

Automotive Cybersecurity: Risk-driven Testing Requirements

Student number: 2216686

Student name: Linda Mafunu



Swansea University Prifysgol Abertawe

Department of Computer Science

30 September 2024

Declaration

Statement 1

This work has not been previously accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed Linda Mafunu (2216686)

Date 30 September 2024 (2216686)

Statement 2

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by citations giving explicit references. A reference list is appended.

Signed Linda Mafunu (2216686)

Date 30 September 2024 (2216686)

Statement 3

The University's ethical procedures have been followed and, where appropriate, ethical approval has been granted.

Signed Linda Mafunu (2216686)

Date 30 September 2024 (2216686)

Abstract

Automobile manufacturers are increasingly focussing their efforts on creating and manufacturing electric vehicles (EVs). These advancements are based on software embedded software incorporated in Electronic Control Units (ECUs) which serve as the primary control area for EVs. The Control Area Network (CAN) is an essential serial communication protocol that enables effective real time communication of ECUs in EVs. The main problem in CAN protocol is that it was not developed with cybersecurity in mind; it has a broadcast nature and can be accessed via different interfaces in a vehicle. A variety of attack vectors attacks can be exploited by reading CAN message contents due to its broadcasting nature. To address this issue the study provides a thorough analysis of the vulnerabilities inherent in the CAN bus on a hardware testbed. A variety of penetration testing approaches such as Fuzzing and Penetration testing, we employed to investigated how malicious attackers might exploit the vulnerabilities uncovered. The study's findings highlight security issues such as lack of encryption, poor message filtering and ineffective messages authentication procedures, all of which expose vehicles to potential cyber-attacks. The study's findings of the study indicate that while the system was able to mitigate against some attacks the CAN protocol's limitations allowed for some exploitation, particularly in Replay Attacks which are the most effective method for bypassing security measures and altering ECU behaviour. The study emphasises the critical requirement for stronger cyber security messages. By resolving the weaknesses identified in the study, manufactures can significantly improve the security of vehicle systems.

Keywords: *Automotive, Cybersecurity, Control Area Network, Electronic Control Units, Pen-testing*

Acknowledgements

First, I would like to thank God for blessing me the wisdom and perseverance to complete this project.

I would like to express my deepest gratitude to my supervisor Professor Siraj Shaikh for his support, guidance, and insightful feedback throughout this research project. His expertise in automotive cybersecurity has been invaluable in shaping the direction and execution of this work.

I am also thankful Dr Hoang Nguyen and Emirhan for their technical assistance, thoughtful discussions which contributed heavily on execution design of my test bed.

Special thanks to Swansea University, Computer Science Department for providing me with resources and environment to carry out conductive research.

Lastly, I would like to thank all my loved ones and church family for their prayers and words of encouragements through this academic journey.

I. Introduction

In the new global economy, sustainability concerns about pollution have grown, prompting automotive companies to focus their efforts on developing and mass production of electric vehicles (EVs). Heneghan et al., (2019) suggest that this trend has resulted in considerable advances in the development of EVs to solve these issues regarding environmental pollution. According to Parkinson et al., (2017), these technological breakthroughs have led development of self-driving EVs and better transportation efficiency using integrated communication networks in EVs. Jo and Choi (2022) point out that the development of EVs is based on software contained in Electronic Control Units (ECUs), which serve as a control centre for all vehicle components. As EVs rely more on computer systems for internal vehicle communication, safeguarding their security against potential external cyber-attacks becomes more critical. The Control Area Network (CAN) is an essential serial communication protocol for effective and real-time distribution control. Because of CAN's uncomplicated design and reliability attributes, it is the industry standard for intra-communication between ECUs. According to Jo and Choi (2022), CAN protocol can be connected to external networks using autonomous vehicle technologies making vehicle status monitoring procedures simpler. However, Mahmood et al., (2022) highlight that the development of vehicle-to-infrastructure and vehicle-to-vehicle communication technologies has created concerns regarding intra-vehicle communication by enabling more access vectors to internal networks. In the case of a breach of the CAN in a target car, attackers might take control of the vehicle by inserting malicious packets, potentially stopping the engine. Morris (2024) reports that according to West Midlands Police and Crime Commissioner (PCC) Simon Foster, thieves employ modern technology to bypass automotive security systems in seconds by duplicating keys or using signal boosters to gain entrance. These vehicles were frequently discovered in 'chop-shops', dismantled, and sold for components on the black market. It has been reported that Ford vehicles were the most stolen in his region, with 2980 thefts in 2023. Other brands, such as Toyota (109% increase), Hyundai (105% increase), and Lexus (408% increase), also see significant increases in thefts from the previous year. Cyber incidents can also have a fiscal impact on the automobile business, with the total cost of security breaches expected to be \$505 billion by 2024 according to *Edgelabs* (2023). Taken together these findings call for better security measures in EVs.

A. Motivation

Despite EVs' disruptive impact on the automotive sector, this invention has created security concerns that require rapid consideration. These concerns might be alleviated by protecting the EVs' internal network, which is why there has been an increase in research dedicated to identifying vulnerabilities in vehicle infrastructure and offering suitable mitigation measures and associated compromises. The underlying security issue with CAN protocol is that it was never designed with cybersecurity in mind, making it easier for attackers to induce purposeful disruptions by exploiting its lack of message authentication, encryption, and error-checking mechanisms. The goal of the study is to find and exploit potential vulnerabilities in CAN communication within car ECUs, followed by development of a security method to address the vulnerabilities discovered. As vehicles become more connected and rely on electronic systems, the risk of malicious attacks increases. Understanding the vulnerability of CAN systems to several types of attacks through fuzzing and analysing their impact on vehicle safety is crucial for developing robust security mechanisms.

B. Aims and Objectives

The primary aim of the study is to explore the vulnerabilities in the CAN protocol utilised in ECUs and determine how the vulnerabilities potentially jeopardise automotive safety and operation. The study presents a penetration testing technique based on fuzzing, message filtering and message authentication to address security concerns within the automotive CAN network. A bench was designed to present a model-based testing approach which depicted the internal vehicle setup of ECU to CAN network architecture. This enabled the visualisation of the CAN protocol's resistance to various attack scenarios when message authentication and message filtering techniques are employed as a vehicle network security strategy. The study aimed to evaluate the protocol's resistance to fuzzing assaults and identify potential vulnerabilities attackers may exploit.

The specific objectives of the research were:

- To study current vulnerabilities in CAN-based communication inside ECUs.
- To assess the implications of exploiting these vulnerabilities on a vehicle's safety, essential operations such as sensor manipulation and control systems,

- To evaluate the effectiveness of adopting security measures like message authentication and filtering against these vulnerabilities.
- To assess the performance and latency consequences of adding security measures to the CAN protocol.

C. Structure of the paper

The rest of the paper is organised as follows: Section II gives a background into the evolution of vehicles over the years, and Section III examines existing research on ECU and CAN security solutions. Section IV describes the study's initial stages of testing methodology (Information gathering and Threat Modelling) as well as the ethical considerations and problems encountered. Section V presents the findings of the penetration testing. Section VI evaluates the test bed's vulnerabilities, exploits them and reports on the observed implications. Section VII concludes the study by summarising the key findings and suggesting potential future research directions. The Appendix provides additional figures references in the main body. In this paper the term 'the study' is used to describe the project in discussion in its entirety.

II. Background

The evolution of electronics had a direct influence on the development of ECUs (Electronic Control Units); as electronic components became smaller and more practical, the production of electronic control systems developed, resulting in the first invention of electronic vehicles in the 1870s. [Fig 1.1](#) (Show et al., 2024) shows a timeline of the evolution of automobiles over the years. The advancement of electronic cars has resulted in self-driving, secure and efficient transportation; yet, this increasing complexity implies more complicated vehicle systems, which serve as a lure for hackers. This section will review the EV components and procedures used in this project to stress the importance of automotive cybersecurity, which can be difficult to manage if something goes wrong.



Fig 1.1: Evolution of Vehicles I

A. Automotive ECU

ECUs are embedded systems in EVs that control one or more electrical systems or subsystems. According to (Choi et al., 2021), ECUs were originally designed to be utilised to guarantee that engine performance is optimal (efficient gasoline and oil consumption), but they are now employed for a different purpose for convenience. These devices are important to the operation and functionality of modern automobiles, managing everything from engine performance to safety systems. Mokhadder et al., (2021) state that ECUs, receive inputs from several sensors throughout the vehicle and pass these inputs via multiple communication protocols to operate actuators and conduct necessary actions. Modern cars now have hundreds of ECUs, each responsible for a distinct part of vehicle functioning. Below are some of the most common ECUs defined by Smith (2016). The study focused on designing a BCM (Body Control Module) and its associated modules that send sensor data.

- **Powertrain** – is the brain of the ECU. Controls over 100 elements, including charging systems, transmissions, emissions, and control modules.
- **Engine Control Module (ECM)** - Controls engine performance and efficiency, including fuel injection, ignition timing, and speed.
- **Transmission Control Module (TCM)** - Oversees transmission functioning and determines gear shift timings to guarantee smooth shifting.
- **Body Control Module (BCM)** - Controls functions connected to the vehicle's body, such as lights and door locking

- **Airbag Control Module** – regulates airbag deployment by analysing sensor data to determine when to deploy airbags in the event of a collision.

As much as the addition of ECUs to automobiles has improved the driving experience, the more ECUs a vehicle has the more complex the software systems are. Mahmood et al., (2022) warn that due to the large code base of ECUs, regular and timely updates are essential to improve functionality but also to resolve security-related vulnerabilities that might be exploited by malicious individuals, jeopardising vehicle security. In the same line, Heneghan et al., (2019) make a compelling case that ECU components are usually obtained from several sources, sometimes without clear documentation, this makes it difficult to connect and test automotive systems. Vehicle systems are real-time systems, so they deal with critical functions however, concurrent systems are difficult to test, especially in race conditions, which leads to system failures. Below are some use cases where the ECU vulnerabilities were exploited.

1. Use Cases

The ramifications of failing to create safe frameworks for ECUs were reported by Mokhadder et al., (2021) 's study, they stated that there was a successful hacking attempt on the ECU of a 2014 Jeep Cherokee. The attack was accomplished by remotely exploiting a weakness in the vehicle's head unit thereby seizing control of critical driving subsystem operations including steering and braking. Parkinson et al., (2017) reported another case demonstrated by researchers at the University of California who proved that when there are no security protections in an ECU, they were able to upload new firmware to the ECU by upgrading firmware via the on-board diagnostic (OBD) without any resistance on the system. Changing the ECU firmware can radically reprogram the vehicle's behaviour, endangering the driver's life. Since connected vehicles are becoming more common, they must follow cybersecurity standards; yet, because this industry is still in its preliminary stages, there is a major shortage of research on the best strategies for adequately safeguarding automobiles.

B. Vehicles Network Protocols

For subsystems in ECUS to communicate with each other they use bus protocols that transfer packets through the network as reported by Smith (2016). Several ECUs and sensors connect using this bus protocol, sending signals that regulate how a vehicle behaves and what information the

network has at any given time. According to Abbott-McCune and Shay (2016), some of the bus protocols include SAE J1850, the oldest communication protocol used for OBD-II and vehicle networking, LIN (Local Interconnect Network), which is used for low-cost applications primarily in body electronics, FlexRay bus for high-speed synchronised data communications, and MOST (Media Orientated Systems Transport) for multimedia communications. Employing the approach of comparing vehicle communication protocols described by Smith (2016) and Aliwa et al., (2021) in [Table 2.1](#) below, CAN is the most recommended communication technology in vehicles. The study focused on the CAN protocol, the current standard internal network protocol for cars.

Protocol	Application	Bitrate
SAE J1850	Basic Diagnostic, OBD-II	Up to 41.7 kbps
MOST	Multimedia Communication	Up to 150 Mbps
LIN	Non-critical applications	1 kbps to 20 kbps
FlexRay	Safety- Critical applications	Up to 10 Mbps
CAN	Critical Real Time Applications	Up to 1 Mbps (CAN), Up to 8Mbps (CAN FD)

Table 2.1: Vehicle Networks 1

1. CAN (Controller Area Network)

Bosh GmbH created CAN in 1983 intending to provide a low-cost communication bus for car applications, but it is now utilised in a variety of industries, including aerospace, trains, and medical equipment (Choi et al.,2021). Smith (2016) claims that the CAN protocol has been the norm for vehicle communication in the United States since 1996, but it was not mandated until 2006, and in Europe in 2001. According to Choi et al., (2021), CAN is the most used communication protocol for ECU (Electronic Control Unit) communication due to its robust communication channels and simple network architecture. The internal vehicle CAN network architecture is divided into two categories: high-speed and low speed (Smith, 2016). The low speed CAN consists of ECUs that execute simple activities such as opening and closing doors on non-critical bus lines. Other crucial systems, such as the engine and brakes, are connected to the high speeds seen on critical bus routes. The illustrated a low-speed CAN BCM scenarios to manipulate Headlight and Door Control modules and a high-speed scenario for car belt status.

a) CAN bus Wiring

According to Jo and Choi (2021) an ECU is composed of three parts: a host processor, a CAN controller and CAN transceiver as shown below in [Fig 1.2](#) to present CAN architecture. CAN transceivers are used to connect CAN Controller to the physical transmission medium (CAN bus) and these transceivers always have two can bus pins CAN high and Can Low.

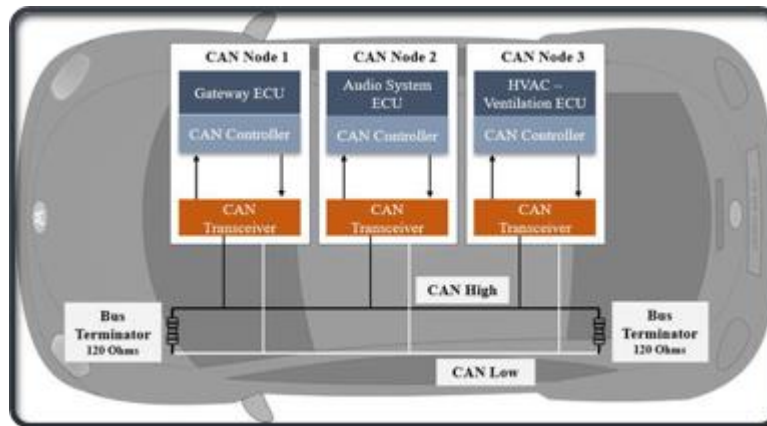


Fig 1.2: CAN Architecture (Boland,2021) 1

According to Smith (2016), by using these two lines a CAN transceiver can output certain voltage output levels to send either a dominant bit (0 bit) or recessive bit (1 bit) on the CAN bus. In contrast to low-speed CAN, CAN employs differential signalling. This implies that when a signal is received, CAN increases the voltage on one line while decreasing the voltage on the other line proportionally. Differential signalling is utilised in applications that require fault tolerance to noise, such as automotive systems and manufacturing. The sensors in an ECU contain a transceiver that checks to see if both signals are activated; if not, the packet is rejected as noise. To terminate the wires, Smith (2016) states that a 120-ohm resistor should be placed across each of them at the end of the term.

b) Can Bus Packet Layout

According to Choi et al., (2021), the CAN protocol offers four types of frames: i) Data frame ii) Remote frame iii) Error frame iv) Overload frame. The study focuses on the data frame to identify anomaly CAN messages sent over the bus. According to Smith (2016) and Werquin et al., (2019), data frames are classified into two types: base formats (Standard CAN – CAN 2.0A), which have an identifier (ID) length of 11 bits, and extended frame formats (Extended CAN – CAN 2.0B),

with an identifier (ID) length of 29 bits, made up of an 11-bit identification and an 18-bit extension. Smith (2016) points out that CAN packets contain 4 key elements:

- **Arbitration ID** – is a broadcast message that indicates the ID of the device attempting to interact, and any connected device may transmit several arbitration IDs. The ID also indicates the message's priority. If more than one CAN message is sent across the CAN bus, the lowest arbitration ID prevails.
- **Identifier Extension (IDE)** - This bit is always set to 0 in Standard CAN. If the bit is 1, it indicates that the 18-bit extended identification is being used. The study used standard CAN messages.
- **Data length code (DLC)** - This is the size of data, which spans between 0 and 8 bytes.
- **Data** – This is the data itself. The maximum size of data carried by the standard CAN is 8 bytes, although certain systems require 8 bytes of padding out the packet.

Werquin et al., (2019) argue that CAN requires a constant data transmission rate and allows recessive bits (one) to be substituted by dominant bits (zero) during transmission. This simplifies message acknowledgement by overwriting the ACK bit at the end of the data frame in real time. Similarly, to execute bus arbitration, CAN transceivers must continuously listen on the bus while supplying the message ID at the beginning of the data frame, and then turn off when one of their ID bits is overwritten. This method ensures that messages with lower IDs receive higher priority. Finally, each CAN frame includes a 16-bit CRC field to identify transmission errors. [Fig 1.5](#) shows a diagram of a CAN Data Frame and [Table 2.2](#) defines the various parts that make up a data frame.

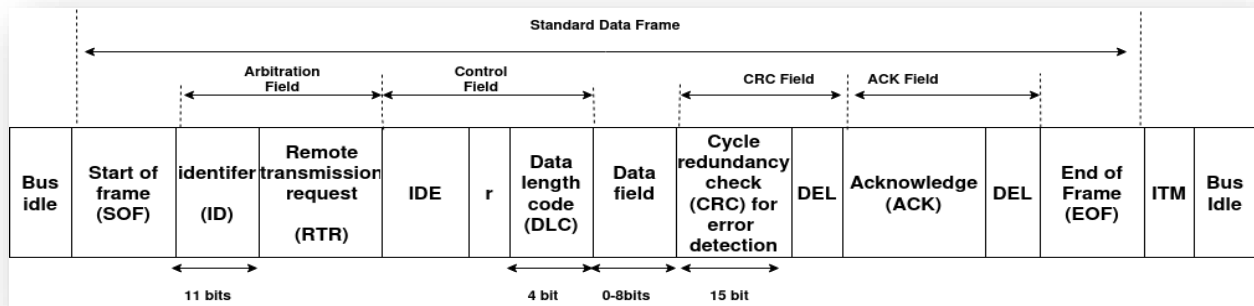


Fig 1.5: Standard CAN data frame 1

(Placeholder1)	Length of bits	Description
Start-of-frame	1	Means the beginning of frame transfer.
Identifier	11	Message ID, determines priority (Lower ID = higher priority)
Remote Transmission Request (RTR)	1	Distinguishes between data frame (0) and remote frame (1)
Identifier Extension (IDE)	1	Distinguishes between standard (0) and extended frames (1)
Reserved bits (r0)	1	Reserved for future user, set to 0
Data Length Code (DLC)	4	Indicates the number of bytes of data (0 to 8 bytes)
Data field	0-64 (0-8 bytes)	Actual data being transmitted (payload)
CRC	15	Cyclic redundancy check error detection
CRC Delimiter	1	Must be recessive (1)
ACK slot	1	Acknowledgment from receivers (dominant bit 0)
ACK delimiter	1	Must be recessive sate (1)
End-of-frame (EOF)	7	Must be recessive sate (1)
Intermission	3 (minimum)	Idle time between consecutive frames

Table 2.2: CAN Frame Components 1

Despite the advantages, CAN lacks security protections leaving the vehicle vulnerable to malicious attacks intended to cause malfunctions and eavesdropping of can messages exchanged on the can bus. Smith (2016) highlights that CAN bus packets are broadcast, so all ECUs on the same network view every packet, and the packets do not include information about which ECU sent the message, making it simple to spoof messages on the CAN. Spoofing is the practice of imitating a genuine network or device to deceive users, obtain unauthorized access, or carry out harmful acts. Supporting Smith (2016)'s argument on lack of security on CAN, Aliwa et al., (2021) state that the CAN bus transmission is not encrypted; therefore, it may be easily read using a data sniffing attack. Supporting this view (Choi et al., 2021) highlight that the major problem with CAN is that it does not support message authentication, which makes it easier for attackers to violate the security properties of the network communication and cause intentional malfunction. Jadoon et al., (2018) describe message authentication as a critical component of vehicle communication

protocols; it assures that each received message arrives in the same state in which it was sent by the sender. A sender's ID, location, and property must be validated to ensure that the sender is legitimate and provides trustworthy information. Aliwa et al., (2021) make a case that accessing the CAN bus network through external interfaces and connections could increase the entry sites for attacks. Below are some use cases where some of the vulnerabilities have been exploited.

(1) Use Cases

As reported by Mokhadder et al., (2021) in September 2016, a team from Tencent's Keen Security Lab hacked a Tesla. The team executed a remote assault on a Tesla Model S while in both parking and driving modes by injecting malicious CAN messages onto the CAN bus using external interfaces (Wi-Fi). Parkinson et al., (2017), reported a more direct assault in his paper, named 'bus tapping', which was carried out on BMW vehicles where the car keys were reprogrammed via the on-board diagnostic (OBD) connection. This type of attack is an extremely low-cost approach that only takes hooking into the OBD to bypass the vehicle's immobilising devices and then reprogramming a new key to start the automobile. These case studies highlight how vulnerabilities can be exploited in vehicles threatening driver's safety which calls for need for better security in electronic vehicles.

C. Penetration testing

Penetration testing (pen-testing) is a security assessment approach used by security testing specialists to conduct security testing by mimicking hackers and launching hostile attacks on the systems being tested to expose security flaws. Luo et al., (2022), state that penetration testing is divided into five categories: information collection, threat modelling, vulnerability analysis, exploitation, and reporting. The initial stage of pen-testing is acquiring security-related information regarding the ECU and CAN architecture to model configuration. The threat modelling step evaluates system threats using threat analysis and risk assessment (TARA) to establish the optimum assault strategy. Vulnerability analysis combines data from the previous two steps to identify attack targets. The exploitation stage entails a real attack on the detected threat vulnerabilities to fulfil the purpose of penetration testing. Finally, the report phase summaries the whole penetration process and condenses test results. Salfer & Eckert (2015) recommended semi-automated penetration tools and techniques which can reveal attack surfaces and estimation of

vulnerabilities. In line with this idea to discover security flows in ECU-CAN network the study employed fuzzing techniques to simulate real-world attacks to test system's defences.

1. Fuzzing Techniques

Following Fowler et al., (2018) and Werquin et al., (2019) definitions of fuzzing techniques: fuzz testing aims to: a) send random inputs to the system interface, b) monitor the responses, c) identify and reset any failures, d) perform the operation an increasing number of times to cover a larger input value range. By combining pen-testing with fuzzing, allows for a more thorough security assessment of the EV system. Fuzzing allows for continuous security testing, which leads to accurate and comprehensive vulnerability discovery. It is suitable for agile development environments in EV manufacturing. Combining human-led penetration testing and fuzzing can lead to a more accurate and comprehensive vulnerability assessment. The study carried out security tests on the ECU's CAN protocol through fuzzing and message sniffing to identify vulnerabilities the gaps may exploit using PyCAN.

2. PyCAN

The study used PyCAN to program ECU functionalities and to generate fuzz tests. PyCAN is a Python library for supporting CAN. It is used to enhance the security of automotive networks by providing a simple interface for Python programs to communicate with CAN protocol, allowing for data collection, diagnostics, control, and monitoring in vehicle systems (*Python-can 4.3.1 manual*, 2023). PyCAN enables penetration testers to directly access and alter CAN bus communication from inside a Python environment, which is critical for evaluating message exchanges across multiple ECUs (Electronic Control Units), finding vulnerabilities, and guaranteeing communication protocol integrity. It supports message injection and manipulation, allowing testers to mimic a variety of attack scenarios such as altering messages, inserting malicious commands, and manipulating sensor data to gain access to the vehicle's ECU network. Using PyCAN in conjunction with fuzzing techniques enables for a more targeted, efficient, and effective identification of vulnerabilities in CAN bus implementations, thereby improving the overall security posture of connected cars.

III. Literature Review

ECUs are critical components of modern EVs as they manage functions across the vehicle's system from engine control to braking systems. The increased integration of ECU with network systems has made vehicles vulnerable to cyber-attacks. According to Heneghan et al., (2019) and Parkinson et al., (2017), manufacturers are using technology that has inherent security vulnerabilities because they are primarily focused on functionality in supply chain manufacturing. This is because of the tight time constraints in EV manufacturing so no additional effort is put into the vehicle's security. Considering the significant financial repercussions of automotive cyber-attacks, many academics (Mahmood et al., (2022), Heneghan et al., (2019), Mokhadder et al., (2021), Jo & Choi (2021), Aliwa et al., (2021), Choi et al., (2021), Luo et al., (2022), Abbott-McCune and Shay (2016), Parkinson et al., (2017)) have indicated an interest in developing cybersecurity solutions to detect and protect vehicles from cyberattacks. Penetration testing has emerged as one of the key strategies to assess vulnerabilities and to secure electronic vehicles. This section reviews prior research on electronic vehicle cybersecurity on how to defend the ECU and vehicle internal communications from malicious assaults, thereby generating the motivation for this project.

A. Automotive Cybersecurity

As electronic automobiles evolved significant efforts have been made to ensure vehicle security at all levels of design and operation. Such efforts are led by the International Organisation for Standards (ISO), an independent non-governmental organisation that develops and publishes standards in a variety of industrial sectors to enhance product and service quality, safety, and efficiency (ISO, 2011). ISO26262 for instance focuses on the functional safety of electrical and electronic systems in motor vehicles. The standard establishes a systematic approach to ensuring that automobiles are designed and manufactured in a manner that reduces the possibility of flaws that might contribute to accidents (ISO, 2011). However, while ISO26262 addresses functional safety it overlooks cybersecurity risks in vehicle internal communication, rendering it vulnerable to remote attacks. A broadly similar point was made by Heneghan et al., (2019) and Parkinson et al., (2017) who hold the view that automotive corporations do not invest in automotive system security due to the excessive costs associated with securing a vehicle's network system; therefore, they primarily focus on functional requirements, leaving vehicles open to malicious attacks.

In parallel, attempts to improve cybersecurity have gained traction. One prominent initiative is the Black Hat conferences, a worldwide renowned cybersecurity series of events conducted since 1997. These conferences bring together researchers, professionals, and industry leaders from the public and private sectors to explore innovative cybersecurity threats and solutions (Black Hat, 2024). For example, Miller and Valasek's 2014 Black Hat presentation revealed significant vulnerabilities that exist in many popular vehicles in the USA, aligning with the event's goal of educating professionals on cybersecurity trends (Parkinson et al., 2017). While Black Hat conferences are useful for raising awareness and exchanging expertise, they may not focus on preventative techniques as much as the automobile industry requires to protect EVs. The ongoing development of the automobile industry, along with quick introduction of new technologies at a rapid pace, makes it difficult for Black Hat conferences to keep up, resulting in presentations that are out of date. There is also a risk that participants would focus on exploiting vulnerabilities rather than adopting effective security measures to protect automobiles. [Fig 1.6](#) shows an image showing some of the different attack vectors that could be exploited in an EV (Kirthi, 2023). The study focuses on securing the vehicle's internal communication channel (CAN) from intruders.

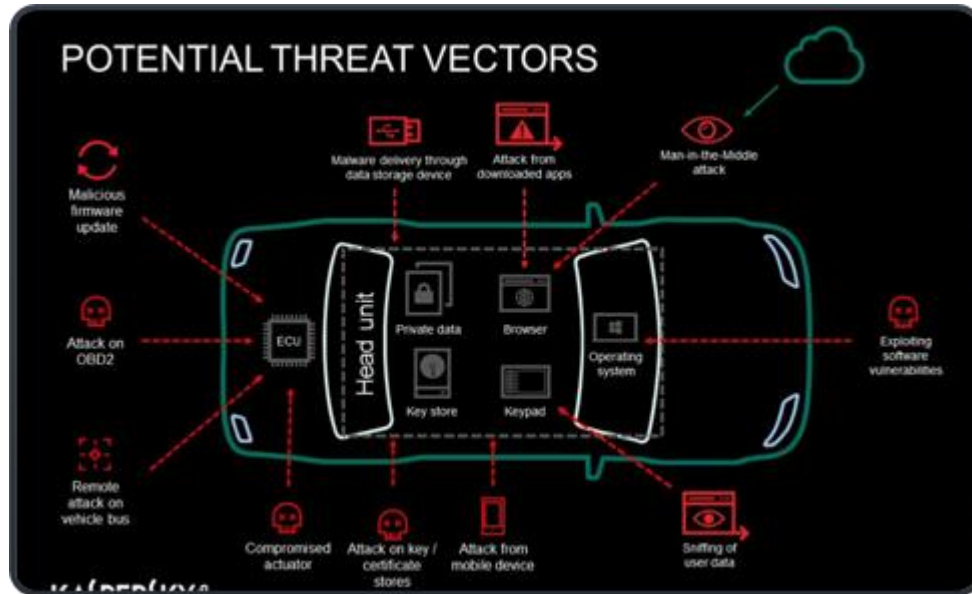


Fig 1.6: Potential Attacks 1

1. ECU Security

ECUs play a critical function in EVs, managing systems such as engine performance, gearbox, brakes, and safety. As cars become more networked and autonomous, ECUs are increasingly

vulnerable to cyber intrusions. An ECU is easily accessible meaning an attacker can easily upload new firmware onto the ECU thereby redefining the functionality of a car. Over-the-air (OTA) secure updates have been suggested by Mahamood et al., (2022) as one of the mitigation strategies to protect ECU firmware from security breaches. Mocnik et al., (2023), state that technology-driven automobile firms utilise OTA software upgrades to update functionality and address software security flaws. This increases the production of vehicles on the market. In Mahamood et al., (2022)'s research employs a systematic threat assessment and model-based testing approach, which combines threat modelling, building attack tree as well as automation of test case generation and execution. The adoption of STRIDE model for threat enumeration is advantageous as it provides an organised way to identify various security threats. The development of propriety software tool for automated test case generation and execution is significant contribution. This tool improves the efficiency and consistency of the security off the testing process. Mahamood et al., (2022) developed 15 separate test cases to validate Uptane framework's reference implementation. The results show the framework's strengths and weaknesses, providing valuable insights for real word applications.

Although the setup in the research paper resisted various assaults launched on the testbed, 7 of the security test scenarios failed. This demonstrates the necessity for additional security measures during the manufacturing of ECUs. This view is supported by Mocnik et al., (2023), who contend that suppliers' significance in the broader cybersecurity environment is underestimated; their assessments identified supply chains as one of the risk factors since the OTA system code must be fully protected from the start to end throughout development. Despite the advantages of OTA, it does not guarantee the complete safety of the vehicle. Mahmood et al., (2022) 's research was able to detect various risks to ECU security and mitigation strategies; however, it did not cover all test cases, such as a physical attack vector in which firmware is flashed to transmit arbitrary CAN signals, leaving significant attack surfaces undressed and open for attackers to exploit vulnerabilities. The research makes various assumptions (such as attackers having privileged access or compromised keys) that may not be valid in real world scenarios. These assumptions may affect the generalizability of the findings. The attack tree creation process is manual, which take time and is prone to human error. Automating this phase could further enhance the efficiency and accuracy of the approach

The study only focused on securing the CAN network however this paper was a good contribution to the literature of automotive cybersecurity concepts.

B. Security of CAN

Vehicle communication protocol is the most critical path of vehicle intra data transmission, CAN is the most common communication protocol used in modern automobiles. Given that the CAN bus lacks confidentiality protection, Aliwa et al., (2021) points out that an attacker can launch various attacks on the CAN bus network; CAN Sniffing, CAN Fuzzing, CAN Bus injection, CAN Replay and DoS attacks.

1. CAN Sniffing

Using an OBD-II it is possible to read and analyse data on the bus to manipulate and generate similar messages. CAN sniffing is an Eavesdropping attack. Jadoon et al., (2018) defines Eavesdropping as the sort of attack in which an adversary obtains sensitive and confidential information for which they are not allowed. It is a passive attack in which an attacker covertly sniffs the data to obtain sensitive information and then uses it for their own gain. Jadoon et al., (2018), Parkinson et al., (2017) and Aliwa et al., (2021) advise using cryptographic approaches to protect against unauthorised access and encryption to avoid the sniffing of CAN frames. Despite a large body of literature advocating encryption as a solution CAN Sniffing a practical solution for the optimal application of encryption has yet to be identified; this field of research is currently still being investigated.

Jadoon et al., (2018) published a comprehensive survey on the different current cryptographic techniques used in automotive cybersecurity. The authors did a deep dive into the specific cryptographic algorithms of both symmetric and asymmetric discussing their accessibility and limitations in automotive networks. Jadoon et al., (2018) identify the and categories security threats specific vehicular networks such as Replay Attacks, Eavesdropping, Denial of Service and many more. Denial of Service attacks are defined as delivering malicious messages to the CAN network to render a victim ECU inaccessible to other legitimate users by flooding, jamming, or widespread DoS assaults. Jadoon et al., (2018) presented details on how cryptographic techniques can counteract these tactics. The article discusses different symmetric algorithms, such as Blowfish, PBAS, Camellia, and CAST, which employ a single key for both encryption and decryption.

Asymmetric algorithms utilise a pair of keys—a public key for encryption and a private key for decryption. Examples described in the article include anonymous keys and certificates, ID-based proxy signatures, elliptic curve cryptography, RSU-aided authentication methods, and smart cards for identification. Given the resource constraints in vehicle networks Jadoon et al., (2018) emphasises light weight cryptographic solutions, this point is supported by Aliwa et al., (2021) who suggest use of light weight encryption to make up for the overload and complexity that come with the use of encryption algorithms.

Jadoon et al., (2018) were able to provide multiple lightweight encryption algorithms, but their research focusses on theoretical elements and literature reviews, rather than empirical validation. The research looks deeper into ARAN, SEAD, and Ariadne encryption algorithms, which employ both symmetric and asymmetric algorithms, indicating a biased review. A more balanced approach would offer a broader perspective on the protocols. The paper also fails to adequately address the scalability of encryption methods. Scalability is critical in vehicle networks since they are dynamic and on a large scale. The study excludes encryption approaches due to the cost of encryption algorithms and the payload byte length limits of standard CAN messages.

2. CAN Fuzzing

CAN bus lacks authentication and data integrity checks as a result the ECU accepts all messages transmitted on the CAN bus and responds to them regardless of origin which might be from a malicious attacker. Fuzzing is used to transmit random CAN data frames for testing the CAN bus and to analyse changes on the vehicle. The study used fuzzing for penetration testing to construct a black box testing approach using fuzz tests from Fowler et al., (2018), Mokhadder et al., (2021) and Werquin et al., (2019) academic papers.

According to Fowler et al., (2018), a fuzzer is software that performs fuzz testing. The fuzz test is a dynamic method of testing against a working system to identify code faults and security flaws. Fowler et al., (2018) demonstrated the straightforward creation of a bespoke fuzz tester to experiment with a target CAN bus for automotive testing. Fuzz testing was performed on the CAN bus, revealing vulnerabilities in its security. When the fuzz test was performed against the instrument cluster, it resulted in instantaneous Malfunction Indicator Illumination (MIL), warning noises, wavering gauge needles, and a digital display that read 'crash repeatedly'. Other car systems

displayed similar tendencies, such as unpredictable engine idle RPMs. Due to the possible damage to the target components, the experiment could only be conducted on a bench-top hardware setup. The benchtop was chosen to illustrate a common characteristic found in connected automobiles: watching the response to malicious signals compared to legitimate messages. The study employs a similar set-up built using Raspberry Pis with CAN Hats to represent ECU to CAN Bus connection. The fuzzer in Fowler et al., (2018)'s paper injected messages at a random rate of 1kHz to generate the correct, unlock command. The experiment successfully activated the unlock mechanism by using a fuzzer to create random CAN messages. The outcomes of this research show that fuzzing can discover vulnerabilities in automotive systems, as well as risks associated with fuzz testing during production, such as component damage and system failure. The research contributes significantly to the study of fuzzing as a security testing tool; Mokhadder et al., (2021) used this research paper as a reference in his research on developing an Automated Current Based Fuzzing System (ACFS); however, it does not propose methods to mitigate the vulnerabilities revealed by fuzzing. The use of a model-based testing approach is often manual and complex which can be time consuming and error-prone. The continuous rapid evolution of cybersecurity attacks makes this research somewhat irrelevant to the emerging attack vectors attacks are now exploiting.

Mokhadder et al., (2021) proposed a system that automates the fuzzing process, significantly reducing the manual effort required compared to Fowler et al., (2018) 's paper. The paper presents a lightweight reverse engineering tool for CAN communications based on Vector Tools that analyse frequency and filter signals to detect CAN message changes. The research used a VNI630A device to inject messages onto the CAN bus and examined live messages with a CANAnalyzer and CANoe software. ACFS is an automatic fuzzer that captures real-time CAN communications and electric current extracted from the vehicle battery. The fuzzing program was designed and tested on a PC running Windows 10, and it solely supported the CAN protocol. The technology was tested on a 2017 production prototype Breadboard Vehicle, showing its practical use. The algorithm was tested on four actions; turn on high beam on, activate windshield wipers and activate left/and right turn signals monitoring its log files. The study adopted some of these actions performed by the BCM ECU to visualise ECU behaviour on the hardware set-up circuit. Mokhadder et al., (2021)'s system was able to extract several messages controlling headlights, turn signals, and information clustering functions in seconds. The results showed that ACFS could

properly detect CAN signals associated with certain vehicle operations. The research proposed a novel approach to vehicle cybersecurity that involves connecting current data with CAN signals.

This research underlines the importance of fuzzing in automobile manufacturing to secure internal systems, since the fuzz tests were able to activate events on the car's ECU outside, assessing how vulnerable vehicles are to cyberattack. However, the ACFS system only supports the CAN protocol. As other vehicle communication protocols, such as CAN-FD in EVs, become more prevalent, the system's use is limited. The research advocates the use of additional sensors to measure vehicle activity, however this was not investigated in depth. Although the system minimises human work, it requires user feedback to validate the occurrence of events during the reverse engineering process, which may restrict the system's ability to be fully automated.

To expand on the fuzzing topic, Werquin et al., (2019) propose the use of sensor harnesses to automate fuzzing oracles for ECU with physical outputs. The purpose was to identify vulnerabilities and reverse engineer ECU functionality in CAN networks. The sensor harness developed by the writers was used to detect the physical responses from ECUs during fuzzing testing. As defined by Werquin et al., (2019)' a Bug Oracle is a program that assesses if a certain execution of the target system violates a specific security policy. In the research, Oracle functions were used to detect physical responses from the sensor harness to inform the fuzzing process. The sensor harness provides real-time feedback to how ECUs respond to fuzzed inputs, the feedback is used to guide the generation of the next inputs during automated exploration. Werquin et al., (2019) define the different fuzz tests to be carried out on the target system; Random Fuzzing, Brute-Force Fuzzing, Mutation Fuzzing, Identify Fuzzing. Random fuzzing selects arbitration IDS and message payloads randomly. The study adopted the first four fuzzing techniques to test the robustness of hardware set-up test bench against fuzzing attacks by monitoring ECU behaviour.

Werquin et al., (2019)'s research discovered that fuzz tests can reveal undocumented functionality, intricate bugs, and security vulnerabilities in ECUs connected to CAN networks, demonstrating that fuzzing is a useful tool in testing and reverse engineering because it covers a wide range of actions that attackers may execute on a vehicle system. The use of a sensor harness was an innovative approach, but it relies heavily on the condition that external environmental factors are stable. Some environmental factors, such as noise or light interference, may interfere with sensor

readings, resulting in false positives, making it difficult to verify its correctness compared to actual ECU behaviour. The accuracy of detecting physical responses from ECUs may vary depending on the implementation and type of sensors used in the EV. Luo et al., (2022) advocate using many sensors to detect different sorts of physical responses to boost accuracy and coverage, while also using machine learning techniques to improve sensor interpretation and anomaly identification. Choi et al., (2021) use machine learning to differential ECUs in their research article “*Identifying ECUs Using Inimitable Characteristics of Signals in Controller Area Networks*” will be reviewed in detail in the sections below.

3. CAN Replay

CAN messages delivered on the CAN bus are captured by sniffing them using tools like OBD-II or directly on the CAN bus. Once collected, the messages can be ‘replayed’ on the can bus later. The replayed messages are identical to the originals, which allows the vehicle to perceive them as genuine data.

4. CAN Bus Injection

The attackers transmit bogus or spoofed CAN messages to the vehicle's CAN bus at an abnormal rate. According to Aliwa et al., (2021), the goal of this attack is to modify the frequency and quantity of CAN frames on the CAN bus, as well as the sequence of legitimate CAN frames and data payloads. Given that there is no message authentication or encryption in the CAN network, data on the bus can be monitored to generate messages that simulate vehicle control events. Aliwa et al., (2021) gave an example of encoding packets in WMA audio files with executable code that sends CAN packets onto the network. WMA audio files support the insertion of images and other text-based information, which is processed by the media player. This has an impact on the vehicle since it may create major changes in its behaviour. Based on the previous section on CAN Fuzzing, message injection could be utilised to implement fuzzing techniques during security testing.

Several researchers (Aliwa et al., (2021), Choi et al., (2021), Luo et al., (2022) and Parkinson et al., (2017)) have advocated using message authentication codes (MAC) to prevent CAN Replay and CAN Bus Injection attacks on the CAN network. Parkinson et al., (2017) ‘s study reports that researchers at Yokohama National University in Japan assessed the utilization of statistical intrusion-based systems using MAC and cryptographic solutions as security mechanisms.

Authentication techniques can be used to assure authentication and data integrity during in-vehicle communication. However, as highlighted by Aliwa et al., (2021) this solution does not provide anonymity because CAN communication is still open to sniffing and reverse engineering attacks, hence they recommend a combination of MAC and encryption. The study however only concentrates on using MAC. Most academic studies (Aliwa et al., (2021), Luo et al., (2022) and Parkinson et al., (2017)) in their papers concentrate on the literature of CAN security rather than the implementation of proposed solutions using model-based testing (MBST).

Mahmood et al., (2022) describe model-based testing as using models to verify that the target system fulfils its security requirements by documenting and producing security test goals systematically and efficiently. MBST is a very new area of research, and there are very few studies that use it. This is because model-based testing needs a high degree of ability; if the system is complicated constructing models can be difficult. The study attempts to bridge the gap by developing an MBST for testing and implementing the use of MAC using HMAC which employs the use of key distribution and pre-shared symmetric. Aliwa et al., (2021) and Parkinson et al., (2017) also suggest the use of short MAC tags (less than 8 bytes) owing to the limited computational capability within ECUs which the study took into consideration in the generation of MAC tags. However, Aliwa et al., (2021) point out the drawback that adding keys to an ECU adds manufacturing complexity and creates compatibility issues when replacing an ECU in a vehicle. To address the shortcomings of MAC, Choi et al., (2021) propose ECU identification methodology that compromises reading electrical signals from the ECU to train machine learning models. The trained models will be used to discriminate ECUs based on their electrical signals. This technique is consistent with the idea proposed by Luo et al., (2022) of employing Artificial Intelligence (AI) to secure communication in EVs and applying access control methods to protect the vehicle from cyberattacks. However, this paper does not employ such techniques, even though it could be a sensible addition to the study in the future.

IV. Methodology

The methodology of the study encompassed several steps aimed at establishing project questions, a controlled testing environment, simulation of ECU functionalities, conducting penetrating testing via fuzzing and implementation of security measures

A. Research Questions

The major goal was to use PyCAN to develop a penetration testing framework for vehicle Electronic Control Units (ECUs) to detect vulnerabilities in communication protocols and potential security problems. This section presents three research questions (RQs) on the intended research:

- RQ1: What current vulnerabilities exist in the ECU's CAN?
- RQ2: How does exploiting these vulnerabilities affect automobile system safety and functionality?
- RQ3: What mitigation strategies can protect CAN against these vulnerabilities and how resistant are they to attacks?
- RQ4: What is the performance latency of employing security protection mechanism?

RQ1 analyses common security vulnerabilities in the communication protocols used by vehicles' ECUs. Research on this topic is critical for developing tailored penetration testing strategies. RQ2 analyses the consequences of effectively exploiting vulnerabilities in vehicles' communication protocols. It seeks to understand the possible risks to vehicle safety and functionality, which is critical in determining the severity of found vulnerabilities. RQ3 focusses on developing security strategies to protect CAN protocol from known vulnerabilities. Its purpose is to suggest and assess viable mitigation measures for improving the security posture of automotive systems. RQ4 analyses the performance latency adding message authentication to ECU communication.

B. Ethical Considerations

The study followed applicable laws, regulations and standards governing automotive cybersecurity. It followed to the National Highway Traffic Safety Administration ISO/SAE 21434 standard for cyber security engineering, the test environment was carefully designed to minimise any impact on real work vehicles or networks. The use of isolated platforms such as Raspberry pi to simulate ECU communication guaranteed that all experiments took place in a controlled and secure setting and would not jeopardise public safety or violate regulations around automotive security testing. The identification of security vulnerabilities was done with the intent to improve, rather than exploit vehicle cybersecurity risks. The vulnerabilities discovered were reported in a way that highlighted the need for security improvements. This approach followed the guidelines of

responsible disclosure ensuring that information shared in the study was constructive and focused on enabling manufactures and researchers to strengthen security. By addressing these points, the study adds to the ongoing endeavour to secure ECUs' internal networks, ensuring that EVs stay safe and resistant to cyberattacks.

C. Penetration Testing

Pen-testing was conducted to determine the weaknesses of a vehicle's CAN bus communication network. This technique included replicating a real-world attack scenario with numerous ECU nodes, including a dedicated attacker node, to assess the efficiency of security features such as MAC. This section describes the phases of pen-testing process: information gathering, threat modelling, vulnerability analysis, exploitation, and reporting. The full code implementation is on this [GitHub repository link](#).

1. Information Collection

The initial stage involved setting up the automobile network which consisted of two Raspberry Pis units equipped with RS485 CAN Hats to present standard vehicle ECU communication. These two devices were connected to the same CAN bus to imitate a typical data transmission between vehicle systems. Raspberry Pis were selected as the preferable embedded device for the study since they are cheaper than the other single-board computers and embedded devices. Raspberry Pi has a huge and active community of users and developers that provide tutorials and forums to advise on how to utilise it. They are also adaptable and versatile, thanks to their GPIO pins, which allow connection to a broad range of peripherals. RS485 CAN HATs were attached to each Raspberry Pi GPIO pin to establish a CAN interface to the CAN bus. A circuit was built with output devices (LEDS) attached to the GPIO pins, to simulate ECU responses on event triggers. The attacker ECU also represented by the third Raspberry Pi with an RS485 CAN Hat was added to this CAN bus. This attacker node was programmed to observe traffic and inject malicious messages as part of the fuzzing simulations. The CAN bus used in this set up shown in [Fig 1.5](#) (Matrickz TV, 2019) adheres to typical automotive communication standards, facilitating the exchange of control messages between ECUs. [Appendix 2](#) shows the complete hardware setup. Below are the steps taken to set up the hardware devices to provide a baseline for regular ECU communication and allow monitoring of normal CAN traffic.

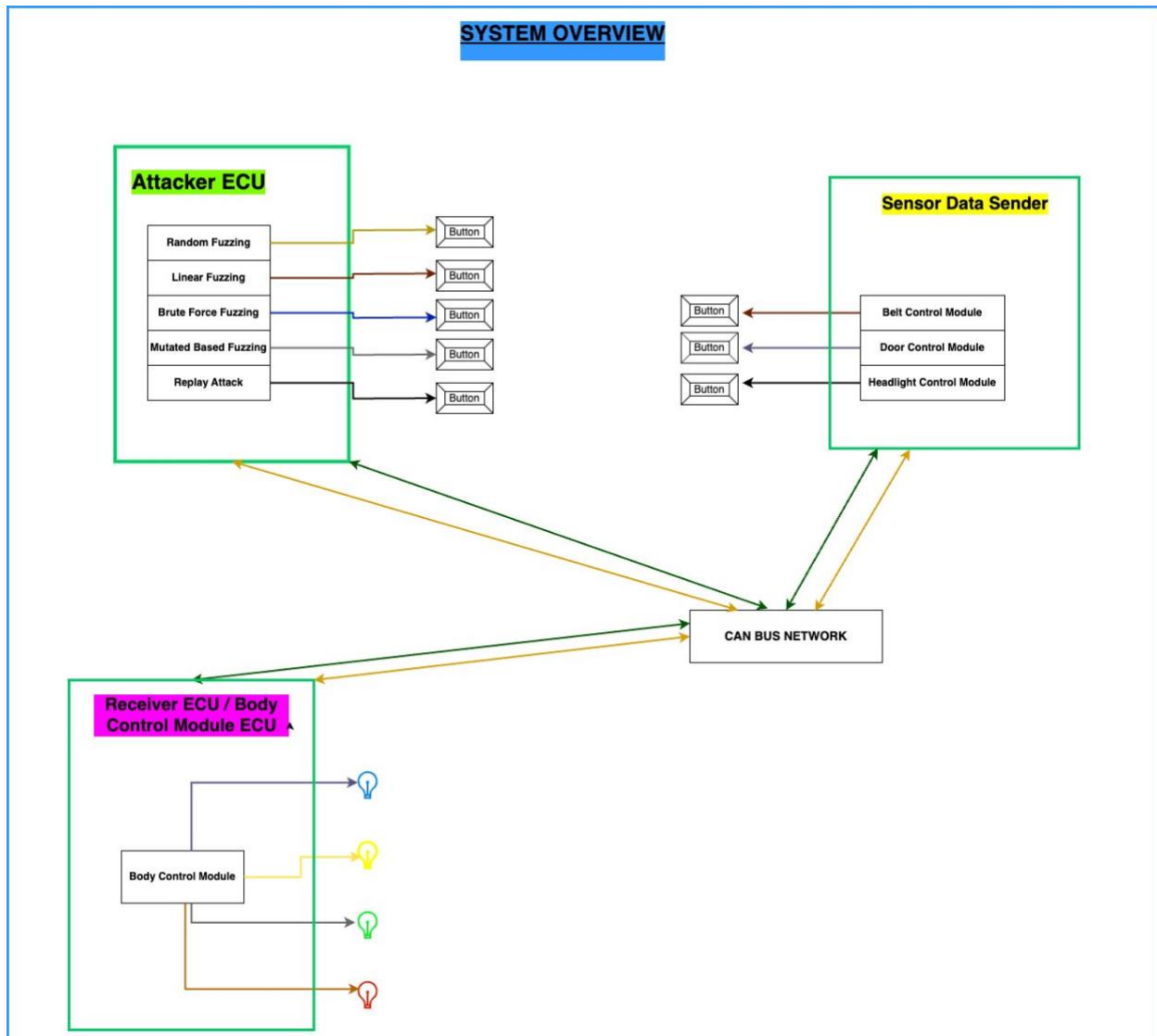


Fig 1.5: Model Setup Plan 1

Steps for circuit connection referring to (RS485 CAN HAT - Waveshare Wiki, (2024) and Morgan, (2019))

1. Set up the Raspberry Pi
 - a. Download the Raspberry Pi Imager Application
 - b. Plug micro-SD using a USB card reader into a laptop, Format Micro SD card
 - c. Enable Wi-Fi connectivity and setup username and password
 - d. Install the Raspberry Pi 4 operating system on Micro SD card

- e. Unplug microSD
- f. Insert the microSD into the raspberry pi, plug a mouse and keyboard into USB ports, plug HDMI cable into monitor, plug the power supply and boot up raspberry pi
- g. Check device's MAC Address
 - i. `cat /sys/class/net/wlan0/address`
- h. Configure network settings to the local location GB
- i. Connect to *SwanseaUniSetUp* Wi-Fi network using device MAC address
2. Update System packages run in a terminal, to ensure all packages are installed:
 - `sudo apt update && sudo upgrade`
3. Download and install VS Code as the chosen IDE for programming ECU code on raspberry pi devices.
4. Sign into GitHub and clone chosen repository to configure cloud version control for project across all devices
5. Install Python- 3 libraries and the CAN-utils library using the below commands in terminal:
 - `sudo apt-get install Python3-pip Python3-dev`
 - `sudo apt-get install Python3-serial`
 - `sudo apt install Python3-can`
 - `sudo apt-get install can-utils`
6. Connect RS485 HAT to Raspberry pi
 - a. Carefully align the pins of the *RS485 CAN HAT* with the GPIO pins on the Raspberry Pi and attach it securely.
 - b. Configure the Raspberry Pi to use the CAN HAT by changing the interface to SPI, which is compatible with the CAN interface. According to the *RS485 CAN HAT - Waveshare User Manual*, the MCP2515 CAN controller is used via SPI interface, with the onboard transceiver *SN65HVD230* operated by UART, it is designed for the Raspberry Pi and enables automated TX/RX control without the need for scripting, allowing for half-duplex communication.
 - i. Enter command in terminal:
 - `sudo raspi-config`
 - Select "Interfacing Options" > "SPI" > "Yes"

- Configure the CAN interface by changing the configuration file (`sudo nano /boot/firmware/config.txt`) and enter the lines below:

```
dtoverlay=spi=on
```

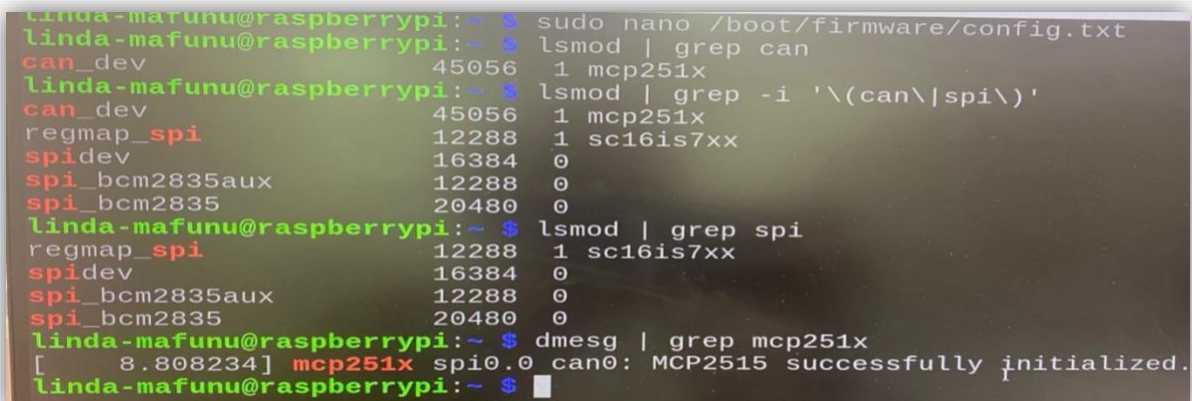
```
dtoverlay=mcp2515can0,oscillator=16000000,  
interrupt=25, spimaxfrequency=1000000
```

The MCP2515 CAN controller was set up on can0 with a 16MHz oscillator and an interrupt on GPIO pin 25 with a maximum frequency of 1MHz. The spimaxfrequency parameter is critical in determining the speed of SPI communication between the Raspberry Pi and the MCP2515.

7. Reboot raspberry pi:

- `sudo reboot`

“After restarting, the drivers for SC16IS752 and mcp251x will be loaded into the system kernel; execute the command to see if the initialization was successful.” (RS485 CAN HAT - Waveshare Wiki, 2024)) [Fig 1.6](#) shows the ways in checking whether the CAN HAT is effectively operating on both devices.



```
linda-mafunu@raspberrypi:~$ sudo nano /boot/firmware/config.txt
linda-mafunu@raspberrypi:~$ lsmod | grep can
can_dev                45056      1 mcp251x
linda-mafunu@raspberrypi:~$ lsmod | grep -i '\(can\|spi\)'
can_dev                45056      1 mcp251x
regmap_spi             12288      1 sc16is7xx
spi_dev                16384      0
spi_bcm2835aux         12288      0
spi_bcm2835            20480      0
linda-mafunu@raspberrypi:~$ lsmod | grep spi
regmap_spi             12288      1 sc16is7xx
spi_dev                16384      0
spi_bcm2835aux         12288      0
spi_bcm2835            20480      0
linda-mafunu@raspberrypi:~$ dmesg | grep mcp251x
[  8.808234] mcp251x spi0.0 can0: MCP2515 successfully initialized.
linda-mafunu@raspberrypi:~$
```

Fig 1.6: CAN Interface Initialised 1

8. Connect the ECUs on the same CAN bus:

- Connect a jumper wire from ECU1 CAN_H to breadboard 60. Connect a jumper wire from the breadboard to the ECU2 CAN_H and ECU3 CAN_H using the same serial line.
- Connect the jumper wire from ECU1 CAN_L to Breadboard 55. Along the same serial line, connect a jumper wire from the breadboard to ECU2 CAN_L and ECU3 CAN_L. The circuit connection is shown in [Fig 1.7: Test bench set-up](#)

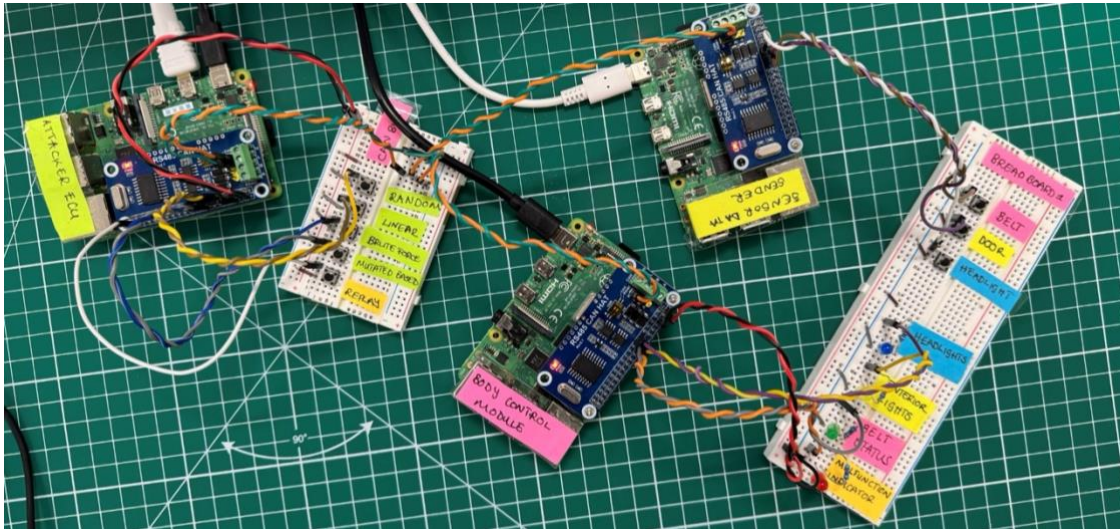


Fig 1.7: Test bench set-up 1

- c. Bring Up the CAN the Interface (RS485 CAN HAT - Waveshare Wiki, 2024):
 - i. Set can0 interface speed to 500 Kbps and sample point 0.875 `sudo ip link set can0 up type can bitrate 500000 sample-point 0.875`
 - ii. Set to can0 to “steady” state:
 - `sudo ip link set can0 up`
 - iii. To bring down interface:
 - `sudo ip link set can0 down`
- d. To get more information about configuration options type:
 - `sudo ip link set can0 type can help`
- e. [Fig 1.8](#) shows how to check the status of the CAN Bus

```

linda-mafunu@raspberrypi:~$ dmesg | grep mcp251x
[ 8.808234] mcp251x spi0.0 can0: MCP2515 successfully initialized.
linda-mafunu@raspberrypi:~$ sudo ip link set can0 up type can bitrate 500000 sample-point 0.875
linda-mafunu@raspberrypi:~$ sudo ip link set can0 up
linda-mafunu@raspberrypi:~$ ip link show can0
3: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc pfifo_fast state UP mode DEFAULT group default qlen 10
    link/can
linda-mafunu@raspberrypi:~$

```

Fig 1.8: CAN interface status 1

9. To test the communication, send a can message to one ecu from 2 ECUs on the can0 interface. The communication is displayed in [fig 1.9](#), [fig 2.0](#) and [fig 2.1](#) below showing a three-way communication between the ECUs.

```

linda-mafunu@raspberrypi:~$ candump can0
can0 01010101 [8] 01 02 03 04 05 06 07 08
can0 01010101 [8] 01 02 03 04 05 06 07 08
can0 01010101 [8] 01 02 03 04 05 06 07 08
^C
linda-mafunu@raspberrypi:~$ candump can0
can0 01010101 [8] 01 02 03 04 05 06 07 08
can0 000 [4] 11 22 33 44
can0 01010101 [8] 01 02 03 04 05 06 07 08
can0 000 [4] 11 22 33 44

```

Fig 1.9: Attacker ECU 1

```

linda-mafunu@raspberrypi:~$ sudo ip link set can0 up type can bitrate 500000
sample-point 0.875
linda-mafunu@raspberrypi:~$ sudo ip link set can0 up
linda-mafunu@raspberrypi:~$ cansend can0 000#11.22.33.44
linda-mafunu@raspberrypi:~$ cansend can0 000#11.22.33.44
linda-mafunu@raspberrypi:~$

```

Fig 2.0: Receiver ECU 1

```

linda-mafunu@raspberrypi:~$ cansend can0 01010101#0102030405060708
linda-mafunu@raspberrypi:~$ cansend can0 01010101#0102030405060708
linda-mafunu@raspberrypi:~$ cansend can0 01010101#0102030405060708
linda-mafunu@raspberrypi:~$ cansend can0 01010101#0102030405060708
linda-mafunu@raspberrypi:~$ cansend can0 01010101#0102030405060708
linda-mafunu@raspberrypi:~$

```

Fig 2.0: Receiver ECU 2

The study employed Thorne (2024)'s Python code to program ECU functionality as a basis shown below:

```

# Import the library
import can

# Create a bus instance using 'with' statement,
# This will cause bus.shutdown() to be called on the block exit;
# Many other interfaces are supported as well (see documentation)
with can.Bus(interface='socketcan',
channel='vcan0', receive_own_messages=True) as bus:
    # Send a message

```



```

    message = can.Message(arbitration_id=123, is_extended_id=True,
data= [0x11, 0x22, 0x33])
    bus.send(message, timeout=0.2)
    # Iterate over received messages
    for msg in bus:
        print(f"{msg.arbitration_id:X}: {msg.data}")
    # or use an asynchronous notifier
    notifier = can.Notifier(bus, [can.Logger("recorded.log"),
can.Printer()])

```

This code, however, was not enough to mimic real-world ECU behaviour scenarios it was just used as a base to have code that enables 3-way communication between ECUs. The below scenarios we used as a basis to program the ECUs functionalities. Sender ECU has Python scripts representing Door Control Module, Belt Status Module and Headlight Control Module. Receiver ECU has the Python script for the Body Control Module. Attacker ECU has Python scripts for fuzz tests.

a) ECU Scenario Implementation

Scenario 1: Automatic Headlight Control

The systems alter the headlights in response to ambient light conditions measured by a light sensor. When the light level goes below a specified threshold, the headlights switch on automatically. Headlight status is not a safety critical scenario so the CAN messages to BCM will be of low-medium priority (CAN ID-0x400)

- **Body Control Module (BCM):** handles the different electrical systems in the car; its role in this case is to receive and interpret light sensor data and regulate the headlights appropriately.
- **Headlight Control Module (HCM):** Measures ambient light levels and delivers the info to the BCM.

Scenario 2: Door Status

The system uses sensors to monitor the door status it sends a warning message if the door is unlocked it triggers interior lights to be on. Door status is not safety critical scenario so the CAN message to BCM will be of medium priority (CAN ID -0x200)

- **Door Control Module (DCM):** checks if the door is locked, or unlocked sends CAN messages to the BCM based on status
- **BCM:** its role in this case is to receive and interpret the door control data and act as an Infotainment Unit by sending diagnostic messages to the diver through output sensors.

Scenario 3: Belt Status

The system uses sensors to monitor the belt status it sends a warning message if the belt is on or not triggers dashboard lights to be on. Seat belt status is a safety critical scenario so the CAN messages to BCM will be of high priority (CAN ID - 0x100)

- **Belt Sensor Module (BSM):** checks if the belt is on or not sends CAN messages to the BCM based on status
- **BCM:** its role in this case is to receive, interpret belt status data and act as an Infotainment Unit by sending diagnostic messages to the diver through output sensors

Scenario 4: Malfunction Indicator and Diagnostic Logging

According to Smith (2016) “When a vehicle encounters a malfunction, it records information related to that fault and activates the engine warning light, commonly known as the malfunction indicator lamp (MIL). The vehicle’s primary ECU conducts these route checks, the powertrain control module (PCM), which can be made up of several ECUs carries out these checks in vehicles. For the study, the primary ECU was the BCM, and the red LED acted as the MIL. The red LED is ON if a CAN message fails the message authentication check, or an abnormal CAN message is received that is not within the range of expected CAN IDs. According to Smith (2016) all vehicles from 2015 and after are obligated to have some sort of black box, called an event data recorder (EDR), The EDR continually saves information, often only around 20 seconds at time. This data was initially saved in a vehicle’s airbag control module (ACM); however, modern cars disperse it among the vehicle’s ECUs. These boxes capture data from other ECUs and sensors and store them for recovery after a crash. The EDR stores the following information: Airbag deployment, Brake

status, Seat belt status, Steering angles, Throttle position and Vehicle speed. Each ECU (BCM, DCM, HCM and BSM) in the study acts as an Event Data Recording (EDR) to record information regarding the ECU communication including CAN ID, payload, origin of can packet, destination of can packet, diagnostic messages, error messages. For BCM is logged sensor status received, and the delay of receiving CAN packet from sender. [Table 2.3](#) below shows a sample of the format of the log files.

BCM.log	DCM.log	Fuzzing.log
2024-09-19 16:22:31,606 CAN ID:512 Data:bytearray(b'\x03f\xecA\xb7\xb6\xac\xe5') Origin:DCM Destination:BCM Status:Door is Locked Error:MAC verification successful Latency:606	2024-09-19 16:26:14,282 CAN ID: 512 Data: bytearray(b'\x03f\xecB\x96\x07\x80Y') Origin: DCM Destination: BCM Diagnostic Msg: Door is Locked Error:None	2024-09-19 15:13:00,251 CAN ID: 1319 Data: bytearray(b'Z\x9cA\xf2\xec\x03IR') Diagonistic message: Random Fuzzing

Table 2.3: Log Files Format 1

b) Security enhancement

- **CAN ID Filtering:** To resolve this the study used CAN ID based filtering by configuring the BCM to accept CAN Messages with specific CAN IDs (0x100,0x200,0x400), while ignoring others.
- **Message Authentication integration:** To improve the security of CAN communications, MAC (Message Authentication Code) was proposed by (Aliwa et al., (2021), Choi et al., (2021), Luo et al., (2022) and Parkinson et al., (2017)). The codes are applied to the messages to validate their integrity and validity upon reception while also blocking authorised alterations. Model-based strategy for testing and implementing message authentication approach. SHA-256 (Secure hash Algorithm) was used for message authentication which is a cryptographic function that produces a fixed 32-byte hash value

from an input of any length. Alongside HMAC was used as suggested by Aliwa et al., (2021)'s paper which used a pair of key distribution and pre-shared symmetric which ensures that the message sent has not been altered during the session. Aliwa et al., (2021) suggested the use of small MAC tags to make up for the overhead of sending CAN messages, so a small MAC tag of 3 bytes was used to make space for the MAC tag and timestamp to calculate latency. The MAC algorithm by Luo et. al. (2022) of using both sender and receiver must share the same key was utilised.

2. Threat Modelling

The primary attack vector identified for this penetration test involved the third Raspberry Pi acting as an attacker ECU. This attacker node could sniff CAN packets transmitted between the legitimate ECUs to inject manipulated packets into the network. Various attack types were conducted, including:

- **Random Fuzzing** – injecting random CAN messages to see if any illicit behaviours are triggered
- **Linear Fuzzing** – systematically sequentially sending altered CAN messages by incrementally changing parts of CAN messages to test boundary values
- **Brute Force Fuzzing** – Sending a high volume of CAN messages to bypass security
- **Mutation-Based Fuzzing** – Flipping bits in CAN messages to simulate unexpected behaviour.
- **Replay Fuzzing** – capturing legitimate CAN network traffic then replaying them back onto the can bus to see if they trigger authorised actions.

The goal of threat modelling is to prioritize the potential attack surfaces and focus on the areas with the highest risk or most significant impacts. Each of these tests' targets potential vulnerabilities in the CAN bus, with a focus on manipulating or bypassing the security mechanisms in place.

D. Challenges Encountered:

Several challenges arose throughout the study that impacted the progress of and efficiency of the research. One of the primary issues was related to test-bed configuration. Initially, the CAN

Adapter was not functioning correctly; it was not activated for CAN injection which limited its capabilities to only sniffing messages from CAN bus. This restriction hampered the ability to perform the intended penetrating testing using a Linux laptop. Additionally, there were complications with circuit connection. The initial set up has incorrect wiring which lead to future delays in the experimentation process. After troubleshooting these issues and consulting online resources, including YouTube tutorials on RS485 CAN hat use the experiment, a decision was made to utilize another Raspberry Pi as the attacker ECU to conduct fuzz testing instead of relying on a CAN adapter. This adjustment allowed for a more streamlined and efficient set up. These setbacks however, resulted in considerable delays in exploring and improving the security mitigation techniques employed by the study.

[Section VI](#) will continue the Pen-Testing discussion going over the Vulnerability Assessments and Exploitation and Report of the pen-testing process.

V. Results

This section presents the key findings from the penetration testing experiments and other relevant data gathered.

A. ECU Communication

[Fig 2.2](#) displays 9 CAN packets sent to Receiver ECU whereas [Fig 2.3](#) shows the CAN packets received. In reference to the literature review section regarding CAN sniffing, [table 2.4](#) shows all 9 CAN messages eavesdropped by Attacker ECU from the CAN bus using Wireshark installed on the Attacker ECU. The CAN messages data however is in hexadecimal format.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Lindanafunu@raspberrypi:~/Desktop/Final-Project $ /usr/bin/python /home/Lindanafunu/Desktop/Final-Project
Sending CAN messages for 10 seconds...
Message with ID 1542, with Data bytearray(b'ECU3') sent on socketcan channel 'can0'
Message with ID 1466, with Data bytearray(b'ECU3') sent on socketcan channel 'can0'
Message with ID 928, with Data bytearray(b'ECU3') sent on socketcan channel 'can0'
Message with ID 675, with Data bytearray(b'ECU3') sent on socketcan channel 'can0'
Message with ID 2002, with Data bytearray(b'ECU3') sent on socketcan channel 'can0'
Message with ID 1385, with Data bytearray(b'ECU3') sent on socketcan channel 'can0'
Message with ID 758, with Data bytearray(b'ECU3') sent on socketcan channel 'can0'
Message with ID 448, with Data bytearray(b'ECU3') sent on socketcan channel 'can0'
Message with ID 881, with Data bytearray(b'ECU3') sent on socketcan channel 'can0'
Message with ID 1345, with Data bytearray(b'ECU3') sent on socketcan channel 'can0'
Lindanafunu@raspberrypi:~/Desktop/Final-Project $ import send.png
```

Fig.2. 2 CAN Packets Sender send 1

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Lindanafunu@raspberrypi:~/Desktop/Final-Project $ /bin/python /home/Lindanafunu/Desktop/Final-Project
Listening on socketcan channel 'can0' with filter [{'can_id': 256, 'can_mask': 1792}] ...
Received message: ID=1542, Data=bytearray(b'ECU3')
Received message: ID=1466, Data=bytearray(b'ECU3')
Received message: ID=928, Data=bytearray(b'ECU3')
Received message: ID=675, Data=bytearray(b'ECU3')
Received message: ID=2002, Data=bytearray(b'ECU3')
Received message: ID=1385, Data=bytearray(b'ECU3')
Received message: ID=758, Data=bytearray(b'ECU3')
Received message: ID=448, Data=bytearray(b'ECU3')
Received message: ID=881, Data=bytearray(b'ECU3')
Received message: ID=1345, Data=bytearray(b'ECU3')
No messages recieved within time period.
No messages recieved within time period.
No messages recieved within time period.
No messages recieved within time period.
^KeyboardInterrupt detected, exiting gracefully.
Cleanup complete. Exiting program.
Lindanafunu@raspberrypi:~/Desktop/Final-Project $ import rc.png
Lindanafunu@raspberrypi:~/Desktop/Final-Project $ import rc.png
```

Fig 2.3: CAN Packets received 1

No.	Time	Source	Destination	Protocol	Length	Info
1	0			CAN	16	ID: 1542 (0x606), Length: 4
2	1.000447259			CAN	16	ID: 1466 (0x5ba), Length: 4
3	2.001148027			CAN	16	ID: 928 (0x3a0), Length: 4
4	3.001510309			CAN	16	ID: 675 (0x2a3), Length: 4
5	4.001646344			CAN	16	ID: 2002 (0x7d2), Length: 4
6	5.002573329			CAN	16	ID: 1385 (0x569), Length: 4

7	6.002735777			CAN	16	ID: 758 (0x2f6), Length: 4
8	7.003102882			CAN	16	ID: 440 (0x1b8), Length: 4
9	8.003426573			CAN	16	ID: 801 (0x321), Length: 4
10	9.004071829			CAN	16	ID: 1345 (0x541), Length: 4

Table 2.4: Captured CAN Frames 1

[Appendix 1](#) demonstrates that ECU1 and ECU3 send CAN messages with their labels in the data section of the CAN message to recipient ECU. Recipient ECU shows CAN messages from both ECU received. This section highlighted the broadcast nature of CAN protocol.

B. Latency Comparison

1. Latency Distribution

[Fig 2.5](#) and [Fig 2.6](#) demonstrate the latency over time for ECU communication with and Without message authentication. [Table 2.5](#) shows latency distribution for the sessions with MAC verification compared to without. For the session with message authentication 835 CAN packets were sent to BCM with the mean latency is 509.920958 milli seconds and standard deviation of 286.945146. For session Without message authentication, 841 CAN packets were delivered to BCM, with mean latency is 506.334126 milli seconds with standard deviation of 284.363393 milliseconds

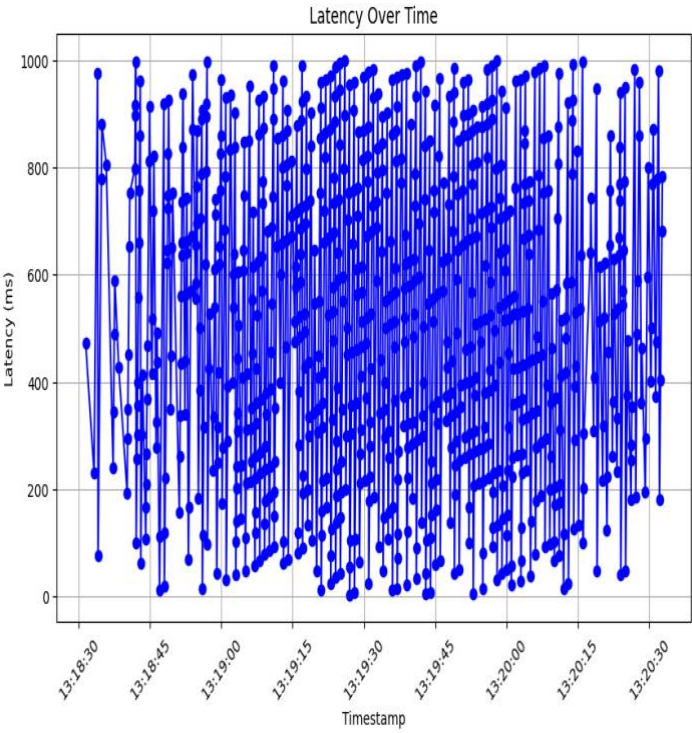


Fig 2.5: Without MAC Verification 1

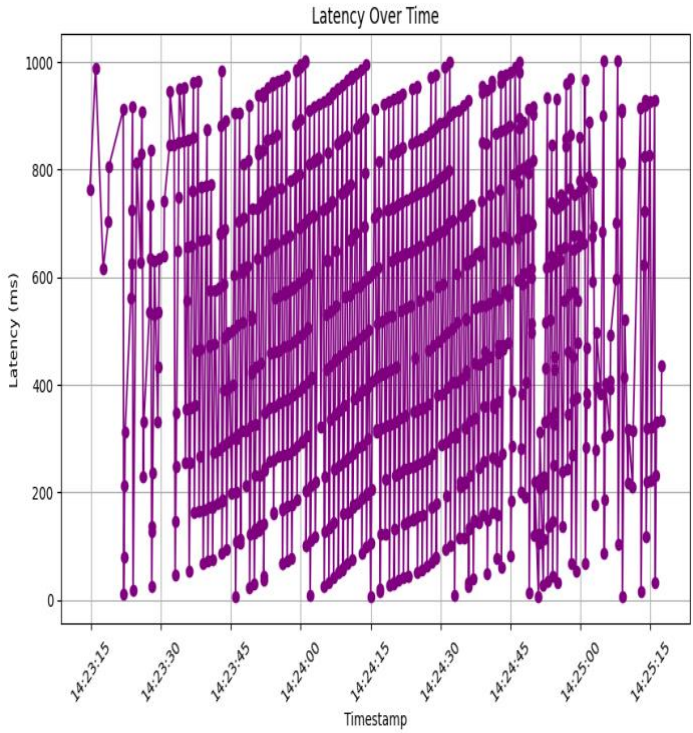


Fig 2.6: With MAC Verification 1

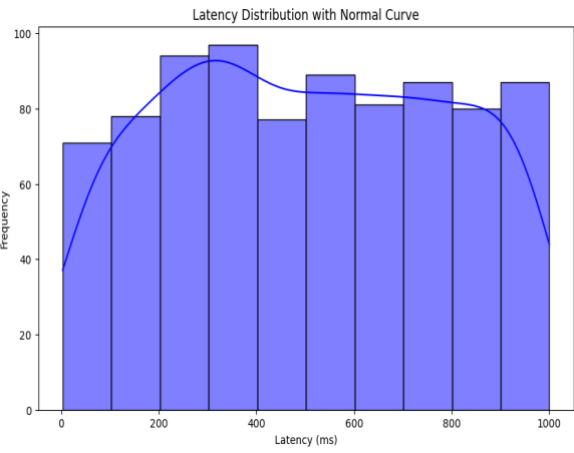


Fig 2.7: Without MAC Verification 1

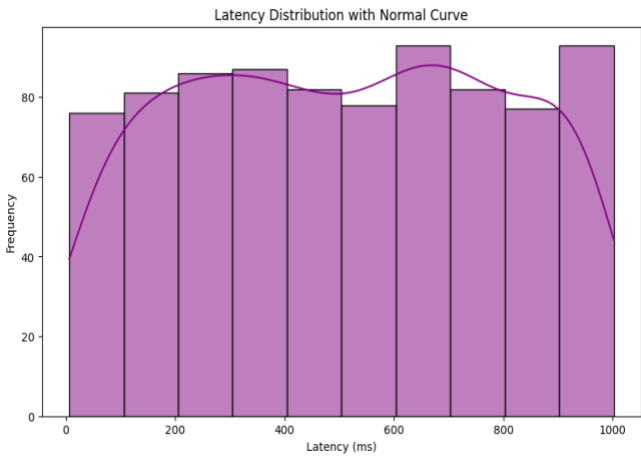


Fig 2.8: With MAC Verification 1

Latency	Distribution (milli secs)	
	No MAC Verification	MAC Verification
Count	841.000000	835.000000

Mean	506.334126	509.920958
Std	284.363393	286.945146
Min	3.000000	6.000000
25%	267.000000	261.500000
50%	509.000000	514.000000
75%	752.000000	752.000000
Max	1000.000000	1002.000000

Table 2.5: Latency Distribution 1

2. CAN Messages Distribution

[Table 2.6](#) contains more Belt Status simulations, because of the CAN ID arbitration priority, CAN messages with lower CAN IDs have priority on the CAN bus. Belt statuses CAN messages have CAN ID of 0x100 while Headlights status have 0x200 and Door has 0x400. The door button was pressed 4 times hence the number of CAN messages set where few. All 835 CAN messages passed the MAC verification check as indicated in [table 2.7](#) All 841 CAN messages were classified as legitimate.

Sensor Status	Count	
	No MAC verification	MAC verification
Belt is OFF	213	214
Belt is ON	213	214
Headlights ON	206	201
Headlights OFF	206	201
Door is Locked	2	3
Door is Unlocked	1	2

Table 2.6.: Sensor Status Distribution 1

Error	No MAC Verification Count	MAC Verification Count
Mac Verification Successful		835
None	841	

Table 2.7: Error Messages Distribution 1

This section demonstrate the delay intoruced by the message authentication technique to ECU communication.

C. Random Fuzzing

1. Latency Distribution:

[Fig 2.8](#) and [Fig 2.9](#) illustrate the difference in the latency over time for sessions with message authentication and without. When MAC verification fails a delay was zero is reported displayed in [fig 2.9](#). Without message authentication, negative latencies are captured demonstrated in [fig 2.8](#) and [table 2.8](#)

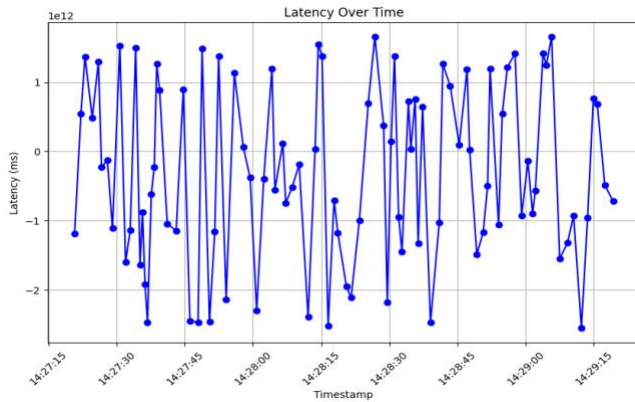


Fig 2.8: Without MAC Verification 1

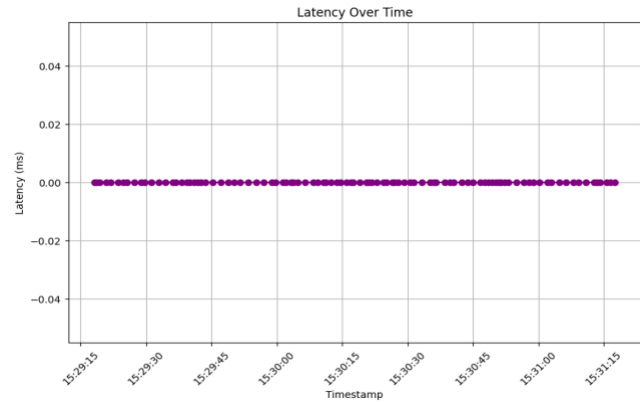


Fig 2.8: Without MAC Verification 1 1

Latency	Distribution (milli secs)	
	No MAC Verification	MAC Verification
Count	9.500000e+01	99
Mean	-3.094250e+11	0
Std	1.264813e+12	0
Min	-2.547441e+12	0
25%	-1.161721e+12	0
50%	-3.915782e+11	0
75%	8.227030e+11	0
Max	1.657010e+12	0

Table 2.8: Latency Distribution 1

2. CAN Messages Distribution

Of the 95 CAN messages sent for no message authentication, no sensor messages were captured to manipulate the output devices shown in [table 2.9](#). The same behaviour was noticed when message authentication was performed. Without message authentication, all CAN messages were classified as abnormal however, when message authentication was performed all CAN messages failed message authentication as indicated in [table 3.0](#).

Sensor Status	Count	
	No MAC verification	MAC verification
None	95	99

Table 2.9: Sensor Status Distribution 1

Error	No MAC Verification Count	MAC Verification Count
Abnormal Message received	95	-
MAC verification failed	-	99

Table 3.0: Error Messages Distribution 1Linear Fuzzing

The results showcase the ECU behaviour based of the Random fuzzing when message authentication is enabled compared to when it is disabled.

D. Linear Fuzzing

1. Latency Distribution:

[Fig 3.0](#) and [fig 3.1](#) demonstrate the latency over time for sessions that used message authentication against those that did not. When MAC verification fails a latency of zero is recorded as displayed in [fig 3.1](#) and [table 3.1](#). Without message authentication, latency increases linearly shown in [fig 3.0](#) with mean $1.727185e+12$.

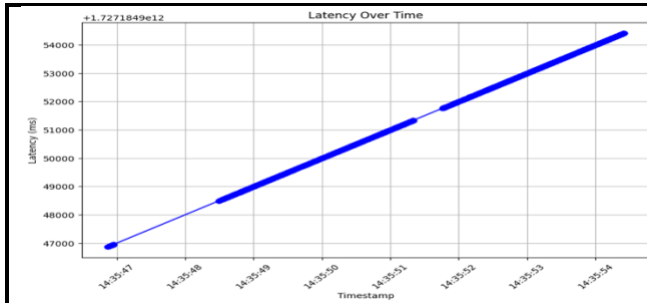


Fig 3.0: Without MAC Verification 1

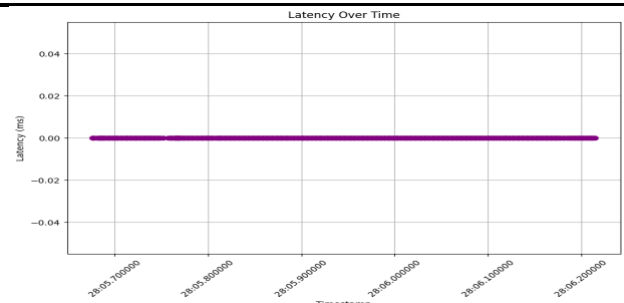


Fig 3.1: With MAC Verification 1

Latency	Distribution (milli secs)	
	No MAC Verification	MAC Verification
Count	3.923000e+03	31534.0
Mean	1.727185e+12	0
Std	1.896720e+03	0
Min	1.727185e+12	0
25%	1.727185e+12	0
50%	1.727185e+12	0
75%	1.727185e+12	0
Max	1.727185e+12	0

Table 3.1: Latency Distribution 1

2. CAN Messages Distribution

Of the 3924 CAN messages sent for no message authentication no sensor messages were captured to manipulate the output devices shown in [table 3.2](#). The same behaviour was observed when 3154 CAN messages were transmitted when message authentication was enabled. Without message authentication, 99.8% of the CAN messages were classified as abnormal and 0.15% were classified as legitimate shown in [table 3.3](#). When message authentication was enabled all 31534 CAN messages failed message authentication shown in [table 3.2](#).

Sensor Status	Count	
	No MAC verification	MAC verification
None	3923	3154

Table 3.2: Sensor Status Distribution 1

Error	No MAC Verification Count	MAC Verification Count
Abnormal Message received	3917	-
MAC verification failed	-	31534
None	6	

Table 3.3: Error Messages Distribution 1

The results showcase the ECU behaviour based of the Linear fuzzing when message authentication is enabled compared to when it is disabled.

E. Brute Force Fuzzing

1. Latency Distribution:

[Fig 3.2](#) and [fig 3.3](#) show the latency over time for sessions when message authentication was performed compared to when it was not enabled. When MAC verification fails a latency of zero is recorded highlighted in [fig 3.3](#) and [table 3. 4](#).

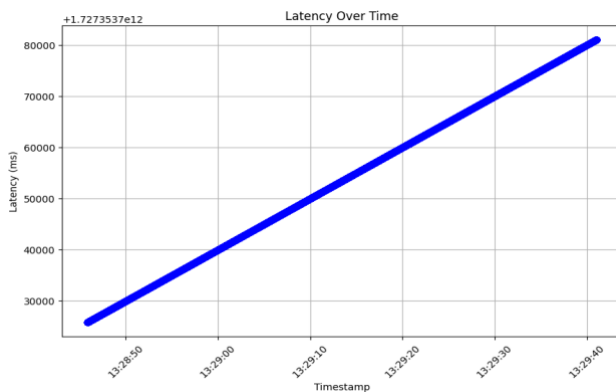


Fig 3.2: Without MAC Verification 1

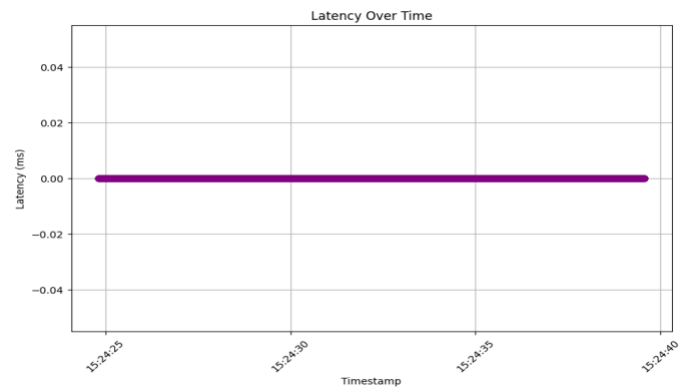


Fig 3.3: With MAC Verification 1

Latency	Distribution (milli secs)	
	No MAC Verification	MAC Verification
Count	3.320800e+04	9715.0
Mean	1.727354e+12	0
Std	1.597150e+04	0
Min	1.727354e+12	0

25%	1.727354e+12	0
50%	1.727354e+12	0
75%	1.727354e+12	0
Max	1.727354e+12	0

Table 3.4: Latency Distribution 1

2. CAN Messages Distribution

[Table 3.5](#) shows that no sensor messages were recorded to alter the output devices among the 9715 CAN messages transmitted for the session with message authentication was enabled, the same behaviour is shown when message authentication was not enabled. Without message authentication, all 33208 CAN messages were classified as legitimate however, when message authentication was enabled all CAN messages failed message authentication shown in [table 3.6](#).

Sensor Status	Count	
	No MAC verification	MAC verification
None	33208	9715

Table 3.5: Sensor Status Distribution 1

Error	No MAC Verification Count	MAC Verification Count
MAC verification failed		9715
None	33208	

Table 3.6: Error Messages Distribution 1

The results showcase the ECU behaviour based of the Brute Force fuzzing when message authentication is enabled compared to when it is disabled.

F. Mutated Based Fuzzing

1. Latency Distribution:

[Fig 3.4](#) and [find 3.5](#) show the latency over time for sessions when message authentication was enabled compared to when it was not. When MAC verification fails a latency of zero is recorded highlighted in [table 3.7](#). [Fig 3.3](#) shows the negative latencies captures in the session message authentication not enabled.

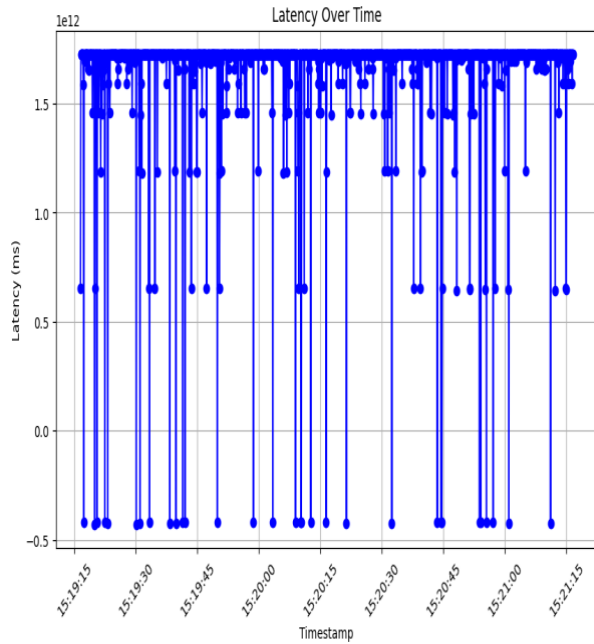


Fig 3.4: Without MAC Verification 1

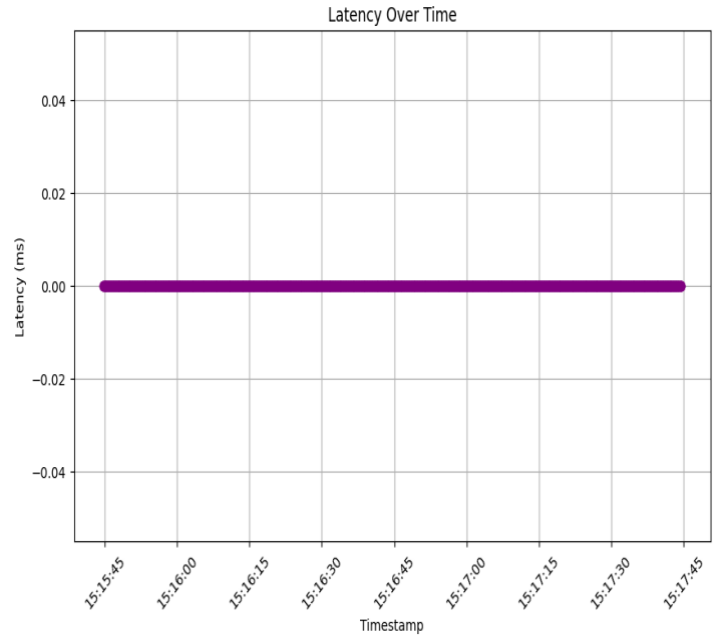


Fig 3.5: With MAC Verification 1

Latency	Distribution (milli secs)	
	No MAC Verification	MAC Verification
Count	1.191000e+03	1191.0
Mean	1.626785e+12	0
Std	3.731259e+11	0
Min	-4.286929e+11	0
25%	1.727188e+12	0
50%	1.727188e+12	0
75%	1.727188e+12	0
Max	1.727188e+12	0

Table 3.7: Latency Distribution 1

2. CAN Messages Distribution

When message authentication was performed all 1191 CAN messages failed message authentication as shown in [table 3.9](#). and they did not trigger normal sensor status. Without message authentication, 1190 messages were able to manipulate ECU behaviour triggering output

devices except 1 message as shown in [table 3.8](#). and they were classified as normal as shown in [table 3.9](#)

Sensor Status	Count	
	No MAC verification	MAC verification
Belt is OFF	414	
Belt is ON	15	
Headlights ON	370	
Headlights OFF	10	
Door is Locked	18	
Door is Unlocked	363	
None	1	1191

Table 3.8: Sensor Status Distribution 1

Error	No MAC Verification Count	MAC Verification Count
MAC verification failed		1191
None	1191	

Table 3.9: Error Messages Distribution 1

The results showcase the ECU behaviour based of the Mutaed based fuzzing when message authentication is enabled comapred to when it is disabled.

G. Replay Fuzzing

This section is depicting a scenario replay attack capturing CAN messages from DCM, BSM and HCM during 30 second session. The screen shots of log files in the last section are for DCM only to demonstrate how packets are captured in the example presented below, because for the session for capturing packets from multiple modules was too lengthy to display. [Fig 3.8](#) shows CAN packets from DCM send over CAN bus, [fig 3.9](#) shows the CAN packets captured, and the replayed packets finally [fig 4.0](#) shows all the received packets at the BCM ECU from both Attacker ECU and DCM.

1. Latency Distribution

[Fig 3.6](#) and [fig 3.7](#) graphs show the latency over time for sessions when message authentication was performed compared to when it was not. Both graphs indicate the beginning of the sessions with small delays then a sudden peak is recorded after timestamps 15:45:50 and 15:41:45 concurrently each graph captures in the session Without message authentication,

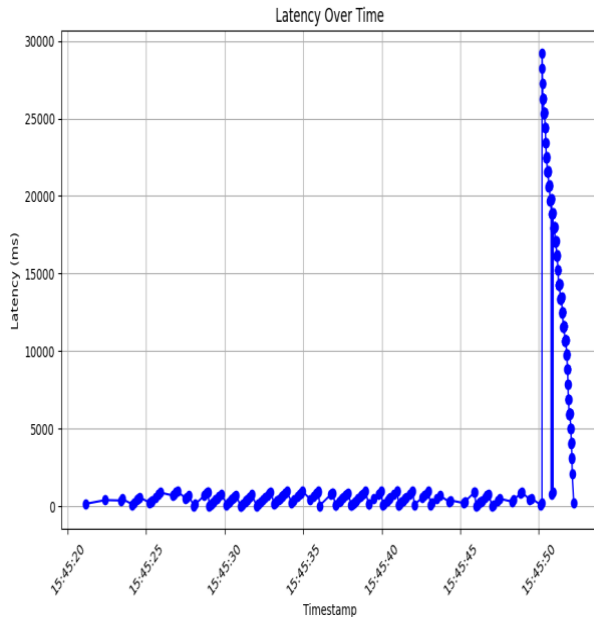


Fig 3.6: Without MAC Verification 1

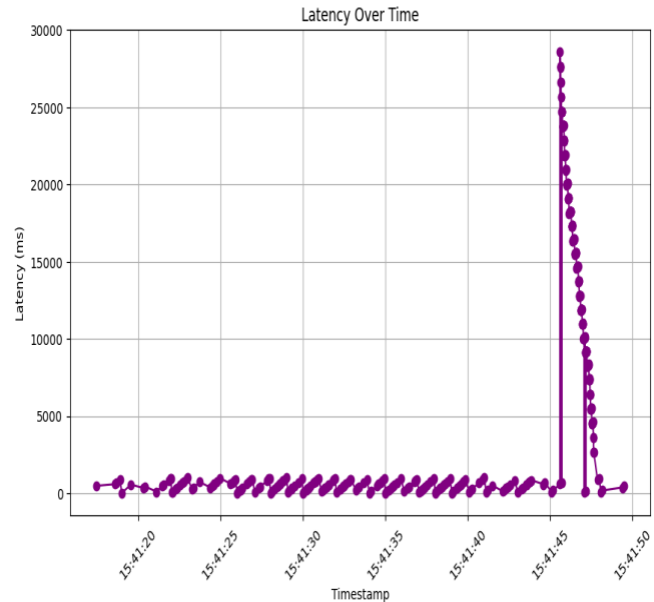


Fig 3.7: With MAC Verification 1

Latency	Distribution (milli secs)	
	No MAC Verification	MAC Verification
Count	368.000000	404.000000
Mean	7988.741848	7379.287129
Std	8810.118657	8315.992278
Min	7.0000000	3.000000
25%	479.000000	474.750000
50%	983.500000	979.000000
75%	16127.000000	14643.750000
Max	29203.000000	28591.000000

Table 4.0: Latency Distribution 1

2. CAN Messages Distribution

Without message authentication, all 368 messages manipulated ECU behaviour triggering output devices as shown in [table 4.1](#). and they were classified as legitimate as shown in [table 4.2](#). All 404 CAN messages sent passed MAC verification shown in [table 4.2](#) and output devices are triggered shown in [table 4.1](#).

Sensor Status	Count	
	No MAC verification	MAC verification
Belt is OFF	75	95
Belt is ON	74	95
Headlights ON	202	102
Headlights OFF	102	102
Door is Locked	8	6
Door is Unlocked	6	4

Table 4.1: Sensor Status Distribution 1

Error	No MAC Verification Count	MAC Verification Count
MAC verification passed		404
None	368	

Table 4.2: Error Messages Distribution 1

```

DCM_Replay2.log
2024-09-19 16:18:34,506 Created a socket
2024-09-19 16:18:34,550 CAN ID: 512
Data: bytearray(b'\x03f\xec\xca\x91\xa2\xf4')
Origin: DCM
Destination: BCM
Diagnostic Msg: Door is Locked
Error:None

2024-09-19 16:18:41,207 CAN ID: 512
Data: bytearray(b'\x02f\xec\xd1\x82+\x9c')
Origin: DCM
Destination: BCM
Diagnostic Msg: Door is Unlocked
Error:None

2024-09-19 16:18:42,272 CAN ID: 512
Data: bytearray(b'\x03f\xec\xd2+\x92 ')
Origin: DCM
Destination: BCM
Diagnostic Msg: Door is Locked
Error:None

2024-09-19 16:18:55,867 KeyboardInterrupt detected, stopping status transmission.

```

Fig 3.8: DCM packets send to BCM 1

```

Replay_Fuzzing_Door_mac.log

2024-09-24 15:50:37,242 Created a socket
2024-09-24 15:50:40,627 CAN ID: 512
Data: bytearray(b'\x03f\xf2\xd1\xc0g\xa3\x9f')
Diagnostic message: None
Error: None

2024-09-24 15:50:42,761 CAN ID: 512
Data: bytearray(b'\x02f\xf2\xd1\xc2\xdbqb')
Diagnostic message: None
Error: None

2024-09-24 15:50:45,454 CAN ID: 512
Data: bytearray(b'\x03f\xf2\xd1\xc5]\x0c\xfe')
Diagnostic message: None
Error: None

2024-09-24 15:50:48,214 CAN ID: 512
Data: bytearray(b'\x02f\xf2\xd1\xc8\x03E\xf6')
Diagnostic message: None
Error: None

2024-09-24 15:50:51,148 CAN ID: 512
Data: bytearray(b'\x03f\xf2\xd1\xcb\x13\xc5\xba')
Diagnostic message: None
Error: None

2024-09-24 15:51:07,297 CAN ID: 512
Data: bytearray(b'\x03f\xf2\xd1\xc0g\xa3\x9f')
Diagnostic message: None
Error: None

2024-09-24 15:51:07,308 CAN ID: 512
Data: bytearray(b'\x02f\xf2\xd1\xc2\xdbqb')
Diagnostic message: None
Error: None

2024-09-24 15:51:07,319 CAN ID: 512
Data: bytearray(b'\x03f\xf2\xd1\xc5]\x0c\xfe')
Diagnostic message: None
Error: None

```

Fig 3.9: Captured and Replayed packets 1

```

BCM_Door_Replay_mac.log

2024-09-24 15:50:40,626 CAN ID:512
Data:bytearray(b'\x03f\xf2\xd1\xc0g\xa3\x9f')
Origin:DCM
Destination:BCM
Status:Door is Locked
Error:MAC verification successful
Latency:626
2024-09-24 15:50:42,759 CAN ID:512
Data:bytearray(b'\x02f\xf2\xd1\xc2\xdbqb')
Origin:DCM
Destination:BCM
Status:Door is Unlocked
Error:MAC verification successful
Latency:759
2024-09-24 15:50:45,453 CAN ID:512
Data:bytearray(b'\x03f\xf2\xd1\xc5]\x0c\xfe')
Origin:DCM
Destination:BCM
Status:Door is Locked
Error:MAC verification successful
Latency:453
2024-09-24 15:50:48,213 CAN ID:512
Data:bytearray(b'\x02f\xf2\xd1\xc8\x03E\xf6')
Origin:DCM
Destination:BCM
Status:Door is Unlocked
Error:MAC verification successful
Latency:213
2024-09-24 15:50:51,146 CAN ID:512
Data:bytearray(b'\x03f\xf2\xd1\xcb\x13\xc5\xba')
Origin:DCM
Destination:BCM
Status:Door is Locked
Error:MAC verification successful
Latency:146
2024-09-24 15:51:07,286 CAN ID:512
Data:bytearray(b'\x03f\xf2\xd1\xc0g\xa3\x9f')
Origin:DCM
Destination:BCM
Status:Door is Locked
Error:MAC verification successful
Latency:27286
2024-09-24 15:51:07,297 CAN ID:512
Data:bytearray(b'\x02f\xf2\xd1\xc2\xdbqb')
Origin:DCM
Destination:BCM
Status:Door is Unlocked
Error:MAC verification successful
Latency:25297
2024-09-24 15:51:07,308 CAN ID:512
Data:bytearray(b'\x03f\xf2\xd1\xc5]\x0c\xfe')
Origin:DCM
Destination:BCM
Status:Door is Locked
Error:MAC verification successful
Latency:22308
2024-09-24 15:51:07,319 CAN ID:512
Data:bytearray(b'\x02f\xf2\xd1\xc8\x03E\xf6')
Origin:DCM
Destination:BCM
Status:Door is Unlocked
Error:MAC verification successful
Latency:19319
2024-09-24 15:51:07,330 CAN ID:512
Data:bytearray(b'\x03f\xf2\xd1\xcb\x13\xc5\xba')

```

Fig 4.0: BCM log for received CAN packet 1

VI. Discussion

This section covers the results discovered from the penetration testing experiments and addresses them in the Vulnerability Assessment, Exploitation and Report sections.

A. Vulnerability Assessment

The major problem with CAN protocols it was designed with no security protocols in place so there are a lot of loopholes attackers can exploit to manipulate a vehicle. The vulnerability assessment section focused on identifying weaknesses in the CAN bus communication protocol and implementation of security measures.

1. Identified weakness:

Lack of encryption: The CAN bus protocol lacks encryption, allowing messages to be intercepted and manipulated. [Fig 2.2](#) in [section V](#) shows CAN messages sent from Sender ECU, [Fig 2.3](#) shows all the CAN messages received by the Receiver ECU. The captured CAN packets on the CAN network by Attacker ECU are displayed in [table 2.4](#). This illustrates how an attacker can capture packets using tools like Wireshark. These captured packets reveal unencrypted data, including CAN ID and payloads of all the CAN packets send from the Sender ECU over the 30 second session. This information can be used to inject malicious packets into the network by an attacker. This is an example of CAN sniffing where an attacker can passively monitor, and capturer messages transmitted over the CAN without interfering or altering the data transmitted. Parkinson, (2017) suggested implementing encryption as a mitigation strategy however, encryption mechanisms add a complexity to the CAN network. To implement encryption, an encryption key should be generated that should be safely shared amongst the legitimate ECUs on the network. Despite the importance of encryption, this study did not implement encryption due to the 8-byte payload limit for Standard CAN messages, which complicates integrating encryption algorithms suggested by Jadoon et al., (2018) since they require more bytes.

No message filtering: One major weakness of CAN protocol is that normally does not filter or block malformed or unexpected messages. [Appendix 1](#) displays three images, the first image shows CAN messages send from Attacker ECU, second images show CAN messages send from Sender ECU and the third image shows all CAN packets send from Attacker ECU and Sender

ECU. The Receiver ECU received and processed CAN packets as legitimate. This shows how easily an attacker can inject malicious CAN packets on the CAN thereby manipulating ECU behaviour which endangers driver's safety. Implementing a CAN filter that limits the accepted CAN IDs to [0x100,0x200 and 0x400] was effective in identifying abnormal messages during Random Fuzzing as shown in [table 3.0](#), Linear fuzzing tests refer to table x, Brute Force Fuzzing table x

Message Authentication: Due to the standard CAN message size limit of 8 bytes, this reduced the space available to store the MAC tag and timestamp. The study added a MAC tag of 3 bytes and a 4 bytes timestamp to CAN payload to calculate the delay of sending packets and to provide message authentication. Message authentication was carried out for every ECU communication session on the BCM ECU, shown in [section VI. Latency Comparison](#) section shows the ECU behaviour for a 120 second session. 835 CAN messages were sent to BCM after message authentication has a mean latency was approximately 509.920958 milliseconds as displayed in section in [table 2.5](#) Another 120 second session was carried out sending 841 CAN packets sent BCM, with message authentication not enabled to the BCM ECU, the mean delay was approximately 506.334126 milli seconds. The mean delay difference this is approximately 3.587 millisecond, which means that there is not much overhead in adding small MAC tags. Depending on how real-time the manufacturer wants to design the ECU using small MAC tags can help cap the overhead in ECU communication. Another 60-session minute session was done with message authentication enabled every 5 second interval this resulted in some messages being processed as legitimate outside the 5 second window. For Brute Force Fuzzing session, the CAN packets they were not able to trigger output device sensors but, for Mutated Based Fuzzing output devices where triggered. This reflected a loophole in the CAN network setup whereby some messages could easily slip in to manipulate ECU. To mitigate this vulnerability the study enabled message authentication throughout the ECU communication sessions. The results of the fuzz tests show that, while MAC verification was effective in most circumstances it was however, insufficient to prevent all assaults, notably replay attacks. The evidence is described in depth in the Fuzzing Results section below.

2. Fuzzing Results

a) *Random Fuzzing*

Random CAN messages were generated with CAN ID range of [0x00-0x7FF] with CAN payload ranging between [0X00-0XFF] of a maximum length of 8 bytes. Based on [Random Fuzzing](#) results section the fuzz test session was done over a 120 second period, the system appropriately classified all 95 CAN messages injected onto the CAN bus from the attacker as ‘Abnormal’ as shown in [table 2.8](#) for the session with message authentication not enabled. The BCM also computed a negative delay as shown in the Latency Over Time graph in [fig 2.8](#), negative delay might suggest that the system received messages in an unexpected order or that timestamps were tampered with during message injection. This is a strong indication of an intrusion attempt. For the session with message authentication enabled, all 99 CAN messages sent to BCM, failed MAC verification as shown in [table 2.8](#). The latency was recorded as zero since for CAN packets that fail message authentication this shown in graph in [fig 2.9](#) for the session.

All the CAN messages triggered the MIL. This aligns with the design described by Smith (2016) when a vehicle encounters a malfunction, it records information related to that fault and activates the engine warning light, commonly known as the Malfunction Indicator Illumination (MIL). For this study when an abnormal message unauthenticated was encountered details regarding the CAN packet is logged including, timestamp of when it occurred, CAN ID, payload, error message, the red LED is turned on as a warning signal. Message filtering and message authentication were able to block the random CAN messages from manipulating ECU behaviour and detected abnormal behaviour in CAN protocol.

b) *Linear Messages*

Sequential CAN messages were injected starting at CAN ID in range [0x000 – 0x7FF] with start payload of 1 byte to 8 bytes onto the CAN bus. Based on [Linear Fuzzing](#) results section V the fuzz test session was done over a 120 second period. The system appropriately classified 99.8% of the CAN messages as abnormal and 0.15% were classified as legitimate shown in [table 3.3](#) for the session with message authentication does not enable as shown in [table 3.2](#). The small 0.15% CAN messages classified as normal showcases that Linear if linear fuzzing could result in some CAN packets manipulating ECU behaviour, indication that message filtering is not a strong barrier so

such an attack. However for the session with message authentication enabled, all 31534 CAN messages sent to BCM ECU failed MAC verification check as shown in [table 3.2](#). The latency was recorded as zero since for CAN packets that fail message authentication, this shown in graph in [fig 2.9](#). The abnormal and unauthenticated CAN messages triggered the red LED to turn on indicating a malfunction or potential intruder. This shows that message authentication was an effective strategy to detect abnormal behaviour in CAN protocol.

c) Brute Force Fuzzing

Exhaustively sent CAN messages over the CAN bus to trigger ECU response. The initial brute force efforts were constructed as a fuzz test with a nested loop that iterates over all conceivable 11-bit CAN IDs and 8-byte payloads. The test was taking a long time to send CAN messages for a 120 second interval it was still generating CAN packets with CAN ID 0x00. After optimising the brute force approach to target legitimate CAN IDs with random payloads, the system classified all messages received by the BCM as legitimate as shown in [table 3.6](#). However, they did not trigger the output sensors. The Brute Force Attack could not trigger any output sensors since the BCM was designed in such a way that certain CAN data commands can trigger output sensors; in situations in which the CAN IDs are used to manipulate ECU behaviour a different outcome could have been observed, this shows that Brute Force Fuzzing can manipulate ECU behaviour.

For session with message authentication enabled all 9715 CAN messages sent could also not manipulate the output devices as shown in [table 3.5](#). The CAN messages CAN messages failed message authentication shown in [table 3.6](#). This triggered the red LED to turn on which is acting at the MIL indicating a malfunction or potential intruder. This shows how message authentication was able to detect the abnormal activity in CAN network, hence protecting ECU.

s to manipulate ECU behaviour.

d) Mutated Based Fuzzing

This was a variation of white box testing, since legitimate CAN messages are used but with certain bits flipped to manipulate BCM behaviour. When message authentication is not enabled, 1190 messages were classified as legitimate as shown in [table 3.9](#). and they manipulated ECU behaviour triggering output devices except 1 message as shown in [table 3.8](#). This shows that the message filter is not strong enough security measure to protect CAN protocol. [Fig 3.3](#) shows the negative

latencies captures in the session which is an indication of an intruder. For the session message authentication enabled all 1191 CAN messages failed message authentication as shown in [table 3.9](#), which triggered the MIL indicating a potential malfunction or intruder. When MAC verification fails a latency of zero is recorded highlighted in [table 3.7](#). Message authentication was able to detect the abnormal activity.

e) Replay Fuzzing

Replay attacks proved to be the most successfully penetration technique. The scenario represented in results section under [Replay Fuzzing](#), the replay attack captured CAN messages from DCM, BSM and HCM for 30 second session. With message authentication not enabled, all 368 messages were able to manipulate ECU behaviour triggering output devices as shown in [table 4.1](#), and they were classified as legitimated as shown in [table 4.2](#). All 404 CAN messages received by BCM passed MAC verification shown in [table 4.2](#) in the session with MAC verification enabled and the CAN messaged triggered output devices shown in [table 4.1](#). The same behaviour was noticed for a 30 second session in [Fig 3.8](#) showing CAN packets from DCM send over CAN bus. [fig 3.9](#) shows the CAN packets captured, and the replayed packets finally [fig 4.0](#) shows all the received packets at the BCM ECU from both Attacker ECU and DCM. By capturing legitimate CAN messages and replaying them on the bus the attacker node was able to pass MAC verification. All together these findings show that message authentication is not effective in blocking and detecting Replay Attacks. As pointed out by Parkinson et al., (2017), attackers can sniff CAN packets from a vehicle CAN bus by simply using an OBD-II and replay them onto the network manipulating ECU behaviour, this endangers driver's safety.

B. Exploitation

The exploitation phase demonstrates how vulnerabilities in the CAN protocol can be manipulated by an attacker to compromise vehicle systems. Below, is an analysis of each attack method based on the results of the fuzz testing and the responses of system under different security conditions.

Random Fuzzing and Linear fuzzing: These fuzzing techniques focused on injecting arbitrary CAN messages and sequential CAN messages, respectively. The system successfully classified the CAN messages as abnormal, and blocked them, due to message filtering and message authentication mechanism as shown in [table 2.8](#) and in [table 3.2](#). This forms part of a defensive

strategy as no actual exploitation was achieved due to message filtering and detection mechanism. While these attacks did not directly manipulate easy use to trigger vehicle systems, they exposed a key vulnerability: the CAN bus does not inherently filter out malicious traffic. However, the mere fact that these messages could be injected highlights potential denial of service attacks (DoS). If the message classification system were overwhelmed by high volume of random or linear messages, such attack would reduce the availability of critical issue functions.

Brute Force Fuzzing: The attack intended to exhaustively send all CAN IDs and payloads within the given range, attempting to manipulate easy use to trigger vehicle components. Brute force attack focused generating CAN messages with legitimate CAN IDs but with varying payloads data. Without MAC verification these messages were treated as legitimate, revealing a weakness in the systems' inability to validate message integrity. However, when MAC verification was enabled, every message failed authentication as shown in [table 3.6](#). This proves that the MAC- based authentication mechanism, can effectively detect and block brute force attempts, which try to overwhelm the system by exploiting the lack of data validation mechanisms. The inability of the brute force to trigger sensor outputs when MAC verification is in place shows that well-implemented message authentication system can be strong deterrent.

Mutated Based Fuzzing: This attack proved to be more sophisticated than brute force, as it used legitimate messages but flipped certain messages bits to alter behaviour. When MAC verification was disabled, these altered messages successfully manipulated sensors, showing that flipping bits in CAN messages can result in authorised behaviour refer to [table 3.8](#). However, with MAC verification enabled, all messages were blocked. This shows that while this attack is more sophisticated approach, the message authentication mechanism is still effective in detecting these message alterations

Replay Attacks: This was the most successful form of exploitation. By capturing legitimate CAN messages and replaying them later, the attacker bypassed MAC verification and successfully manipulated vehicle functions such as controlling sensors and alteration ECU states. This occurred because the MAC tag was based on statistic parameters that the attacker could reuse it without triggering the MIL (red LED). The latency spikes observed in the Latency Over Time in [Fig 3.6](#) and [fig 3.7](#) suggest delays between message capture and replay, further highlighting this attack's

potential for disruption. Replay attacks emphasise the need for more dynamic mechanism to ensure that previously valid messages are not reused to exploit vehicle systems.

C. Reporting

The findings from the penetration testing underscore critical vulnerabilities in the CAN bus communication system and highlight potential risks to vehicle safety, integrity, and availability. The following section summarises the key vulnerabilities identified and offers a thorough analysis of their implications as well as recommendations and future work.

Lack of Encryption: One of the most critical findings from this study is the lack of encryption within the CAN protocol, CAN messages are transmitted in plaintext, which allows attackers to intercept and manipulate data without needing sophisticated technique. This fundamental weakness exposes vehicle networks to a wide range of attacks, including data injection, manipulation, and passive eavesdropping. Through sniffing, it became evident that an attacker could passively capture sensitive data, as shown in the [results section](#). The lack of encryption makes it easier for attackers to craft malicious messages based on intercepted traffic. As EVs become increasingly connected and dependent on digital communication, the need for encryption is paramount.

Recommendation:

To address this, manufacturers can implement encryption on the vehicle network to protect confidentiality and integrity of transmitted messages. However, due to the 8-byte payload limit, the implementation of encryption needs to be lightweight, ensuring it does not introduce significant latency overhead as highlighted by Jadoon et al., (2018). Encryption would add a layer of confidentiality, preventing attackers from understanding and replaying messages, even if they are captured. Jadoon et al., (2018)'s paper presents a literature survey on the different encryption algorithms that can be used in EVs, this is a direction worth exploring adding different encryption algorithms and observing their effectiveness in protecting the CAN protocol.

Weak Message authentication: The penetration test revealed that while message authentication (MAC verification) was in place, it lacked sufficient robustness to prevent all attack types,

particularly replay attacks. The MAC tag did not incorporate dynamic elements such as time stamps or nonces, which allowed attacks to replay captured message at later time without being detected. This allowed the manipulation of vehicle functions like sensors and ECU states as demonstrated in the replay attack experiment. The recommendation for this weakness is addressed in the next section on replay attacks

Replay attacks: Replay attacks can bypass security and pose the severe threat to vehicle safety. They are believed to manipulate easy behaviour by replaying previous valid messages, demonstrating major weakness and how the system handles temporal message validation. This type of attack would result in on the rise control of critical vehicle functions such as breaking and steering.

Recommendation.:

To mitigate replay attacks, Luo et al., (2022) recommends the use of a freshness value in each message to thwart this type of attack. Alternately Baek and Shin (2022) suggest a robust time synchronisation mechanism across all nodes in the CAN Network, implement a message aging mechanism that discards old messages from memory by setting a timeout for each message type basis on its criticality and occurrence frequency. To detect message duplication a sliding window of recently received messages at each node, comparing incoming messages with those in the window then rejecting messages that appear within a brief period. These strategies ensure that if an article captures legitimate message, it cannot really be reused later without detection.

No message filtering: The lack of robust message filtering a lot faster and more informed messages to pass through system unchecked. While a simple range filter was implemented for CAN ID, this approach is not complete comprehensive enough to prevent more sophisticated attacks. The fuzz tests showed that while malformed or unexpected messages were classified and abnormal they were transmitted on the CAN bus. This poses a risk of Dos attacks. Although these messages did not lead to the direct manipulation of vehicle components, they demonstrated the potential to reduce the availability of critical functions.

Recommendation:

Future systems should include more advanced message for doing mechanisms that inspect CAN ID and message payload for inconsistencies. Additionally, an intrusion detection system could be employed to flag abnormal patterns of behaviour, such as high frequency of messages from a particular node which may indicate a brute force or a DoS attack. Machine learning based anomaly detection or contracts away filtering should be investigated as method of identifying potentially malicious messages in real time as suggested by Choi et al., (2021) in their have research.

Performance impact on security measures: The latency introduced by message authentication was found to be minimal, as demonstrated in the latency records. This indicates that the overhead introduced by message authentication is manageable with the constraints to CAN protocol.

Recommendation:

Given the minimal impact on performance, there is room for further security enhancements, such dynamic key generation as suggested by Baek and Shin (2022) without jeopardizing system performance. This would further reduce the risk of exploitation, ensuring that message is consistently maintained.

VII. Conclusion

The study explored vulnerabilities present in CAN bus communication protocol with a focus on how various fuzzing techniques and replay attacks could exploit these weaknesses. The results from the penetration test reveal the inherent security shortcomings on the CAN protocol, which was not designed with modern cybersecurity threats in mind. Key vulnerabilities found include, lack of encryption, inefficient message authentication mechanisms and absence of message filtering, were highlighted as a major risk factor that allow potential exploitation of automotive systems. Exploitation of these vulnerabilities enables manufacturers to of ECU behaviour and state. The overall findings from the tests (Random, Linear, Brute Force and Mutated Based Fuzzing) demonstrate that while some security mechanisms such as message authentication and message filtering were successful in blocking attacks significant vulnerabilities remain particularly in respect to replay attacks and lack of encryption. The delay introduced by the implementation of security mechanisms was minimal since a lightweight message authentication technique was used.

All together these findings highlight the need for comprehensive, multiple layered security approach that includes encryption, stronger message authentication, replay protection and through filtering of erroneous messages. Moving forward a multilayered defence strategy should be employed in automobiles to secure CAN bus against the broad range of attack vectors uncovered in the study. Regular security updates using over the air (OTA) updates suggested by Mahmood et al., (2022), audits, and the integration of new security technologies such as encryption and dynamic MAC verification will be essential in maintaining the integrity of the vehicle's communication system as new threats immerge. The study contributed to the ongoing dialogue about automotive cybersecurity and highlights the urgent need for industry wide adoption of stronger security practices. While this study demonstrated effective detection and prevention mechanisms in some areas the continued evolution of cyber-attacks necessitates continuous improvement of the protocols governing vehicle communication systems. By addressing these vulnerabilities manufactures can enhance vehicle safety, protect consumers, and ensure the resilience of automotive networks against malicious attacks

VIII. References

- Abbott-McCune, S., & Shay, L. A. (2016, October). Techniques in hacking and simulating a modern automotive controller area network. In 2016 IEEE International Carnahan Conference on Security Technology (ICCST) (pp. 1-7). IEEE.
- Aliwa, E., Rana, O., Perera, C., & Burnap, P. (2021). Cyberattacks and Countermeasures for In-Vehicle Networks. *ACM Computing Surveys*, 54(1), 1–37. <https://doi.org/10.1145/3431233>
- Baek, Y., & Shin, S. (2022). Canon: Lightweight and practical cyber-attack detection for automotive controller area networks. *Sensors*, 22(7), 2636.
- Black Hat*. (2024). www.blackhat.com. <https://www.blackhat.com/about.html>
- Choi, W., Lee, S., Joo, K., Jo, H. J., & Lee, D. H. (2021). An enhanced method for reverse engineering CAN data payload. *IEEE Transactions on Vehicular Technology*, 70(4), 3371-3381.
- Fowler, D. S., Bryans, J., Shaikh, S. A., & Wooderson, P. (2018, June). Fuzz testing for automotive cyber-security. In 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W) (pp. 239-246). IEEE.
- Heneghan, J., Shaikh, S. A., Bryans, J., Cheah, M., & Wooderson, P. (2019). Enabling security checking of automotive ecus with formal csp models. 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), 90–97. <https://doi.org/10.1109/DSN-W.2019.00025>
- ISO - International Organization for Standardization. (2011). ISO 26262-1:2011. ISO. <https://www.iso.org/standard/43464.html>
- Jadoon, A. K., Wang, L., Li, T., & Zia, M. A. (2018). Lightweight cryptographic techniques for automotive cybersecurity. *Wireless Communications and Mobile Computing*, 2018(1), 1640167.

- Jo, H. J., & Choi, W. (2022). A survey of attacks on controller area networks and corresponding countermeasures. *IEEE Transactions on Intelligent Transportation Systems*, 23(7), 6123–6141. <https://doi.org/10.1109/TITS.2021.3078740>
- Kirthi. (2023, October 6). Automotive Pen Testing for Beginners. Medium. <https://medium.com/@kirthi21105/automotive-pen-testing-for-beginners-0df5f20d684e>
- Luo, F., Zhang, X., Yang, Z., Jiang, Y., Wang, J., Wu, M., & Feng, W. (2022). Cybersecurity Testing for Automotive Domain: A Survey. *Sensors*, 22(23), 9211. <https://doi.org/10.3390/s22239211>
- Mahmood, S., Fouillade, A., Nguyen, H. N., & Shaikh, S. A. (2020, October). A model-based security testing approach for automotive over-the-air updates. In 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (pp. 6-13). IEEE.
- Mahmood, S., Nguyen, H. N., & Shaikh, S. A. (2022). Systematic threat assessment and security testing of automotive over-the-air (OTA) updates. *Vehicular Communications*, 35, 100468. <https://doi.org/10.1016/j.vehcom.2022.100468>
- Matrickz TV. (2019, June 5). Car Hacking Demo: How to Hack an ECU, Automotive Penetration Testing (2019). YouTube. <https://www.youtube.com/watch?v=4EVualAhHMc&list=PLe37bFaqMeUdXBaJOtrPC79NmyISJYIaU>
- Mocnik, R., Fowler, D. S., & Maple, C. (2023). Vehicular over-the-air software upgrade threat modelling.
- Mokhadder, M. A., Bayan, S., & Mohammad, U. (2021, May). An intelligent approach to reverse engineer CAN messages in automotive systems. In 2021 IEEE International Conference on Electro Information Technology (EIT) (pp. 1-7). IEEE.
- Morgan, R. (2019, December 9). Canbus hacking guide part 1: How to set up pican2 and log or decode canbus messages. YouTube. <https://www.youtube.com/watch?v=XK1qhMWP3-g>

- Parkinson, S., Ward, P., Wilson, K., & Miller, J. (2017). Cyber threats facing autonomous and connected vehicles: Future challenges. *IEEE Transactions on Intelligent Transportation Systems*, 18(11), 2898–2915. <https://doi.org/10.1109/TITS.2017.2665968>
- Python-can 4.3.1 documentation. (n.d.). Python-Can.readthedocs.io. Retrieved April 11, 2024, from <https://Python-can.readthedocs.io/en/stable/>
- RS485 CAN HAT - WaveShare Wiki. (n.d.). [Www.waveshare.com. https://www.waveshare.com/wiki/RS485_CAN_HAT](https://www.waveshare.com/wiki/RS485_CAN_HAT)
- RS485 CAN HAT - WaveShare Wiki. (n.d.). [Www.waveshare.com. https://www.waveshare.com/wiki/RS485_CAN_HAT](https://www.waveshare.com/wiki/RS485_CAN_HAT)
- Salfer, M., & Eckert, C. (2015, July). Attack surface and vulnerability assessment of automotive electronic control units. In 2015 12th International Joint Conference on e-Business and Telecommunications (ICETE) (Vol. 4, pp. 317-326). IEEE.
- Show, office F. C., Bristol, Park, B. S., Crescent, D., Green, E., Bristol, & Bs16 7fr. (n.d.). A Brief History of Electric Vehicles. Fully Charged Show. <https://fullycharged.show/blog/a-brief-history-of-electric-vehicles/>
- Smith, C. (2016). The car hacker's handbook: a guide for the penetration tester.
- Thorne, B. (2024, June 23). GitHub - hardbyte/Python-can: The can package provides controller area network support for Python developers. GitHub. <https://github.com/hardbyte/Python-can/tree/main>
- Werquin, T., Hubrechtsen, M., Thangarajan, A., Piessens, F., & Mühlberg, J. T. (2019, September). Automated fuzzing of automotive control units. In 2019 International Workshop on Secure Internet of Things (SIOT) (pp. 1-8). IEEE.

