Secure In-Vehicle Storage

José Sousa

Master Degree in Network and Information Systems Engineering

Computer Science Department

2022

Supervisor

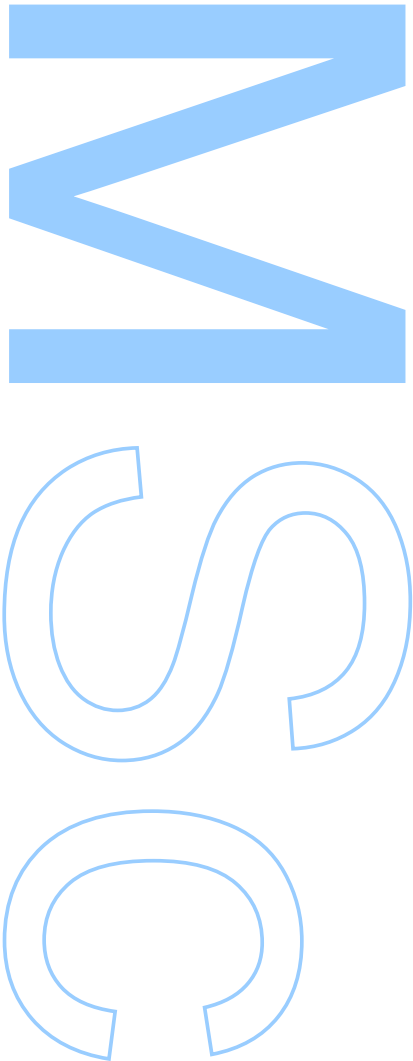
Prof. Hugo Pacheco

Faculdade de Ciências da Universidade do Porto

Co-Supervisor

Fernando Alves

VORTEX-CoLab



U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

UNIVERSIDADE DO PORTO

MASTERS THESIS

Secure In-Vehicle Storage

Author:
José SOUSA

Supervisor:
Hugo PACHECO

Co-supervisor:
Fernando ALVES

A thesis submitted in fulfilment of the requirements
for the degree of MSc. Network and Information Systems Engineering

at the

Faculdade de Ciências da Universidade do Porto
Departamento de Ciência de Computadores

November 8, 2023

Acknowledgements

I want to thank my supervisor, Professor Hugo Pacheco for this opportunity and all the support given during this work.

I want to thank VORTEX-CoLab, mainly to Fernando Alves and Ali Shoker for the opportunity to do this project for my final thesis and for having given the support, the tools and the clarification of all the doubts for the development of the same.

I want to thank my family, specially my parents, for supporting me throughout my academic life in every possible way and always believing in me.

I also want to thank all my friends and colleagues that support me during this step of my life.

UNIVERSIDADE DO PORTO

Abstract

Faculdade de Ciências da Universidade do Porto
Departamento de Ciência de Computadores

MSc. Network and Information Systems Engineering

Secure In-Vehicle Storage

by José SOUSA

Vehicles nowadays are more technologically advanced than ever before due to recent technologies that improve comfort and safety while operating the vehicle. Multiple microcontrollers in the internal networks of the cars create a large quantity of data, mostly diagnostic data, as a result of this expansion of the vehicle's functions. As a result, there is a need for systems that can support and analyze this quantity of data. The goal of this dissertation project is to research and evaluate an actual vehicle dataset. Reverse engineering the dataset, describing its system architecture, and identifying a thorough description of the data that can be found in a contemporary vehicle are the goals of this study and analysis. A solution that can store the data from this dataset in a centralized logging platform and provide data security assurances like integrity and auditability is intended. Thus, this paper provides a description of the system architecture, along with an identification and in-depth description of the data that may be located in a vehicle, as well as the identification and analysis techniques for this study of a dataset from a vehicle. This paper also develops a system architecture and configures an immutable database that can ensure the security guarantees of audibility and integrity of the stored data. A logging storage solution is also offered.

UNIVERSIDADE DO PORTO

Resumo

Faculdade de Ciências da Universidade do Porto
Departamento de Ciência de Computadores

Mestrado em Engenharia de Redes e Sistemas Informáticos

Armazenamento Seguro em Veículos

por José SOUSA

Nos dias de hoje, os veículos estão cada mais modernos em termos de tecnologia pois apresentam novas funcionalidades aumentando o conforto e segurança na condução do veiculo. Devido a esta evolução das funcionalidades no veiculo, vários micro-controladores nas redes internas dos veículos produzem uma quantidade enorme de dados, principalmente dados de diagnostico e por isso surge a necessidade de haver sistemas capazes de suportar e analisar esta quantidade de dados. Este projeto de dissertação tem como o objetivo o estudo e analise de um *dataset* real proveniente de um veiculo. Esse estudo e analise pretende-se fazer *reverse engineering* do *dataset* e apresentar a sua arquitetura do sistema e identificação de uma descrição detalhada dos dados que podem ser encontrados num veículo moderno. Pretende-se uma solução capaz de armazenar os dados deste *dataset* numa plataforma de *logging* centralizada com garantias de segurança dos dados, tais como, integridade e auditabilidade. Assim este documento descreve os procedimentos de identificação e analise a este estudo de um *dataset* proveniente de um veiculo apresentando uma descrição da arquitetura do sistema, bem como, uma identificação e descrição detalhada dos dados que podem ser encontrados num veiculo. Também, neste documento, é apresentada uma solução de armazenamento de *logging* desenvolvendo uma arquitetura para o sistema, bem como a configuração de uma base de dados imutável capaz de assegurar as garantias de segurança de integridade e audatibilidade dos dados armazenados.

Contents

Sworn Statement iii

Acknowledgements v

Abstract vii

Resumo ix

Contents xi

List of Figures xiii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Thesis Statement . . . 2
- 1.3 Contributions . . . 2
- 1.4 Document Structure . . . 2

2 Background 5
- 2.1 Automotive Context . . . 5
  - 2.1.1 Internal Architeture . . . 5
  - 2.1.2 Control Area Network (CAN) . . . 7
  - 2.1.3 Local Interconnect Network (LIN) . . . 8
  - 2.1.4 FlexRay . . . 8
  - 2.1.5 Media Oriented Systems Transport (MOST) . . . 9
  - 2.1.6 Automotive Ethernet . . . 9
    - 2.1.6.1 Ethernet 10BASE-T1S . . . 10
    - 2.1.6.2 Ethernet 100BASE-T1 . . . 11
  - 2.1.7 Eletronic Control Unit (ECU) . . . 11
    - 2.1.7.1 Electronic Brake Control Module (EBCM) . . . 12
    - 2.1.7.2 Powertrain Control Module (PCM) . . . 12
    - 2.1.7.3 Body Control Module (BCM) . . . 12
  - 2.1.8 Mother ECU (MECU) . . . 13
  - 2.1.9 Log Formats . . . 13
    - 2.1.9.1 Diagnostic Log Trace (DLT) . . . 13
    - 2.1.9.2 DLT Message Format . . . 15

3 Related Work **17**
3.1 Over The Air (OTA) Updates . . . 17
3.1.1 Uptane . . . 19
3.2 Automotive Intrusion Detection System . . . 21
3.3 Reverse Engineering . . . 22

4 Automotive Use Case **23**
4.1 Architeture . . . 23
4.1.1 ECUs . . . 24
4.1.2 Logs . . . 24
4.1.2.1 DLT Logs . . . 26
4.2 Security Analysis . . . 30
4.2.1 Assumptions . . . 30
4.2.2 Attaker Capabilities . . . 30

5 Creating a Centralized In-Vehicle Data Store **33**
5.1 Description . . . 33
5.2 Databases . . . 35
5.2.1 ImmuDB . . . 35
5.3 Implementation . . . 37
5.3.1 Payload Organization . . . 37
5.3.2 SQL Schema . . . 37
5.3.3 Database Configuration . . . 39

6 Evaluation **43**
6.1 Experiments . . . 43

7 Conclusions and Future Work **45**
7.1 Main difficulties . . . 46
7.2 Future Work . . . 46

Bibliography **49**

List of Figures

2.1 The evolution of automotive architetures [4]. 5
2.2 Summary of different types of communication [17]. 11
2.3 Location of DLT protocol [22]. 14
2.4 Message format of DLT protocol[23]. 15

3.1 OTA system design [24]. 18
3.2 Uptane design [27]. 20
3.3 Anomaly-based in-vehicle IDS [30]. 21

4.1 Number of Apps/Ctxs in each ECU. 24
4.2 Number of connections on each ECU. 25
4.3 Aplication layer protocols in the dataset. 26
4.4 Number of logs per Bus channel. 28
4.5 Organization of vlans in the dataset. 29
4.6 Dataset architecture 31

5.1 System architecture in a vehicle netowrk. 35
5.2 Payload organization process. 37
5.3 Schema of tables organization. 38

Chapter 1

Introduction

1.1 Motivation

The emphasis on software of modern vehicles has undergone a great evolution compared to a few years ago, increasing the safety and comfort of driving. To this end, various components in vehicles were introduced and improved over time, such as assisted driving, braking prediction in case of detecting an object or person in front of the vehicle, autonomous parking and among others. For this to happen, several micro-controllers and other technologies were introduced or updated in the vehicles to form an internal network that collaborates focusing on software-centric analogy with modern computers.

However, with the increase of functionality in the vehicles, the number of micro controllers in vehicles is also increasing. According to Christoph Hammerschmidt in Market News, the number of micro controllers in vehicles is already more than one hundred and fifty [1]. Also because of this growth of micro controllers, the amount of data generated by these will also rise. According to Western Digital, with the autonomus driving getting closer to appearing on vehicles, data generated in the vehicles can reach a size of approximately one terabyte per year in the near future [2].

This observation is causing several problems in the automotive industry, forcing manufacturers to revisit the internal architectures of the vehicles towards supporting a larger number of micro-controllers and data size in the vehicles. This also enhances the importance of more complex software middleware to manage the growing infrastructure, and the pertinence of security considerations inside a vehicle, especially on what concerns the identification and appropriate handling of critical data [3].

1.2 Thesis Statement

The main objective of this thesis is the detailed analysis of a real-world automotive use case using a dataset from an actual vehicle. Following our understanding of the use case, we will propose a tailored platform for logging the real-time data created by micro controllers in similar settings. Our hope is that this platform will aid further analysis of the vehicle's logs, and facilitate the future reverse engineering of other automotive use cases. Along the way, we will also discuss some details of the particular architecture of the vehicle and look into the security vulnerabilities that it presents.

1.3 Contributions

In more detail, the main contributions in this thesis are:

- The reverse engineering of an automotive use case, through the analysis of a dataset of logs;
- An analysis of the architecture of a modern vehicle, based on information extracted from the logs. We identify important aspects such as data flow, communication, roles, ecus, apps, etc;
- A detailed description of the kinds of data logs can be found in a modern vehicle, including theyr types, structures and function;
- The proposal of a centralized logging platform for modern vehicles that are similar to the one analyzed in the dataset;
- The implementation and configuration of a database, which ensures data integrity and auditability and serves as the core component in our centralized logging platform.

1.4 Document Structure

This thesis is organized in seven chapters, describing a solution for the secure storage of data generated by micro-controllers present in today's vehicles. This chapter introduced some of the challenges that the increase of micro-controllers and data pose to current automotive architectures, as well as some problems that vehicles manufactures may face. To

make this thesis self-contained, Chapter 2 provides a summary of all the concepts used throughout this thesis, such as details of the vehicle architecture and the description of the micro-controllers and data bound to that architecture. Chapter 3 presents some related work related to the security auditing of vehicular architectures, considering both academic work and industrial developments by other authors. Chapter 4 provides a detailed explanation of the selected real-world automotive use case. Chapter 5 presents the design of our centralized secure storage solution. Chapter 6 discusses a preliminary analysis of the solution presented in the previous chapter. Chapter 7 concludes with some final remaks and directions for future work.

Chapter 2

Background

2.1 Automotive Context

2.1.1 Internal Architeture

As mentioned in the introduction, at the time when the various electronic features were being incorporated into vehicles, the internal network in a vehicle had a decentralized architecture, that is, the various micro-controllers communicated with each other without the need for a master micro-controller.

As we can see in architecture (A) from Figure 2.1, the vehicles presented a decentralized architecture, where the various micro-controllers were scattered in a disorganized way throughout the vehicle; this was mostly feasible because the amount of micro-controllers and electronic functionalities was very reduced compared to today.

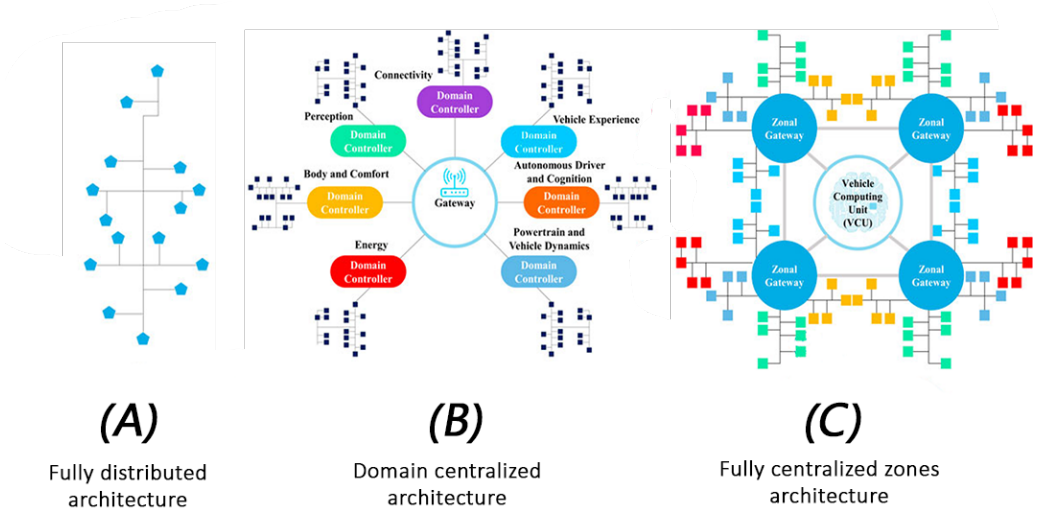


FIGURE 2.1: The evolution of automotive architetures [4].

However, when other features started to appear in the vehicle, and with the increase in the number of micro-controllers, the decentralized architectures in the vehicles started to pose several problems, one of them being that the BUS channel was shared by all micro-controllers. The concern was that the greater the number of micro-controllers in the vehicle, the greater the number of data packets traveling through the network would be, which could cause several congestions on the shared BUS channel. One of the solutions would be to replace the BUS channel with one that supported a higher bandwidth and speed than the traditional one, but the cost of manufacturing the vehicle would be more expensive, which manufacturers try to avoid. Thus, manufacturers have introduced new architectures in the vehicle. These architectures continued in a decentralized way but presented a better organization of the micro-controllers with the introduction of domains, thus allowing the existence of internal sub-networks separated from each other in the vehicle. The micro-controllers of each domain continue to use a shared communication channel BUS but with less congestion. Some of these domains are:

- **Powertrain:** Manages the function of driving of a vehicle, including electric motor control and battery management, engine control, transmission and steering control.
- **Infotainment domain:** Manages entertainment within the vehicle and exchanges information between the vehicle and the outside world, including the head unit, digital cockpit and telematics control module.
- **Body domain:** Manages comfort, convenience and lighting functions in the vehicle, including the body control module, door module and headlight control module.
- **Chassis Domain:** Manages the brake control systems such as antilock brake system, traction control system, and electronic stability control.

Looking at Figure 2.1 again, we can see in architecture (B) that the various micro-controllers are organized by the different types of domains and these are connected to each other through a gateway, like a standard router, so that the various domains can communicate with other domains using various BUS communication channels such as CAN, LIN, FlexRay, etc.

However, with technological developments growing more and more these days, the amount and size of data will be increasing as there is the introduction of new features

such as self-driving, IoT, in-vehicle streaming services, etc. Thus, manufacturers are re-developing architectures that better adapt to these new features. These new architectures of the future will be centralized architectures, that is, assume the existence of a master micro-controller called the Vehicle Computing Unit (VCU). This micro-controller will have a higher computational power and memory capacity than normal micro-controllers because it will not only be responsible as a gateway to the internal network but also because there are more features that require more computational calculations such as self-driving. The VCU will be responsible for calculating these computations or heavier operations of these functionalities.

When analyzing Figure 2.1 again, architecture (C) is centralized, and organized by several zones in the vehicle, with each zone being itself organized according to the various micro-controllers of the different domains. These zones are linked through the VCU.

However, with the change in architectures, micro-controllers and the architetures are not the only evolution to be noticed in the internal network of a vehicle.

2.1.2 Control Area Network (CAN)

At the beginning of the technological era in vehicles, electronic components communicated with each other via CAN (Control Area Network). A CAN is a serial network technology that was originally designed by Bosch in early 80's to support in-vehicle networking for the automotive industry. The CAN is primarily used in embedded systems, like micro-controllers and as its name implies, is a network technology that provides fast communication among microcontrollers up to real-time requirements, eliminating the need for the much more expensive and complex technology [5].

The CAN Bus is a two-wire, half duplex, high-speed network system, that is far superior to conventional serial technologies such as RS232 in regards to functionality and reliability and yet CAN Bus implementations are more cost effective [6].

While, for instance, TCP/IP is designed for the transport of large data amounts, CAN Bus is designed for real-time requirements and with its 1 MBit/sec bandwith rate that can easily beat a 100 MBit/sec TCP/IP connection when it comes to short reaction times. The greatest advantage of Controller Area Network lies in the reduced amount of wiring combined with an ingenious prevention of message collision, meaning no data will be lost during message transmission [6].

However, because of the bandwidth requirements of the automotive industry, the CAN data link layer protocol needed to be improved. In 2011, Bosch started the CAN FD development in close cooperation with vehicle makers and other CAN experts. The improved protocol overcomes to CAN limits by transmiting data faster than with 1 Mbit/s and the payload is now up to 64 byte long and not limited to 8 byte anymore [7].

2.1.3 Local Interconnect Network (LIN)

Also, another bus was developed for the automotive industy called the Local Interconnect Network (LIN) bus. LIN was developed to create a standard for low-cost, low-end multiplexed communication in automotive networks. Though the Controller Area Network (CAN) bus addresses the need for high-bandwidth, advanced error-handling networks, the hardware and software costs of CAN implementation have become prohibitive for lower performance devices such as power window and seat controllers. LIN provides cost-efficient communication in applications where the bandwidth and versatility of CAN are not required, thus modern automotive networks use a combination of LIN for low-cost applications primarily in body electronics and CAN for mainstream powertrain and body communications [8].

2.1.4 FlexRay

For automobiles to continue to improve safety, increase performance, reduce environmental impact, and enhance comfort, the speed, quantity and reliability of data communicated between a vehicle's electronic control units must increase. Advanced control and safety systems combining multiple sensors, actuators and electronic control units are beginning to require synchronization and performance past what the existing standard, Controller Area Network (CAN), can provide. Coupled with growing bandwidth requirements with today's advanced vehicles utilize over five separate CAN buses, automotive engineers are demanding a next-generation, embedded network. After years of partnership with OEMs, tool suppliers, and end users, the FlexRay, a deterministic, fault-tolerant and high-speed bus system standard has emerged as the in-vehicle communications bus to meet these new challenges in the next generation of vehicles [9].

Adoption of a new networking standard in complex embedded designs like automobiles takes time. While FlexRay will be solving current high-end and future mainstream

in-vehicle network challenges, it will not displace the other two dominant in-vehicle standards, CAN, and LIN. In order to optimize cost and reduce transition challenges, the next generation of automobiles will contain FlexRay for high-end applications, CAN for mainstream powertrain communications and LIN for low-cost body electronics.

2.1.5 Media Oriented Systems Transport (MOST)

Another type of bus used in the vehicles is the media oriented systems transport (MOST) bus. This bus was initially intended for implementation on optical fiber to support high bit rates, but fiber and copper transport layers are currently defined and they are expensive. MOST Bus provides a solution for automotive peripherals like radios, CD and DVD players, GPS navigation systems, and infotainment ECUs [10].

MOST was optimized for the automotive sector but is also used in non-automotive applications. It's ideal for those that benefit from a daisy-chain or ring topology and synchronous data communication to transport audio, video, and data signals on plastic optical fiber [11].

2.1.6 Automotive Ethernet

One thing is common within all the newer technologies, which is their requirement for bandwidth. Earlier there were very specific in-vehicle applications like body control, chassis control, powertrain, which used to generate very less amount of data, just in few kbps. But gradually, the need of higher bandwidth started realizing by the vehicle manufacturers and OEMs, as the applications to support new age autonomous functions, connectivity, ADAS systems, high end infotainment systems and vehicle electrification requires high bandwidth of data [12].

The more sophisticated the system is, the higher will be the expected complexity of the electrics/electronics systems. These high end systems generate, process and consume large amounts of data in real-time. For example, a self-driving vehicle of level 4 and level 5 contains multiple LIDAR, RADAR and Camera modules which generate data in few gbps and tbps. To process these huge amounts of data in real time with very low latency, a reliable, high speed network is required and Automotive Ethernet is a right fit for this purpose [12].

Some of the features and specifications of Automotive Ethernet are:

- **Switched Network:** Automotive Ethernet is a point-to-point networking technology, as opposed to bus technology, where one node or electronic device is connected to another one. A switch is being implemented in the system to connect many nodes, allowing several electronic control units to communicate with one another and routing traffic to multiple nodes within a network based on their physical addresses. [13].

- **Specific Standards:** The automobile-specific ethernet standards are 100Base-T1 and 1000Base-T1, which were developed to satisfy the needs of vehicle networking. With the use of audio video bridging, 100BASE-T1 permits the transmission of data (at 100 mbps) across unshielded single twisted-pair cable for linked vehicles, firmware/-software, and calibration data (AVB) [13].

- **Cost Efficient:**Flexray and CAN, whose throughput is only 10 mbps, are not appropriate for automotive applications since they stream audio and video. Automotive ethernet is significantly more cost-effective than other networking protocols, while having a beginning base rate of 100 mbps. Compared to the conventional cabling used for in-vehicle communication, the wiring utilized in automotive ethernet for vehicles is far lighter and more effective. Manufacturers can save the weight and connectivity expenses by up to 80% and 30%, respectively, by using light weight wiring. [13].

- **Networking Topology:** Numerous topologies are employed in automobile technologies. One of the topologies that is frequently used in automotive networking is point-to-point. The majority of automotive Ethernet applications use a star topology, in which all nodes and electronic control units are connected to a single switch. Star topology makes up the majority of the in-vehicle entertainment systems, which are built on automotive ethernet, whereas ring topology is used in the vehicle's safety-critical applications. [13].

2.1.6.1 Ethernet 10BASE-T1S

IEEE defines Ethernet 10BASE-T1S with 802.3cg. Each node is connected to a single cable using a multidrop architecture, with the S in 10BASE-T1S standing for short distance. Switches are not necessary with multidrop topology, which results in fewer cables and

a lower overall cost. Deterministic transmission on a collision-free multidrop network is the main objective of 10BASE-T1S. [14]

2.1.6.2 Ethernet 100BASE-T1

The 100BASE-T1 is a different variety of automobile ethernet that operates at 100 Mbps over at least fifteen meters of communication distance using only a single unshielded twisted-pair cable. Using the audio visual bridging (AVB) group of Ethernet protocols over unshielded single twisted-pair cable, Ethernet 100BASE-T1 can enable the transfer of audio, video, linked vehicle, firmware/software, and calibration data within vehicles. The IEEE Time-Sensitive Networking Task Group established the AVB collection of standards, which have synchronized nodes, low and deterministic latency, and traffic filtering. [15][16].

Network	Max baud rate	Max frame payload	AUTOSAR support	Priority/ timing	Segregation	Topology
LIN	20 kbps	8 Bytes	Y	Schedule	Physical network	Linear
CAN	1 Mbps	8 Bytes	Y	Priority arbitration	Physical network	Linear/Star
CAN-FD	8 Mbps	64 Bytes	Y	Priority arbitration	Physical network	Linear/Star
CAN-XL	10 Mbps	2048 Bytes	In development	Priority arbitration	Physical network	Linear/Star
FlexRay	10 Mbps	254 Bytes	Y	Schedule	Physical network	Linear/Star/ Hybrid
MOST25	25 Mbps	64 Bytes	N	Schedule	Physical network	Ring
10Base-T1S	10 Mbps	1500 kB	Y	AVB/TSN	VLAN	Linear
100Base-T1	100 Mbps	1500 kB	Y	AVB/TSN	VLAN	Switched flexible
1000Base-T1	1 Gbps	1500 kB	Y	AVB/TSN	VLAN	Switched flexible

FIGURE 2.2: Summary of different types of communication [17].

2.1.7 Eletronic Control Unit (ECU)

An Eletronic Control Unit (ECU) was introduced by General Motors in the year 1978 and is essentially an embedded system that is built on an automotive-grade microcontroller. Along with automotive software and communication protocols, an ECU is able to control the electrical systems and sub-systems in a vehicle like sensors and actuators. Some features of an ECU in a vehicle are: systematic transfer of data, dependability and security,

efficient data network, diagnostics, assistance in real-time decision making and improving quality of service [18].

The ECUs are available in 16-bit, 32-bit, and 64-bit versions, however 64-bit is now the market category with the most rapid market growth. Markets & Markets predicts that from 2018 to 2025, the automotive ECU market will expand at a compound yearly growth rate of 5.77%. By 2025, the market for automotive ECUs is anticipated to reach $39.28 billion. [19]. In the next sub-sections we wil present some specific ECUs that already exist in vehicles.

2.1.7.1 Electronic Brake Control Module (EBCM)

This electronic control unit is a smart brake module that offers the driver fully automated control over various actuators. To determine when to apply and release the brakes, the ECU regulates the actuators. This module can be applied to the following scenario: when a collision appears inevitable and any potential risk of collision is detected via radar data or any other data, the EBCM will send a signal to the brake function to automatically apply the brakes in order to lessen the force of the collision. This lessens the likelihood of accidents and helps the driver make quicker judgments, which can help prevent crashes or lessen their damage [19].

2.1.7.2 Powertrain Control Module (PCM)

A PCM is crucial to a vehicle's internal control of numerous subsystems. We must be aware of PCM's two primary subsystems before discussing it. The Transmission Control Unit (TCU) and Engine Control Module are these subsystems (ECM).

By controlling the actuators on the vehicle's internal combustion engine, the ECM controls how well the vehicle performs. On the other hand, a TCU unit uses data from the engine's sensors to optimize the switching behavior of the transmission in response to the driving situation at hand [19].

2.1.7.3 Body Control Module (BCM)

Even though each ECU operates separately, they still need to interact with one another. BCM controls this communication, which takes place over the CAN bus. Although the BCM is an ECU as well, it serves as a connection point for other ECUs. It is made up of a processor that controls numerous bodily processes in a vehicle. The functionalities of the

output devices are controlled by the BCM unit based on the data it receives from various input devices [19].

2.1.8 Mother ECU (MECU)

Another type of ECU is called a MECU. This ECU has greater resources (such as computing power, storage, RAM, etc.) than conventional ECUs. It is also referred to as a master ECU or a domain controller in the literature. It offers interfaces to the gateway and facilitates communication between ECUs [20]. It has command over the designated bus system. For instance, it may deactivate an ECU or permit partial bus operation in order to conserve energy. ECUs are less trustworthy than MECUs and applications that must be accessible during vehicle operation are consequently included into MECUs. Also, controlling the actions of its subordinate ECUs is another duty of this master ECU [21]. Besides all these features, in this dissertation, we will use the advantage of having an MECU for the implementation of a centralized database.

2.1.9 Log Formats

One of the advantages mentioned in relation to ECUs is the fact that they produce diagnostic data called Logs and may contain various information from the ECU itself or other information from other ECUs. These logs can contain information such as timetamp, payload, protocols, source and destination IPs, etc.

2.1.9.1 Diagnostic Log Trace (DLT)

The Diagnostic Log Trace protocol is one of the most popular in the automotive industry. Typically, a communication channel bus cannot directly transport the data (Log Message) sent by an application on the ECU. Prior to transmitting the message over the communication bus, the DLT module changes it into a standard message by adhering to a predetermined format for message sequences that are prohibited by the DLT Protocol according to the AUTOSAR standard. The DLT module gathers data from applications or other software modules from the ECUs and adds metadata, as shown in Figure 2.3. The data can then be transmitted to the communications bus [22].

The DLT protocol enables communication between external logging tools and the DLT module so that filters can be applied to received log entries based on severity level (fatal

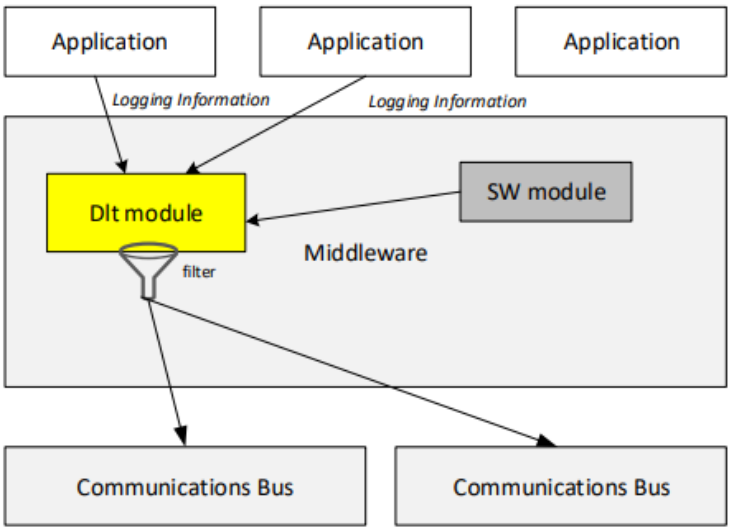


FIGURE 2.3: Location of DLT protocol [22].

error, information or warning). In order to prevent having to configure DLT again, external logging tools can instruct an application to only generate logs that match a specific filter level at runtime, modify the communication bus on which messages are to be transmitted, or keep filter level configuration in non-volatile memory [22].

2.1.9.2 DLT Message Format

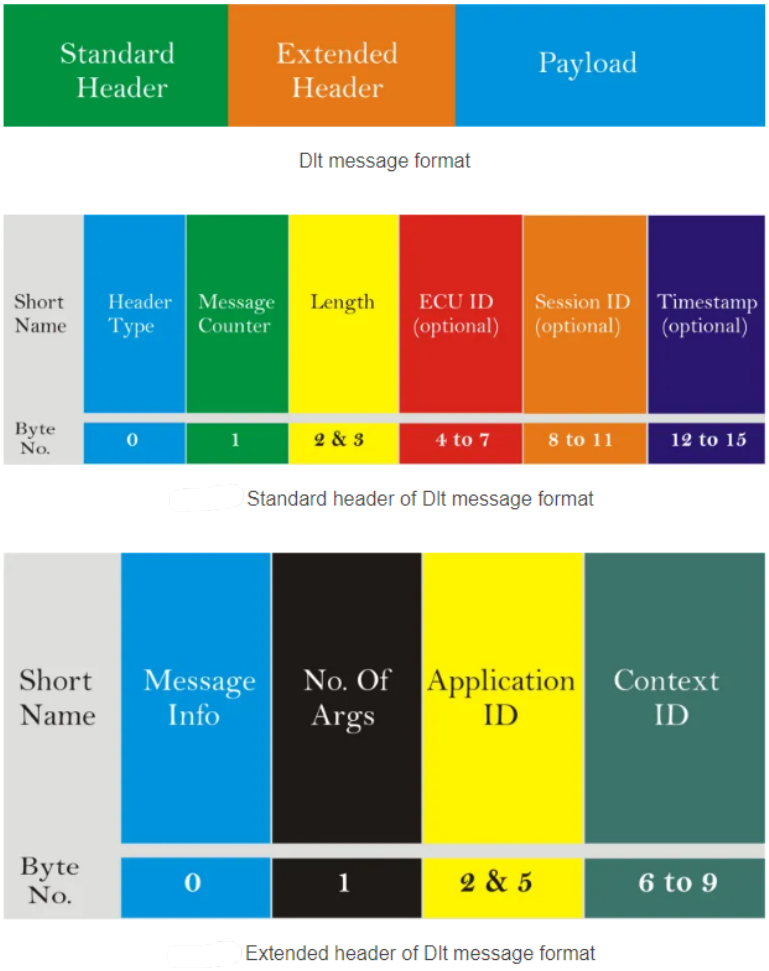


FIGURE 2.4: Message format of DLT protocol[23].

Looking at Figure 2.4 in detail, the message is divided into three parts:

- **Standard Header:** It has the following fields in its 16-byte header: a *header type* that comprises general information about the DLT message and specifies whether or not certain standard header fields are included as well as whether the extended header is being utilized. a *message counter* with an 8-bit capacity that keeps track of the DLT messages the DLT module has received. The *length*, which tells how long the message is. Additionally, it may optionally include an *ECU ID* to help identify the ECU that issued the message: It may include a *Session ID*, which is used to pinpoint the ECU's transmitter of a log or trace message. Additionally, a *Timestamp* that is used to add timing details about when the DLT message was created may be present [23].

- **Extended Header:** The header is 9 bytes long and has the following fields: *Message Info*, which specifies whether the message is verbose or not, and type of the message, which can be one of four different types: Log, Trace, Network, or Control. If the message type is a log message, for example, this field will provide various log message types, such as fatal, error, info, warning, debug, or verbose. The *Number of Arguments* field, which lists the number of arguments in the DLT message's payload, is another one in this header. The application on the ECU that produced the DLT message is likewise identified by an *Application ID*. The *Context ID* is a user-defined ID that logically groups DLT messages produced by an application on the ECU [23].

- **Payload:**This includes control information in addition to the parameters that are logged or tracked. Meta-data that identifies the type of information in the payload, such as boolean, Raw, String, Variable, Fixed point, etc., will be present in verbose mode. However, in non verbose mode, this data must be created as meta-data rather than being broadcast with the payload.

Chapter 3

Related Work

One of the most discussed topics in this new era of automotive is computer security present in vehicles, because with the increase of digital features to be incorporated in the vehicle, sometimes security is one of the last features to be implemented in these systems. With the increase, in recent years, of computer attacks to general software architectures, the concern for vehicular safety and security is simultaneously increasing.

In this chapter, we will mention and discuss scientific work and industrial developments primarily associated with cybersecurity which are related to the automotive context described in Chapter 2, such as how updates work in modern vehicles through the Over The Air technology. Therefore, one of the solutions that help mitigate cyberattacks on vehicles is the introduction of intrusion detection systems in vehicles.

3.1 Over The Air (OTA) Updates

As vehicle functionality increases, different software and firmware must be frequently kept up to date. By updating audio capabilities, optimizing user interfaces for streaming services or other apps and new features, infotainment systems become more usable. Additionally, keeping the firmware updated is crucial for ensuring the effective and safe operation of a vehicle. System improvements or corrections for brakes, advanced driver assistance systems (ADAS), chassis systems, and powertrain systems are common in automotive upgrades [25]. Approximately ten years ago, automobiles had to be serviced at a shop in order to perform these upgrades. The updates were then installed through the OBD-II interface. However, as automotive technology has advanced, most of these updates are now made Over The Air (OTA).

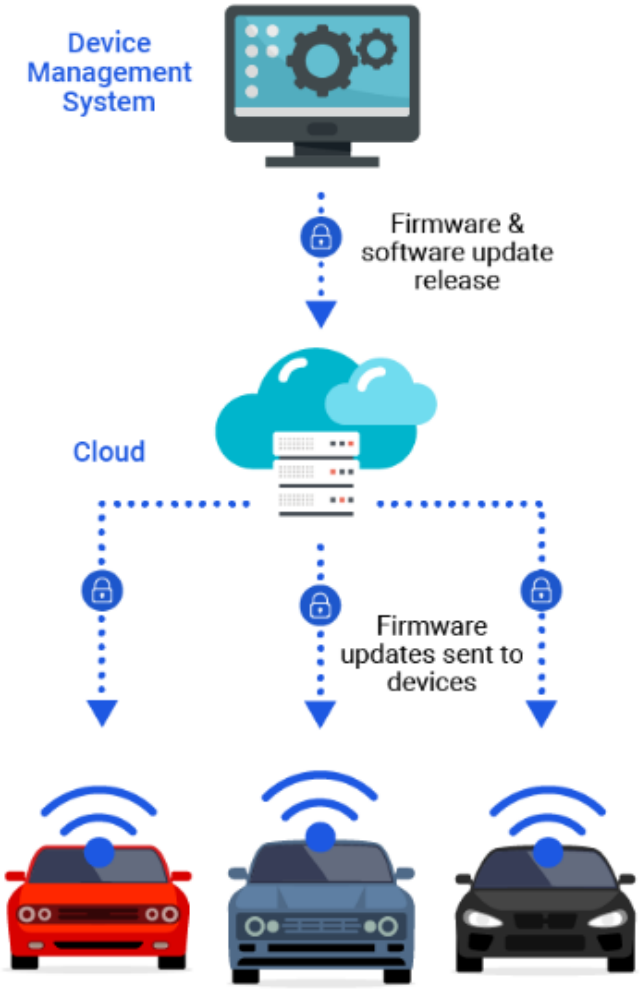


FIGURE 3.1: OTA system design [24].

Updates over the internet it was announced at the same time that the 3G mobile phone technology was, therefore it is not a brand-new invention. The wireless distribution of new software, firmware, or other data to mobile devices is known as an Over-The-Air (OTA) update. [25].

A variety of threats and attacks, including spoofing, tampering, repudiation, privilege escalation, and information leakage, can affect unsecured vehicle OTA updates. Software updates can be signed digitally after they have been encrypted, used with a signed certificate containing the public key of the entity requesting the update, and secured with TLS public key authentication (signed by a reputable Certificate Authority). Clients can also perform hostname verification to make sure they are connecting to a trusted server.

Only distributing updates to approved devices, tamper-proof logging of all significant events, initializing SOTA/FOTA updates with a secure boot mechanism, and software update systems that "fail gracefully" in the event of a denial-of-service (DoS) attack are other mitigation approaches. utilizing anti-malware measures including in-memory security and whitelists. Likewise, make sure that all shared resources are free of sensitive information and keys that were momentarily stored during software updates using compliant SOTA/FOTA software update systems. [24].

Utilizing Hardware Security Modules is another approach to enhancing data security on the ECUs (HSMs). These HSMs secure SOTA/FOTA upgrades, offer secure boot and secure debugging options, and collaborate with additional security features like MACsec, IPsec, and TLS embedded protocol engines to secure network traffic in automobiles. But manufacturers don't want to raise prices when adding new, pricey technology to automobiles because these HSMs are typically highly expensive to integrate into a vehicle. [24].

3.1.1 Uptane

The use of Uptane, which was developed in 2016, was one of the alternatives suggested to the manufacturers. Uptane is a framework for open and secure software updates that safeguards OTA software provided to automotive ECUs. The system guards against hostile actors who might infiltrate networks and servers used for signing and distributing updates. Utilizing a "defense-in-depth" strategy that relies on numerous layers of security procedures rather than a single security device, Uptane handles automobile security compromises. Even if hackers have stolen one or more particular software update keys, Uptane makes it very challenging for them to implant malware on vehicles maintained by OEMs. Attackers must compromise a certain threshold of keys in order to implant malware since a certain minimum number of keys is needed to sign metadata [26].

Some examples of the attacks that Uptane prevents are: **Arbitrary Software Attack** in which an attacker installs arbitrary software on an ECU thereby taking control of the device; **Eavesdrop Attack** in which an attacker reads unencrypted software updates sent from the repository to the vehicle; **Partial Bundle Installation Attack**, this is, an attacker causes software updates to be installed on only a subset of ECUs specified for software updates; **Rollback Attack** is an attack where the hacker installs old obsolete software on ECUs instead of up-to-date software [27].

Uptane requires the automaker to setup two repositories, a director repository and an image repository, and splits signing responsibility between online and offline keys.

The **Image Repository** includes signed metadata about binary images that can be installed. To make sure that no software used by the vehicle is likely to have been modified with as a result of key breach, the image repository metadata is signed using offline keys. The image repository, which exposes a public API to make these images and their metadata available to automobiles, allows the automaker and authorized vendors to contribute photos and the metadata that goes with them. The director repository receives the vehicles ECU configuration information, checks its inventory database, and directs the vehicle as to which images should be installed by producing signed metadata upon request [27].

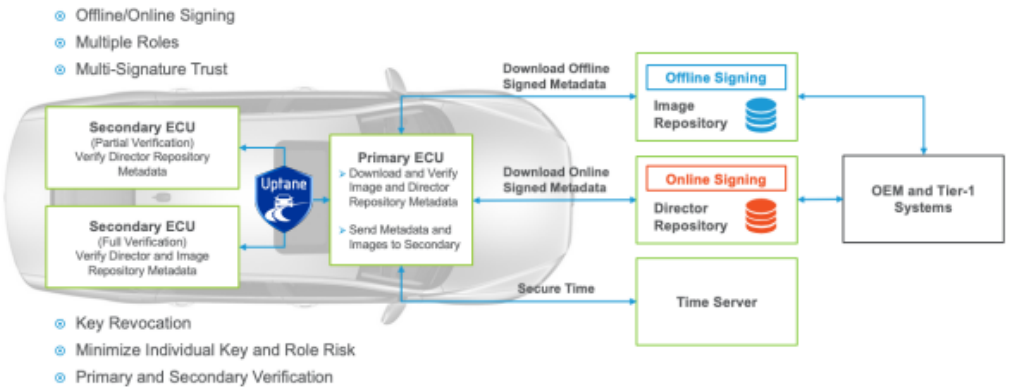


FIGURE 3.2: Uptane design [27].

The **Director Repository** uses online keys to sign all of its metadata. An automaker may provide both customization and security for the ECUs in their vehicles and safely and securely update them by deploying both repositories and using both online and offline metadata signing procedures.

The fact that the vehicle's ECUs offer a range of computing and cryptographic capabilities is another factor taken into account by Uptane. In order to check the image and director repository metadata, the ECUs can use one of two verification strategies: complete verification. Prior to starting any software package installation, the ECU must successfully complete the verification [27].

Even OTA and Uptane are technologies and frameworks that are essential for an architecture in vehicles for the management, operation and security of updates to the firmware of the different ECUs, in this dissertation project, the information and security of the updates may be present in the diagnostic logs of the vehicles.

3.2 Automotive Intrusion Detection System

Vehicle hacks are a pressing concern nowadays, thus it's critical to identify them as soon as possible so that appropriate defensive security measures may be taken. A method for achieving this goal is known as an intrusion detection system (IDS), which is a system that scans network traffic for suspicious activity and sends out notifications when such activity is found. [28].

There are two types of IDS: a **signature-based** system monitors the packets that are traversing the network, it compares these packets to the database and see if is a known attack or virus. On other hand, the **anomaly-based** system can alert the user to suspicious behavior that is unknown [29].

Manufacturers are adopting several forms of in-vehicle IDS in a variety of vehicles nowadays, based on the architecture's design and other characteristics. For instance, Sumitomo Electric Industries Ltd. is creating an in-vehicle anomaly detection system (IDS) with three monitoring levels that are differentiated based on the components of the in-vehicle network as shown in Figure 3.3.At higher levels, the system monitors fragmented subjects, making it simpler to identify the attacked subject and, as a consequence, to create specialized countermeasures. [30].

Therefore, if no other monitoring system is utilized, it is challenging for a high-level monitoring system to monitor the complete in-vehicle network in an on-board environment, which is subject to memory capacity and CPU performance limits. The technology

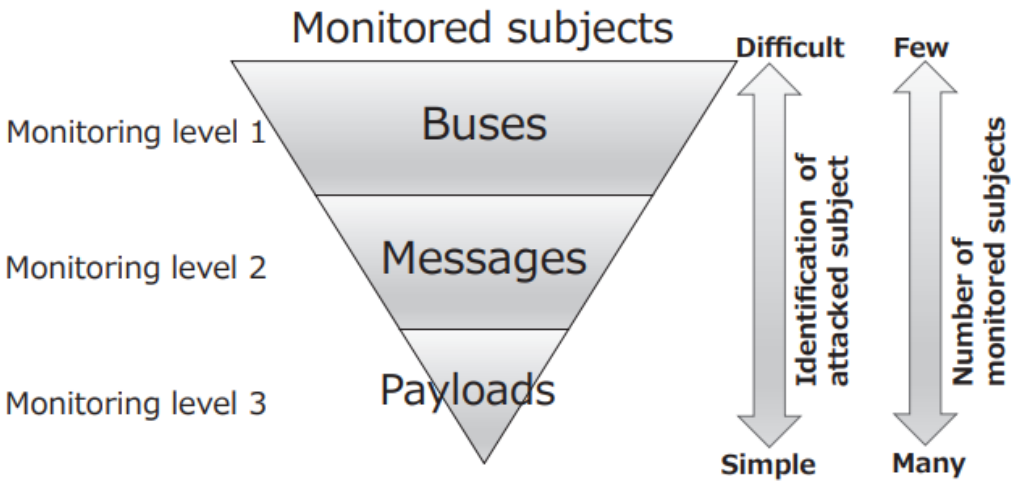


FIGURE 3.3: Anomaly-based in-vehicle IDS [30].

from Sumitomo Electric applies a low-level approach to monitor the complete in-vehicle network by combining the three monitoring levels indicated above [30].

In this dissertation project, the implementing a centralized logging platform, where a dedicated ECU stores the information of logs from the various information about the ECUs and the network and analyzes the data, is another solution to the issue of ECUs having very low computational power and thus not being able to monitor the network and can be used like an Automotive IDS.

3.3 Reverse Engineering

Another way to promote new features and solutions to problems in the automotive is through the use of reverse engineering which is process in which software, machines, aircraft, architectural structures and other products are deconstructed to extract design information from them [31].

Automotive reverse engineering it is very difficult to execute because most of the data is proprietary to the companies and requires some additional tools to be able to obtain the data on the architectures or on the ECUs in the vehicles. However, researchers use reverse engineering in a vehicle to obtain information about the architecture, about how the ECUs are organized in the network and how the communication between them works. Also an investigator called Thomas Huybrechts has reported the use of reverse engineering and other tecnologies such as machine learning to automatically extract data from the CAN Bus to analyse the driver behaviour or the health of the vehicle. Acording to him, doing reverse engineering of traffic on CAN Bus is really a time consuming task that requires a lot knowledge of the inner workings of the vehicle and its different subsystems. Based on this report by Thomas Huybrechts, in this dissertation project we will address something similar, which is the use of reverse engineering through automatic scripts to extract and analyze data from vehicle networks [32].

In this dissertation we will use reverse engineering to obtain and understand the architecture and how the ECUs are communicating with each other on the in-vehicle network using a dataset from a vehicle and thus implement our proposed solution.

Chapter 4

Automotive Use Case

This chapter presents and discusses the use case that was used for this dissertation, based on what we saw in the previous chapters. In this section, we will list the challenges that we have faced along the way, as well as the insights that we have gained from this case study, including what we have discovered about the architecture of the dataset. In another section, we will discuss some security assumptions and security vulnerabilities that are latent and/or relevant in the context of this architecture.

4.1 Architeture

Real data, emerging from a real automobiles, had to be analyzed in order to comprehend how the concrete architecture and ECUs are laid down in a modern vehicle. The majority of this data belongs to the businesses that make vehicles and ECUs, so getting accurate data wasn't simple. However, the company VORTEX-CoLab donated a dataset to enable its usage as this dissertation case study.

To understand how the architecture of the vehicle's internal network and how the ECUs are organized in the vehicle network, we had to reverse engineer the dataset that VORTEX-CoLab provided. For this, as the dataset was formatted in a PCAP file, we ran a first analysis through the WireShark tool which allowed us to have an simple overview of the structure of the logs and which information was more visible. After this first analysis, we started by creating scripts using the Python programming language to better automate the retrieval of data from this dataset.

4.1.1 ECUs

In this dataset it was not possible to find information about the architecture and composition of the system of each ECU, but it was possible to have a perception of how many ECUs there are in the dataset, which are in total about eleven ECUs. Of these eleven ECUs, seven have ID and are the following: *BMTR*, *WAVS*, *WAVA*, *WAVM*, *WAVG* and *BMT*. Unfortunately it is not possible to know what was the function in the network of each ECU.

It was also possible to have a perception of how many applications/programs(Apps) and application contexts/functions(Ctxs) exist in each ECU. This precept is shown in the following Figure 4.1.

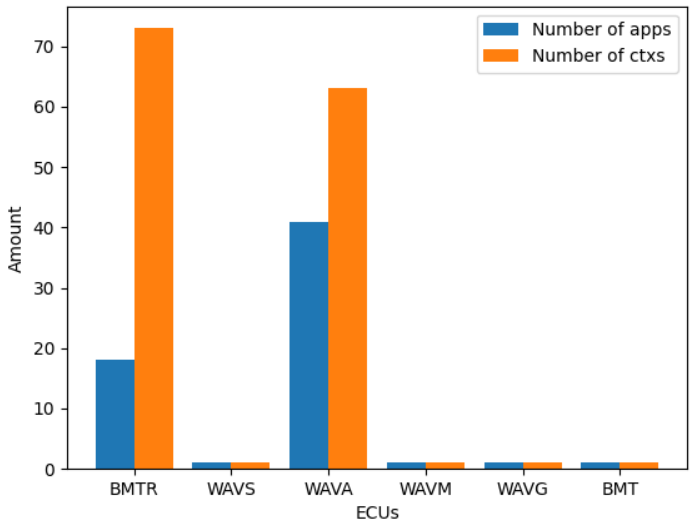


FIGURE 4.1: Number of Apps/Ctxs in each ECU.

In addition to having applications in the repetitive ECUs, in this dataset there are communications between the different ECUs. In the following Figure 4.2 we can see that in addition to the communications between the various ECUs identified by the DLT logs, there are other micro-controllers that do not have identification but are present in the network.

4.1.2 Logs

However, another logs were provided by VORTEX-CoLab that contains DLT data regarding the ECU that was analyzed. This logs have information about programs and their

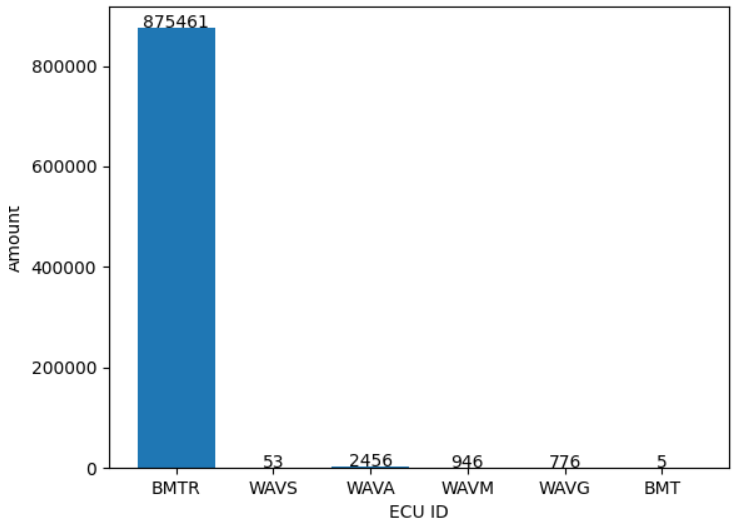


FIGURE 4.2: Number of connections on each ECU.

respective functions to be performed within an ECU. However, this logs were not analyzed in detail because specific tools were needed, which we were not able to obtain because the tools are owned by the manufacturing companies and they are not available to the community. There was still an attempt to reverse engineer it using Python libraries but without success as most of these libraries did not have updated documentation or support for this DLT format.

Thus, in the DLT logs we can see that there are several fields that contain information about what type of log message is being transmitted. This information can vary depending on whether the information is about an application being tested within the ECU or whether it is information about the transport channel the log is being transported to.

Through our scripts, regarding the logs, we can observe that there is a variety of application protocols in this dataset, as shown in the Figure 4.3 in this dataset we have DLT, SomeIP and Atuosar Network Management application layer protocols. Also, there are others such DHCP and DNS protocols in the aplication layer but we did not analyzed in deep because the information on these were not so relevant. However most of the logs in this dataset used DLT protocol and we then decided to analyze these DLT logs in more detail as they are the logs that present the most information about the ECUs.

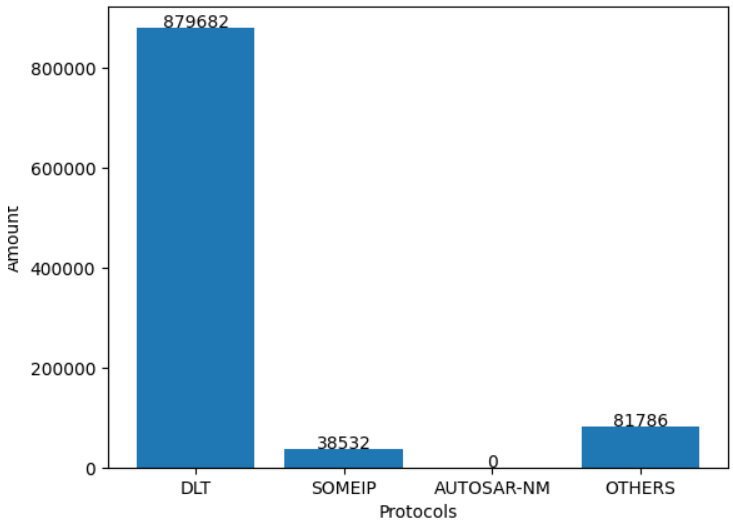


FIGURE 4.3: Aplication layer protocols in the dataset.

4.1.2.1 DLT Logs

As mentioned in Chapter 2, the DLT protocol follows a specific format of the AUTOSAR standard in relation to transmitted messages. This message is divided into three parts, the Stander Header and the Extender Header with information regarding the ECU, application/program, function/process that generated the log, what type of message the log presents. These logs can be of several levels such as:

- *Fatal*: These logs are consired as fatal and they are a unrecoverable error. However, it is rare to occured on systems and can mean, for example, a corrupted boot environment, a hardware component that is vital for system startup fails or is missing or just a critical application, service or other software component exited unexpectedly;
- *Error*: This errors denote conditions that will cause the system to stop working correctly but that might be recoverable. Some examples are: a missing or failing non-vital services, applications or other software components, hardware on which an application depends is inaccessible or a network connection that is required for correct functionality is failing;
- *Warning*: These types opf messages appear when something that is concerning but not causing the operation to abort. The condition might become a problem in the future resulting in an error, or might not. Runtime situations that are undesirable,

unexpected and potentially lead to an error or cause application oddities, but everything still under control. Automatic recovery from the situation exists and the application can continue executing;

- *Info*: Should give an overview of major state changes providing high level context for understanding any warnings or errors that also occur. It can be used on runtime events that are normal but somehow important;
- *Debug*: It is has a level of detailed, diagnostically helpful, information for programmers, normally to use only when debugging a program. This should only be used for development and testing and disabled for production systems;
- *Verbose*: Even more fine-grained information than DEBUG like information about arrays or memory segments or information about loops and iterations.

Amount of log types						
ECU ID	Fatal	Error	Warning	Info	Debug	Verbose
BMTR	0	347	1344	344731	1060	489175
WAVS	0	0	0	53	0	0
WAVA	0	44	175	2237	0	0
WAVM	0	0	0	0	0	0
WAVG	0	0	0	0	0	0
BMT	0	0	0	0	0	0

TABLE 4.1: Types of log messages in each ECU.

As we can see in Table 4.1, most of the Log Messages of type Info come from the ECU called BMTR which can have a lot information depending how many and what applications are being performed in this ECU. It is also worth mentioning that some ECUs such as *WAVM*, *WAVG* and *BMT* did not produce any type of Log Message Type.

Another analysis was performed to see if there was any information about the comunication bus which can be identified by the DLT_NETWORK_MESSAGE type in the DLT logs. This type is not present in all DLT log messages and appear only in messages that need to transmit some kind of network syncronization or check status of the network. As we see in the Figure 4.4 the only bus communication used is Automotive Ethernet, which can reveal that this vehicle is not old, in fact modern vehicles are starting using automotive ethernet. It should also be noted that there are other communications that do not use a bus channel on the network but a communication channel of the ECU itself called IPC

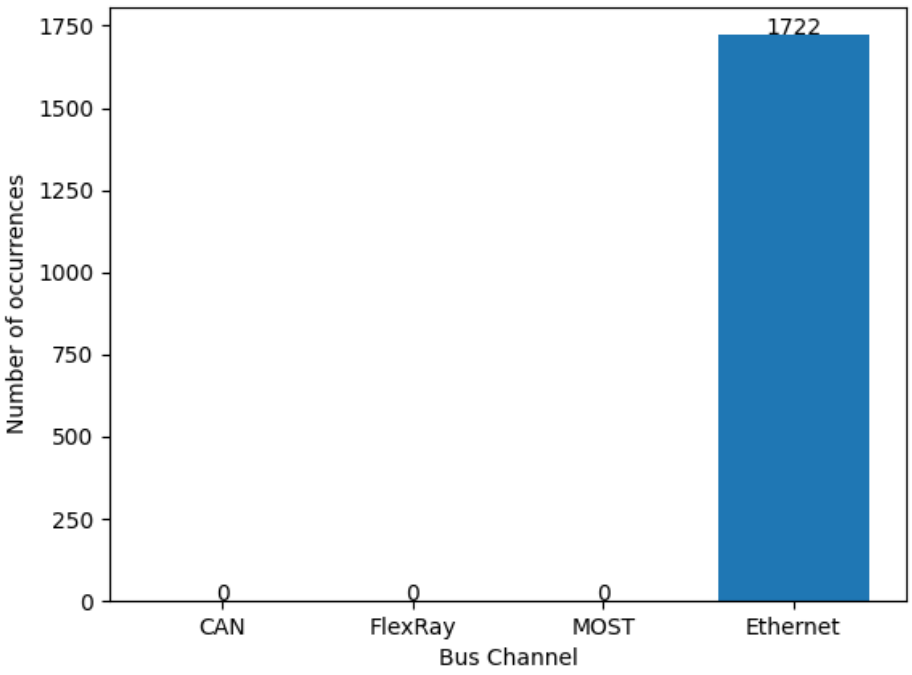


FIGURE 4.4: Number of logs per Bus channel.

(Inter-Process Communication) which refers to the communications of different applications to communicate with others on the same ECU.

Through this dataset, we also hoped that it would be possible to observe updates to the firmware of the different ECUs, but after fully analyzing this dataset of PCAP logs, it was not possible to find any evidence of the existence of updates to firmware or specific software for the ECUs.

However, we not only analyzed the DLT logs of this dataset but also analyzed other information present in this dataset in relation to the vehicle's internal network. We observed that communications between ECUs were carried out through the use of IP addresses and more precisely the use of ports to access certain applications/programs that were running within each ECU.

Regarding the analysis of logs with other protocols in this dataset, the SOME/IP protocol was the second protocol that noticed the most interest because in relation to the occurrences in this dataset it was the second type of log that appeared more often. Thus, a more detailed analysis was made about it to see if it would be possible to have more information about the architecture of the vehicle's network.

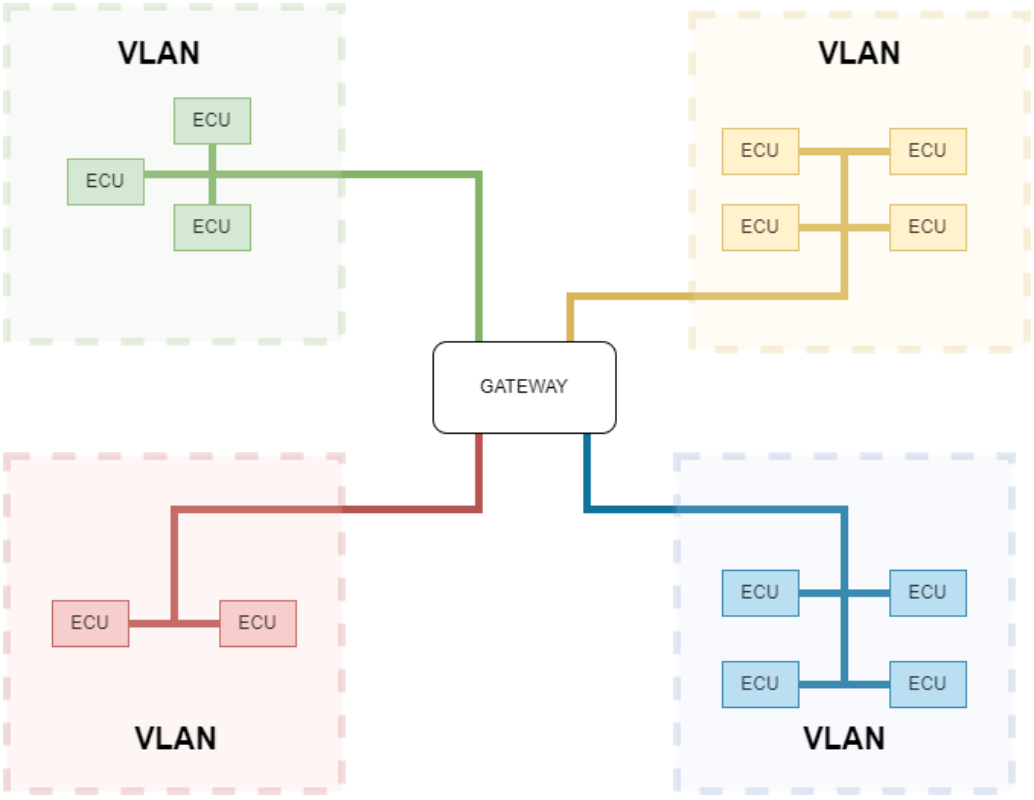


FIGURE 4.5: Organization of vlans in the dataset.

Therefore, at the end of the dataset analysis, it was concluded that the various ECUs are distributed by several subnetworks independent of each other through the use of VLANs. It was necessary to understand if these ECUs were connected to a gateway so that ECUs from one VLAN could communicate with other ECUs from a different VLAN. Thus, by noting this observation, we started to suppose that the architecture of this vehicle was organized by domains in which there were several domains in which each domain was associated with a different VLAN and all VLANs were connected to a single ECU that would serve as a gateway for the network, as shown in the Figure 4.5.

So, concluding with all the analyzes made to the dataset through reverse engineering, we arrived at a system architecture that can be seen in the following Figure 4.6. This architecture shows the various communications of the ECUs, as well as their IP addresses and respective ports. We can see that this architecture shows the various ECUs, some with ID others without ID because the ECUs without ID were obtained by SOMEIP logs in which the only field that identified them different from others was their IP.

4.2 Security Analysis

4.2.1 Assumptions

Now we will move on to an analysis of the security level present in this dataset. Through this dataset it was also possible to verify that the existence of security in terms of communications, logs and ECUs was not verified. Another point that can be observed was the lack of security in communications between ECUs, so if there is no security in terms of communications, several attacks can occur in these systems inside the vehicle network. This lack of security happens because the architectures and networks of the vehicles are closed systems in which there is no interaction with other networks outside the vehicle. However, due to the increase in the introduction of new features in vehicles, these systems begin to relate to other networks and devices outside the vehicle.

4.2.2 Attaker Capabilities

Some attacks that can happen on the vehicle's network based in this dataset and if there is no security to prevent it are the following:

- **Eavesdropping**: In this passive form of attack, the attacker listens to the communication without the systyem being aware of it. The confidentiality of the transmitted messages are compromised and using the collected information of the system comunication, the attacker can compromise the security of the vehicle;
- **Denial of Service (DoS)** The DoS attacks comprise a group of attacks that target network service availability. The attackers' primary objective lies in disrupting the means of communication and disturbing normal services such that they are not available. The attacker intentionally floods the control channel with a large volume of messages so that the ECUs cannot handle such a huge amount of messages, resulting in network disturbances;
- **Message Tampering Attack** The attacker launches this attack to modify, delete or alter a specific part of the message to fulfill its malicious intentions;
- **Drop-Request Attack**: The attacker blocks the network traffic outside or inside the vehicle to prevent an ECU from receiving any updates;

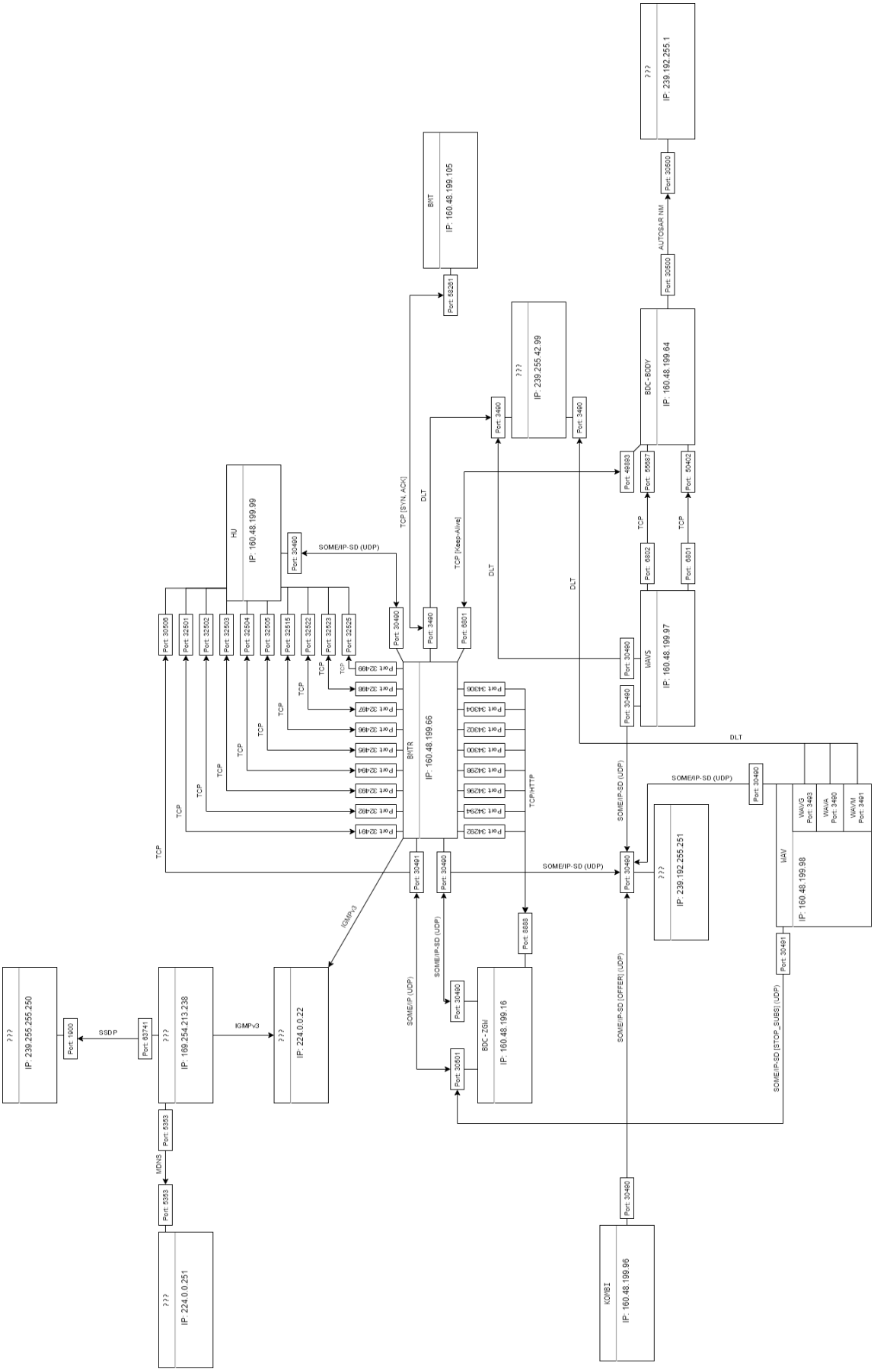


FIGURE 4.6: Dataset architecture

- **Partial Installation Attack**: The attacker my cause some ECUs to not install the latest updates. Attackers can do this by dropping traffic of these ECUs. However, sometimes this attack may happen accidentally, such as if updates are interrupted due to running out of power;
- **Software Attack**: Attackers can cause an ECU to install software of the attacker's choosing. This means that the attacker can arbitrarily modify the vehicle's performance. They overwrite the software on an ECU with malicious software.

Thus, due to the lack of security features in this dataset there is the need to find solutions that help to mitigate these attacks. One of the simplest solutions is to implement an IDS to detect these attacks in near real time by inspecting possible malicious logs and monitoring certain behaviors on the vehicle's internal network. One of the motivations of this thesis is precisely to develop an automotive logging framework, which could simplify the further analysis necessary to implement IDS-like systems, and help mitigating attacks and reinforcing security in vehicle networks.

Chapter 5

Creating a Centralized In-Vehicle Data Store

5.1 Description

The increasing amount of ECUs and the greatly increasing data size pose many unresolved challenges to modern vehicles, such as cost and safety. In particular, as already mentioned in Chapter 2, an ECU requires storage and security capabilities to keep critical data safe, such as software/firmware, backups, configurations, logs, software dependencies from other ECUs, etc. Therefore, there are two paths that the industry can choose between at this time:

- **Expensive Vehicles:** As there are more and more ECUs in the vehicle, it will cost more and reduce competitiveness in the market. So, for example, if the ECUs have two storage modules (A/B), instead of a single module (A), this allows that if the software or firmware that is stored in one of the modules fails to boot the ECU, the second module will load a backup of a version of software or firmware that works on that ECU of the second storage module the existing ECU. This solution solves several challenges regarding data security and the ECUs themselves but will cost more than a simple ECU with only one storage module.
- **Unsafe Vehicles:** As vehicle manufacturers do not want to increase prices in vehicles they tend to reduce the capabilities of ECUs, and then the costs of the ECUs is less, leading to various security threats. Therefore, if an ECU only has a single module, and with the data size increasing, the module will reach its maximum memory

capacity, leading to loss of information, such as logs, before being analyzed. Another problem with the existence of a single storage module is the fact that if one of the software/firmware fails in the ECU boot sequence, it will not have the backup module, causing problems and even the ECU not working.

Thus, as shown in Figure 5.1, the approach that is proposed and presented in this thesis, to improve the solution to the problem mentioned at the beginning of this section, is the creation of a Vehicle Secure Data Store (VSDS) in the vehicle, in which there is a dedicated ECU with memory capacity and computational power greater than normal, with the responsibility for example for storing software/firmware backups and diagnostic logs of the remaining ECUs on the vehicles network, so that the remaining ECUs on the network will be able to securely access this master ECU to obtain data, such as a version of a software/firmware that has been updated and failed for some reason. This master ECU will also have the function of storing all the Logs that were generated by the ECUs on the network, thus allowing, if later on, the manufacturers could analyze these Logs to fix errors or damages in the ECUs themselves or specific software, thus evolving the characteristics of these devices. Some advantages of this system are:

- **Cost Reduction**: Reduces the need for the number of ECUs with two storage modules, as the low cost of these ECUs is not necessary and the security elements are not crucial;
- **Safety**: Store useful data that used to be deleted, eg. old backups and logs for analysis and diagnosis;
- **Efficiency**: Store system level settings in an accessible location instead of redundancy on each ECU;
- **Security**: Leveraging VSDS security features to strengthen data security.