Intelligent Parking, Collision avoidance system, adaptive steering, etc. More than 30 ECU's used by OEM's and several serial buses will be created based on the network protocols, and functions that interconnect the buses to the ECU's to make a modern vehicle. After that, all buses connect to a centralized Diagnostic Gateway. This gateway is used by the OEM's for repair and software upgrade for the ECU's via On-board Diagnostic (OBD) Port. By partitioning the buses, the buses are isolated such that, in the case of an attack in the vehicle network the effect will be lower compared to the simple in-vehicle network [36]. The picture of the advanced in-vehicle network shown in Figure 2.3.

Figure 2.3 shows various serial buses interconnected with the ECU's form a standalone cluster. Each cluster will have an intelligent switch which connects the respective bus and the ECU's. By isolating the critical ECU's in the vehicle system, compromising the whole in-vehicle network will be highly unlikely [58].

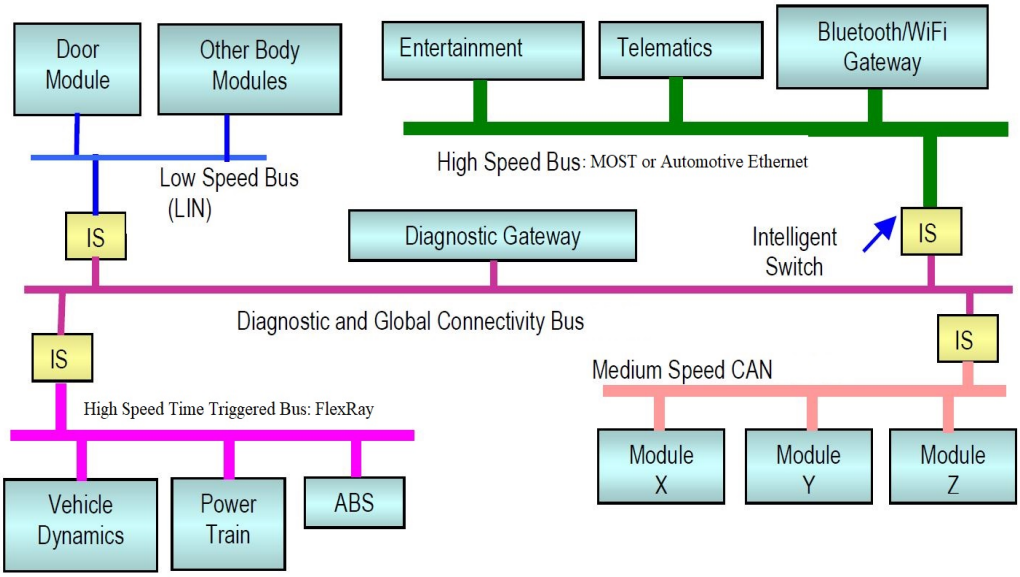


Figure 2.3: Schematic diagram of Advanced in-vehicle network [51].

2.2 Protocols used in Vehicle Architecture

There are different protocols used in the vehicle architecture for communication between the ECUs. Society of Automotive Engineering (SAE) has classified this communication protocols based on the transmission speed and functions.

Table 2.1: Classification of protocols based on SAE Standards.

Class Type	Data Transmission Speed	Protocols
Class A	Less than 10Kbit/s	LIN
Class B	10Kbit/s to 125Kbit/s	Low Speed CAN, PSI5, SENT
Class C	125Kbit/s to 1Mbit/s	High Speed CAN
Class D	Greater than 1Mbit/s	MOST, FlexRay, Automotive Ethernet

The protocols listed in the table are used in different type of vehicle functions based on their needs and each protocol has a unique mechanism to deliver the data to the respective ECUs. The protocols will transmit the message based on a time-triggered or event-triggered event or both.

- Event Triggered: In an event triggered protocols will transmit the message based on the occurrence of a specific event. For example, when the driver brakes the vehicle or turn the indicator ON or turn the lights on, the message will be sent to the ECU's to activate the system. CAN protocol is an Event triggered protocol
- Time Triggered: The message frames will transmit at predetermined time slots allocated by the OEM in the architecture in a time-triggered protocol. The systems which are critical and time sensitive will use the time triggered protocols. For example, the wheel speed of the vehicle will be updated every 10ms in the Toyota Prius.

For this project, CAN protocol is used widely in the simulation and reverse engineering process. In the next paragraphs the background of the CAN protocol is described in detail, and other protocols are described briefly.

2.2.1 Control Area Network (CAN)

CAN [19] was developed by BOSCH and first implemented in BMW 8 series vehicle in 1988. CAN is used to reduce the conjunction of wiring harness between the ECU and flawless communication between the ECUs. After the implementation, BOSCH released several version of CAN and ISO release the CAN standard in several parts (ISO 11898-1 to 4). CAN bus has different signal transmission rate as mentioned in the ISO standards 11898-2 (High-speed CAN bus) and 11898-3 (Low-speed CAN bus). Low-speed CAN bus limits the transmission rate up to 125Kbits/s whereas High-speed CAN bus has a maximum transmission rate up to 1Mbits/s. The most recent version of CAN protocol is version 2.0 published by BOSCH[19].

CAN is a multi-master serial bus architecture and event-driven message system. Even today, OEM is considering CAN as the most reliable data transmission protocols for real-time requirements. The CAN should consist of at least two or more nodes to start a communication between the ECUs with the help transmission medium (CAN bus). CAN bus is made up of unshielded twisted pair (UTP) cable and it can extend up to 40 meters with consisting of 32 nodes. The unshielded twisted pair wire (UTP) will consist of CAN High wire and CAN Low wire [21].

The CAN bus specification is consists of two parts A & B. To be compatible with CAN specification, the implementation of the CAN should be either PART A or B. Part A describes the CAN original message format, whereas, Part B describes both standard and extended Message formats.

CAN bus Frame Formats

They are two different frame formats used in the CAN protocol as mentioned in the Part B CAN specification which is Standard CAN Frame and Extended CAN Frame. The frame formats are

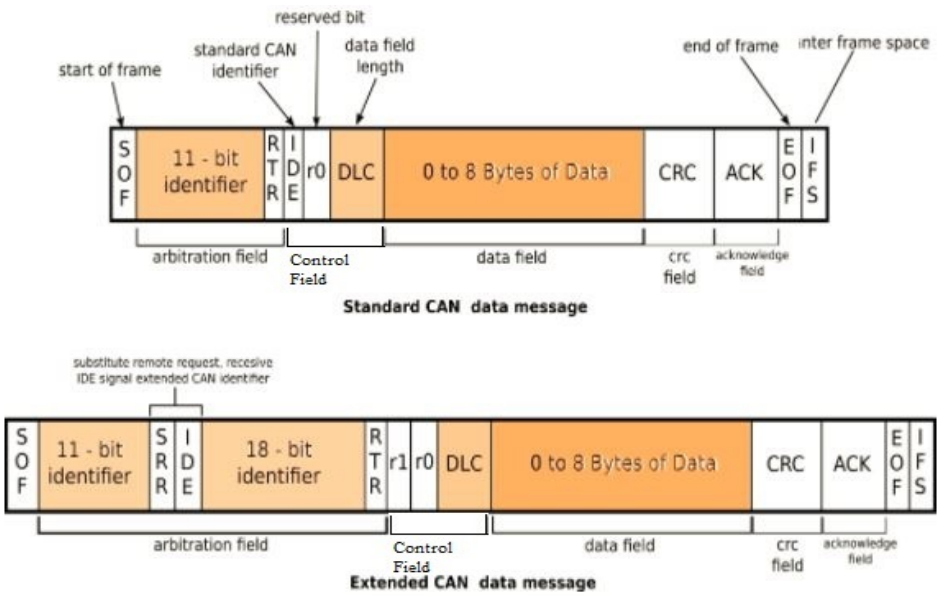


Figure 2.4: CAN Bus Frame Format for Standard and Extended Data Frame [19].

differentiated based on the length of the Identifier of the frames. The standard frame will consist of 11-bit Identifier, whereas, the extended frame consists of the 29-bit identifier. The Figure 2.4 shows the difference between the CAN Data Frames.

The CAN message in the Figure 2.4 consists of Start of Frame (SOF), Arbitration Field, Control Field, Data Field, CRC Field, Acknowledge Field, End of Frame (EOF).

- Start of Frame (SOF): SOF marks the beginning of Data Frames and Remote Frames. It consists of single 'dominant bit' - '0'.
- Arbitration Field: This field is different for the standard and extended frame. In this field, the 11-bit identifier will be transmitted first. Following that RTR (Remote Transmission Request bit), in which if it is a data frame RTR bit will be dominant - '0', whereas, if it is a remote frame RTR bit will be recessive - '1'. In an extended CAN frame, SRR (Substitute Remote request bit) is transmitted in the position of RTR bit to resolve a collision between the standard and extended frame. IDE bit (Identifier Extension) belongs to control field in standard format and arbitration field in an extended format. It will be a dominant bit in a standard format and recessive bit in an extended format.
- Control Field: It consists of six bits. In the standard frame, it consists of IDE and r0(a reserved bit) and DLC (Data Length Code). DLC consists of a number of bytes in the data field. In contrary to standard frame, the extended frame will consist of two reserved bits (r1 & r0) and the IDE will be in arbitration field instead of the control field.
- Data Field: It contains data to be transmitted to the one ECU to other. It will contain up to 0 to 8 bytes.
- CRC Field (Cyclic Redundancy Code) - CRC field consists of CRC sequence and CRC delimiter. CRC sequence is a frame check sequence to identify whether the receiver receives the frame. The transmitter calculates the CRC sequence and sent the value in the CRC sequence field. The receiver will check the CRC sequence value; if both the transmitter and receiver values are identical, then it will send a dominant bit to the transmitter during Acknowledgement field. CRC delimiter will always be a recessive bit, which will be after the CRC sequence field.
- ACK field (Acknowledgement field). ACK field consists of ACK slot and ACK delimiter. In ACK field, transmitting station sends two recessive bits. If the receiver received the message correctly, then it will report to the transmitting station by sending to dominant bit during ACK slot. ACK delimiter will always be a recessive bit followed by an ACK slot.
- End of Frame (EOF)- A flag sequence delimits the data frame, and the remote frame consists of seven recessive bits.

Data Frames and Remote Frames consists of the parameters explained above, whereas, the error frame and overload frame consists of different fields. The upcoming section will explain the CAN bus frame types.

CAN bus Frame Types

Message transfer in the CAN bus can categorize into four different types, such as:

- Data Frame

  Data frame helps to carry data from a transmitter to the receivers. Data Field consists of seven different fields: Start of Frame (SOF), Arbitration Field, Control Field, Data Field, CRC Field, ACK Field and End of Frame(EOF). Data Field in the frame can consist of data from 0 to 8 bytes in a single frame.

- Remote Frame

  When a node wants a data from the other node, it can send the Remote Frame to the respective node. The respective node will send the data frame. The remote frame consists of six different fields: Start of Frame (SOF), Arbitration Field, Control Field, CRC Field,

ACK Field and End of Frame(EOF). Contrary to the data frame, the RTR bit in arbitration field is recessive in the remote frame.

- Error Frame

  Error frame will consist of two different fields. The first field is a superposition of Error flags and Error delimiter. There are two forms of error flags: Active error flag and a passive error flag. The active error flag consists of six consecutive *dominant bits* , whereas, the passive error flag consists of six consecutive *recessive bits*. The error delimiter consists of eight recessive bits.

- Overload Frame

  Overload frame consists of two different fields: Overload flag and overload delimiter. They overload frame will transmitted in three overload conditions in the bus:

  - The internal condition of the receivers which leads to delay the data and remote frame.
  - Detection of a 'dominant' bit at the first and second bit of the intermission.
  - If a can node receives a dominant bit at the eight bit(last bit) of an Error or overload delimiter, it will transmit the overload frame.

  Overload delimiter consists of eight recessive bits. '

- Interframe spacing: Data and Remote frame are from preceding frames by a bit field called Interframe spacing. Interframe space consists of bit fields called Intermission and Bus Idle and for 'error passive stations' which have been a transmitter of the previous message will have suspended transmission first.

  Intermission consists of three recessive bits. During intermission field, the only action signaling is overload condition. No station is allowed to start transmission of data or remote frames actively.

  Suspend Transmission, after an passive error station sends the eight recessive bits following intermission field, before starting to transmit a further message of recognizing bits to idle. If meanwhile, other station starts the transmission, the station will become the receiver of this message. Bus idle is the period of arbitrary length. The bus is recognized to be free, and any station can access the bus to transmit the message. A message is pending during the transmission of other message transmission is started first bit following the intermission. This will be explained in the upcoming section more clearly.

Message Filtering and Validation

Message filtering is based on the identifier of the CAN data frame. In CAN bus, the message transmitted by the node will receive by all nodes in the bus. However, some nodes in the bus do not want to process every message transmitting in the bus. To address this message filtering is implemented in the nodes either by hardware or software level. By doing this, the receiving node can discard the CAN frame. In hardware level, the nodes have a present programmable identifier that is accepted. If not of the identifiers are matched with this programmable identifiers, the message will be rejected by the node.

The message validation will be different for transmitter and receivers of the message.

- Transmitter: The message is valid for the transmitter if there is no error until the EOF.

- Receiver: The message is valid for the receivers if there is no error until the last but one bit of the End of the Frame.

Message Arbitration

In CAN bus communication, every node will receive every message transmitted through the bus, which is called as broadcasting. During broadcasting of messages, there is that possibility the two nodes send messages simultaneously which will be a problem. This situation is avoided by using the CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) methods as mentioned in ISO 11898-1. This method helps the bus to perform the bitwise arbitration method and allow the highest priority message in the CAN bus to go first. The message arbitration in the CAN bus makes it more efficient.

In message arbitration, the important to note is the dominant bit - '0'and recessive bit - '1'. The nodes will start to transmit the message when the bus is idle. Otherwise, it will wait until the bus becomes available. When two nodes start to transmit with different bit such a '0' & '1', the '0'bit will be placed on the bus. The node with dominant bit starts to transmit the frame. However, when two or more nodes start to transmit the exact bits in the bus, the nodes will not know the collision until it places the recessive bit on the bus. When the recessive bit is placed on the bus, the error will send to the sender of the node. The sender will stop the transmission and retransmit the packet when the bus becomes idle. During this, the sender of the dominant bit will not know the collision with another node; this makes the CAN bus more efficient. An example of arbitration shown in the Figure 2.5.

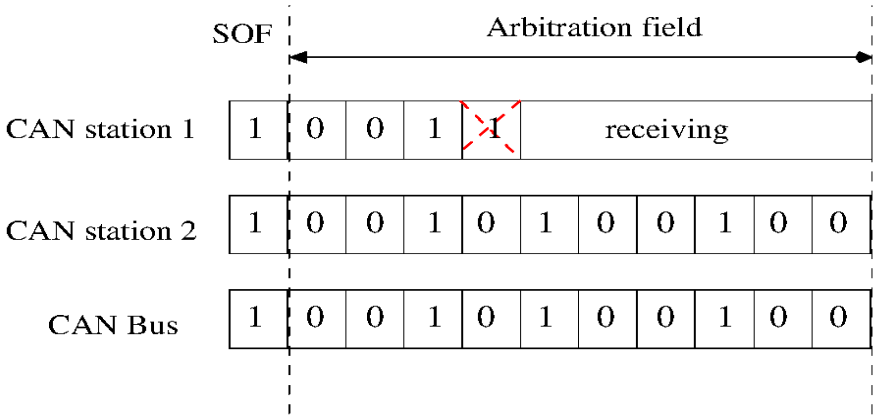


Figure 2.5: CAN Message Arbitration
[30].

Error Detection

The CAN bus robustness is attributed due to the error checking and signaling. When any node in the network detects an error, it will send the active error frame. They are five different types of error in CAN bus, such as:

- Bit Error: The transmitter in the bus which is sending the unit bits also monitor the bus. When the transmitter detects the bit value is different from the bit value sent, bit error will be detected by the transmitter and stops the transmission in the bus.

- Stuff error: The bit stuffing should be known to understand stuff error. The bit stuffing is the process when the five bits transmits with the same value; the sixth bit should be an opposite value. So when the module detects the six bits with the same value, then it will be stuff error in the message.

- CRC Error: The CRC sequence consists of the CRC calculation by the transmitter. The receiver also performs the CRC calculation. If the CRC calculation is not identical with the transmitter, then the receiver will send the CRC error.

- Form Error: When the receiver received the CAN packet, it will check it has received all field with the correct size. If it not received correctly, then it is known as Form error.

- Acknowledgement Error: The transmitter monitors the ACK error, whenever it does not receive the dominant bit in the ACK slot from the receiver, then it is known as Acknowledgement Error.

2.2.2 LIN(Local Interconnect Network)

LIN is used in the automotive network that does not require higher bandwidth networks such as CAN, MOST or FlexRay. The implementation cost is also low compared to other protocols [46] [21]. LIN has data transmission rate up to 20Kbit/s and uses a single wire for the connection between the nodes. LIN protocol is widely used widely for the comfort functions of the vehicles such as doors, windows, power seats. LIN communication is based on time-triggered approach. LIN cluster consists of master and slave nodes. The master node will be connected to the other protocols such as CAN or FlexRay and schedule the timetable for the messages. The system designer does the scheduling in the LIN bus during the development based on the type of message sent by the nodes. The LIN bus scheduling is organized with slots based on message transmission time. The slots are divided into mini-slot in which master node process the schedule for the communication. While scheduling, the Master node is not able to fill the mini-slots, the rest of the frame time will wait in addition to the jitter time for next message transmission. Frame Slot size will be determined by the message type sent by the Master node. This will be predefined by the system designer.

2.2.3 MOST(Media Oriented System Transport)

MOST is a multimedia network specially developed by automotive industry for high sped multimedia data such as audio, video and control. The most network has several bandwidths like 25, 50 and 150 Mbit/s [9]. The MOST device consists of three parts like physical interface, network service, and function blocks. MOST physical layer uses plastic optic fiber (POF) links for interconnection; however, it also uses an unshielded twisted pair (UTP) and coaxes cables in some lightweight applications. The function blocks (FBlocks) is responsible for MOST network and the application. They are three types of FBlocks such as controllers, slaves and Human Machine Interface (HMI). In a MOST network, one of the nodes will be called Timing Master which is master of the network (Controllers) that initiates the message in the network, while other connected nodes in the network are called slaves. In MOST, data transfer will occur in the form of a block. In each block, there will be 16 frames, where each frame consists of 512 bits in MOST 25, 1024 bits in MOST 50 and 3072 bits in MOST 150 [14].

In each frame, there are three communication channels in MOST frame such as: control channel for event-oriented transmission, an asynchronous channel for packet-oriented transmission for large block size and synchronous channel for continuous data streams. Control channel uses CSMA to access the channel and messages are sent by the applications or administrator. Synchronous data uses TDMA to allocate the channels, and asynchronous data uses token between the devices to allocate the channels. Asynchronous data will contain large packet data with high data transmission such as TCP/IP transmissions. The asynchronous packet data will be specified to data frame using the delimiter, by this, we can change the length of synchronous and asynchronous data based on the transmission length [21].

2.3 Toyota Prius

For this thesis work, 2010 Toyota Prius model used for reverse engineering and the vehicle network architecture is used for simulation. The reason for choosing the vehicle is that, all the functions (such as steering, transmission, engine, power management) are controlled by drive by wire systems with the help of Electronic Control Unit (ECU). The vehicle functions can be easily controllable and increase the operational accuracy by the drive by wire systems compared to the other cars where some functions use the mechanical or hydraulic system for some functions [40].

The ECU is responsible for the communication of the drive by wire systems in the form of electrical messages. For this different communication protocol types are used depends on manufacturer preference as mentioned in 2.2. Due to manufacturer proprietary and safety of the vehicle, The communication between the ECU systems and vehicle architecture are not available on the Internet. The reverse engineering process is carried out as mentioned in *Chapter 4* to identify the types of communication protocols (such as CAN, LIN) and the messages transferred between the ECU's. To understand the CAN messages of the vehicle such as which systems should be communicating to another, all the ECU functions should be studied carefully, which will help to map the vehicle function with respective CAN ID's. Since the Prius is the hybrid model, the functionality of the ECU's is different compared to the conventional car. For example, in the conventional car, the accelerator system either will be a mechanical or electric one which will connect to Engine ECU. However, in this Toyota Prius model, the acceleration system is connected to the power management ECU which will transfer the acceleration pedal value to the engine management ECU [43]. So understanding the hybrid system in this vehicle is an important task to know the purpose of CAN messages in the vehicle. The detail information about the engine and hybrid system will be discussed in the next paragraphs.

2.3.1 Types Hybrid System

The level of hybridization in the vehicle is differ based on the capacity of the engine, integration of battery in the vehicle, and electric motor. Different types of hybrid levels are known as the micro, mild, full and plug-in system. The mild, full and plug-in hybrid system will also act as a regenerator (i.e.) when the brake is applied the kinetic energy will be converted to electrical energy and the energy will be stored in the battery which is called as regenerative braking. Micro-hybrid system helps to start/stop the vehicle with integrated alternator and starter. By using this system, the vehicle efficiency is increased by 3-5 %. The mild hybrid system is an advancement of the micro-hybrid system which consists of stronger electrical components. In addition, to start and stop function, the mild hybrid helps for vehicle propulsion, which helps to increase the fuel efficiency by 15-20%. The full hybrid system is an advancement of the mild hybrid system, which helps to drive the vehicle only with electricity for few kilometers depends upon the battery capacity. When using the full hybrid system, the engine size can be downsized. However, the full hybrid system will compensate the power lost due to the engine downsizing. In this full hybrid system, the improvement of fuel efficiency is up to 20-30%. The plug-in hybrid vehicle(PHEV) is similar to the full hybrid system but uses larger electrical components. The vehicle can be driven with the help of electrical energy only for up-to certain Km/s [4][29][1]. The vehicle used for this thesis is a PHEV.

There are two main types of hybrid cars - Series and Parallel [29].

- Series Hybrid system - The electric generator will be coupled with the ICE. This generator will be used to charge the batteries and drive the electric motor in the system, which helps to drive the transmission of the vehicle. In this system, the power of the engine will never go directly to the wheels. Therefore, the during the convention process there will be a loss of energy in the system.
- Parallel Hybrid System - In this system both ICE and electric motor will drive the transmission at the same time. By using this system, the ICE can directly transmit the power

to the wheels, and at the same time with the help of a generator, it can charge the electric battery. By doing this, the efficiency of the vehicle will be quite high compared to the series hybrid system. Toyota Prius using in this project is a parallel hybrid system.

2.3.2 Hybrid Synergy Drive(HSD) of Toyota Prius

Toyota Prius model is a front wheel drive vehicle which is connected to the 98hp 4-Cylinder internal combustion engine (ICE) coupled with the 36hp motor powered by the nickel-metal hybrid battery pack; the system has a total power output of 134hp. The hybrid system in the Prius is called as Hybrid Synergy(HSD) System which includes an ICE, two electric motors/generators, power split device, battery, and two inverters [39].

In conventional vehicles, the powertrain system helps to drive the vehicle wheels with the help of the drive-shaft. The ICE engine will produce the torque an power which will be connected to switchable gearbox system. With the help of gearbox, the driver will send the required torque and power to the drive-shaft of the vehicle[43][56].

However, In HSD conventional systems (such as gearbox, alternator, and starter) are replaced by the electric Motor/Generators(MG), a computerized shunt system, a mechanical power splitter that acts as differential and a battery pack. The Electric Motor/Generator (MG1 & MG2) helps to convert electricity into motion and vice-versa. The following picture2.6 shows the HSD system of Toyota Prius[31].

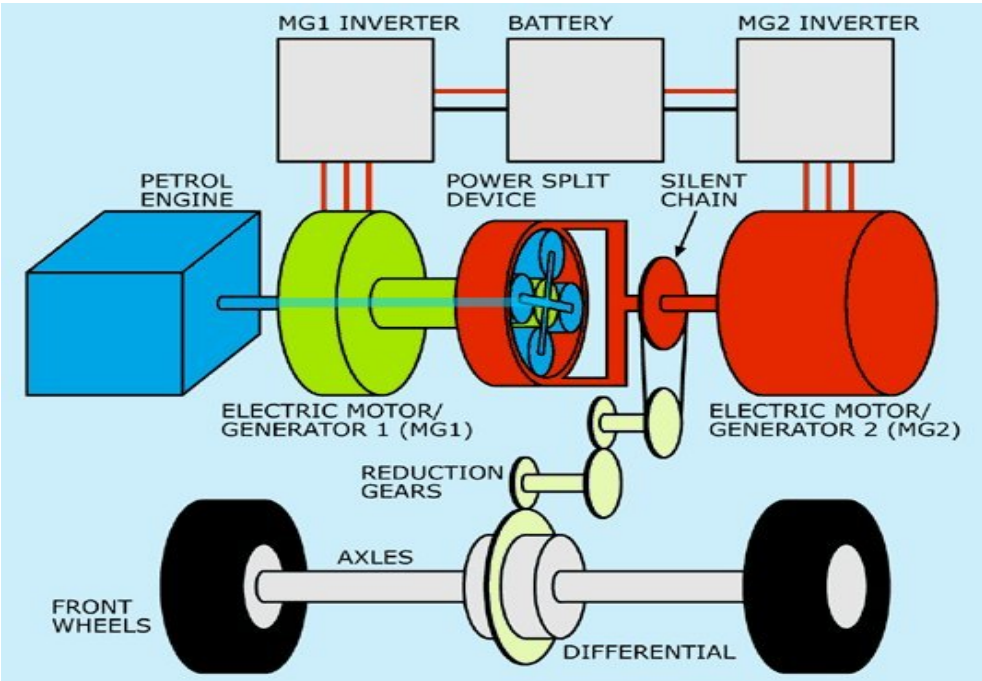


Figure 2.6: Schematic diagram of Prius Hybrid Synergy Drive(HSD) [31].

In HSD, Power Management Control ECU is responsible for oversees the operation of the hybrid system. The ECU will determine when the power from the ICE engine and motor should be used to drive the wheels efficiently. The hybrid system in the vehicle will be working in different phases [52][39][29] which will be explained below.

- Engine Start Phase

During Starting the vehicle, the MG1 will act as a starter and start the vehicle. MG1 will draw the power from the battery and start the vehicle. The power from the motor helps to start the ICE engine in the later phase. This will perform more efficiently than the conventional ICE vehicle.

- Only EV

  At slow speeds, the vehicle can run without ICE power. During this phase, the electricity is supplied to the MG2 which will act as a motor. The MG2 will drive the wheel only with electric power allowing the vehicle to run without fuel for some miles.

- Low Speeds

  In Low speeds, the ICE crankshaft will rotate in high speed compared to the wheels which will not develop sufficient torque. So the ICE power will be fed into the MG1 (*will act as generator*), and output of the MG1 will be coupled with the MG2 (*will act as motor*). The MG2 will drive the wheels with sufficient torque.

- Cruising: ICE Power + Battery Charging

  In this phase, ICE power will be work more efficiently than the electric power. The power from the ICE will directly drive the wheels of the vehicle. During this period, the part of the ICE power will be transferred to MG1(*will act as generator*) with the help of Power split device. The electricity generated from the MG1 will be stored in the battery for the later use.

- Full Acceleration + Battery Power

  In this phase, the ICE Power will directly drive the wheels, in addition to that, the energy stored in the battery will be used by the MG2(*will act as motor*). The MG2 will provide supplement energy in addition to the ICE power during the full acceleration phase (which can be used in overtaking or driving through steep hills). The extra torque and power will be produced by the MG2 till the battery power is exhausted.

- Deceleration/Regenerative Braking

  In a conventional vehicle, during braking, the kinetic energy will be lost as friction and heat. Whereas, in HSD the kinetic energy will be converted into electric energy and stored in the battery. During this phase, the MG2 (*will act as generator*) is powered by the wheels and recharge the battery. This process will be followed during the deceleration the vehicle(i.e., lifting the accelerating pedal).

- Reverse Gear

  As a conventional vehicle, there is no reverse gear in the powertrain. Instead of that, the ECU will reverse the phase sequence of the current in the MG2 (*will act as motor*). When the battery is low, the ICE will power the MG1(*will act as generator*), and the electric energy will be used by the MG2 to reverse the vehicle.

2.4 Automotive Attacks

As discussed in *Chapter 1* the vulnerabilities of the car is seemingly increasing due to the addition of new features. Due to this, the attackers are targeting vehicle to exploit new attack methods and entry point to get access tot he vehicle. The attacks are ranging from turning on/off lights, Air-Conditioning to gain access to the crucial system such as brakes, transmission and steering wheel. This depends upon each E/E architecture of the vehicle [28]. However, some vehicles are still using the mechanical/hydraulic systems for the brakes and steering; the attackers are not able to compromise the system.

To be able to make attacks in the automobile, the attackers need to perform numerous steps, such as *What are all the entry points in the vehicle?, What are all the E/E components can be compromised and How they are interlinked?, What is the message structure and identifiers used in the vehicle?*. For this thesis, to make attacks in a simulation environment, we need to know the things mentioned above. This will be addressed in the following sections below briefly.

2.4.1 Entry points for attacks in the vehicle

Modern automobiles use heterogeneous E/E architecture as we defined in section 2.1 to provide broad range of functionality ranging from Electronic steering, braking, Advanced Driver Assistance System (ADAS) and various entertainment features inside the vehicle. It also created the opportunity for new attack surfaces compromising critical safety systems of the vehicle. In E/E architecture, they are a lot of external interface in the vehicle to provide functionality ranging from comfort to safety (such as OBD -II helps to run diagnostics on the vehicle to find faults and update the vehicle software, whereas the Wi-Fi helps the passenger to connect to the Internet.). The external interface in the vehicle is the entry point for the attackers to compromise the vehicle function. Such entry points are mentioned in the below Table 2.2 [54]. Each entry points will be detailed briefly in the following sections.

Table 2.2: Entry Points for cyber-attacks

	Interface	Physical Range
Wired Entries	OBD-II Connector	0m
	CD/DVD/USB/SD Card	0m
Short Range Wireless Entries	Passive Anti-Theft System (PATS)	∼10cm
	Bluetooth	∼10m
	TPMS	∼ 10m
	Remote Keyless Entry (RKE)	∼20m
	Wi-Fi	∼100m
	V2X	∼300m
	RADAR/Ultra Sonic Sensor	∼100m to 300m
Long Range Wireless Entries	Radio	>10km
	GPS	>10Km
	Cellular	>10Km

Wired/Physical Entries

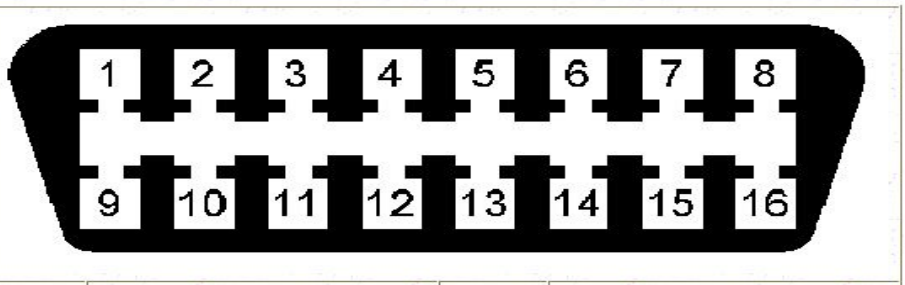
- On-board Diagnostics - II

  The on-board diagnostic system in the vehicle provides the direct physical access to the in-vehicle network of the vehicle. When attacker gain access to this system, he can perform various types of attack [60]. In this thesis, for reverse engineering purpose, the attack simulations are done based on the OBD systems. A detailed study is conducted on OBD systems.

  On-board diagnostic (OBD -I) system introduced in early 1980's, used to reduce the emission level by monitoring the Engine Control Unit. During this period, all manufacturers have their systems and OBD connectors to monitor the vehicle. Due to more adaptation of OBD systems, Society of Automotive Engineering (SAE) set a standard connector plug and requirements of diagnostic test signals from the vehicle in 1988. To meet this standard requirement, manufacturers turned the electronic control fuel feed and ignition. Sensors measures engine performance and reduce the emission and diagnostic assistance to the vehicle related to the engine. Later, OBD systems became more sophisticated in more functions and introduced a new standard (OBD -II) in the mid-1990's. After OBD - II standard is introduced, it is mandatory for all vehicles manufactured in the world. In addition to that, SAE

Standard J1979, J1962 defines many OBD - II PIDs(Parameter ID's). From these IDs, user can request data from vehicle such as: diagnostics and Engine Data. Nowadays, OBD -II systems used for many purposes such as *diagnostics, performance tuning, ECU's firmware updates and listen to the internal networks for reverse engineering purpose.* In the early 2000's, OBD - II system accessed only by the Scan tools provided by the manufacturers. Later, manufacturers are adapted to change the system to more PC-centric with specific tools and software such as BMW ICOM, Toyota Techstream Lite with Mangoose Pro OBD - II toolkits, Mercedes Xentry Connect, by this tools mentioned above all the functionalities of the OBD - II can be achieved. Some other tools provided by third parties such as CANtact, Versus Edge and ELM 327 are used for the same purpose, However, their functionalities are limited [16] [17]. In this thesis, OBD -II port is used to listen to the CAN bus traffic in the network and simulate the attack scenarios in the simulation. OBD -II connector has a different layout for different manufacturers. Since, Toyota Prius vehicle used for this thesis, the pin layout of the OBD - II connector (*is also known as Data Link Connector (DLC)* shown in 2.7.

The Figure 2.7 shows only the CAN protocol. So the raw CAN traffic is seen without



PIN	DESCRIPTION	PIN	DESCRIPTION
1	Vendor Option	9	Vendor Option
2	J1850 Bus +	10	j1850 BUS
3	Vendor Option	11	Vendor Option
4	Chassis Ground	12	Vendor Option
5	Signal Ground	13	Vendor Option
6	CAN (J-2234) High	14	CAN (J-2234) Low
7	ISO 9141-2 K-Line	15	ISO 9141-2 Low
8	Vendor Option	16	Battery Power

OBD-II Connector and Pinout

Figure 2.7: OBD - II pin-layout of Toyota Prius
[31].

any interference. The other protocols can be accessed via OBD -II protocol based on the manufacturer option, not directly. If a manufacturer wants to listen to the specific protocol, they can assign one of the specific pins for the new protocols. Nowadays, Modern vehicles such as BMW, Audi, etc are providing RJ45 Ethernet Pin, in addition to the OBD - II connector. By this pin, with appropriate manufacturers software and tools, one can listen to all the traffic of heterogeneous network that consists of the CAN, LIN, FlexRay, MOST and Automotive Ethernet protocols.

However, many security researchers compromised the in-vehicle networks via the OBD - II physical Interface to evaluate the in-vehicle network security[6][5][13][47][27]. Since attacker need to get access to go inside the vehicle, manufacturers are considering OBD - II as less vulnerable surface. However, vehicle manufacturers are using the OBD - II port to find the vulnerabilities in the vehicle E/E architecture using the researcher. However, Modern

vehicles E/E architecture made the things complicated. The in-vehicle network of the vehicle directly connected to the OBD - II port. So, if attackers compromise any other external interface as mentioned in Table 2.2, they can easily get access to the other system via the OBD system. In this research [27], the researchers connected the OBD - II connector with the third party tool (*ECOM*) and connected to the computer which has an Internet connection. In another car, the researchers are able to control the vehicle remotely and perform some attacks on stationary and moving vehicle. Such as *temporary, increase of RPM, engaging the brakes, increase the idle RPM, Turn ON/OFF (Wipers, Lights, Braking Lights), open the trunk and falsify the speedometer readings.* From this research, the vulnerability of the OBD -ii port is understandable. To perform this types of attacks, the researchers perform the reverse engineering of the vehicle network to know the message transmitting between the systems. In this thesis, the network traffic is simulated, and attack is performed based on the data collected during the reverse engineering process.

- USB/CD/SD Card

  Modern cars entertainment system consist of USB/CD/SD card; the entertainment system is connected to the vehicle network of the vehicle to gain functionalities such as Steering Control Audio Systems and control the system via smartphones. Due to this additional features, entertainment system in the vehicle act as an entry point for an attacker. If an attacker has pre-installed the exploit on SD cars/ USB and inserted in the entertainment system, it can compromise the vehicle. The vulnerabilities of this system are same as OBD - ii port.

Short-Range Wireless Interface

- Passive Anti-Theft System (PATS) and Remote Keyless Entries (RKE)

  All modern vehicles contain the small chip in their key. That will transmit the code to the specific ECU to start the vehicle. In Toyota Prius, *Certification ECU* will transmit the registration RF signals via LIN bus to the *ID Code Box ECU and Transmission ECU.* After this, *ID code box ECU* will send the engine start permission and other registration RF signals to the *Power Management ECU* to start the vehicle. This all happens in less than a second. If one of the above-mentioned ECU does not get the proper signal from the *Certification ECU or ID box ECU* the vehicle will not be able to start. It may be possible to create a Denial of Service (DoS) attack that would cause the car not to start. For this attack, the attacker needs to be very close to the sensor and have a high-frequency tool to jam the RF signal [13]. In this attack, the attacker will not be able to insert the exploit code inside the vehicle. So the vulnerability is very low, apart from the Car Theft.
  In the case of RKE, the DoS attack can be performed not to allow the user to lock/unlock-/start the vehicle [59]. A group researcher performed an attack on the RKE system in 10 different model from 8 different manufacturers. They performed the relay attacks over the RF signal to gain access to the vehicle system. All the vehicle models are vulnerable to this attack since all manufacturer is using the same transponders in their key-fobs[18].

- Bluetooth

  In modern cars, Bluetooth is used to play music or sync the contacts with the entertainment system to make a hands-free call or message remotely through the entertainment system. The automotive systems are using Class 2 Bluetooth devices, which has a typical range of 10 meters. The range of the Bluetooth can be extended via the external antenna and amplifiers [23]. The attacker can gain control of the vehicle unknown to the driver or passenger by access through the Bluetooth and can compromise all the ECU's in the vehicle. Next to the OBD - ii port, Bluetooth is considered the biggest and the most viable attack surface in the vehicle [47].

- (Tire Pressure Monitoring Sensor) TPMS

  The vehicles are using TPMS system to measure the tire pressure in real-time. In Toyota Prius, Tire Pressure Warning ECU connects to Tire Pressure Warning Antenna and Receiver. Each Tire in the vehicle has a sensor, which transmits the transmitter ID and measured Value signal to the Tire pressure Antenna and Receiver Component, which sends the signal to ECU. If the transmitted ID already registered in the ECU, the ECU compares the measured value with the standard value. If it is low, it will send the signal to Combination Meter Display ECU to switch ON the tire pressure warning Light.
  The attacker can perform some action against the system, such as causing the vehicle to think that the Tire pressure is very low in a moving vehicle. Such type of attack has been performed out by the researchers [45][13] to activate the tire pressure warning light in the instrument cluster. But the attacker will not be able to intrude into the network or execute the code in the vehicle, so the attack surface is quite small from this entry.

- Wi-Fi

  In modern vehicles, Wi-Fi is provided by the manufacturers in the form of the dedicated network inside the entertainment system or perform as a repeater (Wi-Fi Hotspot) from the mobile network of the vehicle user. Since the automotive application does not have separate protocols, it uses the same protocol as standard Wi-Fi. So the attacker can exploit the vehicular network by performing attacks similar to regular Wi-Fi networks to get access to the vehicle.

- Ultrasonic Sensor/RADAR/V2X

  For autonomous driving, many manufacturers are using RADARS and Ultrasonic sensors as their input to drive the vehicle (*Such as Adaptive Cruise Control, Lane Departure Warning)*. Majority of automobiles are operated in millimeter wave (mmWave). They are two types of attacks can be performed in this system, such as Jamming and Spoofing. In the Jamming technique, if an attacker knows the operating frequency of the RADAR, he can generate the same frequency signal and disrupt the receiver. This will lead to a collision of the vehicle. In spoofing technique, the attacker will send the signal to the receiver that the distance between the target is shorter than the original. This will disrupt the vehicle behavior while cruising [61]. These techniques are also applicable for Ultrasonic sensors and V2X communications. In this attack, the attacker will not be able to exploit the in-vehicle network such as OBD - ii or Bluetooth. However, this attack will risk the passenger lives in the vehicle.

Long-Range Wireless Interface

- Radio/GPS/Cellular

  In modern vehicles using cellular, Radio & GPS for comfort. The cellular system in the vehicle can also retrieve the traffic and weather information from the Internet. If an attacker exploits the cellular interface, he can control the vehicle remotely from an infinite distance, such types of attacks are performed in real scenarios [13] [59]. Since the entertainment and navigation system are connected to the CAN network of the vehicle, the attacker can perform any attacks on the vehicle. In addition to above entry points, an attacker can spoof the GPS signal of the vehicle and navigate the driver in the wrong direction [42]. So the attack surface through this systems is quite large since it can get access to the critical systems.

2.4.2 Types of Attacks

When attacker gained access to the system via the above-mentioned entry points, based on the purpose and capability of an attacker, the attacker can perform various types of attack. The

attacker can perform suspending the specific ECU's message in the network, fabricating the messages and send it through the network to attain various behavior. In the *Chapter 4* various types of attacks performed in the simulation to know the behavior of the vehicle. The overview of each type of attacks is explained in the below section.

- Denial of Service (DoS) Attack

  The weak attacker can perform DoS attack. To perform this attack, the CAN network traffic should be analyzed and know the highest priority message in the network (e.g., a message with identifier 0x01). The attacker needs to compromise an ECU to send the required messages. After this, the attacker needs to send the highest priority message (0x01) at a high frequency. This will affect the network, by blocking all the message in the specific interval, making the vehicle to lose functionality and perform differently.

- Masquerade Attack

  In a CAN network, the ECU will not know the transmission node address; it will accept the message based on the identifier of the message only. In this attack, the attacker needs to compromise two ECU's. First, the attacker needs to stop the specific message transmitting from the ECU A and inject the same message from the ECU B with the same interval of time. By this attack, the network will not be able to find the difference; then after that, the payload of the message is modified to make the vehicle to behave differently.

- Frame Injection

  In this attack, the attacker will inject the know frames with from the compromised ECU to alter the behavior of the vehicle. To perform this attack, the attacker wants to know the identifier of the critical system (e.g., Brakes) and the respected ECU to perform the specific task in the vehicle (Skid Control ECU in Prius). Then, the attacker needs to send the brake signal from the compromised ECU to the Skid control ECU to perform the braking task. By sending this signal, the vehicle will experience the unauthorized braking while cursing the vehicle, leaving the driver confused.

- Black-hole attack

  In this attack, the attacker needs to compromise the vehicle critical system *(such as Transmission, Engine or Power management* and makes the message drops before it is transmitting leaving the vehicle to halt. For example, if an attacker compromises the Engine Control Management ECU in Toyota Prius, which is responsible for engine function in the vehicle. The attacker makes the ECU drop the transmitting messages, due to this functionality of the vehicle losses and makes the vehicle stop.

2.5 Conclusion

From this *Chapter*, the various types of vehicle architecture are explained in a detailed manner. In addition to this, *Section*2.2 described elaborately about the network protocols used in the vehicle E/E architectures. From this we are able to answer the first research question **What are the different in-vehicle network protocols and architecture used in the automotive system?**. In *section*2.4 of this chapter explained about the various entry points in the modern vehicles and what are all the types of attacks can be performed through this entry points is illustrated. From this we can able to answer the second two research questions asked in *Chapter 1* **What are the in-vehicle network entry points for vulnerabilities? What (and where) traffic, can we monitor and which are the ideal ECUs or bus monitoring points and detection capabilities?**. Apart from this, various types of attacks are explained in this chapter which can be performed in CAN bus. The result of attack simulations are explained are simulated in the *Chapter 4* to know the intended behavior of the vehicle during this attacks.

Chapter 3

In-vehicle Network Simulators

3.1 Different type of In-vehicle Network Simulators

The in-vehicle network simulators help the researchers to replicate the real vehicle behavior and simulate some attack scenarios to study the in-vehicle behavior. There are some network simulators used by researchers and vehicle manufacturers to develop and do research in the in-vehicle network. For our research, the network simulator should be user-friendly and low cost. To choose appropriate network simulator, several network simulators are considered as shown in below Table 3.1

Table 3.1: Comparison of different vehicle simulators

Types of Network Simulators	Ease of Network Modeling	Ease of Packet Capturing	Replication of Vehicle Network Behavior	License
MATLAB	+	+	+	Proprietary
CANoe	++	++	++	Proprietary
RTaW	+	-	+	Open-Source
NS2	-	-	-	Open-Source
OMNET++	++	++	++	Open-Source

The simulators mentioned above are studied individually to know the exact performance of the simulator. In addition to this, the various researches/projects performed by the above mentioned simulators are studied to know the capabilities of the simulators.

3.1.1 MATLAB

MATLAB Simulink is a leading simulation software which can be used for multiple purposes. For the in-vehicle network, a Vehicle Network Toolbox is available with the modules for simulating the network traffic. The toolbox has three different communication protocol: CAN, J1939 and XCP [33]. For this research, we focus only on the CAN communication. To perform the vehicle network behavior, we need to drag and drop the appropriate components from the library browser to the workspace and configure the components based on the user requirements. The CAN communication toolbox consist of: CAN configuration, CAN Message log, CAN pack, CAN receive, CAN Replay, CAN transmit and CAN unpack.

To transmit and receive CAN message in the Simulink, two models should be created in the MATLAB editor window. First, transmit model should be created to send the message. For this, CAN transmission, CAN pack and CAN transmit blocks are dragged into the editor. From the

library, drag the constant block into the editor and configure the constant value as, sample time and output data type. Connect the constant block and CAN Pack block input, also connect the CAN pack output to CAN transmit block. After this, configure the CAN configuration block by specifying device and bus speed. Then configure the CAN pack block such as Data as input, identifier type, identifier and length of the payload. At-last configure the CAN transmit block by specifying the transmit channel to channel 1.

After building the transmission model, receive model for CAN communication is created by dragging the CAN configuration, CAN receive and CAN unpack blocks into the editor. Place the CAN unpack model in the Function-call subsystem block. Connect the CAN unpack input(CAN MSg) to the in the block and connect the CAN unpack output (Data) to the out block. TO see the result of the simulation connect the scope block to the function subsystem output. Connect the CAN receive f() output to the Function call subsystem function(). Connect the CAN Msg to the input of the Function call subsystem(). Configure the CAN receive block to channel 2 and accept all identifiers. Configure CAN unpack subsystem to the name CAN msg, identifier and length mentioned in the transmission block. Save the model and run the simulation. By selecting the scope block, the received data is shown as per the input is given in the transmission model.

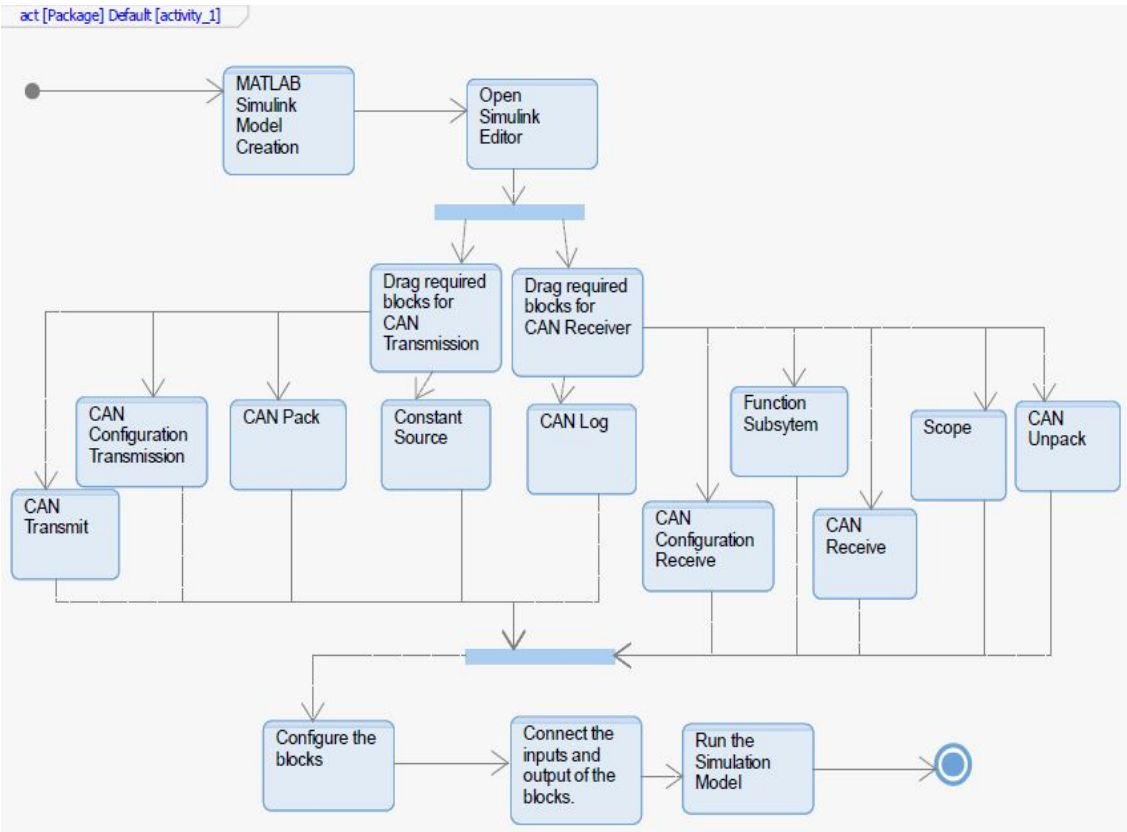


Figure 3.1: Activity diagram for MATLAB process.

The main disadvantage in the Simulink environment is to develop an entire vehicle architecture in the editor and run the simulation concurrently with the more number of transmission and receiving block. This is a tedious process to replicate the in-vehicle network behavior. In addition to this, excluding the scope and normal output screen, there are no blocks to record the receiving messages from different modules. Also, there is no clear idea how to inject new messages and do the dynamic behavior in the Simulink model. Due to this facts, Simulink is not used in this research to replicate the network behavior.

3.1.2 CANoe

A *CANoe* is a tool developed by the Vector Informatik GmbH. This tool can be used for the development, testing and analyzing the entire in-vehicle network architectures and configuring individual ECU's. The *CANoe* is used by several OEM's (Original Equipment Manufacturer) and other companies to develop an in-vehicle network from the develop and configure individual ECU to design an entire in-vehicle network. *CANoe* supports multiple network protocols: CAN, CAN FD, LIN FlexRay and another essential feature, the Automotive Ethernet. *CANoe* uses CAPL(Communication Access Programming Language) which is C like language developed and integrated inside the CANoe environment. In *CANoe* the network database(e.g., DBC file) can be directly imported into the environment. It is also capable of simulating, analyze and test the whole network or individual ECU's. The testing of individual ECU's in the environment is time-consuming and helps to identify the problems efficiently. In *CANoe*, the user can add additional hardware interface easily to know the exact behavior of the vehicle network in addition to the simulation [20].

The *CANoe* environment is simple to learn and understand due to many existing examples. If there is an existing CAN network database, we can import the parameters directly into the developing environment of the *CANoe*. From this creating vehicle network in the *CANoe*, the environment is effortless. Otherwise, CANdb++ is a data management program where the ECU information such as CAN identifiers, time and payload is created. Also, the configuration between the ECU's interface will be done automatically based on individual ECU's data. The result of the analysis also viewed in different formats such as trace window, graphics window, statistic window, scope window and state tracker. Each window has a different purpose while analyzing the traffic. In all this window the user can separate the ECUs information by using filters. In addition to this, there are a lot of additional features available in the *CANoe* software such as linking the simulation with multiple computers and changing the variables during the simulation. In addition to that *CANoe* can be interlinked with the simulink environment in the full version. By doing so, we can change multiple variables during the simulation. Since *CANoe* software is commercial, the environment pricing is quite high. In the demo version, the network cannot be extended beyond 6 ECU's. To perform different types of attacks in the simulation environment, the user should able to change the source code of the environment. However, the environment codes can't be changed based on user requirements therefore we can't perform variety of attacks in the simulation model. Apart from this, to perform changes in the simulation using CAPL is difficult compared to the other simulation environments. This is due to CAPL needs full information of the CAN database (i.e) to describe the purpose of every byte in the payload for each CAN ID's. During the reverse engineering process we have to collect every byte information of the CAN payload, therefore some flexibility to create the simulation network is needed. However, this flexibility is not available in the *CANoe* environment. Due to this reasons, *CANoe* is not chosen to use in this research.

3.1.3 RTaW

Real Time at Work(RTaW) [44] is a timing accurate simulator. In RTaW creating and configuring the CAN network is very simple using the editor window. Since RTaW is based on the timing, the reliability of the CAN network can be assessed and frame response time for each message can be know from the simulation.The results of the simulation can be shown using distribution plots and Gantt diagrams [44].

In the simulation, the frames are created based only on the timings and designated to the respective ECUs and buses. The connection between the frames and ECUS are determined in the bus connection. During the simulation, appropriate offset configuration can be chosen for the running. Before the simulation starts the runtime of the simulation should be manually entered. For this period the simulation will be performed and results given in the graphical representation. However, for this research the simulation with the payload data is essential,because the payload data can be used to determine the exact in-vehicle network behavior of the vehicle. In addition

to this, apart from the graphical representation of results, there are no other ways of analyzing results. The RTaW is, also, not a good candidate for our research purpose. However, we can use this environment to know the busload during different driving conditions.

3.1.4 NS2

NS2 is a discrete event simulator which is designed for research in communication networks [55]. Since the NS2 is an open source environment and written in C++. The possibility to replicate a vehicle network in NS2 simulator is carried out. Since there is no availability of source code for vehicle protocols such as CAN and LIN, to perform an in-vehicle simulation in the NS2 is entirely unrealistic.

3.1.5 OMNET++

OMNeT++ introduced in September 1997 is build on the Eclipse Integrated Development Environment (IDE) [32]. OMNET++ is a component-based C++ simulation library and framework. The environment is primarily for building network simulators and also the modular environment. OMNET++ uses two types of module: simple and Compound. The behavior of the model and algorithms are defined in the simple modules. The event occurs in the OMNET++ simulation with the help of simple modules. The collection of simple modules are the compound modules, which helps to interact with one another component in the environment. In OMNET++ there is an extra plug-in for INET to replicate the normal network behavior such as TCP, UDP, and Ethernet. Based on this, Hamburg University[37][38] has created plug-ins for vehicle network such as CAN, Flex Ray, and Automotive Ethernet. Based on the available plug-ins one can create a real vehicle network. Based on the input given before starting the simulation, the network will simulate the network traffic. In this environment, one can add any number of modules to the network. An easy understanding of the environment helps to create the network both in graphical and traditional coding way. By using the graphical method, one can quickly create the network with interconnection efficiently. In this environment, the source code of the plug-ins can be modified which gives vast space to develop our ideas into the plug-in based on our requirements. In INET module, the traffic in the network is recorded in the *.pcap* [53] format for analyzing the network behavior later, But for vehicle network plug-ins, we do not have any modules to record the traffic. So to address this issue, a PCAP module for CAN network is created. Due to the open source code, it is easy to implement the different types of attack in the simulation, which can be analyzed and real vehicle behavior can be predicted. Due to the simplicity of the environment, open source information and easy to change the module behaviors, this environment has chosen for this research to replicate the network behavior.

3.2 Overview of OMNET++

The main strengths of OMNeT++ are the Graphical User Interface (GUI) and the open-source environment. However, compare with another network simulations that are mentioned above, OMNeT++ is quite more comfortable to learn the way to simulate realistic scenarios of CAN Network.

The building blocks in OMNeT++ simulations are called simple modules, which form the lowest level of simulator hierarchy. Some simple modules can be integrated by the user to form a compound module. Subsequently, multiple compound and straightforward modules can be linked to form a model such as a protocol. To implement the simulation scenario, the model should execute the algorithms contained in the modules. The messages that exchanged between modules can represent frames or packets in communication networks or jobs or customers in queuing networks. Modules can send messages directly to their destinations or other modules along pre-defined paths. In the latter case, the exchange of messages between different modules takes place through the input and output interfaces of modules known as gates. The inter-module connections consist of

configurable parameters, e.g., propagation delay, bit error rate, and data rate. To define the structure of a model describing the message exchanging and communications between the corresponding modules, the user take advantage of a network description language called NED. OMNeT++ is introduced as a general purpose simulation engine, taking advantage of independently developed frameworks, also called packages, to support the simulation of communication networks. Aside from communication networks, OMNeT++ can also be used for modeling queuing networks, multiprocessors and distributed hardware systems and evaluate the performance of hardware and software architectures. Two popular frameworks that provide a comprehensive set of in-vehicle protocols are Core4INET and Fico4OmNeT frameworks[37][38].

FiCo4OMNeT (Fieldbus Communication for OMNeT++) is a set of Fieldbus simulation models. It was initially developed with separate CAN and FlexRay models but later merged to take account of the similarities of Fieldbus technologies. Similar to CoRE4INET, FiCo4OMNeT contains models of oscillators and clocks to allow for precise simulation of the synchronization of FlexRays static segment. Further, it contains application models for CAN and FlexRay applications with simple traffic patterns. The Fieldbus models in FiCo4OMNeT were checked initially against results of the CANoe simulation environment, an industry standard software for the simulation of Fieldbus based in-vehicle networks. Even though field buses are considered legacy technology there are new approaches. The modules present in the *FiC04OMNeT* are described in the next section, which gives an overview of the modules that able to perform as CAN Network in the Simulation.

3.2.1 Basic Modules Present in FiC04OMNeT

The modules required to build a CAN network in the OMNET++ will be available in the *FiC04OMNeT* library, such as CAN Network, CAN bus and CAN node which are called as simple modules. Using these modules, the simulation model will be build in the NED (Network Description Language). In the NED file, presented in *Section 4.2* the user describe the simple modules, connect and assemble them into the compound modules to perform the simulation. Inside each simple module, there will be submodules which help to run the module in the simulation. For example, The CAN bus module in the NED file consists of can bus logic, bus port and pcap recorder as shown in Figure 3.2.

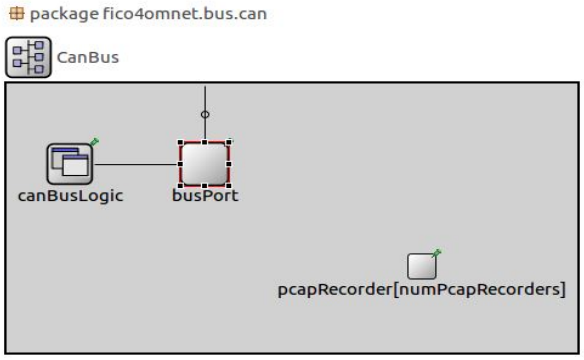


Figure 3.2: NED file of CAN bus Module.

CAN bus logic in the Figure 3.2 is responsible for arbitration in the simulation. The bus logic will allow the node to send a message through the CAN bus which has the highest priority. Bus port in the CAN bus will simulate the physical connection between the nodes and bus. It is responsible for distributing the incoming frames to all the nodes in the network. Pcap re-

corder helps to monitor and capture all the incoming frames coming through the CAN bus during the simulation. From this capture, we can analyze and compare with the real car network behavior

The CAN node module in the NED file consists of source app, buffer out, can clock, buffer in, CAN node port, buffer in, sink app and pcap recorder as shown in Figure 3.3.

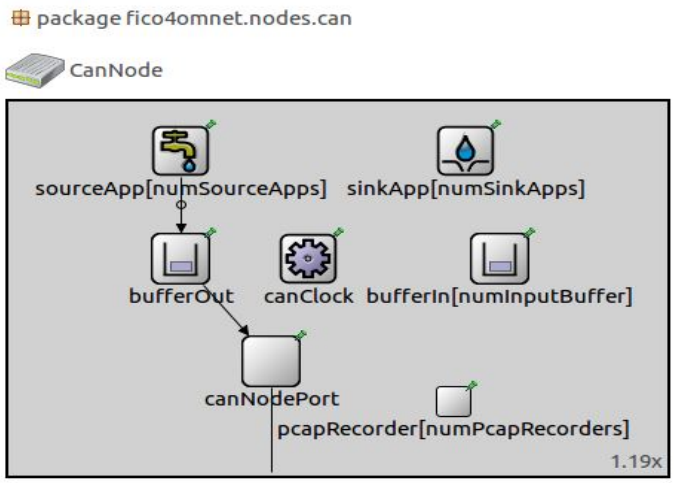


Figure 3.3: NED File of CAN Node Module.

Source App is responsible for generating all data and remote frames from the node based on the initialization file. It will schedule the messages which are sent periodically through the Buffer-out module. Buffer out module helps to contain the generated frames from the source app. When it receives the frame from the source app, it will register the message with the CAN id at the bus for arbitration. After the bus grants the permission it will send the frame to the CAN node port. CAN node port will establish the connection between the buffers and bus. CAN node port contains sub-modules like CAN port input, CAN port output and node port. CAN port output will process the outgoing frames in the network and also generates the error frame if the frame has some error. All receiving frames are processed in CAN port input. It also forwards the frame to the input buffer. Node port will merely establish the connection between the port modules and the bus module.

Buffer-in module will receive the frames from the CAN port input and sends the received message to the sink app, which will process the received message. When a Buffer-in module receives the frame, the module will be informed. When the sink app is idle, it will request the buffer in to send the message. CAN-clock module will simulate the clock drift in the CAN node.
However, the modules inside the FiCo4OMNeT library has some shortcomings such as not able to simulate the network with the CAN payload, monitor and record the CAN traffic in the simulation environment, change the CAN payload based on the vehicle performance and simulate an attack in the simulation environment. So to enhance the FiC04OMNeT library extra modules were built, and existing modules are modified based on the thesis requirements. For example, pcap recorder is a new module added to the FiCo4OMNeT library to capture the network traffic in the CAN node and CAN bus as shown in Figure 3.2 and 3.3. In addition to this existing module such as source app, node port is modified based on the requirements to perform dynamic behavior of the vehicle and attack simulations.

After creating the NED file in the OMNET++, to perform simulation the data of the real vehicle should acquire by the process of reverse engineering. Based on the data collection only,

the simulation environment will be improved in a more realistic way. By having simulation results matching to the real car, performing various attacks in the simulation environment to know the real vehicle behavior

3.3 Conclusion

In this *chapter* the various in-vehicle network simulators are discussed to replicate the in-vehicle network traffic. This chapter mainly described about the methods used by the simulators to simulate the traffic and what are the shortcomings of each simulators. For the most of the simulators, addition of new modules to enhance the functionality of the simulator to make the traffic more realistic is not possible. However, one such simulator know as *OMNET++* has the capability of making the new modules based on the user requirement. By this the network traffic will be more realistic compared to the other simulators. In addition to this, the detailed introduction about the selected simulator for this experiment and the existing modules in the simulator have been described in this chapter. By this the first subquestion, **What is the suitable environment for creating the in-vehicle network model and simulate the network traffic** of the 4th research question **How to replicate the in-vehicle network behavior in the simulation environment** has been answered.

Chapter 4

Experiments and Results

4.1 Reverse Engineering

The primary requirement to simulate the real in-vehicle communication of the vehicle is to get the in-vehicle network information such as protocols used in the networks, the number of ECU's, CAN ID's of the ECU's and interconnection between the ECU's in the network. There is no extended documentation about the requirements mentioned above since the information are proprietary to the vehicle manufactures. To replicate the real in-vehicle communication in the simulation environment, we must do the reverse engineering in a specific vehicle to extract our required information like CAN ID's for the ECUs and an interaction between the ECU's in the network. The following sections 4.1.1, 4.1.2, and 4.1.3 describes the reverse engineering process to obtain the required data for the simulation environment.

4.1.1 Tools and Software used for Reverse Engineering

The first step in the reverse engineering is to choose the vehicle that used for experiments. Since each vehicle and manufacturers have a different kind of network architecture, selecting a particular vehicle will help to obtain more data and analyze it extensively. For the experiments the Toyota Prius - XW30 [Experiment Vehicle of A-Team] Model is chosen. The reason behind choosing Prius based on the face that vehicle is using more electric /electronics component compared to the same price range model vehicle. By using more E/E components, the data flowing through the network is significant and controlling the vehicle using the computers is also possible easily. The E/E configuration of the vehicle is explained in detail in *Chapter 2*.

After choosing the vehicle, the type of On-Board Diagnostic (OBD - II) port is analyzed, and the required tool has been chosen. They are various types of hardware tools available on the market to monitor the OBD port traffic such as CANtact, ELM327, and Mongoose. Each tool has its own property such as ELM 327 is used primarily for diagnosing the vehicle . CANtact is using to monitor the CAN traffic in the network and manufactures proprietary tools which should be used with their software can be used for all the purposes like monitoring, diagnosing, revising and flashing the ECU's in the vehicle. For this thesis, monitoring the CAN traffic is essential to replicate the network in the simulation environment. So for this reverse engineering process, CANtact has been chosen. CANtact has two ports, which are input(VGA) and output(USB) ports. The input port of the CANtact is connected to the vehicle via the VGA to OBD Female port, and the output of the CANtact is connected to the computer via USB type B to USB Type A cable. The can-utils software was used in Ubuntu operating system to monitor the CAN traffic on the vehicle network from the CANtact device [57].Recording the CAN traffic using the CANtact and

CANutils are explained in the section 4.1.3 The below Figure 4.1 shows the connection between the hardware tools as mentioned above.

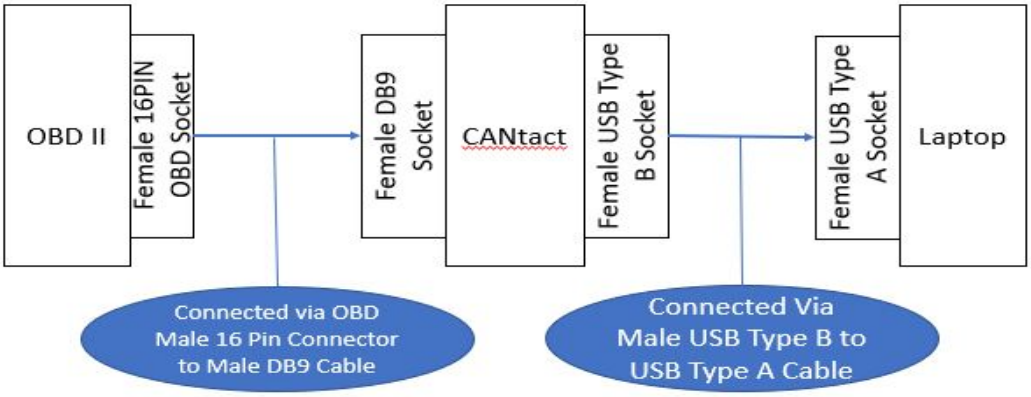


Figure 4.1: Connection between the Hardware tools used in the reverse engineering process.

In addition to that ELM327 is used to monitored the live CAN traffic. For this experiment with ELM327, various softwares are used depends upon the platform such as OBD Auto Doctor for Windows, ELM327WiFiTerminal for ios, and ELM327 OBD Terminal for Android. The hardware connection between the vehicle is very simple. ELM327 plugs into the OBD - II port and connect the ELM327 via WiFi using the above softwares to send and receive messages. However, compared to CANtact, the information receiving from ELM327 is very limited. This is due to property of the device, where ELM327 is used primarily for diagnostic vehicle. Unlike CANtact, the user will not able to record the CAN live traffic for continuous time period. In contrary, the user will request a specific data from the vehicle using OBD - II PID (Parameter IDs) commands. Experiments done using the ELM327 with the help of ELM327WiFiTerminal app are explained later in the section 4.1.3.

4.1.2 In-vehicle Network Architecture

Since the architecture is the vital part of the reverse engineering process, the second step is to know schematics of the in-vehicle network architecture. The architecture is obtained from the Toyota website [11]. By obtaining the in-vehicle architecture, the necessary information such as the number of ECUs, sensors connected to the respected ECU's, functionalities and interconnection between the ECU's are studied. Table 4.1 describes about every ECUs and its function of the Toyota Prius. From the study, the scattered information of the in-vehicle architecture is reproduced in the IBM Rhapsody software as shown in the Figure 4.2 to understand the architecture model entirely during experimentation and for future studies.

Table 4.1: List of ECU's and Sensors in Toyota Prius Architecture

List of ECU's in Toyota Prius and its functions	
ECU and Sensor Name	Basic function of the ECU
Main Body ECU	The ECU is one of the important systems in the vehicle. It helps to manage the basic vehicle functions such as lock and unlock the doors, headlight functions.
Certification ECU	The ECU is responsible for wireless door lock control system, and it performs the ID code identification process when the door lock system engages.

Skid Control ECU	Skid control ECU will monitor and control the regenerative brake system, Anti-lock Brake system (ABS), Electronic Brake-force Distribution, Traction Control, Brake Assist, Vehicle stability control and Hill-start Assist.
Yaw Rate Sensor	The sensor will send the vehicle's angular velocity in the vertical axis. This information is mainly used in the Skid control ECU and Cruise control.
Power Management Control ECU	This ECU helps to drive the vehicle in electric mode in low speed and regenerate the electric batteries while cursing. The ECU helps to generate the force to charge the engines based on the driving conditions.
Engine Control Management (ECM)	It is responsible for controlling the engine process such as ignition system, controlling the performance of the engine based on the data of several sensors such as ignition, mass air flow, air-fuel ratio and crankshaft position etc.
Transmission Control ECU	The ECU helps to control the transmission of the vehicle such as parking, driving, reverse. While driving it helps to work the engine synchronously to manage the torque and control the speed of the vehicle based on the user input.
Center Airbag Sensor Assembly	The deceleration and safing sensor are built into the airbag sensor. So in the case of collision, the distortion is created send through electric signal based on deceleration rate, this helps to deploy the airbag in perfect timing. During the collision, the seat belt pretensioner will operate simultaneously to the airbag sensor.
Steering Angle Sensor	The sensor helps to send the user input of the steering rotation to the Power steering ECU to ease the steering effort while driving.
Power Steering ECU	The ECU helps to generate the torque from the motor based on the steering torque signal and vehicle speed from skid control ECU to assist the steering effort while driving the vehicle. This also helps to control the steering effort based on the speed of the vehicle
DLC3	The DLC3 is On-Board Diagnostic port helps to debug the faulty codes and monitor the CAN data traffic of the vehicle. DLC3 is mandatory in all passenger vehicle in the world.
Combination Meter	The combination Meter will get signal from the important components of the vehicle during driving and start of the vehicle to notify the driver in the dashboard of the vehicle such as engine temperature, SRS airbag, seat belt notification etc.
Air Condition Amplifier	The ECU is responsible for controlling the air condition supply in the vehicle remotely and manually
Seat Belt Control ECU	The ECU helps to work the seat belt pretensioner in a precise manner while driving and collision.
Lane Recognition Sensor	The sensor will help to detect the lane markers on the road and sends the data back to the lane keeping assist system. This helps the driver to notify if the vehicle is move out of the lane.
Millimeter Wave Radar Sensor	The sensor helps to calculate the distance in-front of the vehicle, when the vehicle is driving in the dynamic cruise control mode. From this data and yaw rate sensor data, the vehicle will automatically control the speed based on the front vehicle.
Driving Support ECU	The ECU helps to activate and manage the system in the vehicle such as cruise control, Lane recognizance and parking assist system. This functions are managed by the information given by the particular sensors in the vehicle

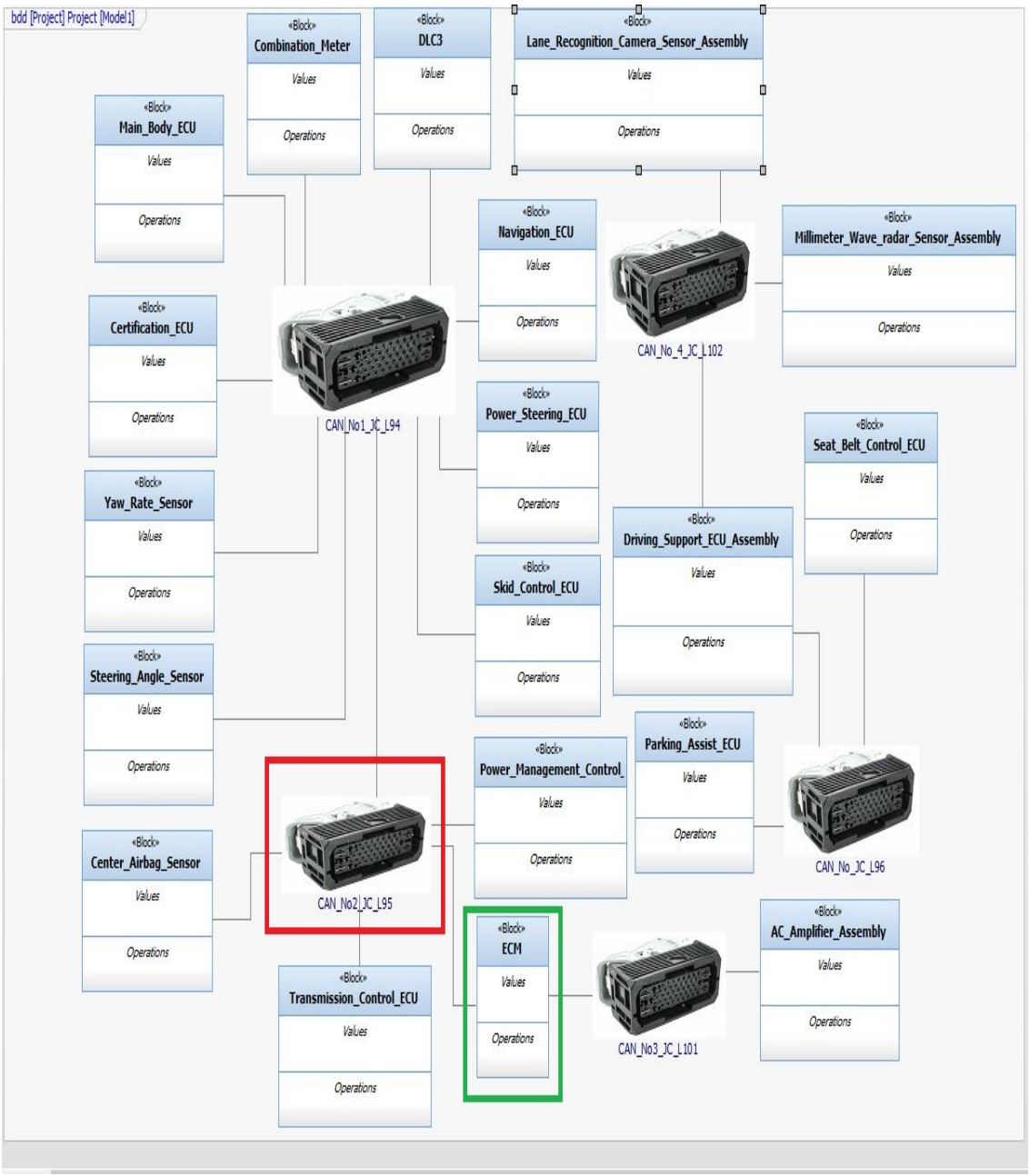


Figure 4.2: Block Diagram of Toyota Prius Architecture.

Toyota Prius architecture consists of Controller Area Network (CAN) bus and Local Interconnected (LIN) bus [11] [10]. The above 4.2 only shows the CAN bus connection. The architecture has five separate CAN bus lines for different ECU's. The Junction connectors do the interconnection between the ECU's and sensors as shown in Figure 4.2 (Red box). The purpose of Junction connector (JC) is the network Gateway for the ECU's. The interconnection between the two different CAN lines is done by the ECU's as shown in Figure 4.2 (Green box). The separation of the CAN bus lines is done by the importance of the ECU's function. For example, important ECU's such as Engine Control Management(ECM), Transmission Control ECU, Power Management Control are isolated in the separate CAN bus lines. Based on the architecture information and model, decoding and analyzing CAN traffic will be more precise during the reverse engineering process.

4.1.3 Data Collection Experiments and Analysis of In-vehicle Network (CAN) Data

Experiments conducted with the help of CANtact to collect the CAN live data

The final step in the reverse engineering is to make a derivative plan to record the CAN data traffic in the in-vehicle network for the reverse engineering process using CANtact. During the experiment, the vehicle data is recorded in three different states: They are 1). When the vehicle is switched off, 2). When the vehicle is in the parking mode (i.e.) The vehicle is powered on, and the vehicle is not driven around or moved and 3). When the vehicle is driving. During the first phase of an experiment to record the CAN traffic, the vehicle network data is recorded during the vehicle is parked and in driving modes. To compare the recording during the analysis in a more efficient way, we monitor and record the vehicle data in both modes to the same period (the data was recorded for 5 minutes in both modes). The live data collection using the CANutils software is shown in the below Figure 4.3.

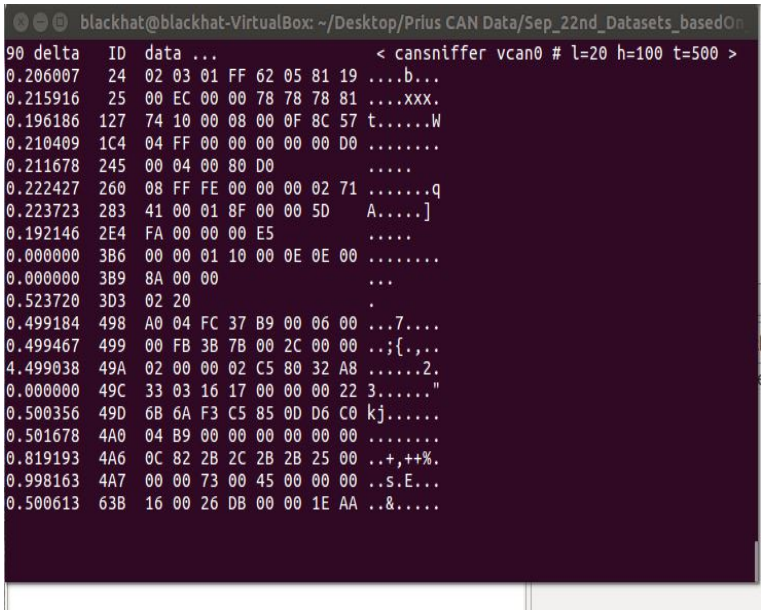


Figure 4.3: Data collection of live CAN traffic using CANtact and CANutils.

By analyzing the first set of recorded CAN data traffic, only few CAN ID's are known such as steering input, engine, and speed of the vehicle. Since the data recorded while parking and driving mode, the understanding about the purpose of CAN ID's are minimal. To find the purpose

of other CAN ID's various stand alone scenarios are created to find the CAN ID's purpose. For example, the scenario such as press the brake pedal, rotate the steering and press the accelerator are monitored and recorded during parking mode. Each scenario should be monitored and recorded separately for more straightforward analysis.
From the experiments, the number of CAN ID's detected while monitoring is about 89 CAN ID's. By recording and analyzing the CAN network traffic based on the scenarios, the description of about 53 CAN Id's have been identified and origination point of 23 CAN ID's have been known. The description of remaining 13 CAN ID's are not able find due to the time and hardware limitations. During the analysis, the interconnections between the ECU's are also studied, and the CAN message transmitter and receiver ECU's are also distinguished. All the data organized from the analysis, the database file(.dbc) for CAN ID's is created from the acquired information. By all this collective information, we reduplicated the CAN network traffic of the Toyota Prius in the simulation environment to simulate the network behavior of the ECU's and inject some attacks in the network to see the behavioral changes of the network.

Experiment conducted with the help of ELM327 to analyze the CAN messages

However, conducting experiments using ELM 327 is different compared to the above method. ELM327 can't able to record continuous traffic. But in contrary, it helps to receive a specific data from the vehicle, when user sends the PID commands. SAE standard J1979 defines the OBD - II PIDs. As per standard, the OBD - II has 10 types of diagnostic service [15]. In each service they are specific PIDs which will give a specific information about the vehicle. For example, if the user needs to find the engine RPM for the current time *01 0C* command is used. *01* is used to choose the diagnostic service to show the current data and *0C* is used to show the engine RPM. The received data is show in the below Figure 4.4.

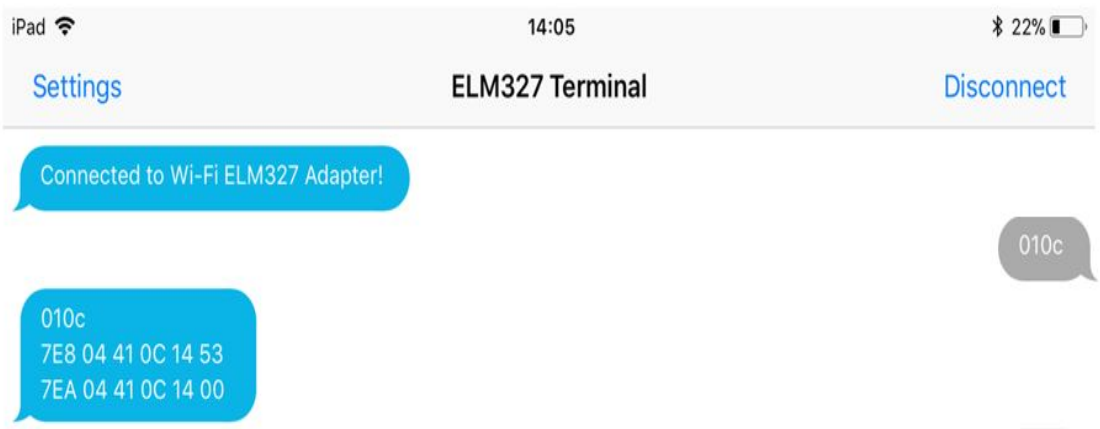


Figure 4.4: ELM327WiFiTermianl Snapshot while receiving message from the vehicle.

The received payload from the vehicle should divide into bytes to understand the information. *7E8 and 7EA (1st byte)* represents the ECU name, whereas *04 (2nd byte)* represent the data length of the message, *41 and 0C (3rd and 4th byte)* refers to the send command, where 41 refers to the mode 1 and 0C is the PID command sent. At-last, *10 C5 and 10 80 (5th and 6th byte)* are the vehicle RPM, which should be decoded from hexadecimal to decimal. From this experiments, for some known PID's such as related to engine RPM, speed and ECU name can be cross checked with the above analyses.

4.2 Simulation in OMNETPP

The in-vehicle network simulation can be done in OMNET++ with the help of Fieldbus Communication for OMNeT++ (FiCo4OMNeT). FiCo4OMNeT is an extension of the OMNET++ INET framework simulation library. Therefore, FiCo4OMNeT models can implement and simulate the automotive legacy Fieldbus communication system such as CAN. It contains all features required for automotive simulation network such as using two channels (A and B) of CAN Line for communication, error and remote frames. In addition to this, it also contains clocks, oscillators and other modules for precise communication.

To simulate a vehicle CAN network in OMNET++, we need an initialization file(.ini), Network Description file(.NED). The initialization file will contain the details of the real vehicle CAN n CAN ID, initial data frame offset, the periodicity of the data frame and payload of the CAN ID. Whereas in the Network Description file, the real CAN Network of the vehicle will be built. In NED file the user will declare the modules based on the requirements to build a CAN network.The design of NED CAN network of the Toyota Prius is shown in Figure 4.5.

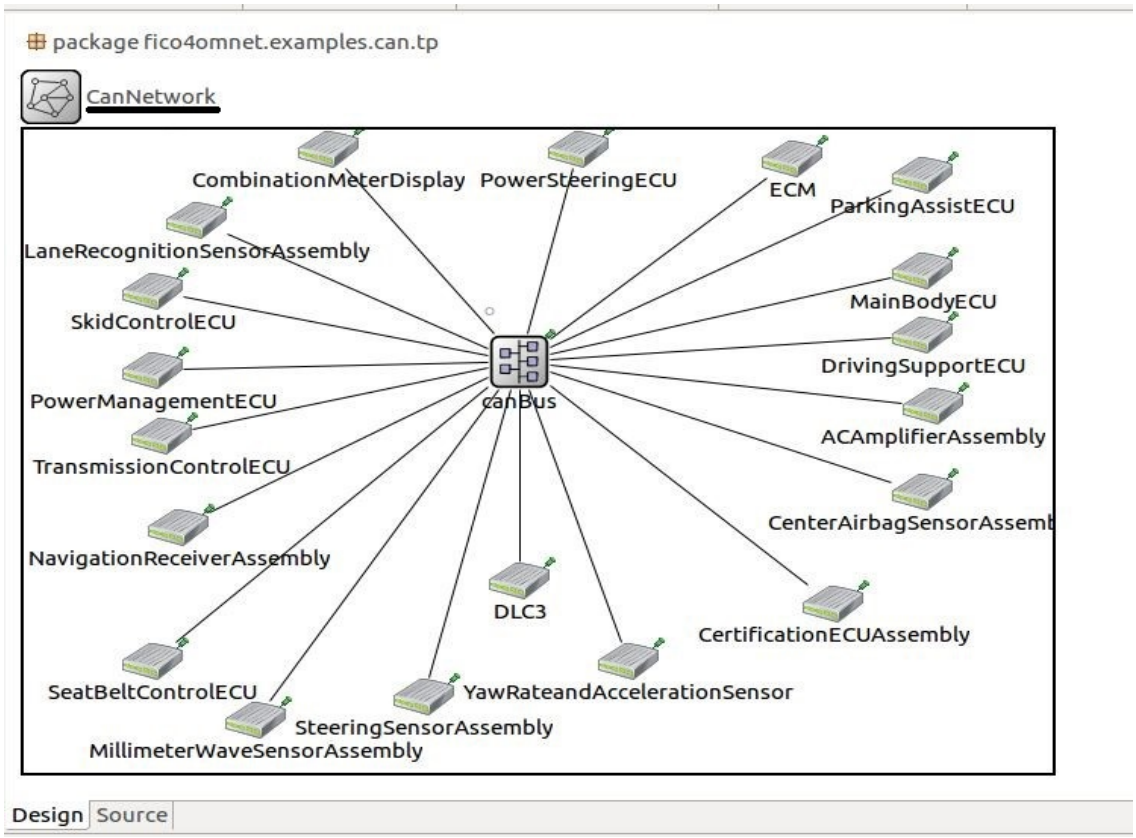


Figure 4.5: The content of the NED file structure, which describe the vehicle architecture.

The simple modules (Such as: CAN Bus, CAN, Node) in the NED File are explained in the *Section 3.2.1 in Chapter 3* of this report.

4.2.1 Simulating the real vehicle network traffic in parking mode

The initialization file in the simulation environment should be created with the respected to the DBC file to simulate the real vehicle network traffic in the simulation. DBC file will consist of CAN ID's, respective origin nodes for all CAN ID's and length of the CAN data payload. In addition to this, the initial data frame offset for each CAN ID's and periodicity occurrence of each CAN ID's should also be included in the *ini* file. For example, the respective CAN ID's and other information of the steering sensor assembly and Power Steering ECU are written in the *ini* file as shown in Figure 4.6. After entering all node details in the *ini* file, the *ini* file can be run based on the NED file network underlined in both figures 4.6 and 4.5.

```
1 [General]
2 network = CanNetwork
3
4 **.bandwidth = 0.5Mbps
5 **.bitStuffingPercentage = 0
6 **.version = "2.0B"
7 **.numPcapRecorders = 1
8
9 **.SteeringSensorAssembly.pcapRecorder[0].pcapFile = "Node_1_Pcap_Log.pcap"
10 CanNetwork.SteeringSensorAssembly.sourceApp[0].idDataFrames = "37, 608, 610"
11 CanNetwork.SteeringSensorAssembly.sourceApp[0].periodicityDataFrames = "0.100, 0.200, 0.400"
12 CanNetwork.SteeringSensorAssembly.sourceApp[0].dataLengthDataFrames = "8, 8, 5"
13 CanNetwork.SteeringSensorAssembly.sourceApp[0].payloadDataFrames = "000088007878781d, 0e00000000000078, 000000069000000"
14 CanNetwork.SteeringSensorAssembly.sourceApp[0].initialDataFrameOffset = "0.2226, 0.2493, 0.2540"
15
16 **.PowerSteeringECU.pcapRecorder[0].pcapFile = "Node_2_Pcap_Log.pcap"
17 CanNetwork.PowerSteeringECU.sourceApp[0].idDataFrames = "32, 36, 643"
18 CanNetwork.PowerSteeringECU.sourceApp[0].periodicityDataFrames = "0.0100, 0.0200, 0.030"
19 CanNetwork.PowerSteeringECU.sourceApp[0].dataLengthDataFrames = "8, 8, 7"
20 CanNetwork.PowerSteeringECU.sourceApp[0].payloadDataFrames = "0000070100002f, 5a0309fe6a078081, 0100800000000e"
21 CanNetwork.PowerSteeringECU.sourceApp[0].initialDataFrameOffset = "0.0415, 0.0783, 0.1536"
22
```

Figure 4.6: Initialization(.ini) file of CAN Network.

After running the initialization(*ini*) file, the simulation screen will appear in the window as shown in Figure 4.7. When the simulation of the network starts to run, the nodes will start to transmit and receive the CAN message in the network. The traffic flow in the network will be captured in the PCAP (Packet Capture) format. Pcap uses to capture network traffic using application programming interface(API) such as Wireshark. This can be later analyzed to see whether the traffic in the simulation and real vehicle are matching.

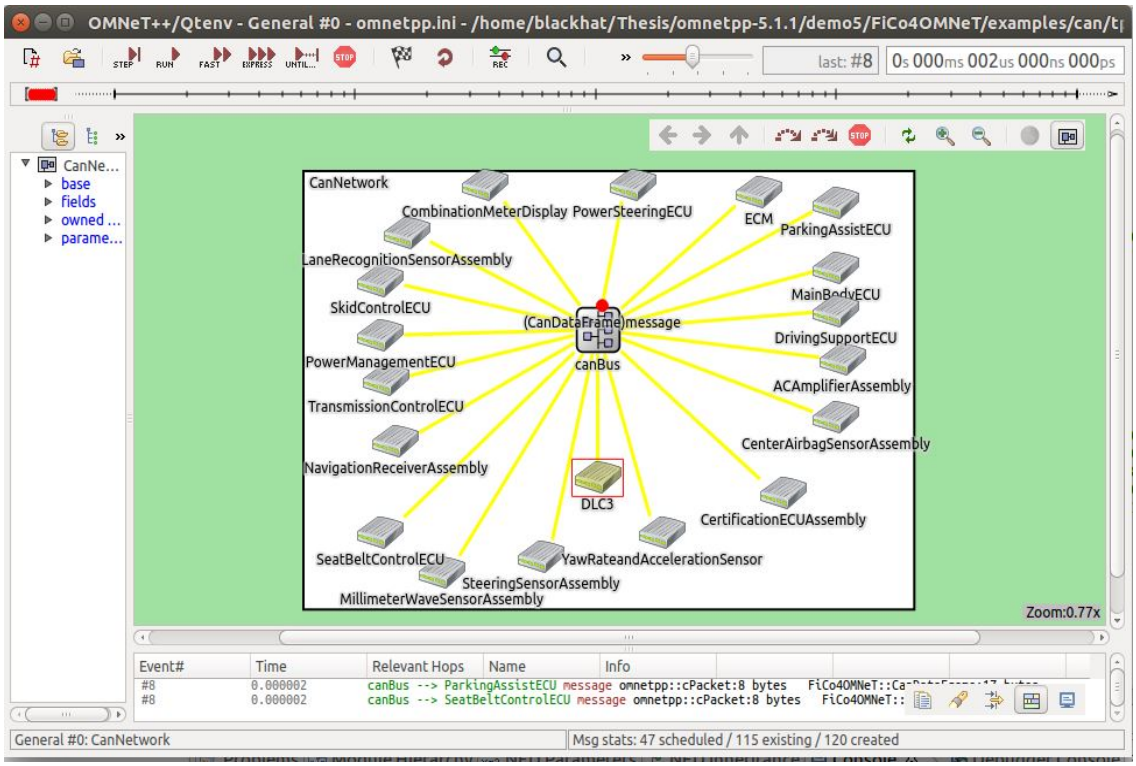


Figure 4.7: Simulation Window of OMNET++.

From the *pcap* captured during the parking mode in the real vehicle and captured traffic in the simulation, both *pcap* files are compared simultaneously in a python script. The results are shown in Figure 4.8. From this result, the simulation environment for the in-vehicle network is 95% accurate compared to the real vehicle. Apart from parking mode, by adding some dynamic formula in the source code of the FiC04OMNeT library, some dynamic behavior scenarios are carried out in the vehicle to compare the network behavior with the car, By doing this, the results of attack in the simulation environment will be more realistic to the real vehicle.

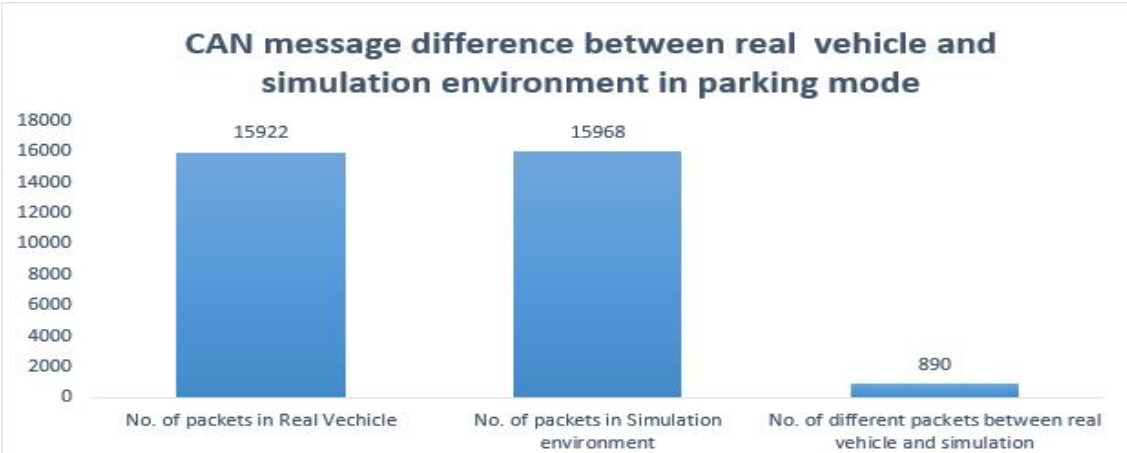


Figure 4.8: Graph represents the difference of No. of Packets in various simulation scenarios.

4.2.2 Simulating the dynamic behavior of the real vehicle

The source code of the source app module in the CAN node is modified to achieve dynamic behavior in the simulation environment. When the simulation starts, payload from the initialization file will be reached source app for transmitting. During this time the payload will be modified and transmitted to the output buffer to achieve the dynamic behavior of the vehicle. Due to time constraints for the project, only some scenarios are chosen for dynamic behavior such as pressing the brakes, rotating the steering wheel, opening and closing the doors, changing the gears in the transmission, switch ON and off the Air conditions and Lights in the vehicle. In addition to this, some CAN ID's have a checksum at the end of the payload. To achieve this, a formula has been inserted in the source app for updating the checksum in each appearance of the messages.

As shown in figure 4.9, the dynamic behavior of the steering wheel is attained from the simulation. The behavior is obtained by adding a function in the CAN traffic source application source code. In the figure 4.9, the y-axis represents the steering angle rotation and x-axis represents the time in seconds. The midpoint of the steering wheel rotation is 0 as shown in the figure. For turning the wheel left the starting point will be 01 and end point of left turn will be 337. When the steering wheel turn right the starting point will be 4095 and end point of right turn will be 3752. Due to this vast difference of starting points, the vast peak is appearing in the figure 4.9.

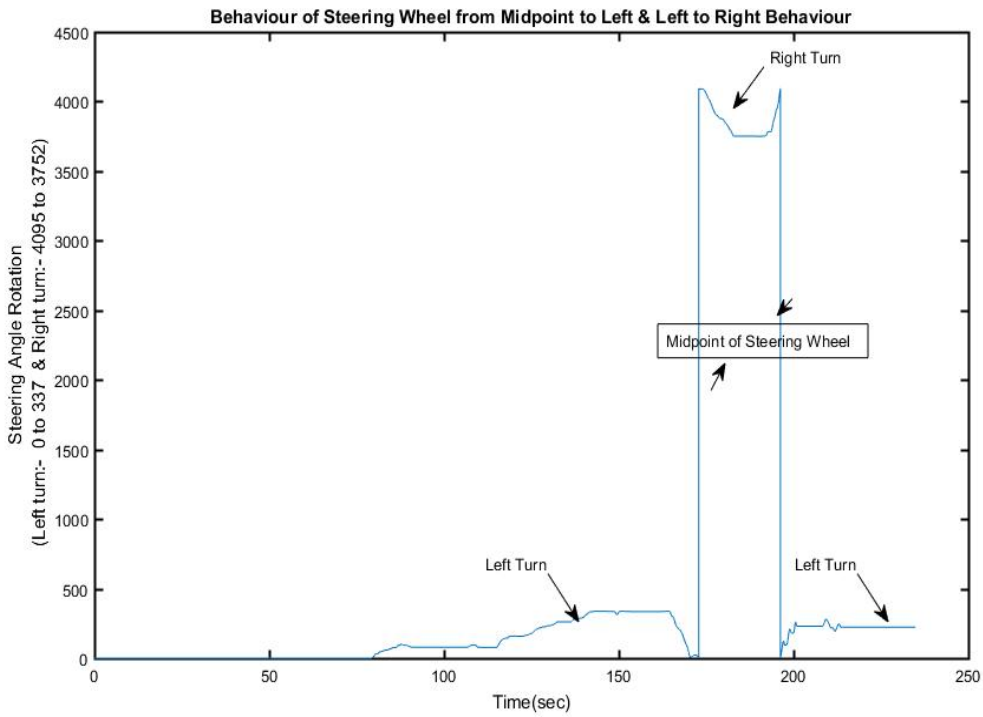


Figure 4.9: Graph represents the behavior of steering wheel.

The Figure 4.10 shows the engagement of the brake pedal and pressure applied during the engagement is showed. The blue line in the graph represents engagement and disengagement of the brake with respect to the time, while the red line represents the amount of brake pressure applied in the pedal during the engagement. From the brake pressure, the severity of the brake can be known. By making this brake application in a simulation environment, the various attacks can be checked for this behavior to understand the vehicle behavior during the attack.

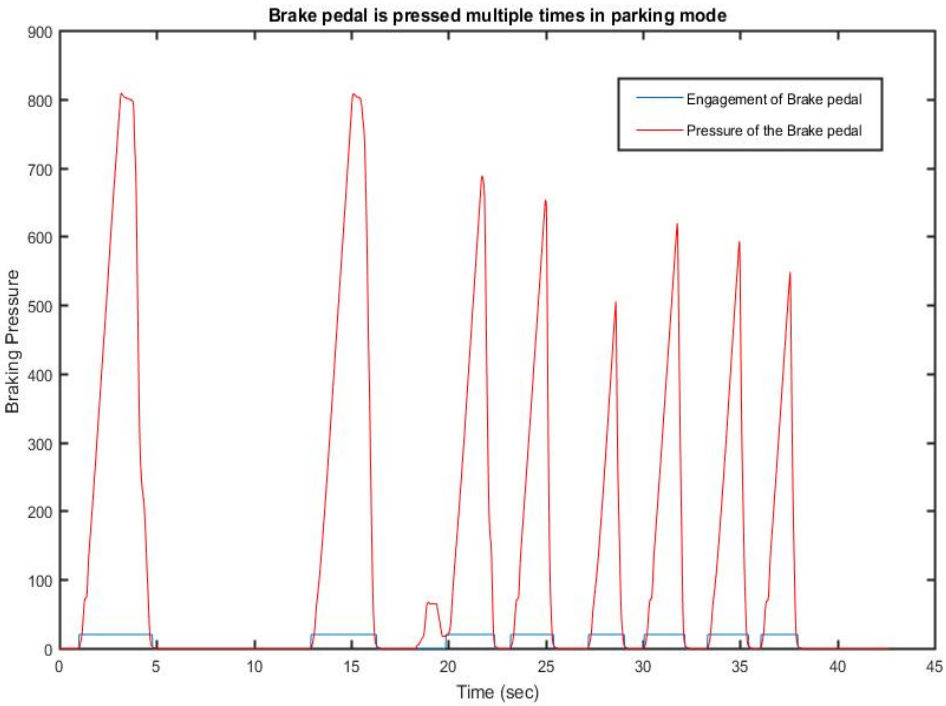


Figure 4.10: Graph represents the engagement of brake in parking mode.

To understand the graph, the Toyota Prius transmission system should be known first. Since the vehicle is automatic transmission, the movement of gear lever will be straightforward. When the vehicle is parked, the gear lever will be in drive mode, while reversing it will change to reverse mode and while driving it will change to drive mode and while down hilling the vehicle, the gear lever will be in B mode. The B mode helps to drive the vehicle in high torque while down hilling the vehicle. The overview of the gear stick is shown in figure 4.11.

Figure 4.11: Outline Picture of Toyota Prius Gear Stick.

In Figure 4.12, the gears of the vehicle is changed by the simulation dynamic behavior. There are two CAN ID's (CAN ID 0x247 & CAN ID 0x3bc) involved in changing the gear lever of the vehicle. The blue line in the figure 4.12 CAN ID 0x247, shows when the vehicle is starting with respect to time and whether the gear is selected to park or reverse mode or in drive mode.

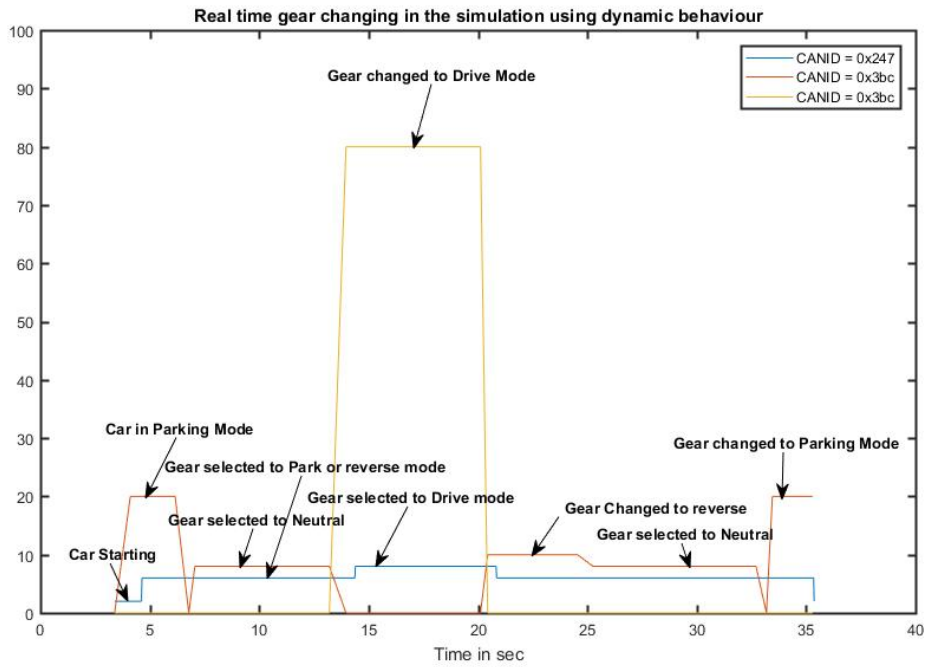


Figure 4.12: Graph represent the gear changing behavior of the vehicle.

4.2.3 Simulating some attacks in the vehicle network to analyze the behavior of the network

In a real vehicle, performing an attack on the in-vehicle network will have unpredicted behavior in both parking and driving mode. So performing an attack in simulation environment will give an idea about in-vehicle network behavior during the attack. It also helps to study the behavior and affected system during the attack. In a simulation environment, different types of attack (such as Denial of Service attack, Frame injection) can be performed, and precaution measures for the attack can be tested before implementing in the real vehicle. Such types of attacks are performed in this experimental study, and the results are discussed in the next sections.
Based on the vehicle architecture study they are two potential entry points such as DLC3 and Navigation receiver assembly for the attacks. Since the experiment vehicle does not contain Navigation receiver assembly, the corresponding data's cannot be recorded. Due to this, the only entry point in the simulation environment is considered as DLC3 unit.

Denial of Service attack

Denial of service attack, is also referred as (DoS attack), is a method of stopping an ECU from normal operations. The outcome of this will cause loss of functionality in the vehicle. For example, if DoS attack is originating from DLC3, then all ECU operation can be compromised based on the intensity of the attack. This attack can be made with respect to the time. When an attacker injects continuous packets for certain time period with high priority CAN ID's then the ECU's which are intended to communicate in the particular time frame will lose the priority, and the vehicle will lose some functionalities which can't be predicted correctly. Performing the DoS attack in the real vehicle will be dangerous, and the effect of the attack will also not be well predicted.

In simulation experiments, we will study the effects of DoS attack and how the system will be affected.

In this experiment we will inject the high priority CAN ID messages from the DLC3 module in a periodic time to know the effect of the network. As shown in Figure 4.13, the comparision between the normal scenario and two different DoS attacks are compared. First, total number of packets are compared in three different scenarios. As presented in Figure 4.13 during DoS Attack:2 the amount of packets are increased. Next two sets of data are compared with the specific CAN ID's 0x230 which is related to the braking of the vehicle When the DoS attack is performed with manipulating the payload of the CANID then the brake response of the vehicle will be affected. As shown in Figure 4.13 during the attack the number of packets are increased extremely and it will affect the braking of the vehicle.

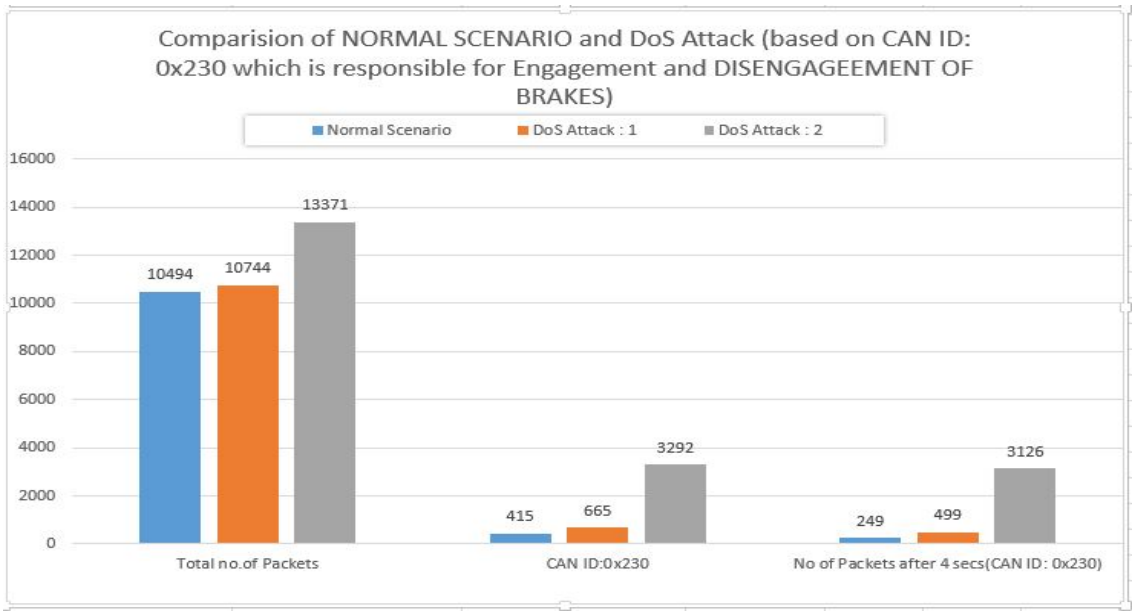


Figure 4.13: Comparison graph of CAN packets in Normal and DoS Attack Scenario.

The activity diagram in the below Figure 4.14 described progressively how the attacker perform the Denial of Service attack. The snippet of code that is used for Denial of Service attack is shown in below figure 4.15.

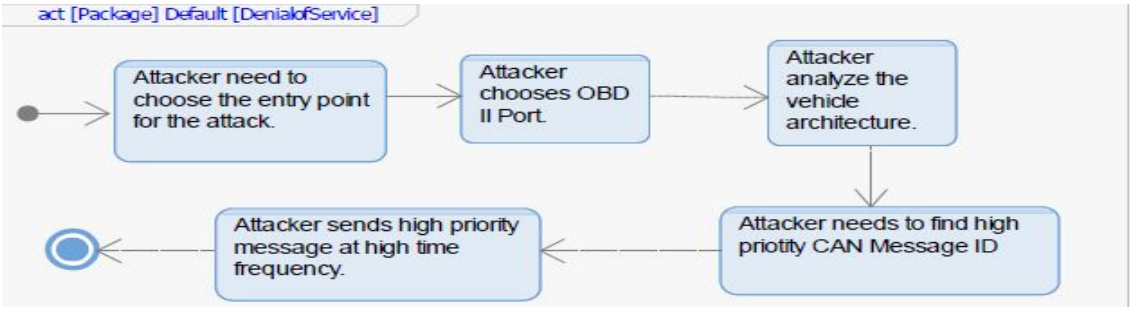


Figure 4.14: Activity diagram for Denial of Service attack.

```
#Start of DoS Attack 1 (Example of brake engaged after 1 sec with low time frequency)
CanNetwork.DLC3.sourceApp[0].idDataFrames = "560"
CanNetwork.DLC3.sourceApp[0].periodicityDataFrames = "0.000400"
CanNetwork.DLC3.sourceApp[0].dataLengthDataFrames = "7"
CanNetwork.DLC3.sourceApp[0].payloadDataFrames = "0012020400005100"
CanNetwork.DLC3.sourceApp[0].initialDataFrameOffset = "1.000000"
#End of DoS Attack 1

#Start of DoS Attack 2 (Example of brake engaged after 1 sec with high time frequency)
CanNetwork.DLC3.sourceApp[0].idDataFrames = "560"
CanNetwork.DLC3.sourceApp[0].periodicityDataFrames = "0.000006"
CanNetwork.DLC3.sourceApp[0].dataLengthDataFrames = "7"
CanNetwork.DLC3.sourceApp[0].payloadDataFrames = "0012020400005100"
CanNetwork.DLC3.sourceApp[0].initialDataFrameOffset = "1.000000"
#End of DoS Attack 2
```

Figure 4.15: Initialization file(.ini) Code Snippet for Denial of Service(DoS) attack.

Masquerade attack

In a vehicle network, each ECU's has set of CAN ID's which consist of a specific message sent to respective ECU. The receiving ECU will not check from which node the message is receiving. It will receive the message only based on the CAN ID's. So in masquerade attack, an attacker can use the least defensible ECU as an entry point and gain illegitimate access to the ECU's functionalities. The activity diagram shown in Figure 4.16 describes how the attacker performs this attack. The snippet of code used for this is shown in the below figure 4.17. For example, if

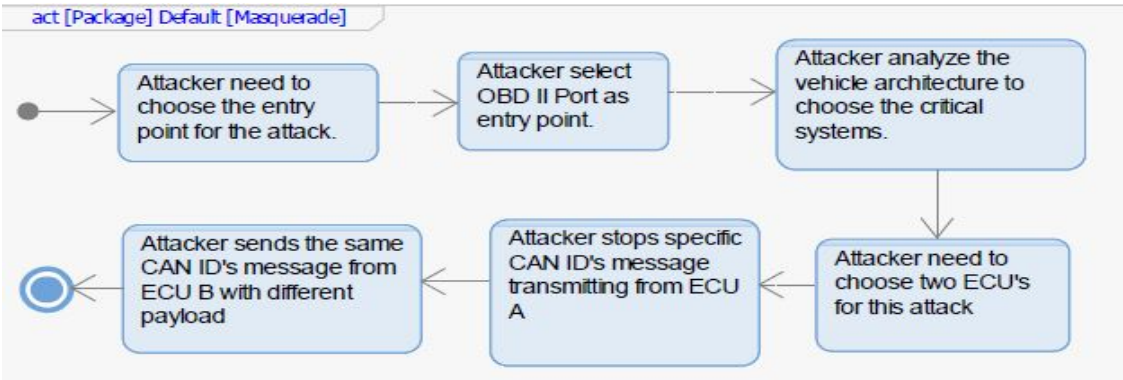


Figure 4.16: Activity diagram for Masquerade attack.

```
#Start of Masquerade attack of Steering Sensor Assembly
#(Example of sending all CAN ID's messages with corrupt payloads from the Steering Sensor Assembly ECU after 2 sec)
CanNetwork.DLC3.sourceApp[0].idDataFrames = "37, 608, 610"
CanNetwork.DLC3.sourceApp[0].periodicityDataFrames = "0.011961, 0.19954, 0.040504"
CanNetwork.DLC3.sourceApp[0].dataLengthDataFrames = "8, 8, 5"
CanNetwork.DLC3.sourceApp[0].payloadDataFrames = "000088007878781d, 0e00000000000078, 0000000069000000"
CanNetwork.DLC3.sourceApp[0].initialDataFrameOffset = "1.171357, 1.213432, 1.21796"
#End of Masquerade attack of Steering Sensor Assembly
```

Figure 4.17: Initialization file(.ini) Code Snippet for Masquerade attack.

an attacker spoof the transmission ECU messages then the vehicle will face an abnormal behavior which will have many effects on the driving behavior.When a car drives in a driving mode, by spoofing the message to change the vehicle to neutral will have a tremendous effect on the cruising vehicle. This scenario has shown in the below Figure 4.18.

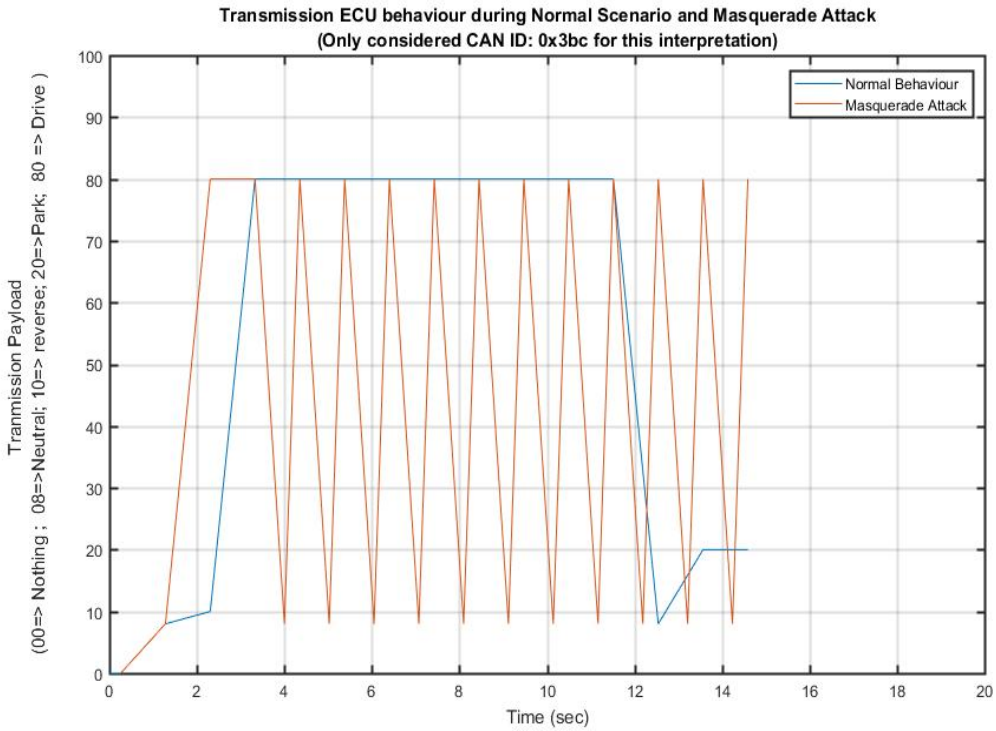


Figure 4.18: Graph represents the transmission ECU behavior during Masquerade attack.

Frame injection

In Prius vehicle network, all the CAN ID's will be periodically sent messages. If some events occurred concerning the CAN ID, it will break the timing manner and sent the messages based on the event. For example, the respective message about Engagement and disengagement of the brakes will be broadcasted to the respective ECU's to every one second. If the brake is engaged or disengaged the message will be sent to the ECU's irrespective of the time. By this, Frame injection attack the attacker can broadcast the wrong information to the particular ECU which will affect the performance of the ECU. The activity diagram for this attack shown in the below Figure 4.19.

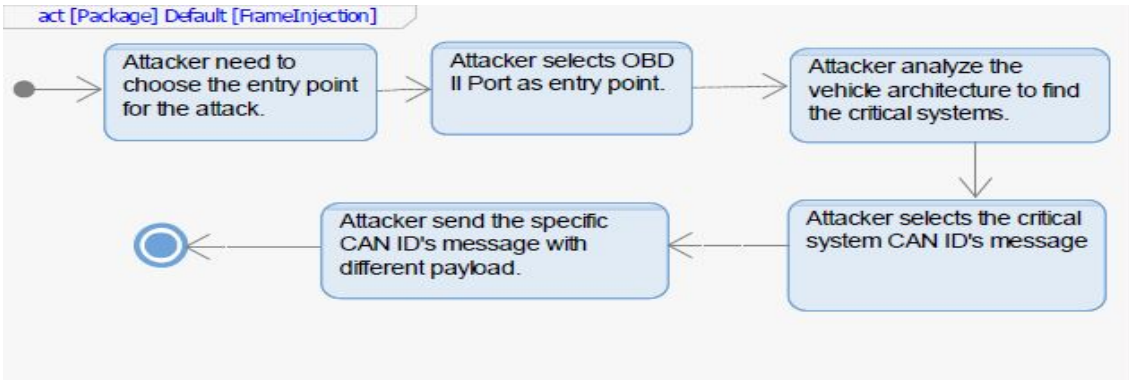


Figure 4.19: Activity diagram for Frame Injection attack.

In this simulation experiment, we will demonstrate the attack with unauthorized brake commands from the DLC3 port which will affect the performance of the vehicle and endanger the passengers. The snippet of code that is used for Frame Injection attack is shown in below Figure 4.21. The below Figure 4.20 shows the normal operation and Frame injection attack.

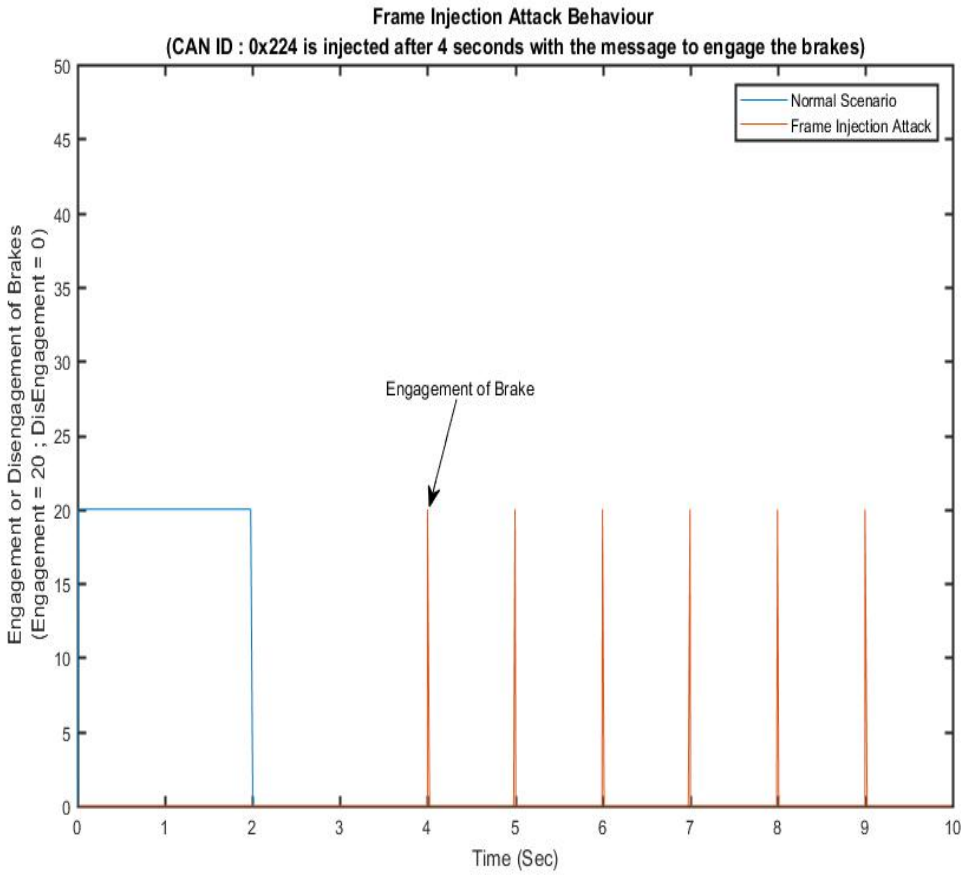


Figure 4.20: Graph shows the behavior in normal scenario and Frame injection attack behavior.

```
#Start of Frame Injection (Example of brake engaged after 4 sec with corrupt payload))
CanNetwork.DLC3.sourceApp[0].idDataFrames = "548"
CanNetwork.DLC3.sourceApp[0].periodicityDataFrames = "1.000000"
CanNetwork.DLC3.sourceApp[0].dataLengthDataFrames = "8"
CanNetwork.DLC3.sourceApp[0].payloadDataFrames = "2000000001bf0008"
CanNetwork.DLC3.sourceApp[0].initialDataFrameOffset = "4.000000"
#End of Frame Injection
```

Figure 4.21: Initialization file(.ini) Code Snippet for Frame Injection attack.

Black-hole attack

In this attack, by sending a specific packet to an ECU will result to drop all the packets in the simulation and stop the simulation from running. By this attack, the vehicle will come to complete halt. In a simulation environment, from DLC port a specific packet is sent to an ECU which results in stopping the simulation. In a real vehicle, it will lead to complete loss of control from the vehicle,

which will cause panic to the driver and passengers. The activity diagram describes the attacker method in the below Figure 4.22.

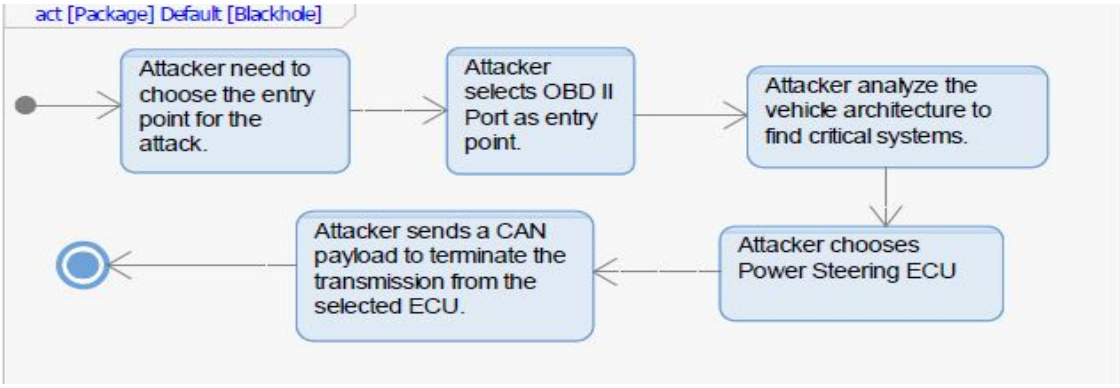


Figure 4.22: Activity diagram for Black hole attack.

In this simulation, a power steering torque request is sent to ECU which leads to a black-hole attack. The reason to choose power steering is due to the periodicity. The periodicity of this message is 0.01 seconds, due to this the attack is hard to detect. The snippet of code that is used for black-hole attack is shown in below Figure 4.23. In the snippet, a dummy message is sent at after 4 seconds from DLC3 to the CAN bus pretended to be Power steering ECU. Due to this the message, the simulation is stopped.

```
#Start of Black-hole Simulation Attack
#(Example of blocking all the Power Sterring ECU CAN Transmissions)
CanNetwork.DLC3.sourceApp[0].idDataFrames = "37"
CanNetwork.DLC3.sourceApp[0].periodicityDataFrames = "0.011961"
CanNetwork.DLC3.sourceApp[0].dataLengthDataFrames = "8"
CanNetwork.DLC3.sourceApp[0].payloadDataFrames = "0000000000000000"
CanNetwork.DLC3.sourceApp[0].initialDataFrameOffset = "4.000000"
#End of Black-hole Simulation Attack
```

Figure 4.23: Initialization file(.ini) Code Snippet for Black-hole attack.

4.3 Conclusion

This chapter explains the reverse engineering that helps to collect the data from the vehicle, which has been used as an input in the simulator to replicate the vehicle network behavior. The tools and methods used during the reverse engineering process has been described and the experiments done in the vehicle to obtain the vehicle data has been discussed in detailed manner. In addition to this, experiments are performed in the simulator to replicate the network traffic in the simulators. From this experiment results, we have concluded that the behavior of the real vehicle can be predicted using the simulator when performing attacks. Various attacks as described in the *Chapter 2* has been performed in the simulator to show the vulnerability of the CAN bus and predicted the behavior of the real vehicle. From this last two research questions, **How to replicate the in-vehicle network behavior in the simulation environment? and What are the outcomes of the attack scenarios in the simulation and what are the possible effects in the real environments based on the results obtained?** have been answered in this chapter.

Chapter 5

Results and Discussion

This chapter presents a summary of the challenges faced during the replicate the vehicle network behavior in a simulation environment and experiments performed in the real car and simulation. In addition to this, the product matrix of the project is also discussed such as Benefits and Limitation of Framework

5.1 Challenges

5.1.1 Selection of Framework

As discussed in chapter 3, the selection of framework for this project is an important thing. Since all the vehicle behavior should be done using the simulation and attacks should be performed in the simulation to know the behavior of the vehicle. The framework is chosen based on two important questions: How complex and large network the environment can be simulated? Moreover, How much output results from the simulation network matches with the real car? While selecting the framework, the first thing was how flexible the environment could be modified for the requirements. Because a lot of environments (such as MATLAB-Simulink, CANoe) don't have the flexibility. Due to this, the more open source environment is considered for this experiment. After some hands-on experience in simulation environment mentioned in chapter 3, OMNNET++ is chosen for this project to replicate the network behavior of the vehicle.

The most challenging phase in the OMNET environment is to get the network behavior result in the readable format (such as .pcap). The most challenge part in recording the network traffic in the simulation is specifying the precise format of the CAN messages in the framework. If the source code is not defined properly for the CAN message format, then the recording packets will be corrupted. So specifying the format is an important thing for recording the packets.

Then simulating the static behavior of the vehicle network is quite straightforward, but simulating the dynamic behavior is a difficult procedure. To simulate the dynamic behavior of the vehicle: the first challenge is to identify the specific module for this operation. Since it is a tedious process, first we simulate the dynamic behavior of the network during recording the CAN messages. Once we understand the effort required to do the dynamic behavior we find the exact place where the real ECU will make the behavior happens. So the dynamic behavior of the network was placed inside the source application of the node. Due to this, the dynamic message is transmitted to the nodes while creating the message itself inside the ECU's.

5.1.2 Experiments

In this section, the challenges faced during the experiments taken place in the vehicle to gather the real-time CAN message data will be discussed elaborately. In addition to this, the challenges faced during conducting attack experiments in the simulation environment is also discussed.

Challenges faced during the reverse engineering process

The first challenge in the reverse engineering process is finding the right equipment and the vehicle. The vehicle manufacturers have their proprietary tools for diagnosing the vehicle malfunction and see the network data and behavior of the vehicle. All manufacturers use their own devices and software. Since the proprietary tools are expensive, some enhanced tools are considered to listening to the CAN network of the vehicle. The tools required that can connect to OBD - II port are grouped into three categories: generic tool, enhanced tool, and OEM specific tool. Generic tools have limited functionality such as diagnostic and OEM specific tools can do everything using its software, but it is too expensive. So the enhanced tool is selected for this project. Since it is cost-effective and required information for this project can be obtained. For this project, CANtact tool is used as mentioned in chapter 4. Then connecting the device to the laptop with accurate software helps to monitor the CAN network in the vehicle.

Challenges faced during performing experiments in the simulation

In OMNET++ performing attack scenario is not an inbuilt feature. So to achieve this, the source code in the simulation environments are modified to this experiments. During this process, some challenges are faced. First, when performing attacks mentioned in chapter 4 the feasibility of entry points in the real -scenarios and simulation environment should be examined. In OMNET++ attacks such as DoS can be performed by changing some variables in the initialization file(.ini) in addition to the original input. However, an attack like a black hole can be performed by modifying the source code in the appropriate area to know the behavior.

Another important challenge is analyzing the results from the simulation environment. So by simulation the attack scenario, the results can be seen in the pcap. However, to get a clear insight of vehicle behavior, the graphical information will easily understand. To achieve this, we want to filter the required CAN ID's and export to the .csv format. To decode the required information from the CSV file, JAVA code is developed, and results are separated such as (Fragment of the payload based on time, number of packets). Based on the fragments of the payload based on time, the graph is generated in the MATLAB to show the difference between normal and abnormality of the vehicle network during the attack. These graphs are shown in the chapter 4 results.

5.2 Product Matrix

5.2.1 Benefits and Limitation of Framework

Benefits

The important benefit of the framework is the flexibility of adding and modifying modules base on the user experience. As discussed in chapter 3, no other framework has such capabilities. By modifying the existing modules source code, we can simulate various scenarios of in-vehicle network behavior based on user requirements and analyze the results more constructively. In addition to this, there is no restriction on adding the extra nodes and various variables in the single initialization file and perform the simulation. This cannot be carried out in simulation environments like Matlab. Another benefit of this framework is other protocols like FlexRay, LIN, and Automotive Ethernet can be simulated in this framework.

Limitation

The limitation of this framework is not able to integrate with the hardware platform (such as MATLAB-Simulink and CANoe). However, to overcome this limitation some background research should be done to connect with hardware. Since for this project this feature does not need, the effort was not taken. However, in future, the framework can be developed to integrate with the hardware and other software.

Another limitation of the framework the number of modules given in the initialization file will be created as instances during the simulation. So the major problem with this feature is the source code of the module is common for every node represented during the initialization file and ned file in the framework. Due to this, the gateway in the CAN network is not replicated in the simulation environment. However, this does not affect the results of the experiments. However, if we are able to overcome this limitation, the in-vehicle network will be more realistic to the original network in the vehicle.

Chapter 6

Conclusion

The ultimate goal of this project is to know the competence of the simulation environment to perform the in-vehicle network traffic, that will equivalent to the real-vehicle network traffic. In addition to this, some attacks are performed in the simulation model to know the behavior of the vehicle in real life from the simulation results. However, to achieve this ultimate goal, we have formulated some research questions in the *Section 1.1 of Chapter 1.* By answering this questions, the targeted outcome is achieved.

1. **What are the different in-vehicle network protocols and architecture used in the automotive system?** Initially, a literature study is conducted to identify various in-vehicle architecture of different manufactures. *Section 2.1 of Chapter 2* described elaborately about the evolution of ion-vehicle architecture and a need for the need for it. Furthermore, the various types of architecture used by manufactures based on the vehicle requirement is explained. The widely used protocols in the in-vehicle architecture such as: CAN,LIN and MOST are also explained.

2. **What are the in-vehicle network entry points for vulnerabilities?** In *Section 2.4 of Chapter 2* the various in-vehicle network entry points are classified based on the type of interface such as: Wired or Wireless. From this classification, every entry points are explained in the detailed manner such as: How they are vulnerable to cyber attacks?. Furthermore, the attacks performed by the researchers via the entry points are also mentioned to know the vulnerability of the system. The entry points of the in-vehicle network will be vary by each vehicle and their architecture. Compromising the different vehicles from the same entry-point will not be possible, since every in-vehicle architecture is different and the entry points are connected to different systems. So while performing the attacks, the attacker also need to know the architecture of the vehicle.

3. **What (and where) traffic, can we monitor and which are the ideal ECUs or bus monitoring points and detection capabilities?** On-board Diagnostic(OBD - II) is the ideal point to capture the network traffic in the vehicle. Since, OBD - II port is mandatory to all the vehicles it's easier to monitor the in-vehicle network traffic. However, they are certain factors should be considered before monitoring the traffic or using the port for detecting the network anomalies. First, the connection between the OBD - II port and rest of the network (such as: ECU's) will be different for each manufactures. For example: The vehicle used for this experiment (Toyota Prius), the OBD - II is connected in the Main CAN Bus and it's considered as an another ECU. So all the CAN network traffic can be monitored using this port. However, vehicle like BMW's the OBD - II port is isolated from the rest of the network so monitoring every traffic will be difficult. The amount of traffic monitoring is also depends upon the device connected to the OBD - II port.

4. **How to replicate the in-vehicle network behavior in the simulation environment?** This question can be refined into three other sub-question, which are:

   - **What is the suitable environment for creating the in-vehicle network model and simulate the network traffic?** In *Chapter 3* various simulation environment was analyzed and compared for this project, to develop and simulate the in-vehicle network traffic. The research was identified OMNET++ as most promising simulation environment for this project. The main reason to choose OMNET++ is due to an open-source platform and able to do modifications based on the user requirements compared to other environments.
   - **What are the extra model based components added in the simulation environment to replicate the real in-vehicle communication behavior?** We created some extra modules and modified the source code in the OMNET++ simulator to log the traffic in the simulation, to perform the vehicle in dynamic behavior and to perform attacks on the simulation model. By this, we able to perform the real vehicle network behavior in the simulation model.
   - **How realistic will be the traffic of the real-vehicle and simulation model in the environment?** In *Section4.2 of Chapter 4* performs various experiments in the simulation environment and compared with the real vehicle traffic. From the experiments, we can say that the traffic of the simulator matches over 92% of the real-vehicle traffic. However, the real vehicle traffic collected using a device called *CANtact* in which we are able to record 89 different CAN ID's traffic, when using some other tools the number of CAN ID's will be increased. In such case, the CAN ID should be analyzed and the new data should be given as input in the simulator. So the simulator can perform as close to the real vehicle in-vehicle traffic.

5. **What are the outcomes of the attack scenarios in the simulation and what are the possible effects in the real environments based on the results obtained?** Based on our experiments and results, the vulnerability of the vehicle during different attacks has been performed in the *Section 4.2.3 of Chapter 4*. Various attacks have been demonstrated in the simulator to predict the vehicle behavior. However, the real outcome in the real scenario will be much more catastrophic or less risk based on the particular type of attack and environment. Nevertheless, the vulnerability of the vehicle entry points in the architecture and protocol will be same.

The contribution of this project is to know the vulnerability of the CAN bus in the vehicle architecture. For this, we have selected the Toyota Prius vehicle to perform the reverse engineering to collect the required data to perform the attacks in the simulators. Furthermore, the result from the simulator during the attacks shows the intended behavior of the car. From this the significance of the developing a safe vehicle from cyber-attacks can be acknowledged.

Bibliography

[1] *The state of the art of electric and hybrid vehicles*, volume 90, Feb 2002. 14

[2] A.Greenberg. Gm took 5 years to fix a full-takeover hack in millions of onstar cars. https://www.wired.com/2015/09/gm-took-5-years-fix-full-takeover-hack-millions-onstar-cars/, sep 2015. Accessed on: 2017-04-10. 1

[3] A.Greenberg. A new wireless hack can unlock 100 million volkswagens. https://www.wired.com/2016/08/oh-good-new-hack-can-unlock-100-millionvolkswagens/, oct 2016. 1

[4] CenterforAdvancedAutomotiveTechnology. *HEV Levels*. 14

[5] Dr. Charlie Miller Chris Valasek. Adventures in automotive networks and control units. techreport, IOActive, 2014. 3, 18

[6] Dr. Charlie Miller Chris Valasek. A survey of remote automotive attack surfaces. techreport, IOActive, 2014. 1, 18

[7] R.B.Boatright C.M.Kozierok, C. Correa and J.Quesnelle. *Automotive Ethernet: The Definitive Guide*. Intrepid Control System, 2014. 6

[8] FlexRay Consotium. Flexray communications system protocol specification version 3.0.1, oct 2010. 6

[9] MOST Cooperation. Most (media oriented systems transport) specification rev. 3.0 e2, jul 2010. 6, 13

[10] Toyota Motor Corporation. 2010 - toyota prius electrical wiring diagram, Aug 2009. 35

[11] Toyota Motor Corporation. 2010 - toyota prius repair manual, Aug 2009. 32, 35

[12] Roderick Currie. Developments in car hacking. resreport, SANS Institute, December 2015. 3

[13] Chris Valasek Dr. Charlie Miller. Remote exploitation of an unaltered passenger vehicle. techreport, IOActive, Defcon 23, Las Vegas, aug 2015. 1, 3, 18, 19, 20

[14] Prof. Dr. Ing Andreas Grzemba (Editor). *MOST The Automotive Multimedia Networks*. Franzis Verlag GmbH, jan 2011. 13

[15] ELM Electronics. Elm 327 obd to rs232 interpreter. `https://www.elmelectronics.com/wp-content/uploads/2016/07/ELM327DS.pdf`. 36

[16] ELM Electronics. Elm327. `https://www.elmelectronics.com/`. 18

[17] Eric Evenchick. Cantact. `http://linklayer.github.io/cantact/`. 18

[18] Aurlien Francillon, Boris Danev, and Srdjan Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *IN PROCEEDINGS OF THE 18TH ANNUAL NETWORK AND DISTRIBUTED SYSTEM SECURITY SYMPOSIUM. THE INTERNET SOCIETY*, 2011. 19

[19] Robert Bosch GmbH. Can specication version 2.0, 1991. 6, 9

[20] Vector Informatik GmbH. Canoe software development tool. `https://vector.com/vi_canoe_en.html`. 25

[21] Vector Informatik GmbH. Vector e-learning for network protocols. `https://elearning.vector.com/vl_index_en.html`. 9, 13

[22] J. Greenough. The connected car is creating a massive new business opportunity for auto, tech, and telecom companies. http://www.businessinsider.com/connected-car-forecasts-top-manufacturers-2015-2, mar 2015. Accessed on: 2017-04-10. 1

[23] Bluetooth Special Interest Group. Basics of bluetooth. `https://www.bluetooth.com/`. 19

[24] Keigo Kawahara, Yutaka Matsubara, and Hiroaki Takada. A simulation environment and preliminary evaluation for automotive can-ethernet AVB networks. *CoRR*, abs/1409.0998, 2014. 4

[25] KeenLab. Car hacking research: Remote attack tesla motors. techreport, Tencent Keen Security Lab, sep 2016. 1, 3

[26] Andreas Kern. *Ethernet and IP for Automotive E/E-Architectures - Technology Analysis, Migration Concepts and Infrastructure.* phdthesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2013. 6

[27] 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 *2010 IEEE Symposium on Security and Privacy*, pages 447–462, May 2010. 18, 19

[28] ALJOSCHA LAUTENBACH. *On Cyber-Security for In-Vehicle Software.* PhD thesis, CHALMERS UNIVERSITY OF TECHNOLOGY, nov 2017. 16

[29] Ivar Roskifte Leikarnes. Modelling and simulating a hybrid electric vehicle. mathesis, The Artic University of Norway, June 2017. 14, 15

[30] Alexis Lekidis. *Design flow for the rigorous development of networked embedded systems. Embedded Systems.* PhD thesis, Universite Grenoble Alpes, 2015. 12

[31] Yap Leong, Akhtar Razali, Gigih Priyandoko, and Nazrul Idzham Kasim. Review on automotive power generation system on plug-in hybrid electric vehicles & electric vehicles. *MATEC Web of Conferences Vol:38 (2016) - UTP-UMP Symposium on Energy Systems 2015 (SES 2015)*, 38:02002, 01 2016. 15, 18

[32] OpenSim Ltd. Omnet++ discrete event simulator. `https://www.omnetpp.org/`. 26

[33] MathWorks. Phyton. `https://nl.mathworks.com/products/vehicle-network.html`. 23

[34] McAfee. Automotive security best practices. Technical report, McAfee. Part of Intel Security, 2015. https://www.mcafee.com/us/resources/white-papers/wp-automotive-security.pdf. 2

[35] Dennis K. Nilsson and Ulf Larson. A defense-in-depth approach to securing the wireless vehicle infrastructure. *JNW*, 4:552–564, 2009. 1

[36] Roman Obermaisser, Philipp Peti, and Fulvio Tagliabo. An integrated architecture for future car generations. *Real-Time Syst.*, 36(1-2):101–133, July 2007. 7

[37] Hamburg University of Applied Science. Core simulation models for real-time networks. `https://core4inet.core-rg.de/trac/wiki/CoRE4INET_Background`. 4, 26, 27

[38] Hamburg University of Applied Science. Fieldbus communication for omnet. `https://core4inet.core-rg.de/trac/wiki/FiCo4OMNeT_Background`. 4, 26, 27

[39] University of Toyota. 2010 toyota prius media preview, February 2009. 15

[40] Pratik Parsania and Ketan Saradava. *Drive-By-Wire Systems In Automobiles*, December 2012. 14

[41] A. Pretschner, M. Broy, I. H. Kruger, and T. Stauner. Software engineering for automotive systems: A roadmap. In *Future of Software Engineering, 2007. FOSE '07*, pages 55–71, May 2007. 1

[42] M. L. Psiaki and T. E. Humphreys. Gnss spoofing and detection. *Proceedings of the IEEE*, 104(6):1258–1270, June 2016. 20

[43] J. N. Chiasson R. H. Staunton C. W. Ayers, L. D. Marlino and T. A. Burress. Evaluation of 2004 toyota prius hybrid electric drive system. techreport, Oak Ridge National Laboratory, OAK RIDGE NATIONAL LABORATORY Oak Ridge, Tennessee 37831, May 2006. 14, 15

[44] INRIA research institute. Real time-at-work. `http://www.realtimeatwork.com/`. 25

[45] Ishtiaq Rouf, Rob Miller, Hossen Mustafa, Travis Taylor, Sangho Oh, Wenyuan Xu, Marco Gruteser, Wade Trappe, and Ivan Seskar. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association. 20

[46] M. Ruff. Evolution of local interconnect network (lin) solutions. In *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No.03CH37484)*, volume 5, pages 3382–3389 Vol.5, Oct 2003. 6, 13

[47] B. Kantor S. Checkoway, D. Mccoy and D. Anderson. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX Security Symposium*, pages 77–92, San Francisco, CA, aug 2011. USENIX Association. 18, 19

[48] SAE. Sae j3061 cybersecurity guidebook for cyber-physical vehicle systems, jan 2016. 1

[49] HARMAN Cybersecurity Solutions. Towersec ecu shield. `http://tower-sec.com/ecushield/`. 3

[50] Till Steinbach, Philipp Meyer, Stefan Buschmann, and Franz Korf. Extending omnet++ towards a platform for the design of future in-vehicle network architectures. In Anna Foerster, Vladimír Vesely, Antonio Virdis, and Michael Kirsche, editors, *Proceedings of the 3rd OMNeT++ Community Summit, Brno, Czech Republic, September 15, 2016*. ArXiv e-prints, September 2016. 4

[51] Sheran Alles Syed Masud Mahmud. In-vehicle network architecture for the next-generation vehicles. techreport, SAE International, SAE World Congress, Detroit, Michigan, April 2005. 7, 8

[52] C. L. Coomer T. A. Burress, S. L. Campbell and C. W. Ayers. Evaluation of the 2010 toyota prius hybrid synergy drive system. techreport, OAK RIDGE NATIONAL LABORATORY, OAK RIDGE NATIONAL LABORATORY Oak Ridge, Tennessee 37831, March 2011. 15

[53] TCPdump Team. Tcpdump and libpcap. https://www.tcpdump.org/. 26

[54] Corey Thuen. Commonalities in vehicle vulnerabilities. techreport, IOActive, 2016. 17

[55] NS tools. Network simulator - 2. https://www.isi.edu/nsnam/ns/. 26

[56] Matthew Shirk Tyler Gray. 2010 toyota prius vin 6063 hybrid electric vehicle battery test results. techreport, U.S. Department of Energy, January 2013. 15

[57] SocketCAN utilities. Can-utils. https://github.com/linux-can/can-utils. 31

[58] Timo van Roermund. Secure connected cars for a smarter world. techreport, NXP Semiconductors, December 2015. 7

[59] Marko Wolf, André Weimerskirch, and Thomas Wollinger. State of the art: Embedding security in vehicles. *EURASIP Journal on Embedded Systems*, 2007(1):074706, Jun 2007. 19, 20

[60] Marko Wolf, Andr Weimerskirch, and Christof Paar. Security in automotive bus systems. In *IN: PROCEEDINGS OF THE WORKSHOP ON EMBEDDED SECURITY IN CARS (ESCAR)04*, 2004. 17

[61] E Yeh, Junil Choi, N Prelcic, C Bhat, and R Heath Jr. Security in automotive radar and vehicular networks. *Microwave Journal*, 2016. 20

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