CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Attack Traffic Generation for Network-based Intrusion Detection System

Master's thesis in Computer science and engineering

Chandrika Neelap
Harsh Vardhan Bhandari

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Attack Traffic Generation for Network-based Intrusion Detection System

Chandrika Neelap
Harsh Vardhan Bhandari

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Attack Traffic Generation for Network-based Intrusion Detection System

Chandrika Neelap
Harsh Vardhan Bhandari

Supervisor: Magnus Almgren, Department of Computer Science and Engineering
Advisor: Hjalmar Wennerström, Joergen Nilsson, Robert Bosch AB
Examiner: Magnus Almgren, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

Abstract

The automotive industry is constantly coming up with technological advances making automotive vehicles a complex system consisting of a multitude of electronic, mechanical, and software components. A critical part of such systems is the electronic control unit (ECU) which is responsible for controlling specific functions. Nowadays, automotive vehicles are equipped with more than 100 ECUs that control a wide range of functions, from essential (engine and power steering control) to comfort (windows, seats, etc) to critical (airbags). The Controller Area Network (CAN) helps the ECUs communicate with one another using a common bus. The CAN bus is a message-based protocol that offers reliable, priority-driven communication of essential control data.

The CAN bus, despite its reliability and efficiency, is prone to a variety of cyber attacks. The vulnerabilities of CAN towards cyber attacks can be reduced by the deployment of an Intrusion Detection System (IDS). IDS detects intrusions by observing the events or by validating the range of different parameters in an attempt to identify malicious content that could potentially be an attack. This acts as a line of defence against cyber attacks and can play a huge role in safeguarding CAN based systems.

In order to check the efficiency and reliability of security mechanisms like IDSs, they must be tested against malicious data to assess their ability to detect various types of attacks. However, the availability of malicious data is not ubiquitous. The objective of this thesis is to investigate a methodology and develop a software that can manipulate and add known attack traffic into already existing data sets. The abilities and effectiveness of this attack traffic generating (ATG) software in mimicking real-life cyber attacks is evaluated through a series of experiments while highlighting its strengths and weaknesses. The experiments reveal that the developed software succeeds in introducing malicious traffic into benign traffic in a random fashion, which mimics real-life attack traffic. The time in which the software introduces these attacks is a function of $\mathcal{O}(n^2)$.

Keywords: Electronic Control Unit, Controller Area Network, Intrusion Detection System, Attack traffic generator.

Acknowledgements

We would like to acknowledge our supervisor from Chalmers University of Technology, Magnus Almgren, our supervisors from Robert Bosch AB, Hjalmar Wennerström, and Jörgen Nilsson for their support. We would also like to acknowledge Sravan Tatipala, PhD Candidate at Product Development Research Laboratory-BTH. Without their guidance, the success of this thesis would be unimaginable.

Chandrika Neelap, Gothenburg, 2023-09-06
Harsh Vardhan Bhandari, Gothenburg, 2023-09-06

Contents

List of Figures

List of Tables

1

Introduction

Automotive systems in vehicles nowadays have become a vital part of life all around the globe. The most common application of automotive systems is the transportation of populations. Modern automotive systems have improved the quality of transportation not just by increasing the range of travel but also by making the journey more comfortable [1]. Vehicles today are data-powered with a plethora of customizable features that focus on improved safety, efficiency, and comfort for the driver and the passengers by communicating with other vehicles and systems. To improve the quality of transportation, automotive vehicles comprise multiple in-vehicular electronic systems. Each system controls a unique function in the vehicle. These electronic systems inside a vehicle are all controlled by Electronic Control Units (ECUs). Due to the increasing need for comfort, a large number of electronic systems now exist inside a vehicle, each controlled by its own ECU [1].

This has led to a drastic increase in the number of ECUs that communicate not just with each other but also with the environment. The most common way for communication between ECUs used in vehicles today is Controller Area Network (CAN). Its real-time properties, simplicity, and low cost make it favorable to use in automotive vehicles.

Improved connectivity is a boon and a bane as it opens up an avenue for adversaries to attack these systems which can lead to catastrophes . Therefore, the need for improved automotive security arises in order to protect the communication integrity inside these systems. The majority of real-world attacks target ECUs connected through CAN [1].

With the rapid development of electronic and smart appliances in modern vehicles, several ECUs are integrated into the conventional CAN bus. It was highlighted by Huang et al. in [2] that the CAN bus does not offer protection against most attacks, and it is vulnerable to a variety of manipulations. Vulnerabilities in an auxiliary onboard appliance can turn it into relays, giving attackers an opportunity to interact with the CAN bus [2]. With this in mind, a lot of security mechanisms such as intrusion detection systems (IDSs) are being developed. To improve the ability of such systems to detect known attacks, they are tested against data sets of attack traffic mixed with benign traffic. A major challenge, both in academia

and industry is the lack of available traffic data, especially data containing known attacks [1]. In this thesis, we aim to come up with a solution for this problem for a Network-based IDS developed for CAN.

1.1 Background and Motivation

The Controller Area Network (CAN) is a network protocol that is commonly used in modern vehicles to communicate between electronic control units (ECUs). It is a message-based protocol that allows for real-time information transmission between various car systems such as engine control, brakes, and safety systems [3]. The CAN protocol has been shown to be efficient and successful in facilitating vehicle-to-vehicle communication. However, its security flaws have become an increasing source of concern [4]. The increasing complexity of modern automobiles, with their various interconnected ECUs, has resulted in an increase in the number of potential attack vectors. As automobiles become increasingly integrated and autonomous, they also become increasingly susceptible to attacks. Unauthorized access to the CAN network can jeopardize vehicle safety, privacy, and functioning, endangering drivers, passengers, and other road users [5]. Malicious actors can launch network attacks by exploiting different vulnerabilities in the CAN protocol, including message spoofing, injection, and alteration [6].

Intrusion Detection Systems (IDS) were created as an important line of protection against CAN-based attacks. IDS scans the network for suspicious activity and seeks to detect and neutralize any malicious behavior [7]. However, the efficacy of intrusion detection systems is dependent on their ability to effectively identify and respond to established attack patterns. The lack of a comprehensive data set of CAN-based attacks makes designing and evaluating IDS for this protocol difficult [8]. A critical component of testing network-based IDS for the CAN protocol is the generation of realistic attack traffic patterns. Finding an innovative approach to generate traffic data that contains established attack patterns, on the other hand, can be difficult due to the limited availability of existing attack traffic data.

The goal of this thesis is to devise a methodology for designing and implementing a software tool for generating synthetic attack traffic patterns that can mimic the characteristics of real-world attacks. It is to analyze attack data characteristics, generate, and change attack traffic in order to test network-based Intrusion Detection Systems over the CAN network. The idea is to develop software that can take a typical traffic trace file as input and insert abnormal traffic patterns using an attack template. The ability to create traffic data with known attack patterns is critical for assessing the performance and efficacy of intrusion detection systems [9]. The developed software tool is tested on it's ability to introduce malicious traffic patterns into normal CAN traffic.

The proposed thesis aims to contribute to the field of CAN network security and the evaluation of intrusion detection systems. By conducting an in-depth analysis of attack data characteristics and developing a tool for generating attack traffic, this

thesis seeks to enhance the ability to detect and mitigate attacks, thereby improving the overall security of the CAN protocol in modern vehicles.

1.2 Aim and Scope

The objective of this thesis is to analyze the characteristics of attack data, generate and modify attack traffic in order to test the network-based Intrusion Detection System (IDS) over CAN. The aim is to develop a framework that can take a trace file with normal traffic as input and insert known malicious traffic patterns using an attack template. Finding a novel way to generate traffic data that contains known attack patterns is difficult due to the limited availability of existing traffic data that contains documented attacks. Additionally, this area of study is not extensively explored in the literature.

The scope of the thesis involves the development of the framework, to introduce attack traffic and its evaluation. Furthermore, the attack traffic in the files should be injected without the need for user inputs indicating where these attacks should be injected.

1.2.1 Objectives

The objectives of the thesis are:

- To perform a thorough literature review on automotive data-communication security, CAN standards and its weaknesses, state-of-the-art electrical and electronics architectures (involving CAN network), protocols such as CAN, network management, IDS, and known attacks on vehicles relating to CAN.
- Review common attack patterns (such as Denial of Service, Flooding), understand their implementation, and select the most suitable patterns to implement in the framework.
- Develop an overarching framework in incremental steps that is capable of a) analyzing trace files, b) generating a set of defined attack profiles, and c) introducing the attack profiles on the selected trace files.
- Generalize the framework to handle any trace file and improve the attack profiles to model an attacker for example by introducing randomness, history, and learning.
- Evaluate the developed framework using defined metrics.

1.2.2 Research Questions

In this thesis, we aim to address three key research questions pertaining to the development of a software tool for generating CAN traffic with known attack patterns.

The objectives of the thesis are explored through the following research questions:

- How can a software tool be developed to generate traffic data containing known attack patterns, based on a trace file containing normal traffic which addresses the challenge of limited availability of such data in the field of automotive security research?

- What are the limitations and challenges of generating synthetic traffic data for CAN-based networks?

- How can the attack profiles be improved to make detection challenging for a security mechanism?

1.2.3 Outline

The subsequent chapters of this thesis are organized as follows. Chapter 1 introduces the research by providing the background and motivation, outlining the aim, scope, objectives, and research questions. Chapter 2 presents a comprehensive review of the literature, identifying gaps and areas for further investigation. In Chapter 3, the conceptual foundation is established, covering the Controller Area Network (CAN) architecture, CAN frames, security vulnerabilities, traffic analysis, pattern recognition, and Intrusion Detection Systems (IDS). Chapter 4 focuses on the implementation, providing an overview of the system, trace file analysis, attack generation techniques, and the evaluation framework. Chapter 5 presents the conducted experiments and their results, including comparisons of trace files, randomness of the output trace file, and performance metrics of the framework. Finally, Chapter 6 concludes the thesis, summarizing the findings, discussing limitations, and suggesting potential areas for future work.

2

Review of Literature

In order to proceed with the project, relevant knowledge of various aspects of automotive systems was required. It was only after we familiarized ourselves with the know-how of autonomous communication protocols and standards that generating attacks became possible. The following ideas discussed in various scientific publications proved useful in reaching the level of information sufficient to achieve the objective of this thesis. Below we first provide a literature review before discussing the most relevant work, compared to our own.

2.1 Literature Review

Initially, a small set of papers were obtained from our supervisor at Bosch, which helped us understand the basic concepts of automotive systems and their security. Through the references of these papers, more related researches were explored that primarily focused on CAN architecture along with its vulnerabilities and the file formats associated with it. It was important to be able to understand the information stored in these files to be able to manipulate them. The various attacks that are prevalent in CAN were identified and how different security mechanisms work to detect these attacks revealed information that was important to identify the characteristics of each attack. To gain an idea of what CAN traffic looks like, various existing CAN datasets were explored that contained both normal and malicious data. Learning how to analyze CAN traffic to extract as much information as possible help in finding different ways to introduce malicious traffic into benign traffic. Before proceeding with the implementation, looking at existing traffic generators for both TCP/IP networks and CAN gave us insights on how to develop the software.

2.1.1 CAN attacks and countermeasures

After gaining enough understanding of the CAN bus, different known attacks were studied in order to gain insights on which attacks prevail in CAN buses and how they can be performed.

Bozdal et al. [10] examine the security challenges associated with the CAN bus. The authors identify various security threats that CAN bus networks are vulnerable to,

such as message injection, replay, and denial-of-service attacks. They also discuss the limitations of current security mechanisms used in CAN bus networks. The paper emphasizes the importance of addressing these security challenges and proposes various approaches to improve the security of CAN bus networks, including intrusion detection systems, protocols for secure authentication, and methods for data encryption. Overall, the paper highlights the need for robust security solutions for CAN bus networks to ensure the safety and reliability of critical systems.

Jo and Choi in [11] present a survey of attacks on CAN. The authors discuss the various types of attacks that CAN networks are vulnerable to, such as message injection, replay, and denial-of-service (DoS) attacks. The paper reviews existing countermeasures used to prevent these attacks. These measures include message authentication, encryption, and even IDSs. The authors evaluate the effectiveness of these countermeasures and highlight their limitations. The authors suggest future research to improve the security of CAN networks.

A related research carried out by Dibaei et al. in [12] covers a wide range of attacks that can be performed on intelligent connected vehicles and measures that can be used to mitigate those attacks. The paper presents attacks that can exploit the CAN bus, the vehicle's ECU as well as the sensors and actuators. It also discusses the potential damage that these attacks can cause, such as risks to safety and financial damage. The paper then describes various defense mechanisms that can be employed to protect intelligent connected vehicles, including secure communication protocols, intrusion detection systems, and hardware-based security solutions. The implementation of attacks on the CAN bus is one of the most important parts of this thesis. Some research papers presented a few common CAN attacks and how they can be performed.

The attack presented by Thirumavalavasethurayar and Ravi in [13] is called a replay attack. This research describes a method for simulating a replay attack on the CAN bus. The paper begins by explaining what a replay attack is and how it can be utilized by an adversary to compromise the security of a CAN bus. Then a detailed description of how the replay attack is implemented is discussed along with its different components like the attacker, the victim, and the replay device. The results of their simulation showed that the replay attack was successful in compromising the security of the CAN bus.

2.1.2 Security Mechanisms for CAN

We also explored the different techniques used in various security mechanisms that focus on protecting the network by detecting malicious traffic.

Laufenberg et al. in [14] have performed a research that presents an approach widely used in automotive and industrial systems for detecting attacks on CAN communication. The proposed approach analyzes the CAN message structure and identifies suspicious patterns that can indicate the presence of an attack using a machine

learning-based classification method to distinguish normal traffic from malicious CAN messages.

Another research presented by Lenard and Bolboaca in [15] proposes a stateful firewall and IDS for the CAN bus. The objective of these systems is to detect and prevent various security threats by inspecting and filtering CAN bus traffic. Additionally, the authors propose a mechanism for storing the data and events generated by the firewall and IDS in a secure manner (secure logging). The paper lays emphasis on the importance of secure logging in IDSs and suggests that future research could be done on improving the efficiency of the proposed system.

Fürst and Bechter [16] talk about the use of AUTOSAR for connected and autonomous vehicles. The paper highlights the benefits of using AUTOSAR for connected and autonomous vehicles, including enhanced scalability, reliability, and security. The authors also discuss the challenges of improving the capabilities of AUTOSAR in the case of such vehicles, like the need for advanced communication protocols. Overall, the paper suggests that AUTOSAR is a promising approach for developing software systems in connected and autonomous vehicles.

2.1.3 Existing CAN Traffic Datasets

After developing techniques for analyzing the traffic, the next step was to see how the extracted information can be used to cleverly inject attack patterns into benign traffic. For this we turned to existing datasets of attack traffic, to get an idea of what normal and malicious CAN traffic looks like.

Hollifield et al. in [17] provides a detailed list of existing datasets for CAN intrusion detection along with their characteristics. It emphasizes on the importance of developing a standardized methodology for the evaluation of an IDS's performance for CAN networks. The paper introduces a new dataset called "ROAD Dataset" for this purpose. The dataset includes both benign and malicious traffic patterns. The authors of this paper present an analysis of the contents of the dataset along with attack frequencies and noticeable patterns.

Zago et al. in [18] evaluate a dataset of CAN messages for the purpose of reverse engineering. This dataset contains over 5,000 CAN messages captured from a real-world vehicular setting. Each message in this ReCAN dataset is unique. The paper describes how the dataset was put together and its characteristics like the distribution of different types of messages and the frequency in which they occur. The dataset helps reverse engineer CAN messages to message signals and types along with decoding the data in the payloads.

Sharafaldin et al. in [19] focus on the development of a new data set for IDS and highlights the importance of such diverse and representative data sets that can be used to train IDS in an effective manner. It highlights the shortcomings of existing datasets for TCP/IP networks and discusses a unique methodology for generating

a new dataset that is more realistic. The process of collecting network traffic data from various sources and preprocessing it is explained in the paper. The authors of the paper use an algorithm to generate synthetic malicious traffic by combining real network traffic with simulated attacks in an attempt to capture real-world attacks in their datasets. The dataset is evaluated by comparing the performance enhancements of IDS when using this dataset against existing datasets.

2.1.4 Analysing CAN Traffic

A crucial part of the project was to analyze the traffic of the CAN bus to find patterns and extract as much information about the traffic and its characteristics as possible and the following researches helped gain insights about the same.

Ezeobi et al. in [20] explore unsupervised machine learning techniques to analyze and understand the meaning and function of CAN signals in the payload. In order to identify signal patterns in the data, clustering algorithms are used to create groups of similar messages. Anomaly detection is used to identify system malfunctions or harmful traffic. The paper shows unsupervised machine learning as a useful tool for reverse engineering CAN messages to establish signal boundaries and identify those signals.

A similar study was presented by [21] which presented A Modular Four-Step Pipeline for Comprehensively Decoding Controller Area Network Data called CAN-D. The pipeline mentioned in this paper includes analysis in four modules: the physical layer, message layer, signal layer, and application layer analysis. Each module has a unique job like identifying the type of CAN messages or decoding the signals in the payload. This helps users gain valuable insights into the functioning of the vehicle's ECUs.

Verma et al. in [22] propose an approach for tokenizing and translating CAN messages in vehicles. This approach is called Automotive CAN tokenization and Translation (ACTT), which makes use of a machine learning algorithm to divide the CAN messages into tokens in an attempt to translate them into a human-readable format. This study describes how ACTT can be implemented and evaluates its performance against a dataset of CAN traffic. According to the results, ACTT shows high accuracy and effectiveness in translating CAN messages in real time.

Young et al. in [23] propose a machine-learning-based solution to reverse engineering of CAN messages. The process is described in four steps including data preprocessing, feature extraction, model selection, and performance evaluation. Then the models obtained by this process are evaluated using a dataset of CAN messages. The results reveal that this approach is effective in decoding CAN messages and performs better than existing methods in terms of accuracy and processing speed. This approach has the potential to enhance the accuracy and efficiency of the analysis of CAN messages.

Lestyan et al. in [24] gives a way of re-identifying drivers based on the patterns of their driving with the help of sensor data extracted from CAN bus logs. This method includes feature extraction, dimensionality reduction, and clustering of the sensor signals. The authors test the effectiveness of their proposed method in its ability to accurately re-identifying drivers from a dataset of real-world CAN logs.

2.1.5 Traffic Generators for TCP/IP

To gain more insights about how to proceed with developing our own framework, we explored various existing work that was similar to what we intend to achieve in this thesis. These researches are performed for different protocols.

Puketza et al. in [25] discuss a software platform developed for testing IDS. The platform tests IDS in a more controlled and realistic manner, focusing on the evaluation of the IDS's performance against different attacks. The paper describes the design and architecture of the software, which has a traffic generator to produce realistic network traffic, an attack generator for generating a variety of distinct attacks, and a data collector for analyzing the IDS's output. The authors discuss the challenges associated with testing of IDS, such as the shortage of standardized datasets and the problem of reproducing real-world attacks.

Behal and Kumar in [26] covers a wide variety of DDoS attack tools, some popular ones are LOIC (Low Orbit Ion Cannon), HOIC (High Orbit Ion Cannon), and XerXes. Other more sophisticated tools like Slowloris and RUDY are also covered in this study. These tools are compared by the authors based on different criteria like supported attack vectors, the intensity of attacks, stealthiness, and ease of use. It analyzes tools like hping, TCPReplay, and Iperf, which are traffic generators used to simulate DDoS attacks for testing and evaluation purposes, and their ability to generate realistic attack traffic patterns.

A study that addresses the objectives of this thesis was performed by Erlacher and Dressler in [27]. The paper presents the design and implementation of GENESIDS, along with its evaluation using several IDS. The system automates the process of generating attacks by utilizing an algorithm, which can generate new attacks by combining existing ones. The authors highlight the importance of testing IDS to detect security threats. The results show that the system is capable of generating a wide range of attacks for testing IDSs.

2.1.6 Traffic Generators for CAN

After gaining insights on what attack traffic generators look like for traditional TCP/IP networks, we explored similar existing work done by researchers for CAN.

A related tool that also generates attack traffic is presented by Huang et al. in [28]. The paper presents a tool called ATG (Attack Traffic Generation) that is designed for security testing of the CAN bus. The tool is developed to generate attack traffic to evaluate the security of CAN bus systems specifically found inside vehicles. The

authors discuss the limitations of current security testing tools and propose ATG as a solution that can generate various attack scenarios.

Yang et al. in [29] focused on the development of a penetration testing platform that can be used to evaluate the security of embedded systems that make use of the CAN bus. The platform is designed to provide a comprehensive and automated testing process that includes various attack scenarios, such as message injection, replay, and flooding. The study highlights the importance of penetration testing platforms that can effectively evaluate their security.

The review of different literature presented different ideas that could be implemented in this thesis and aided in defining the scope of this thesis. The information gained from these articles helped in devising an approach and finding resources that proved critical to the advancement of the project.

2.2 Related Work

The CAN bus is the most commonly used solution in in-vehicle networks. Hence, several platforms for various CAN bus testing hardware and software implementations are already maintained by several commercial vendors. For example, CANoe is a proprietary test software developed by Vector Informatik which is used for developing, testing, and analyzing electronic systems inside of vehicles. It provides a simulation environment that supports multiple bus systems and protocols like Local Interconnect Network (LIN), FlexRay, ethernet, etc. Through CANoe, one can simulate ECUs within a network and analyze them individually to detect anomalies in communication. It comes with tools for monitoring and analyzing communication on the CAN bus and allows users to create complex test scenarios. These test scenarios can be automated and the results can be analyzed. Another tool, closely related to CANoe is CANalyser, also developed by Vector Informatik. It provides similar functionality as CANoe does with the added capabilities of logging and the ease of integration with other tools. UDSim (Unified diagnostic services simulator) is another such tool developed by Vector Informatik. This tool was later extended to work on vehicular networks as well.

There are a number of tools similar to the ones discussed above that have the same functionality and prove extremely useful when it comes to simulating and analyzing in-vehicular networks and systems. BusMaster is another popular lightweight open-source tool released by BOSCH. These tools focus on testing the functionality of the system but have little to no provision for security testing. The focus of this thesis is on generating and injecting attack traffic into the CAN traffic.

There are several research works done on open-source software packages which work with cheap, commonly used hardware configurations. Tools like SavyCAN help record messages in different file formats and have provisions for critically analyzing the data payload of CAN messages which can be later represented visually in the form of plots. Hounsinou et al. in [30] developed a CAN network analysis tool

called CarShark that helps decode and distinguish control messages and provides means for visualizing these messages. SavyCAN allows message filtering based on specific criteria that allow users to focus on relevant data. OCTANE (Open Car Testbed and Network Experiments) is a platform that aims to provide an open and collaborative environment for developers and researchers in the industry. It provides a testbed infrastructure that mimics real-life automotive systems. It allows for its researchers to study and evaluate a variety of automotive network protocols and communication technologies.

There are several attack tools that work over traditional TCP/IP networks. Hping, a command-line tool, generates and transmits customized network packets which can be utilized to create various malicious attacks that can lead to Denial of Service (DoS). A popular framework called Metasploit, which is used for vulnerability assessments and penetration testing, consists of modules that can generate known attacks for different systems from a wide range of attack vectors. Low orbit Ion Cannon (LOIC) is a network stress testing tool that generates high volumes of TCP, UDP and HTTP flood attacks. This tool is misused to perform Distributed DoS attacks (DDoS). Scapy is a python-based interactive packet manipulation tool. It is a powerful tool that firsts sniff packets before manipulating them to perform reconnaissance. Similar to Scapy, Impacket is a python library that provides limited capabilities to the user for packet manipulation. Tor's Hammer is a python-based DoS testing tool that works through the TOR network. It uses random source IP addresses to make it difficult to trace back the attacks to the source. Behal and Kumar in [26] cover many such tools and draw a comparison highlighting the capabilities of each tool.

The above tools provide a means for attacking systems. These tools are attack tools that are different from traffic generators (what we aim to develop in this thesis). Some traffic generator tools for TCP/IP networks are discussed below.

ByteBlower is a tool developed by Excentis for testing the performance of a network. The primary aim of this tool is to perform testing of the devices connected in the network bu generating different types of traffic like UDP and TCP with varying payload sizes and rates. The generated synthetic traffic can simulate VoIP, web browsing and even video streaming. Another tool, called Geist traffic generator (GTG) developed by Geist Technologies is capable of mimicking the traffic patterns of various applications. It supports the testing and analysis of network protocols like IPv4, IPv6, UDP, ICMP, TCP and many more. GTG has features that enable analysis of network characteristics and generating reports of its analysis. Harpoon is a vulnerability scanner tool that scans the network for common vulnerabilities and exposures and any other known security problems.

Along with these, the tools that were initially developed for traditional TCP/IP networks are now being extended to work on CAN networks as well, for example, UDSim. Many python Libraries were developed to support development for CAN. Python-CAN is a widely used library that provides a simple mechanism for sending and receiving CAN messages. This library supports various CAN hardware inter-

faces like socketCAN and CANtact as discussed above. Other tools like CANard, pyvit and canmatrix make it easy for developers to interact with CAN networks to develop applications. CANtools is a library that helps work with CAN databases (DBC files). It gives its users the ability to parse database files in different formats. These database files contain information about decoding CAN messages into human-readable format.

Sharafaldin et al. in [19] work on creating a single dataset that eliminates the shortcomings of older datasets which proved to be outdated and unreliable. This new dataset contains seven new and common attack patterns which are introduced into benign data by simulating them in order to meet real world criteria. This dataset is developed for TCP/IP networks. This work differs from the software tool developed in this thesis as it does not focus on CAN networks. Moreover, the software tool developed in this thesis generates a completely different output log or dataset on every iteration.

There are a number of tools that have been developed to generate attack traffic specifically for in-vehicular networks. They have similar capabilities as the tools discussed above but were developed with CAN in mind. CANard is an open-source Python-based framework designed for CAN. It has the ability to send and receive CAN messages through hardware interfaces and can manipulate these messages as desired. This tool can generate custom CAN traffic and mimic different scenarios which makes it extremely useful for testing and developing CAN-based environments. SocketCAN is another such tool that have similar capabilities but additionally is equipped with mechanisms for message filtering and routing. The application can use CAN identifiers and other fields of the frame to set filters to gather messages of high relevance. CANtact is a tool developed by Erik Evenchick that has gained popularity among developers. It provides a USB-to-CAN interface that connects to the USB port to interface with CAN networks. Additionally, it enables its users to capture CAN traffic and analyze the traffic which proves extremely useful for debugging and security analysis of CAN networks.

Apart from existing tools and libraries, there are several research projects that aim to address similar problems as discussed in this thesis. One such research was performed by Huang et al. in [28]. They work on developing an Attack Traffic Generation Tool (ATG). ATG provides a free and functional toolkit for automotive security researchers for easy and effective interaction with real or simulated CAN buses. This tool generates attack traffic for the evaluation of security mechanisms developed for CAN systems. The authors compare their work with other developed tools and highlight their strengths and weaknesses. The attack generation capabilities of ATG differ from that of the software tool discussed in this thesis. ATG takes a set of attacks and injects them into a log file which is then sent to the CAN bus in real time. Each attack is configured before hand and a fixed number of such attacks are injected into the log files in fixed intervals of time. The software tool developed in this thesis is capable of analysing the CAN log files and based on it's analysis, a variable number of malicious attacks are injected at varying intervals of time.

Another study is performed by Erlacher and Dressler in [27]. The tool developed by the authors is called GENESIDS that automatically generates user defined HTTP attacks. This allows network traces to be created in a straightforward manner. The tool depends on the rules present in Snort (NIDS) to generate attacks that would trigger the corresponding rules. This tool differs from the software tool developed in this thesis as it is not designed for automotive networks. Additionally, the tool developed in this thesis does not rely on rules to generate attack traffic, instead it uses information obtained by analysing CAN logs to generate attack traffic.

A similar study by Palanca et al. [31] explores a specific type of cyber-attack on vehicular networks that operates on the link layer which is responsible for the communication between devices connected in the network. This attack is shown to be stealthy as it is designed to be difficult to detect and trace back to the source. It severs communication between ECUs and provides a detailed explanation of how it does so along with some countermeasures against this attack. This attack selectively targets ECUs, which makes it all the more dangerous as it can disrupt communication between ECUs performing critical tasks which can cause a lot of damage.

Radu and Andreea-Ina in [32] focus on the implementation of security measures in order to protect the in-vehicle network from unauthorized access, potential attacks, and data breaches. The paper discusses different layers of security, including the architecture of the network used, authentication mechanisms, and intrusion detection systems. In any security-related research, the objective is to ensure the safety and integrity of communication within the in-vehicle network. The paper dives into different security techniques and best practices that can be adopted to safeguard any system against cyber threats, such as firewalls, IDS, secure boot processes, access control mechanisms, and secure update mechanisms for ECUs.

There are many other pieces of research that have not been mentioned in this section that have contributed to the field of automotive security. All these studies have been a source of motivation to proceed and find answers to the problems discussed in this thesis. In order to know how the framework was developed the right knowledge of some concepts is required which are presented in the following chapter.

3

Conceptual Foundation

3.1 Controller Area Network (CAN)

The Controller Area Network was an idea birthed by engineers at the Robert Bosch GmbH in Germany in February of 1986. Their aim was to devise a system that would enable communication between multiple ECUs in vehicles. Since then, the protocol has found its way into every mode of transport from cars to trains to even ships. Every modern automotive system has at least one CAN network installed. CAN is a bus-based protocol that has proved to be a very reliable communication protocol for communication between vehicles and their surroundings [10].

The CAN bus uses serial communication which reduces the number of wires inside the system. Even though this was not the main intention when developing this protocol, it proved to be a useful by-product of the protocol. The use of multiple processors improves the performance of the system. The decrease in the cost of microcontroller chips at that time made a multi-processor architecture in a single system feasible. CAN was originally devised with automotive systems in mind but today it has found its way into all the fields where inter-microprocessor communication is required.

3.1.1 CAN Architecture

CAN is a 2 wire half-duplex high-speed network system that is suitable for High-speed real-time applications as it has low memory and CPU requirements. It also provides collision detection and prevention as it makes use of CSMA/CD. Every ECU must wait for a specific period of inactivity on the bus before a message can be transmitted by the same ECU. Every message is given a unique identifier which also contains the priority of the message, which is used to resolve any collisions that might occur. Lower values in the identifier indicate higher-priority messages. The implementation of the CAN bus is straightforward. It bypasses 4 layers of the OSI model (presentation to the Network layer). By doing so it saves memory resources and gains performance. CAN operate both on a datalink and physical layer.

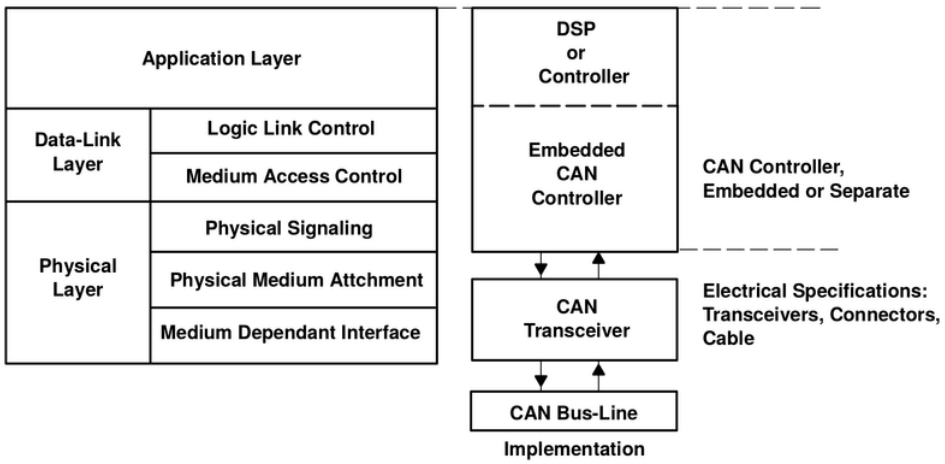

Figure 3.1: CAN architecture as compared to the OSI model, taken from [33]

Every entity participating in communication over the CAN bus is called a node. A CAN node is a functional ECU that participates in the CAN network. Each node can send/receive a different number of messages onto the bus. The frequency of sending and receiving differs greatly as well.

To participate in CAN communication every ECU should have a CAN interface that comprises a CAN controller and transceiver. The CAN Controller is responsible for processing information to and from the CAN bus. The CAN bus is the physical transmission medium that links all the participating nodes via a CAN interface. The transceiver connects the controller to the physical transmission medium.

In order to transmit signals onto the bus, the transceiver is equipped with 2 bus pins, the CAN (High) and CAN (Low) pins. In order to signal the logical 0 and 1 signals, the transceiver creates a differential voltage i.e., the difference in voltage between CAN (High) and CAN (Low). The dominant bit (Logical 0) is assigned when a differential voltage of 2 V is observed and the recessive bit (Logical 1) is assigned when a differential voltage of 0 V is seen.

The transmission of CAN messages does not follow any time sequence but rather is event-driven. The communication channel is only busy when there is a message to be transmitted. This makes access to the bus quick. Every CAN message on the bus can be received by any connected CAN node (broadcasting) if the message can be identified using a valid unique message identifier. A node can choose to either accept or reject a CAN message based on the relevance of the message to that node.

3.1.2 CAN Frames

Messages in CAN are enclosed in a frame. The frame consists of various fields that facilitate the successful transmission of data over the bus. Figure 3.2 shows the format of CAN frames.

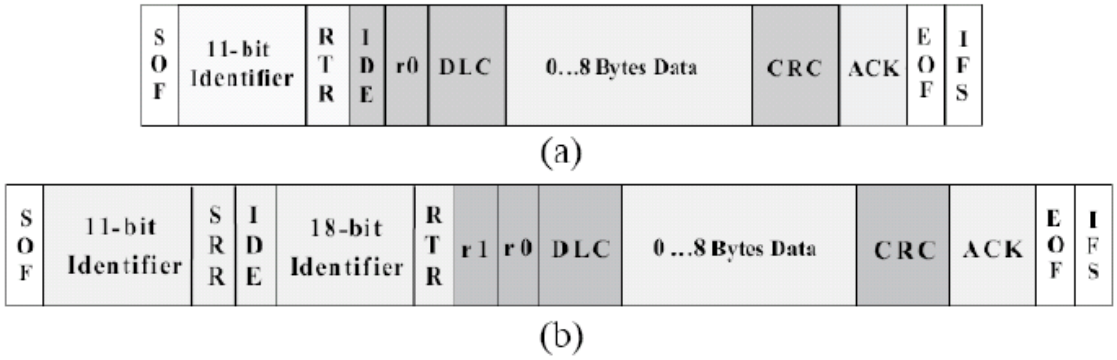

Figure 3.2: (a) standard CAN frame, (b) extended CAN frame, taken from [34]

- **SOF:** start of frame marks the start of a message.
- **Identifier:** 11 bits, establishes priority of each message. Lower the binary value, higher the priority.
- **RTR:** 1-bit, Remote transmission request, set to 0 when information required from another node.
- **IDE:** 1-bit, identifier extension. 0 means standard CAN msg with no extensions transmitted.
- **r0:** Reserved bit.
- **DLC:** 4 bits, Data length code. Number of bytes being transmitted.
- **DATA:** 64 bits data.
- **CRC:** 16 bits, checksum for error detection.
- **ACK:** Originally contains 1, if the message is valid and error-free all recipients overwrite this to 0. If not, all nodes overwrite this bit, then the message is discarded, and the sender re-sends the message after corrections. All nodes acknowledge the integrity of message.
- **EOF:** 7 bits, end of CAN data frame.
- **IFS:** 7 bits, stores time required by controller to move a correctly received frame to its proper position in message buffer area.

An extended CAN message has all the above fields in the data frame along with the following additional ones:

- **SRR:** Substitute remote request. 0 means standard CAN message and 1 means extended CAN message.

- **IDE:** 1 in IDE implies there are more identifier bits to follow.
- **r1:** additional reserve bit.

The ID in the extended format of CAN has 29 bits as compared to the standard 11 bits. If the last bit of EOF is 1 then the message is error free and the frame is valid. If 0 is the last bit in the EOF field then the transmission of the message is repeated.

In the CAN protocol, there are 4 different types of frames that can be transmitted.

- **Data Frame:** Initiates, and maintains communication between partners. Transmits and protects user data, and establishes communication relationships defined in the communication matrix. The RTR bit is 0.
- **Remote frame:** Request for transmission of data from another node. Like data frames except do not carry any data. The RTR bit is set to 1. Not used often. Lacks the data field but otherwise the frame format is the same. CAN controllers respond to a remote frame by sending the desired data frame.
- **Error Frame:** Violates formatting rules of CAN messages as it is a special message. Transmitted when an error in a message is encountered in which case all other nodes are also forced to transmit the error frame. The original transmitter then transmits the re-transmit message. A node cannot tie up a bus by repeatedly transmitting error frames.
- **Overload Frame:** Special message that is transmitted when a node becomes too busy. An extra delay of messages is then implemented.

3.1.3 CAN Security and Vulnerabilities

The increasing connectivity of vehicles has exposed Controller Area Network (CAN) networks to various security challenges. While CAN was originally designed with a focus on reliability and efficiency, it lacks built-in security mechanisms. CAN networks are vulnerable to a range of attacks that can compromise the confidentiality, integrity, and availability of communication. This makes it susceptible to a range of security vulnerabilities and attacks [35].

One of the primary vulnerabilities in CAN networks is message injection. An attacker can inject malicious messages into the network, either by compromising a legitimate ECU or by gaining unauthorized access to the network. An attacker can also impersonate a legitimate ECU by sending messages with a spoofed source identifier. This can deceive other ECUs into accepting and executing malicious commands, potentially compromising the integrity and safety of the vehicle's operations [35].

An attacker can intercept valid CAN messages and replay them at a later time. This

can result in the re-execution of previously executed commands, leading to unwanted or dangerous actions by the ECUs [35]. Denial of Service (DoS) attacks pose yet another threat to CAN networks. By flooding the network with excessive messages or by targeting specific ECUs, an attacker can disrupt the normal functioning of the network. This can result in the loss of critical information or the unavailability of vital vehicle functionalities [35]. Understanding these vulnerabilities is essential for developing effective security mechanisms and intrusion detection systems.

3.2 Traffic Analysis and Pattern Recognition

Traffic analysis and pattern recognition techniques play a crucial role in identifying anomalies and detecting attacks in Controller Area Network (CAN) networks. By analyzing the parameters and contents of CAN messages, these techniques enable the identification of patterns indicative of normal or malicious activities [36].

One aspect of traffic analysis is the examination of message parameters, such as the identifier, data length code, and data field of CAN messages. Analyzing the identifier can reveal patterns in the message sources or identify messages associated with specific functionalities or ECUs within the network [36]. Analyzing DLC distribution can help identify abnormal message sizes. Analyzing data field contents can provide insights into the nature of the messages and the information being communicated [36].

Furthermore, analyzing the contents of the data field can involve examining specific data bytes or bit patterns within the payload. CAN messages often carry critical information, such as sensor data, vehicle status, or control commands [37]. By analyzing the content of these messages, patterns associated with normal or malicious behaviors can be identified. For example, abnormal values or ranges in sensor data can indicate sensor tampering or spoofing attacks. Similarly, the presence of specific bit patterns or command sequences can be indicative of unauthorized access attempts or malicious commands [37].

Pattern recognition techniques, including machine learning algorithms, can be applied to analyze the parameters and contents of CAN messages [38]. These algorithms can learn from labeled data to identify patterns associated with normal or malicious behavior. By training models on a dataset containing both normal and attack traffic, the algorithms can learn to distinguish between them and detect new or previously unseen attack patterns. Machine learning algorithms such as Support Vector Machines (SVM), Random Forests, and Neural Networks have been applied to analyze CAN message parameters and contents with good results [38].

Combining traffic analysis with pattern recognition techniques allows for a comprehensive understanding of the CAN network's behavior and the ability to detect anomalies and attacks in real time. By examining message parameters and contents, researchers and practitioners can gain insights into the normal operation of the network, identify deviations from expected behavior, and detect potential security

threats.

3.3 Intrusion Detection Systems (IDS)

Intrusion Detection Systems play a vital role in safeguarding CAN networks by monitoring network traffic and detecting suspicious activities or attacks. There are two main categories of Intrusion Detection systems that are used in CAN networks: IDS and IPS.

An Intrusion Detection System (IDS) in the context of CAN refers to a security mechanism that monitors the network for any suspicious or malicious activities. Its primary purpose is to identify potential intrusions or attacks targeting the CAN bus. The IDS analyzes network traffic, examines CAN message payloads, and compares them against predefined patterns or known attack signatures to detect any anomalies or deviations from expected behavior. When an intrusion is detected, the IDS can generate alerts or take preventive actions to mitigate the potential impact of the attack [39].

On the other hand, an Intrusion Prevention System (IPS) goes beyond detection and aims to actively prevent or block intrusions in real time. It operates in a similar manner as an IDS by monitoring network traffic and analyzing message payloads. However, an IPS is equipped with the ability to take immediate action to prevent or mitigate potential threats. It can automatically generate and deploy security policies or rules to block malicious messages or suspicious activities. By actively preventing intrusions, an IPS enhances the overall security posture of the CAN network [40].

IDSs in CAN networks employ various techniques for traffic monitoring and analysis. These include deep packet inspection, statistical analysis, machine learning algorithms, and rule-based systems. Deep packet inspection allows for the detailed analysis of CAN messages, examining their content and identifying any suspicious patterns or anomalies [40]. Machine learning algorithms can be employed to train these models using labeled or unlabeled data, enabling the detection of unknown or emerging attacks. Rule-based systems utilize predefined rules or conditions to identify specific types of attacks based on their characteristic features [40].

By deploying IDSs in CAN networks, organizations can enhance the security of their vehicle systems, detect and respond to potential intrusions, and protect the integrity and safety of the vehicles and their occupants [39].

3.4 Attack Traffic Generation

To analyze the security posture of CAN networks and gauge the efficacy of intrusion detection systems, realistic and representative attack traffic must be generated. There are numerous methods for producing attack traffic [41]. To mimic particular types of attacks, attack templates offer predetermined attack patterns or scenar-

ios. Researchers can create traffic that simulates actual attack scenarios using these templates, which capture the traits and behaviors of known attacks. The addition of anomalies, changing packet parameters, or introducing differences in timing and payload content are examples of various methods that can be used to alter regular traffic patterns, one of which we'll be using in our thesis as well [41].

Using this method, researchers may simulate complex and nuanced attack changes to assess how resistant IDSs are. Furthermore, by using machine learning algorithms to study and imitate well-known attack patterns, attack traffic that closely mimics actual attacks can be produced [41]. Researchers can create traffic that displays similar traits and behavior as seen in actual attacks by employing machine learning models trained on historical attack data [4]. Generating accurate and representative attack traffic is crucial for evaluating the detection capabilities of IDSs and improving the overall security of CAN networks [41].

3.4.1 Characteristics of Attack Data

Understanding attack data characteristics is critical for generating realistic and representative attack traffic. While it is challenging to guarantee an exact replication of actual attack situations, certain factors of attack data, such as packet size, timing, payload content, and traffic patterns, are taken into account to generate attack traffic that closely resembles real-world scenarios. These features can be better understood by analyzing previous attack data, which allows the building of models that are suitable for producing attack traffic [42].

Studying the statistical properties of attack packet sizes, for example, can be helpful in determining the size distribution that should be replicated in generated traffic. Analyzing the historical trends of attack traffic, on the other hand, can help generate traffic with realistic timing characteristics [42]. The created attack traffic can closely match actual attack scenarios by capturing the nuanced aspects of attack data, resulting in more accurate and appropriate evaluations of IDS and CAN network security [42].

3.4.2 Experimental Evaluation and Validation

Controlled experiments and precise validation methodologies are required to evaluate the efficacy of attack traffic creation techniques and intrusion detection systems. In experimental evaluations, synthetic attack traffic is created using a variety of methodologies, and the effectiveness of IDS in identifying and preventing such attacks is evaluated [43]. The effectiveness of IDS and security measures is evaluated by performance assessment metrics like detection accuracy, false positive rates, and reaction time.

These measurements also help validate the created attack patterns. For reliable and accurate evaluations, adequate experimental design, including relevant datasets and plausible attack scenarios, is essential [43]. The robustness and reliability of the

experimental evaluation procedure are also increased by comparing it against current intrusion detection systems and comparing results with approved requirements [43].

3.4.3 Trace Files

Trace files help record communication that takes place over the CAN bus and store the streams of bytes in human-understandable formats. There are several trace file formats that are of high relevance to all automotive systems. In this research, 3 types of trace file formats were observed:

DBC (Database for CAN) is a standardized file format developed by Vector Informatik GmbH for interpreting communication data in Controller Area Network (CAN) networks. It serves as a protocol specification, converting raw CAN bus data into understandable values by extracting signals from bytes. DBC files are widely used for configuring ECUs, analyzing CAN bus traffic, and developing software applications. They define message identifiers, signal definitions, scaling factors, and message encoding rules. DBC files are proprietary and specific to each manufacturer or tool provider.

ARXML (AUTOSAR XML) is another standardized file format used in CAN networks, particularly in the context of AUTOSAR (AUTOmotive open System ARchitecture) systems. It represents the configuration of an ECU and contains detailed information about message structures, signal definitions, and parameter values. ARXML files are crucial for integrating and configuring AUTOSAR-based systems, allowing the exchange of ECU configurations between different tools and platforms. They follow the AUTOSAR architecture, ensuring compatibility and interoperability across various automotive systems.

ASCII (American Standard Code for Information Interchange) files are a text-based file format commonly used for storing CAN network communication data. Unlike DBC and ARXML files, ASCII files do not provide detailed message and signal definitions. Instead, they present CAN data as a sequential list of messages with timestamps, making them human-readable. ASCII files capture the raw streams of CAN messages, including message identifiers, data payloads, and timestamps. As can be seen in Figure 3.3, the second column displays the timestamps, followed by the bus number and message identifiers in the subsequent columns. These identifiers hold the priority assigned to each CAN message. The last columns represent the payload length in bytes, and the message payload is shown in the final column in hexadecimal format.

```
1  0.000000  2  00000220x       Rx   d 8   29 c5 26 55 6a 67 02 5d
2  0.000868  1  000004b1x       Rx   d 8   5e 51 cf b7 4c 99 aa 97
3  0.001212  1  000002b0x       Rx   d 5   3a ff 00 07 2c
4  0.001452  2  00000165x       Rx   d 8   00 08 80 02 00 00 0c 86
```

Figure 3.3: Snippet of an ASCII format tracefile

While they lack the decoding rules and structure provided by DBC and ARXML files, ASCII files are useful for logging and analyzing CAN bus traffic. They offer a convenient way to examine the sequence of messages, identify anomalies, and gain insights into the behavior of the CAN network.

There are a lot of open-source tools that hackers use to analyze captured CAN traffic. CANalyser, CANoe, and Raptor-CAN are some popular ones. The captured data contains similar information about the CAN network as that of an ASCII file which means an attacker would only have access to messages as they appear in an .asc file, another reason why ASCII trace files were considered for this research.

3.5 Attack Models

This thesis focuses on the study of the following selected attacks that are prevalent on the CAN bus. Some of these attacks were introduced in the form of malicious traffic into trace files for the purpose of simulating attack traffic and testing the reaction of the Intrusion Detection Systems toward these attacks.

In section 3.5.1 we introduce we introduce our envisioned attacker. In the following subsections, we then first describe the general attack, followed by how our attacker would use it.

3.5.1 Attacker Approach

For the purpose of this thesis, let us consider an adversary named BhanuPratap that is trying to exploit an automotive vehicle system in order to maximize the damage caused. BhanuPratap is a skilled adversary that understands the architecture of the CAN. The adversary has the ability to sniff the data frames being transmitted over the bus and the tools to translate the messages into an understandable format. By observing these messages, the attacker can understand the values set in the data frame and can interpret signal values that are critical in determining the state of the automotive system. BhanuPratap understands the impact that changing these signal and data frame values will have on the system.

There are other ways to access raw CAN data and make sense of the information in the messages. ARXML and DBC files give a detailed description of the data present in the messages along with the different signal values present in them. These files contain information on how one can convert raw CAN bus data into human-readable values. However, it's unlikely for an adversary to have access to these files as they are proprietary, and hence the adversary's access is limited to only the raw information.

Table 3.1: List of studied attacks

Sr.No.	Attack Name
1	Replay Attack
2	Denial Of Service (DoS)
3	Spoofing Attack
4	Fuzzy Attack
5	Flooding Attack
6	Isolation Attack
7	Overwrite Attack

3.5.2 The Replay Attack

In the case of a traditional TCP network, a replay attack is defined as a network attack where valid transmissions of data are captured and repeated or delayed with malicious intent. It is performed by an adversary by intercepting data and re-transmitting it at a later instance in time. This attack is often a part of Spoofing attacks where an adversary tries to trick the system into believing that the attacker is a trusted entity in order to gain illegitimate access.

CAN is a message-based protocol with no provision for addressing the nodes. This makes transmission of malicious messages easier and furthermore, the identification of the source of these malicious messages becomes difficult. Replay attacks can occur over secure systems that use encryption, and this makes them all the more dangerous and a good choice for adversaries like BhanuPratap. In order to prevent replay attacks, timestamps, and unique sequence numbers can be used.

3.5.3 Denial of Service

The aim of the DoS attack is to make the system resources unavailable to its intended users. This is done by disrupting the services of a host connected to the network either temporarily or indefinitely. Practically, this is achieved by flooding the target machine with trivial requests to an extent where it is incapable of handling or accepting new requests. By doing so the system becomes incapable of processing requests coming from legitimate users. In the case where an adversary has control over multiple systems, they can send requests to the victim through different sources which produces the same effect but is a faster way of attacking the victim. This type of attack is called Distributed Denial of Service or DDoS.

A CAN node broadcasts its message over the bus so every node receives the message sent by every other node. Broadcasting a high volume of messages causes the bus to overload. Collisions are possible over the CAN bus. In order to detect and resolve collisions, CAN uses message identifiers to assign priorities to every message. The lower the value of the identifier, the higher the priority of the message. In case of a collision, the lower-priority messages are simply discarded. An adversary like

BhanuPratap can use this to their advantage to craft messages with small identifier values in order to produce a Denial of Service attack.

3.5.4 Spoofing Attack

The aim of spoofing is for an adversary to masquerade as a trusted entity. The aim of such attacks is to hide the adversary's identity while exploiting resources the victim has access to for personal gain. Spoofing can be done in a plethora of ways, by sending fake emails and text messages, IP spoofing, etc. Man in the middle is also a variant of spoofing attack that involves three parties, the user, the server, and the adversary. The adversary makes an individual connection with the server and the user and relays the messages between them, tricking them into thinking that the user and the server are communicating over a secure channel where in reality the attacker can listen to the entire conversation.

Since CAN provides no means of authentication, i.e. there is no mechanism to determine which message comes from which CAN node, this type of attack becomes easier to perform. BhanuPratap can compromise an ECU and send data frames with a modified ID field with malicious intent in order to achieve a desired effect or cause damage to the system.

3.5.5 Fuzzy Attack

Fuzzy attacks are used to gather information about the system in an attempt to find an entry point to exploit. Fuzzing attacks are performed with the aim of applying pressure to a system causing it to behave unexpectedly. In practice, fuzzing involves feeding the system invalid or random data as input and observing the changes in the state of the system caused by these inputs in an attempt to find a vulnerability. This attack can also be used by professionals to assess the security of a system by identifying vulnerabilities in systems and reporting them which in turn can help improve the security of the system. Fuzzing points out the vulnerabilities and shows how an adversary can interact with it to exploit the system. It also demonstrates the impact of fixing the found vulnerabilities on the security of the system.

In an automotive system, an attacker like BhanuPratap focuses mainly on randomizing the IDs and data of every message that they inject. Consider a fuzzer, which is a program that generates messages with randomized data and ID repeatedly. The attacker observes the system for every such injected packet. If the system proceeds to exhibit normal behavior the randomized fields do not seem interesting. However, if the system behaves unexpectedly due to the injected message, then the randomized field values are saved and injected again to monitor the changes closely.

3.5.6 Flooding Attack

Flooding attacks are a type of DoS attacks which aim at overwhelming a network or a server with a large number of requests or traffic. By sending in traffic in such high volumes, the attacker makes the server or network exceed the limits of processing traffic rendering it incapable of handling any more traffic. By doing so, legitimate users become unable to access the services the network or server is supposed to provide. A DoS attack can take many forms and a flooding attack is one of them. The aim of most DoS attacks is to exhaust the resources of a server so that legitimate users cannot access them. The way flooding attacks achieve the same is through sheer traffic volume. All flooding attacks are DoS attacks but not all DoS attacks are flooding attacks. Furthermore, flooding attacks can be of various types like UDP, Ping, SYN, and HTTP flood where a large volume of their respective request types are generated and sent to the system.

In the case of CAN, BhanuPratap sends a large number of messages to the CAN bus, more than what the network can handle resulting in a DoS attack. The impact of a CAN flooding can be catastrophic, as it can lead to the loss of vehicle control or the disruption of critical industrial processes. This can cause the system to malfunction and can even cause safety hazards which may lead to loss of life.

3.5.7 Isolation Attack

In an isolation attack, an attacker gains access to the victim and then attempts to cut off its communication from the rest of the network. There are various methods in which an adversary can perform this attack, like, by disabling network interfaces or creating rules in the firewall that prevent incoming and outgoing traffic to and from the victim. The objective of an isolation attack is to isolate the victim from other devices and systems, increasing the difficulty of detecting and responding to the attack.

In the case of CAN, this attack aims to isolate one or more ECUs from the rest of the network, thereby disrupting the functionality of the entire system. BhanuPratap has various means of performing this attack on the target system. Once an ECU is isolated, BhanuPratap can cause a lot of damage by modifying its firmware or configuration. Since, there might be some ECUs that rely on signals coming from other ECUs, an isolation attack can have serious consequences. Isolating an ECU that sends out signals which other ECUs rely on can cause the entire system to fail.

3.5.8 Overwrite Attack

An overwrite attack is a type of attack where an attacker maliciously modifies the content of a CAN message to overwrite or manipulate critical data within the message [44]. This attack aims to disrupt the normal operation of the CAN network

and can have severe consequences in automotive systems. Each message has a fixed data field size, which contains the payload or information being communicated. The way this attack is performed in the case of in-vehicular networks is by creating a malicious message with the same parameters as the targeted message. The data field of the malicious message is altered.

This malicious message is sent right before the targeted message, so the system accepts the malicious message and adopts the signal values of the malicious message instead of the legitimate message causing the system to exhibit a different behavior than expected. The impact of an overwrite attack in a CAN network depends on the nature of the modified data. For example, in a vehicle system, an attacker like BhanuPratap could modify the sensor data being transmitted over the network, leading to inaccurate or misleading information being processed by the receiving nodes or an attacker could overwrite a message that contains signals for the speedometer reading of a car, causing it to display incorrect speed information to the driver which could lead to over speeding or excessive breaking. Overwrite attacks can potentially result in improper vehicle operation, compromised safety, or unauthorized control of certain vehicle functionalities.

4

Design and Implementation

.

This section of the report discusses the design and implementation, as well as the specific methods used in the process of developing the final framework. Additionally, ways of evaluating the framework's performance along with the results it produces will be elaborated on in a step-by-step manner.

4.1 System Overview

In order to understand the working of the framework, a high-level description of its different components is essential. The objective of this thesis is to develop an attack traffic generator that introduces attack traffic into the available trace files.

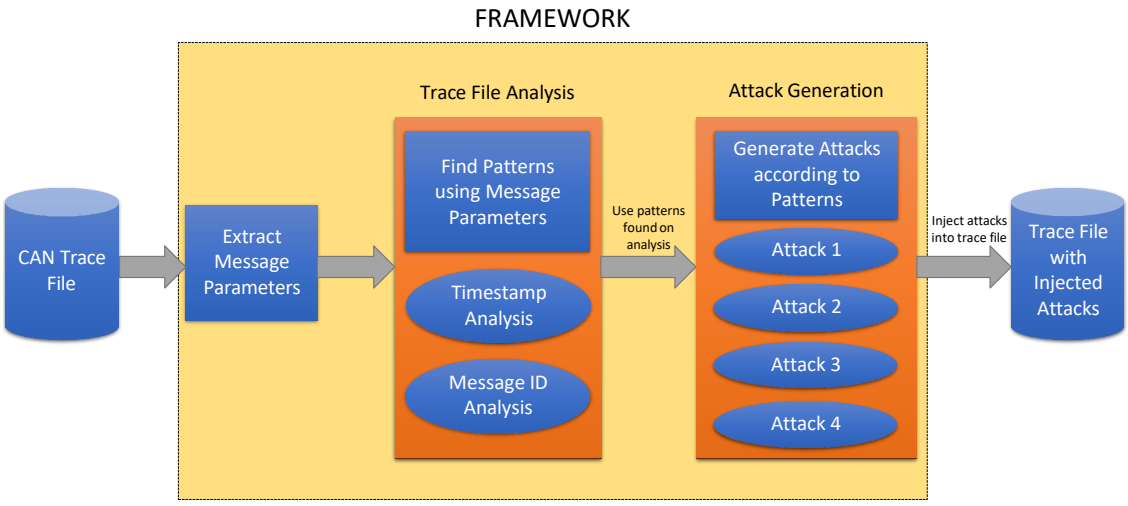

Figure 4.1: System overview flowchart

The implementation of the framework revolves around ASCII trace files for the reasons discussed in 3.4.3 on page 22. Additionally, the attacks should be injected in a clever manner by analyzing trace files for noticeable patterns and sequences of messages. This analysis helps reduce the need for user input to generate and insert attacks into trace files. The main intention of this analysis is to improve attack profiles by introducing randomness and observing existing patterns to model an attacker more realistically. The framework contains a set of defined attack profiles that are applied to a selection of trace files. The framework must be capable of generalizing this method to handle any trace file.

The capabilities of this framework can be divided into two distinct components that will be discussed in detail in the sections to follow. These components are trace file analysis and attack generation. Both of these components yield results that will be evaluated along with the performance of the framework based on a few defined metrics.

4.2 Trace File Analysis

This section provides a detailed overview of our analysis methodology and its implementation in the framework. The analysis is based on 2 parameters of the CAN message, the message IDs and timestamps. By examining these parameters, patterns, and characteristics were identified that could be exploited to attack the system in a clever manner. Identifying patterns in the frequency of occurrence of messages, repeating sequences of messages, and the interval in which these messages arrive can reveal a comprehensive understanding of the trace file's dynamics.

4.2.1 Extracting Message Parameters

To be able to analyze the traffic in an effective manner, the different parameters of CAN messages must be extracted. To automate the extraction and analysis process, we developed specific functions within our code. The way this is achieved in the framework is by reading all the lines of the trace file and then splitting each entry into its constituent parameters. The first function called **parse_files()** takes the path of the file as input and reads the entire trace file. Then the file is read line by line and a second function called **split_entry()** takes each line and splits it with white space as the delimiter. The entry after being split is stored as a list from which a third function called **get_parameters()** extracts each parameter and adds it to its corresponding list. In the end, we get a list for each parameter for the entire trace file.

CAN messages have a unique format that allows the framework to differentiate between standard CAN messages and comments and variants of CAN like CANFD. Since the timestamps are the only unique parameter in the trace file, a dictionary is created to keep track of each entry with the timestamp as the key. Dictionaries do not allow duplicates to exist which is a useful property. This makes the injection of attacks and the manipulation of messages easier as dictionaries can be sorted

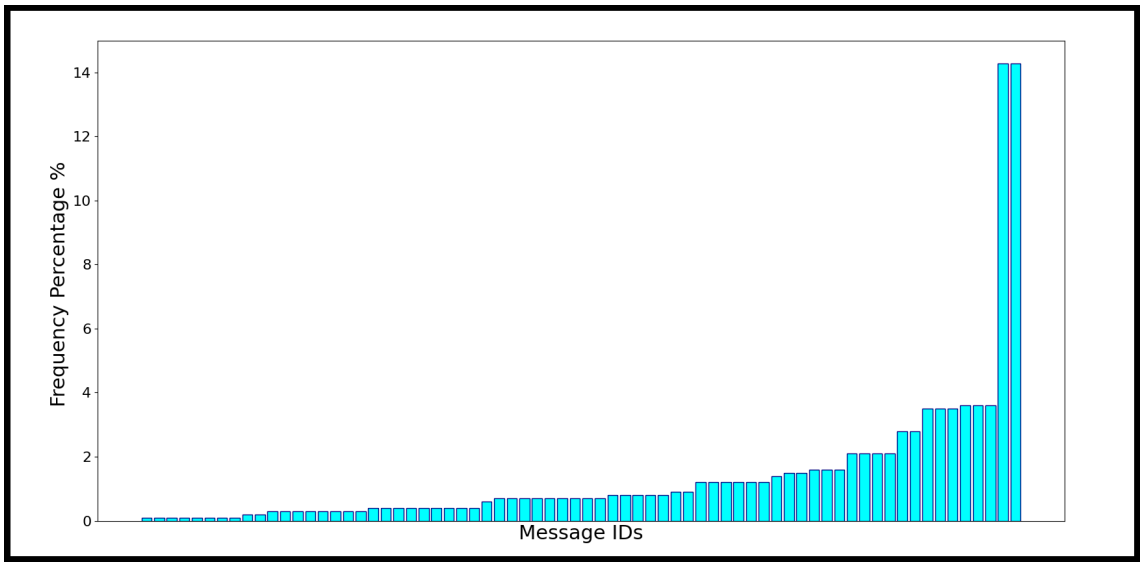

Figure 4.2: Message ID frequency of a trace file

easily and each message can be accessed using just the key. Having each parameter available as a list makes their analysis easier for the framework.

4.2.2 Message ID Analysis

Every message in the trace file is closely related to its message ID. The objective of analyzing this parameter is to find which message IDs appear the most and if there are groups of messages that appear together repeatedly throughout the trace file. The existence of such messages shows their importance in the system and targeting such messages might make attacks more effective and devastating.

4.2.2.1 Frequency of Occurrence

By analyzing the extracted list of message IDs, their frequency of occurrence can be easily determined using the **get_id_stats()** function. This function takes the list of IDs and simply counts the number of times each ID occurs in the list. By doing so for all message IDs, a dictionary can be created which can be then used to easily access the most frequently occurring messages.

The figure 4.2 is a graphical representation of the dictionary obtained by using the function. If malicious messages imitate the messages that appear frequently, the probability of these malicious messages being accepted as valid CAN messages may increase.

4.2.2.2 Repeating Sequences

In addition to frequency analysis, we searched for repeating sequences of message IDs within the trace file. Identifying message sequences that frequently occur together can reveal important relationships and dependencies between different messages.

These sequences may indicate specific system operations or processes that can be targeted or exploited by an attacker. The **establish_correlation()** function takes the list of IDs and returns a dictionary containing all sequences of messages that occur together along with the number of times they are observed to be together. The sequences that occur together frequently imply the dependency of the messages in that sequence on each other.

4.2.3 Timestamps Analysis

One of the unique parameters of messages is their timestamp which gives information about when the message arrived on the CAN bus. A lot can be said about a message just by observing the time at which different instances of that message occur. In order to be able to observe and extract meaningful information, the timestamp of the occurrence of each individual message must be obtained. The function **get_msgs_by_id()** takes a message ID, the timestamp list, and the ID list of the entire trace file and returns a list of timestamps of that particular message. This process is done for each distinct ID in the trace file and the list returned by the function is directly used for analysis.

Cyclic and Partially Cyclic Messages: To understand the purpose of timestamp analysis, the concept of cyclic messages must be known. If the time difference between any two instances of messages that have the same ID is exactly the same, then that message ID is said to occur in a cyclic manner in the system. This difference in time is referred to as the cyclic difference for the purpose of this thesis. Since not all messages can always be cyclic, the concept of partially cyclic messages arises. When messages with the same Message ID appear in a cyclic manner most of the time, i.e, their cyclic difference is the same for the majority of their instances, then we consider such messages to be partially cyclic. In the case of the framework, for the message ID to be partially cyclic it must exhibit cyclic behavior at least 50% of the time. If a message ID is not cyclic or partially cyclic, it is considered to be acyclic.

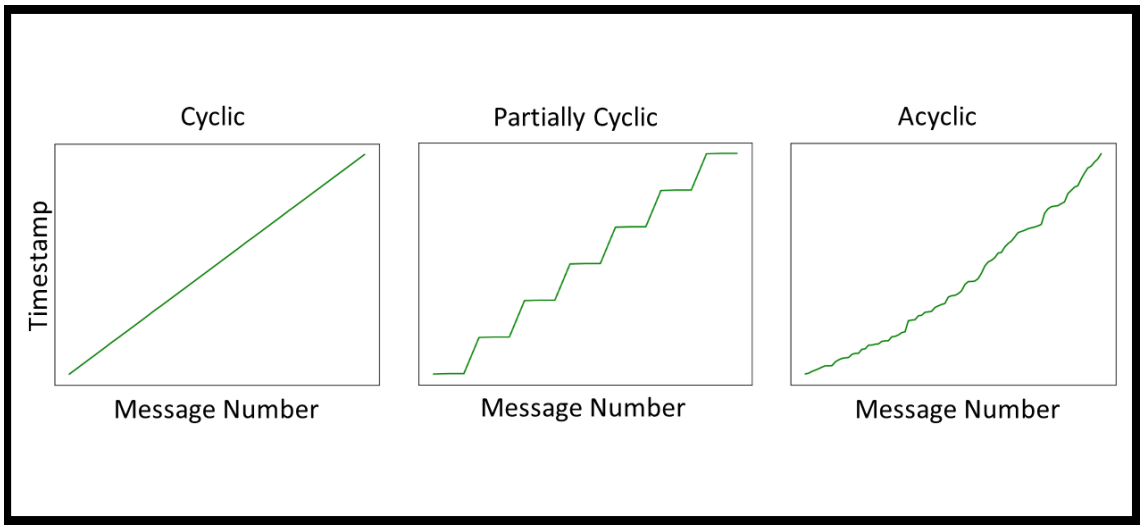

Figure 4.3: Graphical representation of different message IDs based on their cyclic difference

The function **analyse_timestamp()** takes the timestamp list of specific message IDs and returns a list of time differences between 2 consecutive instances of messages with that message ID. All occurrences of the unique time differences (delta) are counted and converted into a dictionary. This process makes it easier to classify message IDs as cyclic, partially cyclic, or acyclic (see figure 4.3). Furthermore, the period in which a message ID exhibits cyclic behavior can be obtained.

The regularity in the timing of cyclic messages allows for precise prediction of when these messages will occur. The variability in the occurrence of partially cyclic messages adds some level of unpredictability, making their timing less consistent compared to fully cyclic ones. This information proved useful for exploiting these patterns by targeting specific time windows or disrupting the regularity of cyclic messages.

4.3 Attack Generation

Once the analysis was in place, it was time to use the insights gained from it to perform injections of attacks. Finalizing a smaller set of attacks from the list of studied attacks presented in table 3.1 was the first step. For the scope of this thesis, 4 attacks were selected and implemented. These attacks are fuzzy, spoofing, replay and overwrite attacks. The overwrite and spoofing attacks were selected due to their ease of understanding and implementation. The fuzzy attack seems simple but if successful can reveal a lot about any system. This attack is usually the first attack performed by adversaries in an attempt to find an entry point into the system and thus is an important attack to cover. The replay attack was selected due to its effectiveness against encrypted systems. It appears easy to implement but can have a devastating effect on a system. Furthermore, these attacks are commonly encountered in real-world scenarios and are well-documented in related

research literature. This section discusses the details of the implementation of each attack and the functions that aided in the process of implementation. In the end, the average attack-to benign traffic ratio is presented for the output trace files generated for every attack.

4.3.1 Fuzzy Attack

The fuzzy attack makes use of randomized IDs and message payloads, so the first step in implementing this attack was to generate randomized IDs. However, the IDs generated in this framework are not completely random. An algorithm to mimic an attacker's approach of manually randomizing IDs was created. The function **gen_random_ids()** takes a list of IDs and performs a set of operations on each ID present in that list to create a unique set of IDs for fuzzing. The input list given to this function is created by taking the top 33% of the most frequently occurring messages obtained through message ID analysis. Every ID in the input list given to the function generates a series of at least 6 IDs. After removing all the duplicates a big list of pseudo-random IDs is obtained. This algorithm generates both standard and extended CAN message IDs in both decimal and hexadecimal formats.

Then, the bus and data length code (DLC) values are generated in a random manner. Using the randomised DLC as input, another function called **get_random_data()** generates a randomised payload of the same size in bytes as the DLC value. This randomised payload is modified into a specific format. The function **get_duration()** returns the first and last timestamps of the trace file. These values are used to generate a random decimal timestamp between the first and last timestamp of the trace file. After obtaining all the parameters required to form a malicious message, a malicious entry is created and inserted into the dictionary with the generated random timestamp as key. All of these tasks come together in a single function called **fuzzy_attack()**.

```
123  0.048435 3  C000F27x        Rx   d 8 FC FF FF FE FF FF FF 13  Length = 552000 BitCount = 142 ID = 201330471x
124  0.049011 3  C000127x        Rx   d 8 DF F8 FF 82 FF FF FF 04  Length = 552000 BitCount = 142 ID = 201326887x
125  0.049108 4  419362301       Rx   d 8 CF F7 85 1B C9 6E 10 B2         //This is a Fuzzy attack
126  0.049587 3  CFF0027x        Rx   d 8 00 4B C0 12 6C FF FF FF  Length = 552000 BitCount = 142 ID = 218038311x
127  0.051444 2  C000021x        Rx   d 8 23 FF FF FA FF FF FF F7  Length = 560000 BitCount = 144 ID = 201326625x
128  0.051661 3  201400687x      Rx   d 8 84 7C CC 5C 70 34 83 CF         //This is a Fuzzy attack
129  0.051872 3  CF00400x        Rx   d 8 10 FE 87 9E 12 27 FF 7D  Length = 540000 BitCount = 139 ID = 217056256x
130  0.052032 2  18FF07E7x       Rx   d 8 00 FF 00 FF 11 85 00 FF  Length = 560000 BitCount = 144 ID = 419366887x
```

Figure 4.4: Injected fuzzy attack traffic into normal CAN traffic

Figure 4.4 is a snippet from the output file generated after injecting fuzzy attack traffic into the trace file.

4.3.2 Replay Attack

To perform a replay attack, the target messages are re-transmitted at a later interval in the system while all the other parameters of the target message remain unchanged. The first step towards the implementation of this attack is to prepare a list of target messages. To do so, the methods discussed for analysis of the trace file prove to be

very handy. First, the top 3 most frequently occurring message IDs are added to the list. Assume this list is called "MsgList". Then the longest sequence of messages that appear together is obtained and all the message IDs present in the sequence are added to "MsgList". Now, for all the unique message IDs in "MsgList", the cyclic time difference is calculated.

For the messages in "MsgList" all other message parameters are obtained. Malicious messages are created using these parameters but with a new timestamp, which is the sum of cyclic difference of the corresponding message and the original timestamp of the target message. Additionally, the last occurrence of each message ID present in "MsgList" is obtained and is replayed with two different timestamps. The first timestamp is the sum of the original message timestamp and its cyclic difference and the second timestamp is the sum of the original message timestamp and a random decimal number between the first and last timestamp of the trace file. This way the messages are replayed in a cyclic as well as a randomised manner. The **replay_attack()** function does all of the tasks mentioned above to successfully inject replay traffic into the input trace file.

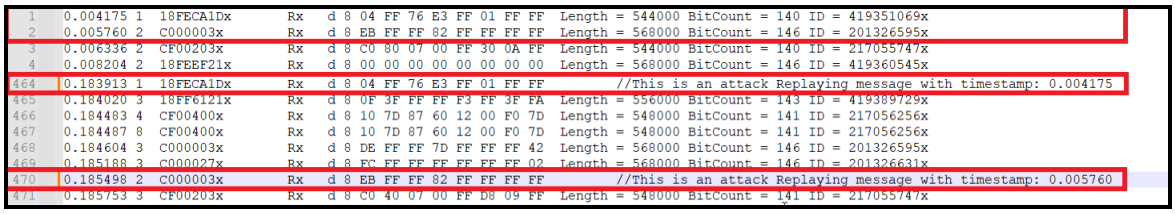

Figure 4.5: Injected replay attack traffic into normal CAN traffic

Figure 4.5 shows how messages at line 1 and 2 are being replayed at lines 464 and 470 respectively. The parameters of the replayed messages are identical to their target messages.

4.3.3 Overwrite Attack

The idea behind the overwrite attack is to inject a malicious message right before a target message. This malicious message should have identical parameters as compared to the target but with an altered payload. The implementation of this attack is similar to the Replay attack. Instead of considering only the more frequently occurring message IDs, all unique message IDs are accounted for and their cyclic time difference is calculated. Using the **get_same_period_diff**, the timestamp of the message IDs is obtained for the period in which they exhibit cyclic behavior. These timestamps are converted into a list.

In order to be able to inject a malicious message right before the target messages arrive on the CAN bus, subtraction of the least significant digit of the timestamp by 1 is performed. All the other parameters of the target message are retrieved and a randomised DLC is used to generate a new random hexadecimal payload. Now, that all the required parameters have been obtained, the malicious messages are

created and inserted into the dictionary with the new timestamp (least significant digit subtracted by 1) as key. The function that performs all of the above mentioned tasks is called **overwrite_attack()**.

```
160   0.064685 2   18EB0172x        Rx    d 8 00 00 00 00 00 00 00 FF   Length = 560000 BitCount = 144 ID = 418054514x
161   0.0646949    8    CF00400x  Rx d   8    AB F8 E4 AA 1F 0A 90 94          //This is an overwrite attack
162   0.064695 8   CF00400x         Rx    d 8 10 7D 87 A0 12 00 F0 7D   Length = 552000 BitCount = 142 ID = 217056256x
163   0.064742 3   C000003x         Rx    d 8 DE FF FF 7D FF FF FF 16   Length = 560000 BitCount = 144 ID = 201326595x
164   0.065249 2   CF00300x         Rx    d 8 E1 00 1A FF FF FF 6C FF   Length = 548000 BitCount = 141 ID = 217056000x
165   0.065322 3   C000027x         Rx    d 8 FC FF FF FF FF FF FF 46   Length = 564000 BitCount = 145 ID = 201326631x
166   0.065827 4   CF00300x         Rx    d 8 E1 00 1A FF FF FF 6C FF   Length = 548000 BitCount = 141 ID = 217056000x
167   0.0658309    8    CF00300x  Rx d   8    B1 77 FA B7 47 7D 4C 48          //This is an overwrite attack
168   0.065831 8   CF00300x         Rx    d 8 E1 00 1A FF FF FF 6C FF   Length = 548000 BitCount = 141 ID = 217056000x
169   0.065833 2   C000003x         Rx    d 8 EB FF FF 82 FF FF FF FF   Length = 568000 BitCount = 146 ID = 201326595x
```

Figure 4.6: Injected overwrite attack traffic into normal CAN traffic

In figure 4.6, it can be observed that the attack messages and the target messages have similar parameters like message ID but have completely different payloads. The timestamps of the attack messages appear right before the target messages.

4.3.4 Spoofing Attack

The spoofing attack is different from the other attacks implemented in the framework. The biggest difference is that spoofing attack simply manipulates messages to change their payload parameter. Where the other attacks create attack traffic and inject it into the trace file, spoofing does not perform any injection of attacks. The procedure of cleverly selecting messages for spoofing is similar to how the overwrite attack is implemented. Instead of considering all of the messages, the top 15 most frequent messages are considered and their timestamps are obtained for the period in which they exhibit cyclic behavior.

All of the parameters corresponding to the targeted messages are retrieved and a random payload is generated using a random DLC value. Since dictionaries do not allow duplicate key values to appear, using the same timestamp as the target message with a manipulated message payload replaces the target message with the malicious one in place. The **spoofing_attack()** function performs all of the above-mentioned tasks.

```
16     0.014312 2   CF00400x         Rx    d 8 10 7D 87 A8 12 00 F0 7D   Length = 548000 BitCount = 141 ID = 217056256x
17     0.014337 8   18FDA517x        Rx    d 8 41 10 04 00 00 00 00 00   Length = 556000 BitCount = 143 ID = 419276055x
18     0.014337 4   18FDA517x        Rx    d 8 41 10 04 00 00 00 00 00   Length = 556000 BitCount = 143 ID = 419276055x
19     0.014665 3   C000003x         Rx    d 8 DE FF FF 7D FF FF FF 31   Length = 560000 BitCount = 144 ID = 201326595x
20     0.014884 2   CF00300x         Rx    d 8 E1 00 1A FF FF FF 6C FF   Length = 548000 BitCount = 141 ID = 217056000x
21     0.014901 8   CF00400x         Rx    d 8 10 7D 87 A8 12 00 F0 7D   Length = 548000 BitCount = 141 ID = 217056256x
22     0.014901 4   CF00400x         Rx    d 8 10 7D 87 A8 12 00 F0 7D   Length = 548000 BitCount = 141 ID = 217056256x
16   0.014312 2   CF00400x         Rx    d 8 10 7D 87 A8 12 00 F0 7D   Length = 548000 BitCount = 141 ID = 217056256x
17   0.014337 8   18FDA517x         Rx    d 8 B8 13 85 5D 81 55 C8 36          //This is a spoofing attack
18   0.014665 3   C000003x         Rx    d 8 DE FF FF 7D FF FF FF 31   Length = 560000 BitCount = 144 ID = 201326595x
19   0.014884 2   CF00300x         Rx    d 8 E1 00 1A FF FF FF 6C FF   Length = 548000 BitCount = 141 ID = 217056000x
20   0.014901 4   CF00400x         Rx    d 8 10 7D 87 A8 12 00 F0 7D   Length = 548000 BitCount = 141 ID = 217056256x
21   0.015245 3   C000027x         Rx    d 8 FC FF FF FF FF FF FF 71   Length = 564000 BitCount = 145 ID = 201326631x
22   0.015469 8   CF00300x         Rx    d 8 04 1C 7C D3 41 D1 9D 6B          //This is a spoofing attack
23   0.015805 3   CF00203x         Rx    d 8 C0 80 07 00 FF 30 0A FF   Length = 544000 BitCount = 140 ID = 217055747x
```

Figure 4.7: Spoofed messages compared to original attack-free trace file

Figure 4.7 compares the original attack-free trace file and the output generated after

introducing spoofing attacks to the same file. Lines 17 and 22 (highlighted in the image) show the exact same messages but with varying payloads.

4.4 Evaluation Framework

In this thesis, there are three distinct types of results that need to be evaluated in order to gain meaningful insights from them. These results are output trace files with attack traffic, the performance of the framework, and a comparison of traffic analysis of trace files between different real-world and synthetic trace files.

4.4.1 Comparison of Trace Files: Real world vs Synthetic traffic

Analysis of the trace files can reveal a lot of information about the behavior of the traffic. The idea behind comparing the insights gained from drawing a contrast between different trace files is to be able to distinguish between the trends followed in different systems. For this, a set of trace files containing **real-world (obtained from real-life vehicles) and synthetic (man-made) trace files** are used. The analysis results are represented graphically so it becomes easier to draw insights for discussion.

The trace files were subjected to multiple experiments that analyzed the differences in parameters and patterns that can be observed. These experiments make use of the analysis functionalities implemented in the framework. The same parameters that were used for clever attack traffic generation namely, Message ID and timestamps of messages are analyzed to find the differences in patterns.

Analyses like the frequency of message IDs, the standard deviation of timestamps, and the ratio of cyclic to acyclic messages helped gain more information about these trace files that highlighted noticeable differences between them. It was determined that for the trace files considered, the message ID and the length of the data had the highest correlation. Using the spearman correlation coefficient, the magnitude of the correlation between the message ID and data length was determined.

4.4.2 Output Trace File: Randomness

To evaluate the output trace file obtained after execution of the framework, the metric considered is called **Randomness**. Randomness refers to the ability of the framework to generate varying outputs. This is a desirable effect for attack traffic generators to test various security mechanisms. It serves two important perspectives. In actual cyber-attacks, the attacker's behavior is often unpredictable and can vary significantly. By incorporating randomness into attack traffic generation, the simulation of a more realistic and diverse set of attack patterns becomes possible, making security testing more robust. Secondly, randomness expands the coverage of security testing by exploring different combinations within attacks. Relying solely on

predetermined attack patterns may overlook some vulnerabilities or attack vectors that are not explicitly defined.

In the framework, a certain level of randomness is introduced while inserting attack traffic into the trace file. This randomness appears in the form of variable number of malicious messages introduced and the way they are scattered throughout the trace file. The framework is designed in a way that lets the number of attacks generated be increased or decreased as required, yet, there still exists variations in the number of malicious messages introduced for every instance of execution.

4.4.3 Framework: Execution time, Complexity

There are several ways in which a framework can be evaluated. For the scope of this thesis, the **execution time** for creating and inserting each attack is compared for trace files having different lengths. The execution time is the time from when the trace file is read till the time the output trace file with attack patterns is generated.

Additionally, the scalability of the framework must be measured to see if it works well for larger trace files as well as it does for smaller ones. For doing so, a metric called the "scaling factor" (this name is used for the purpose of this thesis) is used which is calculated as:

$$Scaling\ Factor = \frac{Execution\ Time\ for\ insertion\ of\ the\ attack}{Number\ of\ messages\ in\ Trace\ File}$$

Using this scaling factor, the expected execution time for a trace file with X number of messages can be obtained by simply multiplying the scaling factor by X. The expected and observed execution times are compared and plotted against each other. After obtaining the plots, an equation for the observed execution time of each attack can be obtained which indicates the **Time Complexity** of implementation of each attack.

5

Experiments and Results

In this section, a discussion of the experiments performed, results obtained, and their evaluation is presented. The following sections give all the necessary information about the requirements and procedure of these experiments, and display the results obtained. After these results are evaluated, the inferences drawn from them are discussed.

5.1 Comparison of Trace Files

For this section, a set of trace files are collected and analyzed based on different parameters. This thesis was done under the supervision of Robert Bosch AB, Lund and some of the trace files used in the following experiments were provided by them.In this set of trace files, there are 2 real-world trace files called "real1.asc" and "real2.asc" and 2 synthetic trace files called "synthetic1.asc" and "synthetic2.asc". The real-world trace file "real1.asc" was obtained by formatting a trace file available online to make it readable for our framework and "real2.asc" was obtained by considering a small portion of a proprietary trace file[1]. The synthetic trace file "synthetic1.asc" was obtained by formatting a proprietary trace file used for testing and "synthetic2.asc" was obtained by taking a portion of a synthetic trace file available online and formatting it (the link to the dataset can be found here [45]). The aim behind comparing the analyses of these trace files is to draw some inferences regarding the differences between real-life and synthetically created traffic. The size of the trace files varies, as shown in table 5.1.

Table 5.1: Size of trace files used in experiments

No.	**Tracefile name**	**No. of Messages**	**Source**
1	real1.asc	1004	Open
2	real2.asc	1002	Proprietary
3	synthetic1.asc	805	Proprietary
4	synthetic2.asc	815	Open

[1]Proprietary trace file obtained from Bosch.

5.1.1 TF1: Ratio of Cyclic Messages

In this experiment, the set of 4 trace files mentioned in table 5.1 are analyzed. The first analysis is performed by obtaining the ratio of message IDs that appear in a cyclic manner (both cyclic and partially cyclic messages are considered here) to those message IDs that are acyclic. By doing so for all the trace files a clear distinction between real-life and synthetic trace files was observed by looking at the values in table 5.2.

Table 5.2: Cyclic to acyclic messages ratio

Trace Filename	**Cyclic messages %**	**Acyclic messages %**
real1.asc	14.6	85.4
real2.asc	16.1	83.9
synthetic1.asc	28.1	71.9
synthetic2.asc	29.4	70.6

For the files considered in this experiment, real-life traffic is observed to have a lesser number of messages that appear in a cyclic manner by a significant amount as compared to the synthetic traffic.

5.1.2 TF2: Message Frequency

This experiment aims to draw a contrast between real-life and synthetic trace files by making use of the analysis implemented in the framework for clever attack traffic generation. The message ID and timestamp analysis of these trace files reveals information that distinguishes the two categories of trace files.

For each real-world trace file the percentage of occurrences of each message ID is calculated and plotted.

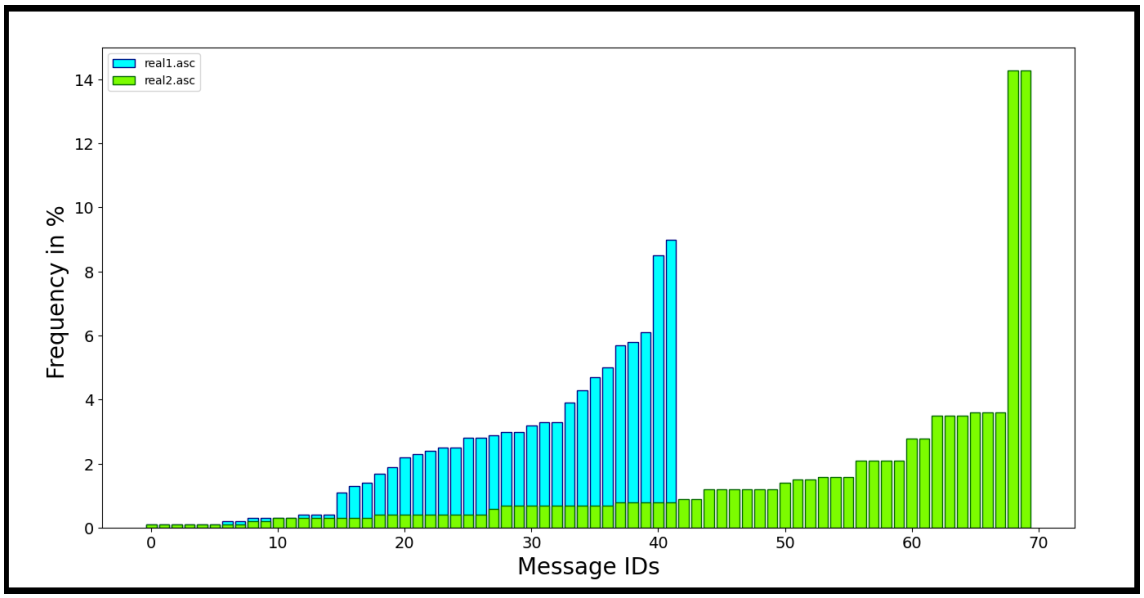

Figure 5.1: Message ID frequency of real-world trace files

The same process is repeated for synthetic trace files and the differences were observed. The discussion of the observations are made later in this report.

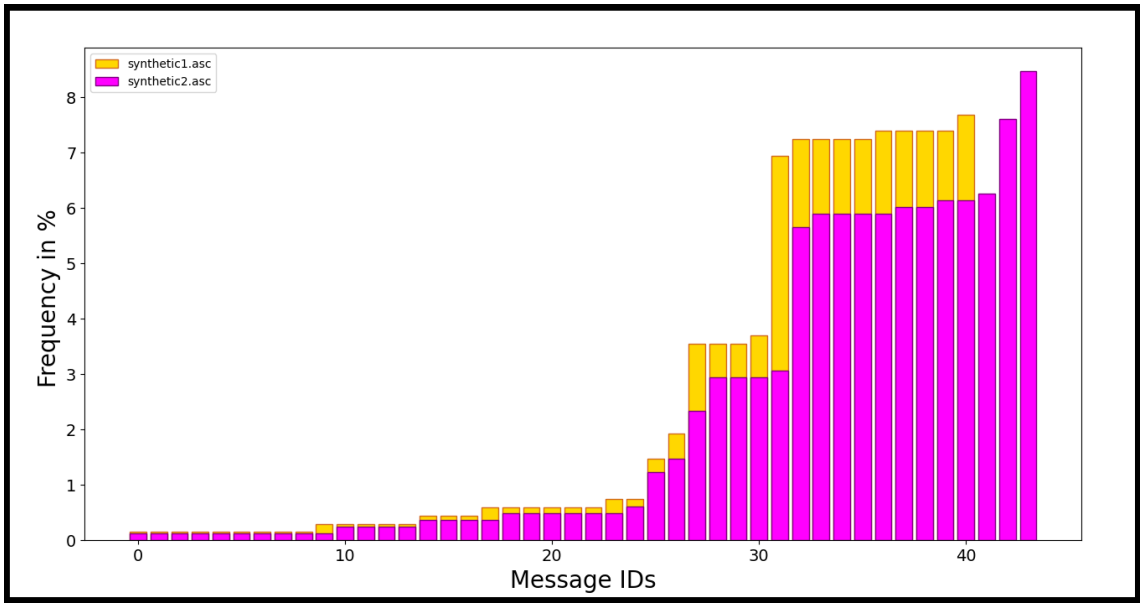

Figure 5.2: Message ID frequency of synthetically created trace files

In the case of real-world trace files, very few messages have a very high frequency of occurrence as compared to the rest. However, in the case of synthetic trace files, a significantly higher number of messages have a higher frequency of occurrence.

5.1.3 TF3: Standard Deviation

In this experiment, the standard deviation of the time interval between the occurrences of messages with the same ID was calculated for each message ID and plotted together in a graph. This process was repeated for both the real-world and synthetic trace files. The observed differences and their implications are discussed later in this report.

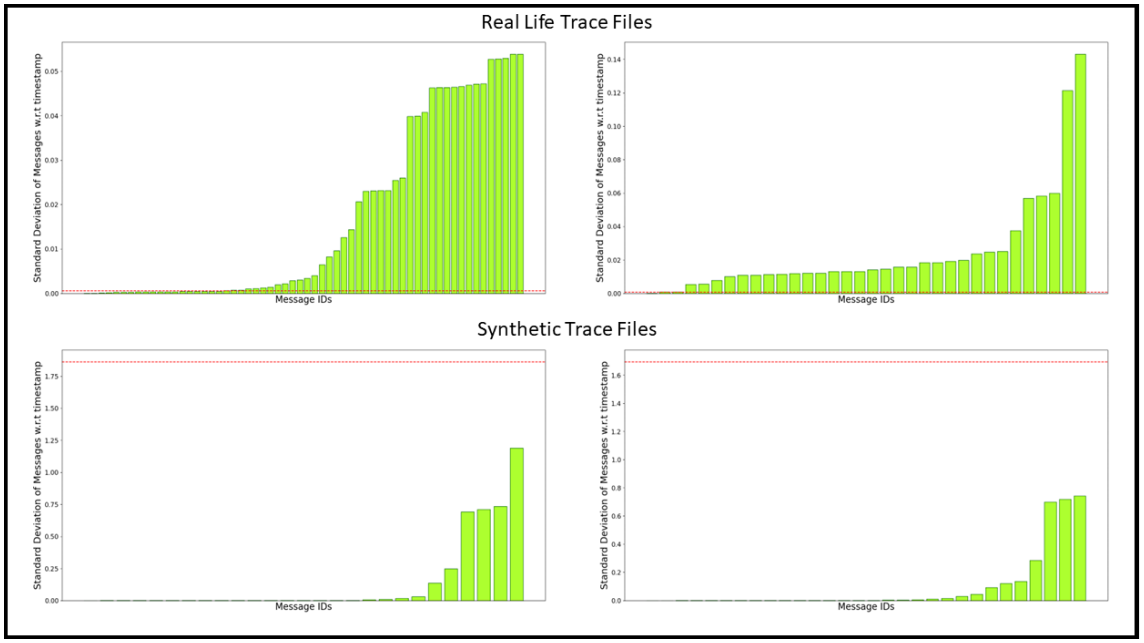

Figure 5.3: Standard deviation of time intervals of different messages

The dotted red line represents the standard deviation of timestamps in the entire trace file. The points on the graph represent the standard deviation of every message ID. The messages in real-world trace files are observed to have a lower standard deviation as compared to synthetic trace files. However, the number of messages that show higher levels of standard deviation when compared to the entire trace file is very high. On the contrary, synthetic trace files show a high standard deviation but the number of messages that exhibit this behavior is low.

5.1.4 TF4: Parameter Correlation

In this experiment, the change in magnitude of the correlation between the message ID and data length for real-world and synthetic trace files was calculated. The differences observed in table 5.3 reveal implications that are discussed later in this report.

Table 5.3: Correlation between IDs and data lengths

Trace Filename	**Correlation**
real.asc	−0.0607
real1.asc	0.0802
synthetic.asc	−0.5162
synthetic1.asc	−0.4693

Through this experiment, it is observed that the synthetic trace files have a significantly higher degree of correlation between message ID and data length as compared to real-world trace files.

5.2 Output Trace File

To check if the framework introduces a degree of randomness, we perform three experiments. The first two experiments will be performed on a set of five output trace files generated by applying the same attack profile on a single input trace file five times. A set of five output trace files is obtained for every attack implemented in the framework. The input trace file was created by taking the first 1002 lines of a proprietary real2.asc trace file. The third experiment will be performed on a single trace file and the outcome will be observed.

5.2.1 R1: Attack to benign ratio

The first experiment simply takes the list of five output trace files generated by the framework and counts the number of malicious messages in each file. This process was repeated for all attack patterns and the results obtained were represented graphically. The figure 5.4 below show the comparison of the number of malicious messages in different output files for every attack.

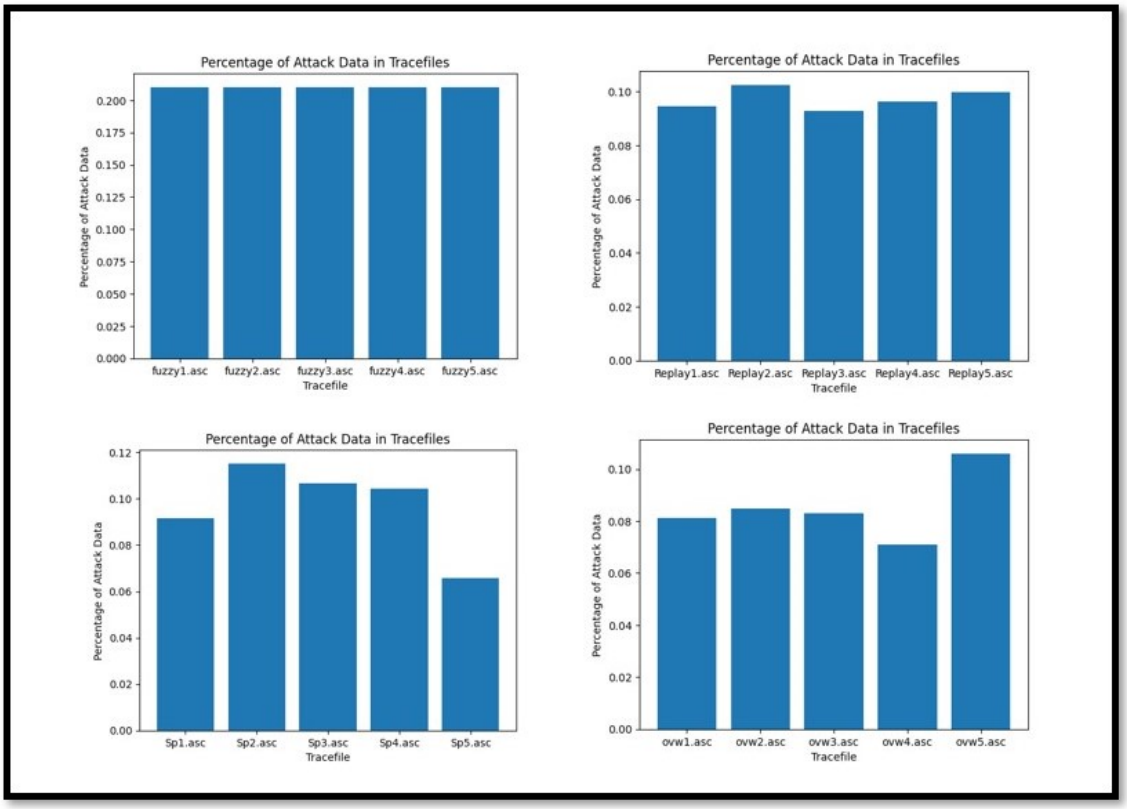

Figure 5.4: Attack plots

Through this experiment, it was revealed that the number of malicious messages generated in different instances of execution varies for all attacks except for the attack. In order to investigate that randomness exists not just in the number of malicious messages generated but also in how they are injected throughout the trace file, we followed up with experiment R2, described next.

5.2.2 R2: Spread of attacks within trace file

The second experiment aims to check how randomly the attacks are distributed throughout the trace file. To do so, the trace file is divided into five sections, with each section containing 20% of the total messages. The five sections are 0-20%, 20-40%, 40-60%, 60-80%, and 80-100%. Then the percentage of attacks falling within each section is calculated. The same process is repeated until all five instances of output trace files generated for the same attack are covered. The results for all the instances are presented on the same plot. This process is repeated until all attack profiles are covered.

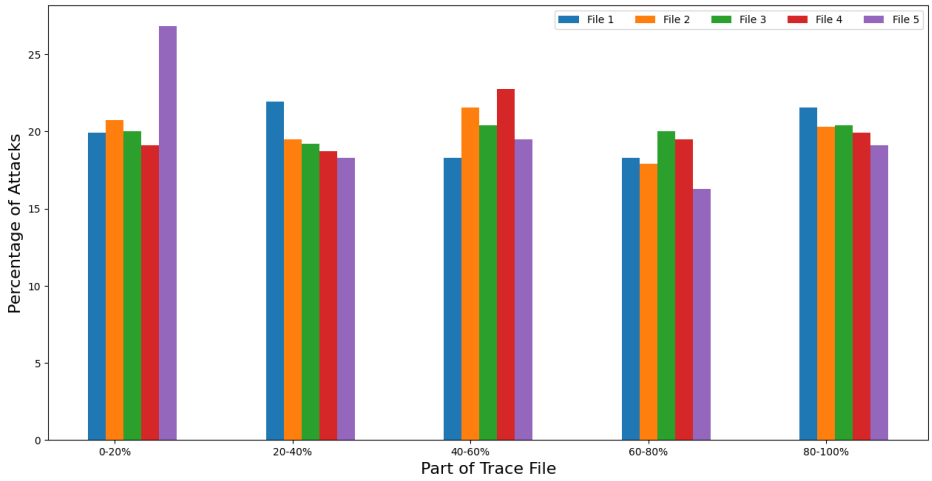

Figure 5.5: Fuzzy attack randomness

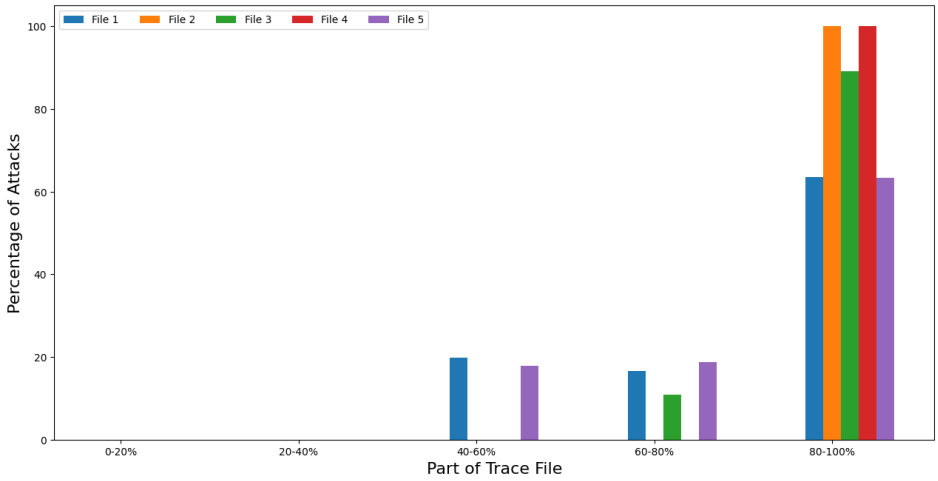

Figure 5.6: Replay attack randomness

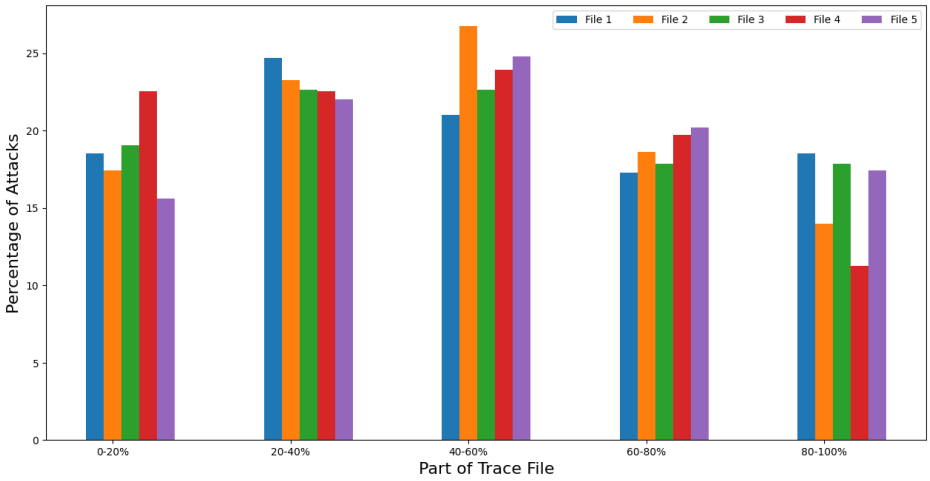

Figure 5.7: Overwrite attack randomness

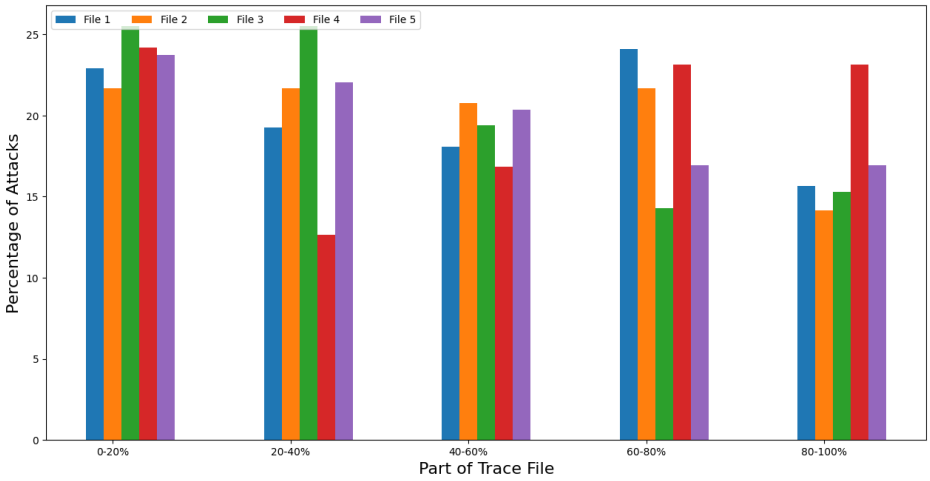

Figure 5.8: Spoofing attack randomness

Through this experiment, it is observed that for every instance of execution, the malicious messages generated are injected unevenly or randomly throughout the output trace files. This is true for all attacks without any exceptions.

5.2.3 R3: Cyclic to acyclic ratio

The third experiment goes a level deeper into evaluating the output trace file. It aims to analyze the randomness that occurs in the parameters, and their behavior and patterns after the attacks are injected. For this, we take a trace file and observe the differences in the frequency of message IDs before and after an attack is generated.

We also analyze the timestamps to see changes in the cyclic behavior of the messages in the attack-free trace files and the infected trace files. The results for all the instances are presented in plots and tables. This process is repeated until all attack profiles are covered.

Table 5.4: Cyclic to acyclic messages ratio of attack-free and infected trace files

Trace File	**Cyclic messages %**	**Acyclic messages %**
Attack Free	16.1	83.9
Fuzzy Attack	14.5	85.5
Replay Attack	16.1	83.9
Overwrite Attack	11.3	88.7
Spoofing Attack	17.7	82.3

The contents of table 5.4 indicate the changes that each attack technique has on the cyclic behavior of messages in the trace file. The Fuzzy Attack shows a slightly lower percentage of cyclic messages at 14.5%, while the Spoofing Attack has a higher percentage at 17.7%. The Replay Attack and the attack-free scenario both have a consistent 16.1% of cyclic messages. On the other hand, the Overwrite Attack exhibits the lowest percentage of cyclic messages at 11.3%. These variations suggest that different attack profiles introduce different levels of randomness and disrupt the regular patterns of message IDs to varying extents.

5.3 Framework Performance

In this section, the performance of the framework is evaluated based on its execution time for each attack. For every attack, four trace files with different numbers of CAN messages are used to measure the execution time for every attack across all the trace files. These four trace files originate from the same trace file[1] which was used to create "real2.asc" as mentioned in section 5.1. The details about the trace files are given in table 5.5

Table 5.5: Size of trace files used in experiments

No.	**Trace Filename**	**No. of Messages**
1	part1k.asc	1002
2	part10k.asc	10278
3	part20k.asc	20074
4	part30k.asc	30336

5.3.1 P1: Execution time

The first experiment calculates the execution time from the time a trace file is taken as input by the framework till the time output is generated by it with the selected

attack traffic. All the values obtained in this experiment are presented in table 5.6. These values are measured in seconds.

Table 5.6: Computation time for each attack

Trace Filename	Fuzzy	Replay	Overwrite	Spoofing
part1k.asc	0.02604	0.06699	0.05093	0.04899
part10k.asc	1.19304	2.34770	1.621134	1.62146
part20k.asc	4.38175	7.34836	5.622835	5.53729
part30k.asc	10.2493	16.9120	11.88421	11.8128

Replay attack takes the longest time to execute in case of all trace files followed by an overwriting attack, spoofing attack, and fuzzy attack respectively. This means that the complexity of implementation of each attack follows the same trend with the replay attack being the most complex followed by overwrite attack, spoofing attack, and fuzzy attack respectively.

5.3.2 P2: Complexity

For this experiment, the scaling factor is calculated for each attack against part1k.asc. All the values in table 5.7 are multiplied by 10^{-5}.

Table 5.7: Scaling factor for each attack

Trace Filename	Fuzzy	Replay	Overwrite	Spoofing
part1k.asc	2.6424	6.6864	5.0830	4.8900

For example, if the expected execution time for "part20k.asc" needs to be calculated in case of a fuzzy attack from "part1k.asc" using the above method, the result should be 0.53043. In reality the execution time for a fuzzy attack run against the "part20k.asc" file given an execution time of 4.38175 which is almost 9 times what was expected. This means that the framework does not do a very good job of scaling up to bigger trace files.

Using the scaling factor the expected time is calculated for each attack against the same trace files used in table 5.6. Using this expected time, a comparison between the expected and observed execution time can be drawn. The values were plotted to obtain the following graphs.

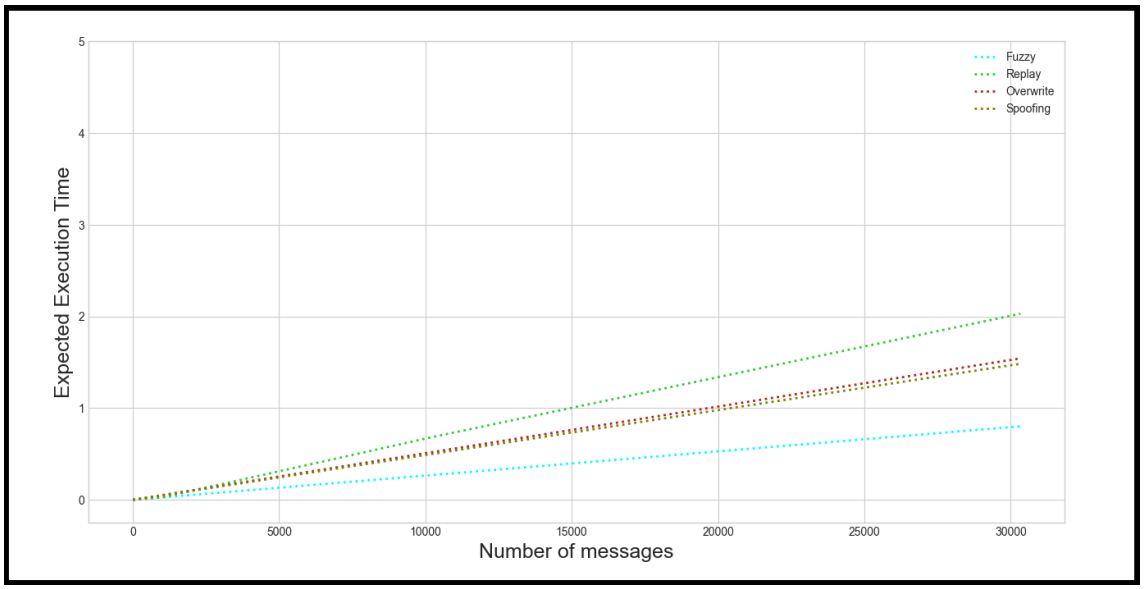

Figure 5.9: Expected execution time of each attack

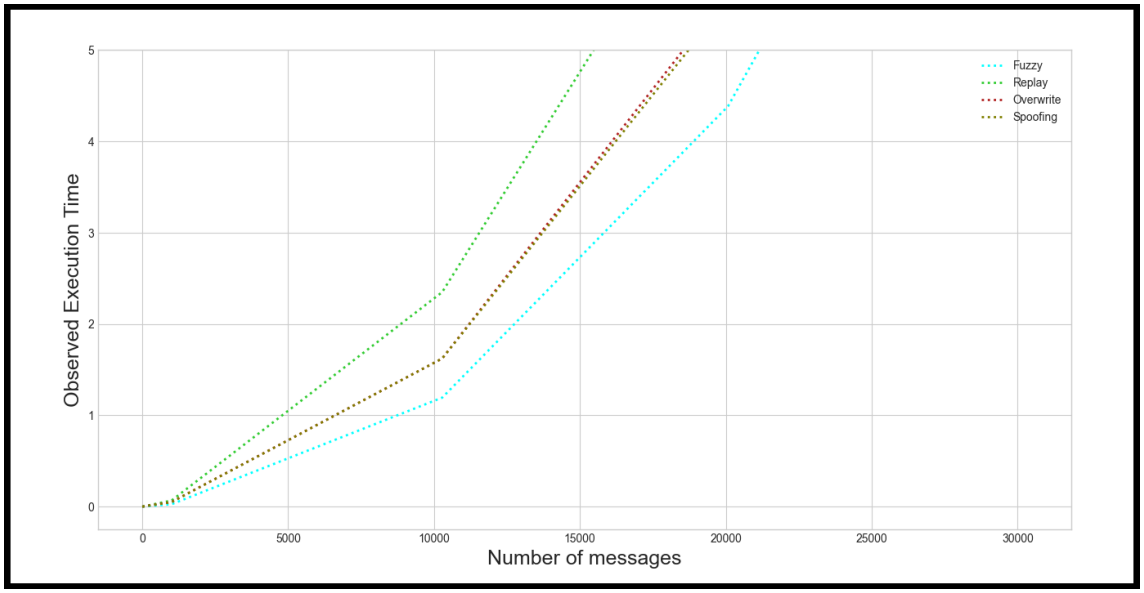

Figure 5.10: Observed execution time of each attack

By observing the graphs, it can be implied that the expected execution time from our model grows linearly as the size of the input trace file increases. However, the same cannot be implied in the case of the observed execution time which appears to grow in a non-linear fashion. The equation for the observed execution time was calculated through the plot as can be seen in table 5.8.

Table 5.8: Equation for each attack

Sr. No.	Attack	Equation
1	Fuzzy	$1.1541\times10^{-8}x^2$ - 0.000013x + 0.0483
2	Replay	1.7659×10^{-8} x^2 + 0.000016x + 0.1065
3	Overwrite	1.1222×10^{-8} x^2 + 0.000053x - 0.0394
4	Spoofing	1.1269×10^{-8} x^2 + 0.000048x - 0.0268

Considering the most dominant term of each equation it can be inferred that each attack implemented in the framework has a worst-case time complexity of $\mathcal{O}(n^2)$

5.4 Discussion

This thesis covers a series of experiments that compare different trace files, check the level of randomness of output trace files generated by the framework, and evaluate the performance of the framework in terms of execution time. Certain inferences can be drawn from these experiments regarding the trace files and the framework's performance. These inferences will be discussed in detail in this section.

5.4.1 Comparison of Trace Files

The first set of experiments focused on comparing real-world and synthetic trace files in an attempt to make clear distinctions between them. The results from the experiments conducted in this section denote a higher percentage of cyclic messages in synthetic trace files. This indicates the lower degree of randomness in the occurrence of messages in synthetic trace files.

On observing the frequency of occurrence of each unique message ID, it was observed that throughout the tracefile, real-life trace files have very few messages that appear frequently whereas a higher number of messages appear frequently for synthetic trace files. This means that a higher number of message IDs need frequent updating in the case of synthetic trace files as compared to real-world trace files.

The standard deviation of the messages based on their time interval indicates that in the case of real-world trace files, the timestamp of the messages deviates from the mean moderately but a higher number of messages exhibit this behavior. In the case of synthetic trace files, the deviation is drastic but fewer messages exhibit this deviation. This is another indication of the lower levels of randomness in the case of synthetic trace files as compared to real-life trace files.

Additionally, on comparing the correlation between the message IDs and data lengths, synthetic trace files appear to have a higher correlation between the two by almost 6 times. This indicates that the size of the message payload is heavily dependent on the message ID in the case of synthetic trace files. This is not the case for real-life trace files where the correlation is much lower.

The high percentage of cyclic messages and the high correlation coefficient indicate that the synthetic files used in the case of these experiments have lower levels of randomness. The messages appear in a more uniform manner and do not deviate from the average interval (standard deviation), which indicates a lower level of randomness. Lower levels of randomness mean that these synthetic files do not cover a wide range of scenarios. Due to this reason, security mechanisms can easily find patterns and anomalies in such files. Testing security mechanisms against files with low randomness levels would not be very helpful in testing and improving those mechanisms. Additionally, using synthetic files to train machine learning models would result in biased decisions, overfitting, and a less generalized solution to the problem. Real-life trace files which have a higher level of randomness should be used for the purposes mentioned above.

5.4.2 Output Trace Files

Randomness has been seen as an attractive quality for CAN traffic throughout this study. Here, the randomness in the insertion of attack traffic is discussed. Through the experiments performed on the output trace files to test the levels of randomness, we observe that for all attack profiles, except fuzzy attack, the number of attacks introduced in each instance of the output trace file is different.

The reason that the fuzzy attack does not exhibit similar behavior is because of its implementation. The number of attacks introduced in the case of the fuzzy profile is directly dependent on the number of pseudo-random IDs generated. The algorithm to generate pseudo-random IDs performs the same operation on a fixed number of unique message IDs from the trace file to obtain a finite set of new pseudo-random IDs. Since all instances of the output files are obtained from the same input file, the set of pseudo-random IDs remains the same and hence, the number of attack messages remains the same.

In another experiment, the impact of introducing attacks on the cyclic properties of messages is observed. In case of a fuzzy attack, there are new message IDs introduced in an acyclic manner which reduces the percentage of cyclic messages in the trace file. In the case of overwrite attacks, the cyclic property of existing messages is disrupted by introducing attack data right before the legitimate data thereby making the targeted messages acyclic. Therefore, the largest drop in cyclic message percentage is observed. Since replay attacks just re-transmit targeted messages both in a cyclic and an acyclic manner which does not disrupt the cyclic property of any message. The percentage of cyclic messages remains unchanged.

The change in the percentage of cyclic messages for the 3 attacks discussed above meets the expectations. A spoofing attack, on the other hand, is implemented similarly to a replay attack but instead of injecting malicious messages, the target messages are manipulated in place. This should not affect the cyclic properties of any message and the percentage of cyclic messages should have remained the same. The increase in the percentage of cyclic messages in the case of a spoofing attack is

completely unexpected.

5.4.3 Framework Performance

Through the experiments performed to evaluate the performance of the framework, the time complexity of each attack was obtained using the execution time. Even though the worst-case time complexity of each attack appears to be the same, there is a clear distinction in which attack is the most complex in the framework. The replay attack exhibits a higher execution time and a higher scaling factor.

The reason for this is the increased amount of analysis performed while implementing this attack. Replay attack makes use of the longest sequences of messages that appear together repeatedly throughout the trace file along with checking the messages which have the highest frequency and the cyclic time difference of those messages. After this analysis, targeted messages are replayed both in a cyclic and acyclic manner. In comparison to this, the other attacks do not require checking the longest sequence of messages that appear together repeatedly in the trace file. Due to this reason, the replay attack is the most complex attack out of the lot.

5.5 Limitations

Several challenges were faced in generating synthetic attack traffic data for CAN-based networks. These included:

- **Limited Real-world traffic:** The availability of trace files that accurately represent the traffic on a CAN-based network is limited, which makes it difficult to generate realistic synthetic/artificial traffic data.

- **Lack of diversity of Scenarios:** The trace files available for analysis may not provide a diverse enough range of scenarios and behaviors to accurately model the full spectrum of possible attack patterns.

- **Analyzing Trace files:** Additionally, analyzing ASCII trace files can be challenging, as the parameters are not in a human-readable format, making it difficult to understand the traffic and identify attack patterns.

5.6 Future Work

In this thesis, we were only able to scratch the surface of trace file analysis and attack traffic generation. However, there are several areas that offer the potential for further exploration and development. In this section, a discussion of four potential directions for future research and expansion of the work presented in this thesis are highlighted.

- **Deeper Analysis of the Trace File:**

  Even though important information was extracted from the trace files that were used for this study, there is still a lot more that can be discovered. Currently, most of the information extracted from these trace files is based on just two distinct parameters. Analyzing more parameters would reveal much more information about them. One such analysis can be performed on the message payload. Since the message payload contains multiple signals for various parameters of the system, it is possible to identify the boundaries of each signal in the payload. By doing so individual signals can be targeted resulting in a more intelligent way of attacking the system while using the setup already available in this thesis.

- **Tester Mode:**

  Currently, the framework operates on an adversary model, which limits our access to ASC files and restricts the amount of information that can be obtained. Using the "tester mode" instead, which makes use of DBC files to obtain a detailed description of every message would reveal a lot more information about the CAN bus system. It would give insights into CAN message payloads, signal definitions with their meaning as well as network configurations. With tester mode, one can simulate more realistic testing scenarios by injecting a wider range of attack patterns that are injected into the CAN bus network in a manner that leverages insider knowledge. This will help explore potential attack vectors that may not be accessible under the adversary model.

- **Wider Coverage of Attacks:**

  While our framework currently includes four existing attacks, there is room for expanding the range of covered attacks. By including a broader variety of attack patterns, the depth of this framework can be enhanced to provide a more realistic representation of the potential attack scenarios faced by CAN bus networks. Incorporating attacks such as message isolation, and denial-of-service attacks would strengthen the scope of attack injections and evaluation metrics.

- **Machine Learning Approach:**

  There are many studies that use machine learning to obtain more information. Traditionally, analyzing CAN bus data involves a manual examination of individual signals, which can be time-consuming and limited in detecting complex patterns or anomalies. However, machine learning can automate and enhance this process. It can be trained on a large chunk of CAN bus data to learn normal behavior and patterns within the system. Using machine learning, a deeper analysis of the trace file can be done to reveal more parameters that can be manipulated to make attacks sneakier and more intelligent.

5.7 Ethical concerns and sustainability

Although this research contributes to understanding system vulnerabilities and developing countermeasures, it is essential to consider and discuss potential ethical concerns and sustainability aspects.

5.7.1 Ethical issues

This thesis develops and deploys various attack patterns and there exists concerns about the unintentional spread of these attack patterns in real-world scenarios. In order to avoid this ethical concern, the research is conducted in a carefully controlled environment. Since this thesis involves deliberate attempts to compromise automotive network security (for testing purposes), the utmost care has been taken to prevent unintended disruption to any system.

Furthermore, any sensitive information in the data used for analysis and testing is handled ensuring compliance with user privacy and data protection regulations. Consequently, ethical concerns in the context of scientific integrity and rigor have also been considered. Related research works have been cited appropriately when needed. The thesis follows a scientific methodology to ensure a systematic approach is adopted.

5.7.2 Sustainability

The sustainability perspective in this research includes designing a framework and methodologies that are adaptive, shareable, and environmentally conscious. This thesis is intended to be open-source so that it can contribute to future research conducted in similar domains. Since the framework developed focuses more on analysis of trace files than injections of attacks, the thesis should remain relevant even when the attack patterns and methodologies for attacking autonomous systems change.Furthermore, the algorithms are designed with efficiency in mind to ensure that minimal resources are utilised for generating attack patterns and analyzing trace files.

These aspects have been addressed in this thesis to allow the research to have a lasting positive impact in the field of automotive security while ensuring sustainability concerns are also met.

6

Conclusion

In this chapter, the research questions are revisited to see whether or not they have been answered in the course of this thesis.

To address the challenge of limited availability of attack traffic data, a software tool was developed that takes a trace file with normal traffic as input and inserts new traffic patterns using attack templates. This tool addresses the need for creating realistic traffic patterns by analyzing normal traffic and generating appropriate attack patterns to mimic traffic behavior from the perspective of an adversary. With a tool like this, researchers and practitioners in the field of automotive security can have access to a wider range of data for testing and evaluation purposes of security mechanisms.

To improve attack profiles and make it more challenging for security mechanisms to detect them, it is important to incorporate randomness and use historical data obtained through analysis while injecting attack patterns. Through the analysis of both synthetic and real-life trace files, clear distinctions between their parameters were observed. This finding highlights the importance of considering real-world traffic patterns when designing and evaluating intrusion detection systems for CAN bus networks. On implementation, the randomness introduced by the framework in generating output trace files is tested and evaluated. The experiments demonstrated that the framework successfully introduced randomness in terms of the distribution of malicious messages through the output trace file.

This research contributes to the development of a mechanism to generate synthetic traffic with attack patterns for CAN. The analysis of the input trace file makes it possible for the framework to do so while introducing randomness. The complexity of the framework is assessed for each implemented attack for researchers and practitioners to optimize the performance. These findings contribute to the advancement of automotive security research and offer guidance for the development and evaluation of security mechanisms in CAN-based networks. By applying the recommendations for the future and further exploring these areas, more realistic synthetic traffic can be generated and researchers can improve the security measures and better protect automotive systems from potential attacks.

Bibliography

[1] Wikipedia, "Automotive security," 2022. [Online]. Available: https://shorturl.at/IJPQX.

[2] T. Huang, J. Zhou, and A. Bytes, "ATG: An attack traffic generation tool for security testing of in-vehicle can bus," 2018. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3230833.3230843.

[3] Bosch GmbH, "CAN specification version 2.0," Tech. Rep., 1991. [Online]. Available: https://www.semanticscholar.org/paper/CAN-Specification-Version-2.0-Bosch/883b50beab09ff6f07a8d5714b2084f4bbfe6dc0.

[4] Y. Shen, D. Huang, and X. Lin, "Security analysis of controller area network (CAN) bus," in *2017 4th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)*, IEEE, 2017, pp. 1–6. DOI: 10.1109/CSCloud.2017.57.

[5] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of the 20th USENIX Conference on Security*, 2011, pp. 6–6. [Online]. Available: https://www.usenix.org/conference/usenixsecurity11/comprehensive-experimental-analyses-automotive-attack-surfaces.

[6] A. Costin, A. Francillon, and D. Balzarotti, "Let's drive it off the lot: A study of carmaker app markets and their security posture," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 150–161. DOI: 10.1145/2382196.2382216.

[7] C. Wressnegger, C. Schlehuber, O. Hohlfeld, and G. Carle, "Evaluating intrusion detection for in-vehicle networks," in *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2016, pp. 336–341. DOI: 10.1109/CCNC.2016.7444807.

[8] M. Ring, K. Krombholz, and M. Huber, "On the feasibility of intrusion detection systems for can," in *International Conference on Information Systems Security and Privacy*, 2015, pp. 255–274. DOI: 10.1007/978-3-319-19069-0_13.

[9] M. Chiang, K. Liao, and S. Yeh, "CAN traffic generation based on machine learning models," in *2019 22nd International Conference on Intelligent Transportation Systems (ITSC)*, 2019, pp. 3906–3911. DOI: 10.1109/ITSC.2019.8917412.

[10] M. Bozdal, M. Samie, S. Aslam, and I. Jennions, "Evaluation of CAN bus security challenges," *Sensors*, vol. 20, no. 8, p. 2364, 2020.

[11] H. J. Jo and W. Choi, "A survey of attacks on controller area networks and corresponding countermeasures," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6123–6141, 2021.

[12] M. Dibaei, X. Zheng, K. Jiang, R. Abbas, S. Liu, Y. Zhang, Y. Xiang, and S. Yu, "Attacks and defenses on intelligent connected vehicles: A survey," *Digital Communications and Networks*, vol. 6, no. 4, pp. 399–421, 2020.

[13] P. Thirumavalavasethurayar and T. Ravi, "Implementation of replay attack in controller area network bus using universal verification methodology," in *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, IEEE, 2021, pp. 1142–1146.

[14] J. Laufenberg, T. Kropf, and O. Bringmann, "Static analysis of controller area network communication for attack detection," *European Journal for Security Research*, vol. 6, no. 2, pp. 171–187, 2021.

[15] T. Lenard and R. Bolboaca, "A stateful firewall and intrusion detection system enforced with secure logging for controller area network," in *European Interdisciplinary Cybersecurity Conference*, 2021, pp. 39–45.

[16] Fürst, Simon and Bechter, Markus, "AUTOSAR for connected and autonomous vehicles: The AUTOSAR adaptive platform," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, 2016, pp. 215–217. DOI: 10.1109/DSN-W.2016.24.

[17] M. E. Verma, M. D. Iannacone, R. A. Bridges, S. C. Hollifield, P. Moriano, B. Kay, and F. L. Combs, "Addressing the lack of comparability & testing in can intrusion detection research: A comprehensive guide to CAN IDS data & introduction of the road dataset," *arXiv preprint arXiv:2012.14600*, 2020.

[18] M. Zago, S. Longari, A. Tricarico, M. Carminati, M. G. Pérez, G. M. Pérez, and S. Zanero, "ReCAN–dataset for reverse engineering of controller area networks," *Data in brief*, vol. 29, p. 105 149, 2020.

[19] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization.," *ICISSp*, vol. 1, pp. 108–116, 2018.

[20] U. Ezeobi, H. Olufowobi, C. Young, J. Zambreno, and G. Bloom, "Reverse engineering controller area network messages using unsupervised machine learning," *IEEE Consumer Electronics Magazine*, vol. 11, no. 1, pp. 50–56, 2022. DOI: 10.1109/MCE.2020.3023538.

[21] M. E. Verma, R. A. Bridges, J. J. Sosnowski, S. C. Hollifield, and M. D. Iannacone, "CAN-D: A modular four-step pipeline for comprehensively decoding controller area network data," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 10, pp. 9685–9700, 2021.

[22] M. Verma, R. Bridges, and S. Hollifield, "ACTT: Automotive can tokenization and translation," in *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, IEEE, 2018, pp. 278–283.

[23] C. Young, J. Svoboda, and J. Zambreno, "Towards reverse engineering controller area network messages using machine learning," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, IEEE, 2020, pp. 1–6.

[24] S. Lestyan, G. Acs, G. Biczók, and Z. Szalay, "Extracting vehicle sensor signals from CAN logs for driver re-identification," *arXiv preprint arXiv:1902.08956*, 2019.

[25] N. Puketza, M. Chung, R. A. Olsson, and B. Mukherjee, "A software platform for testing intrusion detection systems," *IEEE software*, vol. 14, no. 5, pp. 43–51, 1997.

[26] S. Behal and K. Kumar, "Characterization and comparison of DDoS attack tools and traffic generators: A review.," *Int. J. Netw. Secur.*, vol. 19, no. 3, pp. 383–393, 2017.

[27] F. Erlacher and F. Dressler, "How to test an IDS? GENESIDS: An automated system for generating attack traffic," in *Proceedings of the 2018 Workshop on Traffic Measurements for Cybersecurity*, 2018, pp. 46–51.

[28] T. Huang, J. Zhou, and A. Bytes, "ATG: An attack traffic generation tool for security testing of in-vehicle CAN bus," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, 2018, pp. 1–6.

[29] W. Yang, C. Hu, and S. Ma, "Study on penetration testing platform oriented to CAN bus embedded system," in *2022 7th International Conference on Signal and Image Processing (ICSIP)*, IEEE, 2022, pp. 400–404.

[30] S. Hounsinou, M. Stidd, U. Ezeobi, H. Olufowobi, M. Nasri, and G. Bloom, "Vulnerability of controller area network to schedule-based attacks," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 495–507. DOI: 10.1109/RTSS52674.2021.00051.

[31] A. Palanca, E. Evenchick, F. Maggi, and S. Zanero, "A stealth, selective, link-layer denial-of-service attack against automotive networks," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14*, Springer, 2017, pp. 185–206.

[32] A.-I. Radu, "Securing the in-vehicle network," Ph.D. dissertation, University of Birmingham, 2020.

[33] Matvej Mikael Yli-Olli, *Machine learning for secure vehicular communication: An empirical study*, [Online; accessed April 27, 2023], 2019. [Online]. Available: https://www.researchgate.net/publication/334416564_Machine_Learning_for_Secure_Vehicular_Communication_an_Empirical_Study/figures?lo=1.

[34] Borhanuddin Mohd Ali, *Review of researches in controller area networks evolution and applications*, [Online; accessed April 27, 2023], 2010. [Online]. Available: https://www.researchgate.net/publication/228957774_Review_of_Researches_in_Controller_Area_Networks_Evolution_and_Applications/figures?lo=1.

[35] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, and K. Koscher, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of the 20th USENIX conference on Security*, USENIX Association, 2011, pp. 6–6.

[36] S. Rethinam, E. Sundararajan, and M. Ponnavaikko, "Intrusion detection system using message parameter analysis for controller area network," in *2016 International Conference on Emerging Trends in Engineering, Technology and*

Science (ICETETS), IEEE, 2016, pp. 1–5. DOI: 10.1109/ICETETS.2016.7602891.

[37] C. Yang and F. Ren, "Anomaly detection for controller area network traffic based on machine learning," *Sensors*, vol. 19, no. 13, p. 2951, 2019. DOI: 10.3390/s19132951.

[38] S. M. Mahmood and K. McLaughlin, "Intrusion detection in controller area network using machine learning," in *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. DOI: 10.1109/WiMOB.2018.8589143.

[39] F. D. Garcia, G. Vigna, and R. A. Kemmerer, "Anomaly detection of web-based attacks," *Journal of Computer Security*, vol. 17, no. 3, pp. 261–290, 2009.

[40] K. McLaughlin and P. Koopman, "CANsee: An open-source CAN bus monitor for security applications," in *2016 IEEE 18th International Conference on High-Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, IEEE, vol. 1, 2016, pp. 1901–1906. DOI: 10.1109/HPCC-SmartCity-DSS.2016.0269.

[41] C. Young, J. Svoboda, and J. Zambreno, "Towards reverse engineering controller area network messages using machine learning," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, IEEE, 2020, pp. 1–6. DOI: 10.1109/WF-IoT47822.2020.9227604.

[42] P. S. Lekshmi and R. R. Kumar, "An overview of intrusion detection systems: Architectures, techniques, and challenges," *Procedia Technology*, vol. 24, pp. 1079–1086, 2016.

[43] C. Liao, D. S. Wong, and K. K. Leung, "A framework for benchmarking intrusion detection systems," in *Proceedings of an unspecified conference in 2013*.

[44] N. Davas, I. Anagnostopoulos, and S. Hadjiefthymiades, "Securing the Controller Area Network (CAN) communications in automotive applications: Challenges and solutions," *Sensors*, vol. 18, no. 4, p. 1213, 2018. DOI: 10.3390/s18041213.

[45] H. M. Song and H. K. Kim, *Discovering CAN specification using on-board diagnostics*, [Online; accessed April 27, 2023], 2021. [Online]. Available: https://ocslab.hksecurity.net/Datasets/can-signal-extraction-and-translation-dataset.