THESIS

CYBERSECURITY VULNERABILITIES IN ELECTRONIC LOGGING DEVICES AND DEVELOPMENT OF A SOFTWARE DEFINED TRUCK TESTBED

Submitted by

Jacob Jepson

Department of Systems Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2024

Master's Committee:

Advisor: Jeremy Daily

Steve Simske
Indrajit Ray

ABSTRACT

CYBERSECURITY VULNERABILITIES IN ELECTRONIC LOGGING DEVICES AND DEVELOPMENT OF A SOFTWARE DEFINED TRUCK TESTBED

This thesis addresses critical cybersecurity vulnerabilities in Electronic Logging Devices (ELDs), mandated equipment for modern commercial trucks, and introduces an innovative solution for comprehensive system testing. Through extensive reverse engineering and practical testing, significant security flaws in commonly used ELDs are uncovered. These vulnerabilities enable unauthorized control over vehicle systems through arbitrary CAN message injection, allow upload of malicious firmware, and most alarmingly, present the potential for a self-propagating truck-to-truck worm.

To demonstrate these vulnerabilities, bench-level testing and real-world experiments were conducted using a 2014 Kenworth T270 Class 6 research truck equipped with a vulnerable ELD. The findings reveal how these security weaknesses could lead to widespread disruptions in commercial fleets, with severe safety and operational implications.

Addressing the fundamental challenge of disparate design and testing of after-market systems in trucks, this research introduces CANLay, a key networking component of the Software Defined Truck (SDT) concept. CANLay enables the virtualization of in-vehicle networks, facilitating the transportation of Controller Area Network (CAN) data and sensor signals over long-distance networks. This innovation allows for holistic security assessments and efficient testing of integrated vehicle systems, accounting for emergent behaviors that arise from system integration.

The efficacy of CANLay in heavy vehicle network performance testing is demonstrated, showcasing its potential to streamline system integration and verification efforts in a versatile digital engineering environment. This work contributes to the field by illuminating current vulnerabilities

in mandated trucking technology, demonstrating potential attack vectors, and providing a framework for more comprehensive and efficient testing of integrated vehicle systems.

This research underscores the urgent need to improve the security posture of ELD systems and offers recommendations for enhancing their security. The findings and proposed solutions have significant implications for improving cybersecurity in the trucking industry and, by extension, safeguarding critical supply chains.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my mentor and advisor, Dr. Jeremy Daily, for his invaluable guidance, support, and encouragement throughout my research journey. His expertise and insights have been instrumental in shaping this thesis and my academic growth.

I am sincerely grateful to Dr. Steve Simske and Dr. Indrajit Ray for serving on my thesis committee. Their constructive feedback and diverse perspectives have greatly enhanced the quality of this work.

My heartfelt thanks go to all my research colleagues who have contributed to this journey. Their collaboration, intellectual discussions, and moral support have been crucial in overcoming challenges and advancing my research.

I owe a debt of gratitude to my parents and brothers for their unwavering support, patience, and encouragement throughout my academic pursuits. Their belief in me has been a constant source of motivation.

Finally, I would like to acknowledge Colorado State University for providing the education, experience, and facilities necessary to conduct this research. The opportunities and resources offered by the university have been fundamental to the successful completion of this thesis.

To all those mentioned and many others who have contributed in various ways, thank you for being part of this significant chapter in my academic life.

DEDICATION

This thesis is dedicated to my late papa, Dr. C Neal Jepson, whose inspiration from my early years taught me the value of hard work and pursuing one's passions. And my parents, for their unwavering support and encouragement throughout this journey and beyond. Your guidance and love have been the foundation of my success.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

Commercial vehicles, specifically medium and heavy-duty trucks, serve as the backbone of supply chains and critical infrastructure globally. These trucks are not just vehicles, but pivotal cogs in the vast machinery that drives the world's economies. Their role extends far beyond simple transportation; they are integral in the distribution of a wide range of goods, from consumer products to industrial materials. This distribution network, supported by these commercial vehicles, is essential for maintaining the continuity and efficiency of global economic systems.

According to the US Bureau of Transportation Statistics, the United States alone has over 14 million medium and heavy-duty trucks registered, underscoring their prevalence and importance in national infrastructure [1]. Moreover, the American Trucking Association's report highlighted these trucks moved approximately 72.6% of the nation's freight by weight in recent years, showcasing their critical role in the country's freight transportation system [2]. This statistic further emphasizes the reliance of economies on these vehicles, not only for domestic transport but also for international trade and commerce. The seamless operation of these commercial vehicles is vital for the smooth functioning of supply chains, directly impacting everything from local businesses to international markets.

Given the crucial role of these vehicles, ensuring their efficient operation and compliance with safety regulations is paramount. One significant development in this area has been the introduction of Electronic Logging Devices (ELDs). These devices have transformed the way driving hours are recorded and monitored - from a legacy paper-based system to a moderized electronic system.

However, the integration of ELDs and other after-market systems into trucks presents a fundamental challenge in the realm of cybersecurity and system integration. ELDs, along with many other after-market systems, are typically designed, tested, and secured separately from the trucks they are installed in. This separation fails to account for the emergent behaviors, particularly in

terms of security, that arise when these systems are integrated into the complex ecosystem of a modern heavy vehicle.

To address this critical gap, this thesis introduces CANLay, a key networking component that enables the realization of the Software Defined Truck (SDT) concept. CANLay and SDT provide a comprehensive platform for testing and analyzing integrated vehicle systems, allowing researchers and developers to:

- Observe and study the emergent behaviors that occur when different systems interact within the truck's network.
- Conduct holistic security assessments that consider the entire ecosystem of the truck, rather than individual components in isolation.
- Identify potential vulnerabilities and security risks that may only become apparent when systems are integrated.
- Test and validate security measures in a controlled, yet realistic environment that mimics the complexity of actual truck systems.
- Iterate quickly on different configurations and scenarios without the need for physical hardware changes, significantly reducing the cost and time required for comprehensive testing.

This research not only illuminates current vulnerabilities in systems like ELDs but also paves the way for a more holistic approach to cybersecurity in the trucking industry. By providing a flexible, cost-effective platform for simulating and testing integrated vehicle systems, CANLay and SDT address the fundamental challenges in securing and optimizing the increasingly complex ecosystem of modern heavy vehicles.

1.0.1 Organization

The remainder of this thesis is organized as follows:

Chapter 2 provides a detailed analysis of Electronic Logging Devices (ELDs), including their system definition, architecture, and vulnerabilities discovered through reverse engineering. It also describes the development and evaluation of malicious ELD firmware. Furthermore, this chapter introduces the concept of a Truck-to-Truck Worm, detailing its design, implementation, and evaluation. This chapter demonstrates how vulnerabilities in ELDs could be exploited to spread malware across multiple vehicles.

Chapter 3 presents CANLay, a key networking component of the Software Defined Truck (SDT) concept. It covers the system design and development of CANLay, its components, operation, and example use cases. This chapter illustrates how CANLay addresses the challenges of testing integrated vehicle systems.

Chapter 4 concludes the thesis by summarizing the key findings, discussing the implications of the research, and suggesting directions for future work in enhancing the security of trucking technology and developing more comprehensive testing frameworks.

1.1 Background

Many heavy vehicles are required to be equipped with ELDs, since they are mandated by the Federal Motor Carrier Safety Administration (FMCSA) under the ELD Final Rule [3]. This so-called ELD Mandate is a component of the Moving Ahead for Progress in the 21st Century Act (MAP-21) and it went into effect December 18, 2017. It's worth noting that the mandate faced significant protest from some sectors of the trucking industry, highlighting the complexity of implementing such widespread technological changes [4]. ELDs are essential for recording driving hours and ensuring compliance with Hours of Service (HOS) regulations, which were implemented to prevent accidents due to driver fatigue [5]. These devices automate the process of capturing data on engine operation, vehicle movement, and miles driven, serving as a modernized alternative to traditional paper logbooks. The legal framework for ELDs requires self-certification and registration with the FMCSA. The regulation also includes provisions to protect drivers from harassment based on ELD data [3].

Modern vehicles, including heavy-duty trucks, are equipped with numerous computerized systems that control various aspects of vehicle operation. These systems communicate with each other through a vehicle network. ELDs acquire data by communicating with the vehicle's engine control module (ECM) through this network.

Specifically, ELDs typically interface with the Controller Area Network (CAN) bus, a robust and efficient communication system used in most vehicles, and the J1939 protocol, a standard specifically designed for heavy-duty vehicles. This connection is made through either the diagnostic connector or RP1226 connector. This setup enables ELDs to record mandated information such as engine hours, vehicle motion, and distance traveled, which are necessary for adherence to HOS regulations.

Notably, the J1939 standard suggests the engine hours parameter is available **only** "on request" over a J1939 network. This requirement necessitates that the ELD has the ability to write to the network and request the engine hours data, highlighting the bi-directional nature of ELD-vehicle communication. The diagnostic connector, such as a the 9-pin connector defined in SAE J1939-13 or the J1962 connector, allows the ELD to interface with the vehicle's communication network [6]. Additional connections defined by RP1226 provide dedicated access to third-party devices on the vehicle networks [7].

The CAN bus, known for its robustness, low latency, and efficiency, operates at the data link and physical layers of the Open Systems Interconnection (OSI) model, similar to how Ethernet functions in computer networks. It provides the basic infrastructure for communication between various modules in the vehicle [8]. Building upon this foundation, the J1939 protocol operates at higher layers, including the application layer, much like how HTTP operates over TCP/IP in internet communications. J1939 standardizes how messages are structured and transmitted specifically in heavy-duty vehicles [9]. Within the J1939 protocol, messages are organized into Parameter Groups (PG), each identified by a unique Parameter Group Number (PGN), analogous to how different types of data are organized in internet protocols. These messages carry operational parameters, described as Suspect Parameter Numbers (SPNs), and are encapsulated in the J1939

Protocol Data Unit (PDU). The PDU, akin to a packet in internet protocols, also contains metadata such as source and destination addresses, message priority, and the actual data payload.

Despite their widespread use, neither CAN nor J1939 are renowned for their security features. While a comprehensive analysis of CAN and J1939 vulnerabilities is beyond the scope of this thesis, they provide relevant context for this research. Notable research includes Miller et al.'s identification of a network-level vulnerability leading to network overload, and Burakova et al.'s explanation of application-layer vulnerabilities capable of controlling truck engines and disabling critical functions [10], [11]. Murvay et al. and Campo et al. highlighted network management layer weaknesses, such as ECU isolation and denial-of-service vulnerabilities [12], [13], while Mukherjee et al. and Chatterjee et al. demonstrated data-link layer vulnerabilities, including ECU overloads and denial of service via open connections [14], [15].

The lack of built-in security and ever-increasing connectivity, including ELDs, in the automotive industry opens up new possibilities for cyber attacks, including those that have traditionally targeted standard computer networks [16]–[18]. One such type of attack is the computer worm. In cybersecurity, a worm is defined as self-replicating malware that autonomously propagates across a network, exploiting security vulnerabilities or software flaws.

This convergence of traditional cybersecurity threats with the unique challenges of vehicle networks underscores the need for sophisticated engineering methodologies in the trucking industry. As the sector becomes increasingly reliant on digital technologies, ensuring both efficiency and security has become paramount. One approach gaining traction is Digital Engineering, a methodology that has seen great success in other sectors such as mechanical engineering with the ubiquity of Computer Aided Design (CAD).

Digital Engineering is an integrated approach to system design that utilizes authoritative sources of system data and models to support lifecycle activities from concept to disposal [19]. Its growing popularity in systems engineering and life cycle design is evident in initiatives like the Department of Defense's Digital Engineering Strategy, which mandates its use in designing the nation's most

expensive and complex systems [20]. Consequently, many companies have adopted this practice, particularly in cyber-physical system design.

The relevance of Digital Engineering in the trucking context is particularly pronounced due to the complexity and integration required across various vehicle systems and Original Equipment Manufacturers (OEMs). Modern trucks are equipped with numerous Electronic Control Units (ECUs) that manage everything from the engine and transmission to braking systems and driver interfaces. Ensuring these systems work harmoniously and securely requires the advanced tools and methodologies that Digital Engineering provides.

However, in horizontally integrated industries like heavy vehicles, OEMs may lack access to the complete technology stack, which can hinder comprehensive digital engineering. To address this challenge, the digital engineering approach must incorporate embedded hardware to represent the unavailable intellectual property. This hybrid method offers systems engineers a high-fidelity test and evaluation setup for rapid verification and validation before full deployment, enabling swift assessments of different architectural configurations and vendor solutions.

CANLay, a network-based testbed, is introduced as a key component in realizing the Software Defined Truck (SDT) concept. It facilitates the creation of hardware-in-the-loop testbeds using real ECUs, enabling early implementation testing and verification activities. By encapsulating traditional CAN frames within Ethernet frames, CANLay virtualizes ECU testbeds, eliminating the need for specific hardware test benches and allowing for software-based reconfiguration of system architectures.

One of CANLay's key advantages is its ability to leverage ubiquitous Ethernet infrastructure. This allows the testbed to be distributed, with CANLay nodes located in separate physical locations, significantly enhancing flexibility and efficiency in testing complex vehicle systems.

CANLay's application to heavy vehicle systems serves as a prime example of its utility. A baseline heavy vehicle typically comprises multiple controllers that communicate using the SAE J1939 protocol, which is implemented on top of the CAN bus. Unlike passenger cars, which may offer limited customization options through different trim levels, heavy-duty vehicles often fea-

ture extensive customization possibilities. For instance, customers may choose between different engine types (e.g., diesel, natural gas, or electric), various transmission options, multiple brake system configurations, and a range of telematics and fleet management systems. This high degree of variability in vehicle configurations necessitates a flexible test system.

CANLay provides the technology to construct such versatile test systems. It can accommodate proprietary systems from various vendors, allowing for the integration and testing of different combinations of components that might be present in a customized heavy vehicle. This capability enables a hybrid approach that capitalizes on modern digital engineering benefits while addressing the challenges posed by the diverse and modular nature of heavy vehicle systems.

International Council of Systems Engineers (INCOSE) defines 14 Systems Engineering Technical Processes from ISO/IEC/IEEE 15288 in the Systems Engineering Handbook [21]. These processes allow systems engineers to identify system needs and shape the design across engineering disciplines, delivering a system that meets both specifications and stakeholder requirements. These technical processes encompass the entire lifecycle, from conceptual design to development and operation through disposal. The Integration Process synthesizes a set of system elements into a realized system that satisfies the requirements [22], often assembling hardware, software, and other resources to verify implementation correctness. This process focuses on the interfaces between system elements. CANLay serves as an effective tool for executing this process in vehicle network design, providing a digital environment for early hardware and software verification activities.

The Verification Process, as defined in the INCOSE Handbook, aims to offer objective evidence that the system or its elements meet specified requirements [21], [22]. CANLay supplies a digital environment to gather this evidence, facilitating faster design iteration and integration with multiple system configurations. This work builds on the Software Defined Truck (SDT) concept presented at INCOSE IS in 2021 [23].

1.1.1 Contributions

This research encompasses several significant contributions to the domain of cybersecurity, particularly in the context of Electronic Logging Devices (ELDs) and vehicular networks. These contributions are itemized as follows:

- **Reverse-Engineering Analysis of an ELD** This work entails a detailed reverse-engineering process of a common off-the-shelf ELD. It thoroughly investigates its firmware update mechanism and delineates a potential method to compromise its firmware. This compromise could result in detrimental effects on both the vehicle and its immediate environment. The study also examines the implications of the malicious code on the device's standard operational performance.

- **Design and Demonstration of a Novel Truck-to-Truck Worm** This work introduces the design of an innovative worm, specifically conceptualized for truck networks, which is capable of autonomous propagation from one ELD to another. This self-replicating worm is practically demonstrated using ESP32 development boards. The demonstration serves to validate the feasibility and operational mechanics of the worm in a controlled environment.

- **Empirical Assessment of the Compromised ELD and Truck Worm** The research includes an evaluation of the effects induced by both the compromised ELD and the prototype truck worm. This assessment is conducted through a series of experiments designed to demonstrate the impact of the worm in real-world scenarios, providing empirical evidence of its potential repercussions.

- **Discussion of Countermeasures and Defensive Strategies** Finally, the thesis discusses a range of potential countermeasures and defense strategies aimed at mitigating the risks posed by the identified vulnerabilities. These strategies are proposed to enhance the security posture of ELDs against similar attacks in the future, thereby contributing to the broader field of vehicular cybersecurity.

- **Detailed Systems Engineering Design of the SDT's Networking Subsystem, known as CANLay** This thesis provides an in-depth systems engineering design of CANLay, the networking subsystem of the Software Defined Truck (SDT).

- **Demonstrations of CANLay's Utility in Conducting System Integration and Verification Processes** The utility of CANLay in system integration and verification processes is demonstrated, highlighting its role in facilitating faster design iteration and integration with multiple system configurations.

Chapter 2

Electronic Logging Device

2.0.1 System Definition

The system of interest is composed of two distinct systems that come together for use: a heavy truck and an electronic logging device. This distinction is important because it illustrates the interplay and interdependence within complex systems, which may exhibit emergent behaviors. In systems engineering, the integration of the heavy truck (the primary operational system) and the electronic logging device (ELD, a key data management subsystem) create a composed architecture. Each system by itself may not have cybersecurity concerns; the truck without an ELD does not have a wireless connection, and the ELD by itself cannot command a truck. The heavy truck acts as a dynamic operational platform, encompassing various mechanical and electronic components, while the ELD serves as a mandated interface for data logging, regulatory compliance, and potentially, vehicle control. This systemic integration, therefore, not only enhances operational capabilities but also introduces potential vulnerabilities and points of failure that can be exploited. The combination of a truck and ELD creates a system with an exploitable wireless connection that affects the operation of the truck.

2.0.2 Threat Models

Typical cyber-physical control oriented threat scenarios for ELDs or On-Board Diagnostic scanners (OBD-II scanners) have primarily involved either close proximity wireless access (e.g. via Wi-Fi or Bluetooth) or long-range access via cellular networks. In both scenarios, attackers exploit security flaws to launch attacks on vehicles, using the attached device as a proxy to the vehicle's internal CAN bus [24], [25].

This thesis introduces a novel approach for diagnostic plug-based attacks that significantly expands the potential reach and impact of such exploits. The key innovation lies in the worm-like behavior of the attack, which enables it to spread from truck to truck along the driver's route using

close proximity wireless access methods. This technique effectively combines the localized nature of short-range wireless attacks with the potential for long-reaching effects typically associated with cellular network-based attacks.

By leveraging the frequent interactions between trucks at common stopping points such as rest areas, weigh stations, and distribution centers, this worm-based approach could potentially affect a large number of vehicles across a wide geographic area. This method of propagation presents a unique challenge to cybersecurity in the trucking industry, as it exploits the mobile nature of the vehicles themselves to spread the attack.

From an attacker's perspective, a widespread worm-like attack of this nature would still face significant challenges. It would necessitate the compromise of numerous ELDs from multiple companies to inflict substantial damage. Additionally, since the late 2010s, vehicle manufacturers have integrated vehicle network gateways as a countermeasure to such exploits. These gateways, analogous to firewalls, are designed to obstruct malicious traffic while permitting legitimate communications. Furthermore, an ELD's physical disconnection from the vehicle by the trucker can promptly impede the attack, providing a simple but effective countermeasure.

However, a deeper analysis reveals that these perceived challenges are less formidable than they initially appear. Among the approximately 880 ELDs registered with the FMCSA [26], this research uncovers a much narrower diversity of devices than the number suggests. In reality, only a few tens of distinct ELD models are actually in widespread use. This limited variety is primarily due to a high degree of rebranding, where many ELDs are essentially clones of each other with minimal variations, sharing similar form factors, execution environments, and codebases.

Consequently, the differences between models are often limited to firmware variations, which do not significantly hinder the portability of malicious firmware across devices from the same manufacturer. This homogeneity in the ELD ecosystem significantly reduces the complexity of creating a worm capable of infecting a large number of devices.

In addition, recent studies have identified attacks capable of bypassing vehicle gateways by exploiting flaws in protocols permitted through these gateways [15]. Regarding the potential for

drivers to disconnect ELDs as a countermeasure, this action is less likely and less effective than it might seem. The legal requirement for drivers to use ELDs significantly reduces the likelihood of device removal, except during diagnosis by a technician. Typically, a technician disconnects the ELD to access the diagnostic port to scan for codes; however, this practice may actually obscure the diagnosis of issues originating from the ELD itself.

As an illustrative proof of concept, a malicious variant of the firmware for a widely used consumer-grade ELD, was developed to demonstrate its potential to initiate an attack against the vehicle. A comprehensive proof of concept for the worm-like capabilities is not constructed or published. Instead, the focus shifts to the developer board of the same ESP32 platform. Utilizing these boards, a proof of concept for a malicious firmware worm that propagates from one device to another will be exhibited and assessed.

The attacker initially requires the compromise of at least one device, achievable through a drive-by attack or by positioning at locations frequented by trucks, such as truck stops, rest stops, hubs, distribution centers, or ports. The attacker wirelessly connects to the device and uploads the malicious firmware. Once re-flashed, the device commences scanning for similar devices through the unutilized interface (either Wi-Fi or Bluetooth Low Energy). Upon detecting another device, it attempts to connect using default credentials if Wi-Fi is inactive [1], or without credentials for Bluetooth. Success leads to an over-the-air (OTA) firmware upgrade, transferring the malicious firmware to the subsequent device. There is a brief interruption in the other device's service before it resumes its standard functions and concurrently seeks additional devices for propagation. After spreading to a predetermined number of devices (X), set by the attacker, the device waits for a specific set of conditions (Y), akin to a logic bomb, to execute an attack (Z).

While the emphasis is not on devising the most destructive attack sequence, a potential scenario is offered for illustrative purposes: The device initially waits until it has propagated to multiple devices or a certain amount of time has elapsed. This stage ensures broader dissemination of the

[1]The ELD discussed throughout this work left all interfaces on by default. Therefore "active" and "inactive" refers to whether the interface on the compromised ELD is in use by a connected device.

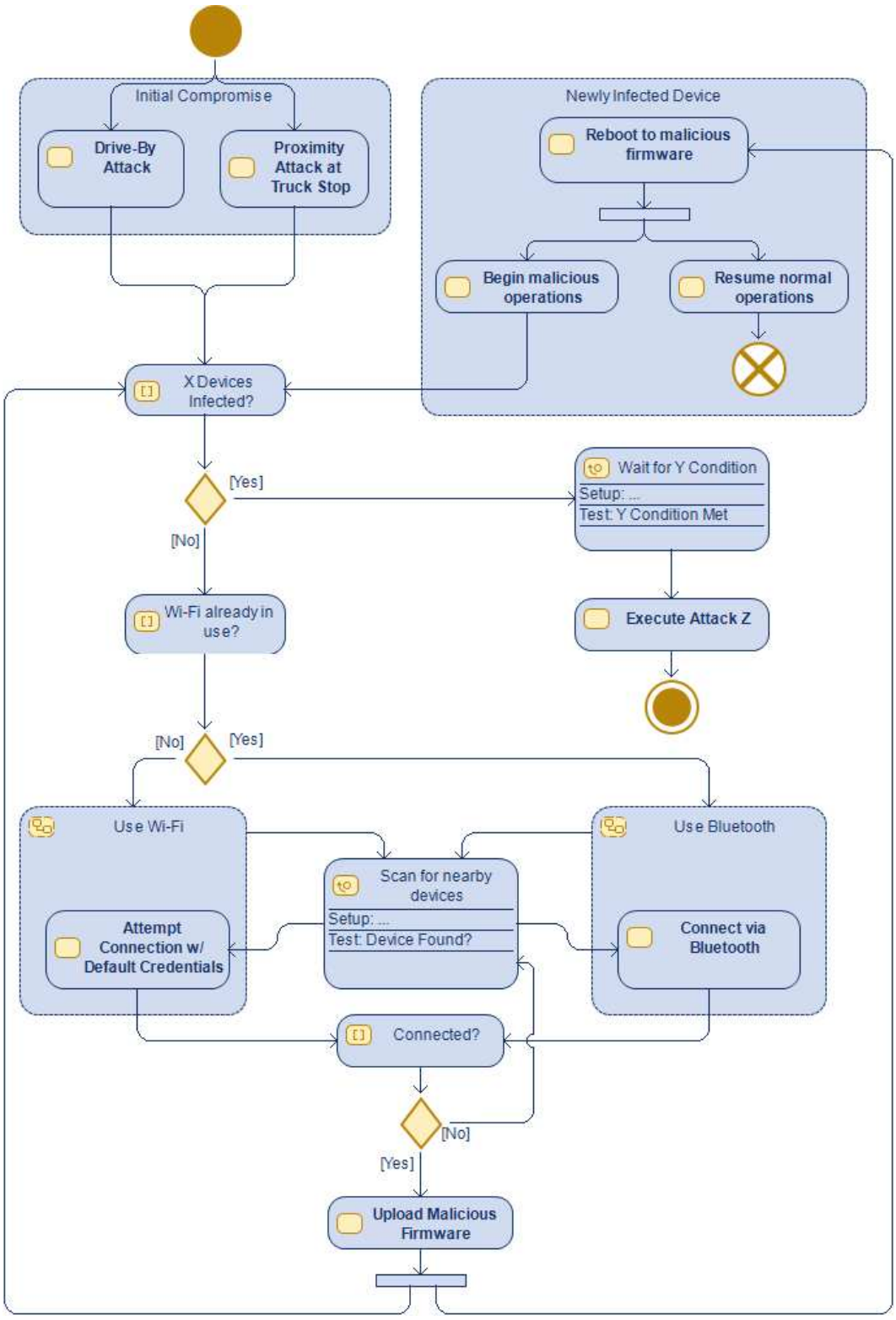


Figure 2.1: Overview of Infection/Spreading Operations

worm before executing an attack, which could otherwise impair the ELD and limit its infectious potential. Subsequently, the device waits until the vehicle reaches a vulnerable state (as indicated by CAN bus data), such as the first signs of deceleration after achieving highway speeds (65+ mph). At this time, the malicious ELD may flood the CAN bus with Torque/Speed Control 1 (TSC1) messages, commanding the engine to spin to maximum capacity, potentially causing the vehicle to unexpectedly collide with an object it was attempting to slow down and avoid.

2.1 Modifying The Firmware

This section describes how the firmware of a popular consumer off-the-shelf ELD was modified to execute an attack on the vehicle. While the focus is on a specific ELD for the purpose of this thesis, research indicates that these types of problems are not limited to this specific model or manufacturer. Therefore, disclosing the manufacturer's name or device model is not considered valuable for this thesis.

2.1.1 ELD Architecture

The hardware and software described here are specific to the device analyzed. However, multiple ELDs from the same manufacturer were observed to utilize almost identical architectures. Additionally, large overlapping similarities were noted between many of the diagnostic port-mounted ELDs.

Physical Device

The ELD analyzed is a small, hand-held device that plugins into the 9-pin diagnostic port on Heavy Vehicles. The device gathers data from the vehicle through the CAN channels exposed on the diagnostic port and is powered by the diagnostic port. The device has no other ports on it as all other communication is performed over wireless interfaces. The ELD presents 3 wireless interfaces namely: Wi-Fi, Bluetooth, and GPS. GPS is used for gathering location data while Wi-Fi and Bluetooth are used to communicate with an application on the user's phone or tablet. Depending upon the reseller, either Wi-Fi or Bluetooth Low Energy (BLE) is used, but an instance

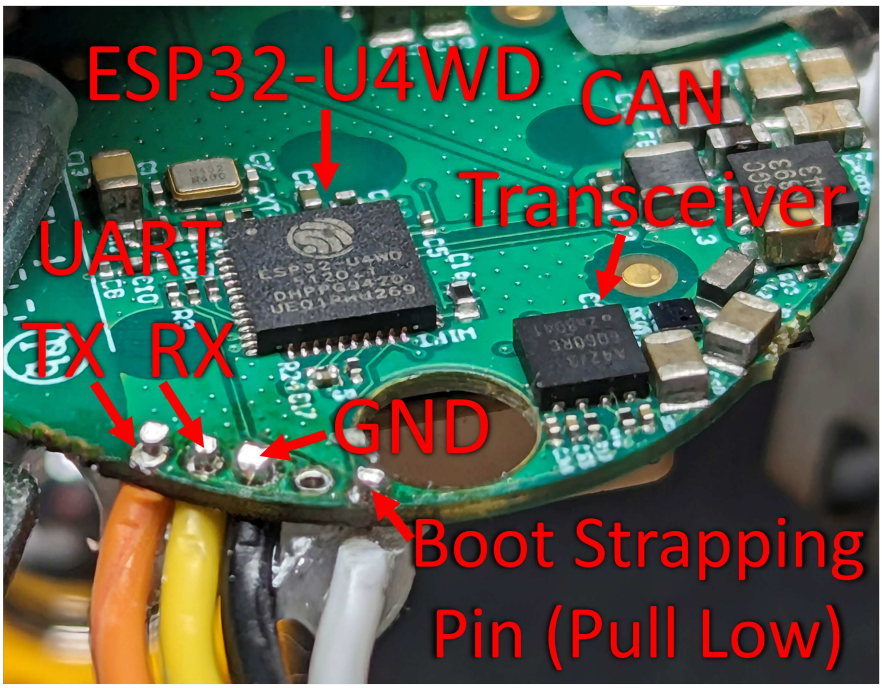


Figure 2.2: ELD PCB Up Close

where both were necessary was never discovered. An ESP32 chip is in charge of controlling the device. While the exact chip model varied between devices, they all were dual threaded with integrated storage, and supported Wi-Fi, BLE, GPS, and CAN. Other notable features include a small PCB antenna and an ESP programming port.

Execution Environment

The ESP32 chips executes a 32-bit Xtensa instruction set architecture (ISA) that blends 16-bit and 24-bit instructions in a RISC-based framework [27]. The device utilizes the Espressif Integrated Development Framework (ESP-IDF) which makes use of a Symmetric Multiprocessing implementation of FreeRTOS, a lightweight and fast Real Time Operating System (RTOS) [28]. If the General Purpose Input/Output (GPIO) pin 0, which is shown in Figure 2.2, is pulled down, the board will boot into the serial bootloader. Otherwise the board will boot the program code stored in flash. The bootloader connects to a host computer via a serial connection which allows for reading/writing to the flash memory, setting EFuses, and more [28].

Software Architecture

Since the ESP32 IDF utilizes FreeRTOS, most of the device's functionality is segmented into tasks or threads. At the core of the ELD's architecture is the main thread, initiated after basic hardware setup and system checks. This thread begins by initializing critical components like flash memory, GPIO pins, and the TCP/IP adapter. Following this initialization phase, the software architecture diverges into several distinct threads, each dedicated to a specific function. Notable threads include, operating the Wi-Fi interface for network connectivity, data transmission and device control, a similar thread for Bluetooth, a dedicated thread for Wi-Fi debugging, an upgrade thread which hosts a web server for easy firmware updates, and another for data collection and logging related to vehicle interfacing.

One of the most crucial aspects of this architecture, as it relates to this thesis, is the over-the-air (OTA) update mechanism. This feature allows the ELD to update its firmware wirelessly, using Wi-Fi or Bluetooth. To support this functionality, the device's partition tables are configured to include two OTA application partition slots and an OTA Data Partition. During an update, the system checks the checksum and SHA256 hash of the new firmware image for integrity violations before writing it to an inactive OTA slot. Once verified, the OTA Data partition is updated to boot from the new firmware.

2.1.2 Related Vulnerabilities Discovered

In the evaluation of ELD units procured from various resellers, factory default firmware settings were found to present considerable security risks. Reverse engineering efforts exposed an Application Programming Interface (API) that permits extensive device feature configuration, including over-the-air (OTA) updates. Contrary to expectations, initial connections with the re-seller's mobile applications do not activate a setup or reconfiguration routine towards more secure settings.

- **Default Network Settings** By default, Wi-Fi and Bluetooth functionalities are activated. The Wi-Fi Service Set Identifier (SSID) is predictable, following the format:

  `<Vendor Name> ELD: <MAC_ADDR>`

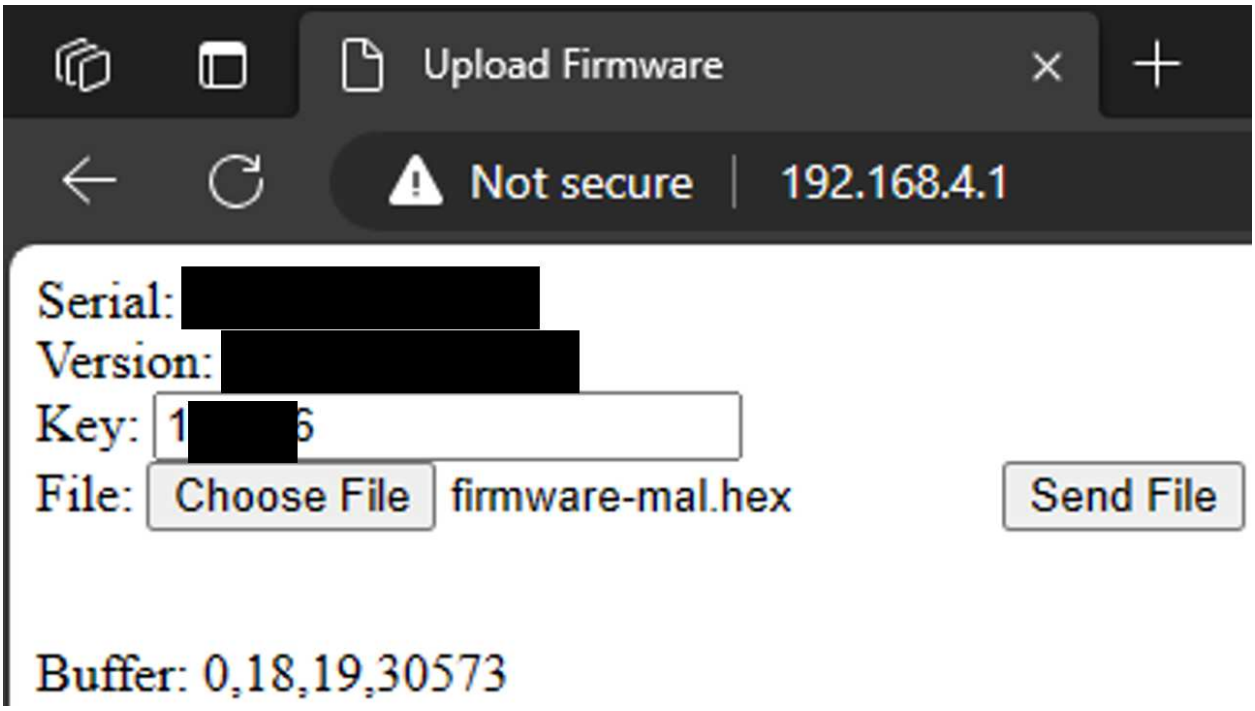


Figure 2.3: Firmware Upload Page

and is secured with a default password of minimal strength, `de*******77`[2].

- **Web Server and OTA Update Endpoint** The device hosts a web server, accessible via the IP address `192.168.4.1` which exposes numerous interesting endpoints. A critical endpoint on this server is `/upload.php`[3], which facilitates OTA firmware updates. This page is safeguarded by a basic password (`1****6`), rendering it susceptible to unauthorized access.

- **Exposed Debug Thread and APIs** The device features a wirelessly accessible echo service on port 22 used to debug the device's socket functionality. More significantly, it exposes an API that enables extensive control over the device, including the transmission and reception of arbitrary CAN messages and Over-The-Air (OTA) updates with no protection mechanisms in place. What's particularly noteworthy is that this API is accessible through two distinct

[2]At the request of the manufacturer the passwords were obscured and the commands changed. Although their form and function remains the same.

[3]Although, the endpoint ends in `.php` there was no evidence to indicate it used PHP.

communication stacks using the same set of commands: as a Bluetooth Low Energy (BLE) GATT (Generic Attribute Profile) service, and over Wi-Fi using the Telnet protocol on port 23. The API endpoints and parameters pertinent to the security vulnerabilities discussed in this thesis are outlined as follows, though this is not an exhaustive list:

- **CONFIGURE/ACTIVATE/DEACTIVATE Commands:** These commands are instrumental in configuring the device, encompassing tasks such as disabling unused interfaces and modifying default passwords. While these parameters were present in the firmware, their application within the reverse-engineered Android companion applications was minimal or non-existent.
- **INITIATEUPDATE/COMPLETEUPDATE Commands:** These commands are utilized for conducting OTA updates on the ELD. It is critical to note that, in contrast to the web-based update mechanism which requires a password, these commands do not require any form of authentication for OTA update initiation.
- **DATASTREAM Command:** This command activates the streaming of raw CAN messages to a connected device. Enabling this setting is a prerequisite for utilizing the ELD to dispatch arbitrary CAN messages onto the connected CAN bus.
- **Arbitrary CAN Message Transmission:** Unlike the other commands that employ ASCII-based words, the commands for sending arbitrary CAN messages consist of specific non-printable hexadecimal characters. These commands, along with the DATASTREAM command, were absent from all the Android applications from the resellers which were examined through reverse engineering, suggesting a reliance on the obscurity of these features as a form of security measure, though such a strategy was not explicitly confirmed. A snippet of the decompiled function that handles these commands is shown in Figure 2.4. The serial output after issuing these commands is shown in Figure 2.5. The following list delineates the initial character of each command, the corresponding CAN bus, and the format of the CAN ID (standard 11-bit or extended 29-bit IDs):

Figure 2.4: Decompiled Command Handle Showing Initial Commands Followed by Send Can

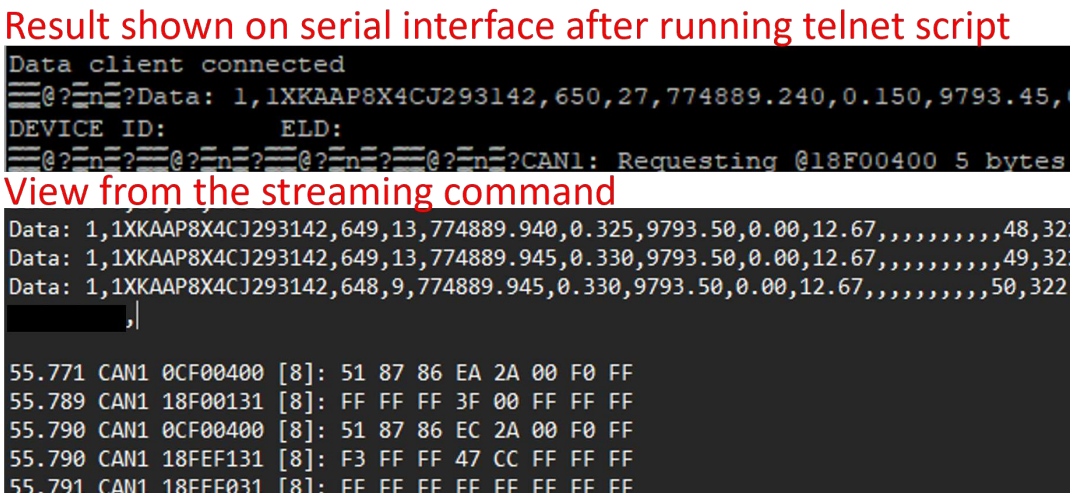


Figure 2.5: Serial Console Showing DATASTREAM Command and Arbitrary CAN Message Transmission

  - 0xF4: can1, extended.
  - 0xF5: can1, standard.
  - 0xF6: can2, extended.
  - 0xF7: can2, standard.
  - 0xF8: can3, extended.
  - 0xF9: can3, standard.

- **Firmware Security Flaws** The device lacks enforcement of firmware signing for authenticity, relying only on rudimentary integrity checks via checksums and hashes, which are inadequate for assuring firmware security. Digitally signed firmware is a better alternative.

Criteria for Successful Exploitation

- **Proximity for Wireless Access** Attackers must be within the wireless range to leverage Wi-Fi and Bluetooth vulnerabilities for actions like sending arbitrary CAN messages or uploading malicious firmware. The specific distance and time requirements for the attack will vary based on environmental factors such as physical obstacles, interference, and the power of the wireless transmitters involved. Experiments are conducted to quantify some of these parameters under different conditions in Sections 2.1.5 and 2.2.2.

- **Exploiting Default Settings**

  - **Bluetooth Access** The device uses a predictable Bluetooth identifier and employs the `Just Works` pairing mode, which is significantly less secure than other pairing methods such as `Numeric Comparison` or `Passkey Entry`. These more secure methods require additional information or user interaction to complete pairing, providing an extra layer of security. In contrast, the `Just Works` mode allows attackers to connect to the device if the driver is not currently connected [29]. Once connected, the exposed API allows them to either upload malicious firmware or manipulate CAN messages.

  - **Wi-Fi Access** The device's Wi-Fi, identifiable by a predictable SSID and protected by a weak password, grants network access. Once connected, attackers can either utilize the `/upload.php` endpoint on the internal web server for firmware uploads or exploit the API on port 23 via Telnet for sending/receiving CAN messages or firmware manipulation.

2.1.3 Analysis Techniques

Firmware Extraction and Acquisition

The initial phase of the research involved gathering information from public sources such as reference manuals, official documentation, and various online forums, which provided foundational knowledge about the ELD's operation and potential security issues [26], [28], [30]. Using this background information, the firmware was extraction from the ELD using ESPTool.py from Espressif [31]. The extraction process was facilitated by a serial to USB converter, which connected to the programming port on the ELD.

Additionally, newer firmware versions were obtained from the update servers, the addresses of which were identified through reverse engineering of the mobile applications associated with the ELD using JADX, a popular Dex to Java decompiler, shown in Figure 2.7 [32]. The discovery of

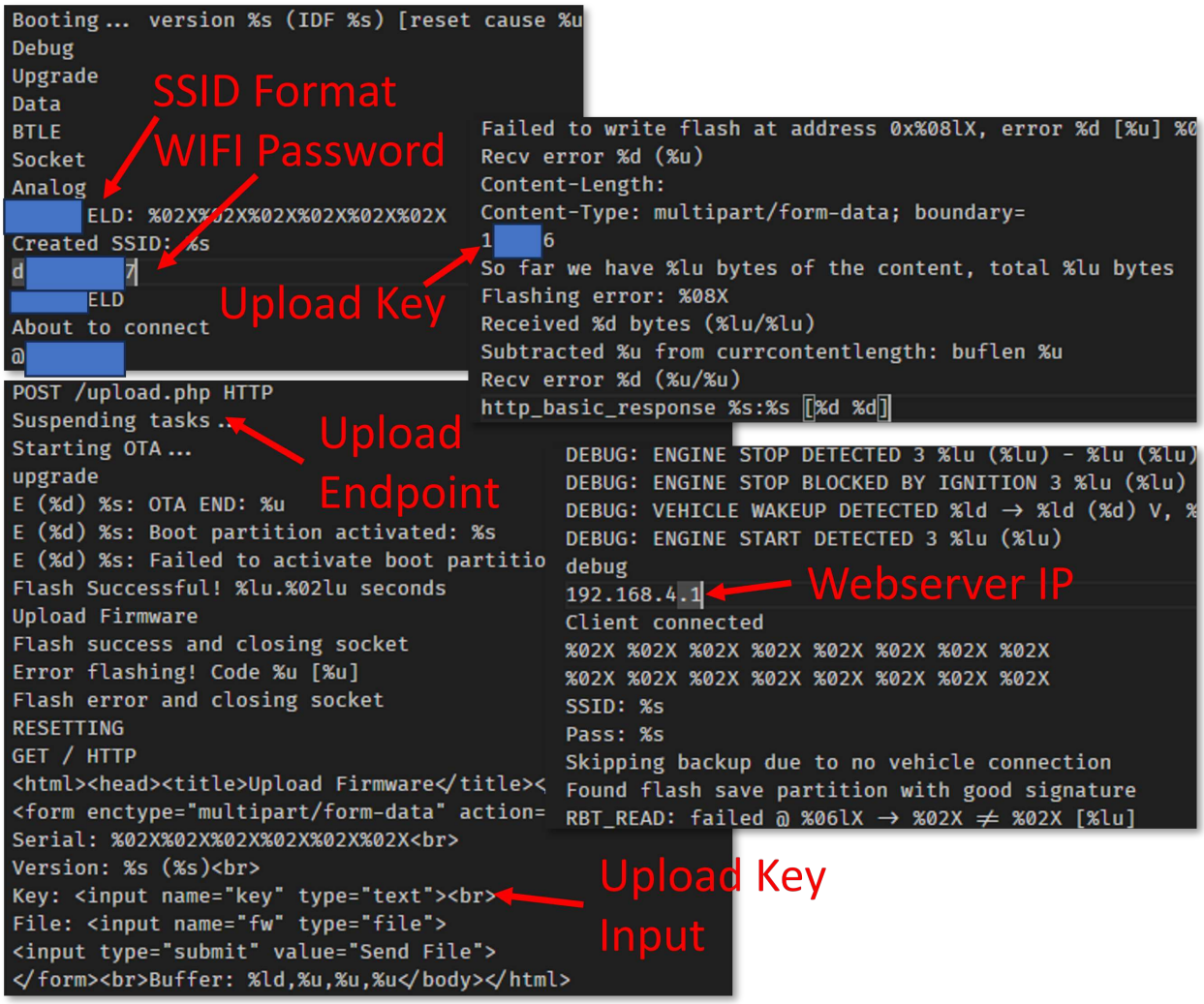


Figure 2.6: Credentials and Endpoints Found in Strings Output

```
public EldBleError StartUpdate(EldFirmwareUpdateCallback eldFirmwareUpdateCallback, String str) {
    Uri parse;
    if (this.managerState != EldManagerStates.MANAGER_CONNECTED || this.mNotifyCharacteristic == null) {
        return EldBleError.ELD_NOT_CONNECTED;
    }
    this.disconnectDetected = false;
    this.mEldFirmwareUpdateCallback = eldFirmwareUpdateCallback;
    this.f1221dm = new EldFirmwareDownload();
    if (str != null && str.length() > 0) {
        parse = Uri.parse("https://[redacted]/get.php?mac=" + this.mEldDevice.getAddress().replace(":",
    } else {
        parse = Uri.parse("https://[redacted]/get.php?mac=" + this.mEldDevice.getAddress().replace(":",
    }
    this.fwDownloadId = this.f1221dm.DownloadData(parse, mInstance);
    this.mEldFirmwareUpdateCallback.onUpdateNotification("Status: Downloading firmware update file from server");
    Thread thread = new Thread(new EldFirmwareUpdater(mInstance));
    this.updateThread = thread;
    thread.start();
    return EldBleError.SUCCESS;
}
```

Figure 2.7: ELD Android Library Firmware Update Function

credentials and webpage endpoints was achieved by running the GNU `strings` command on the firmware, as shown in Figure 2.6 [33].

Firmware Analysis Methodology

The analysis of the ELD's firmware involved a suite of tools, each contributing to understanding of its structure and potential vulnerabilities. Ghidra, a software reverse engineering tool, played a central role in this analysis [34]. The integration of ESP32-specific plugins into Ghidra was crucial as these plugins enabled the intake of firmware, creation of memory maps, and disassembly of the assembly code, specifically tailored to the ESP32 architecture [35], [36].

Network Analysis

The network footprint of the ELD, including open network ports and services, was mapped out using Nmap [37]. The analysis revealed various open ports and the services associated with them, providing insights into the network-based aspects of the ELD's security architecture. The ports and services are discussed in depth in previous sections but are summarized here for clarity:

- Port 22 - Socket Debug (echo server)
- Port 23 - Command API via Telnet
- Port 80 - HTTP Webserver

Observation of Default Credential Usage

A significant finding was the ease of access to the ELD's default Wi-Fi credentials, which were readily available through a basic internet search. This observation highlights a concerning security practice where default settings are often left unchanged, increasing the risk of unauthorized access and exploitation. The widespread availability of such sensitive information emphasizes the need for more robust and conscientious security measures in the deployment and management of ELD systems.

2.1.4 Development of Malicious ELD Firmware

In the context of this research, the development of the malicious firmware for ELDs was approached with a focus on demonstrating the feasibility of manipulating ELD firmware to perform an attack on a truck, rather than creating highly destructive software. To this end, the constructed firmware was relatively simple and designed for safe testing.

Listing 2.1: Malicious Firmware Inserted in Debug Thread

```
void debug_thread(void) {
  while (((STATUS & 2) == 0) && ((STATUS & 4) == 0)) {
    vTaskDelay(1000);
  }
  while(true) {
    can_data = 0xCB000000FF00FFFF;
    send_can(0xC000003, &can_data, 8);
    vTaskDelay(1);
  }
}
```

The code in Listing 2.1 is explained in the following subsections.

Construction of the TSC1 Message

The J1939 Torque/Speed Command 1 (TSC1) message is a legitimate command used to control engine output over the J1939 network. For example, the transmission control unit (TCU) broadcasts this message to inform the engine when the vehicle reaches the electronic governor's speed limit. However, this message can be abused. If an attacker gains access to the vehicle's network, they could potentially send false TSC1 messages to manipulate engine output, leading to unexpected and potentially dangerous vehicle behavior. The following details explain how the message was composed.

CAN ID: The CAN ID used is a 29-bit identifier, following the J1939 standard. The PGN for TSC1 is 0, and the default priority is 3, which was retained for this experiment. The source address selected was 3 (0x03), corresponding to the spoofed Transmission Controller. These elements combined to form the ID: 0x0C000003.

J1939 Suspect Parameter Number (SPN) Specifications:

- SPN 695 controlled the Engine Override Control Mode, with values ranging from 'Override Disabled' to 'Speed/Torque Limit Control'.
- SPN 696 defined the Engine Speed Control Conditions, optimized for various driveline engagement scenarios.
- SPN 879 set the Override Control Mode Priority, ranging from 'Highest Priority' for urgent safety actions to 'Low Priority' for driver comfort.
- SPN 898 and SPN 518 specified the Engine Speed/Speed Limit and Engine Torque/Torque Limit, respectively, offering control over engine speed and torque based on external input.
- The other SPNs are not important for this scenario and are set to FF's as default values.

Data Packet Formation: The final data packet for the TSC1 message was composed as 0xCB000000FF00FFFF.

Integration into the Firmware

The malicious code was integrated by replacing the code in the firmware's debug thread that was left in the production devices. This strategic placement allowed the malicious code to function as its own FreeRTOS thread/task. Upon creation, the thread/task begins by checking a global `STATUS` variable. While the true meaning of the `STATUS` variable's states remains elusive, a set of states appeared to indicate when the CAN interfaces completed their setup routines, including automatic baud rate detection.

The core function, termed `send_can`, was responsible for dispatching the constructed CAN messages. It accepted parameters including the ID, data buffer, and data length. After invoking

`send_can` with the formulated TSC1 message data, a delay of 1 millisecond was introduced. This delay served two purposes: ensuring adequate propagation time for the message onto the CAN bus, and allowing for message flooding to override legitimate TSC1 messages.

To bypass integrity checks after modifying the firmware, esptool.py was utilized to recalculate the correct checksum and SHA256 hash. A custom Python script automated the entire process: reading the original firmware, modifying the binary to include malicious code, and then invoking esptool.py for hash recalculation. This approach ensured the modified firmware appeared legitimate to the device's validation mechanisms.

Safety Measures in Experimentation

To ensure safety, the experimental use of the TSC1 message was carried out on a private and empty, airfield runway. The parameters chosen for the TSC1 message were set to slow down the truck instead of speeding it up, presenting a safer testing scenario. This controlled environment allowed for a detailed evaluation of the firmware's capabilities without posing a risk to public safety or property.

2.1.5 Evaluation of the Malicious Firmware and its Deployment

The evaluation of the malicious firmware and its deployment was conducted through two primary types of tests: bench-top tests and a real-world drive-by attack simulation.

Bench Top Testing

The bench-top test setup, as depicted in Figure 2.8, provided a controlled environment for initial evaluations. In this setup, the ELD, with its ESP programming port connected to a USB-to-serial adapter, facilitated monitoring via the serial console. The ELD was also connected to a CAN bus shared with a Macchina M2, replaying logged data from a truck, simulating a real vehicle CAN bus in a bench-top setting [38]. The firmware was uploaded to the ELD, through the previously described interfaces and APIs and averaged around 12 seconds for flashing, with additional operations bringing the total average duration to approximately 30 seconds. These tests

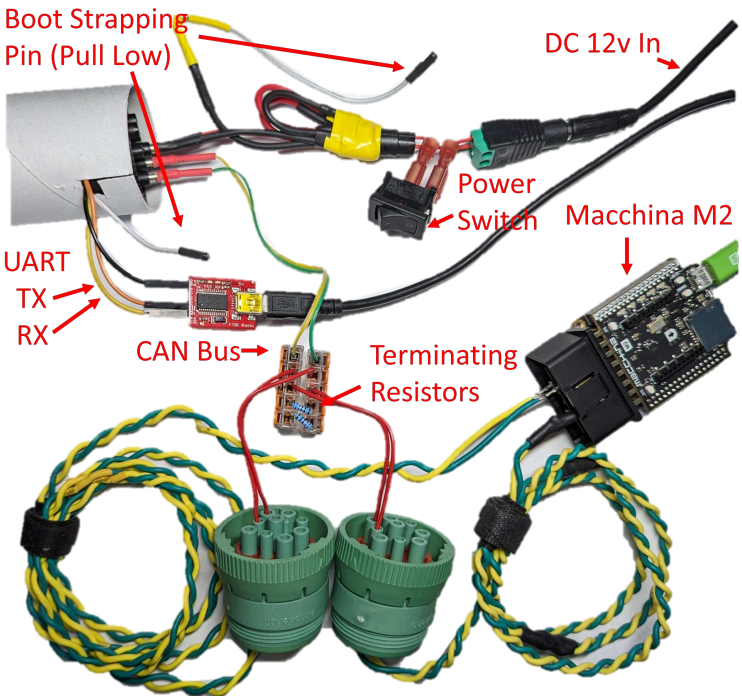


Figure 2.8: Table Top Lab Setup

```
(1703173683.505126) can0 0CF00203#C5AA14FFF7592503
(1703173683.511326) can0 18FEF200#6A00DB082D09E1FF
(1703173683.514475) can0 0CF00331#FFFFFFFFFFFFFFFF
(1703173683.515158) can0 0CF00203#C5AA14FFF7592503
(1703173683.516326) can0 18F00503#7F12077F3520324C
(1703173683.536750) can0 0CF00400#0195956F25000F95
(1703173683.543460) can0 0C010305#FFFFFFFFFFF3FFFF
(1703173683.574101) can0 18F0010B#CFFFF0FFFFCDFFFF
(1703173683.581152) can0 18F03300#FFFFFFFFFFFFFFFF
(1703173683.586606) can0 18FEBF0B#90177E7C7C7EFFFF
(1703173683.587527) can0 18FEDF00#87A0287D7DFFFFF5
(1703173683.588105) can0 18FE4A03#030F4FFFFFF3FFFF
(1703173683.588680) can0 0C000003#CB000000FF00FFFF
(1703173683.589229) can0 0CF00203#C58914FFF7302503
(1703173683.589801) can0 0C000003#CB000000FF00FFFF
(1703173683.590379) can0 0CF00C03#E8030D25FFFFFFFF
(1703173683.590932) can0 18FEE000#4B9100001C283600
(1703173683.591508) can0 0C000003#CB000000FF00FFFF
(1703173683.592076) can0 0CF00300#D0511EFFFF0F7E81
(1703173683.592626) can0 18FEF200#6B00DC082C09E1FF
(1703173683.593229) can0 0C000003#CB000000FF00FFFF
(1703173683.593801) can0 0C010305#FFFFFFFFFFF3FFFF
(1703173683.594376) can0 0C000003#CB000000FF00FFFF
(1703173683.594955) can0 18F00F00#D80EFFFFC1BFFFFF
(1703173683.595500) can0 0CF00203#C58914FFF7302503
(1703173683.596076) can0 0C000003#CB000000FF00FFFF
```

Figure 2.9: Start of the attack (highlighted in green)

aimed to verify the firmware's functionality, particularly focusing on the data logged on the CAN bus. The evaluations confirmed that both the malicious firmware and the API access point, which enabled sending and receiving arbitrary CAN messages, performed as expected.

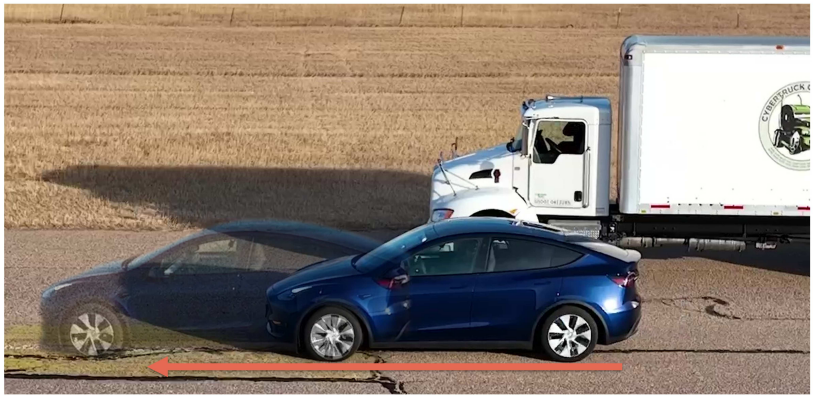
Real-World Drive-By Attack Simulation

The drive-by attack simulation was conducted on an empty mile-long airfield. This test aimed to demonstrate the potential of the ELD to spread in scenarios where trucks congregate, such as rest stops, or even while in transit. For this experiment, a 2014 Kenworth T270 Class 6 research truck, equipped with the ELD, was driven at approximately 20 miles per hour down the runway. The attacker, a passenger in a Tesla Model Y with a laptop and an Alfa extended range wireless adapter, executed the attack. This setup was chosen to expedite the attack due to the airfield's limited length, although subsequent experiments (discussed in the next section) suggest that an extended range adapter might not be necessary if more time is available to conduct the attack.

Figure 2.10: Research Truck (top) with ELD plugged into Diagnostic Port and hanging to the right of the door edge (bottom).

1
Connect to the ELD's Wi-Fi beside the truck and reflash it with malicious firmware.

3
The truck driver presses the accelerator petal to no avail.



2
After rebooting, the ELD spams TSC1 “Petal Jam” messages slowing the truck to a crawl.

Figure 2.11: Drive By Attack Steps

The attack vehicle initially followed the truck, then, to simulate a passing scenario, moved alongside it. The attacker connected to the truck's Wi-Fi and used the ELD's web interface to re-flash the device while both vehicles were in motion. Notably, the signal strength inversely affected the flashing duration; however, the extended range adapter allowed the re-flashing process to be completed in approximately 14 seconds. Around 20 seconds later, allowing time for the ELD to initialize its CAN interfaces, the truck began to slow down as the ELD flooded the CAN bus with malicious TSC1 messages. This test successfully demonstrated the feasibility of a drive-by attack, highlighting the realistic possibility of such an attack occurring under actual driving conditions. Figure 2.11 can be referenced for additional context.

2.2 Truck to Truck Worm

2.2.1 Overview

The Truck to Truck Worm (TtTW), developed for ESP32 development boards, executes an infection and propagation strategy tailored to Electronic Logging Devices (ELDs). The concept demonstrated with development boards is shown in Figure 2.12.

Initial Setup and Network Configuration

The worm begins by initializing the essential system components for its operation. This setup involves configuring network interfaces in both access point (AP) and station (STA) modes, preparing the file systems, and setting up CAN bus communication protocols.

Scanning and Identification of Targets

During the scanning phase, the worm utilizes its Wi-Fi capabilities to search for nearby ELDs that are potential targets. It specifically looks for devices with SSIDs starting with "VULNERABLE ELD:". Although this may sound contrived the SSID of the ELD being examined was predictable and could be used to identify the vulnerable devices.

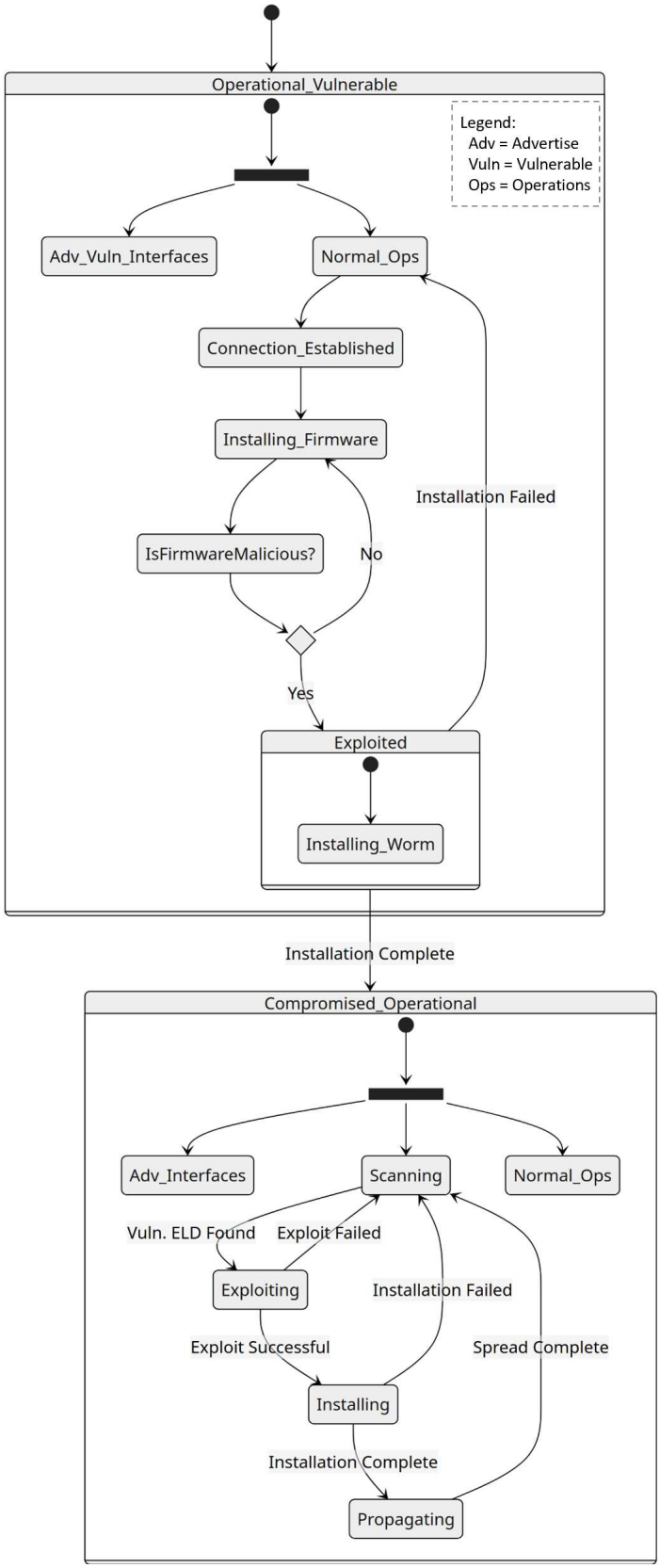


Figure 2.12: ELD State Machine Diagram

Infection Process

Once a vulnerable ELD is identified, the Truck to Truck Worm proceeds with the infection process. The worm exploits a vulnerability by connecting to these ELDs using hardcoded default credentials. Upon establishing a connection, the worm transfers its malicious payload to the target ELD in the form of an over the air (OTA) update. This payload is designed to embed the worm within the system, overwrite existing firmware, and prepare the device for its role in further propagation.

Post-Infection State and Propagation Readiness

After successfully infecting an ELD, the worm alters the Wi-Fi AP of the infected device to "INFECTED ELD." This change serves an important purpose: it signals to other instances of the worm that the device is already infected, preventing unnecessary re-infection attempts. Moreover, this post-infection state sets the stage for the device to become an active participant in spreading the worm. The infected ELD, now operating under the control of the worm, continues the cycle of scanning, identifying, and infecting other vulnerable ELDs, perpetuating the spread of the Truck to Truck Worm.

2.2.2 Truck to Truck Worm Evaluation

A test was conducted to evaluate the Truck to Truck Worm developed on ESP32 development boards, focusing on its range and effectiveness in stationary conditions. [4]

Testing Worm Spread in Stationary Conditions

The test aimed to assess the worm's spreading distance and effectiveness in a fully parked, stationary setting. For this, two ESP32 development boards, which closely emulate the power and antenna size of actual ELD devices were used. This parity in hardware specifications is critical

[4]The firmware for the conceptual Truck to Truck Worm Design can be found here: https://github.com/SystemsCyber/Truck-Worm

for obtaining realistic results, especially in contrast to the earlier drive-by test where an extended range Wi-Fi antenna, with its superior power and larger size, was used.

The procedure began with positioning one ESP32 development board adjacent to the diagnostic port of the SystemsCyber research truck. Simultaneously, the programs on both ESP32 devices were activated: one functioning as a Wi-Fi network host and the other programmed to scan and connect to this network. The key aspect of this test was to methodically move from one parking space to another, placing the second ESP32 device near the metal door of each truck to gauge the maximum effective connection distance.

The findings revealed that, even in a full parking lot, a connection could be established up to approximately 12 parking spots away, equivalent to about 120 feet. This distance demonstrated a considerable range for the Truck to Truck Worm to spread in environments where trucks commonly congregate. Furthermore, the test indicated that the proximity of trucks is a critical factor for the worm's spreading efficiency, with closer distances leading to quicker and more successful propagation. Under optimal conditions the Truck to Truck Worm was capable of spreading between ESP32 devices in under 30 seconds.

2.3 Prevention

To address the vulnerabilities identified in this work and effectively prevent Truck-to-Truck Worm attacks in electronic logging devices (ELDs), a multifaceted, or otherwise known as a defense-in-depth, approach is required. This approach encompasses the enhancement of default security settings, implementation of robust firmware integrity and authenticity checks, and the elimination of unnecessary and high-risk features. The specific recommendations for enhancing the security of ELD systems, as outlined in this section, are tailored to balance stringent cybersecurity needs with the practical requirements of affordability, reliability, and user-friendliness inherent in ELDs. This balance is crucial in the context of the trucking industry, where ELDs are a widespread and mandatory technology. These mitigations are inspired by the National Motor

Freight Traffic Association, Inc. (NMFTA) with their Cybersecurity Requirements for Telematics Systems [39]. The detailed strategies are as follows:

2.3.1 Enhancing Default Security Settings

Disabling Unused Interfaces and Services

To minimize the attack surface, it is crucial to disable any interfaces and services that are not in active use. This work revealed that while some resellers utilized Bluetooth, others employed Wi-Fi, but none concurrently used both interfaces or the web server. Therefore, ELDs should be configured to disable unused wireless interfaces and the internal web server by default. Furthermore, when the ELD establishes a connection via one interface, it should disable the other during this session. For example, if a user connects to the ELD via Bluetooth, the Wi-Fi network should be disabled while the Bluetooth connection is active. Similarly, if the user connects to the Wi-Fi and uses one of the APIs (either the web server or the Telnet API), the other should be disabled for the remainder of the connection (in addition to disabling Bluetooth).

High-Entropy Default Passwords Implementing high-entropy passwords for initial device access significantly enhances security. However, balancing security with usability in ELDs presents a unique challenge. ELDs are often installed in hard-to-reach places, are small in size, and may be shared among multiple drivers within a company. These factors make entering complex passwords on small mobile screens impractical and time-consuming. To address this, two approaches could be considered:

- *Complex Randomized Passwords* For infrequently accessed ELDs, such as those managed by IT or OT personel in large fleets: it is recommended to use long, completely random passwords that are unique to each device. These passwords should be securely labeled on it, akin to practices observed in modern router configurations.

- *Randomized Pins* Alternatively, a random pin could be used and labeled on each device. This method also ensures a degree of randomness while maintaining user convenience.

Firmware Integrity and Authenticity Checks

Secure Firmware Signing Mechanism: It is imperative to utilize a secure firmware signing mechanism. This involves cryptographically signing firmware updates using a private key held by the manufacturer. The ELD would then use the corresponding public key to verify the signature before installing any update. This process ensures that only authentic and untampered firmware from the verified source is installed on the ELDs, thereby preventing the installation of malicious firmware.

Restriction on Arbitrary CAN Message Functionality

Eliminating Unnecessary API Features: The findings suggest that the ability to send and receive arbitrary CAN messages via an API in a production ELD presents an unwarranted risk. While the manufacturer may have a valid use case for remotely sending arbitrary CAN messages wirelessly, this was not discovered during this research. From a systems engineering perspective, this feature presents a high risk that needs to be managed. Proper risk analysis and mitigation techniques should be applied, such as:

- Conducting a thorough risk assessment to determine the potential impact of this feature.
- If the feature is deemed necessary, implementing strong authentication and encryption mechanisms to secure its use.
- Applying the principle of least privilege, limiting the scope of CAN messages that can be sent.
- Considering a whitelist approach, where only pre-approved CAN messages are allowed.

If no critical use case is identified, it is recommended to eliminate this feature from ELDs entirely. Restricting this functionality will significantly reduce the risk of unauthorized access and control over the vehicle's CAN network, thereby mitigating potential security threats

2.3.2 Implementing Telematic Device Firewalls/Gateways for Enhanced Security

In addition to the device-centric preventative measures previously discussed, trucking companies should consider adopting proactive security practices for their fleets, particularly in the case of older vehicles that might not have built-in security gateways. A significant step in this direction involves the use of Telematic Device Firewalls or Gateways, which serve as an intermediary layer of security between the Electronic Logging Devices (ELDs) and the vehicle's diagnostic port. This recommendation aligns with guidance from the National Motor Freight Trucking Association (NMFTA) [39].

Functionality and Benefits: Telematic Device Gateways are designed to filter the data between an ELD and the vehicle's diagnostic port, essentially acting as an bolt-on gateway or firewall. These gateways can help prevent unauthorized access and malicious messages from reaching critical vehicle control systems through the diagnostic port [40]. This approach is particularly important for older vehicles, which may lack some cybersecurity features found in newer models.

In summary, implementing these preventative measures would enhance the security of ELD systems against Truck to Truck Worm attacks. By focusing on more secure default settings, firmware integrity checks, adoption of Telematic Device Gateways, and the elimination of unnecessary and risky functionalities, manufacturers and users of ELDs can reduce the likelihood and impact of such cybersecurity threats.

However, while these measures address many of the current vulnerabilities in ELD products, they don't fully resolve some fundamental issues. Vehicles still inherently trust third-party devices, and there remains a significant challenge in efficiently and reliably testing the interoperability and security of vehicles and ELD systems. To address these challenges, CANLay is introduced in the next chapter.

Chapter 3

CANLay

The Software Defined Truck (SDT) aims to offer a distributed network virtualized platform for conducting in-vehicle security and networking experiments. Although primarily proposed for heavy trucks, SDT can be easily adapted for lightweight passenger vehicles. SDT's information exchange objective is illustrated in Figure 3.1, where CAN frames and physical signals must be exchanged between ECUs and vehicle simulators in different subnetworks. CANLay serves as the networking backbone of the Software Defined Truck, providing the necessary infrastructure for a distributed service.

Previous research has established two critical criteria for quality evaluation of automotive networking test beds: fidelity and adaptability [41]. Fidelity refers to the capacity to emulate real-world in-vehicle networking infrastructure, while adaptability concerns the ability to simulate various real-world infrastructures. Optimizing both may be challenging, as an adaptable system necessitates virtualized components that can quickly adjust to user needs, potentially compromising fidelity. CANLay enables configuring experimental networks on-demand and provides a real-time health report for the underlying network, allowing users to assess overlay fidelity in terms of standard networking metrics such as latency and packet drop rate.

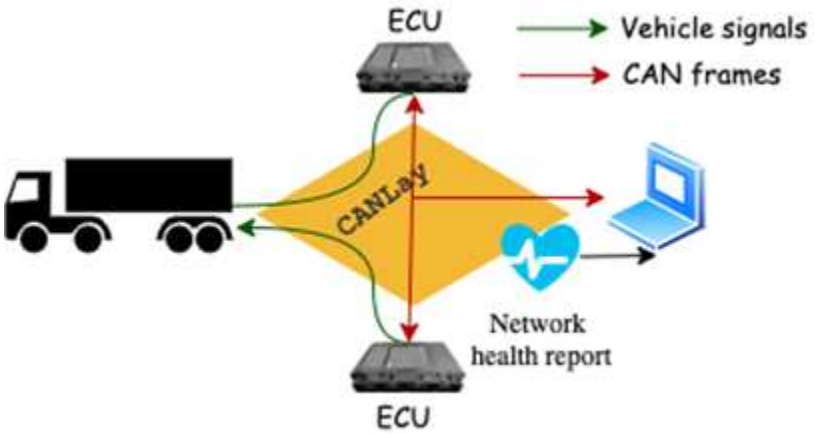


Figure 3.1: Software Defined Truck Information Exchange Overview

Although CAN is newer and has a narrower application scope than Transmission Control Protocol/ Internet Protocol (TCP/IP) communication technology, attempts have been made to virtualize its operations. First, researchers have adapted the software-defined networking paradigm for CAN [42]–[44], which is mainly hardware-based and tailored for in-vehicle networking on CAN physical channels rather than long-range overlays. For range relaying of CAN frames, CAN-to-Ethernet research has been conducted [45], [46] to transport data logged from one network to a remote endpoint, but this did not address configurability or network performance. X-in-the-loop (hardware, driver, vehicle, etc.) simulation-based in-vehicle test beds have been proposed [47]; however, signals from simulators have been transported over physical connections rather than reconfigurable, long-range network overlays. Vendors like Vector VT, NI LabView, and dSpace Systems offer X-in-the-loop simulation-based test bed solutions for hardware, drivers, and vehicles [48]–[50]. These solutions focus on simulating sensor inputs and testing complex embedded systems found in vehicles, but security testing is usually conducted earlier in the ECU development cycle in a virtualized manner. Cloud-based solutions like those offered by ETAs provide virtual security testing for ECUs, but they are limited to ECUs from the same manufacturer and do not represent a typical CAN network, as most production vehicles contain ECUs from various manufacturers [51].

CANLay has the following functional objectives:

- Transport of CAN frames over a distributed overlay network of electronic control units.
- Transport of sensor signals to a distributed network of electronic control units.
- Support for on-demand session creation.
- Provision of runtime metrics to estimate the network health during the ongoing experiment.

In addition, these top-level requirements were developed for CANLay:

- The CANLay system shall support network virtualization to enable seamless communication between different physical nodes in the Software-Defined Truck (SDT) environment.

- The CANLay system shall provide a real-time health monitoring service that measures network parameters such as packet loss, latency, jitter, and goodput, and makes this information available to users.
- The CANLay system shall be scalable, capable of supporting various network architectures and adapting to different cyber-physical system development scenarios.
- The CANLay system shall be compatible with existing CAN-based communication protocols and standards used in vehicle systems and other cyber-physical systems.
- The CANLay system shall provide an intuitive and easy-to-use interface for users to interact with and manage the network, design experiments, and monitor network health.

The remainder of this chapter describes CANLay's design, discusses its usage in an example scenario, and concludes with remarks on future directions.

3.1 CANLay System Design and Development

Figure 3.2 illustrates the proposed system design of CANLay, which serves three functions:

- Online configuration of the network overlay.
- CAN frame exchange at runtime.
- Vehicle signal exchange at runtime.

The components and their roles in the system are described first, followed by an explanation of the system's behavioral aspects, which together achieve the functional objectives.

3.1.1 CANLay Components

Smart Sensor Simulator and Forwarder (SSSF)

The SSSF serves as a gateway, enabling the ECU to access and be accessed by the CANLay system. During an active experiment, the SSSF acts as a forwarder between the Controller and the

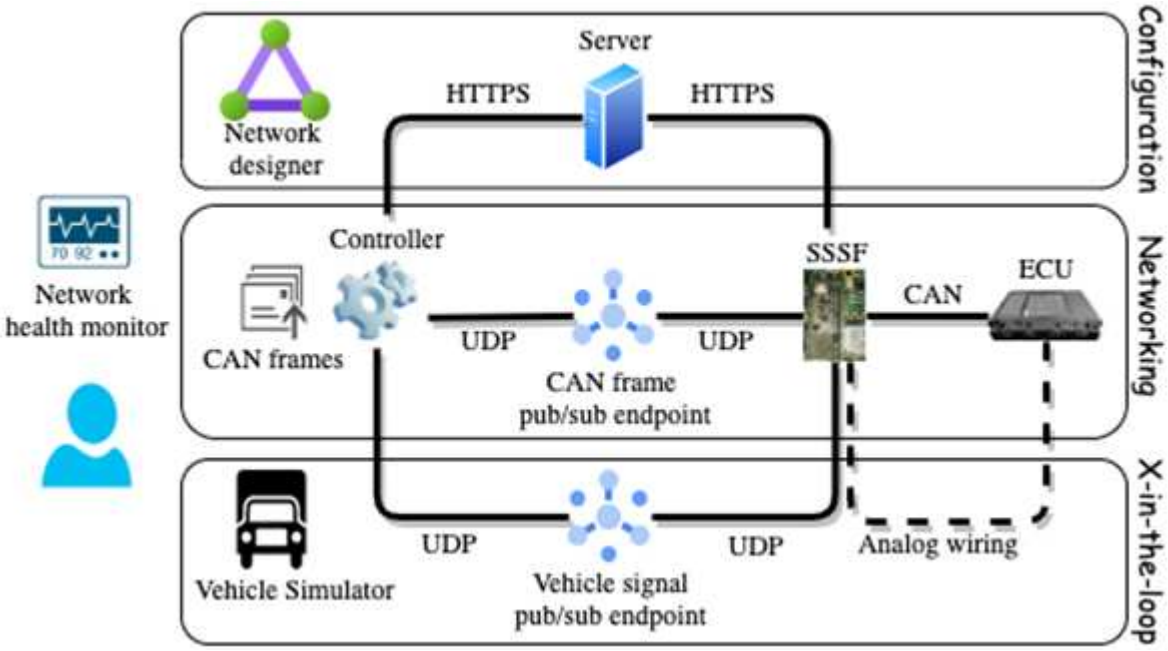


Figure 3.2: Layered System Architecture for CANLay

ECU via User Datagram Protocol (UDP) channels and CAN interfaces. SSSFs forward two message types: one carrying signals from the vehicle simulator, which may eventually be transmitted along the analog wiring shown in Figure 3.2 using a dashed line, and another carrying CAN data between the ECUs in the current experiment. Through SSSFs, multiple ECUs can communicate, creating a rich testing environment. The Smart Sensor Simulator's description and evolution can be found in [52].

Controller

The Controller serves as the user interface to the CANLay system, enabling vehicle simulators to communicate with the CANLay network. The Controller User Interface (UI) assists users in building their virtual test bed by communicating with the central Server via Hypertext Transfer Protocol (HTTP). Once the experiment setup is complete, the Controller transitions to acting as a gateway for a graphical vehicle simulator, forwarding the simulator's outputs to the publish/subscribe (pub/sub) endpoint and listening for CAN messages.

The Controller, a multithreaded graphical application built using Python, exposes two UI components: the Network Designer and the Network Health Monitor. The Controller manages the Vehicle Simulator's execution, updates the Network Designer's device catalog, and relays users' virtual network designs to the Server. Upon session initiation, the Controller oversees the aggregation and presentation of network health reports, collecting current statistics from all devices in the session and displaying results using heat matrices.

Server

The Server assists in setting up publishers and subscribers for an experiment. Each device maintains a persistent TCP connection with the Server while participating in the CANLay system. After establishing the TCP connection, devices communicate with the Server via HTTP Application Programming Interfaces (API). This mechanism enables the Server to track free SSSFs and pub/sub endpoints, validate new experiment requests, and allocate requested devices and endpoints without encountering race conditions or double-use issues. The Server also monitors ongoing sessions and ensures proper session closure when a device encounters problems.

The Server is a single-threaded session broker that multiplexes HTTP API call handling. Devices perform setup and teardown actions such as registering, deregistering, starting, and stopping a session by sending HTTP requests to different API endpoints. The Server responds using standard HTTP response messages and codes interpretable by devices and users.

Publish/Subscribe Endpoints

UDP connects publishers and subscribers in the CANLay system. The pub/sub model was chosen because it easily emulates the broadcast nature of CAN [53]. The main criteria for selecting a suitable pub/sub mechanism resembling a CAN network was a transport mechanism supporting a many-to-many relationship without significant duplication overhead. UDP multicasting is simple, mature, and suitable, although multicasting outside a local network may incur increased implementation costs. Other potential pub/sub implementations, such as MQTT, may also achieve pub/sub system goals [19].

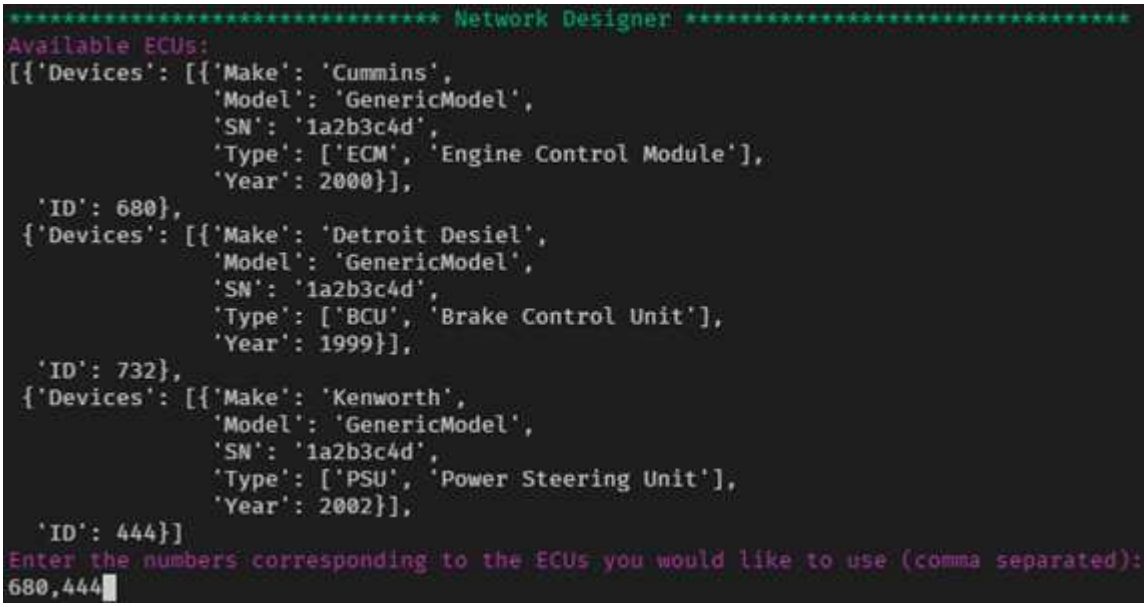


Figure 3.3: Example in the Network Designer displaying available ECUs

Front-End Components

As shown in Figure 3.3, the Network Designer consists of a device catalog and the ability to select requested devices. Upon startup, the Controller contacts the Server and requests the set of available devices. The Controller then presents these options to the user, allowing them to select requested devices from the catalog of available ECUs. The network designer is controlled through the command line but in the future, a GUI could be built to make the system more user-friendly.

3.1.2 CANLay Operation

Network Setup

Figure 3.4 illustrates the network setup process. When the server is operational, SSSFs connect and initiate a setup procedure by reading their built-in SD card. The connected ECU's type, year, make, model, and serial number are obtained from a predefined file on the SD card. The SSSF gathers its MAC address and list of attached devices and sends this information as a JSON to the server via the HTTP API endpoint SSSF/Register.

If registration fails, the server responds with an HTTP 4XX error code, indicating the reason for failure. If successful, the server responds with an HTTP 202 code. The SSSF then awaits further

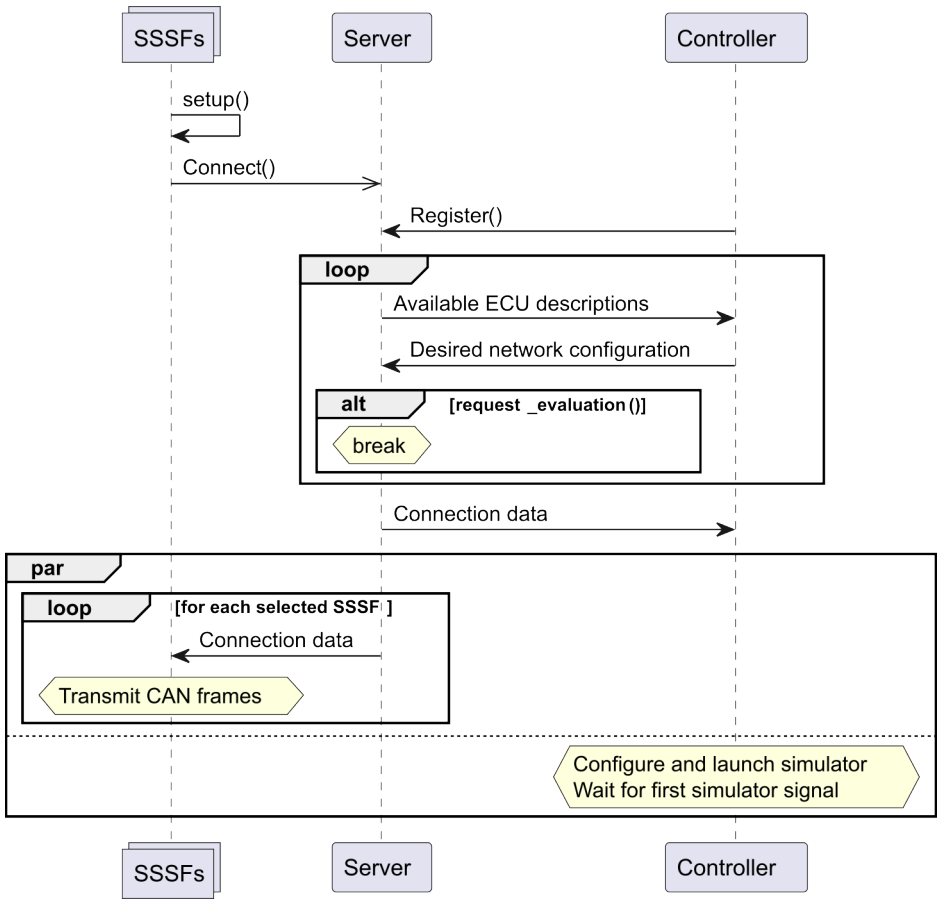


Figure 3.4: Network Setup Activity Sequence Diagram

instructions. The controller registers with the server similarly to the SSSF, only sending its MAC address. After successful registration, the controller requests a list of available ECUs and presents them to the user via the Network Designer. The user finalizes their selection, and the controller sends the selected devices to the server via an HTTP POST.

The server verifies the request, checks the availability of the devices, selects a vacant pub/sub endpoint, and assigns an index to each device. It then sends connection data to the controller and SSSFs. Connection data contains a unique identifier (UID), the index, a multicast IP address and port acting as the pub/sub endpoint, and a list containing the IDs and attached devices of other nodes in the experiment. Upon receiving the connection data, the endpoints synchronize time with Simple Network Time Protocol (SNTP), allocate space for data structures, and begin listening

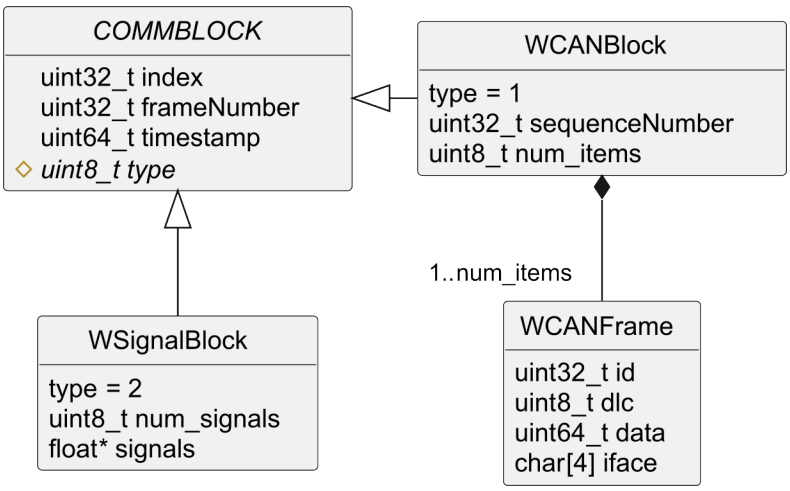


Figure 3.5: Communication Data Structures

for and forwarding messages to and from the pub/sub endpoint. The experiment setup is then complete.

Sensor Signal Communication

Figure 3.5 displays the hierarchy of communications. The communications block creates a template for either CAN messages or messages for signals. The `WCANBlock` comprises one or more CAN frames.

Upon establishing an active session, the controller starts forwarding sensor signals to the pub/-sub endpoint. Before sending the sensor signals, the controller wraps them in a `WSensorBlock` and then a `COMMBlock`. The `COMMBlock` requires several additional pieces of information, but the focus here is on the type and frame number. The type indicates the subclass that the `COMMBlock` is carrying. In this case, the type will be 2, signifying that it is transporting a `WSensorBlock`. The frame number is added to the `COMMBlock` and incremented by one every time the controller forwards a sensor message to the CANLay network. When SSSFs send out CAN messages, they set their outgoing `COMMBlock`'s frame number to the last received frame number from the controller. When the controller receives a CAN message, it reads the frame number field and determines the latest sensor message that the SSSF received. This technique creates an acknowledgment feedback loop, enabling the controller to discern which devices have received

a frame and when a frame was lost. This acknowledgment feedback loop is possible because CAN messages are sent at a higher rate than the sensor messages.

The benefit of this acknowledgment mechanism is that it does not require additional replies from the receiver. Before starting the session, the controller allows the user to select the maximum number of retransmissions, which is factored into the timeout value of a sensor frame.

The timeout value is calculated as follows:

$$\texttt{timeout} = \frac{1}{\texttt{simulator\_frame\_rate} \times \texttt{max\_retransmissions}} \tag{3.1}$$

When the controller detects that the last frame has timed out, it first checks if it has reached its maximum number of retransmissions. If not, it resends the sensor message, increments the number of retransmissions, and resets the timeout timer. This process repeats until the controller receives a CAN message with a frame number equal to its current frame number or reaches the maximum number of retransmissions. In the latter case, the frame is considered lost, and a counter (`lost_simulator_frames`) is updated to display to the user later.

When an SSSF receives a sensor frame, it first sets its last seen frame number equal to the message's frame number. Next, the SSSF applies any necessary transformations to the sensor frame before forwarding it onto its device's CAN network(s). For this thesis, the transformations served mainly as a proof of concept and were kept simple, often sending just the raw sensor value with the closest matching CAN frame.

CAN Communication

When the SSSF participates in an active experiment, it attempts to read a message from the CAN network. Upon doing so, it creates the `COMMBlock` data structure to be written to the CANLay network, as seen in Figure 3.7.

Additional information is added to the `COMMBlock` before it is written to the network. Several pieces of information are required, but for now the focus is on type and sequence number. The type indicates the subclass that the `COMMBlock` is carrying. In this case, the type will be 1,

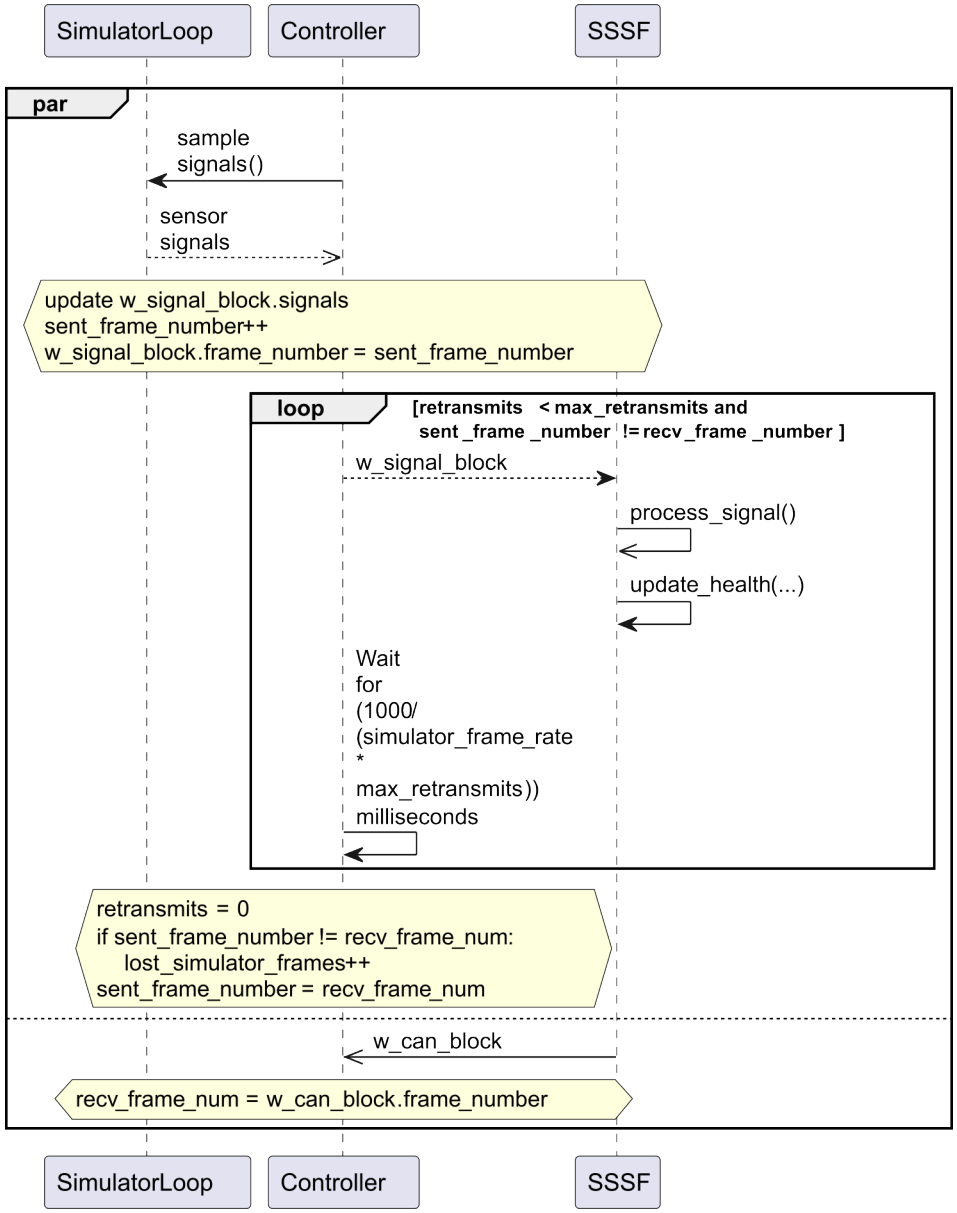


Figure 3.6: SignalTransmission Activity

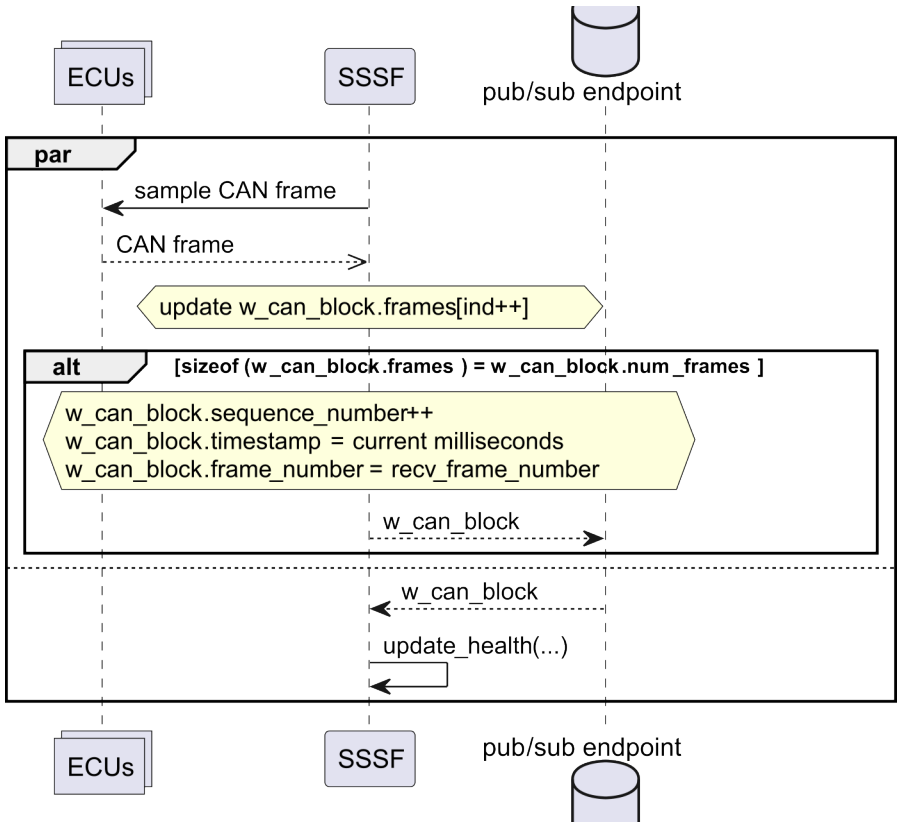


Figure 3.7: Controller Area Network Activity

implying that it is carrying a `WCANBlock`. The sequence number is added to the `WCANBlock` and incremented every time a CAN message gets sent from the device. The packet loss mechanism described later depends on this sequence number. After the SSSF finishes checking for CAN messages from the CAN network, it checks for messages from the pub/sub network. Upon receiving a CAN message wrapped in the `w_can_block`, the SSSF updates its network statistics, extracts the `WCANFrame`, and writes it onto its available CAN networks.

Network health monitoring

Four network statistics for every device on the network include packet loss, latency, jitter, and goodput. Packet loss is the number of packets determined to be lost along a network edge. Latency is the time it takes for a message to travel from the sender to the receiver. Jitter is the variance in latency. The simplest form of network jitter measure, often called packet jitter or constant jitter, is "the variation in latency as measured in the variability over time of the end-to-end delay across

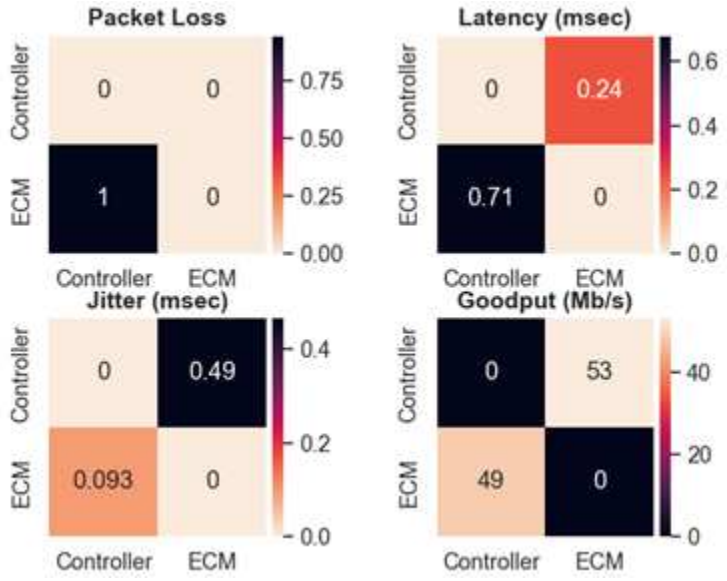


Figure 3.8: Network Matrices showing packet loss, latency, jitter, and goodput

a network" [54]. Lastly, goodput is the measurement of application-level throughput and is often displayed in megabits per second.

As seen in Figure 3.8, the Network Health Monitor displays real-time network statistics that describe the current state of the network. These statistics are presented using heat matrices. Each cell in the matrix represents a directed communication channel of the network. The color of the cells depends on their values. For packet loss, latency, and jitter, lower numbers are better and have lighter colors. For goodput, higher numbers are better and have lighter colors. The contrast between light and dark colors allows the user to quickly identify underperforming parts of the network.

Monitoring the network's health is crucial to ensure that unnecessary delays and packet loss do not adversely affect the experiment's output. Each message structure is loaded with additional information enabling the devices to collect network statistics during an active experiment. The first piece of additional information is a frame or sequence number. When other devices spot gaps in these numbers greater than one, they know that a message was lost. In addition, the timestamp gets included to allow devices to calculate the latency along the network edge from the sending device to the receiving device without interrupting the testing.

Figure 3.9: CANLay in CARLA

3.2 Example CANLay Use Cases

3.2.1 CARLA Simulation

Figure 3.9 illustrates CANLay in operation, with windows displaying CAN frames on the left, the vehicle simulator on the bottom right, and CANLay's network health monitoring on the top right. The vehicle simulator employs CARLA for graphics and vehicle simulation [55].

Although CARLA primarily focuses on autonomous driving research, it provides access to in-game signals through a user-friendly Python API and accurately represents scientific details within the simulator. The realistic and precise representation of signals simplifies their conversion into CAN messages. In this example, a specific CAN frame, defined by the SAE-J1939 standards [56], displays engine parameters transmitted by an engine control module (ECM). The CAN frame's data bytes are shown, with the third byte, representing the driver's throttle demand percentage, changing. This value is also non-zero in the CARLA simulator's frontend.

The network health monitoring window, located in the top right, displays four matrices representing packet loss, latency, jitter, and goodput metrics for network health estimation. The previous section detailed the significance and calculation methods for these metrics. In this case, the experiment was conducted over a gigabit local area network with a Layer 3 switch connecting an SSSF

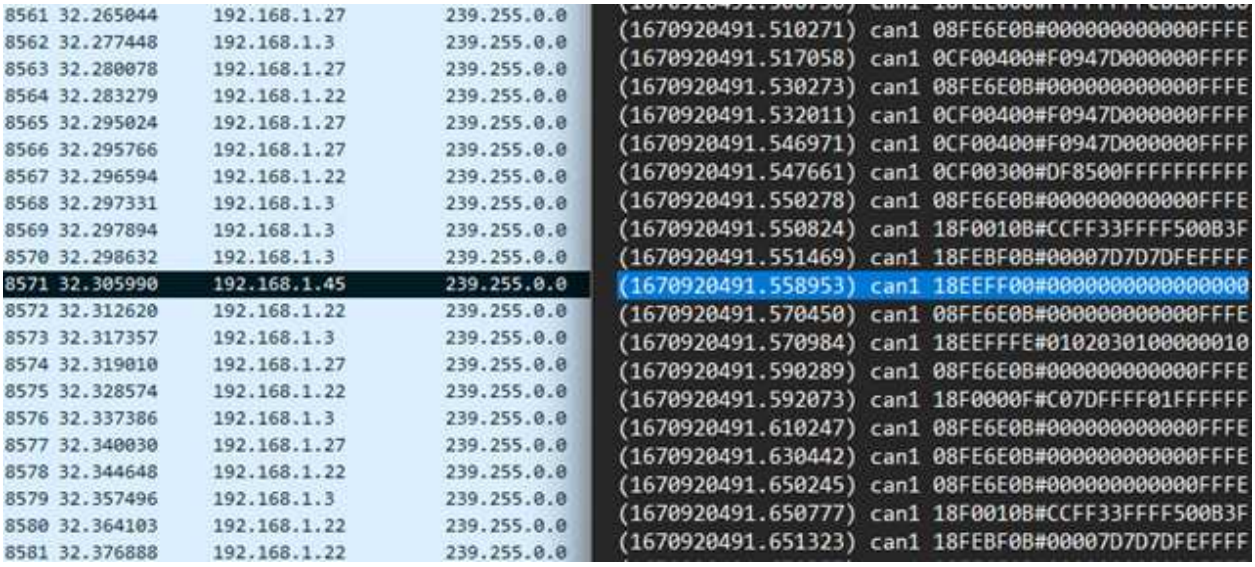


Figure 3.10: CANLay Used to Perform Address Claim Attack

and a Controller. Each `WCANBlock` contains one CAN frame. The figure reveals no packet loss, and the last health report cycle had a latency of less than a millisecond between endpoints.

While CANLay does not explicitly perform latency reduction, test scenarios showed that SSSFs and Controllers forward information in sub-milliseconds. Prior researchers observed similar results when designing CAN-to-Ethernet gateways [46]. Consequently, the delay stems solely from the overlay transmission. A latency of less than a millisecond is sustainable for seamless CARLA emulation at standard frame rates. In this example, the CARLA emulation frame rate was set at 60 frames per second. Jitter is low compared to latency, and the goodput, or application data rate, is higher for the ECM due to the larger number of frames transmitted compared to the controller.

3.2.2 SAE J1939 Address Claim Attack

SAE J1939 enables ECUs to claim addresses for network communication. An attacker can claim an address by sending an address claim message with specific data. Upon processing the received message, the target ECU relinquishes its address(es), rendering it inoperative on the network.

To conduct the address claim attack using CANLay the setup was as follows. First, a Caterpillar ECU was connected through an SSS1 to a CAN-to-Ethernet device. The Smart Sensor Simulators

(SSS) 1 was discussed earlier, but in this case, it acts as a forwarder of ECU traffic and power supply to the ECU. In this case, the SSS1 and CAN-to-Ethernet device combination functioned as the SSSF. A Bendix brake controller and a malicious node (an SSSF sending a malicious message) were also connected to the CANLay network.

The operator initiated a controller instance, chose the devices listed above in the Network Designer, and started the session. Subsequently, the malicious node executed the attack by sending an address claim message to claim the engine controller's address on the network. The engine controller was observed relinquishing its address, and all transmissions from it ceased.

Shown in Figure 3.10 above, the image is split between the UDP Multicast Wireshark trace on the left and the CAN log on the right. Multiple IP addresses communicate with the multicast address 239.255.0.0 on the left. The IP addresses are assigned as follows:

- 192.168.1.22 - Controller sending CARLA frames. The SSSF devices were instructed to not translate the CARLA frames to CAN messages for this experiment.
- 192.168.1.27 - Caterpillar ECU + SSS1.
- 192.168.1.3 - Bendix Brake Controller.
- 192.168.1.45 - Malicious Node.

On the right, the malicious message is highlighted. Before malicious messages, messages from the engine controller (ending in x00) can be observed. After the malicious messages, no messages from the engine controller are observed.

Chapter 4

Conclusion

4.1 Conclusion and Future Work

This research has made significant contributions to the field of trucking technology and cybersecurity, addressing both the vulnerabilities in existing systems and proposing innovative solutions for future developments. The key contributions of this thesis include:

- Identification and demonstration of significant vulnerabilities in Electronic Logging Devices (ELDs), a mandated technology in the trucking industry.
- Development and testing of a proof-of-concept Truck to Truck Worm, highlighting the practical risks and potential impacts of these vulnerabilities.
- Proposal of practical and effective recommendations for enhancing ELD security, considering constraints of cost, reliability, and user-friendliness.
- Introduction of the Software Defined Truck (SDT) concept as a comprehensive platform for testing and analyzing integrated vehicle systems.
- Development of CANLay, a key networking component enabling the realization of the SDT concept, facilitating the transportation of CAN data over long-distance networks.

The study began by illuminating significant vulnerabilities in ELDs. Through testing, both in controlled settings and in real-world environments, this research demonstrated the practical risks and potential impacts of a Truck-to-Truck Worm facilitated by these devices.

The findings from this study highlight the importance of security in technologies that are not only integral to operational efficiency but also legally mandated. The vulnerability of such systems poses a broader risk to the entire supply chain, making it imperative that security measures evolve in tandem with technological advancements.

Recommendations for bolstering ELD security, such as optimizing default security settings, ensuring firmware integrity, and limiting unnecessary API features, were designed to be practical and effective, considering the constraints of cost, reliability, and user-friendliness. These steps are crucial for mitigating the risks identified and setting a foundation for more secure operations.

However, this research recognized that addressing vulnerabilities in individual systems like ELDs does not solve the root issue. The fundamental challenge lies in the fact that ELDs, along with many other after-market systems, are designed, tested, and secured separately from the trucks they are installed in. This separation fails to account for the emergent behaviors, particularly in terms of security, that arise when these systems are integrated.

SDT concept directly address this challenge by providing a comprehensive platform for testing and analyzing integrated vehicle systems. By enabling the simulation of various truck configurations and the integration of multiple after-market systems within a virtualized environment, SDT allows researchers and developers to:

- Observe and study the emergent behaviors that occur when different systems interact within the truck's network.
- Conduct holistic security assessments that consider the entire ecosystem of the truck, rather than individual components in isolation.
- Identify potential vulnerabilities and security risks that may only become apparent when systems are integrated.
- Test and validate security measures in a controlled, yet realistic environment that mimics the complexity of actual truck systems.
- Iterate quickly on different configurations and scenarios without the need for physical hardware changes, significantly reducing the cost and time required for comprehensive testing.

This approach enables a more thorough understanding of the security implications of integrating various systems within a truck, allowing for the development of more robust and holistic security measures that account for the complexities of the entire system.

Until now, SDT lacked the technological backing to acheive the above objective. Therefore, this research introduced CANLay, a key networking component that enables the realization of the SDT concept. CANLay serves as the networking backbone for SDT, facilitating the transportation of CAN data over long-distance networks. By enabling network virtualization for SDT, CANLay addresses several system goals:

- Transporting CAN frames and sensor signals over a distributed overlay network of electronic control units
- Supporting on-demand session creation
- Providing runtime metrics to estimate network health during ongoing experiments

While CANLay does not explicitly ensure reliability and low latency like CAN, it offers a health monitoring service that delivers real-time measures of network parameters to users. In doing so, the system meets its requirements for network virtualization, real-time health monitoring, scalability, compatibility with existing CAN-based communication protocols, and a user-friendly interface.

Although CANLay effectively satisfies its goals and requirements, three potential enhancements have been identified for further improvement:

- Enabling the routing of actuation signals from the ECU to the vehicle simulator for visualization of ECU actuation effects.
- Implementing a selective acknowledgment scheme for CAN frames, enforcing acknowledgment for critical user-identified frames.
- Introducing a dynamic buffer adjustment scheme when accumulating CAN frames in a `WCANBlock` to optimize available bandwidth usage while supporting high CAN busloads.

4.2 Future Work

Looking ahead, it is clear that continuous innovation and vigilance in cybersecurity are essential, especially for technologies mandated by regulatory bodies. Furthermore, regulating bodies need to be aware of the increased security risks associated with mandated technologies that interface with deployed control networks.

Future research should focus on:

- Developing and implementing advanced, adaptable security measures that can protect against evolving threats while ensuring seamless operational integration.
- Exploring the full potential of the SDT concept and CANLay for comprehensive testing and research in the heavy vehicle industry.
- Investigating ways to enhance the flexibility and capabilities of CANLay, including the potential enhancements identified in this research.
- Studying the broader implications of integrated vehicle systems on cybersecurity and developing strategies to address emergent vulnerabilities.
- Collaborating with industry stakeholders and regulatory bodies to establish comprehensive security standards that account for the complexities of integrated vehicle systems.

In conclusion, this research not only illuminates current vulnerabilities and proposes immediate solutions but also paves the way for future innovations in trucking technology. By addressing both immediate security concerns and developing forward-looking experimental frameworks like CANLay, this work sets a foundation for more holistic approaches to security, compliance testing, and system integration in the trucking industry. The balance between security, functionality, and regulatory compliance remains vital for safeguarding the trucking industry and, by extension, the critical supply chains it supports.

Bibliography

[1] United States. Department of Transportation. Bureau of Transportation Statistics, *National Transportation Statistics (NTS)*, en, 2019. DOI: 10.21949/1503663. [Online]. Available: https://tinyurl.com/rosapntlbtsNTS (visited on 07/23/2024).

[2] Costello, Bob, *Economics and Industry Data*. [Online]. Available: https://www.trucking.org/economics-and-industry-data.

[3] Federal Motor Carrier Safety Administration, "Electronic Logging Devices and Hours of Service Supporting Documents," Department of Transportation, Tech. Rep. FMCSA–2010–0167, Dec. 2015. [Online]. Available: https://www.govinfo.gov/content/pkg/FR-2015-12-16/pdf/2015-31336.pdf (visited on 11/30/2023).

[4] Todd Dills, *ELD protests Day 2: Truckers roll in, stage along rigs in front of DOT headquarters*, Oct. 2017. [Online]. Available: https://www.overdriveonline.com/electronic-logging-devices/article/14893192/eld-protests-day-2-truckers-roll-in-stage-along-rigs-in-front-of-dot-headquarters.

[5] Federal Motor Carrier Safety Administration, *Hours of Service of Drivers (2011 Revision)*, 2011.

[6] Society of Automotive Engineers, *J1939-13 Off-Board Diagnostic Connector*.

[7] Technology and Maintenance Council, *RP1210D*.

[8] Robert BOSCH GmbH, *CAN Specification Version 2.0*, 1991.

[9] Society of Automotive Engineers, *SAE J1939 Standards Collection*. [Online]. Available: https://www.sae.org/standardsdev/groundvehicle/j1939a.htm (visited on 11/09/2021).

[10] C. Miller and C. Valasek, *Remote Exploitation of an Unaltered Passenger Vehicle*, en, Aug. 2015. [Online]. Available: https://illmatics.com/Remote%20Car%20Hacking.pdf.

[11] Y. Burakova, B. Hass, L. Millar, and A. Weimerskirch, "Truck Hacking: An Experimental Analysis of the SAE J1939 Standard," en, in *Proceedings of the 10th USENIX Conference on Offensive Technologies*, ser. WOOT'16, USENIX Association, 2016. [Online]. Available: https://www.usenix.org/system/files/conference/woot16/woot16-paper-burakova.pdf.

[12] P.-S. Murvay and B. Groza, "Security Shortcomings and Countermeasures for the SAE J1939 Commercial Vehicle Bus Protocol," en, *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 4325–4339, May 2018, ISSN: 0018-9545, 1939-9359. DOI: 10.1109/TVT.2018.2795384. [Online]. Available: https://ieeexplore.ieee.org/document/8263125/ (visited on 04/20/2023).

[13] M. T. Campo, S. Mukherjee, and J. Daily, "Real-Time Network Defense of SAE J1939 Address Claim Attacks," en, *SAE International Journal of Commercial Vehicles*, vol. 14, no. 3, pp. 02–14–03–0026, Aug. 2021, ISSN: 1946-3928. DOI: 10.4271/02-14-03-0026. [Online]. Available: https://www.sae.org/content/02-14-03-0026/ (visited on 12/17/2023).

[14] S. Mukherjee, H. Shirazi, I. Ray, J. Daily, and R. Gamble, "Practical DoS Attacks on Embedded Networks in Commercial Vehicles," in *International Conference on Information Systems Security*, ser. LNCS 10063, Jaipur, Rajasthan, India: Springer International Publishing, 2016, pp. 23–42. DOI: 10.1007/978-3-319-49806-52. [Online]. Available: https://www.cs.colostate.edu/~cs557/papers/MSR+16.pdf.

[15] R. Chatterjee, S. Mukherjee, and J. Daily, "Exploiting Transport Protocol Vulnerabilities in SAE J1939 Networks," en, in *Proceedings Inaugural International Symposium on Vehicle Security & Privacy*, San Diego, CA, USA: Internet Society, 2023, ISBN: 978-1-891562-88-4. DOI: 10.14722/vehiclesec.2023.23053. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2023/02/vehiclesec2023-23053-paper.pdf (visited on 04/19/2023).

[16] C. Miller and C. Valasek, *A Survey of Remote Automotive Attack Surfaces*, en, 2014. [Online]. Available: https://ioactive.com/wp-content/uploads/2018/05/IOActive_Remote_Attack_Surfaces.pdf.

[17] S. Checkoway, D. McCoy, B. Kantor, *et al.*, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," en, in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11, San Francisco, CA, USA: USENIX Association, 2011, p. 6. DOI: 10.5555/2028067.2028073.

[18] K. Koscher, A. Czeskis, F. Roesner, *et al.*, "Experimental Security Analysis of a Modern Automobile," en, in *2010 IEEE Symposium on Security and Privacy*, Oakland, CA, USA: IEEE, 2010, pp. 447–462, ISBN: 978-1-4244-6894-2. DOI: 10.1109/SP.2010.34. [Online]. Available: http://ieeexplore.ieee.org/document/5504804/ (visited on 12/13/2022).

[19] AiDA, *Digital Enginineering*. [Online]. Available: https://aida.mitre.org/digital-engineering/.

[20] Department of Defense, *Digital Engineering Strategy*, Jun. 2018. [Online]. Available: https://ac.cto.mil/wp-content/uploads/2019/06/2018-Digital-Engineering-Strategy_Approved_PrintVersion.pdf.

[21] D. D. Walden, G. J. Roedler, K. J. Forsberg, R. D. Hamelin, and T. M. Shortell, *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, 4th ed.* 4th. John Wiley & Sons, Inc., 2015, ISBN: 978-1-118-99940-0. [Online]. Available: https://www.wiley.com/en-us/INCOSE+Systems+Engineering+Handbook%3A+A+Guide+for+System+Life+Cycle+Processes+and+Activities%2C+4th+Edition-p-9781118999400.

[22] ISO/IEC/IEEE, *ISO/IEC/IEEE 15288:2023 Systems and software engineering — System life cycle processes*, May 2023. [Online]. Available: https://www.iso.org/standard/63711.html.

[23] S. Mukherjee and J. Daily, "Towards a Software Defined Truck," en, *INCOSE International Symposium*, vol. 31, no. 1, pp. 1019–1034, Jul. 2021, ISSN: 2334-5837, 2334-5837. DOI:

10.1002/j.2334-5837.2021.00884.x. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/j.2334-5837.2021.00884.x (visited on 04/19/2023).

[24] D. Klinedinst and C. King, "On Board Diagnostics: Risks and Vulnerabilities of the Connected Vehicle," en, Software Engineering Institute Carnegie Mellon University, CERT Coordination Center, Tech. Rep. DM-0003466, Mar. 2016.

[25] Corey, Theun, *Heavy Truck and Electronic Logging Device: What Could Go Wrong*, DEF CON 25 Car Hacking Village, 2017. [Online]. Available: https://media.defcon.org/DEF%20CON%2025/DEF%20CON%2025%20villages/DEF%20CON%2025%20Car%20Hacking%20Village%20-%20Corey%20Theun%20-%20Heavy%20Truck%20and%20Electronic%20Logging%20Devices-%20What%20Could%20Go%20Wrong.mp4 (visited on 11/30/2023).

[26] Federal Motor Carrier Safety Administration, *ELD List*. [Online]. Available: https://eld.fmcsa.dot.gov/List.

[27] Espressif Systems, *Xtensa Instruction Set Architecture (ISA) Reference Manual*, en.

[28] Espressif Systems, *ESP32 IDF Github*, Github. [Online]. Available: https://github.com/espressif/esp-idf/.

[29] Kai Ren, *Bluetooth Pairing Part 2 Key Generation Methods*, Jun. 2016. [Online]. Available: https://www.bluetooth.com/blog/bluetooth-pairing-part-2-key-generation-methods/.

[30] Espressif Systems, *ESP32 Technical Reference Manual*. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf.

[31] Espressif Systems, Fredrik, Ahlberg, and Angus Gratton, *ESPTool*, Github. [Online]. Available: https://github.com/espressif/esptool.

[32] Skylot, *JADX*, Github. [Online]. Available: https://github.com/skylot/jadx.

[33] GNU Operating System, *GNU Binutils*. [Online]. Available: https://www.gnu.org/software/binutils/.

[34] National Security Agency, *Ghidra*. [Online]. Available: https://ghidra-sre.org/.

[35] Dynacylabs, Austin Tyler Conn, Ebiroll, and Yath, *Ghidra-Xtensa*, Github. [Online]. Available: https://github.com/dynacylabs/ghidra-xtensa.

[36] Leveldown Security, *SVD Loader*, Github. [Online]. Available: https://github.com/leveldown-security/SVD-Loader-Ghidra.

[37] Gordon Lyon, *Nmap*. [Online]. Available: https://nmap.org/.

[38] Macchina, *Macchina M2 (Under-the-Dash) Automotive Interface*. [Online]. Available: https://www.macchina.cc/catalog/m2-boards/m2-under-dash.

[39] National Motor Freight Traffic Association, Inc., *Cybersecurity Requirements for Telematics Systems*, Dec. 2023. [Online]. Available: https://nmfta.org/wp-content/media/2022/11/NMFTA-Cybersecurity-Requirements-for-Telematics-Systems-v1.5.pdf.

[40] National Motor Freight Traffic Association, Inc., *Vehicle Network Gateway Devices Security Requirements*, 2024. [Online]. Available: https://nmfta-repo.github.io/vcr-experiment/vcr-experiment/01_gateways.html.

[41] S. Mahmood, H. N. Nguyen, and S. A. Shaikh, "Automotive Cybersecurity Testing: Survey of Testbeds and Methods," en, in *Digital Transformation, Cyber Security and Resilience of Modern Societies*, T. Tagarev, K. T. Atanassov, V. Kharchenko, and J. Kacprzyk, Eds., vol. 84, Series Title: Studies in Big Data, Cham: Springer International Publishing, 2021, pp. 219–243, ISBN: 978-3-030-65721-5 978-3-030-65722-2. DOI: 10.1007/978-3-030-65722-2_14. [Online]. Available: https://link.springer.com/10.1007/978-3-030-65722-2_14 (visited on 04/19/2023).

[42] R. Rotermund, T. Häckel, P. Meyer, F. Korf, and T. C. Schmidt, "Requirements Analysis and Performance Evaluation of SDN Controllers for Automotive Use Cases," in *2020 IEEE Vehicular Networking Conference (VNC)*, New York, NY, USA: IEEE, 2020, pp. 1–8. DOI: 10.1109/VNC51378.2020.9318378. [Online]. Available: https://ieeexplore.ieee.org/document/9318378.

[43] M. Doering and M. Wagner, "Retrofitting SDN to classical in-vehicle networks: SDN4CAN," en, Universitat Tubinge, p. 2. DOI: 10.15496/publikation-19541. [Online]. Available: http://dx.doi.org/10.15496/publikation-19541.

[44] D. Grewe, N. Nayak, D. Babu, W. Chen, S. Schildt, and C. Schroff, "BloomyCAN: Probabilistic Data Structures for Software-defined Controller Area Networks," in *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*, IEEE, 2021, pp. 1–6.

[45] M. Johanson, L. Karlsson, and T. Risch, "Relaying Controller Area Network Frames over Wireless Internetworks for Automotive Testing Applications," in *2009 Fourth International Conference on Systems and Networks Communications*, IEEE, 2009, pp. 1–5. DOI: 10.1109/ICSNC.2009.68. [Online]. Available: https://ieeexplore.ieee.org/document/5279400.

[46] Florian Polzlbauer and Allan Tengg, "Experience Report: Lightweight Implementation of a Controller Area Network to Ethernet Gateway," in *Proceedings of the Brief Presentation Track of the RTAS'19 Conference*, ser. Brief Presentations Proceedings, RTAS 2019, Montreal, Canada: IEEE, Apr. 2019, pp. 7–10. [Online]. Available: https://2019.rtas.org/wp-content/uploads/2019/04/RTAS19_BP_proceedings.pdf.

[47] M. Appel, P. S. Oruganti, Q. Ahmed, J. Wilkerson, and R. Sekar, "A Safety and Security Testbed for Assured Autonomy in Vehicles," en, Apr. 2020, pp. 2020–01–1291. DOI: 10.4271/2020-01-1291. [Online]. Available: https://www.sae.org/content/2020-01-1291/ (visited on 04/19/2023).

[48] Vector, *Scalable and Modular Systems for Efficient HIL Testing*. [Online]. Available: https://www.vector.com/int/en/products/products-a-z/hardware/vt-system/.

[49] dSpace, *Hardware-in-the-Loop Testing*. [Online]. Available: https://www.dspace.com/en/inc/home/applicationfields/foo/hil-testing.cfm.

[50] National Instruments, *HIL and Real-Time Test Software Suite*. [Online]. Available: https://www.ni.com/en-us/shop/software/products/hil-and-real-time-test-software-suite.html.

[51] ETAs, *Welcome to ETAs*. [Online]. Available: https://www.etas.com/en/.

[52] Ram Rohit Gannavarapu, "SECURE REMOTE SENSOR SIMULATOR FOR HEAVY VEHICLE ELECTRONIC CONTROL UNITS," Publication Title: M.S. Thesis for Electrical and Computer Engineering at Colorado State University, Colorado State University, Colorado, Dec. 2021.

[53] J. Kaiser and M. Mock, "Implementing the real-time publisher/subscriber model on the controller area network (CAN)," in *Proceedings 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'99) (Cat. No.99-61702)*, ser. ISORC, Saint-Malo, France: IEEE, 1999, pp. 172–181. DOI: 10.1109/ISORC.1999.776373. [Online]. Available: https://ieeexplore.ieee.org/document/776373.

[54] Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta, *OASIS MQTT Version 5.0*, Mar. 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf.

[55] A. Dosovitskiy, German, Ros, Felipe, Codevilla, Antonio Lopez, and Vladlen Koltun, "CARLA: An Open Urban Driving Simulator," en, in *Proceedings of the 1st Annual Conference on Robot Learning*, vol. 78, PMLR, 2017, pp. 1–16. [Online]. Available: https://proceedings.mlr.press/v78/dosovitskiy17a.html.

[56] Network Encyclopedia, *Jitter*, Aug. 2019. [Online]. Available: https://networkencyclopedia.com/jitter/.