Patching an Engine Control Module to Enhance Security

Abstract

The architecture of Control Area Network (CAN)-based protocols offers straightforward, centralized, and cost-efficient methods for various Electronic Control Units (ECUs) to communicate via the CAN bus. However, the CAN protocol was not designed with security as a priority. The CAN bus used in vehicles lacks built-in security features, (i.e., messages are broadcast without authentication or encryption). This makes CAN vulnerable to eavesdropping, spoofing, and replay attacks. Any compromised node can inject false messages (e.g., impersonating the brakes or engine controller) with no cryptographic checks to stop it. The protocol's primary integrity safeguard, a Cyclic Redundancy Check (CRC), was designed solely for detecting transmission errors and is easily manipulated, offering no real protection against malicious adversaries. Implementing cryptography on CAN is challenging due to CAN's most popular limited 8-byte data payload, real-time latency requirements, and the need for compatibility with millions of existing CAN devices. To address this issue, this study focuses on developing a Cryptographic Message Authentication Code (CMAC) Intrusion Detection System (IDS) implementation for CAN bus communication using existing ECUs.

1. Introduction

The security risks associated with CAN have increased as automotive systems transition from physical network connections to wireless interfaces. Previously, attacks required direct physical access and were difficult to execute at scale across multiple vehicles or fleets. However, with the increasing integration of wireless technologies in modern vehicles, these attacks have become more scalable and widespread.

To address this issue, this study focuses on developing a Cryptographic Message Authentication Code (CMAC) Intrusion Detection System (IDS) implementation for CAN bus communication using existing ECUs. A process was successfully developed for installing a modified executable binary on an Engine Control Module, with its functionality verified and the added CAN message handlers demonstrated to identify and act on messages based on 29-bit CAN identifiers. This research presents a practical approach to enhancing CAN bus security in vehicles without requiring additional hardware, ensuring data authenticity and system integrity with minimal performance overhead. In addition, it explores alternative methods for third-party manufacturers and right-to-repair enthusiasts to achieve cost-effective ECU modifications and improve CAN bus security.

The System of Interest (SoI) in this project is a 2014 Kenworth T270 Class 6 Truck shown in Figure 1. It is powered by an ISB 6.7L inline 6 diesel engine with an Allison automated transmission. The proof of concept for this research is an Engine Control Module (ECM), also located on the Kenworth research truck, composed of an NXP MPC5674F processor and four Controller Area Network (CAN) 2.0b channels [1]. In a typical environment, the ECM is programmed over the CAN bus using a diagnostics tool. While modifying an Electronic Control Unit (ECU) is technically possible over the Unified Diagnostic Protocol (UDS) [2], there is finer-grained control over the device when connected directly to the Joint Test Action Group (JTAG) test access port available on the ECM [3].

Figure 1: Kenworth research truck

Furthermore, there are existing studies providing evidence into the CAN protocols vulnerabilities, and in particular, relating to heavy-vehicles, J1939 exploits. The cybersecurity landscape of commercial vehicles includes many vulnerabilities across multiple layers of the SAE J1939 protocol [4]. Research has identified significant weaknesses in the application layer, where attackers can exploit specific J1939 messages to gain continuous control over critical vehicle functions, such as engine braking and accelerator input, posing serious operational risks. At the data-link layer, excessive request messages targeting an ECU can overload processing capacity, leading to denial-of-service scenarios by blocking legitimate requests. Network integrity is also at risk, as address claim message flooding can incapacitate multiple ECUs, while abrupt termination of multi-packet data transfers can disrupt ECU operations [5–7].

For the past several years, researchers have explored various approaches to developing CAN Intrusion Detection Systems (IDS) [8] and cryptographic approaches [9–19]. However, these solutions often require significant modifications to existing infrastructure or the integration of specialized hardware components. In contrast, this project demonstrates a feasible approach to enhancing CAN bus security in vehicles by leveraging pre-existing embedded systems. This research also demonstrates the ability for a wide variety of stakeholders to form customized firmware, whether that's increasing security in compliance with ISO 21434 [20], improving diagnostic capabilities, or enabling enhanced performance monitoring.

Siti-Farhana et al. describe the gateway of modern vehicles to be a critical component in the CAN, serving as a bridge for information exchange between various internal systems and external networks [8]. These gateways facilitate communication across subsystems such as the engine, braking, and telematics, while also enabling connectivity with external interfaces like WiFi, Bluetooth, and USB [21]. Gateways also provide an interface for diagnostics (i.e., UDS), which has been found to have vul-

nerabilities as well [22]. This connectivity introduces security vulnerabilities, as malicious actors can exploit these gateways to compromise vehicle systems. The study emphasizes the importance of securing these gateways to prevent potential attacks and ensure the safety and functionality of modern vehicles.

The threat model of this project encompasses the CAN bus and the connected ECMs within it. Threats considered include: a malicious ECM installed onto the CAN bus, a supply chain attack in which a new node is installed that has malicious code pre-programmed, or a Man-in-The-Middle (MITM) across a CAN bus. For instance, on a J1939 network, an attacker could spoof a Cruise Control/Vehicle Speed 1 message from the engine to change the vehicle [23]. Any node or ECM within a vehicle might behave in a way that disrupts the standard rules for message transmission. Therefore, it is essential to maintain the integrity of messages coming across the CAN bus, while also maintaining confidentiality.

In this sense, this research opts to introduce the process of firmware modification using tools like the Open Firmware Reverse Analysis Konsole (OFRAK) [24], and the Lauterbach Debugger [25], adding a layer of security the the CAN bus, compliant with the RFC 4493 (AES-CMAC) standard, hoping to contribute a means to provide backward-compatibility with all types of ECUs.

2. Related Work

Previous research highlights the viability of using low-level forensic techniques to extract and analyze data from ECMs. Specifically, Daily et al. [26] demonstrated methods for retrieving non-volatile memory contents using both JTAG and chip-level extractions. Their work confirms that flash memory, Electronically Erasable Programmable Read-Only Memory (EEPROM), and processor memory can be accessed and cloned in a forensically sound manner, preserving event data such as sudden deceleration records, engine speed, vehicle speed, and fault code snapshots. These techniques enable direct binary extraction without relying on network-based downloads, thus avoiding potential data loss due to power interruptions or corrupted diagnostic reads.

Additionally, advancements in CAN bus IDS' have predominantly focused on leveraging Machine Learning (ML) techniques [27]. While these methods demonstrate significant potential, their probabilistic nature inherently limits their ability to achieve complete accuracy. Threat actors continuously adapt and develop novel attack vectors, which can outpace the capabilities of ML models trained on static datasets. Adapting and evolving such models demands substantial time and resources [8, 27, 28].

To address these challenges, cryptographic message authentication codes (CMACs) have been explored as a more deterministic alternative. Researchers at Colorado State University proposed a CAN conditioner that integrates a hardware-based security module for generating and verifying CMACs [21]. This system enhances message integrity and authenticity on the CAN bus through unique session keys established via Elliptic-Curve Diffie-Hellman (ECDH) key exchange, RFC 8418. Additionally, it enforces allowlist/blocklist mechanisms to mitigate spoofing attacks and incorporates a secure gateway to monitor and validate node behavior. However, this introduces an external module onto the CAN bus.

Securing automotive CAN buses with cryptography is an active field of research and industry development. Methods like CANAuth, CAN-Crypto (AES encryption), vatiCAN, and others each contribute different ideas – from purely software-based HMAC authentication to hardware-accelerated full encryption. Comparisons show a clear trade-off between security level and system overhead. These studies prove security can be accomplished (preventing or greatly limiting CAN bus attacks) and in some cases maintaining backward compatibility, yet this includes added latency, increased message traffic, and the need for new hardware or complex key management. One survey concisely notes that encryption "hardens attacks" and allows confidential transmission, but at the cost of increased computational load and traffic, and potential weaknesses due to CAN frame size constraints. Likewise, authentication alone secures data integrity but still adds overhead and doesn't protect privacy [29]. Automotive manufacturers are beginning to adopt a mix of these techniques – often using message authentication as a baseline and selective encryption for the most sensitive data. As vehicles continue to incorporate more connectivity, the deployment of robust cryptographic security on CAN (and newer in-vehicle networks like CAN-FD - Flexible Data-rate, Ethernet, etc.) is expected to become the norm, balancing safety and security with practical performance limits.

It's worth noting that some manufacturers have deployed proprietary encryption on their powertrain CAN buses (to prevent aftermarket tuning, for instance), but security through obscurity has proven weak. Researchers have repeatedly cracked such schemes – for example, the encryption on a luxury car's ECU was broken by tuners in a matter of months. This underscores that how cryptography is implemented (key management, algorithm strength, integration) is crucial; simply "encrypting CAN" is not a silver bullet if done improperly [29].

Thomas et al. have advanced methods for efficient malware analysis and firmware reverse engineering [30, 31]. While these techniques provide excellent analytical insights, they often lack a minimally invasive mechanism to modify and reintroduce binaries after analysis.

Building on these efforts, this article presents a novel approach that integrates previous advances in vehicular security with binary modification of the ECM as a proof of concept. It's important to note that researchers have also been able to wirelessly modify binaries through Vehicle Diagnostic Adapters (VDAs) provided by manufacturers to remove, or 'de-feature' wireless interfaces, further improving security with OFRAK [32]. By combining OFRAK with robust live debugging capabilities, this article proposes a systematic process that facilitates efficient and holistic security enhancements, without the need for additional hardware, which could slow, or bottleneck the CAN bus. In addition, verification and validation processes play a key role in the engineering development phase, ensuring that security controls are implemented effectively and tested against adversarial threats. The Systems Engineering Handbook [33] underscores the importance of rigorous risk management and ongoing system evaluation to maintain resilience against evolving cybersecurity challenges. By introducing a concept that provides users with a means to implement security or performance changes early, or later, in the development stage, or production phase, the hope is that this work will inspire further changes in the automotive community towards complete security of road vehicles.

3. Methodology

This section outlines the methodology used by the researchers. The approach described here was specifically designed and implemented to accelerate development processes. The enhanced efficiency was achieved by using the in-circuit debug port on the ECM, which becomes accessible only when the engine controller is disassembled (i.e., the back cover is removed). Without this modification, firmware reading and writing would have been dependent on the CAN bus, a significantly slower alternative.

3.1 Load Factory Calibration

The calibration for an engine controller refers to the executable binary excluding the bootloader. This term is commonly used in the industry to describe the program operating on an ECM. The standard method for loading a calibration (firmware update) onto a ECM is by utilizing the service tool. The steps outlined below detail the process of installing the firmware onto the ECM. This procedure ensures the ECM is properly prepared for this project. While it is possible to conduct the experiment using the original ECM, it is more practical to perform the testing on a spare unit that can be disassembled if needed. The following steps outline the calibration process:

1. Locate the Fleet Calibration file corresponding to the part number indicated on the ECM label (see Figure 2) and load it into the ECM. Data transfer from the service computer to the engine controller is facilitated through an RP1210 vehicle diagnostic adapter.

Figure 2: Engine control module

2. Transfer a parameter template derived from the research truck to configure the ECM firmware. This step establishes the appropriate parameters for the engine's operation within its current system, including governed road speed and other truck-specific configurations and parameters (see Appendix B).

3.2 Confirm the New ECM Calibration Works

In this phase, the newly calibrated ECM was installed in the truck, displayed as Figure 3, and subsequent testing was conducted. The fresh factory calibration successfully started the engine, and the truck appeared to operate as expected. However, a Stop Engine lamp was illuminated on the dashboard with the installation of this new ECM. This issue was not resolved prior to proceeding to the next phase. As a result, the Stop Engine lamp persisted in subsequent tests but was attributed to an incomplete installation process. Importantly, this condition did not impact the quality of the work performed or the overall fulfillment of the project scope.

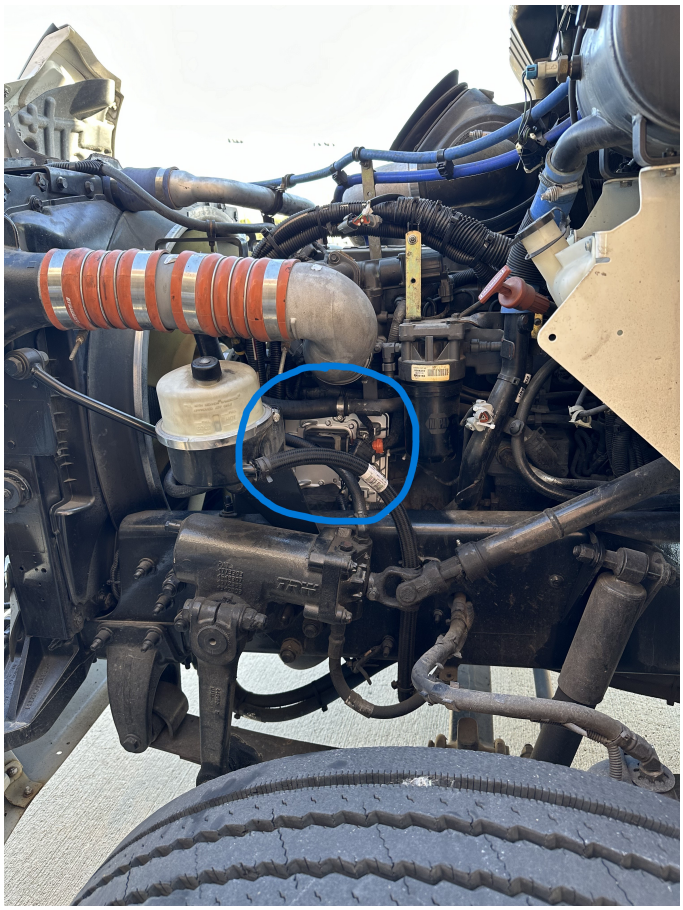

Figure 3: Location of ECM; Circled in blue

3.3 Backup the ECM Firmware

Before modifying anything, make sure to back up the firmware file in case something goes wrong. The Lauterbach Trace32 tools located at the default installation location on a Windows computer at 'C:\T32\bin\windows64\t32mppc.exe' helped assist in quick facilitation in dumping firmware contents.

To run the program, the Lauterbach Trace tool must connect to the Nexus debug port of the ECM. This port is located within the ECM, where only the circuit board pads are exposed. To facilitate this connection, a Samtec QSE-020 connector was soldered onto the ECM PCB, providing the necessary interface for debugging tools.

However, the Samtec connector is not directly compatible with the Lauterbach Nexus Tracing tool and probe. To address this, an adapter board was designed to convert the 40-pin Samtec connector on the ECM to the Mictor-38 connector required by the Lauterbach tool. The adapter board was developed using Altium Designer, with design files that include schematics, project files (compatible with Altium Designer version 24), and the Gerber and NC Drill files necessary for PCB production. This adapter

facilitates the connection between the Lauterbach tool and the ECM debug port, as illustrated in Figure 4.

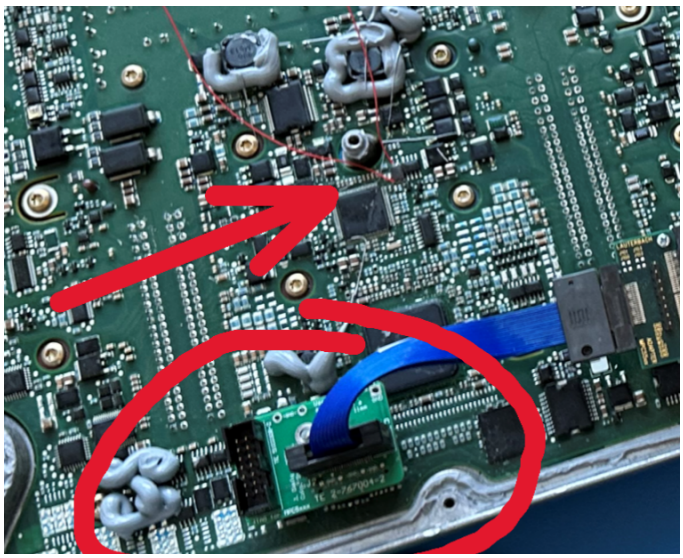

Figure 4: Lauterbach debugger and custom mictor to samtec adapter; Circled in red

To use the Lauterbach system, the hardware watchdog must be soldered together as shown by the red arrow in figure 4. It's important to make sure to disconnect the short when finished because it was found to disrupt the start of the engine.

The flash memory on board the MCP5674F processor is available between addresses `0x00000000-0x003FFFFF` as shown in Appendix A. After utilizing a template provided by Lauterbach [25], a PRACTICE (Praxis) script was further developed to read firmware from the MCP5674F processor in the ECM. The PRACTICE scripting language has C-like syntax with support for debugger control, conditionals, variable handling, read/write of memory, and file I/O. Once a reliable physical connection is made, the scripted commands with the Lauterbach tool provided the functionality to read and monitor the process. This produces a binary file from the contents of the flash memory in the processor.

3.4 OFRAK

The firmware modification process for the ECM was conducted using the Open Firmware Reverse Analysis Konsole (OFRAK) framework. OFRAK was previously developed under the DARPA Assured Micropatching (AMP) program [34] and was initially designed for packing and unpacking XCAL file formats. The XCAL format is typically used for calibrations of hardware components and sensors in certain trucks.

The ECM firmware verifies its integrity through a validation mechanism, which was originally located in the XCAL format. However, since not everyone has access to the proprietary firmware update tool, it was needed to determine the physical location of the validation mechanism in memory through reverse engineering.

3.4.1 Overview of OFRAK-Based Firmware Modification

OFRAK was used to implement a firmware modification process that followed four key stages:

1. **Unpacking**: Extracting the original firmware components and identifying key memory regions.
2. **Analysis**: Reverse-engineering the binary to understand its internal functions and locating modification points.
3. **Modification**: Injecting security features, including Source Address (SA) filtering and Cipher-based Message Authentication Code (CMAC) authentication.
4. **Packing**: Reassembling the modified firmware.

3.4.2 Unpacking Stage

The first step in modifying the firmware involved extracting key components from the binary firmware to understand its structure and validation mechanisms. Since modifying large data elements or extending regions within the binary was not necessary, the unpacking stage focused primarily on:

- Extracting the validation mechanism configuration.
- Identifying memory segments and data structures relevant to CAN message handling.

Through binary analysis, the validation mechanism was pinpointed. This discovery was critical, as the firmware would refuse to execute if the validation failed after modifications.

3.4.3 Analysis Stage

Once the firmware was unpacked, the next step involved parsing and interpreting the extracted components. The analysis focused on:

- Understanding the memory layout of the firmware.
- Identifying the validation mechanism and the address ranges.
- Reverse-engineering the CAN message handling functions to locate injection points for new security features.

To achieve this, breakpoints were set using the Lauterbach Trace32 tool to monitor firmware execution and determine which functions interacted with the CAN controller's message buffers.

3.4.4 Modification Stage

The core firmware modifications were implemented using *PatchMaker*, a specialized sub-tool within OFRAK that allows for injecting custom code into a binary executable.

- Hooking into key firmware functions responsible for sending and receiving CAN messages.
- Injecting the payload with additional security measures:
    1. **Source Address (SA) Filtering**: Preventing unauthorized ECUs from transmitting spoofed CAN messages.
    2. **CMAC Authentication**: Introducing cryptographic validation for transmitted messages.

PatchMaker integrates with existing compiler toolchains (GCC for PowerPC in this case) to generate and inject patches into the firmware binary. The process includes:

1. **Toolchain Integration**: Compiling the injected code with appropriate settings for the NXP MPC5674F processor.
2. **Patch Creation**: Writing new security mechanisms in C and compiling them into binary form.
3. **Linking and Compilation**: Ensuring correct memory mapping and symbol resolution.
4. **Injection into Firmware**: Replacing specific memory segments with the newly compiled functionality.

A key challenge was locating unused memory space within the ECMs' limited RAM to store additional global state variables. This was resolved by leveraging the Enhanced Timing Processing Unit's (eTPU) RAM, which was found to be underutilized.

3.4.5 Packing Stage

After modifying the firmware, it was necessary to repack the binary to ensure proper execution. This process involved:

- Recalculating the validation mechanism.
- Integrating the modified firmware components into a flashable image.
- Verifying that the modified binary maintained the same execution flow as the original firmware.

3.5 Load Modified Firmware

Now it's possible to load the firmware back onto the ECM. Another PRACTICE script for flashing the ECM with the modified firmware was developed, also provided as a template by the Lauterbach team [25].

4. Results and Analysis

This section documents the results obtained from the procedure outlined in the methodology. The tests were conducted using a hybrid gateway implementation (BeagleBone Black) but can be replicated on other ECUs for analysis, such as a Gateway, Brake Controller, and ECM. The modified firmware was flashed onto the ECM using a Lauterbach Trace32 script, which automated the following steps:

1. Resetting the processor.
2. Initializing the Memory Management Unit (MMU) for flash access.
3. Writing the modified firmware to the flash memory.
4. Rebooting the system and verifying execution.

To ensure correctness, the validation mechanism was intentionally disabled during one test, causing the ECM to reject the firmware. This confirmed that verification was functioning properly.

After flashing, the modified firmware was tested both in a controlled laboratory environment and on a Kenworth T270 truck. Key testing procedures included the following.

4.1 Message Timing Analysis

To determine whether the modifications affected real-time CAN message processing, a Saleae Logic Analyzer was used to capture and analyze CAN message intervals. The results showed that:

- The timing distribution of messages remained consistent with the unmodified firmware.
- The SA filtering and CMAC authentication mechanisms did not introduce noticeable latency.

Using the Logic Pro Software [35], the RX and TX pins of the powered CAN transceiver were connected to the network and analyzed. This bypasses all CAN controller hardware and application software, and allows for the investigation of CAN bus timing between the vanilla, or otherwise unmodified version and modified version of the ECM firmware. The results can be seen in figure 5.

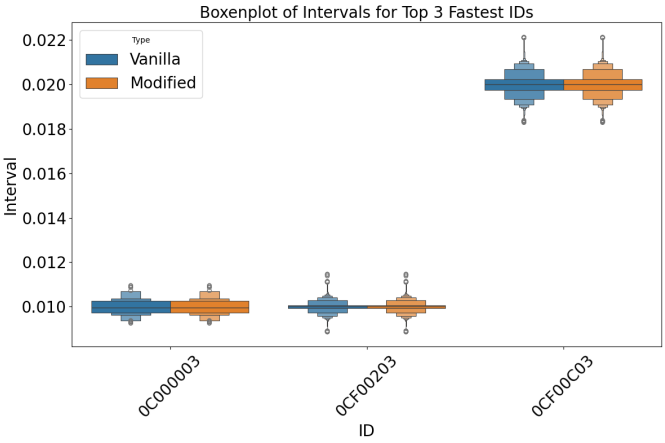

Figure 5: Boxenplot showcasing similarities in timing between vanilla and modified ECM

It was found from busload analysis with PCAN-View [36], that the modified firmware did not show any noticeable increase in the busload (i.e., a 5.3% change in max throughput).

4.2 Functional Testing Verified the Proper Execution of Security Features

After performing the tests with the Saleae, a gateway program written in python was introduced to serve as the proof of concept in communication between the ECM and gateway in a secure fashion. Given this project is in the first phases of it's realization, it was an in the interest of time, and given more resources, the team could've successfully implemented the protocol in two ECUs (e.g., ECM and Diagnostics Gateway). The gateway program, instead implemented on a BeagleBone Black, was able to successfully detect the intrusion. Overall, the ability to deploy the CAN conditioner code onto an existing 'simulated' ECM without modifying or introducing physical hardware was demonstrated in Figure 6 and 7. It's important to note that a naive approach toward the false positive portion of the gateway code was intended to increment given known intrusive messages.

To test a true positive, the PCAN View program was setup to transmit a CAN message that used the same source address (`0x00`) of the engine controller. This message was transmitted once by the Peak USB-CAN adapter (. Since this message went through an external device, the message is not included in

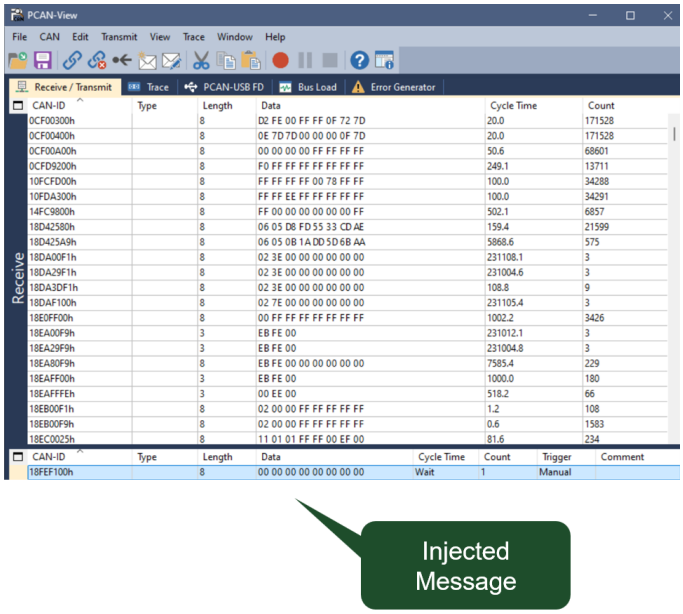

Figure 6: Injected J1939 spoofed engine controller message

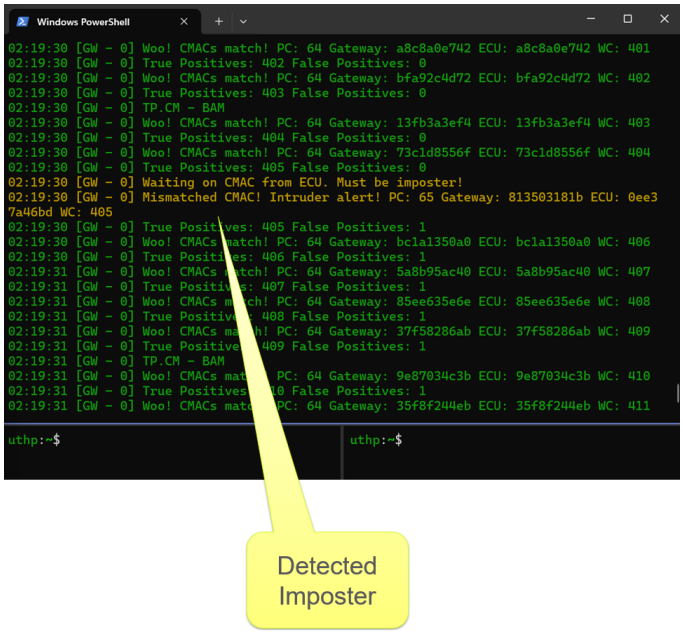

Figure 7: CMAC IDS implementation

the CMAC calculation that's running on the engine controller. Therefore the CMAC broadcast from the code provided by RBS will be different than the CMAC digest calculated by the gateway. This result is shown in Figure 8.

4.3 Compute Resource Analysis

The computational overhead introduced by the modifications was analyzed using the Lauterbach Trace32 tool. Profiling results indicated:

- The modified firmware used only 0.5% of the CPU's processing power, with a margin of error of 0.0383%.
- CMAC calculations executed in an average of 32 microseconds, well within acceptable limits (a single CAN message takes about 250-500 microseconds to transmit over the CAN bus).

4.4 Data Logs

After collecting logs utilizing the popular Linux tools available under the 'can-utils' package, it was verified the modified version of the firmware was sending different CAN IDs that was previously captured from the vanilla ECM. Each ID is decomposed into the J1939 protocol data unit elements of priority, parameter group number (PGN), destination address (DA), and source address (SA).

1. `18D42580` - priority = 6, PGN = `0xD400`, DA = `0x25`, SA = `0x80`
2. `18D425A9` - priority = 6, PGN = `0xD400`, DA = `0x25`, SA = `0xA9`
3. `18FECA25` - priority = 6, PGN = `0xFECA`, DA = `0xFF` SA = `0x25`

There are three source/desination addresses of interest: the first is `0x25` (37dec), which is the secure gateway application.

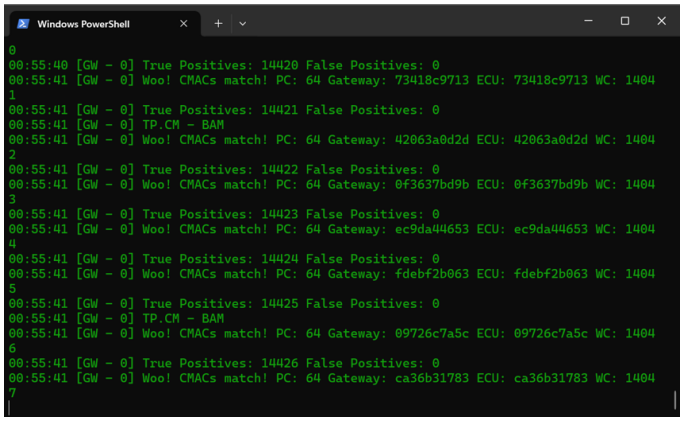

Figure 8: Gateway after around 55 minutes with no intrusions, showcasing no false positives

The second is `0x80` (128dec), which is the can conditioner application in the ECM that is protecting data from the engine controller. The third is `0xA9`, which is the CAN Conditioner for the engine retarder (exhaust brake). The parameter group (PG) label for `0xD400` is the Diagnostics Message 18 (DM18) message, which was customized for this application. Finally, PGN `0xFECA` is J1939 Diagnostic Message 1, which was used to higlight any intrusions detected from the Secure Gateway. These unique IDs were introduced after installing the modified ECM, verified the frames for sending the CMACs were on the CAN bus.

Overall, the CMAC IDS solution was found to be successful in detection with a 100% true positive rate, and very limited performance overhead.

5. Future Work

While this research demonstrated the feasibility of an IDS implemented with whitelisted CAN source addresses and cryptographic message authentication, further investigation should be performed. The gateway in this research served as a demon-

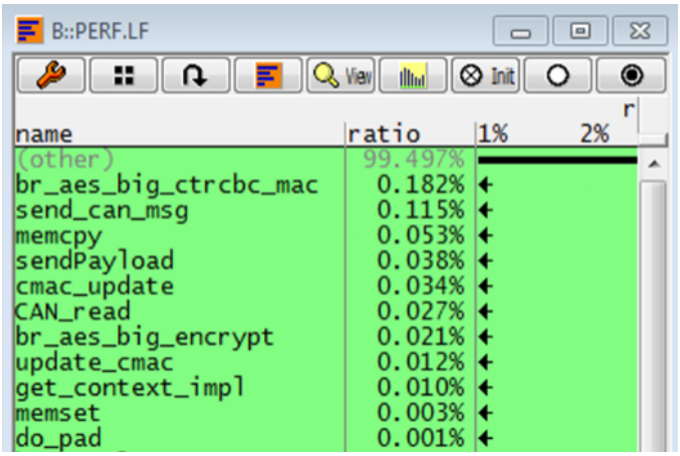

Figure 9: Lauterbach debugger timing analysis of modified test bench ECM

stration, meaning that future research, the gateway would be integrated in an existing system, with security fixes that could be added similarly to the operations performed in this study. It's also been demonstrated that VDAs provide other interfaces (e.g., WiFi or Bluetooth) [32] that could lead to faster security approaches mentioned in this research. Future work could assess the practicality of adding wireless security mechanisms for secure over-the-air (OTA) updates.

Key management and session key establishment could prove even more secure [21]. This could provide engineers with the ability to establish an infrastructure that facilitates the integration of security of the CAN-bus on legacy systems. Furthermore, future research should focus on deploying these system at scale, evaluating its performance across different vehicle types and fleets. The integration of Software Bill of Materials (SBOM) and Firmware Bill of Materials (FBOM) technologies would further strengthen security by ensuring transparency in the automotive domain.

6. Conclusion

This research demonstrates that cryptographic operations, specifically the implementation of a CMAC-based intrusion detection system, can be performed on the CAN protocol without requiring additional hardware. By leveraging existing ECUs and modifying firmware, this study presents a practical and cost-effective approach to improving CAN security while maintaining compatibility with current vehicle architectures.

The proof-of-concept implementation successfully modified an ECM to authenticate messages based on 29-bit CAN identifiers, enhancing message integrity without introducing significant latency or performance degradation. These findings validate the feasibility of applying cryptographic security mechanisms to legacy automotive networks, addressing long-standing vulnerabilities in the CAN protocol.

7. Acknowledgements

This material is based upon work supported by the Government under Contract No. W912CH-24-C-0008. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the corresponding government agencies.

References

[1] International Organization for Standardization. Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signaling, 2003. URL https://www.iso.org/standard/33422.html. Specifies the CAN protocol, including its broadcast nature and lack of authentication mechanisms.

[2] International Organization for Standardization. Road vehicles – Unified diagnostic services (UDS) – Part 1: Application layer, 2020. URL https://www.iso.org/standard/72439.html. Defines diagnostic communication for vehicle ECUs, including security access and fault memory handling.

[3] Intel Corporation. Jtag 101: Ieee 1149.x and software debug. https://web.archive.org/web/20170830070123/http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/jtag-101-ieee-1149x-paper.pdf, 2009.

[4] Society of Automotive Engineers. SAE J1939 Standards Collection, 2009. URL https://www.sae.org/standardsdev/groundvehicle/j1939a.htm. Defines higher-layer protocols for CAN in heavy-duty vehicles and industrial applications.

[5] Rik Chatterjee, Subhojeet Mukherjee, and Jeremy Daily. Exploiting transport protocol vulnerabilities in SAE J1939 networks. In *Proceedings of the Inaugural International Symposium on Vehicle Security & Privacy*, San Diego, CA, USA, 2023. Internet Society. ISBN 978-1-891562-88-4. doi:10.14722/vehiclesec.2023.23053. URL https://www.ndss-symposium.org/wp-content/uploads/2023/02/vehiclesec2023-23053-paper.pdf.

[6] S. Mukherjee, H. Shirazi, I. Ray, J. Daily, and R. Gamble. Practical DoS attacks on embedded networks in commercial vehicles. In *Proceedings of the 12th International Conference on Information Systems Security*, pages 23–42, 2016.

[7] P. Murvay and B. Groza. Security shortcomings and countermeasures for the sae j1939 commercial vehicle bus protocol. *IEEE Transactions on Vehicular Technology*, 67(5): 4325–4339, 2018.

[8] Siti-Farhana Lokman, Abu Talib Othman, and Muhammad-Husaini Abu-Bakar. Intrusion detection system for automotive controller area network (can) bus system: a review. *EURASIP Journal on Wireless Communications and Networking*, 2019(184):1–17, 2019. doi:10.1186/s13638-019-1484-3. URL https://doi.org/10.1186/s13638-019-1484-3.

[9] Arnaud Van Herrewege, Daniël Singelee, and Ingrid Verbauwhede. Canauth - a simple, backwards compatible broadcast authentication protocol for can bus. In *Proceedings of the ECRYPT Workshop on Lightweight Cryptography*, pages 1–8, 2011.

[10] Sy V. Doan and Rajesh Ganesan. Can-crypto: Controller area network encryption for protecting can bus messages. *IEEE Transactions on Vehicular Technology*, 66(6):10779–10789, 2017. doi:10.1109/TVT.2017.2682609.

[11] Sebastian Nürnberger and Christian Rossow. vatican - vetted, authenticated can bus. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 1067–1080, 2016. doi:10.1145/2976749.2978301.

[12] Milan Jukl and Jan Čupera. Security enhancement of can protocol using tea algorithm. *International Journal of Computer Applications*, 975(8887):1–6, 2016.

[13] Petr Hanacek and Martin Sysel. Three-des based secure communication on controller area network. In *Proceedings of the International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 89–94, 2016. doi:10.1109/ICACSIS.2016.7872762.

[14] Olaf Pfeiffer and Christian Keydel. Cancrypt – authentication and encryption for canopen and other can protocols. *Embedded Systems Academy White Paper*, pages 1–12, 2017.

[15] Weiming Wang and Mohan Sawhney. Vecure: A practical security framework to protect the can bus. In *IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 1–6, 2014. doi:10.1109/ICVES.2014.7063693.

[16] Hossam Farag. Cantrack: An intuitive can bus security approach. *International Journal of Cyber Security and Digital Forensics*, 6(3):224–237, 2017. doi:10.17781/P002342.

[17] Ken Tindell. Cryptocan: Securing the can bus with cryptography. *Canis Labs White Paper*, pages 1–9, 2020.

[18] Trillium Secure Inc. Securecan: Lightweight cryptographic protection for can bus. *Trillium Secure White Paper*, pages 1–10, 2018.

[19] AUTOSAR Consortium. Secure onboard communication (secoc). *AUTOSAR Specification*, pages 1–45, 2019.

[20] Road vehicles — cybersecurity engineering, 2021. URL https://www.iso.org/standard/70918.html.

[21] Jeremy Daily, David Nnaji, and Ben Ettlinger. Securing can traffic on j1939 networks. In *Workshop on Automotive and Autonomous Vehicle Security (AutoSec)*, pages 1–7, Virtual, February 2021. NDSS Symposium. doi:10.14722/autosec.2021.23031. URL https://dx.doi.org/10.14722/autosec.2021.23031.

[22] Rik Chatterjee, Carson Green, and Jeremy Daily. Exploiting diagnostic protocol vulnerabilities on embedded networks in commercial vehicles. In *Symposium on Vehicles Security and Privacy (VehicleSec) 2024*, San Diego, CA, USA, February 2024. NDSS Symposium. ISBN 979-8-9894372-7-6. doi:10.14722/vehiclesec.2024.23046. URL https://www.ndss-symposium.org/wp-content/uploads/vehiclesec2024-46-paper.pdf.

[23] Yelizaveta Burakova, Bill Hass, Leif Millar, and André Weimerskirch. Truck hacking: An experimental analysis of the SAE j1939 standard. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, Austin, TX, August 2016. USENIX Association. URL https://www.usenix.org/conference/woot16/workshop-program/presentation/burakova.

[24] OFRAK. Ofrak official website. https://ofrak.com/, 2025. URL https://ofrak.com/.

[25] Lauterbach. Lauterbach official website. https://www.lauterbach.com/, 2025. URL https://www.lauterbach.com/.

[26] Jeremy Daily, Matthew DiSogra, and Duy Van. Chip and board level digital forensics of cummins heavy vehicle event data recorders. In *SAE Technical Paper 2020-01-1326*, 2020. doi:10.4271/2020-01-1326.

[27] Minki Nam, Seungyoung Park, and D. Kim. Intrusion detection method using bi-directional gpt for in-vehicle controller area networks. *IEEE Access*, 9:124931–124944, 2021. doi:10.1109/ACCESS.2021.3110524.

[28] Mohammed Almehdhar, Abdullatif Albaseer, Muhammad Asif Khan, Mohamed Abdallah, Hamid Menouar, Saif Al-Kuwari, and Ala Al-Fuqaha. Deep learning in the fast lane: A survey on advanced intrusion detection systems for intelligent vehicle networks. *IEEE Open Journal of Vehicular Technology*, 5:869–906, 2024. doi:10.1109/OJVT.2024.3422253.

[29] Mehmet Bozdal, Mohammad Samie, Sohaib Aslam, and Ian Jennions. Evaluation of can bus security challenges. *Sensors*, 20(8):2364, 2020. doi:10.3390/s20082364. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7219335/.

[30] Sam L. Thomas, Jan Van den Herrewegen, G. Vasilakis, Zitai Chen, Mihai Ordean, and Flavio D. Garcia. Cutting through the complexity of reverse engineering embedded devices. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021: 360–389, 2021. doi:10.46586/tches.v2021.i3.360-389.

[31] Yuhao Qiu. An improved method for reverse engineering ecu firmware. 13175:131751A – 131751A–6, 2024. doi:10.1117/12.3032054.

[32] Edward Larson, Wyatt Ford, Sam Lerner, and Jeremy Daily. Vehicle diagnostics adapter cybersecurity concerns with wireless connectivity. *SAE Technical Paper*, (2023-01-0034), April 2023. doi:10.4271/2023-01-0034. URL https://doi.org/10.4271/2023-01-0034.

[33] David D. Walden, Thomas M. Shortell, Garry J. Roedler, Bernardo A. Delicado, Odile Mornas, Yip Yew-Seng, and David Endler. *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. John Wiley & Sons Ltd., Hoboken, NJ, USA, 5th edition, 2023. ISBN 9781119814290.

[34] Rik Chatterjee, Ben Karel, Ricardo Baratto, Michael Gordon, and Jeremy Daily. Assured micropatching of race

conditions in legacy real-time embedded systems. In *Proceedings of the 2024 International Conference on Embedded Systems*, Denver, CO, USA, 2024. URL https://www.researchgate.net/publication/382214314_Assured_Micropatching_of_Race_Conditions_in_Legacy_Real-time_Embedded_Systems.

[35] Inc. Saleae. Saleae - logic analyzers. https://www.saleae.com/, 2025. URL https://www.saleae.com/.

[36] Peak-System Technik GmbH. Peak-system - can and industrial communication solutions. https://www.peak-system.com, 2025. URL https://www.peak-system.com.

A. Memory Map of MPC5674

Block	Address Range
Flash (4MB)	0x0000_0000 — 0x003F_FFFF
Reserved	0x0040_0000 — 0x00EF_BFFF
Flash B Shadow Block	0x00EF_C000 — 0x00EF_FFFF
Reserved	0x00F0_0000 — 0x00FF_BFFF
Flash A Shadow Block	0x00FF_C000 — 0x00FF_FFFF
Emulation re-mapping of flash	0x0100_0000 — 0x1FFF_FFFF
External Development Memory	0x2000_0000 — 0x3FFF_FFFF
Internal Standby SRAM (32 KB)	0x4000_0000 — 0x4000_7FFF
Internal SRAM (224 KB)	0x4000_8000 — 0x4003_FFFF
Reserved	0x4004_0000 — 0xC3EF_FFFF
Peripheral Bridge A Registers	0xC3F0_0000 — 0xC3F0_3FFF
Reserved	0xC3F0_4000 — 0xC3F7_FFFF
FMPLL	0xC3F8_0000 — 0xC3F8_3FFF
EBI Configuration	0xC3F8_4000 — 0xC3F8_7FFF
Flash A Configuration	0xC3F8_8000 — 0xC3F8_BFFF
Flash B Configuration	0xC3F8_C000 — 0xC3F8_FFFF
SIU	0xC3F9_0000 — 0xC3F9_3FFF
Reserved	0xC3F9_4000 — 0xC3F9_FFFF
eMIOS	0xC3FA_0000 — 0xC3FA_3FFF
Reserved	0xC3FA_4000 — 0xC3FB_BFFF
PMC	0xC3FB_C000 — 0xC3FB_FFFF
eTPU Registers	0xC3FC_0000 — 0xC3FC_3FFF
eTPU Code RAM	0xC3FD_0000 — 0xC3FD_5FFF
PIT / RTI	0xC3FF_0000 — 0xC3FF_3FFF
Peripheral Bridge B Registers	0xFFF0_0000 — 0xFFF0_3FFF
XBAR (AXBS)	0xFFF0_4000 — 0xFFF0_7FFF
MPU	0xFFF1_0000 — 0xFFF1_3FFF
SWT	0xFFF3_8000 — 0xFFF3_BFFF
STM	0xFFF3_C000 — 0xFFF3_FFFF
ECSM	0xFFF4_0000 — 0xFFF4_3FFF
eDMA_A	0xFFF4_4000 — 0xFFF4_7FFF
INTC	0xFFF4_8000 — 0xFFF4_BFFF
eDMA_B	0xFFF5_4000 — 0xFFF5_7FFF
eQADC_A	0xFFF8_0000 — 0xFFF8_3FFF
eQADC_B	0xFFF8_4000 — 0xFFF8_7FFF
DSPI_A	0xFFF9_0000 — 0xFFF9_3FFF
DSPI_B	0xFFF9_4000 — 0xFFF9_7FFF
DSPI_C	0xFFF9_8000 — 0xFFF9_BFFF
DSPI_D	0xFFF9_C000 — 0xFFF9_FFFF
eSCI_A	0xFFFB_0000 — 0xFFFB_3FFF
eSCI_B	0xFFFB_4000 — 0xFFFB_7FFF
eSCI_C	0xFFFB_8000 — 0xFFFB_BFFF
FlexCAN_A	0xFFFC_0000 — 0xFFFC_3FFF
FlexCAN_B	0xFFFC_4000 — 0xFFFC_7FFF
FlexCAN_C	0xFFFC_8000 — 0xFFFC_BFFF
FlexCAN_D	0xFFFC_C000 — 0xFFFC_FFFF
FlexRay	0xFFFE_0000 — 0xFFFE_3FFF
System Information Module	0xFFFF_0000 — 0xFFFF_3FFF

B. Diagnostic Tool User Interface