Dottorato di Ricerca in
Computer Engineering and science
Scuola di Dottorato in
Information and Communication Technologies

XXXII CICLO

Università degli studi di MODENA e REGGIO EMILIA

TESI PER IL CONSEGUIMENTO DEL TITOLO DI DOTTORE DI RICERCA

Defending internal network communications of modern vehicles from cyber-attacks

Candidato:
Ing. Dario Stabili

Relatore:
Prof. Mirco Marchetti
Correlatore:
Prof. Michele Colajanni
Coordinatore:
Prof. Sonia Bergamaschi

Abstract

Italiano

I veicoli moderni rappresentano un esempio rilevante di un sistema cyber-fisico estremamente diffuso. I meccanismi controllati tramite software spaziano da semplici funzionalità attivate dall'autista, come i tergicristalli o i finestrini elettrici, fino a funzionalità più automatizzate, complesse, e riguardanti sistemi legati alla sicurezza fisica, come il controllo motore, il servosterzo, il sistema elettronico di stabilità, o il sistema anti-bloccaggio. Queste funzionalità sono efficaci nel ridurre il numero totale di incidenti automobilistici e morti stradali. Nonostante tutto però offrono nuove vie agli attaccanti informatici, che possono esplorare e sfruttare un'ampia gamma di attacchi software control la logica di controllo implementata nelle centraline. Queste minacce non sono solo teoriche. Recenti studi e inchieste giornalistiche hanno dimostrato gli effetti di diversi attacchi informatici contro veicoli non modificati, sfruttando le connessioni cellulari per ottenere accessi non autorizzati nella rete interna del veicolo e controllare le funzionalità legate al motore, ai freni, e al sistema di sterzo. Questa tesi propone diverse soluzioni per aumentare la sicurezza informatica delle reti interne dei veicoli moderni, analizzando il ciclo di vita completo a partire dalla prevenzione degli attacchi fino al ripristino delle condizioni sicure a seguito del rilevamento di una intrusione. La prevenzione degli attacchi informatici richiede l'adozione di protocolli di comunicazione sicuri che includono integrità e autenticazione delle comunicazioni per comunicazioni interne che richiedono elevata sicurezza. In questo campo, questa tesi esplora i diversi compromessi richiesti dalle strategie di gestione di materiale crittografico, considerando il ciclo di vita del veicolo. La rilevazione di attacchi informatici rappresenta l'argomento principale di questa tesi, proponendo diversi algoritmi di rilevazione delle intrusioni progettati particolarmente per essere efficaci nella rilevazione di intrusioni sulle reti interne dei moderni autoveicoli. Tutti gli algoritmi proposti sono validati e testati tramite analisi sperimentale su dati provenienti da un veicolo reale. Per sopperire alle limitazioni causate dalla mancanza di specifiche pubbliche delle comunicazioni interne dei veicoli, questa tesi propone inoltre un algoritmo per l'estrazione automatica di segnali da pacchetti di dato provenienti dalle reti veicolari interne.

Questo algoritmo permette inoltre di applicare metodologie di rilevazione di anomalie più precise sui singoli segnali estratti. Infine, questa tesi propone due nuove strategie per ripristinare il veicolo in uno stato sicuro a seguito di intrusioni non autorizzate. Il primo approccio utilizza tecniche adattate dalla teoria dei controlli, mentre il secondo approccio sfrutta la modalità *limp-home* (un meccanismo di protezione presente nelle centraline) a servizio della sicurezza informatica.

English

Modern vehicles represent a prominent example of widespread cyber-physical systems. Mechanisms controlled by software range from simple tasks activated by drivers, such as windshield wipers or power windows, to completely automated, complex and safety-relevant systems, such as engine control, power steering, Electronic Stability Program (ESP) or the Anti-lock Braking System (ABS). These features are effective in reducing the overall number of car accidents and fatalities. However they also open new avenue for cyber-attackers, that can now explore (and possibly exploit) a wide range of software-based attacks against the control logic implemented by ECUs. Similar threats are not only theoretical. Recent research and media reports showcased several cyber-attacks against recent, unmodified licensed vehicles, which exploited cellular connections to penetrate the automotive network and obtain remote control over the engine, brakes and power steering systems. This thesis proposes many solutions for improving the cyber-security of the internal network communications of modern vehicles, and addresses the whole cyber-security life-cycle ranging from the prevention of cyber-attacks to their detection in operational vehicles and up to the proposal of automatic countermeasure that can mitigate the physical consequences of cyber-attacks. Prevention of cyber-attacks requires the adoption of secure protocols that include integrity and authentication guarantees for safety-relevant in-vehicle communications. In this field this thesis explores the trade-offs among different strategies for the management and distribution of cryptographic material, taking into consideration the full life-cycle of a modern vehicle. Attack detection represents the main focus of this thesis, that proposes several novel intrusion detection algorithms specifically designed for the detection of realistic cyber-attacks against modern internal vehicle networks. All the proposed intrusion detection algorithms have been validated through experiments carried out over real communications among ECUs, gathered from modern unmodified vehicles. To overcome the limitations caused by the lack of public specifications of internal communications in real vehicles, this thesis also proposes a novel algorithm for automatic reverse-engineering of automotive dataframes that allows to apply more fine-grained intrusion detection algorithms. Finally, the thesis proposes two novel strategies for reacting to a detected cyber-attack. The first is based on approaches borrowed from the control theory domain. The second leverages the limp-home mode (a protection mechanism already implemented by ECUs) in the service of cybersecurity.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cyber-physical systems (CPS) are highly integrated mechanisms in which one or more subsystems are monitored and controlled by software, possibly with a high degree of autonomy and minimal requirements of external inputs coming from users. A prominent example of widespread cyber-physical systems is represented by modern passenger vehicles, that are composed by many mechanical parts controlled by Electronic Control Units (ECUs), which are programmed to perform different tasks in the automotive system. Mechanisms controlled through ECUs range from simple tasks activated by drivers, such as windshield wipers or power windows, to completely automated, complex and real-time systems, such as engine control, power steering, Electronic Stability Program (ESP) or the Anti-lock Braking System (ABS). These software-driven safety-relevant features are extremely effective in reducing the overall number of car accidents and fatalities. However they also open new ways for cyber-attackers, that can now explore (and possibly exploit) a wide range of software-based attacks against the control logic implemented by ECUs. These threats are also magnified by the current trend toward an increasing connectivity of modern vehicles. It is now common that vehicles integrates Bluetooth connectivity with smartphones (hence an indirect connection to the Internet) or direct Internet connectivity through cellular networks.

Similar threats are not only theoretical. Recent research and media reports showcased several cyber-attacks against modern, unmodified licensed vehicles, which exploited cellular connections to penetrate the automotive network and obtain remote control over the engine, brakes and power steering systems. These recent works exposed different vulnerabilities of the networking protocols and communication buses enabling communication among safety-relevant ECUs. These systems are based on outdated standards, that have been designed for simpler ECUs and completely isolated networks, and do not provide any security guarantee. The lack of proper security mechanisms resulted in several unauthorized accesses to the internal networks of the vehicle, allowing attackers to sniff, spoof, and forge arbitrary messages. Several works already proposed the application of anomaly detection algorithms to protect internal vehicle networks, despite limited evaluation over real data gathered from one of the many in-vehicle networks.

This thesis proposes many solutions for improving the cyber-security of the internal network communications of modern vehicles, and addresses the whole cyber-security life-cycle ranging from the prevention of cyber-attacks to their detection in operational vehicles and up to the proposal of automatic countermeasure that can mitigate the physical consequences of cyber-attacks.

Anomaly detection represents the main focus of this thesis. The vehicle system is one of most prominent examples of cyber-physical system, thus it is necessary to inspect the security solutions related to the vehicle cyber-physical system instead of focusing only on the inspection of the communication protocols. Many different anomaly detection algorithms have already been published that addresses security on cyber-physical systems, but none of the previous works consider the particular scenario of the automotive cyber-physical system. This thesis proposes five novel algorithms for the anomaly detection specifically designed for the automotive domain. All the proposed algorithms have been validated through experiments carried out over real communications among ECUs, gathered from modern unmodified vehicles. The proposed algorithms meet the hard computational and memory constraints of common automotive ECUs. The detection results of the proposed algorithms are compared over the same real dataset (in-vehicle messages generated by a recent, unmodified, licensed car). Moreover, this thesis also presents a comprehensive anomaly detection framework based on these original algorithms. A further improvement of the state of the art is represented by a novel algorithm for the automatic reverse engineering of automotive messages. This algorithm enables fine-grained analysis of individual signals even without knowing their proprietary specifications. The signals extracted with this algorithm are then classified based on their evolution over time.

Prevention of cyber-attacks requires the adoption of secure protocols that include integrity and authentication guarantees for safety-relevant in-vehicle communications. Existing solutions proposed lightweight cryptographic protocols for guaranteeing data integrity and authenticity that are effective in preventing many classes of attacks, e.g., by adding some sort of message authentication mechanisms that allows well-behaving ECUs to recognize and reject illicit messages injected in the CAN bus by an attacker. However they do not consider all the implications of the management of cryptographic material for the whole life-cycle of huge vehicle fleets. This thesis advances the state of the art by analyzing existing solutions and highlighting their consequences on the whole vehicle manufacturing, from the design to its disposal.

The only reaction method already published in the literature focuses on the proposal of a notification framework designed for the dashboard of modern vehicles. This thesis presents an approach for mitigating the physical consequences of cyber-attacks to the automotive systems based on security countermeasures borrowed from the control theory domain to increase the security of the powertrain system from cyber-attacks. Moreover, this thesis also presents a concept for another reaction mechanism that leverages the *limp-home* mode (a protection mechanism implemented in ECUs) for preventing mechanical damages to the hijacked vehicle and limiting the vehicles functionality to a safe operational mode, activating additional security counter-measures.

To summarize, the contributions to the state-of-the-art presented in this thesis are the following:

- This thesis proposes 5 novel anomaly detection algorithms specifically designed for the analysis of the CAN bus and without requiring the full vehicle's specifications. The proposed algorithms are designed to meet the hard constraints of modern microcontrollers.

- This thesis proposes a novel reverse engineering algorithm developed for extracting and label meaningful signals from CAN data frames without requiring the full vehicle's specifications.

- This thesis proposes two different anomaly reaction mechanisms for internal vehicle networks, one based on the application of control theory solutions and one designed to exploit existing safety mechanisms deployed in modern vehicles. The former solution is tested in a simulated environment, while the latter provides an accurate and in-depth description of its functioning.

This thesis is organized as follows. Chapter 2 presents the technical fundamentals required for the understanding of the following Chapters and describes the realistic threat model considered in this thesis. Chapter 3 presents the state-of-the-art in the automotive cyber security literature and highlights the gaps that this thesis aims to fill. Chapter 4 explores the trade-offs among different strategies for the management and distribution of cryptographic material, considering the full life-cycle of a modern vehicle. Chapter 5 presents several novel intrusion detection algorithms specifically designed for the detection of realistic cyber-attacks against modern internal vehicle networks. Chapter 6 describes an algorithm for the automatic reverse-engineering of automotive messages. Analysis of cyber-attacks to the cyber-physical model representing the powertrain section of a generic combustion engine vehicle are proposed in Chapter 7, which also presents a novel reaction methodology based on control theory techniques. Chapter 8 describes a reaction strategy that limits the potential damage of a detected attack. Finally, Chapter 9 presents the final remarks and conclusion of this thesis.

Chapter 2

Background and Threat Model

This chapter introduces the fundamentals required for the understanding of this thesis. A brief introduction of the networking protocols deployed in modern vehicles is provided in Section 2.1, while an accurate description of the Controller Area Network protocol is provided in Section 2.2. The threat model describing real vulnerabilities which modern automotive networks are exposed to are described in Section 2.3.

2.1 Vehicular Networks

Modern vehicles are complex cyber-physical systems composed by many microcontrollers called Electronic Control Units (ECUs). ECUs implement all the control logic of software-driven features, including many safety-critical functions such as the steering and braking, and many other less-critical features such as the air conditioning and the infotainment system. ECUs are physically connected to the vehicle network, and communicate among each others through specialized protocols that satisfy all functional requirements of the automotive domain, such as real-time capabilities and message prioritization, while guaranteeing the lowest possible production costs.

Multiple networks can co-exist within the same vehicle to isolate different functionalities, but a single network usually allows to satisfy multiple purposes optimizing costs. As an example, entry-level vehicles might be provided with only one network that connect ECUs involved in safety-critical features and other ECUs implementing the infotainment system. This design choice is one of the causes of recent successful cyber-attacks to vehicle networks, in which attackers exploited vulnerable connectivity interfaces exposed by the infotainment system to access safety-critical functions [90].

Vehicular networks are designed to meet the specific requirements of the automotive scenario:

- low cost;
- plug-in capability;
- immunity from external noises;
- ability to operate in harsh environments;
- overall robustness and reliability.

Some of the most deployed in-vehicle networks are:

- **Controller Area Network (CAN):** is one of the first networking protocols designed specifically for automotive applications. CAN is a broadcast protocol and the bus is composed only by a twisted pair, thus increasing electromagnetic resilience and containing deployment costs. Since its first proposal, CAN has been deployed on different industrial scenarios and not only on internal vehicular networks. CAN is the most deployed networking protocol in modern vehicles and is the one targeted by cyber-analysts or hackers to subvert the vehicle system [29, 45, 59]. Since CAN is the primary focus of this thesis, the details of the CAN bus are depicted in Section 2.2.

- **Local Interconnect Network (LIN):** is a serial network developed as a cheaper alternative to CAN. LIN offers the same capabilities of the CAN bus but on restricted networks (up to 16 nodes on each network segment) arranged in a star topology.

- **Media Oriented System Transport (MOST):** is a high-speed multimedia network technology optimized for automotive industry. MOST uses a ring topology network and synchronous data communication for audio, video, voice and data signal via plastic optic fiber.

- **FlexRay:** is an automotive network communication protocol developed to enable ECUs to communicate with each others. It is designed to be more reliable and faster than CAN, despite being more expensive.

2.2 Controller Area Network

The Controller Area Network (CAN) is an asynchronous serial communication protocol which supports distributed real-time control, targeted to industrial applications and designed to allow data exchange among microcontrollers without requiring a host computer. This technology represents the *de-facto* standard for implementing in-vehicle communication buses among Electronic Control Units deployed in modern cars. The current and final version of the CAN specification is CAN 2.0, published in 1991 [26]. CAN specification is composed by two parts: part A refers to the standard CAN format, while part B refers to the extended format. The only difference between the two formats is that the standard format uses 11-bits identifiers, while the extended format uses identifiers composed by 29-bits. CAN devices using 11-bits identifiers are usually referred to as CAN 2.0A, while devices using 29-bits identifiers are referred to as CAN 2.0B.

Synchronization on the CAN protocol is achieved at node level to align all the nodes on a particular CAN segment to the same bit sample point. The synchronization mechanism of the nodes is started at network setup (i.e. at the ignition of the vehicle), and the time required for synchronization is called *nominal bit time*. The nominal bit time is composed by four sequences of non-overlapping time segments, as depicted in Figure 2.1.

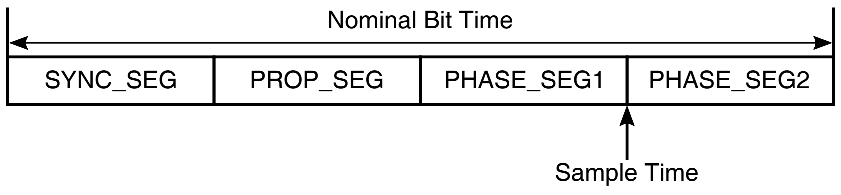


Figure 2.1: Controller Area Network bit time partitioning

The four time segments used for partitioning the nominal bit time are:

1. **SYNC_SEG:** part of the bit time used for the synchronization of the nodes. An edge is expected in this time segment.

2. **PROG_SEG:** part of the bit time used for the compensation of the physical delay of the signal.

3. **PHASE_SEG1/2:** parts of the bit time used for the compensation of the edge phase error.

The bit sample point is the instant in which the bus level is read and interpreted as the value of that particular bit, and is located between **PHASE_SEG 1** and **PHASE_SEG 2**.

The duration of the nominal bit time is evaluated from the nominal baud rate as show in Equation 2.1.

$$t_{\mathcal{B}} = \frac{1}{BR} \tag{2.1}$$

The CAN protocol supports baud rates up to 1MBps, despite its most common deployment uses 500kbps (for high-speed CAN) and 125kbps (for low-speed CAN). Synchronization of the nodes is required for data transmission on the network, thus nodes on the network are re-synchronized if needed in between data transmission. The re-synchronization mechanism is achieved via a bit-stuffing mechanism which inserts one bit of opposite polarity every 5 bits of equal polarity. This mechanism is necessary due to the *non-return to zero encoding* used in the CAN. The stuffed messages are de-stuffed by the receivers upon reception. Data transmission on the CAN bus uses a loss-less bit-wise arbitration method for contention resolution. The CAN specification defines each bit sent on the network as either *"dominant"* (logical value 0, actively driven to voltage level by the transmitter) or *"recessive"* (logical value 1, passively driven to ground level by the transmitter). The idle state of the network is represented by the recessive value. During data transmission, if one node sends a dominant value on the network and another node sends a recessive value, the node sending the dominant value of the network will win arbitration and any collision on the network is avoided, thus allowing to send high-priority messages on the network without any delay due to transmission errors. To avoid collisions between nodes, each transmitting node reads the logical value of the bus after writing each bit of the message. If any transmitting node reads a bus value different from the one it has written on the bus it stops transmission and attempts re-transmission of the message after the current transmission is concluded by the sender. In the example depicted in Figure 2.2 three sending nodes are concurrently sending their data on the CAN bus.

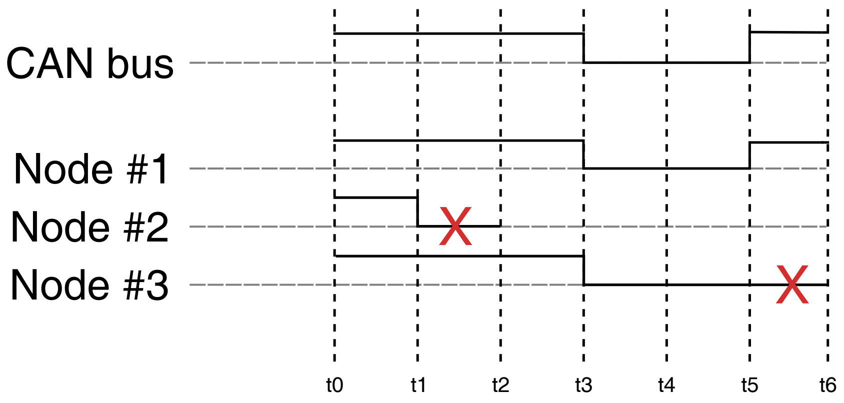


Figure 2.2: Data transmission arbitration on CAN

At time t_0 transmission is started, and each node sends a dominant value. At time t_1 Node #2 sends a recessive value, while both Node #1 and Node

#3 send a dominant value, thus Node #2 loses arbitration and stops its transmission. Nodes #1 and #3 both send the same value on the bus until time t_5, where Node #3 sends a recessive value, thus losing arbitration to Node #1. After time t_5 the only sending node on the network is Node #1, while each other node will not re-attempt any transmission until Node #1 has finished. Data transmission in the CAN protocol is defined as a message-based communication between different microcontrollers. The CAN protocol defines four different message frames:

- **Data frame**, which carries data from the transmitter node to the receivers;
- **Remote frame**, which is transmitted from a node to request the transmission of a specific data frame;
- **Error frame**, which is transmitted by any node upon detection of errors on the bus;
- **Overload frame**, which is transmitted to provide an extra delay between two consecutive data or remote frames.

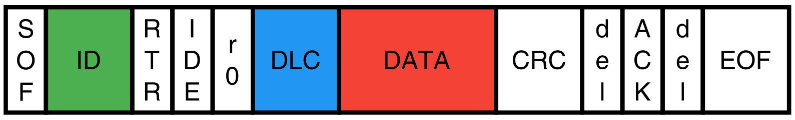
Data and remote frames are separated from previous frames by an **interframe space**. The interframe space contains the bit fields *intermission* and *bus idle*, while part of the *bus idle* field is composed by the *suspend transmission* field for the only node that sent the last message on the network. The *intermission field* is composed by 3 recessive bits, and while the intermission is sent on the network the nodes are not allowed to start transmission of either data nor remote frame. Following the *intermission* field the *bus idle* state is reached, during which an indefinite number of recessive values are found on the CAN bus. The first 8 bits of this indefinite sequence are sent from the node which has transmitted the last message, with the meaning of *suspend transmission* for the transmitting node, thus preventing it to start the transmission of its next message.

The description of the different frames is provided in the following paragraphs.

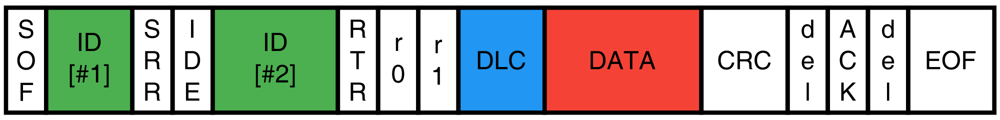
2.2.1 Data frame

The CAN data frame carries data from the transmitter node to the receivers. A graphical representation of the structures of the standard and extended CAN data frames are shown in Figure 2.3a and 2.3b, respectively.

The main fields composing the CAN data frame are:



(a) Base format



(b) Extended format

Figure 2.3: Data frame types comparison

- **Start-of-Frame (SoF):** a single bit set to dominant level;
- **Identifier (ID):** the identifier (ID) value assigned to the message. Its value is used to interpret the values encoded in the data field. IDs need to be unique on each CAN segment. The length of the message ID is defined at 11 bits in the standard format, while the extended format has 18 more bits encoded separately for compatibility between the two formats.
- **Remote Transmission Request (RTR):** a single bit that must be dominant for data frames and recessive for remote frames. If the message is in the extended format, this bit value is defined as *substitute remote request (SRR)* and must be recessive, while the bit with RTR meaning is placed after the second part of the message identifier.
- **Identifier Extension Bit (IDE):** a single bit that must be dominant for messages in the base format and recessive for messages in the extended format.
- **Reserved bit (r_0):** a single reserved bit in the base format, while there are two reserved bits (r_0 and r_1) in the extended format.
- **Data Length Code (DLC):** a sequence of 4 bits expressing the length (in bytes) of the following data field. Since the data field has a maximum length of 64 bits, DLC values representing lengths from 9 to 15 are left unused.
- **Data:** sequence of bits with a variable length (ranging from 0 up to 64) containing the data sent from the transmitter to the receivers.
- **Cyclic Redundancy Check (CRC):** field containing the CRC evaluated on the previous fields of the data frame. The CRC is evaluated

on the previous bits of the message with a CRC function designed for frames with less than 127 bits. For the evaluation of the CRC a generator polynomial is defined by the protocol and depicted in Equation 2.2. The CRC is evaluated as the reminder of the polynomial division (coefficients are evaluated modulo-2) between the polynomial created with the coefficients of the message's bits (from the SoF field to the data, if available) with 15 bits set to 0 as the lowest coefficients, and the generator polynomial.

$$X^{15} + X^{14} + X^{10} + X^{8} + X^{7} + X^{4} + X^{3} + 1 \tag{2.2}$$

- **CRC delimiter:** a single recessive bit used to delimit the previous CRC sequence.
- **Acknowledgment slot (ACK):** a single bit used for the acknowledgment of the message. This bit is sent by the transmitter node as a recessive value. The CRC is used for determine the consistency of the messages between transmitter and receivers. If the received CRC and the CRC evaluated by the receiver node are different, than the message is marked as not consistent and not acknowledged by the receiver nodes by writing a dominant value on the bus in this time slot.
- **ACK delimiter:** a single recessive bit used to delimit the previous ACK slot.
- **End-of-Frame (Eof):** a sequence of 7 recessive bits.

These fields are grouped in different frame segments:

- **Arbitration field:** composed by the ID and the RTR fields in case of base format, while in case of extended format it is composed by the first 11 bits of the ID, the SRR, the IDE, the other 18 bits composing the ID and the RTR;
- **Control field:** composed by the IDE, r_0, and the DLC field case of base format, while in the extended format it is composed by r_0, r_1 and the DLC field;
- **Data field:** composed by the data field;
- **CRC field:** composed by the CRC and the CRC delimiter.

The bit-stuffing mechanism is applied to all the fields composing the data frame with exception of the *CRC delimiter, ACK field* and *EoF*, that have a fixed form.

2.2.2 Remote frame

The CAN remote frame is sent from any receiver node in case a particular data frame is required for its functioning. The receiver node sends a remote frame with the ID value set to the ID corresponding to the required message. The remote frame structure is the same of the data frame (as already depicted in Figure 2.3), and it is possible to send remote frames in both standard and extended format. The differences between a data frame and a remote frame resides in the value of the RTR field (dominant for data frames, recessive for remote frames), and the absence of Data field in the remote frame. The different value of the RTR field allows data frames to win arbitration of the network in case a data frame and a remote frame with the same arbitration field are sent on the network concurrently. An example of a standard format remote frame is depicted in Figure 2.4.

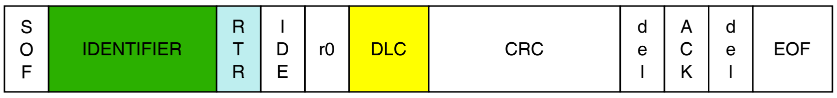


Figure 2.4: CAN Remote frame structure

2.2.3 Error frame

The CAN error frame is sent on the network from any node upon detection of a network error. The error frame is composed by two different fields:

- **Error flags:** from 6 to 12 different bits, of which the first 6 bits are of equal value (either dominant or recessive), while the other bits are flag errors raised by different nodes of the network.
- **Error delimiter:** this field is composed by 8 recessive bits and it is used to delimit the end of the error frame.

Each node on the network is in either *error active* or *error passive* state. If a node is in error active state, upon detection of a network error it sends the error flag as a sequence of 6 dominant bits, while an error passive node transmits a sequence of 6 recessive bits. Upon detection of an error active flag, all the nodes on the network start sending their error flags. The final sequence of dominant bits observed on the network is the result of the superposition of different error flags transmitted by each different node. The error flag is terminated by at least 3 bit times of *bus idle* state. Each node on the network has two different error counters, namely **Transmit Error Counter (TEC)** or **Receive Error Counter (REC)**, which are increased by the node upon detection of a transmission or reception error, respectively. When at least one between TEC or REC is greater than 127 and lower than 255, the node is in error passive state and upon detection of an error it sends the relative error flag. When both TEC and REC are smaller or equal than 127, the node is in error active state and will send an active error flag upon detection of errors. If TEC is greater than 255 the node enters the *bus-off state*,which prevents the node to send any other frame on the network.

A graphical representation of the CAN error frame is depicted in Figure 2.5, while Figure 2.6 depicts the error handling mechanism used by the nodes.

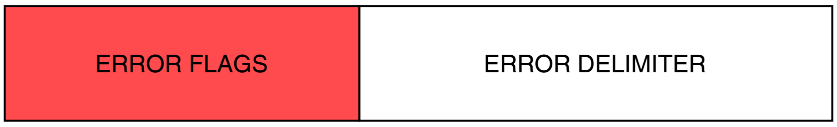


Figure 2.5: CAN error frame structure

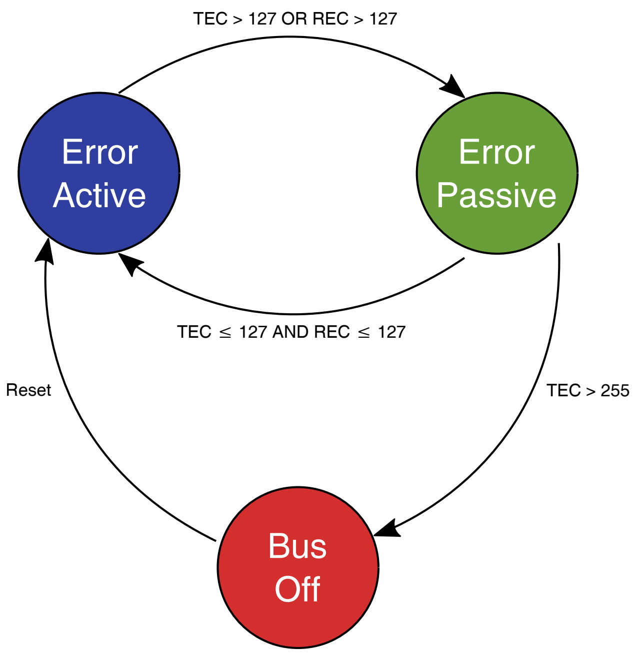


Figure 2.6: CAN error handling mechanism

2.2.4 Overload frame

Overload frames are sent on the CAN bus when it is necessary to increase the delay between two consecutive data or remote frames. The overload frame is composed by two different fields:

- **Overload Flag:** composed by 6 dominant bits, which corresponds to the *active error flag.*
- **Overload Delimiter:** composed by 8 recessive bits, used to denote the end of an overload flag.

Overload frames are sent by receiver nodes in case they require a delay of the next Data or Remote frame due to internal conditions; otherwise they are sent by any node of the network upon detection of a dominant bit transmitted during *intermission.* The start of an overload frame due to the former condition is only allowed at the first bit time of an expected *intermission*, while the overload frame is sent at the next bit following the detection of the dominant bit in case of the latter condition. At most two overload frames are allowed between two consecutive data or remote frames.

A graphical representation of the CAN overload frame is depicted in Figure 2.7.

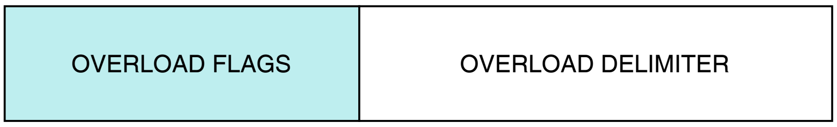


Figure 2.7: CAN overload frame structure

2.3 Threat Model

While addressing the cyber-security of internal network communication of modern vehicles it is necessary to define the different scenarios inspected by both academia and industry researchers demonstrated vulnerable to real threats [29, 44, 45, 59, 90]. The threat model inspected in this thesis is focused on the *Controller Area Network* protocol, which is the in-vehicle protocol most exposed to real threats. The CAN protocol enables communications via a message-oriented paradigm among groups of ECUs by using the CAN ID field. In the definition of the internal vehicle network by car makers, each ECU is programmed to send messages with a particular pool of IDs and only reads messages with a defined set of IDs. Since CAN uses a broadcast channel, only ECUs programmed to read messages with a particular set of IDs forward the content of the message to the operational level of the microcontroller, while other ECUs only checks for integrity of the message by CRC comparison. Similarly to most protocols for cyber-physical systems (e.g., ModBus for SCADA applications), CAN does not include any cyber security functionality because its original design assumed its usage in a physically isolated system without external interfaces. As a result, any ECU can inject messages with arbitrary field values on the CAN bus. To protect against cyber attacks, researchers are putting efforts in extending the CAN protocol. In the following, an outline on how it is possible to attack vehicles networks is proposed.

Cyber-attacks to modern vehicles aim to modify the dynamic of the vehicle or a target safety-relevant feature through the multi-step process described as follows:

1. obtain a privileged access to the vehicle network;
2. acquire information about the protocol adopted in the network;
3. inject malicious messages in the vehicle network to manipulate the information processed by the target ECU.

(1.) As represented in Figure 2.8, a modern vehicle offers *internal*, *edge* and *remote* attack surfaces:

- the *internal* surface represents any type of direct access to a segment of the vehicle network via a compromised ECU or by physically hijacking the twisted pair of the CAN bus. In this thesis a powerful attacker that managed to gain full control of an ECU connected to the same CAN bus segment of the target ECU is considered. This scenario allows the attacker the to access both volatile and persistent memory, and to execute arbitrary code on the compromised ECU [52];
- the *edge* surface represents cabled interfaces originally exposed within the vehicle, usually for diagnostic reasons, such as the On-Board Diagnostics (OBD-II) port of the CAN. It is necessary to assume that the manufacturer assigned limited privileges to this interface, such as read-only access to the messages exchanged on the network. Note that if the manufacturer exposes directly the CAN bus on this interfaces, then this attack surface falls back to the *internal* case;
- the *remote* surface represents external interfaces provided by wireless communication services, such as Bluetooth or WiFi, possibly connected to the Internet. Breaking security mechanisms of these interfaces, or of related exposed applications, might allow attackers to obtain privileged access to the ECUs that implement them and escalate to *internal* access privileges.

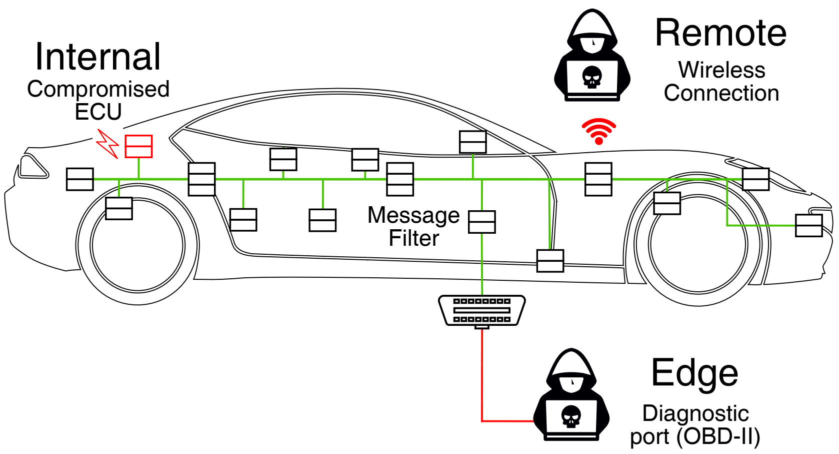


Figure 2.8: Attack surfaces of a modern vehicle

(2.) A successful targeted attack to the vehicle network requires detailed knowledge about the syntax and semantic of messages transmitted over the CAN bus. Although some aspects of the syntax of vehicle protocols are mandated by public standards, each manufacturer adopts different choices to encode message identifiers and signals within the message payloads. As an example, although the CAN specifications are public, each vehicle uses different CAN IDs and represent information by using different encoding techniques (e.g., the engine revolutions per minute are represented through different binary encoding and scales). This information is kept confidential among car manufactures and suppliers and it is not released in the public domain, thus the attacker has to apply reverse-engineering techniques to infer these information from the CAN traffic [56, 73]. If the attack is designed to target a specific vehicle model, the attacker can run this preliminary phase as an offline operation on an identical vehicle at his disposal. If the attacker targets a wide-range of vehicles, reverse-engineering is executed by sniffing the traffic within the vehicle network [90]. As an example, *fuzzing attacks* have been proved effective to gain useful insights about the vehicle network topology and to map the connected ECUs [51].

(3.) A compromised ECU connected to the CAN network is able to operate many types of safety-critical attacks.

Known examples include: *ECU shutdown* [69] and *ECU impersonation* [16] (also, *masquerade* attack) to activate the *bus-off state* mode on a target ECU, thus excluding the ECU from the network and mimicking its behavior by sending properly forged payloads to subvert the vehicle functions; *denial-of-service* or *replay* attacks that are able to manipulate the normal behavior of the vehicle dynamic. All of these attacks can be executed from different

ECUs of the vehicle. However, certain scenarios might require that a specific ECU processes some fake data injected in the network, while others only require that any ECU accepts and processes the forged data. As an example, a typical trade-off is to design security measures that isolate ECUs of different segments of the network, such as powertrain, body, infotainment or a group identified by a CAN ID.

A description of the attack scenarios inspected in this thesis is provided in the following Sections. These attacks represent the different attack scenarios that exposed different vulnerabilities already exploited by security researchers.

2.3.1 Replay Attack

Replay attacks on automotive internal networks are conducted by injecting messages (or sequence of messages) on the CAN bus. The sequence of messages used for the injection on the CAN bus is selected by the attacker after an initial phase of reverse engineering of the values encoded in the payload, thus injecting messages to obtaining control over the dynamics of the vehicle. For the purposes of this thesis, two different typologies of replay attack are considered, according to the length of the injected message sequence:

- **Single ID Replay:** A Single Message ID is injected on the normal flow of the CAN bus with a particular frequency;

- **Sequence Replay:** A sequence of messages is injected on the normal flow of the CAN bus with a particular frequency. The length of the injected sequence is a parameter that will be inspected in the detection process.

2.3.2 Fuzzing Attack

Fuzzing attacks on automotive internal networks consist in the injection of messages forged to study the behavior of the system against unexpected inputs. Fuzzing is one of the activities used by reverse engineers to understand the meaning of the values encoded in the payload of a targeted message ID. Fuzzing can be subdivided in two different typologies according to the injected message:

- **ID Fuzzing:** Injection of messages with IDs values never observed on the CAN bus and randomly generated payloads;

- **Payload Fuzzing:** Injection of messages with IDs values already observed on the CAN bus and maliciously forged payload values.

The former attack is usually deployed as a method to discover potential unused IDs which carries information to different parts of the vehicle, while the latter attack is used in the manual reverse engineer process of the payloads of a particular inspected message ID.

2.3.3 Denial of Service

Denial of Service (DoS) of automotive internal networks consists in the injection of messages with the highest priority to prevent all the other messages from being transmitted on the bus. Since transmission on the CAN bus uses a bit-wise arbitration method for contention resolution, a denial-of-service attack on the CAN bus requires the injection of data or remote frames composed by sequences of dominant bits in the arbitration field, thus preventing any other node to obtain access to the network for its normal operation. Denial-of-Service attacks on the CAN bus can be simulated in two different ways:

- **Highest priority message injection:** Injection of messages with the highest priority value in the arbitration field from the set of messages of the inspected vehicle;
- **Dominant message injection:** Injection of messages with all the bits composing the arbitration field set to dominant value.

In both Denial-of-Service attack scenarios, any attacker willing to reproduce successfully a CAN DoS must avoid the transmission of the *suspend transmission* bits following the *interframe space* as described in Section 2.2.

2.3.4 ECU shutdown and inhibition attacks

ECU shutdown and inhibition attacks are two different attacks designed to remove messages from the CAN bus. The ECU Shutdown attack temporarily removes messages with a target message ID from the network, while the ECU inhibition attack completely removes the target message ID by disabling the sender node. As a consequence of both ECU Shutdown and ECU inhibition attacks, the attacker can replace the valid ECU by sending malicious messages on the CAN bus with the removed ID by impersonating the offline ECU. This attack is also known as *ECU impersonation*, and its consequences have been already inspected by security researchers in [15, 69].

Chapter 3

Related work

The work proposed in this thesis has been inspired by the lack of many different aspects in the automotive cyber-security literature aimed to improve the security of the internal communication networks of modern vehicles. This Chapter describes the security solutions already proposed in literature and analyzes the missing aspects of each inspected group of solutions.

3.1 Security solutions for intra-vehicular networks

In this Section an analysis of the existing proposals for the design of secure vehicle networks is provided. In this analysis, two main challenges in the extension of the CAN protocol with security guarantees are found: the design of a secure transport protocol to authenticate data (Section 3.1.1) and the distribution of the required cryptographic material (keys, certificates, shared secrets) to all interested ECUs (Section 3.1.2).

3.1.1 Secure transport protocol for CAN

Standard security solutions designed to guarantee data integrity and authenticity usually compute and concatenate to the message a tag generated with a Message Authentication Code (MAC) protocol. This approach guarantees that any illegitimate modification on the data (i.e., by someone that does not know the secret key) can be detected by the recipient. However, the CAN protocol uses fixed small-size packets and low bandwidth channels that makes it unfeasible to attach a MAC to each message. The simplest way to deploy efficient and secure authenticated protocols would be to use intra-vehicular network protocols that are more flexible than CAN, such as the CAN+ extension [97]. This protocol supports larger messages and allows to associate MAC tags to messages in a more standard fashion [91]. However, car manufacturers are not prone to adopt them due to the increased costs. Two approaches for extending the CAN protocol with guarantees of data authenticity are identified. The first approach is to allow each ECU to produce additional CAN messages to transmit MAC tags. This approach offers the best security, however it also introduces huge network overhead that makes it unfeasible in most automotive networks. The second approach represents a trade-off between performance and security. It requires each ECU to authenticate batch of messages by using a single MAC tag, to fragment it and to send the fragments within CAN headers of the messages [67]. The main disadvantage of the approach is that a recipient can verify authenticity of data only once every few messages: since CAN is a real-time protocol and ECUs process messages as soon as they are received, an attacker can inject malicious messages without being detected for a certain time interval. A second disadvantage is that MAC fragments are transmitted within existing fields of CAN headers, such as the CRC, possibly affecting existing protocols design.

Guaranteeing protection against replay attacks also requires additional design choices at the application and architectural levels, as typical for standard communication protocols. In the context of vehicle networks, proposals exist based on centralized time-servers [36], distributed counters [52] and key-derivation approaches [74]. However, it is necessary to highlight that these solutions do not impact architectural design choices because they do not involve the distribution of additional persistent cryptographic material. Thus, the adoption of any of these solutions is orthogonal to the analysis proposed in Chapter 4.

3.1.2 Intra-vehicular ECU keys distribution

Three types of approaches to distribute key material to the ECUs are considered: *pre-shared ECU keys*, *in-vehicle key distribution centers* and *certificate-based key authentication.*

Pre-shared ECU keys. The first approach to deploy secure protocols is to install symmetric keys in the ECUs persistent storage, where the same key is deployed within all ECUs that must communicate [5, 6]. Different key distribution strategies can be adopted to obtain different trade-offs in terms of storage overhead and security guarantees. As an example, a master key could be stored in all ECUs or multiple group keys could be selectively stored in ECUs depending on their roles within the vehicle (e.g., their associated CAN IDs).

In-vehicle key distribution centers. The second approach based on symmetric cryptography requires to install additional ECUs within the vehicle that act as Key Distribution Center (KDC) [30,36,68]. Each KDC knows the secret keys of a subset of ECUs and releases the due session keys to enable pair-wise or group communications. As in standard secure communication protocols, this approach enables good security guarantees, low storage overhead and easier management of persistent keys distribution. In vehicle networks, it has the disadvantage of introducing additional costs due to the additional ECUs and some network overhead.

Certificate-based key authentication. The last approach is to adopt primitives based on asymmetric cryptography, such as digital signatures and certificates, to deploy solutions that are similar to the Internet Public Key Infrastructure (PKI) architecture [63]. Each ECU is configured with a list of trusted Certification Authorities (CAs) and stores a key pair signed by one of the trusted CAs. ECUs can communicate with each other by exchanging symmetric session keys by using key exchange protocols. An optional variant of this architecture is to also include a specialized ECU within the vehicle that can revoke invalid certificates. Despite great advantages in terms of security and easier management, most ECUs have tight resource constraints and do not support asymmetric cryptography. Thus, deploying these solutions require the addition of specialized Hardware Security Modules with increased costs.

3.2 Anomaly Detection

Anomaly detection refers to the problem of identifying rare items, events, or observations that are suspicious and significantly different from the majority of the data [98]. Another definition of anomaly detection is provided in [10], in which the authors define the anomaly detection as the problem of finding patterns in data that do not conform to the expected behavior. In both these works, the authors refer to two different terms: anomaly and outlier. The term anomaly is used to address the process of identifying something unexpected depending to the context: an anomalous traffic pattern in computer network usually implies that a hacked device is sending sensitive data to unauthorized destination [49], while an anomalous MRI image may indicate the presence of malignant tumor [79]. The term outlier is used for the definition of statistical anomalies in the data, which has been studied in the statistics community from the late 19th century [54].

As a more general approach, it is possible to define an anomaly as a pattern that does not conform to the expected behavior, and different straightforward approaches in the definition of the region representing the normal behavior could be used. Classification-based anomaly detection techniques are one of the most popular approaches, in which the normal model is created from a set of labeled data and a test instance is classified into one of the classes defined in the model [22, 84]. This approach requires data to be labeled and equally distributed in the different classes, otherwise it is not possible to equally compare the classes for the classification of the test instances. Nearest neighbor-based anomaly detection techniques are an extension of the classification-based anomaly detection techniques, in which only valid data is used for the definition of the normal groups. Nearest neighbor-based anomaly detection techniques are based on the assumption that normal data instances occur in dense neighborhoods, while anomalies occur far from their closest neighbors, thus requiring a distance measure between two data instances [7,12]. This approach allows to create detection models based on the analysis of valid data, thus not requiring malicious data to be in the training dataset. Another anomaly detection technique uses clustering to group similar data instances into clusters [42, 84]. Clustering-based anomaly detection techniques are primarily based on unsupervised learning, despite semi-supervised clustering has also been explored by researchers [4]. Clustering-based anomaly detection techniques allow the creation of clusters of data (similar to the nearest neighbor-based anomaly detection techniques) without requiring labeled data. This class of anomaly detection techniques automatically generates the clusters of data by inspecting different features, thus further reducing the data labeling issue. A similar approach that does not require labeled data but only requires legit data are the statistical-based anomaly detection techniques, which uses the underlying principle that *"an anomaly is an observation which is suspected of being partially or wholly irrelevant because it is not generated by the stochastic model assumed"* [2]. Statistical anomaly detection techniques use a statistical model created for the description of the normal behavior of the given data and then apply a statistical inference test to determine if an unseen instance belongs to this model or not. Instances that have a low probability of being generated from the model, based on the applied test statistic, are declared as anomalies. Both parametric as well as non-parametric techniques have been applied to fit a statistical model. While parametric techniques assume the knowledge of the underlying distribution and estimate the parameters from the given data [24], non-parametric techniques do not generally assume knowledge of the underlying distribution [20].

For the purposes of this thesis, novel solutions designed to increase the detection capabilities of current state-of-the-art detection techniques for the cybersecurity of automotive networks are proposed in Chapters 5 and 6.

3.3 Information security for automotive networks

Several works already depicted different attack typologies that can be performed by sniffing and injecting messages over the CAN bus [11,37], as well as by mangling messages used by the Tire Pressure Monitoring System [77] and passive key-less entry systems [25]. Of particular interest are papers describing attacks carried out remotely by exploiting the permanent Internet connection of *connected* vehicles [41, 59]. The first remarkable work that highlights the potential vulnerabilities of modern vehicles is presented in [48], where the authors exposed and demonstrated the vulnerabilities related to in-vehicle networks and microcontrollers that can control safety-relevant features of a vehicle. These vulnerabilities have already been proven real and exploitable by malicious attackers, and quickly gained media visibility [29, 45, 59]. These works inspired many cyber-security researchers from both academia and industries to inspect the connected vehicle vulnerabilities and to propose effective countermeasures to mitigate the problem. More related approaches propose the implementation of anomaly detectors to analyze CAN traffic and identify possible anomalies related to malicious activities. Several different detectors are proposed in [65] and [81], however these works do not provide an experimental evaluation of their proposals. Other approaches for anomaly detection in the internal vehicle networks are based on Support Vector Machines (SVM) [32, 88, 92]. However, it is necessary to remark that ECUs are microcontrollers with limited computational power and small memory, thus many of these solutions might require heavy modifications before being deployable to these devices.

Many state-of-the-art intrusion detection systems designed for the CAN have already been proposed, and different aspects of the network are inspected to create the normal reference used for the definition of the normal operational region.

CAN communications are composed by messages sent cyclically between ECUs on the shared bus, thus one of the features inspected for the definition of an Intrusion Detection System for in-vehicle networks is frequency of the messages. The authors of [27] proposed an algorithm that inspects the inter-arrival time of messages with the same ID to detect possible attacks. In their attack model they consider both cases of injection of messages from an external device or from a compromised internal ECU. The authors of [62] further inspect the inter-arrival time of the message IDs, and provide a practical demonstration of the capabilities of their detection algorithm against a simulated Denial-of-Service attack. These two proposals have been proven capable to achieve good detection results, but the limited adversary model on which both algorithms are applied prevents to consider the frequency of the messages as a complete documented feature for the detection of anomalies on the CAN bus. Another frequency-related approach has been proposed in [85], where a periodical inspection of the CAN bus allows the evaluation of a *score* index over the data collected in a particular time interval, by means of a *One-Class Support Vector Machine (OCSVM)*. The result of the OCSVM is used to detect the injection of malicious packets or to identify missing packets from the network. The approach proposed in [85] is tested against the injection of malicious packets with different frequencies, deploying normal models created with different parameters for the detection phase and demonstrating that models created with higher frequency values achieve better detection rates than lower-frequency ones. Despite the extensive practical evaluation, the authors do not provide any detail on the configuration parameters for their implementation of the OCSVM, thus making their work not reproducible.

A different approach based on the inter-arrival time of CAN messages has been proposed in [16], where an innovative method that exploits the time intervals of messages to fingerprint the sender ECU is used. This method allows the detection of malicious messages and identifies the ECUs sending the messages on the network. The authors tested their algorithm against different adversary models, such as *single ID replay*, *ECU shutdown* and *ECU masquerading* (as consequence of the ECU shutdown attack) with high detection rates. The authors also claim that their algorithm performs badly against the detection of messages sent by an ECU sending non periodic messages. Compared to previous works, the work described in [16] provides highly accurate description of the methods and formulae used for their implementation and the described adversary model is comparable to the one described in this thesis.

However, none of the previous work highlighted the hazards involved in the inspection of only the message inter-arrival times on the CAN bus. There are particular messages in a vehicular network that can be either periodical or non-periodical, depending on the scenarios. For example, consider the case of two CAN messages related to the Cruise Control (CC), in which one message encodes the state of the CC in its payload (active/off) while the other encodes the cruise speed in its payload. The latter message frequency depends on the value encoded in the former message. It is logical to assume that, if the CC is in set to *off*, then the message carrying the cruise speed would not appear on the CAN bus, while if the CC status is *active*, the message carrying the cruise speed is expected to appear on the bus with a high frequency, since it carries safety-critical functioning information. In this scenario it is logical to assume that, for messages whose periodicity is related to the status of some functionality of the vehicle, it is impossible to create an accurate detection model based on the frequency of message inter-arrival times.

This thesis work further inspects the capabilities of a frequency-based anomaly detection algorithm for the CAN bus by proposing an algorithm designed for the detection of a *bus-off* attack on the CAN bus in Section 5.2. The proposed algorithm is tested against two different attack scenarios and proven extremely effective in the detection of bus-off attacks.

Different IDS based on the analyses of statistical features of the vehicular networks have already been proposed in literature. The authors of [64] designed an IDS based on the inspection of the *entropy* of the data of the CAN bus, observing sensitive variation of the entropy value in case of injection of malicious packets. This work evaluated that the entropy value depends on the frequency of injection of the messages. However, this detection method does not allow the identification of the anomalous message, thus another method based on the *self-information entropy* evaluated between consecutive values of the same signal encoded in the message payload is also proposed in [64]. This method allows to identify the injection of malicious values of signal if the difference from the expected value if significant, resulting unnoticed in case of injection of values very close to the real ones. However, the experimental evaluation of this attack is limited, and spans over just 15 seconds of CAN traffic including only a single class of CAN messages that are not safety-relevant. This thesis work further inspects the boundaries of detection algorithms based on the entropy measure by proposing a novel algorithm in Section 5.5 that inspects the mean entropy value over variable window sizes. The novel algorithm is tested against message replay and fuzzing attacks, providing an extensive experimental evaluation. Moreover, since it is possible to evaluate the load of the bus and that any attack on the CAN bus could affect the utilization of the transmission network, an algorithm that inspects the bus utilization is proposed in Section 5.6. This algorithm inspects the normal utilization of the bus and raises an anomaly when the utilization is outside of the defined normal utilization.

Other detection algorithms are based on the inspection of the fields of the CAN data frames. The simplest approach is to detect CAN bus messages having an invalid ID [53]. In this case, the normal model is just a set of valid message IDs, either gathered from the formal specification of a given vehicle, or learned by sniffing messages from the CAN bus. This approach allows easy and precise identification of attacks that inject CAN messages with an invalid ID, but can be easily foiled by attackers clever enough to inject arbitrary messages with valid IDs. Hence this approach can be useful against basic fuzzing techniques [51], but becomes useless against more sophisticated or determined attacks. Since messages in the CAN bus follows a particular frequency it is possible to assume that there are recurrent patterns between messages flowing on the CAN bus, which can be inspected for the detection of unknown patterns between message IDs. For this purpose, in Section 5.3 an algorithm that inspects the sequences of message IDs of CAN data frames is proposed, and its detection capabilities are evaluated with an extensive experimental evaluation. Another approach to design an IDS for vehicular networks is proposed in [43], where deep neural networks are investigated as a possible solution to detect anomalies in the payloads of the message. Despite the novelty of the proposed approach, current ECUs deployed in the vehicles have not enough computational power to implement the detection logic on-board. In Section 5.4 an algorithm for the anomaly detection of messages based on the inspection of the payloads is proposed. This algorithm evaluates the difference between consecutive payloads of the same ID by means of the Hamming distance. This design choice has several advantages with respect to the state of the art. First, it does not require full knowledge of the syntax and semantics of CAN messages. Normal features can be learned just by analyzing traffic traces sniffed from a licensed vehicle. Moreover, experimental evaluations demonstrate that the proposed algorithm is able to detect stealth attacks that involve the injection of very few CAN messages. It is also necessary to remark that the solution proposed to improve the CAN bus security complies with the hardware constraints of a typical automotive ECUs, having very low memory and computational requirements. Other researches focused their effort in the identification normal electrical characteristics of CAN transceivers in [14] and [46]. Both these methods inspect the voltage levels of the transceivers as their detection metric, and use sophisticated analysis based on the inspection of the voltage level for the identification of the transmitting ECU. Despite these solutions are proven effective in the detection of anomalies injected on the CAN bus, both focuses on the identification of the ECU transmitting the message. The work presented in this manuscript uses a wider attack scenario, thus considering the ability of the attacker to subvert the normal behavior of the single ECU on the network.

3.4 Reverse Engineering of automotive messages

Cyber attacks to modern vehicles executed by injecting forged and malicious messages in the CAN bus [15, 44, 58, 59] spawned several research efforts aimed to improve the security level of modern vehicles. Some works aim to improve the security of communications over the CAN bus by applying cryptographic protocols [31, 74]. However similar solutions require to modify all the ECUs involved in secure communication and have profound impacts on the whole life-cycle of the vehicle. Other less intrusive approaches apply anomaly detection [10] and security analytic algorithms to the traffic flowing on the CAN bus. Several algorithms for the identification of intrusions over the CAN bus have already been proposed, mainly by applying and adapting approaches borrowed from the IT and network security domain to the specific characteristics of the CAN bus and its messages.

It is possible to observe that the main limitation shared by all the aforementioned research efforts lies in the very few features that can be extracted and analyzed from a generic traffic trace containing CAN messages. Indeed, message arrival time, ID and the binary blob of its payload only enable a very coarse-grained message classification, that can be useful in detecting simple attacks but is bound to fail in detecting more stealth intrusions comprising the injection of few well-designed malicious messages. This issue could be mitigated by having access to the complete formal specifications of CAN messages, including the list and boundaries of all signals encoded in their payload. Unfortunately this information is only available to car makers and their suppliers, and cannot be accessed by the general public, including the vast majority of academic researchers. Knowledge of the semantic of CAN messages would also be extremely useful in reconstructing the state of the vehicle before a crash, possibly identifying driver mistakes, failures of the vehicle, or anomalous activities attributable to a cyber attack. Authors of [66] are the first to address this issue by proposing a list of requirements for detection, data collection and event reconstruction following a crash. This aspect has been further inspected by the authors of [55], who proposed a reliable, secure, privacy-preserving and efficient mechanism to build a forensics data collection and storage system. Despite those solutions, it is clear that analysis of raw CAN messages require to manually inspect high volumes of data to reverse-engineer messages syntax and semantic, reconstruct the vehicle dynamic and contextualize the messages and their contents.

The network traffic analysis literature already includes many proposals aiming to automatically recognize the nature of a given network packet or flow, as an example by attributing a network communication to a specific application or protocol. These works are mostly based on three main approaches: matching of known signatures within network packets; analysis of source and destination port numbers at the transport layer; applying a classification algorithm to packet metadata [8, 61, 95]. It is necessary to remark that these works have been designed to analyze TCP/IP network traffic and to identify only well known network applications (such as web browsing, email, chat and file transfer protocols). All these assumptions make these approaches inapplicable to the automotive domain characterized by in-vehicle networks that leverage completely different protocols and communication patterns. As an example, CAN messages do not include source and destination addresses nor port numbers, are broadcast communications that do not establish a bidirectional communication flow, and lack the clear separation between network, transport and application layers that are typical of IT networks.

Some information about semantic and syntax of CAN messages can be extrapolated through reverse engineering, as proposed in [58, 59, 78]. However, all these approaches are based on manual inspection of a high number of CAN bus messages by a reverse engineer with experience in the automotive domain, that is a daunting and human-intensive task.

The first proposal toward automatic reverse engineering of signals conveyed in CAN messages, specifically aimed to providing more useful features for anomaly detection algorithms, can be found in [57]. This work proposes an algorithm that analyzes the payload of CAN messages and tries to extract signals and their boundaries by observing how the payloads of messages sharing the same ID evolve over time. However, the heuristics proposed in [57] have never been tested against real CAN messages, since their experimental evaluation is based on CAN traffic generated by a laboratory environment with simulated ECUs, rather than on real licensed vehicles.

This thesis work proposes a novel algorithm for the automatic identification of signals embedded in the payload of CAN messages (Section 6.1) that outperforms previous work [57] by detecting more than twice the number of correct signals and exhibiting much lower execution times. Rather than being an incremental improvement over [57], the novel algorithm includes a completely novel set of heuristics and a completely different processing algorithm. These heuristics reflect the domain knowledge acquired by authors while manually reverse-engineering CAN messages generated by several real and modern vehicles of different models and makers, and facilitate the reverse engineering process by automatically extracting and labeling individual signals from unknown CAN traffic traces.

Moreover, this algorithm automatically associates a descriptive label to all extracted signals that helps a human analyst in making sense of the data. The labels used by the algorithm are specific to the automotive domain and convey a precise semantic meaning, while labels produced by [57] only depend on how the signal evolves over time and do not try to describe its meaning. Finally, the results of this algorithm and the one proposed in [57] and compared against the complete formal specifications of a real, licensed, unmodified road vehicle.

3.5 Cyber-security of control systems

Modern vehicles are composed by different and complex subsystems, each one related to one of the different parts of the vehicle ecosystem. One of these subsystems is the powertrain, which is one of the most analyzes subsystem in literature [13,21]. The powertrain subsystem has been inspected for the proposal of novel models that allow to increase the performances of the vehicle or to reduce its emissions. One of the most established fields of research related to the powertrain is the cruise control, which is a system often deployed in conjunction with the powertrain that allows to remove the human component from the powertrain management, thus allowing the controller to change the vehicle speed according to the necessities. Despite these two systems are heavily interconnected to each other, the cruise control system has only been inspected in conjunction with the model representing the vehicle dynamic, and never inspected in conjunction with the powertrain on which it is deployed. Since cyber-attacks to vehicle system often includes the injection of maliciously-forged messages on the internal vehicle networks [58], to inspect the consequences of these attacks it is necessary to describe the model on which these attack are conducted. A generic attack comprises different steps, from remote exploitation of wireless connection to the re-programming of the critical ECUs to allow external messages to reach the internal network of the vehicle [29,45,59]. In Section 7.1 the consequences of the injection of different messages on the CAN bus are inspected by modeling the powertrain section of a generic internal combustion engine on which a cruise controller is deployed. Attacks are simulated on the system and its consequences are inspected, highlighting possible solutions aimed to increasing the security of similar safety-critical systems.

3.6 Reaction to detected cyber-attacks

The inspection of the different issues related to the proposal of reaction solutions for the mitigation of cyber attacks in vehicular networks is still a poorly addressed field of research. The only work available in literature that partially addresses the issues is proposed in [38], in which the authors inspected the different notifications typologies available on modern vehicles, through the instrument cluster or via the infotainment units. For this purpose, this thesis proposes a concept for a reaction mechanism in Chapter 8. The proposed concept is based on alerts generated from a security framework deployed in the internal vehicle network, enabling the analysis of the vehicle state and proposing the correct countermeasure depending from the severity of the detected attack and from the vehicle state. The proposed solution is focused on define a complete reaction framework, that also includes the solution proposed in [38] for the complete management of the reaction process.

Chapter 4

Cryptographic key management for modern vehicles

To prevent attackers from obtaining access to the internal vehicle networks, it is necessary to improve the defense mechanisms of the vehicle to provide higher security guarantees. Since this security analysis covers the complete fleet of vehicles manufactured by a car maker, it is necessary to inspect the whole vehicle life-cycle to deploy effective countermeasures.

4.1 Inter-vehicle security independence

To obtain unauthorized access to a vehicle, it is possible to hypothesize that a malicious attacker might use an extension of the multi-step approach described in Section 2.3, by leveraging the fact that multiple vehicles are produced through an industrial serialized methodology. If multiple vehicles share some secret information used in security protocols, the attacker can extract information about a target vehicle from another vehicle, of the same model or from the same Original Equipment Manufacturer (OEM). Intuitively, this might ease the reverse engineering process described in phase (2) of the threat model (Section 2.3) and it allows to obtain information about security protocols deployed in the vehicle and potentially of secret cryptographic keys. As an example, if different vehicles use the same cryptographic keys, even if OEMs use these keys to protect ECUs communications with high granularity, then the adversary can obtain the secret keys from a similar vehicle at his disposal, thus using them to compromise the security of another target vehicle. The security guarantee is defined as *inter-vehicle security independence*, which defend vehicles against these attacks, thus preventing an attacker from gaining advantages in accessing secret information stored in a vehicle by attacking any other vehicles.

This security guarantee is strictly related to security measures that protect devices against white box attacks. If the assumption that an adversary can have the same advantage in attacking a vehicle by having physical access to any other vehicle is considered, than deploying white-box security defenses such as temper-resistant hardware modules or white-box cryptography is of paramount importance. However, if it is possible to distribute independent keys for each vehicle and to guarantee inter-vehicle security independence, then car manufacturers might achieve similar levels of security without white-box defenses. Indeed, attacking a target vehicle would require the adversary to physically access that very same vehicle, that is a much weaker security assumption. This last case is by far the most critical, because it is based on the assumption that the attacker has gained enough knowledge of the internal network composition to forge a custom command targeted to a specific ECU. It is impossible to prevent any malicious attacker from gaining physical access to the vehicle, thus it is necessary to address the reproducibility of the cyber-attack as part of the inspected scenario.

4.2 Secure architectures for vehicles life-cycle

The vehicle life-cycle is modeled by considering three main actors:

- *OEM*: the company that designs and produces the vehicle;
- *supplier*: a company that produces vehicle components. It is responsible for providing maintenance, assistance and replacement parts during the vehicle life-cycle, that includes software in case of electronic components;
- *owner*: a person that buys the vehicle and uses it;
- *maintainer*: a company or a private that operates maintenance on the vehicle.

The vehicle life-cycle is represented as a finite-state machine model, where each state represents a phase in the life-cycle of the vehicle. In each state of the vehicle life-cycle an actor is associated (*authoritative* actor), which has exclusive access to the vehicle during the corresponding phase. The actor can cause a transition to another state of the life-cycle, possibly passing the authority over the vehicle to another actor. The model assumes that the actor associated to a state has physical access to the vehicle. Note that this does not imply that the actor has full access over the vehicle, as this might be limited by his knowledge of the vehicle components and his technical capabilities. As an example, it is possible to assume that a maintainer can accomplish advanced repair operations, but it is also necessary to assume that the owner might only be able to drive the vehicle.

In this paragraph a description of the details of the model is provided by referring to Figure 4.1, which shows the different states of the model and highlights the authoritative actor for each phase. The first state of the diagram is the *design* of the vehicular network, where the specifications of each ECU that will be deployed in the vehicle are defined. The authoritative actor of this state is the OEM. One of the many results of the *design* state is the ECUs specifications of the vehicle, which are given as input for the *production* step, where the software to be installed on the ECUs is generated. The ECUs hardware is bought from an external hardware producer directly by the supplier, which is the authoritative actor for this state. The OEM can designate different suppliers to produce part of the vehicle components or a single supplier to produce all the components. The third state of the vehicle life-cycle is the *assembling* state, in which the ECUs with the software already installed are delivered to the OEM and assembled together with the mechanical parts of the vehicle. The authoritative actor of this step is the OEM. After the assembling state the vehicle is available on the market, and after it is sold the *operational* state begins. The vehicle in this state is considered fully operational and at the disposal of the owner (which is also the authoritative actor for this state). During the operational phase both ordinary or extra-ordinary service operations are required, thus the maintainer gains control over the vehicle and the *maintenance* state is entered. Maintainers are the authoritative actors of this step, acts as intermediaries between the vehicle and the OEM, requiring special access to the vehicle components, its configuration and privileged access to parts of the vehicle if needed. Once the service is completed, the vehicle returns to the *operational* state. Multiple transitions between the *operational* and *maintenance* states are expected in the normal vehicle life-cycle.

Supporting security solutions for vehicle networks requires the design of an architecture that supports either distribution or access of the cryptographic material by actors involved in the life-cycle. In particular, the main non-trivial design choices regard the *production* and *assembling* phases implemented by the *supplier* and the *OEM*, that must generate and share inter-dependent secret information by obtaining the best security guarantees. Depending on the designed architecture, *maintainers* might have to interact in different ways to support their customers. In this analysis the details of each protocol required within the architecture are not inspected, thus no inspection is made on the information sharing protocols among players or the secure management of cryptographic keys, but peculiar traits of the architecture that depends

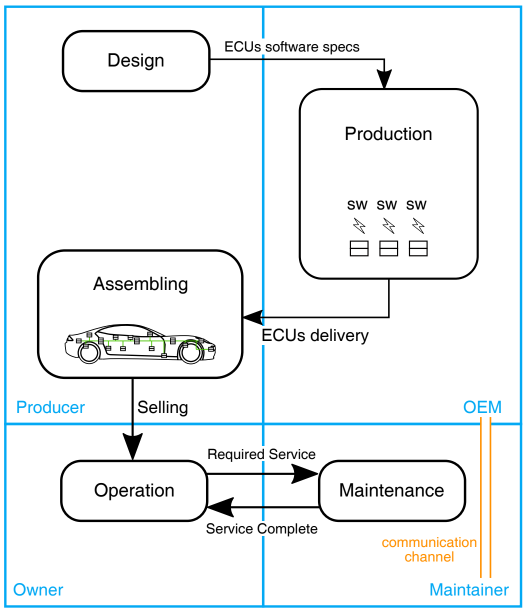


Figure 4.1: Vehicle life-cycle

from the adopted security solutions for intra-vehicle communications are highlighted for sake of completeness. Moreover, a discussion about how these architectures can guarantee the *inter-vehicle security independence* is proposed, as discussed in Section 2.3. In the following, existing security protocols are inspected and distinguished in *pre-shared ECU keys* (Section 4.2.1), *in-vehicle key distribution centers* (Section 4.2.2) and *certificate-based key authentication* (Section 4.2.3), as described in Chapter 3.

4.2.1 Pre-shared ECU keys

Deploying a security protocol based on pre-shared ECU keys requires multiple suppliers to share cryptographic keys with each other and with the OEM. This requirement is mandatory for any level of granularity, such as using a global master key, group keys, or pair-wise keys. However, some design choice might be influenced by key granularity. If multiple suppliers produce components that communicate with each other (e.g., associated to the same CAN IDs), then the OEM is the only player that has a global view of the system and that can take care of key generation and distribution. In this case, the OEM can decide to choose either a master, group or pair-wise keys and distribute them to suppliers accordingly. Otherwise, if a supplier has exclusive responsibility for a certain group and a group key strategy is used, then he can autonomously generate and manage the secret key. However, the main issue in these architectures is the generation and deployment of different cryptographic keys for different vehicles due to the reconciliation of the keys at the assembly phase. To enable inter-vehicle security independence, suppliers must install different keys on each component and keep track of the components that share the same keys. Then, the OEM should handle the reconciliation of all components that share secret keys to assemble them in the same vehicle. Although this kind of management seems theoretically feasible, it puts a lot of burden on both the suppliers and the OEM. Moreover, since components are not interchangeable, it introduces complex issues in case of failures. As a result, solutions based on pre-shared ECU keys do not seem a viable design choice to guarantee inter-vehicle security independence.

4.2.2 In-vehicle key distribution centers

Deploying a security protocol based on in-vehicle Key Distribution Centers (KDC) requires suppliers and the OEM to share pair-wise cryptographic keys between "normal" ECUs and the "special" ECU that implements the KDC (also, KDC-ECU). This class of solutions might be implemented by using different design strategies. To implement an efficient and scalable architecture, a solution might be to allow the supplier of the KDC-ECU to monitor the assembly phase. As an example, the OEM could maintain the production and management of the KDC-ECU in-house. By considering this assumption, the management of cryptographic keys can be implemented as following. The inspected scenario is based on the requirement that the supplier received orders by the OEM for a certain amount of components. The main objective is to store secure cryptographic material that allows each component to communicate with the KDC. Thus, at flashing time the supplier generates random keys (or it uses a key derivation function) and installs them in the ECU. Then, the supplier sends the ECUs together with keys to the OEM. Before the assembly phase, the KDC-ECU must contain the keys of all the ECUs that will be installed in the same vehicle. This operation seems feasible because all dependencies are resolved in the assembly phase. However, this architecture might require additional efforts to deploy maintenance operations. In case of ECU failures, substituting an ECU either requires to obtain a new ECU that stores the same key of the failed one or to update the KDC with the key of the new ECU. Either design choices could be deployed with some effort, although the second option, that would require an update of the KDC-ECU storage, seems more convenient. Indeed, requiring suppliers to flash a single ECU on-demand might be expensive.

4.2.3 Certificate-based key authentication

Deploying security protocols based on asymmetric cryptography enables the application of an operation flow that is similar to that of a standard PKI. The assumption behind this design choice is that each supplier generates a certain number of secret keys and, for each of these keys, it produces a Certificate Sign Request (CSR). All CSRs are issued to the OEM, that approves them and returns the corresponding certificates. In each ECU the supplier installs a secret key, the associated certificate, and the public key of the OEM. The software installed by the ECU will establish connections with ECUs that can produce certificates signed by the installed OEM public key. This architecture represents an efficient approach to install the due cryptographic material in the ECUs, and has the great advantage of not distributing secret keys outside suppliers and outside a single ECU. However, it does not seem able to guarantee inter-vehicle security independence. Implementing this security guarantee would require the OEM to use a different certificate for each vehicle and to sign the CSRs of the suppliers accordingly. Then, the OEM would have to reconcile ECUs as described for the pre-shared ECU keys approach, that seems an unfeasible task. As a result, to obtain inter-vehicle security independence, the introduction of a centralized point of control that allows to define authorization policy on a per-vehicle basis at assembly time is not an optional choice, even when asymmetric cryptography is used.

Chapter 5

Detection techniques based on the CAN standard

In this Chapter different algorithms for the detection of anomalies on the CAN bus based on CAN standard are proposed. These algorithms are designed to be trained from data gathered from raw CAN traces, as described in Section 2.2, thus without requiring any additional information. This Chapter provides an initial description of the dataset gathered from an unmodified, licensed vehicle in Section 5.1, which is required for the understanding of the different inspected algorithms. From Section 5.2 to Section 5.6 different detection techniques based on the CAN standard are proposed. Each one of the proposed detection algorithm is tested against the same dataset. A final comparison of the proposed algorithm is provided in Section 5.7, that proposes also a detection framework based on the inspected algorithms.

5.1 Dataset description

This Section proposes a description of the dataset used in the training and validation processes of the proposed algorithms. The description of the dataset containing simulated anomalies based on the threat model proposed in Section 2.3 is provided in Section 5.1.2. The latter part of the dataset is used for the experimental evaluation of the detection capabilities of the proposed algorithms.

5.1.1 Clean dataset description

The dataset used for the design and the test of the proposed algorithms is collected from the high-speed CAN buses of an unmodified, licensed 2016

Volvo V40 by physically connecting a laptop to the OBD-II port with a PCAN-USB adapter by Peak System [71] and a D-Sub to OBD-II cable. According to international standards, the high-speed CAN bus exposed on the OBD-II port is the *powertrain* segment, which is composed by ECUs that control different subsystems of the vehicle dynamic, such as the cruise control system, the anti-braking system, the electronic stability control, and many optional Advanced Driver Assistance Systems (ADAS). The CAN recording process is configured to save metadata information about the CAN traffic (such as the timestamp and the type of the message) in conjunction with the fields composing the messages (CAN ID, the DLC value, and the bytes composing the data field). These data are gathered during several driving sessions performed on different road types (urban, suburban and highways), traffic conditions, weather conditions and on different geographical areas (plain, hill and mountain). The whole dataset includes an aggregated amount of more than 10 hours of driving. The fields composing the dataset are:

- **Timestamp:** the relative timestamp of the message since the ignition of the vehicle;
- **Message type:** type of the message frame;
- **Message ID:** the identifier of the message;
- **Data Length Code:** the length of the message payload, expressed in bytes;
- **Payload:** hexadecimal representation of the payload of the message.

All the recorded messages are *data frames*, thus the message type field is removed from the traces. The final clean dataset is composed by 7 CAN traffic traces, including more than 8 million messages belonging to 50 unique message IDs. The clean dataset is publicly available at [80].

5.1.2 Infected dataset description

The detection performance of the detection algorithms are tested against a set of traces in which anomalies for each attack inspected in Section 2.3 are replicated. These attacks are simulated over the traces composing the clean dataset. To avoid any possible bias toward unrealistic detection results due to the different frequencies of messages composing the clean dataset, IDs with different probability distributions are used for the simulation of attacks on the clean dataset. The probability distribution of the dataset is depicted in Figure 5.1. On the *x-axis* of Figure 5.1 the IDs of the dataset are depicted omitting one value each two for readability purposes, while the probability distribution for each message ID is displayed on the *y-axis*. The bars highlighted with a different color represent IDs belonging to the 10_{th}, the 50_{th}, the 75_{th}, and the 100_{th} percentiles, which are used for the simulation of different attack scenarios on the CAN bus.

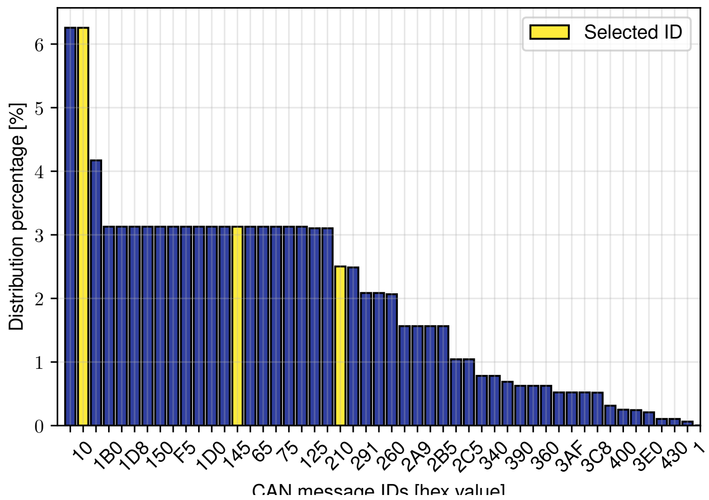


Figure 5.1: Distribution of probability of the IDs composing the dataset

The infected dataset is composed by the following attack scenarios:

- **Replay attack**
  - *Single ID:* Set of traces in which a single valid ID is injected. This set is composed by 4 different traces, each one corresponding to the injection of one of the 4 IDs representing the $10_{th}, 50_{th}, 75_{th}$ and 100_{th} percentile.
  - *Valid Sequence:* Set of traces in which a sequence of IDs observed in the traces is injected. Different traces are generated by injecting sequences with different length, ranging from 2 to 10;
  - *Invalid Sequence:* Set of traces in which a random generated sequence of valid IDs is injected. Different traces are generated by injecting sequences with different length, ranging from 2 to 10.
- **Fuzzing attack**
  - *Random ID:* Set of traces in which a single invalid ID is injected;

  - *Random payload:* Set of traces where a single valid ID is injected with randomly generated payloads. The valid ID used for the injection of a random payload is selected using the same criteria used for the single ID replay attack;

- **Denial of Service attack** In case of a Denial of Service attack, the selection of the duration is critical for the analysis of the detection capabilities of the algorithm. Since the duration time of the attack affects the detection capabilities, a denial of service attack is defined as an attack that is able to disrupt the normal communication on the internal network for at least twice the frequency of the most frequent message on the network. On the gathered dataset, the most frequent message has a cycle time of 0.01 seconds, thus the duration time for a denial of service attack is set to 0.02 seconds. Considering the scenario in which the CAN bus is configured with an operating baud rate of $500kbps$, to disrupt all the communication for the chosen time it is necessary to inject at least $10k$ bits. A generic CAN data frames composed by all the available fields has a minimum length of 108 bits and a maximum length of 134 bits by considering the the worst case for the bit-stuffing application, thus it is necessary to inject from 76.9(77) to 92.6(93) messages to successfully disrupt communications for at least 0.02 seconds. For the simulation of this attack scenario 100 messages have been injected. Two different typologies of denial of service are inspected:

  - *Lowest ID:* set of traces in which a sequence of messages with the ID equals to the the lowest value observed in the dataset is injected;
  - *Zero ID:* set of traces in which a sequence of messages with the ID equals to 0 is injected. *(Note that the "Lowest ID" and "Zero ID" denial of service attacks are the same in case the ID with value* 0 *is defined in the vehicle specifications and observed in the clean dataset).*

- **ECU shutdown** Set of traces where CAN messages are removed for a variable amount of time (ranging from 10 to 120 seconds) from the CAN bus, thus emulating an attacker that manages to drive in *bus-off* [15,69] a target ECU of a vehicle for a given amount of time.

- **ECU inhibition** Set of traces where CAN messages with a particular ID are completely removed. This attack scenario represents an attacker that is able to permanently disable a target ECU. This set is composed by 4 different traces, each one corresponding to the removal of one of the 4 IDs representing the $10_{th}, 50_{th}, 75_{th}$ and 100_{th} percentile.

5.2 Detection algorithm based on CAN timings

5.2.1 Fundamentals

Current literature already proposed different algorithms designed for the detection of anomalies in the CAN bus through inspection of the timings of the messages [53, 62]. In these works, the authors inspected different methodologies to create an anomaly detection algorithm based on the analysis of messages with the same ID. These algorithms have been proven able to detect an anomaly after a message is received, thus there are no guarantees of detection against both *ECU shutdown and inhibition* attacks since messages are permanently removed from the network, as described in Section 2.3.4. To provide a valuable contribution to the state of the art, the anomaly detection algorithm exploiting message timings for its detection purposes proposed in this thesis is designed to detect anomalies specifically for the scenario of a *ECU shutdown and inhibition* attack. As a preliminary analysis, the distribution of the inter-arrival times of the different message IDs composing the dataset is inspected to determine if the probability distribution of the inter-arrival times follows a particular trend. As a representative example, the distribution of inter-arrival times for the ID $1B5$ is depicted in Figure 5.2, where the *x-axis* represents the values of the inter-arrival time (expressed in milliseconds) and the *y-axis* shows the percentage of messages with the same inter-arrival. From the analysis of Figure 5.2 it is possible to observe that the distribution of inter-arrival times of the inspected message ID follows a normal distribution centered at 20 milliseconds. All the other 49 IDs composing our dataset exhibit similar distributions, although the mean value may vary depending on the message ID.

The metric inspected for the detection of anomalies on the CAN bus uses the *cycle time* of each message ID of the vehicle for its detection purposes. The cycle time of each message ID is computed as the mean value of the inter-arrival time between consecutive messages of the same ID according to Equation 5.1, where ct^{ID} is the cycle time evaluated for each different message ID, $t_{(i-1)}$ and t_i are the timestamps associated to the $(i-1)_{th}$, and i_{th} messages of that ID respectively, and N is the number of messages with the inspected ID.

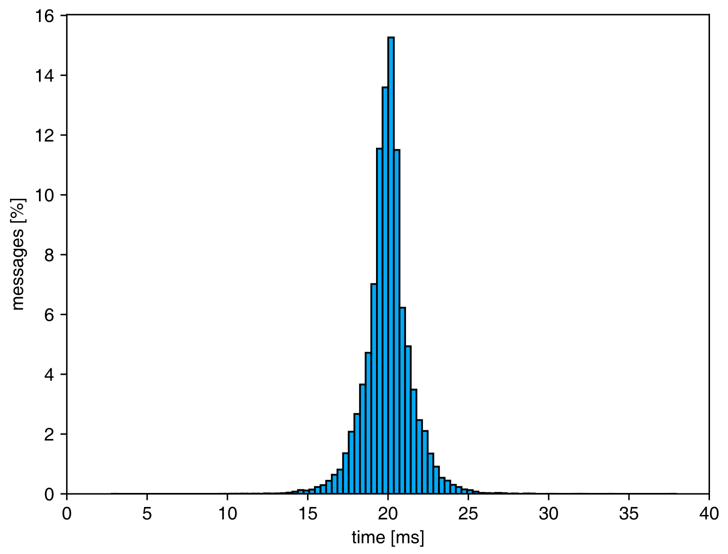


Figure 5.2: Probability distribution of Inter-arrival times of messages with ID $1B5$

$$ct^{ID} = \frac{\sum_{i=1}^{N}(t_i - t_{(i-1)})}{N} \tag{5.1}$$

The cycle time is then rounded to the closest integer millisecond value.

5.2.2 Comparison with the state-of-the-art

The state-of-the-art for the anomaly detection on CAN (described in Section 3.3) is composed by algorithms that inspect the different features available in CAN, one being the inspection based on timing/frequency analysis. Compared to the state-of-the-art, the algorithm proposed in this Section is designed to distinguish between periodical and non-periodical messages. Moreover, the attack scenario against which the algorithm is tested is novel and never considered in current state-of-the-art.

5.2.3 Message timing detection algorithm

The detection algorithm uses the evaluated cycle time for the definition of the valid *waiting time* that can normally occur between two consecutive messages of the same ID. The valid waiting time for an ID is defined as $ct^{ID} \cdot k^{ID}$, where ct^{ID} is the evaluated cycle time and k^{ID} is a tuning parameter of the algorithm that is set during the validation process for leverage imperfect timings and delays during normal CAN communications. The final goal of the validation process is to tune the detection algorithm to achieve zero false positives. For the missing message algorithm, a message is considered missing if $time_{CAN} - t^{ID} > ct^{ID} \cdot k^{ID}$, where $time_{CAN}$ is the current time of the network, t^{ID} is the timestamp of the last message having the inspected ID, and $ct^{ID} \cdot k^{ID}$ is valid waiting time computed for that ID. The validation phase is described as follows. At first the value of k^{ID} is initially set to 1, and the detection algorithm is executed on the clean traffic traces. Since these traces do not contain any attack, the detection algorithm should not generate any alert. However, real CAN frames might incur in small delays, due to clock drifts in the ECUs or contention on the access of the CAN bus. Hence if the valid waiting time is equal to the average cycle time the detection algorithm will most likely generate at least a few false positives. To avoid this issue we repeat the validation process and increase the value of k^{ID} by 1 at each iteration. This process terminates when the value of the parameter k^{ID} is enough to prevent the generation of false positives on the clean traffic traces. As an example, results of the validation process for the ID $1B5$ are shown in Figure 5.3. On the *y-axis* the different values of k^{1B5} are depicted, while the *x-axis* represents the number of false positives. For readability reasons, the x-axis represents its values in a logarithmic scale (with a small offset to to display the value 0). As shown in Figure 5.3, the number of false positives

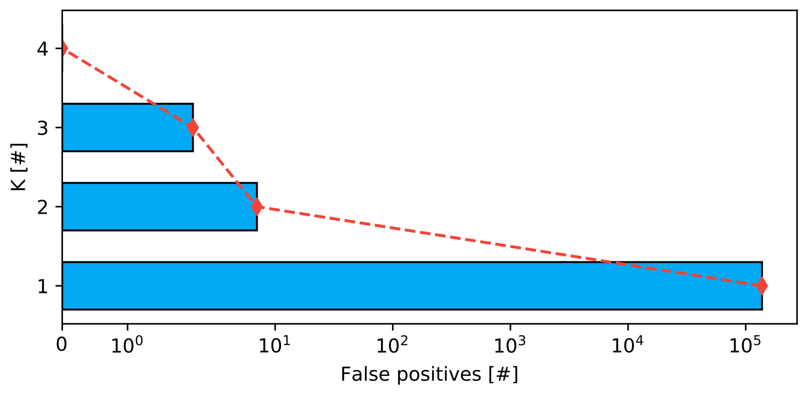


Figure 5.3: Number of false positives detected with different values k for the message ID $1B5$

for the ID $1B5$ drops by 4 orders of magnitude from $k = 1$ to $k = 2$, while no false positives are detected with a minimum value of $k = 4$. The pseudo-code describing the detection phase of the algorithm in shown in Algorithm 1.

Algorithm 1 Missing message detection algorithm

```
1: $loadReferences(timingModel)$
2: for $msg$ in $CAN_{stream}$ do
3:     $time_{CAN} \leftarrow msg.timestamp$
4:     for $msgID$ in $timingModel$ do
5:         $timeDiff \leftarrow msgID.timestamp - time_{CAN}$
6:         $\mu \leftarrow msgID.ct$
7:         $\kappa \leftarrow msgID.k$
8:         $validWT \leftarrow \mu \times \kappa$
9:         if $timeDiff > validWT$ then
10:            raise $anomaly$
```

For the tests proposed in this thesis, the missing message algorithm is implemented with Python 3 and tested on a server equipped with an Intel® Core™ i7–7700HQ CPU @3.8 GHz and with 16 GB of RAM running Fedora 27 $x64$.

5.2.4 Detection performance of the Missing Message algorithm

Since the missing message detection algorithm is designed for the detection of messages missing from the inspected CAN bus, as already described in Section 5.2.1 the detection performance of the algorithm are evaluated only against the ECU inhibition and shutdown attack scenarios described in 2.3.

The algorithm is based on the definition of a *valid waiting time* which depends on the evaluated cycle time ct^{ID} and its parameter k^{ID}. Since a message is flagged as missing after the valid waiting time expired, an initial *detection delay* is introduced by the algorithm logic. The consequences of the detection delay in the detection phase need to consider both *best* and *worst* case scenarios for the algorithm. Best and worst case scenarios are related to the precise point in time at which the attack on the targeted ECU is carried out, thus determining the instant in which the targeted ECU is sent into the *bus-off* state. The best case scenario happens when the attacker manages to activate the bus-off state on the target ECU just before a message is sent. In this case it is possible to assume that the time elapsed between the last valid message and the attack equals $ct^{ID} - \epsilon$, where ϵ is a very small amount of time. The *reaction time* of the proposed detection algorithm is defined as the amount of time that passes between the attack and its detection. The reaction time of the algorithm in the best case is equal to the waiting time $(ct^{ID} * k^{ID})$ - $(ct^{ID} - \epsilon)$, which equals to $ct^{ID} * (k^{ID} - 1) + \epsilon$. In the worst case scenario the attacker is able to send the target ECU to the bus-off state just after a message is sent. In this scenario it is possible to assume that the time between the last valid message and the attack equals to ϵ, thus the reaction time of the algorithm is equal to $ct^{ID} * k^{ID} - \epsilon$. It is clear that the reaction time is lower for messages with low cycle times (and high frequency) and for small values of k^{ID}, meaning that attacks on IDs that exhibit stable cycle times can be detected faster with respect to attacks on IDs with low frequency and high variability. As an example, considering the message with ID $1B5$, which has a cycle time of 20ms and a value of k^{1B5} equal to 4. Hence the reaction time of the proposed detection algorithm against attacks on ID $1B5$ is evaluated to be ranging from $60 + \epsilon$ milliseconds to $80 - \epsilon$ milliseconds

The best-case and worst-case theoretical detection delay are compared with the evaluated detection delays achieved by the algorithm against the ECU shutdown and ECU inhibition attacks simulated in the infected dataset.

ECU Shutdown detection performances

The analysis of the detection delays of the algorithm against the ECU shutdown attack is depicted in Figure 5.4. The *x-axis* of Figure 5.4 represents the different message IDs on which the attack is simulated, while the *y-axis* represents the detection delay expressed in milliseconds. Results in Figure 5.4 are depicted using box-plots to highlight the different results achieved throughout different attack simulations, while the dashed lines are used to represents best (green, bottom) and worst (red, top) theoretical performances.

Figure 5.4 shows that the missing message algorithm is able to correctly identify missing messages from the infected dataset within the best and worst theoretical limits. Moreover, it is noticeable that the worst case value never exceeds $120ms$, and the difference between the worst and best case is below $30ms$.

ECU Inhibition detection performances

The analysis of the detection delays of the algorithm against the ECU inhibition attack is depicted in Figure 5.5. The *x-axis* of Figure 5.5 represents the different message IDs on which the attack is simulated, while the *y-axis* represents the detection delay time expressed in milliseconds. As for the results depicted in Figure 5.4, the dashed lines represents the best and worst theoretical scenarios. In case of the inhibition attack, the detection delays of the algorithm always matches the worst case theoretical scenario, which is equal to $ct^{ID} * k^{ID}$ for each message ID.

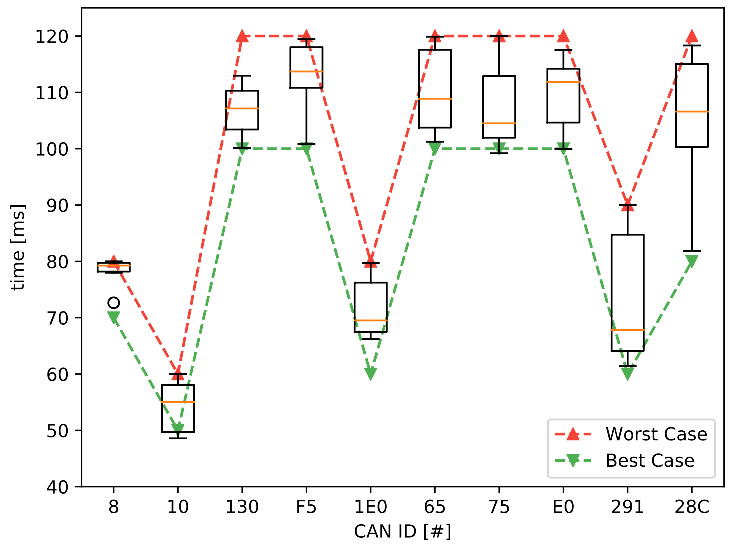


Figure 5.4: Comparison of the theoretical delays with the performances achieved by the algorithm against the ECU shutdown attack

5.3 Detection algorithm based on sequences of message IDs

5.3.1 Fundamentals

Another field inspected for the design of an anomaly detector algorithm for CAN is based on the analysis of the message IDs. In particular, the message IDs of CAN data frames are analyzed using detection methods based on the N-gram analysis. N-grams have been proposed at first for natural language processing [9,82], and have later became a methodology for the deployment of Intrusion Detection Systems [1,39,47,50,72,75,93,94,96]. The n-gram-based detection method requires the definition of the inspected alphabet A and of its different elements k. As an example, in the network packet inspection the single element k is represented by the byte-strings composing the payload of the network packets, while the alphabet A is composed by all the different byte-strings observed in the training dataset. The n-grams are extracted from the different elements composing the alphabet A from the temporal sequence of the elements k, using a sliding window of size n. The extracted n-grams are used for the definition of the normal model deployed for anomaly detection. In the automotive scenario inspected for this detection method,

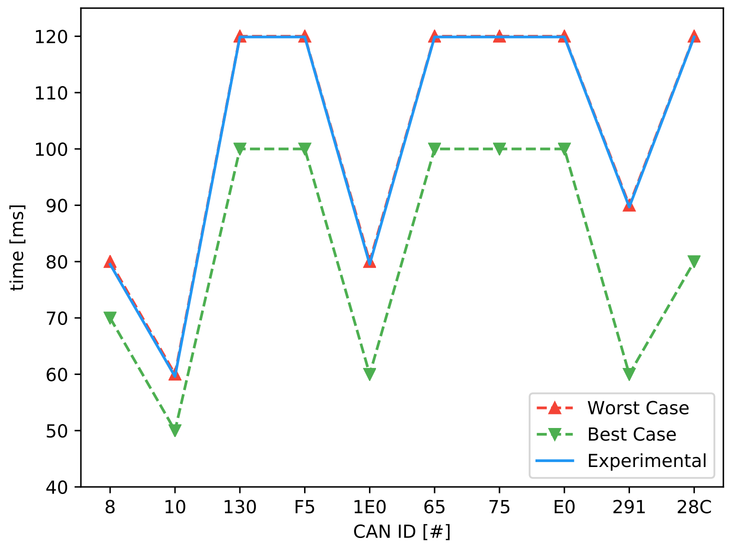


Figure 5.5: Comparison of the theoretical delays with the performances achieved by the algorithm against the ECU Inhibition attack

the alphabet A is composed by the unique values of the message IDs found in the CAN bus of the tested vehicle. Since the length of the sliding windows is a configurable parameter, for the analysis proposed in this Section different anomaly detection models are created by changing the value of the parameter n.

5.3.2 Comparison with the state-of-the-art

The state-of-the-art for the anomaly detection on CAN (described in Section 3.3) is composed by algorithms that inspect the different features available in CAN, one being the inspection of the fields composing the CAN data frame. Compared to the state-of-the-art, the algorithm proposed in this Section is designed to analyze not only the validity of the ID values of CAN data frames, but also to create a detection model based on the analysis of the n-grams composed by the ID values, thus detecting anomalous messages in the sequence of the message IDs. Moreover, previous work never inspected the constraint of the microcontrollers for the implementation of their algorithms. The algorithm proposed in this Section is designed considering this key aspect, allowing different configuration according to the specifications of the microcontrollers on which the algorithm is deployed.

5.3.3 Message sequence detection algorithm

In the definitions of the n-grams used for anomaly detection, different values of the parameter n are inspected to provide an accurate analysis of the detection performance of the n-gram based detection algorithm. However, since the value of the parameter n does not affect the definition of the algorithm, a generic description of the phases composing the detection algorithm is provided. The detection algorithm based on n-gram analysis is composed by two different phases: the *training* phase and the *detection* phase. The n-grams composing the detection model are extracted from the dataset in the *training* phase. It is necessary to remark that the choice to use the whole dataset for training purposes allows to achieve *zero-false positives* in the validation process of the algorithm, thus preventing other configuration parameters to influence the detection performance of the algorithm. In the *detection* phase the detection models are used for the detecting anomalous n-grams evaluated over the monitored CAN segment.

Training phase

In the training phase the model used for the detection phase is created by using the CAN traffic traces gathered from the test vehicle. The n-grams are extracted from the traces composing the clean dataset. The value of the parameter n used for the n-gram extraction ranges from 1 to 10. The algorithm used for the model creation is described as follows: at first, the n-grams are extracted from each different traffic trace. Each n-gram is composed by n consequent values of the message ID extracted from the valid traces in a sliding window fashion. The list of all the n-grams extracted from the clean dataset composes the detection model. It is necessary to remark that different detection models are created, one for each of the values of the parameter n. Moreover, since the frequency of the n-grams is not inspected in the detection phase, only unique n-grams are found in the model, thus without any duplicate. The pseudo-code for the training phase of the algorithm is shown in Algorithm 2.

Detection phase

In the detection phase the models created in the previous phase are used for the detection of anomalies in the CAN bus. Since the configuration parameter n allows the definition of multiple detection models, it is necessary to use the same value of the parameter n used in the model creation in the detection phase, thus allocating a fixed window of size n that is used for the definition of n-grams on the CAN traffic. This window is filled with the last n IDs found

Algorithm 2 Training phase of the N-Gram based detection algorithm

```
1: $model \leftarrow empty\_list()$
2: for $trace$ in $dataset$ do
3:     $ixe \leftarrow N$
4:     while $ixe \neq trace.length$ do
5:         $sequence \leftarrow trace[ixe - N : ixe].message\_ID$
6:         if $sequence \not\subset model$ then
7:             $model.append()$
8:         $ixe \leftarrow ixe + 1$
9: $save(model)$
```

in the CAN message stream. When the window is fully populated, the current n-gram is compared with the detection model. If the n-gram created over the CAN traffic stream is not found in the detection model, than an anomaly is raised. The algorithm than inspects the next n-gram by proceeding in a sliding-window fashion, thus requiring only one more message ID for each iteration. The pseudo-code for the detection phase of the algorithm is shown in Algorithm 3.

Algorithm 3 Detection phase of the N-Gram based detection algorithm

```
1: $n, ref \leftarrow load(model)$
2: $seq \leftarrow list()$
3: for $msg\_ID$ in $ID\_stream$ do
4:     if $seq.length < n - 1$ then
5:         $seq.append(msg\_ID)$
6:     else
7:         $seq.append(msg\_ID)$
8:         if not $ValidNGram(seq, ref)$ then
9:             raise $anomaly$
10:        $seq.pop()$
11: function ValidNGram($sequence, model$)
12:     if $sequence \in model$ then
13:         return True
14:     return False
```

For the tests proposed in this thesis, the message sequence algorithm is implemented with Python 3 and tested on a server equipped with an Intel® Core™ i7–7700HQ CPU @3.8 GHz and with 16 GB of RAM running Fedora 27 $x64$.

5.3.4 Detection performance of the ID sequence algorithm

The n-gram based detection algorithm proposed in this Section inspects the sequences of the message IDs on the network. Since each attack scenario described in 2.3 implies a direct modification of the sequences of the messages on the network, the detection capabilities of the proposed algorithm are evaluated against all the attacks described in 5.1.2.

The detection performance of the algorithm are evaluated by means of the $\mathcal{F}-measure$, which is the harmonic mean of *precision* and the *recall*. The precision is the number of correct anomalies detected by the algorithm over the total number of anomalies in the dataset, while the recall is the number of correct anomalies over the total number of anomalies detected by the algorithm. The formulas used for the evaluation of precision, recall and $\mathcal{F}$-measure are shown in Equation 5.2, Equation 5.3, and Equation 5.4, respectively. The symbols T_p, F_p, and F_n used in the formulas denotes the *true positives*, *false positives*, and *false negatives*, respectively.

$$precision = \frac{T_p}{T_p + F_p} \tag{5.2}$$

$$recall = \frac{T_p}{T_p + F_n} \tag{5.3}$$

$$\mathcal{F} = 2 * \frac{precision * recall}{precision + recall} \tag{5.4}$$

The $\mathcal{F}-$measure index ranges from 0 to 1, where 0 denotes extremely poor detection capabilities and 1 denotes perfect detection capabilities.

Replay attack detection

The detection performance of the algorithm against the replay attack scenario (described in Section 5.1.2) are presented in this section. For the evaluation of the detection performance of the algorithm, the values of the parameter n used in the definition of the normal model ranges from 1 to 10. The results of the algorithm against the *single ID* replay attack scenario are depicted in Figure 5.6. The plot in Figure 5.6 compares the $\mathcal{F}-$score (*y-axis*) achieved by the detection algorithm by deploying detection models created with different values of the parameter n (*x-axis*).

Figure 5.6 depicts the detection results of the algorithm against the single ID replay attack scenario. The different messages used for the definition of each attack scenario are depicted in different colors, while box-plots are used

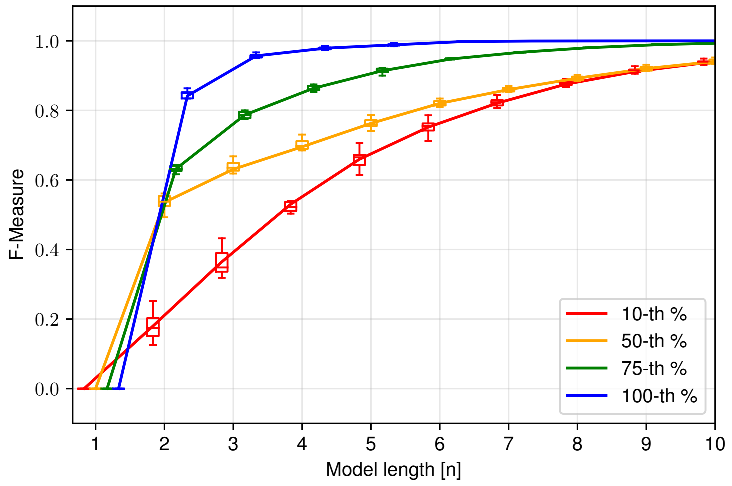


Figure 5.6: Detection performance of the N-gram based detection algorithm against the *single ID* replay attack

to highlight variability of the results over different tests. The box-plots are grouped according to the value of the parameter n. From the comparison of the detection results achieved by the algorithm in the different attack scenarios it is possible to notice that the detection performance achieved by the algorithm depend from the distribution of the ID used for the attack simulation: by injecting messages representing higher percentiles (thus less frequent messages) it is possible to achieve higher detection performance compared to the injection of messages representing lower percentiles (this more frequent messages), regardless from the value of the parameter n. However, in case of injection of the ID belonging to the 10_{th} percentile (red plot) it is possible to notice that the detection results stabilizes near $\mathcal{F}-$measure $= 0.9$ by using a value of $n \geqslant 5$.

In case of the *valid sequence* replay attack scenario, the median results achieved by the detection algorithm against the simulated attacks are depicted in Figure 5.7.

The results in Figure 5.7 depict the median detection results achieved by detection algorithm against the *valid sequence* replay attack. The length of the attack is depicted on the *y-axis*, while the *x-axis* represents the value of the parameter n used for the definition of the detection model. The median $\mathcal{F}-$measure values are depicted with different colors, each one representing a different range. For readability purposes the ranges are defined using non linear intervals to increase the accuracy of the plots for higher values

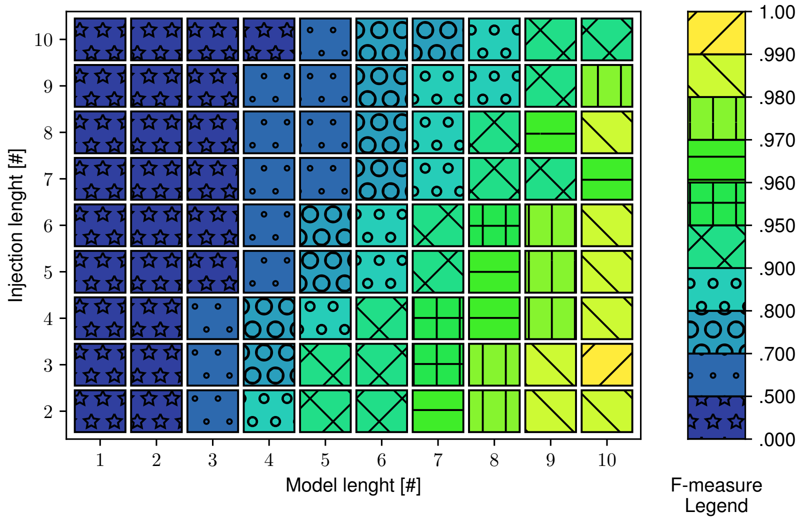


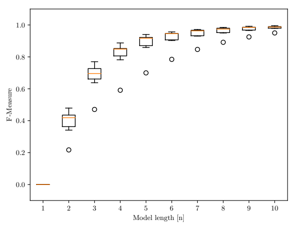
Figure 5.7: Detection performance of the N-gram based detection algorithm against the *valid sequence* replay attack

of $\mathcal{F}-$measure. The range reference is depicted on the right of the plot. The analysis of the results depicted in Figure 5.7 shows that by increasing the value of the parameter n the algorithm detection performance increases, while by increasing the length of the replay attack the overall detection performance decrease. The detailed detection performances of the algorithm are depicted in Figure 5.8, in which from Figure 5.8a to Figure 5.8i the results of the algorithm against the injection of n-grams created with values of the parameter n ranging 2 to 10 are shown. Detection results achieved with the model created with $n = 1$ are not depicted in Figure 5.8 since the algorithm is not able to detect any anomaly as already shown in Figure 5.7.

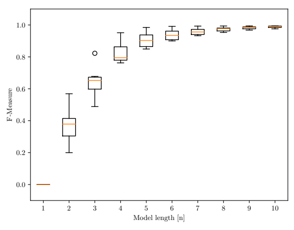
The results depicted in Figure 5.8 shown that the detection performance follows the same trend already described from the analysis of the results depicted in Figure 5.7.

The results achieved by the algorithm against the *Invalid sequence* replay attack scenario are depicted in Figure 5.9.

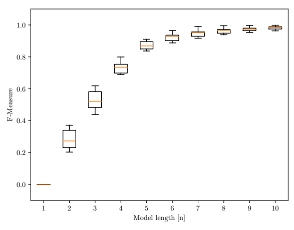
Figure 5.9 depicts the median detection results achieved by the algorithm against the *invalid sequence* replay attack. As already described for the results depicted in Figure 5.7, the *y-axis* presents the length of the injected sequence, while the *x-axis* describes the different values of the parameter n. The median $\mathcal{F}-$measure values are depicted with different colors, each one representing a different range. For readability purposes the ranges are defined using non linear intervals to increase the accuracy of the plots for



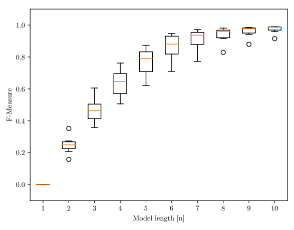
(a) Injection of n-grams with $n = 2$



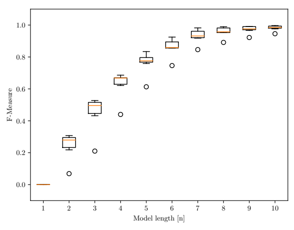
(b) Injection of n-grams with $n = 3$



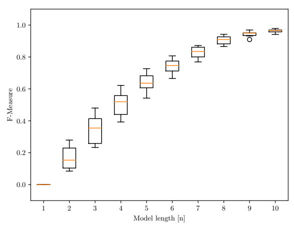
(c) Injection of n-grams with $n = 4$



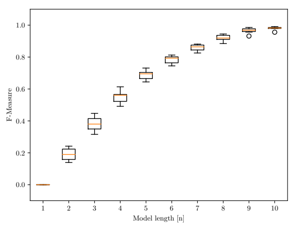
(d) Injection of n-grams with $n = 5$



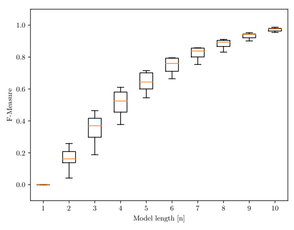
(e) Injection of n-grams with $n = 6$



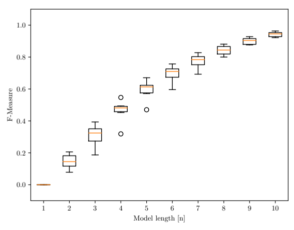
(f) Injection of n-grams with $n = 7$



(g) Injection of n-grams with $n = 8$



(h) Injection of n-grams with $n = 9$



(i) Injection of n-grams with $n = 10$

Figure 5.8: F-Measures evaluated over the *valid sequence* replay attack simulated with different values of n

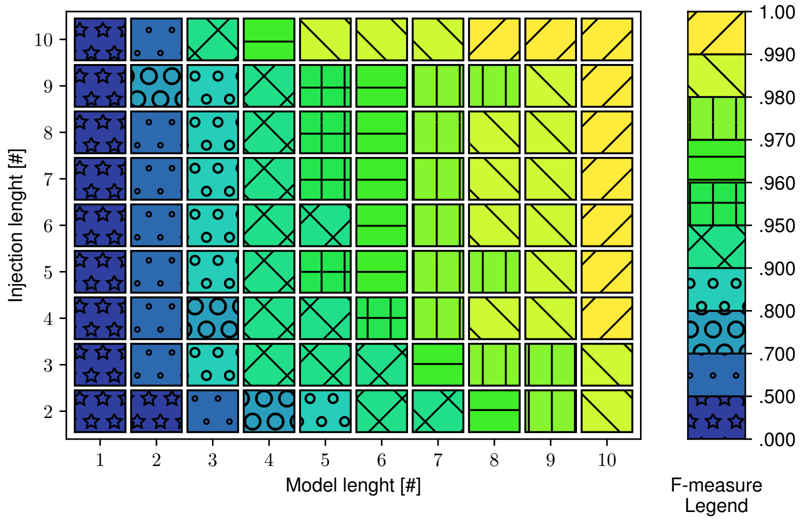
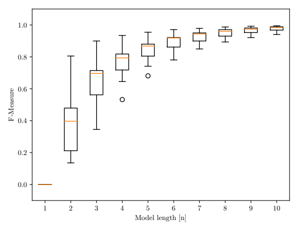


Figure 5.9: Detection performance of the N-gram based detection algorithm against the *invalid sequence* replay attack

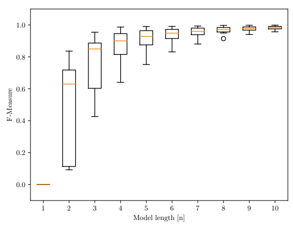
higher values of $\mathcal{F}-$measure. The range reference is depicted on the right of the plot. The detection results depicted in Figure 5.9 show that by increasing either the length of the injected sequence or the value of the parameter n the detection performance of the algorithm increases, achieving $\mathcal{F}-$measure near 1.0 by deploying a model created with $n \geqslant 6$ regardless from the length of the injected sequence. The detailed detection performance depicted in Figure 5.10 confirm the analysis result presented in Figure 5.9, in which by increasing the value of the parameter n it is possible to achieve higher detection performance. As for the previous attack scenario, detection results achieved with the model created with $n = 1$ are not depicted in Figure 5.10 since the algorithm is not able to detect any anomaly as already presented in Figure 5.9.

Fuzzing attack

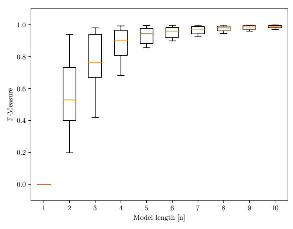
The results of the detection algorithm tested against the *random ID* fuzzing attack demonstrated that the algorithm is always able to achieve perfect detection of the inspect attack scenario, achieving $\mathcal{F}-$ measures of 1.0 regardless from the value of n. Since the random ID fuzzing attack (as described in Section 2.3.2) is simulated by injecting messages with ID values not found in the datasets, it is possible to detect any random ID fuzzing attack with maximum precision and recall. The results of the detection algorithm against the *random payload* fuzzing attack are the same already shown in Figure 5.6,



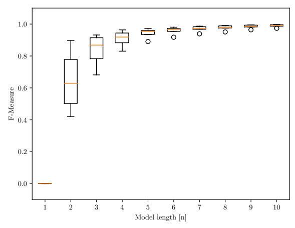
(a) Injection of n-grams with $n = 2$



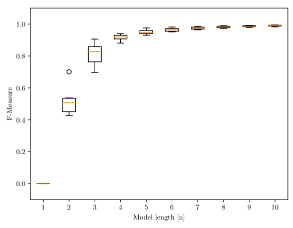
(b) Injection of n-grams with $n = 3$



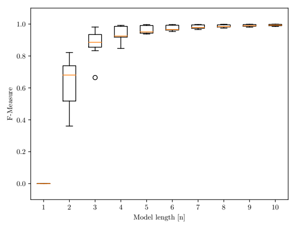
(c) Injection of n-grams with $n = 4$



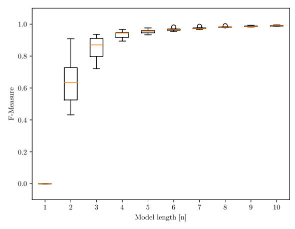
(d) Injection of n-grams with $n = 5$



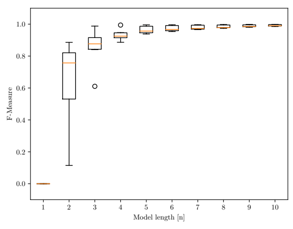
(e) Injection of n-grams with $n = 6$



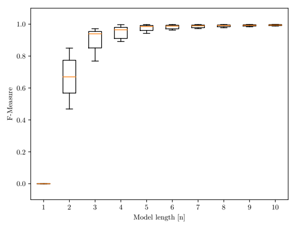
(f) Injection of n-grams with $n = 7$



(g) Injection of n-grams with $n = 8$



(h) Injection of n-grams with $n = 9$



(i) Injection of n-grams with $n = 10$

Figure 5.10: F-Measures evaluated over the *invalid sequence* replay attack simulated with different values of n

since the message IDs used for the generation of both attack scenarios are the same.

Denial of Service attack

For the analysis of the detection performance of the algorithm against the Denial-of-Service attacks, only the scenario of the *lowest ID* denial-of-service is inspected. In fact, since the message with ID value 0 is not found in the dataset, the detection performance achieved by the algorithm against the *zero ID* denial-of-service attack are the same already presented in the *fuzzing attack scenario.*

The detection results of the algorithm against the *lowest ID* denial-of-service attack are presented in Table 5.1. The rows of Table 5.1 represent the value of the configuration parameter n, while the values of the columns represent the different trace of the dataset on which the attacks are simulated. The results are depicted by means of $\mathcal{F}-$measure evaluated by the detection algorithm.

	test #						
n	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.995	0.995	0.995	0.995	0.994	0.994	0.992
3	0.998	0.999	0.998	0.998	0.999	0.998	0.998
4	0.999	0.999	0.999	0.999	0.999	0.999	0.999
5	0.999	0.999	0.999	0.999	0.999	0.999	1.0
6	1.0	1.0	1.0	1.0	1.0	1.0	1.0
7	1.0	1.0	1.0	1.0	1.0	1.0	1.0
8	1.0	1.0	1.0	1.0	1.0	1.0	1.0
9	1.0	1.0	1.0	1.0	1.0	1.0	1.0
10	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Table 5.1: F-Measures evaluated with the N-gram based detection algorithm against the lowest ID denial of service scenario

The analysis of the results presented in Table 5.1 show that the algorithm is able to achieve near-perfect detection of the denial-of-service attack by using a model created with a minimum value of the parameter $n = 2$, while perfect results are achieved with a value of $n \geqslant 6$. Furthermore, the ID used for the generation of this attack scenario is the message with ID value equals to 1, which is also the ID representing the 100_{th} percentile in the distribution of probabilities depicted in Figure 5.1.

ECU inhibition attack

The results of the detection algorithm against the *ECU inhibition* attack are depicted in Figure 5.11. Each plot of Figure 5.11 represents the detection percentages achieved by the algorithm, since it is not possible to evaluate the $\mathcal{F}$-measure on the inspected attack scenario. The detection percentage is evaluated as the overall number of anomalies detected over the total number of removed messages.

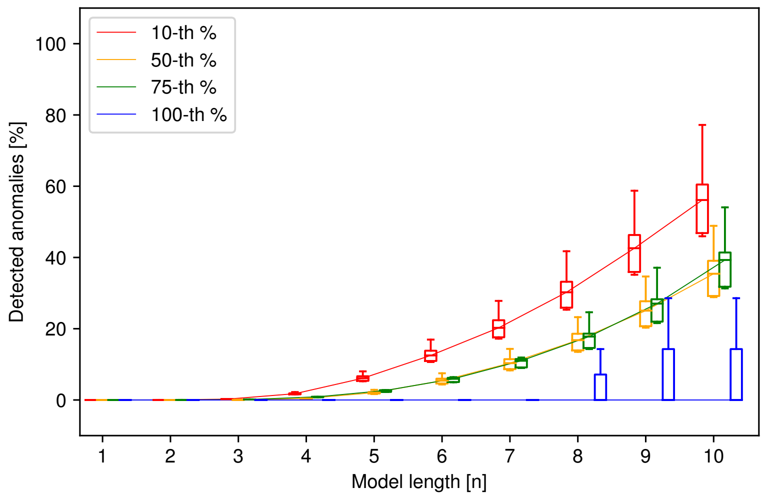


Figure 5.11: Detection percentages of the N-gram based detection algorithm against the *ECU inhibition* attack

The results depicted in Figure 5.11 show that the n-gram based detection algorithm detection performances are affected by the value of the parameter n. The algorithm is able to detect more anomalies by using models created with higher values of n, regardless of the message used for the attack. It is necessary to remark however, that the algorithm struggles to detect more than 80% of the anomalies in the best scenario.

5.4 Detection algorithm based on Hamming distance

5.4.1 Fundamentals

The successful attacks to the automotive networks have been conducted by modifying the values of the signals encoded in the data field (as already described in Section 2.3), thus it is necessary to inspect the bits composing the data field of the messages as a metric for anomaly detection. For this purpose, the metric used for the definition of the normal model of the vehicle through inspection of the binary representation of the message data fields evaluates the *hamming distance* on the sequences of data fields sharing the same ID. The Hamming distance [35] is used to measures the minimum number of substitutions required to change one string into one another, and it is widely used in several disciplines including information theory, coding theory and cryptography. The generic formula for the evaluation of the Hamming distance between two words of equal length k can be found in Equation 1:

$$\mathcal{H}_d(x,\ y) = \sum_{i=1}^{k} |x_i - y_i| \tag{1}$$

The evaluated $x_i - y_i$ shown in Equation 1 is equal to 0 when x_i equals y_i, 1 otherwise. In this particular scenario, the Hamming distance is evaluated over two binary strings of the same length. This scenario has already been inspected by many researcher and it is known as the Hamming cube, a variation of the classic Hamming distance applied on strings which are comparable to vertices of an hypercube graph. A generic binary string of length n is defined as vector in IR^n, in which each symbol in the string is considered as a coordinate of the real plan, thus each string is evaluated as a particular vertex of the $n-$dimensional hypercube. In this scenario, the Hamming distance of two binary strings is equivalent to the Manhattan distance between the vertices of the generated hyper-line. For detection purposes, this metric evaluates the Hamming distance between two binary string, reducing the hypercube to a single hyper-vector. If the length of the payload for a particular message ID equals 64 bits, the resulting hyperspace containing all the possible vertices would have a maximum size of 264. In this scenario, the Hamming distance between two binary strings of length 64 is evaluated as shown in Equation 2, where p_t is a generic payload at time t and p_t^i is the i_{th} bit of that payload.

$$\mathcal{H}_d(p_t, p_{t+1}) = \sum_{i=1}^{64} p_t^i \otimes p_{t+1}^i \tag{2}$$

5.4.2 Comparison with the state-of-the-art

The state-of-the-art for the anomaly detection on CAN (described in Section 3.3) is composed by algorithms that inspect the different features available in CAN, one being the analysis of statistical indexes evaluated over the CAN traffic. Compared to the state-of-the-art, the algorithm proposed in this

Section is designed detect anomalies without requiring the full knowledge of the syntax and semantics of the CAN data field. Moreover, compared to the other works based on statistical analysis, this algorithm is able to detect stealth attacks that involve the injection of very few CAN messages.

5.4.3 Hamming distance detection algorithm

The algorithm designed to use the Hamming distance as its detection feature is composed by two different phases: one phase for the creation and the validation of the normal model, and another phase for the online detection of anomalies. In the first phase the algorithm inspects the traces of the dataset to compute the minimum and maximum Hamming distance of sequence of data fields extracted from each different message ID.

The model creation phase uses 20% of the dataset, while the remaining 80% is used for validation purposes. Since the detection algorithm inspects the difference in the Hamming distance between consecutive data fields of CAN messages with the same ID, a preliminary phase during which messages with the same CAN ID are grouped together is performed for each trace composing the dataset. These sub-traces are used for the evaluation of the Hamming distance between consecutive data fields, using the formula shown in Formula 2. The minimum and maximum distances are saved for each message ID, composing the non-validated model. In the validation phase the minimum and maximum values are used for the detection of false positives. The algorithm evaluates the Hamming distance between consecutive messages with the same ID and, in case the evaluated distance falls outside the range defined by $[min, max]$then an anomaly is raised. During the validation phase of the algorithm 0 false positives are detected, thus implying that the minimum and maximum Hamming distance evaluated during the training phase represent a steady feature that can be used to identify message injections. The pseudo-code for this phase is represented in Algorithm 4.

In the online detection phase the algorithm in is able to detect anomalies by evaluating the Hamming distance of two consecutive payloads of the same message ID. When the evaluated distance is outside the Hamming range associated to that particular ID, an anomaly is raised. The pseudo-code for the online detection phase of the Hamming distance algorithm is shown in Algorithm 5

For the tests proposed in this thesis, the Hamming-based algorithm is implemented with Python 3 and tested on a server equipped with an Intel® Core™ i7–7700HQ CPU @3.8 GHz and with 16 GB of RAM running Fedora 27 $x64$.

Algorithm 4 Model creation and validation for the Hamming distance detection algorithm

```
function $HammingDistance(df_a, df_b)$
    $dist \leftarrow 0$
    for $ix$ in $range(0, len(df_a))$ do
        if $df_a[ix] \neq df_b[ix]$ then
            $dist \leftarrow dist + 1$
    return $dist$
function CREATEHAMMINGREFERENCE($data_{seq}$)
    $prevDF \leftarrow data_{seq}[0]$
    $minH \leftarrow 64$
    $maxH \leftarrow 0$
    for $currDF$ in $data_{seq}[1:]$ do
        $currDist \leftarrow HammingDistance(prevDF, currDF)$
        if $currDist > maxH$ then
            $maxH \leftarrow currDist$
        else if $currDist < minH$ then
            $minH \leftarrow currDist$
    return $minH, maxH$
function VALIDATEHAMMINGREFERENCE($data_{seq}$)
    $x \leftarrow len(data_{seq}) * 0.2$
    $minH, maxH \leftarrow CreateHammingReference(data_{seq}[: x])$
    $newMinH \leftarrow minH$
    $newMaxH \leftarrow maxH$
    $prevDF \leftarrow data_{seq}[len]$
    for $currDF$ in $data_{seq}[len + 1 :]$ do
        $currDist \leftarrow HammingDistance(prevDF, currDF)$
        if $minH \leqslant currDist \leqslant maxH$ then
            $fp \leftarrow fp + 1$
            if $currDist > maxH$ then
                $maxH \leftarrow currDist$
            else if $currDist < minH$ then
                $minH \leftarrow currDist$
```

Algorithm 5 Online detection for the Hamming distance algorithm

```
1: $model \leftarrow load(HammingReferences)$
2: $prevDF \leftarrow emptyHashTable$
3: for $mIDinmodel.ID$ do
4:     $prevDF[mID] \leftarrow NULL$
5: for $msg$ in $CAN\_stream$ do
6:     $mID \leftarrow msg.CAN_{ID}$
7:     $mData \leftarrow msg.data$
8:     if $prevDF[mID]notNULL$ then
9:         $currDist \leftarrow HammingDistance(prevDF[mID], currDF)$
10:        $minH, maxH \leftarrow model(mID)$
11:        if $not(minH \leqslant currDist \leqslant maxH)$ then
12:            raise $anomaly$
13:    $prevDF[mID] \leftarrow mData$
```

5.4.4 Detection performance of the Hamming-based algorithm

Since the algorithm that leverages the Hamming distance to detect anomalies in the CAN traffic flows is designed to inspect the distance between between consecutive payloads of the same message ID, its detection performances are evaluated against the only two attack scenarios that modify the content of the data field of the message, which are the *fuzzing payload* attack and the *single ID replay attack*.

Preliminary analyses on the detection results highlighted that many IDs had similar detection rates for both *fuzzing payload* and *single ID* replay attacks. These behavior has been further inspected, denoting that IDs with close detection results have similar Hamming ranges. Statistical analysis of the Hamming ranges is shown in Figure 5.12, where the *y-axis* represent the Hamming range and the *x-axis* is an identifier of the 50 different IDs that have been found in the CAN traffic traces. IDs have been ordered with respect to the associated Hamming range, and IDs with an Hamming range evaluated to be equal to 0 are highlighted at the end of the plot with a light color, making them visible. It is possible to observe that IDs can be naturally classified in three main categories:

- **NoRange**: IDs for which the Hamming distance between consecutive messages is always constant, hence the minimum and maximum Hamming distances are equal and the Hamming range is 0;

- **SmallRange**: IDs for which the distance between the maximum and minimum Hamming distances (Hamming range) is always lower than a σ reference value;
- **MidRange**: IDs for which the distance between the maximum and minimum Hamming distances (Hamming range) is always higher than σ.

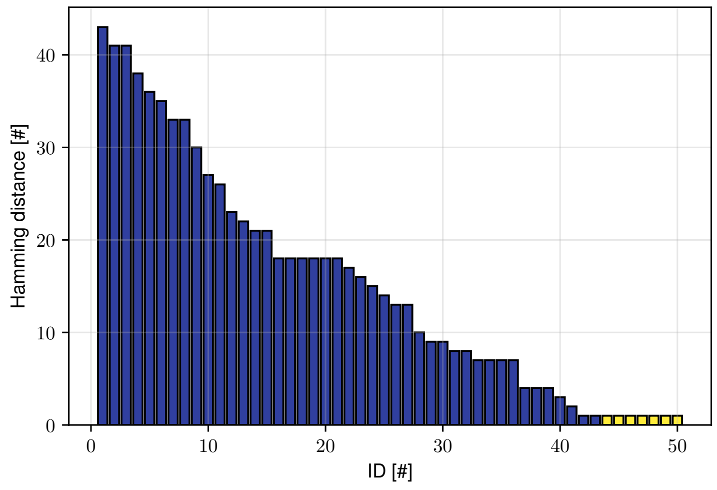


Figure 5.12: Hamming ranges evaluated for the clean dataset

The value of σ has been empirically determined to be equal to 18. This choice allows to group together IDs with very similar detection rates, and is motivated by the relatively high gap that exists between IDs 15_{th} and 16_{th}, as shown in Figure 5.12. According to this classification the *NoRange*, *SmallRange* and *MidRange* classes comprise 7, 15 and 28 message IDs, respectively.

Fuzzing attack detection

Figure 5.13 represents the detection results of the Hamming algorithm against the payload fuzzing attack described in Section 2.3. For better inspection of the detection capabilities of the algorithm, the attacks have been simulated by injecting a malicious payload each $10, 25$ and 50 normal messages. These attack frequencies allows to better inspect the detection capabilities of the algorithm against different injection of malicious messages.

The detection results of the algorithm are depicted in Figure 5.13, in which the leftmost set of bars refers to an injection every 10 messages, the middle set of bars to an injection every 25 messages, and the rightmost set of bars refers to an injection every 50 messages. Within each set of bars, the first refers to the NoRange class, the second to the SmallRange class, and the third to the MidRange class. The *x-axis* represents the attack frequency, while the *y-axis* represents the detection rate. The results are depicted by means of detection percentage, presenting for each class both true positives and the false positives detected by the algorithm with different shades of the same color.

Results in Figure 5.13 show that the proposed algorithm is able to detect the injection of attack with percentages close to 100% in case of both NoRange and SmallRange classes. In particular, the detection rate for the NoRange class is always higher than 98% and for the SmallRange class is always higher than 97%.

On the other hand, the detection rate is lower for attacks involving the IDs classified as MidRange. In this class the detection rate varies from 20% to 30% depending on the intensity of the attack. Attacks with a shorter period and a higher injection frequency present better detection results with respect to those characterized by a longer period and a lower injection frequency.

Poor detection results in case of the MidRange class are a direct consequence of the relatively high Hamming range that characterize IDs belonging to this class. MidRange class is composed by IDs with Hamming range above the selected σ, meaning that the Hamming distance between payloads of IDs belonging to this class could change significantly during the vehicle dynamic. Thus, by injecting randomly generated payloads there are higher probabilities that the injected payload is close enough in terms of Hamming distance to it neighbors, generating higher false negatives and keeping the detection rates smaller compared to the other classes.

Single ID Replay attack detection

Figure 5.14 represents the detection results of the Hamming based detection algorithm against the *single ID* replay attack scenario. As for the previous attack scenario, for better inspection of the detection capabilities of the algorithm the attacks have been simulated by injecting a single message each $10, 25$ and 50 normal messages.

Similarly to the previous attack scenario, for each class of attacks the three different detection results are shown, represented as sets of three vertical bars. Left to right, the sets of bars refer to a replay message injected every $10, 25$ and 50 messages, respectively. Within each set of bars, the first refers

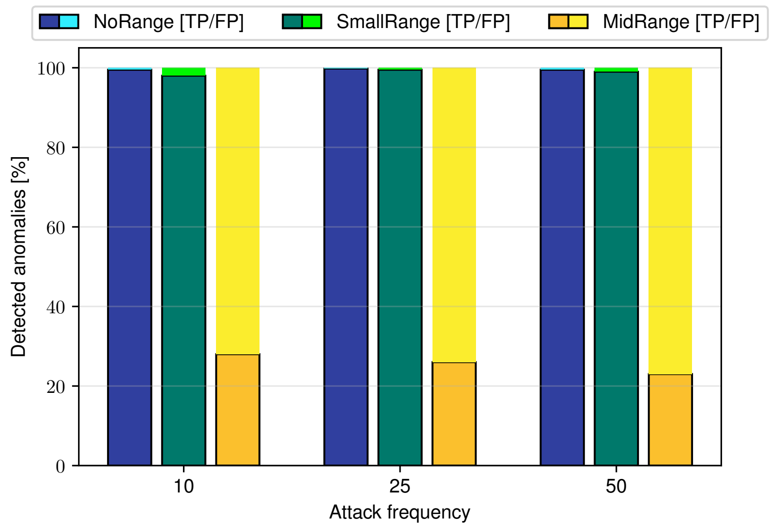


Figure 5.13: Detection results of the Hamming based algorithm against the *payload fuzzing* attack scenario

to the NoRange class, the second to the SmallRange class, and the third to the MidRange class. The *x-axis* represents the attack frequency, while the *y-axis* represents the detection rate. The results are depicted by means of detection percentage, presenting for each class both true positives and the false positives detected by the algorithm with different shades of the same color.

It is possible to observe that the proposed method is not suitable for the detection of replay attacks, achieving very poor results in case of the NoRange and SmallRange classes. In particular, the proposed method never raises an alert for replay attacks on IDs belonging to the NoRange class, while detection rates for the SmallRange class are always below 2%. This result could be explained by the very low Hamming range that characterize legit messages, that are very similar among themselves. Since the replay attack is executed by injecting a legit message out-of-place within the CAN traffic, the anomalous message will always be injected before or after a message with very similar bits, thus preventing the algorithm to detecting it.

Better results can be achieved for IDs belonging to the MidRange class, and characterized by higher Hamming ranges (and higher message variability). Detection rate for this class varies between 10% and 20%, depending on the attack frequency. While these detection rates are small, it is necessary to remark that the proposed detection method achieved a false positive rate of 0.

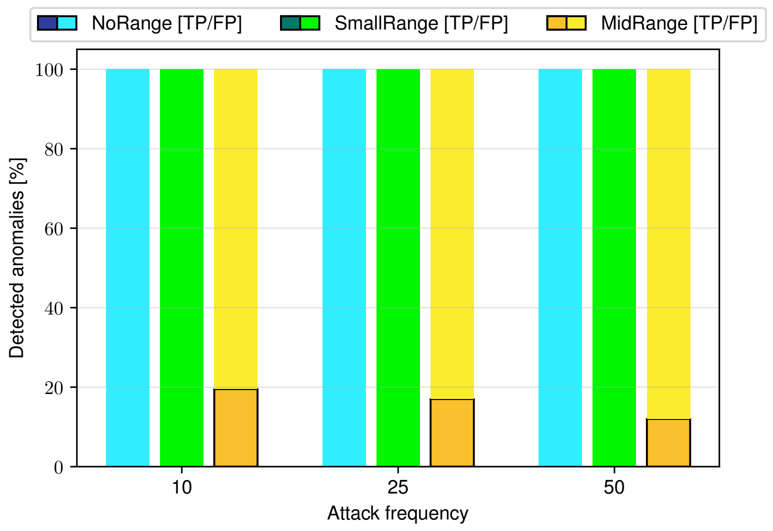


Figure 5.14: Detection results of the Hamming based algorithm against the *single ID* replay attack scenario

5.4.5 Computational costs

An accurate inspection of computational complexity and memory requirements for the proposed algorithm is given in this paragraph to prove its low computational requirements that make it applicable to low-end ECUs deployed in common vehicles.

Computational complexity: The Hamming based algorithm for live detection requires to compare the current payload value of a specific ID with the previous payload of the same message ID. The evaluation of the Hamming distance of the two payloads takes place in a single loop, that compares all the N bits composing the payload and sums the result of a bit-wise XOR operation among the bits at the same index of the two different payloads, adding one in case the two bits are different from each other, 0 otherwise. At the end of this loop, the evaluated distance is compared to the reference values for that message ID, raising an anomaly if the value is outside the valid range. Computational complexity of the final implementation of the live detection has been evaluated as $\mathcal{O}(N)$. Where N is equal to the number of bits forming the payloads, with a maximum value of 64.

Memory requirements: The proposed detector uses one indexed data structure to store the previous payload for each ID. The size of this structure is evaluated as $N_{id} \times L_{id}$, where N_{id} represents the number of unique IDs flowing on the internal network and L_{id} denotes the number of bits composing the payload for that particular ID. Maximum memory requirements could be evaluated as $N_{id} \times 64$, where 64 is the maximum allowed value for L_{id}.

From the previous analysis it is possible to evaluate that for the vehicle used in the experimental evaluation a maximum of 400 Bytes to store the previous messages are required, thus requiring 8 Bytes for each of the 50 unique message IDs flowing on the inspected CAN bus.

Common low-end ECUs are generally composed by microcontrollers with 1 computational core, having a working frequency in the orders of hundreds of Mega Hertz, and with few hundreds of Kilo Bytes of RAM. For the tested vehicle the live detector only requires up to 400 Bytes of memory, and its operations can be carried out by a common microcontroller equipped with a single core. Hence, the proposed live-detection algorithm can be implemented on common low-end ECUs.

5.5 Detection algorithm based on Entropy

5.5.1 Fundamentals

Following the definition of multiple metrics based on the inspection of the fields composing the CAN frames in the previous Sections, a metric based on the inspection of the combination of multiple fields is proposed. This detection method is designed to inspect the entropy of the data flowing on the CAN bus for the detection of anomalies. Entropy-based anomaly detection algorithms characterize the normal behavior of a set of data based on their level of statistical entropy [19]. The entropy $\mathcal{H}$ of a dataset comprising i different symbols is defined according to equation 5.5:

$$\mathcal{H} = \sum_i p\left(i\right) \log_2 \left[\frac{1}{p\left(i\right)}\right] \tag{5.5}$$

where $p\left(i\right)$ represents the probability of occurrence of the i_{th} symbol. In information theory, entropy represents the amount of information conveyed in the dataset, expressed in bits. As an example, a dataset composed by only one symbol has $\mathcal{H} = 0$ independently of its length, meaning that it conveys 0 bits of information. On the other hand, a dataset containing n independent and identically distributed symbols has $\mathcal{H} = \log_2(n)$. $\mathcal{H}$ also represents the expected amount of information conveyed by each message belonging to the dataset. The value of $\mathcal{H}$ is also used to measure the randomness of an information source. The use of entropy as a mean to describe the normal behavior of an information source relies on the following underlying assumptions:

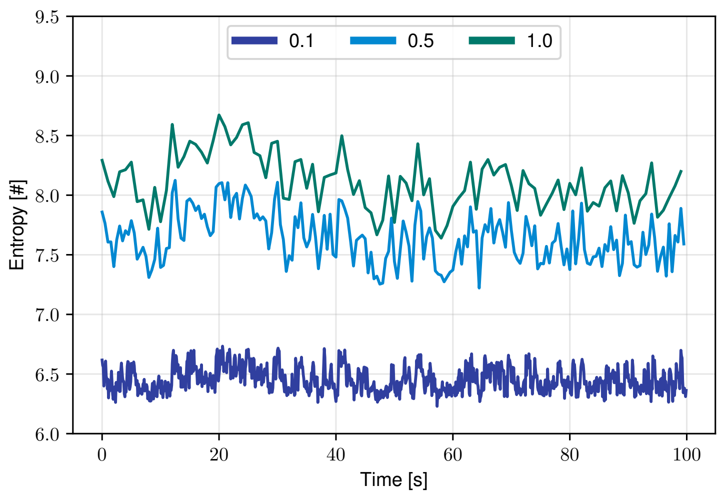


Figure 5.15: Preliminary analysis of entropy values evaluated on different time windows

- the entropy of messages generated by the information source exhibits stable statistical characteristics;
- relevant anomalies (that is: anomalies that should be detected by the algorithm) introduce significant deviations in the statistical characteristics of the entropy.

A feasibility study of this detection technique is conducted by analyzing the first 100 seconds of the first trace of the dataset. This initial sequence of messages has been further divided into non overlapping time windows of 1, 0.5 and 0.1 seconds, and used equation 5.5 to compute the entropy of the set of CAN messages included in each time window. Hence, three different time series representing the evolution of entropy (y-axis) over time (x-axis) for the three different time granularity have been generated and depicted in Figure 5.15.

Figure 5.15 depicts the evolution over time (shown on the *x-axis*) of the entropy values (shown on the *y-axis*) evaluated over different time windows (top to bottom: 1, 0.5, and 0.1 seconds). From the results of the preliminary analysis depicted in Figure 5.15 it is possible to observe that the entropy value is stable and independent from specific driving conditions (such as changes in speed, sudden brakes, road turns, activation of turning lights). As expected, entropy computed over larger time frames is higher than entropy computed over shorter time frames, that includes fewer messages.

To identify suitable criteria for anomaly detection the distribution of entropy values are also analyzed and shown in Figure 5.16, where the x-axis depicts the entropy value and the y-axis represents the number of time windows that fall within each bin.

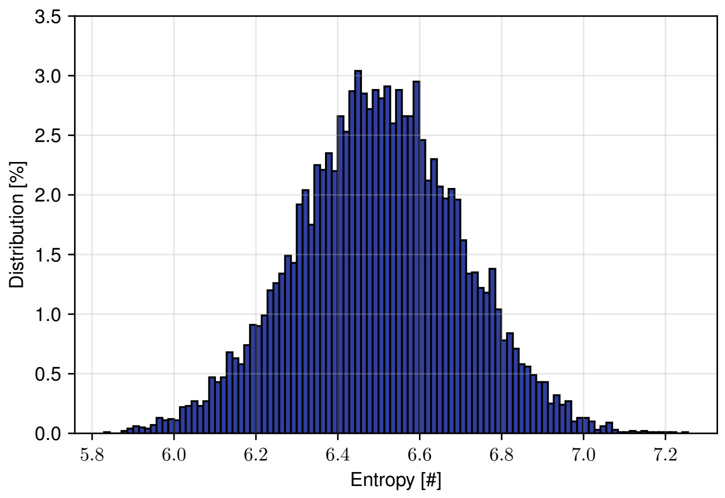


Figure 5.16: Distribution of the CAN entropy measured with a time window of 0.1 seconds.

Figure 5.16 refers to a subset of 100 seconds of CAN messages, and entropy values are computed with a time window of 0.1 seconds. Similar distributions are achieved for all time granularity and it is possible to compare them to the normal distribution.

5.5.2 Comparison with the state-of-the-art

The state-of-the-art for the anomaly detection on CAN (described in Section 3.3) is composed by algorithms that inspect the different features available in CAN, one being the analysis of statistical indexes evaluated over the CAN traffic. Compared to the state-of-the-art, the algorithm proposed in this Section expands the analysis already proposed in literature and inspects the boundaries of entropy-based anomaly detectors for the CAN bus.

5.5.3 Entropy-based detection algorithm

The algorithms based on the evaluation of Entropy values is composed by two different phases: an initial *model creation and validation* phase, that generates all the references for each ID of the car model and validates the preliminary results against other traces of the dataset; and the *live detection* phase, designed and implemented to test the detection model against real attack scenarios. Since entropy values appear to be rather stable over time and distributed according to a normal distribution as inspected in the previous Section, the anomaly detection algorithm based on the inspection of the entropy value is based on the assumption that entropy values that are too distant from the average entropy are unlikely, and should be considered as anomalies. For each of the three time granularity considered in the previous Section, two descriptive parameters are computed: the average entropy value μ_e and its standard deviation σ_e. For each time window t, the anomaly detection algorithm leverages equation 5.5 to compute $\mathcal{H}_t$, that is the entropy of all CAN messages included in the time frame t. An anomaly is raised if $\mathcal{H}_t$ is not within in the range $[\mu_e - k\sigma_e, \mu_e + k\sigma_e,]$, where k is a model parameter that defines the sensitivity of the algorithm with respect to deviations from μ_e. To tune the value of the parameter k the validation phase of the algorithm inspects a part of CAN dataset not used for the definition of the values μ_e and σ_e of the dataset with an initial value of $k = 1$. The value of k is increased by one until the algorithm raises no anomalies, thus achieving 0 false positives. The lowest value of k that generates 0 false positives is $k = 3$ for all three inspected time granularity. The pseudo-code describing the *model creation and validation* and *live detection* phases are depicted in Algorithm 6 and Algorithm 7, respectively.

For the tests proposed in this thesis, the Entropy-based algorithm is implemented with Python 3 and tested on a server equipped with an Intel® Core™ i7–7700HQ CPU @3.8 GHz and with 16 GB of RAM running Fedora 27 $x64$.

5.5.4 Detection performance of the Entropy-based algorithm

This section provides an experimental evaluation of the effectiveness of entropy-based anomaly detection against all the attack scenarios presented in Section 5.1.2. The model used for the detection of anomalies is the model created with a window of 0.1 seconds, i.e. the model with the lowest entropy values from Figure 5.15.

Replay attack

The detection performance of the entropy-based algorithm are tested against the injection of messages already seen on the CAN bus. Since the algorithm

Algorithm 6 Model creation and validation for the Entropy-based algorithm

```
1:  function ENTROPY($messageWin_{CAN}$)
2:      $wLen \leftarrow len(messageWin_{CAN})$
3:      $counters \leftarrow InitializeCounters()$
4:      for $msg$ in $messageWin_{CAN}$ do
5:          $counters[msg] + +$
6:      $probs \leftarrow counters/wLen$
7:      $entropy \leftarrow 0$
8:      for $i$ in $messageWin_{CAN}$ do
9:          $entropy+ = probs[i] * \log_2 \left[\frac{1}{probs[i]}\right]$
10:     return $entropy$
11: function MODELCREATION($tw$)
12:     $entVals \leftarrow emptyList()$
13:     $currWin \leftarrow emptyList()$
14:     $startTw \leftarrow 0$
15:     $endTw \leftarrow tw$
16:     for $winDF$ in $data_{seq}[startTw : endTw]$ do
17:         $entVals.append(Entropy(winDF))$
18:         $startTw \leftarrow endTw$
19:         $endTw \leftarrow endTw + tw$
20:     $save(tw, \mu_{entVals}, \sigma_{entVals})$
21: function MODELVALIDATION($model$)
22:     $tw, \mu_{tw}, \sigma_{tw} \leftarrow load(model)$
23:     $k \leftarrow 1$
24:     $startTw \leftarrow 0$
25:     $endTw \leftarrow tw$
26:     for $winDF$ in $data_{seq}[startTw : endTw]$ do
27:         $currEnt \leftarrow Entropy(winDF)$
28:         if not $\mu_{tw} - k * \sigma_{tw} < currEnt < \mu_{tw} + k * \sigma_{tw}$ then
29:             $k \leftarrow k + 1$
30:             $startTw \leftarrow 0$
31:             $endTw \leftarrow tw$
32:         else
33:             $startTw \leftarrow endTw$
34:             $endTw \leftarrow endTw + tw$
35:     $save(tw, \mu_{tw}, \sigma_{tw}, k)$
```

Algorithm 7 Live detection algorithm for Entropy inspection

```
 1: function ENTROPY($messageWin_{CAN}$)
 2:     $wLen \leftarrow len(messageWin_{CAN})$
 3:     $counters \leftarrow InitializeCounters()$
 4:     for $msg$ in $messageWin_{CAN}$ do
 5:         $counters[msg] ++$
 6:     $probs \leftarrow counters/wLen$
 7:     $entropy \leftarrow 0$
 8:     for $i$ in $messageWin_{CAN}$ do
 9:         $entropy += probs[i] * \log_2 \left[\frac{1}{probs[i]}\right]$
10:     return $entropy$
11: function LIVEDETECTION($model$)
12:     $tw, \mu, \sigma, k \leftarrow load(model)$
13:     $startTw \leftarrow 0$
14:     $endTw \leftarrow tw$
15:     for $winDF$ in $data_{seq}[startTw : endTw]$ do
16:         $currEnt \leftarrow Entropy(winDF)$
17:         if not $\mu - k * \sigma < currEnt < \mu + k * \sigma$ then
18:             raise $anomaly$
19:         $startTw \leftarrow endTw$
20:         $endTw \leftarrow endTw + tw$
```

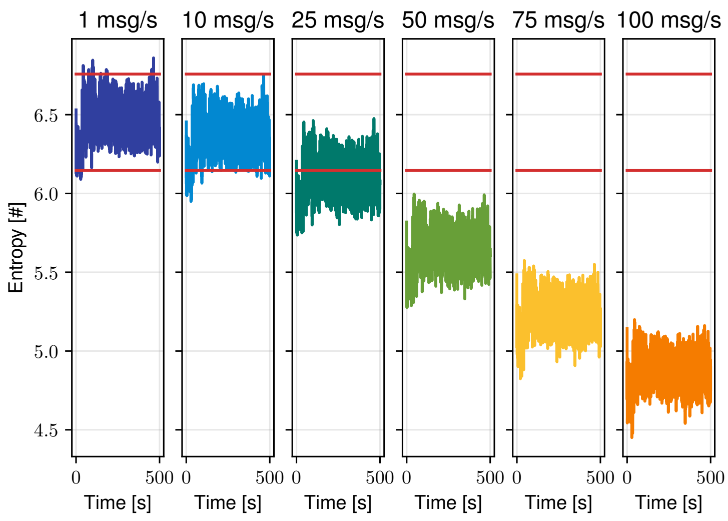


Figure 5.17: Entropy values evaluated over the Single ID replay attack scenario with different attack frequency

inspects the messages belonging to different time windows for the detection, it is necessary to consider different frequencies of injection to provide an accurate analysis of the detection performances of the algorithm. For this purpose, each attack is replicated in each time window $1, 10, 25, 50, 75$, and 100 times.

Single ID Replay In this attack scenario a single message is injected in each time window many times. Since the statistical distribution of the messages evaluated on the whole dataset does not influences the outcome of the algorithm, for this attack scenario a single message belonging to each time window is repeated multiple times. The detection results of the algorithm against this attack scenario are depicted in Figure 5.17, in which (left to right) each plot represents the entropy evaluated over different time windows against the Single ID replay attack with different frequency ($1, 10, 25, 50, 75$, and 100 injected messages). The *x-axis* of each plot of Figure 5.17 represents the time, while the *y-axis* represents the entropy value. Two horizontal lines in each plot are used to define the normal entropy range within no anomaly is raised.

From the analysis of the results depicted in Figure 5.17 it is possible to notice that the algorithm is not able to detect anomalies consistently in case of a replay attack with frequencies below 50 messages each time window. The algorithm is able to detect anomalies consistently starting with the injection of 50 messages each time window. The results also depicts that, by injecting

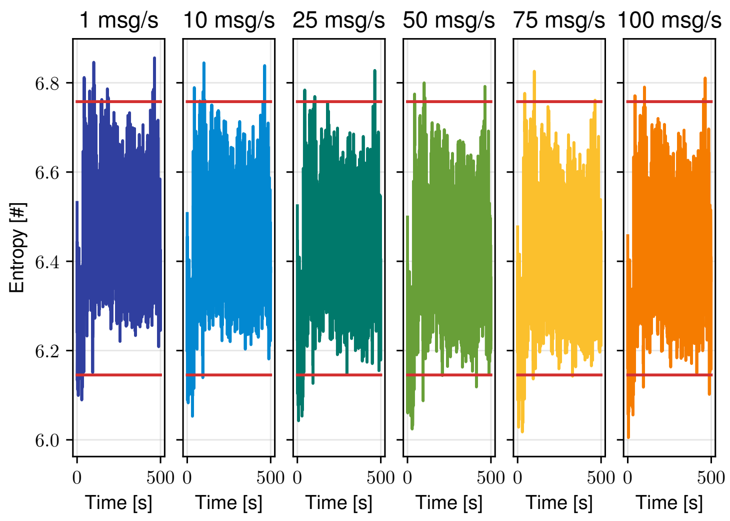


Figure 5.18: Entropy values evaluated over the Sequence replay attack scenario with different attack frequency

multiple time the same message in the time window the entropy values follows a decreasing trend. This is explained by considering that, by adding multiple times a single message to the same time window, the probability of that message increases and thus, according to Equation 5.5 the value of $mathcalH$ decreases. Moreover, it is interesting to notice that by injecting less than 50 messages in each time windows it is not possible to always detect an anomaly. However, it is also to notice that depending on the time window, even the injection of a single message could be detected, despite the injection attack has higher probabilities of being unnoticed.

Sequence Replay In this attack scenario multiple messages are injected in each time window many times by choosing a random message between the ones belonging to the time window. The detection results of the algorithm against this attack scenario are depicted in Figure 5.18, in which (left to right) each plot represents the entropy evaluated over different time windows against the Sequence replay attack with different frequency $(1, 10, 25, 50, 75,$ and 100 injected messages). The *x-axis* of each plot of Figure 5.18 represents the time, while the *y-axis* represents the entropy value. Two horizontal lines in each plot are used to define the normal entropy range within no anomaly is raised. From the analysis of the results depicted in Figure 5.18 it is possible to notice that the algorithm struggles to detect anomalies consistently despite the increasing number of messages injected in each time window. This result is explained by considering the simulated attack scenario. Since the attack

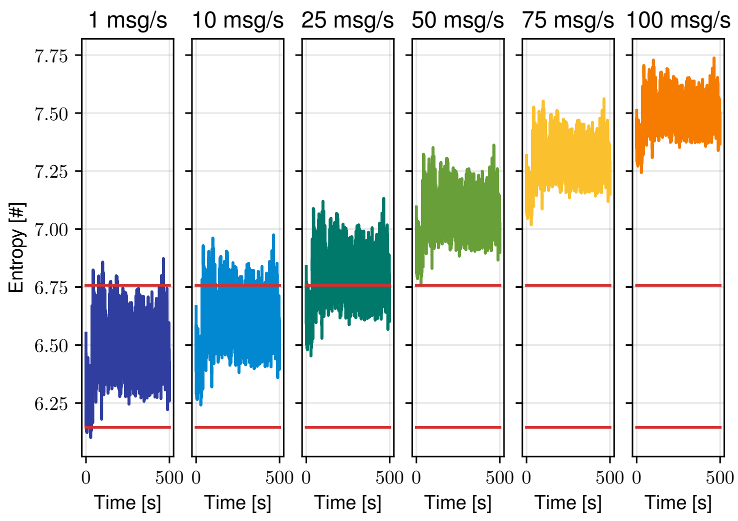


Figure 5.19: Entropy values evaluated over different fuzzing attack scenarios

is simulated by replaying different messages randomly chosen between the ones belonging to the inspected time window, despite the number of overall messages in the time window increases, the probabilities of each message do not change. As a consequence, since for the evaluation of $\mathcal{H}$ the probability of the message is considered, the evaluated entropy value does not change significantly from the normal reference, thus preventing the algorithm from detecting any anomaly.

Fuzzing attack

In this attack scenario multiple messages are injected in each time window. Each message is generated with a random ID and data field, thus the Fuzzing ID and Fuzzing payload attack scenario are inspected at the same time. The detection results of the algorithm against this attack scenario are depicted in Figure 5.19, in which (left to right) each plot represents the entropy evaluated over different time windows against the Fuzzing attack with different frequency $(1, 10, 25, 50, 75,$ and 100 injected messages). The *x-axis* of each plot of Figure 5.19 represents the time, while the *y-axis* represents the entropy value. Two horizontal lines in each plot are used to define the normal entropy range within no anomaly is raised.

From the analysis of the results depicted in Figure 5.19 it is possible to notice that, as for the results depicted against the Single ID replay attack (Section 5.5.4), the algorithm is able to consistently detect anomalies with an injection of at least 50 messages each time window. Another interesting behavior to notice is that the injection of random messages causes the entropy value to increase, which is the exact opposite behavior observed against the single ID replay attack. This trend is explained using the same criteria already presented in 5.5.4. The fuzzing injection attack randomly generates messages to inject in each time window. Since these messages are randomly-generated, there are high probability that the generated messages are not seen in the time windows, meaning that the injected message has an high informative meaning, thus increasing the entropy evaluated in the window as a consequence.

ECU Inhibition

In this attack scenario no different assumptions have been made with respect to the definition of the attack provided in Section 5.1.2. Since this attack removes messages from the network, the number of messages removed from the network changes according to the percentile of the removed ID, thus removing a fixed number of messages from each time window is not representative of a real-case attack scenario. The detection results of the algorithm against this attack scenario are depicted in Figure 5.20, in which (left to right) each plot represents the entropy evaluated over different time windows against the removal of messages belonging to different percentiles $(10, 50, 75$, and $100)$. The *x-axis* of each plot of Figure 5.19 represents the time, while the *y-axis* represents the entropy value. Two horizontal lines in each plot are used to define the normal entropy range within no anomaly is raised.

From the analysis of the results depicted in Figure 5.20 it is possible to notice that the algorithm struggles to detect anomalies consistently independently from the distribution of the ID removed. This result is explained by considering the simulated attack scenario. Since the attack is simulated by removing messages, the resulting time windows are composed by fewer messages. The analysis of the resulting time windows depicted that each time window is composed by up to 5 less messages compared to the time windows used for the definition of the model, thus the resulting $\mathcal{H}$ value does not change significantly from the normal reference.

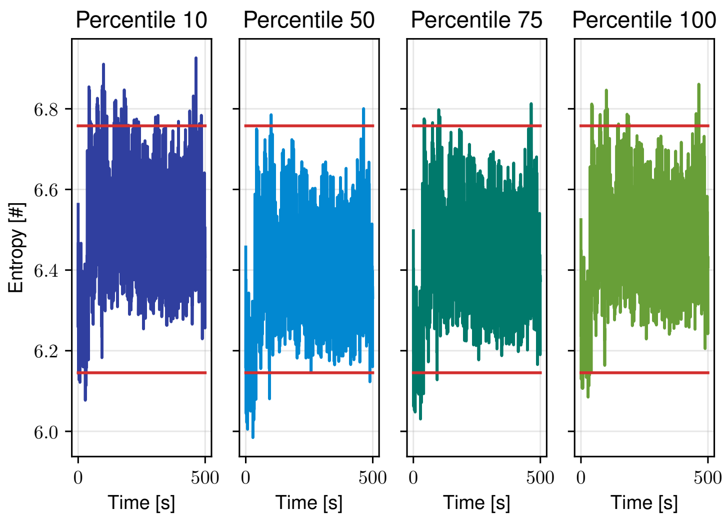


Figure 5.20: Entropy values evaluated over different ECU inhibition attack scenarios

5.6 Detection algorithm based on bus utilization

5.6.1 Fundamentals

The bus utilization is a metric used for the evaluation of the bandwidth used by the nodes of the network for their communication. The utilization $\mathcal{U}$ of each second of bus communication in which i messages are sent is defined according to equation 5.6.

$$\mathcal{U} = \frac{\sum_i len(CANmsg_i)}{baudrate} \tag{5.6}$$

where $len(CANmsg_i)$ represents the number of bits of the message $CANmsg_i$ belonging to the inspected time interval, while *baudrate* is a constant value representing the operating bit-rate of the network. A feasibility study on this detection technique is conducted by analyzing the first 30 minutes of the first trace of the dataset. The results of this preliminary analysis are depicted in Figure 5.21, which shows the distribution of the utilization evaluated over the first minutes of traffic.

From the analysis of the preliminary results depicted in Figure 5.21 it is possible to notice that the distribution of the utilization (x-axis) is comparable

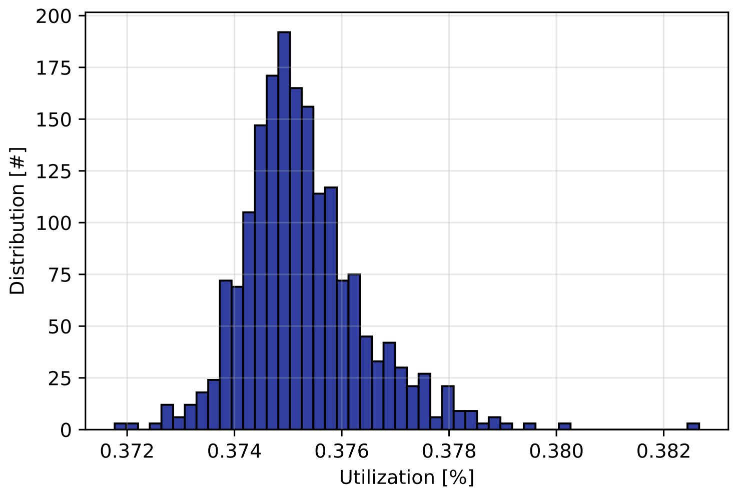


Figure 5.21: Distribution of the utilization in the first 1800 seconds of the first trace of the dataset

to the distribution of a normal function centered around the mean value (0.375257).

5.6.2 Comparison with the state-of-the-art

The state-of-the-art for the anomaly detection on CAN (described in Section 3.3) is composed by algorithms that inspect the different features available in CAN, one being the analysis of statistical indexes evaluated over the CAN traffic. Compared to the state-of-the-art, to the best of our knowledge, the algorithm proposed in this Section is the first algorithm that inspects the CAN bus utilization as its detection feature.

5.6.3 Busload detection algorithm

The algorithm based on the evaluation of the bus utilization is composed by two different phases: an initial *training phase*, in which the parameters that minimizes the number of false positives are evaluated, and the *detection phase*, which is used for the detection of anomalies. The detection algorithm uses the same assumptions used for the Entropy-based detection algorithm (Section 5.5). Since the bus utilization values appear to be stable over time and distributed according to a normal distribution, in each non-overlapping time window t (fixed at 1.0 second), the utilization $\mathcal{U}$ is evaluated using equation 5.6 and compared to the normal utilization range defined as $[\mu - k \times \sigma, \mu + k \times \sigma]$, where μ is the utilization mean value, σ is its standard deviation, and k is a tuning parameter. If the utilization falls outside of the defined normal range an anomaly is raised. The training phase is used for the evaluation of the parameters μ and σ required for the validation of the algorithm. In the validation process the value of k is initially set to 1, and it is incremented by 1 each iteration to achieve 0 false positives on the clean traces. The pseudocode for the *training and validation* phase is depicted in Algorithm 8, while the *detection* phase is depicted in Algorithm 9.

For the tests proposed in this thesis, the bus utilization algorithm is implemented with Python 3 and tested on a server equipped with an Intel® Core™ i7–7700HQ CPU @3.8 GHz and with 16 GB of RAM running Fedora 27 $x64$.

5.6.4 Detection performances of the bus utilization algorithm

The detection results of the bus utilization algorithm are tested against all the attack scenarios described in Section 5.1.2, since the proposed detection method inspects the utilization of the network, it is possible to apply it to the different attack scenarios described in 2.3 by considering the manipulation of the network required for the attack. For this purpose it is necessary to define a novel description of the attack scenarios that do not considers only the typology of the attack but inspects the required network manipulations. The attacks previously defined as *replay attack*, *fuzzing attack* and *denial of service attack* require the injection of extra messages on the network. Despite the different characteristics of the different attacks, since the bus utilization algorithm only inspects the number of bits in each second of the network, all these attack scenarios could be inspected as an equivalent injection of messages under the scenario of the **Message injection attack scenario**. The attacks described as *ECU shutdown* and *ECU inhibition* both consider the attack scenario in which messages are removed from the network. For this purpose, since the ECU shutdown attack is a limited case of ECU inhibition, only the detection results of the algorithm against the ECU inhibition are inspected under the scenario of the **Message removal attack scenario**.

Message injection attack detection

For the simulation of the different message injection scenarios, a random message is injected 1, 10, 25, 50, 75, and 100 times each second. The detection results of the bus utilization algorithm against the message injection scenario

Algorithm 8 Training and validation algorithm for the bus utilization inspection

```
function UTILIZATION($messageWin_{CAN}$)
    $bitCount \leftarrow 0$
    for $msg$ in $messageWin_{CAN}$ do
        $bitCount += len(msg)$
    return $bitCount/baudrate$
function MODELCREATION
    $utilList \leftarrow emptyList()$
    $startTw \leftarrow 0$
    $endTw \leftarrow 1$
    for $win$ in $data_{seq}[startTw : endTw]$ do
        $utilList.append(Utilization(win))$
        $startTw \leftarrow endTw$
        $endTw \leftarrow endTw + 1$
    $save(\mu_{utilList}, \sigma_{utilList})$
function MODELVALIDATION($model$)
    $\mu_{tw}, \sigma_{tw} \leftarrow load(model)$
    $k \leftarrow 1$
    $startTw \leftarrow 0$
    $endTw \leftarrow 1$
    for $win$ in $data_{seq}[startTw : endTw]$ do
        $currUtil \leftarrow Utilization(win)$
        if not $\mu_{tw} - k * \sigma_{tw} < currEnt < \mu_{tw} + k * \sigma_{tw}$ then
            $k \leftarrow k + 1$
            $startTw \leftarrow 0$
            $endTw \leftarrow tw$
        else
            $startTw \leftarrow endTw$
            $endTw \leftarrow endTw + 1$
    $save(\mu_{tw}, \sigma_{tw}, k)$
```

Algorithm 9 Detection algorithm for the bus utilization inspection

```
 1: function LiveDetection($model$)
 2:     $\mu, \sigma, k \leftarrow load(model)$
 3:     $startTw \leftarrow 0$
 4:     $endTw \leftarrow 1$
 5:     for $win$ in $data_{seq}[startTw : endTw]$ do
 6:         $currUtil \leftarrow Utilization(win)$
 7:         if not $\mu - k * \sigma < currEnt < \mu + k * \sigma$ then
 8:             raise $anomaly$
 9:         $startTw \leftarrow endTw$
10:         $endTw \leftarrow endTw + 1$
```

are depicted in Figure 5.22, in which the x-axis represents the time of the inspected traffic trace while the y-axis represents the evaluated percentage of bus utilization. The values depicted in Figure 5.22 represent (bottom to top) the utilization percentage evaluated by injecting $1, 10, 25, 50, 75$, and 100 messages each second.

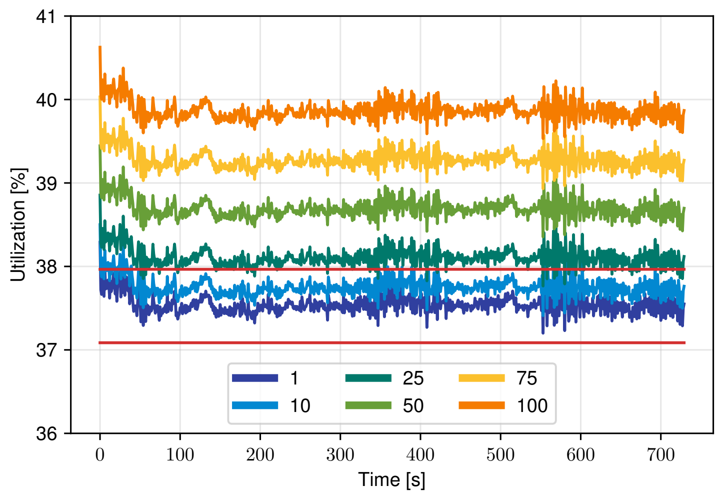


Figure 5.22: Detection results of the bus utilization algorithm

The results depicted in Figure 5.22 show that to achieve stable detection results against the injection attack it is necessary to inject 25 or more messages each second. By injecting 1 single message each second the attack goes undetected, while with an injection of 10 messages each second it is not possible to achieve stable results, despite some anomalies are detected by the algorithm.

Message removal attack detection

The detection results of the bus utilization algorithm are depicted in Figure 5.23, in which the x-axis represents the time of the inspected traffic trace while the y-axis represents the evaluated percentage of bus utilization. The values depicted in Figure 5.23 represent (bottom to top) the utilization percentage evaluated by removing messages representing the 10_{th}, 50_{th}, 75_{th}, and 100_{th} percentile.

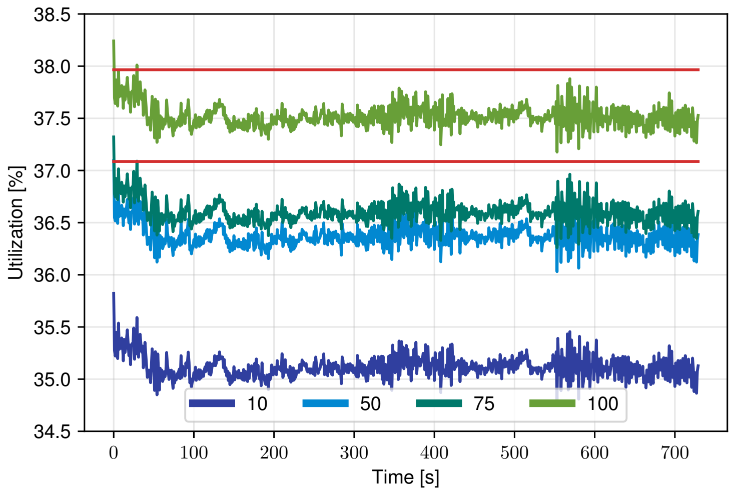


Figure 5.23: Detection results of the bus utilization algorithm

The results depicted in Figure 5.22 show that the detection capabilities of the bus utilization algorithm are related to the percentile of the removed message. By removing messages representing lower percentile values (i.e. messages that appears on the bus with higher frequencies) the algorithm is able to detect those removal accurately in each second of the inspected traffic trace, with an utilization that drops around 35.5%, 36.25% and 36.75% for the messages representing the 10_{th}, 50_{th} and 75_{th} percentiles, respectively. As already presented in Section 5.3.3, in case of removing the ID representing the 100_{th} percentile only one message is removed from each trace, thus the bus utilization algorithm fails to detect any anomaly.

			5.2	5.3	5.4	5.5	5.6
Message Injection	Replay	Single ID		X	X	X	X
		Valid Sequence		X		X	X
		Invalid Sequence		X		X	X
	Fuzzing	Random ID		X		X	X
		Random Payload		X	X	X	X
	Denial of Service	Lowest ID		X		X	X
		Zero ID		X		X	X
Message Removal		ECU Shutdown	X	X		X	X
		ECU Inhibition	X	X		X	X

Table 5.2: Summary of the attack detection capabilities of the proposed algorithms against the inspected attack scenarios

5.7 Detection method summary

This Section summarizes all the different approaches presented in this Chapter and evaluates their effectiveness in the detection of the attack scenarios presented in Section 5.1.2.

5.7.1 Single ID replay attack

The Single ID replay attack is tested against four detection algorithms proposed in this Chapter: Algorithm 5.3 (message ID sequence), 5.4 (Hamming distance), 5.5 (entropy), and 5.6 (bus utilization). From the comparison of the detection results achieved by the aforementioned algorithms against this attack scenario, different conclusions are inspected. At first, it is noticeable that 5.4 achieves poor detection results against this attack scenario. Similar results are achieved for both 5.5 and 5.6: despite both algorithms are proven effective in the detection of the injection of a big number of messages, real case attack scenario often uses a small number of message to achieve their goals, thus the algorithms are not suitable to detect this typology of attack. The only algorithm proven effective in the detection of a Single ID replay attack is 5.3, which is able to achieve good detection results against any of the inspected scenarios, despite it is necessary to deploy the detection model created with the highest value of the parameter n.

5.7.2 Valid and Invalid sequence replay attack

Both Valid and Invalid sequence replay attack are tested against the same three detection algorithms proposed in this Chapter: Algorithm 5.3 (message ID sequence), 5.5 (entropy), and 5.6 (bus utilization). From the comparison of the detection results achieved by the aforementioned algorithms against both attack scenarios, it is possible to notice that the detection results of Algorithm 5.3 outperform the ones achieved by 5.5 and 5.6. With Algorithm 5.3 it is possible to detect the injection of both valid and invalid sequences of messages, with different models achieving different results according to the length of the attack (from a sequence of length 2 to a sequence of length 10). In case of 5.5 and 5.6, by injecting a valid or invalid sequence composed by 10 messages in each time window, the evaluated entropy or utilization of the window is not outside the normal range, thus the algorithms are not able to detect any of this attacks. Moreover, in 5.3 the attacks are simulated with a frequency of 1 sequence each second.

5.7.3 Fuzzing ID attack

The Fuzzing ID replay attack is tested against three detection algorithms proposed in this Chapter: Algorithm 5.3 (message ID sequence), 5.5 (entropy), and 5.6 (bus utilization). From the comparison of the detection results achieved by the aforementioned algorithms against the Fuzzing ID attack scenario, it is possible to notice that 5.3 is by far the most effective and simple solution to deploy for the detection of IDs never observed on the network. The model with length $n = 1$ is composed by all the IDs observed in the training phase of Algorithm 5.3, thus if an ID is not observed in the training phase, 5.3 automatically detects it. The only drawback of this approach is that, since 5.3 is limited on the analysis of the dataset collected for the inspected vehicle, valid IDs never observed in the training dataset would be mistakenly classified as anomalous. However, from the comparison of the results achieved by both 5.5 and 5.6 it is noticeable that, despite the injected ID is created randomly, both algorithms are able to detect anomalies consistently while injecting 50 or more messages in the inspected time reference.

5.7.4 Fuzzing payload attack

The Fuzzing payload replay attack is tested against four detection algorithms proposed in this Chapter: Algorithm 5.3 (message ID sequence), 5.4 (Hamming distance), 5.5 (entropy), and 5.6 (bus utilization). From the comparison of the detection results achieved by the aforementioned algorithms it is possible to notice that Algorithm 5.5 and 5.6 achieve the same results achieved against the fuzzing ID attack scenario (5.7.3). Moreover, it is interesting to notice that, despite 5.3 is not designed to detect anomalies on the payload and each payload is related to a particular ID, it is possible to inspect this feature for the detection of anomalies with good results by deploying models created with high value of n. However, 5.4 is designed to detect anomalies directly on the payload of the messages and provides good detection results against this attack scenario, almost achieving 100% detection of anomalies in the inspected class of IDs.

5.7.5 Denial of service attack

Since the assumptions in the generation of the infected dataset for this attack scenario demonstrated that the Lowest ID and the Zero ID attacks are equal, the results of the algorithms tested against this attack scenario are presented for both attacks at the same time. Both Denial-of-Service replay attacks are tested against three detection algorithms proposed in this Chapter: Algorithm 5.3 (message ID sequence), 5.5 (entropy), and 5.6 (bus utilization), and all of these algorithms are able to achieve high detection results against both Denial-of-Service scenario attacks. Algorithm 5.3 achieves perfect results by deploying a detection model created with a value of $n \geqslant 6$, thus it does not require memory-expensive models. Algorithm 5.5 and 5.6 both are able to identify the injection of 100 messages in the respective time frame with absolute precision. It is to notice that, despite the Denial-of-Service attacks are simulated with the injection of 100 extra messages each second in 5.3 and 5.6, the attacks simulated in 5.5 injects 100 messages each time window with duration of 0.1 seconds. Despite the inspected time window is different, and since the Denial-of-Service attack targets the whole network by injecting at least 100 messages to disrupt the communication for 0.2 seconds (see Section 5.1.2), it is possible to deploy 5.5 for the detection of Denial-of-Service attacks efficiently, since it is able to detect correctly anomalies achieved by injecting 50 messages each window with duration 0.1 seconds (thus 100 messages spawned in a window of duration 0.2 seconds).

5.7.6 ECU shutdown attack

The ECU shutdown attack is tested against four detection algorithms proposed in this Chapter: Algorithm 5.2 (missing message), 5.3 (message ID sequence), 5.5 (entropy), and 5.6 (bus utilization). From the comparison of the detection results achieved by the aforementioned algorithms, it is possible to notice that Algorithm 5.2 outperforms all the other detection methods, and is the best performing detection algorithm in this attack scenario. Algorithm 5.6 is able to achieve good detection results, but its results are related to the removed message. This result also applies for 5.3, but its overall detection performances are lower. Algorithm 5.5 achieved the worst performance in the inspected attack scenario, since the removal of messages does not reflects in a consistent change of the entropy of the values.

5.7.7 ECU inhibition attack

The ECU inhibition attack is tested against four detection algorithms proposed in this Chapter: Algorithm 5.2 (missing message), 5.3 (message ID sequence), 5.5 (entropy), and 5.6 (bus utilization). The results presented in Section 5.7.6 are the same achieved by the algorithms in this scenario. Algorithm 5.5 is not able to detect any anomaly since the removal of messages from the network does not affect the entropy to significant changes. Algorithm 5.3 is able to detect anomalies but its performance are not as good as the ones achieved with 5.6, which are tested against the removal of messages with different percentiles. Algorithm 5.2 is still the best detection method to deploy for the detection of any message removal attack.

5.8 Detection framework

Based on the results presented in Section 5.7, this Section proposes a unified detection framework that leverages the best algorithm for each attack scenario. Figure 5.2 depicts a summary of the effectiveness of the detection algorithm against the different attack scenario, as already discussed in Section 5.7. The detection performance of the algorithms are categorized with different symbols and colors:

- △: good detection results against the simulated attack;
- ◊: the algorithm is able to detect anomalies despite some limitations;
- ▽: the algorithm is not able to detect anomalies.

From the final summary shown in Table 5.3 it is possible to notice that for the definition of a minimal detection framework able to detect anomalies generated as described in Section 2.3 it is necessary to deploy only 2 of the inspected algorithms: the missing message algorithm 5.2 and the message

	5.2	5.3	5.4	5.5	5.6
Single ID		△	▽	▽	▽
Valid Sequence		△		▽	▽
Invalid Sequence		△		▽	▽
Random ID		△		◇	▽
Random Payload		△	◇	◇	▽
Lowest ID		△		△	△
Zero ID		△		△	△
ECU Shutdown	△	◇		▽	◇
ECU Inhibition	△	◇		▽	◇

Table 5.3: Summary of the attack detection capabilities of the proposed algorithms against the inspected attack scenarios

sequence algorithm 5.3, which are the best performing algorithms against the message removal scenarios and the message injection scenarios, respectively. As a final remark, it is necessary to recall that both Entropy and Utilization detection algorithm are affected by the number of injected anomalies and, despite they under-performed against the simulated attacks, their detection capabilities against other attack scenario has already proven effective.

Chapter 6

Detection techniques based on CAN data frame payloads

In this Chapter different algorithms for the detection of anomalies on the CAN bus based on closed standard are proposed. Since these algorithms inspect the values of the signals encoded in the data field of a CAN data frame, it is necessary to extract those signals from the data field to inspect them. There are two different ways to extract the signals from a CAN data frame. The first solution requires access to the specification of the vehicle model, which are encoded in a file called DBC. The DBC file contains all the specifications of a particular vehicle model: the list of message IDs sent on the network, their cycle-time, the ECUs that generates each ID, the encoding of the payload of the messages, and the reference of each signal (the name, the offset, the scaling factor, the measurement unit of the signal and so on). Since the DBC allows us to obtain complete knowledge of the inspected CAN segment, car manufacturers keep these files extremely confidential and they are not publicly available. Another method to access the signals encoded in a CAN data frame is to inspect the content of the messages to reverse-engineer the boundaries of each signal. The reverse engineering of CAN data frames is a tasks that has been used by security researchers to obtain knowledge of the network, allowing them to inject maliciously forged messages for their purposes. Despite this task has been widely used by many security researchers, each vehicle model requires a dedicated reverse-engineering process, thus security researchers often inspect the messages manually to reverse-engineer them, making the whole process extremely time consuming. For this purpose an algorithm for the *Reverse Engineering of Automotive Data frames* (READ) is presented in this Chapter at first. The output of READ is used for the definition of different detection algorithms focused on the analysis of the signals encoded in the data field.

6.1 Reverse Engineering of Automotive Data frames

The Reverse Engineering of Automotive Data frame (READ) algorithm proposed in this Section is used for the extraction of individual signals from CAN traffic, by inspecting all the bits of the data field of all observed CAN messages and evaluating their evolution over time.

ECUs communicate by exchanging messages that deliver values gathered from different sensors to actuators that control several subsystems of the vehicle. In particular, most of the signals generated by sensors encode the current value of a given physical phenomena, such as the speed at which a wheel is rotating or the acceleration measured along a given axis. The evolution over time of similar signals is unpredictable, since it depends on the road and driving conditions. However it is clearly limited by physical constraints. Since many signals are issued on the CAN bus according to a predefined cycle time (such as a hundredth or a tenth of a second), the difference among two consecutive values of a signal representing a physical phenomena is necessarily small, and constrained by the cycle time and by the nature of the observed phenomena. As an example, consider the CAN signal representing the rotation speed of the front left tire, assuming that this signal is sent over the CAN bus every 10 milliseconds. Consecutive values will necessarily be very similar, hence only their less significant bits will change. However, over time the tire rotational speed will change significantly, thus leading to the modification of the more significant bits of the signal.

READ analyzes the ordered sequence of payloads of CAN messages having the same ID. Each bit of the payload is analyzed to determine the frequency of changes in the bit value among consecutive payloads (bit-flip). The READ algorithm applies several novel heuristics to:

- define and extract the boundaries of different signals within payload of messages having the same CAN ID;
- label extracted signals according to different classes representing the signal type.

The different phases of the READ algorithm are described in the following sections.

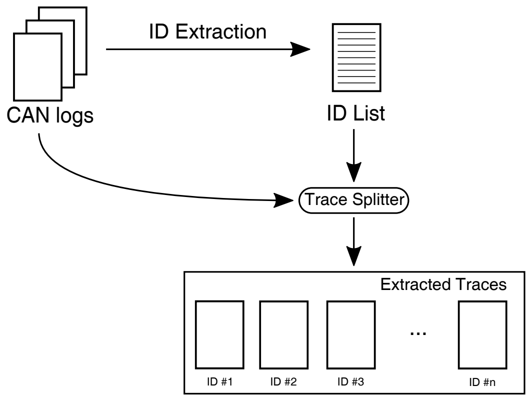


Figure 6.1: Preparation phase of READ

6.1.1 The READ Algorithm

Data preparation

The preliminary step required to execute READ is to create ordered lists of CAN messages having the same ID. The input data consists of a CAN traffic trace including all CAN messages exactly as they were transmitted over the CAN bus of a licensed vehicle. The CAN traces are recorded by an unmodified, licensed vehicle provided to us by an undisclosed industrial partner, alongside the full specifications of the internal vehicle network. In this preliminary phase, the single trace is split into several sub-traces, one for each different ID included in the original input. Each sub-trace only contains the payload of CAN messages having a single ID, in the same order as they appear in the original input. Since READ analyzes each sub-trace independently, it is possible to have multiple instances of READ running in parallel on different sub-traces. An overview of the data preparation phase of the READ algorithm is given in Figure 6.1.

READ processing steps

The algorithm does not rely on any a-priori knowledge about the nature of the message payloads nor of the signals encoded within the payloads. The input of the algorithm is composed by a list of payload values composed by a number of bits dependent on the payload length as specified by the DLC field. The output is represented by the list of signals included in the payload, their boundaries, and a label describing the type of each signal. An overview of the READ algorithm workflow is given in Figure 6.2.

Figure 6.3 shows the detailed steps performed by the READ algorithm.

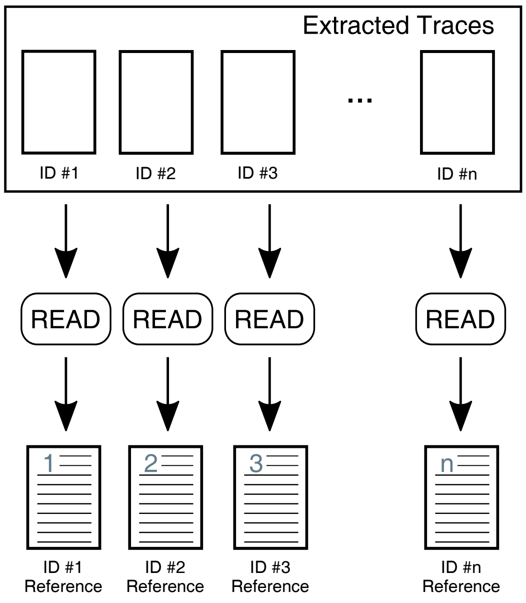


Figure 6.2: READ work-flow

The pre-processing phase analyzes the message payloads and computes the metadata used by the next phases. Phase 1 evaluates the preliminary references for the signals that are further refined in Phase 2.

Pre-processing The bit-flip rate is evaluated for each bit of the payload, independently of its neighbors. First READ counts the number of bit-flips (from 0 to 1 and vice-versa) occurrences among consecutive messages. Then the bit-flip rate is obtained by dividing the number of bit-flips for the number of payloads. The result of this phase is an array of n elements, each representing the bit-flip rate for a single bit of the data field of a given ID, where n represents the number of bits included in the payload as defined by the value

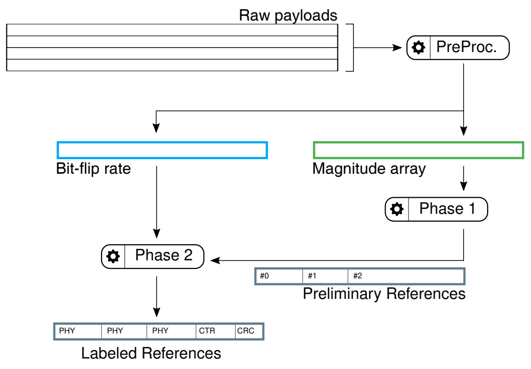


Figure 6.3: Processing steps of the READ algorithm

of the *DLC* field.

This intermediate result is an input for the computation of another array of n elements, defined as the *magnitude* array. The formula used to compute the magnitude array is shown in Equation 6.1, where M_i is the i_{th} element of the magnitude array and B_i is the i_{th} element of the bit-flip array.

$$M_i = \lceil \log_{10}(B_i) \rceil, \quad 0 \leqslant i < n \tag{6.1}$$

Values of the magnitude array represent the different orders of magnitude of the bit-flip rate for each bit of the payload. The pseudo-code describing the pre-processing step of the READ algorithm is included in Algorithm 10.

Algorithm 10 Pseudo-code of the pre-processing step

```
1: function PRE-PROCESSING($messageList, DLC$)
2:     $payloadLen \leftarrow$ len($messageList$)
3:     $bitFlip \leftarrow$ array($DLC$)
4:     $magnitude \leftarrow$array($DLC$)
5:     $previous \leftarrow messageList[0]$
6:     while $item$ in $messageList$ do
7:         for $ix$ in range $(1..DLC)$ do
8:             if $item[ix] \neq previous[ix]$ then
9:                 $bitFlip[ix] + +$
10:    for $ix = 0; ix < DLC; ix + +$ do
11:        $bitFlip[ix] \leftarrow bitFlip[ix]/payloadLen$
12:        $magnitude[ix] \leftarrow \lceil \log_{10}(bitFlip[ix]) \rceil$
13:    return $bitFlip, magnitude$
```

After computing the bit-flip rate and magnitude arrays, the algorithm inspects them to identify the signal boundaries. This process involves two phases. The first phase only considers the magnitude array, and produces a preliminary list of signal boundaries as output. The second phase leverages these preliminary boundaries and the bit-flip rate to identify the precise boundaries of each signal and to label them according to their nature.

Phase 1 In this phase the magnitude array is used for the definition of preliminary signal boundaries. The algorithm scans the magnitude array looking for couples of consecutive bits in which the first bit is characterized by a bit-flip magnitude that is higher then the second one. Whenever a similar couple is found, a preliminary boundary is set between the two bits. This heuristic is effective in the identification of signal boundaries that represent physical values, since drops in the bit-flip magnitude are caused by a less significant bit of a signal immediately followed by the most significant bit of the adjacent signal. The pseudo-code describing the main processing steps of Phase 1 is given in Algorithm 11

Algorithm 11 Signal boundary identification

```
1: function PHASE1($magnitude, DLC$)
2:     $ref \leftarrow list()$
3:     $prevMagnitude \leftarrow magnitude[0]$
4:     $ixS \leftarrow 0$
5:     for $ix$ in range($1..DLC$) do
6:         if $magnitude[ix] < prevMagnitude$ then
7:             $ref.add((ixS, ix - 1))$
8:             $ixS \leftarrow ix$
9:         $prevMagnitude \leftarrow magnitude[ix]$
10:    $ref.add((ixS, DLC - 1))$
11:    return $ref$
```

Phase 2 The second phase takes as input the preliminary boundaries identified in the previous phase together with the bit-flip rate array to identify and correctly label signals that do not represent physical values. Car manufacturers often include metadata in the payload of safety-critical messages to implement two naïve protection strategies that are effective against basic replay attacks [83]. Common solutions adopted by several car manufacturers include two additional types of fields: *Counters* and *CRCs*. Since these metadata are encoded in the payload of CAN messages together with other signals that convey physical values, it is helpful for an analyst to quickly identify them and tell them apart. This is the rationale behind the definition of heuristics specifically tailored to identify counters and CRCs.

Counters are signals whose value always increases by one with respect to the counter of the previous message with the same ID. Counters allow the receiving ECU to recognize a re-transmission of a CAN frame that has already been received correctly, as well as messages that are received out of order. Similar conditions happen when an attacker that sniffed a message from the CAN bus performs a replay attack by injecting the same message over the bus.

Counters exhibit two peculiar features that differentiate them from messages conveying the value of a physical phenomena:

1. the magnitude of the least significant bit equals 0, since it has a bit flip probability of 1;
2. the bit-flip rate of the elements doubles every step from the most to the least significant one, reaching 1 in the least significant bit.

By looking for similar patterns it is possible to identify counters within signals extracted in the previous phase. As example let us consider the case of a 4-bits counter having the following bit-flip rates: [0.125, 0.25, 0.5, 1]. By applying Equation 6.1 the magnitude array will be: [0, 0, 0, 0]. Since magnitude values do not change, Phase 1 will fail in identifying its boundaries, and the counter will remain embedded in other signals. However, since bit-flip rates vary according to the defined heuristic, the counter boundaries will be correctly identified by Phase 2.

CRCs are signals that contain the result of a cyclic redundancy check on the message payload to detect random transmission errors in safety-relevant signals. We remark that CRC signals are included within the payload of CAN messages, and do not replace (and should not be mistaken for) the CRC field that follows the payload in CAN data frames (see Figure 2.3). The algorithm for computing the CRC that follows the payload is public and described in the CAN bus specifications [26], while the algorithm used for evaluating the CRC signals within the message payload is proprietary. Empirical analyses of CAN messages including CRC signals allows us to conclude that CRC signals as exhibiting the following distinctive features:

1. the magnitude of all bits is equal to 0;
2. the bit-flip rate of all bits is distributed according to a normal probability distribution centered in 0.5.

Similarly to the counter fields, boundaries of CRC fields are not detected in Phase 1, and are identified in this phase by looking for the aforementioned pattern of bit-flip rates. If this pattern is found, precise boundaries are set and the signal is labeled as CRC. Algorithm 12 shows the pseudo-code related to Phase 2.

The final output of READ is a list of all the IDs found in the input CAN traffic traces, in which each ID is associated to the number of signals identified by READ and their boundaries. Moreover, each signal is classified as a *Physical value*, a *Counter* or a *CRC*.

Analysis of the READ algorithm

Three different aspects of READ are analyzed: computational complexity, correctness and convergence requirements.

READ sequentially applies the three processing steps previously described, hence it is possible to evaluate the computational complexity of each step. The computational complexity of the pre-processing phase (see Algorithm 10) depends from two factors: the size of the payload of the analyzed CAN

Algorithm 12 Signal boundary enhancement and labeling

```
1: function PHASE2($ref, bitFlip$)
2:     $rRef \leftarrow list()$
3:     for $sign$ in $ref$ do
4:         $ixS, ixE \leftarrow sign$
5:         $mu \leftarrow mean(bitFlip[ixS : ixE])$
6:         $std \leftarrow stdDev(bitFlip[ixS : ixE])$
7:         if $bitFlip[ixE] = 0$ and $matchCounter(bitFlip[ixS : ixE])$ then    ▷ $matchCounter$ returns the bit-flip pattern defined in the Counter heuristic
8:             $rRef.add((ixS, ixE, COUNTER))$
9:         else if all$(bitFlip[ixS : ixE] = 0)$ and $0.5 - std \leqslant mu \leqslant 0.5 + std$ then
10:            $rRef.add((ixS, ixE, CRC))$
11:        else
12:            $rRef.add((ixS, ixE, PHYS))$
13:    return $rRef$
```

messages and the number of messages included in the analyzed trace. While the payload size varies among different CAN IDs it is constant for all messages having the same ID. Moreover, it is bounded by the constant value of 64. On the other hand, the number of iterations performed by the outer cycle grows linearly with the number of messages included in the analyzed traffic trace. Hence the computational complexity of the pre-processing step grows linearly with the number of messages. The computational complexity of Phase 1 (see Algorithm 11) only depends on the payload size, hence this step has a constant computational complexity. Finally, the computational complexity of Phase 2 (see Algorithm 12) depends on the number of signals extracted by Phase 1. While the number of signals is not known a-priori, it is limited by the payload size. Hence the computational complexity of Phase 2 is also constant. From the previous analysis it is possible to conclude that the overall computational complexity of READ grows linearly with the number of analyzed messages. An experimental evaluation of the execution times of READ over real CAN traffic traces is provided in Section 6.1.2.

For READ, as for any data-driven reverse engineering approach, convergence and correctness are two closely related concepts. READ results are correct if they agree with the formal specification of CAN messages, that represent an arbitrary design choice rather than a theoretical correct or optimal solution. Moreover, READ can converge to correct results only after the analysis of a number of messages. In the worst case, all messages analyzed by READ have an identical payload value. Borrowing from the information theory, we can say that a similar sequence of messages has a conditional entropy [28] equal to 0. Hence READ (nor any other algorithm) can acquire any knowledge from analyzing the sequence of messages, independently of its length. It is possible to conclude that in this worst, degenerate case all reverse-engineering algorithms will never converge to a correct solution. An example of this phenomena is included in Section 6.1.2 and shown in Figure 6.7c.

On the other hand, in the best case READ can converge to optimal results with a very low number of messages. For the correct extraction and labeling of *Physical* signals, READ requires only two consecutive values in which just the least significant bit flips while all other bits remain constant, independently of the signal size. For *CRCs* READ needs to analyze only two consecutive values in which all bits flip, independently of the signal size. For a *Counter* of size c bits, in the best scenario READ needs to analyze $2^c/2 + 1$ consecutive values. As an example, let us consider the realistic case of 4-bit counters. READ can properly extract and label these signals by observing 9 consecutive values that range from 0 to 8. Hence, in this optimistic scenarios it is possible to conclude that the theoretical lower bound to the number of messages that READ has to analyze to converge to correct results is 9.

Both the worst and the best case scenarios are not representative of real READ use cases. An evaluation of the convergence requirements in an “average” scenario would require a-priori and arbitrary assumptions on the evolution over time of signal values, that in reality are influenced by many imponderable and unforeseeable factors such as driving path, traffic conditions and driving style. Hence, an experimental evaluation of the correctness and convergence requirements over real data are provided in Sections 6.1.2.

6.1.2 Performance evaluation

This section evaluates the performance of READ on two different datasets. In both cases, the output generated by READ and by an implementation of the algorithm already proposed in [57] are compared with respect to the ground truth represented by formal specification of CAN messages. For a fair comparison of execution times, both algorithms are implemented in Python, run on the same hardware and analyze the same input CAN traffic traces.

The two algorithms have been evaluated according to three key performance indicators:

- *Correctness of signal extraction and labeling:* this performance indicator measures the number of signals that the algorithm is able to correctly extract and label;

- *Execution time:* the computational costs of the two algorithms are evaluated in terms of their execution time;
- *Convergence requirements:* the number of messages that the two algorithms need to analyze before achieving their best signal extraction and labeling performance.

Datasets description

The two datasets used on the experimental evaluation are called the *synthetic* dataset and the *real* dataset.

The former dataset is composed by synthetic messages generated using manually reverse-engineered boundaries publicly available online as reference. The latter dataset includes real CAN messages recorded from an unmodified licensed vehicle together with their correct formal specifications provided by an undisclosed industrial partner.

The *synthetic* dataset includes CAN messages that are generated from an algorithm. To guarantee a high level of realism despite the lack of public formal specifications, the reference signal boundaries of the Acura ILX 2016 CAN messages available online [17] and manually reverse-engineered by researchers of Comma.AI [18] have been used. It is necessary to remark that these reverse engineering results are partial, and large portions of the payload have not been analyzed for many CAN IDs. For the required evaluation three of the CAN IDs for which the reverse-engineered process led to the definition of almost complete message specifications are selected. The selected messages have IDs **158** (Powertrain data), **1D0** (Wheel Speeds) and **201** (Gas Sensor). A description of the signal boundaries used for the generation of the *synthetic* dataset is given in Tables 6.1, 6.2 and 6.3, for messages 158, $1D0$ and 201, respectively.

The final *synthetic* dataset is composed by $1,500,000$ messages, $500,000$ for each of the three IDs.

The *real* dataset is composed by 25 different CAN traffic traces. These traces have been collected from the same unmodified licensed vehicle under different driving conditions on real roads and subject to real traffic conditions. The complete dataset represents more than 14 hours of data and more than 125 millions of CAN data frames. The longest trace lasts more than 38 minutes, while the shortest is just 32 minutes long. A quantitative description of the *real* data set is given in Table 6.4, that summarizes the number of messages for each trace and its duration.

Table 6.1: Signals included in the message ID 158

Name	Type	Length (bit)
XMISSION_SPEED	Physical Value	16
ENGINE_RPM	Physical Value	16
XMISSION_SPEED2	Physical Value	16
ODOMETER	Physical Value	8
unlabeled	N/A (constant)	2
COUNTER	Counter	2
CHECKSUM	CRC	4
Total		64

Table 6.2: Signals included in the message ID $1D0$

Name	Type	Length (bit)
WHEEL_SPEED_FL	Physical Value	15
WHEEL_SPEED_FR	Physical Value	15
WHEEL_SPEED_RL	Physical Value	15
WHEEL_SPEED_RR	Physical Value	15
CHECKSUM	CRC	4
Total		64

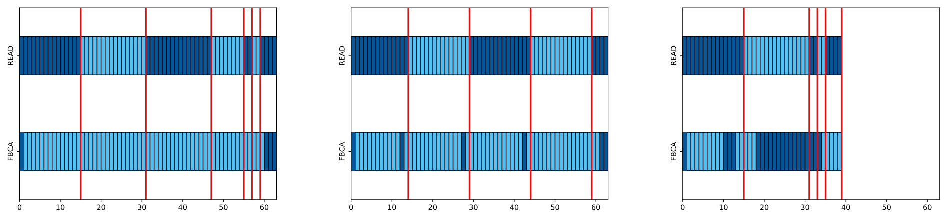
Signal extraction and labeling

The main performance indicator of READ is the total number of signals correctly extracted from the CAN traffic traces included in the *synthetic* and *real* datasets.

Signal extraction results of both READ and the algorithm described in [57] (labeled as FBCA) for the *synthetic* dataset are represented in Figure 6.4. In all the three charts of Figure 6.4, the x-axis represents the single bits of the payload, ranging from 1 to n (with n being the size of the payload), while the y-axis represents the outputs for both READ (top) and FBCA (bottom). Vertical rows represent the *correct* boundaries of the signals as identified by the message specifications of Tables 6.1, 6.2 and 6.3. To better identify the signal boundaries extracted by the two algorithms, consecutive signals are colored with different shades. Signal extraction is correct if vertical rows overlap color changes that identify the boundaries of consecutive signals as computed by READ and FBCA. Figures 6.4a, 6.4b and 6.4c represent the signal boundaries that have been reverse engineered for IDs 158, $1D0$ and 201, respectively. Since the payload of ID 201 is only 40 bits long, a white

Table 6.3: Signals included in the message ID 201

Name	Type	Length (bit)
INTERCEPTOR_GAS	Physical Value	16
INTERCEPTOR_GAS2	Physical Value	16
unlabeled	N/A (constant)	2
COUNTER	Counter	2
CHECKSUM	CRC	4
Total		40



(a) Boundaries extracted for ID 158 (b) Boundaries extracted for ID 1D0 (c) Boundaries extracted for ID 201

Figure 6.4: Signal boundaries extracted from the synthetic dataset by READ and FBCA

tail is used to represent the trailing unused bits of the payload.

From the analysis of Figure 6.4, it is clear that READ is able to correctly extract all the signals of the *synthetic* dataset. On the other hand, none of the boundaries identified by FBCA on this dataset is correct.

Another relevant metric is the number of labels that the READ algorithm correctly associates to the extracted messages. It is necessary to highlight that READ and FBCA use two different sets of labels: READ labels signals as *Physical*, *Counter* and *CRC*, while FBCA labels signals as *Constant*, *Multi-Value* and *Counter/Sensor*. This difference prevents a direct comparison among the two algorithms. Moreover, while READ produces labels that imply a different field semantic, the labels produced by FBCA are only representative of the signal variability over time, and do not attempt to describe their meaning. As a result, only the labels produced by READ are comparable with respect to the ground truth represented by the formal specifications of CAN messages. Moreover, FBCA did not manage to correctly extract any signal from the synthetic dataset. Hence, we only evaluate labeling correctness for signals correctly extracted by READ. For the ID 158 (Table 6.1)

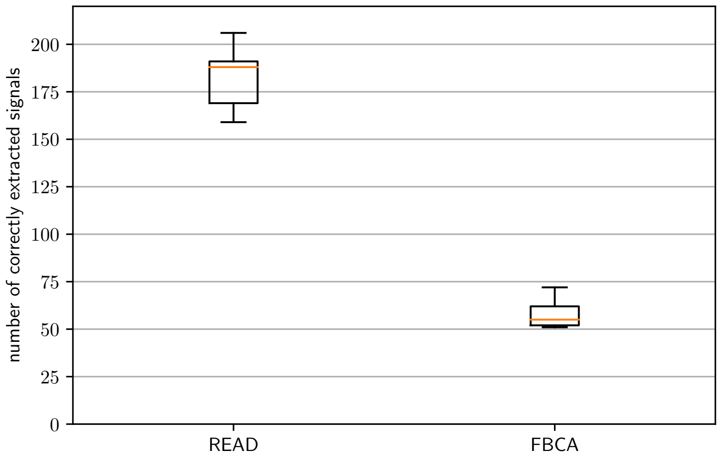


Figure 6.5: Comparison of the number of signals correctly extracted by READ and FBCA from the analysis of the *real* dataset

READ correctly labels the first four signals as *Physical*, the sixth as *Counter* and the seventh as *CRC*. For the ID $1D0$ (Table 6.2) READ correctly labels the first four signals as *Physical* and the fifth signal as *CRC*. For the ID 201 (Table 6.3) READ correctly labels the first two signals as *Physical*, the fourth as *Counter* and the fifth as *CRC*. Since this evaluation is based on incomplete specifications it is not possible to verify the labeling of the fifth signal of ID 158 and of the third signal of ID 201.

Concerning the *real* dataset, signal extraction performance are shown in Figure 6.5. The two box plots represent the number of signals that are correctly identified by READ and FBCA across 25 experiments carried out over the available traffic traces (see Table 6.4).

From Figure 6.5 it is possible to notice that the overall number of correctly identified signals varies among different traces for both READ and FBCA. The main reasons behind this variability are the different number of CAN messages included within each trace and the different values that the same messages assumed in different traces. READ manages to extract a number of signals that ranges from 159 to 206 with a median of 188, whereas FBCA correctly identifies a number of signals ranging from 51 to 72, with a median of 55.

By comparing results of the two algorithms on the same traffic trace it is possible to observe that in all the 25 experiments READ correctly identifies more than thrice the number of signals detected by FBCA. Comparing the

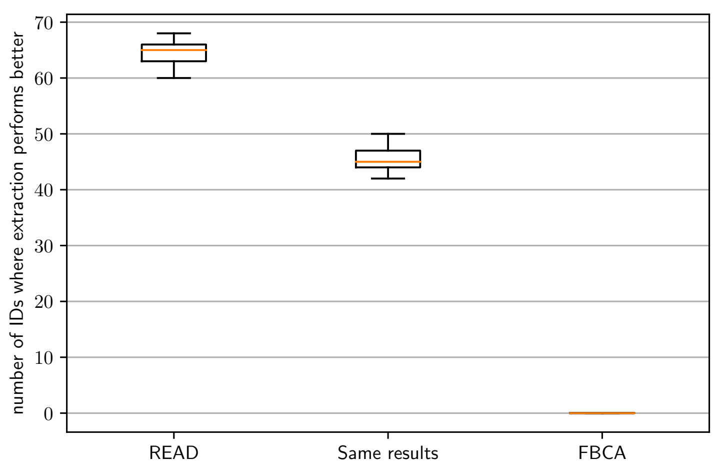


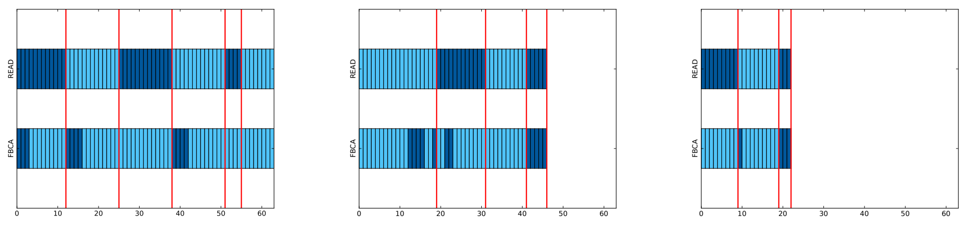
Figure 6.6: Number of IDs for which one of the two algorithms (READ and FBCA) outperforms the other one in the *real* dataset.

worst-case experiment for READ with the best case of FBCA, READ is always able to recognize more than twice the number of signals.

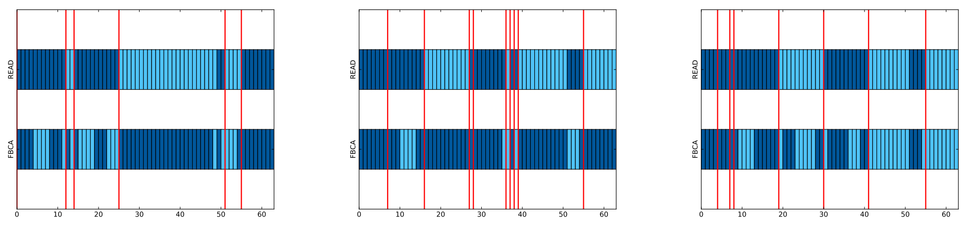
Besides counting the aggregate number of extracted messages, an evaluation on the performance of the two algorithms on different message IDs is made. This analysis is motivated by the fact that payloads of messages associated to different IDs can have a completely different structure in terms of number, position and types of embedded signals.

Results of this analysis are summarized in Figure 6.6. The first box plot represents the number of IDs for which READ extracts more correct signals than FBCA, the second box plot represents the number of IDs for which READ and FBCA extracts the same number of correct signals, while the third box plot represents the number of IDs for which FBCA extracts more correct signals than READ.

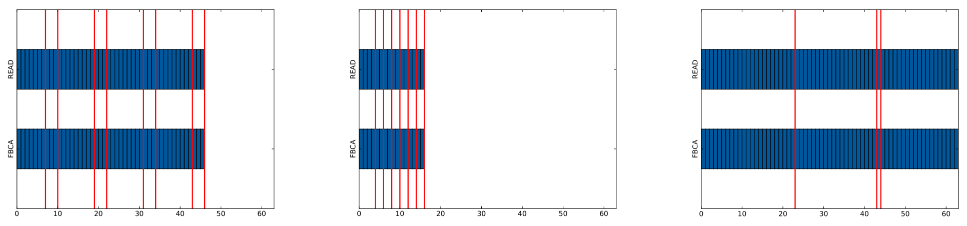
From Figure 6.6 it is possible to observe that READ is always able to match the results of FBCA, and to outperform it for the majority of IDs. Depending on the traffic traces, the two algorithms correctly extract the same number of signals for a number of IDs that ranges from 42 to 50. On the other hand, the number of IDs for which READ extracts more correct signals than FBCA ranges from 60 to 68. It is necessary to highlight that FBCA never achieves better performance than READ. To provide a better understanding of the performance achieved by READ and FBCA a graphical representation of the outputs of both algorithms is shown in Figure 6.7.



(a) Message IDs for which READ is able to correctly extract all signals



(b) Message IDs for which READ is not able to correctly extract all signals



(c) Message IDs for which READ is not able to correctly extract a signal

Figure 6.7: Representative examples of signal boundaries extracted by READ and FBCA, compared with the ground truth for the *real* dataset.

In all the nine charts of Figure 6.7, the x-axis represents the single bits of the payload, ranging from 1 to n (with n being the size of the payload for that particular ID), while the y-axis represents outputs of both READ (top) and FBCA (bottom). Vertical rows represent the correct boundaries of the signals as mandated by the formal specifications. To highlight the signal boundaries extracted by the two algorithms, consecutive signals are colored with different shades. Signal extraction is correct if vertical rows overlap color changes that identify the boundaries of consecutive signals as computed by READ and FBCA. Since for some payloads the number of bits n is lower than 64, a white tail is used to represents the trailing unused bits.

Figures are grouped in three different rows based on the effectiveness of READ in extracting correct signal boundaries. Row 6.7a includes three representative examples of message IDs for which READ correctly extracts all signals. For these three different message IDs the signals extracted by FBCA have wrong boundaries. It is interesting to notice that many errors in FBCA's output are caused by the incorrect identification of spurious signals that do not exist in the formal specifications. Row 6.7b shows three representative examples of message IDs for which both READ and FBCA are not entirely accurate. In particular, in all the three message IDs it is clear that both algorithms exhibit sub-optimal performance in the extraction of small signals. Those signals usually convey information that is unrelated to physical phenomena, and are mainly composed by bit masks used for sensing and controlling the state of specific functionality of the vehicle, such as fan speed and air conditioning. Finally, row 6.7c shows three representative examples of real message IDs for which both READ and FBCA are not able to extract a single correct signal. This happens for the message IDs in which payloads never change over the collected traffic traces (bit-flip rate is always equal to 0 for all bits). This final result shows that the quality of the collected data impacts the quality of results generated by both algorithms. However it should be noted that these messages do not convey safety-critical information related to the vehicle dynamics, and are mostly related to optional features.

An interesting aspect to notice is that Figure 6.7 does not contain any example in which FBCA extracts all signals from an ID. This is due to the fact that FBCA has never been able to achieve full accuracy for any of the IDs included in the tested traffic traces. Besides signal extraction, an evaluation of the correctness of signal labeling is also made. This evaluation has only been performed on signals for which READ managed to correctly identify the boundaries, since it cannot be performed for FBCA due to the different labels that only describe the signal evolution over time and do not have any semantic meaning. Results of this analysis are summarized in Figure 6.8. The three box plots of this figure refer to signals that where correctly labeled as

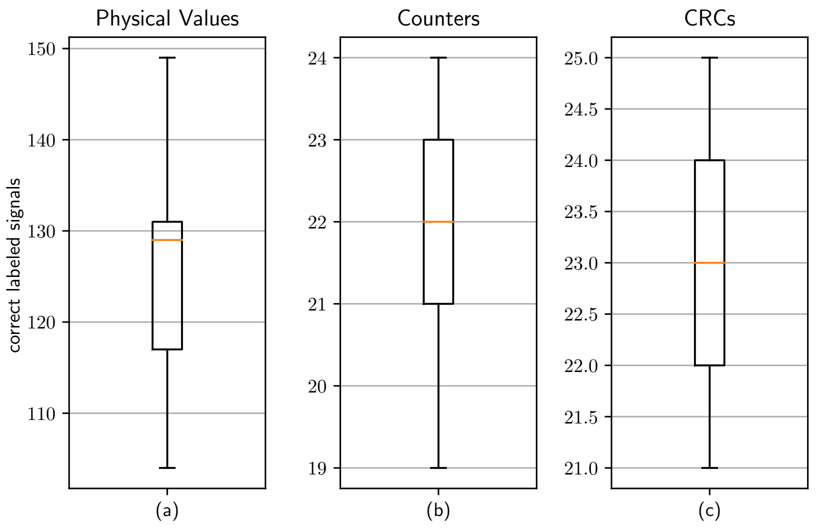


Figure 6.8: Signals that are correctly extracted and correctly labeled by READ

Physical values (Figure 6.8a), Counters (Figure 6.8b) and CRCs (Figure 6.8c). In all three box plots the y-axis represents the number of correctly labeled signals. Please note that for better readability the three y-axis use a different scale.

Figure 6.8 shows that the number of Counter and CRC signals that are correctly labeled by READ exhibit a small variability among all the different traces. This implies that the heuristics used to recognize these fields lead to stable results. Moreover, results shown in Figure 6.8 highlights that the correct labeling of signals representing Physical values exhibit a higher variability among the analyzed traffic traces, ranging from 104 in the worst case to 149 in the best case, with a median of 129. This variability is motivated by the different driving styles, road conditions and lengths associated to each traffic trace. Signal labeling performance have been further analyzed and the results have been summarized in Table 6.5, which compares the number of signals that were correctly extracted and labeled by READ with respect to the vehicle specifications in the best and worst case scenarios.

Table 6.5 shows that, even in the worst scenario, READ correctly labels more than the 90% of the extracted Physical signals, while in the best case it is able to correctly label more than 98% of them. Results for both Counter and CRC labels are exactly the same in both cases (85.71% and 96.00% correct labeling percentage, respectively), meaning that the heuristics used for labeling those particular signals are strong and consistent. With those

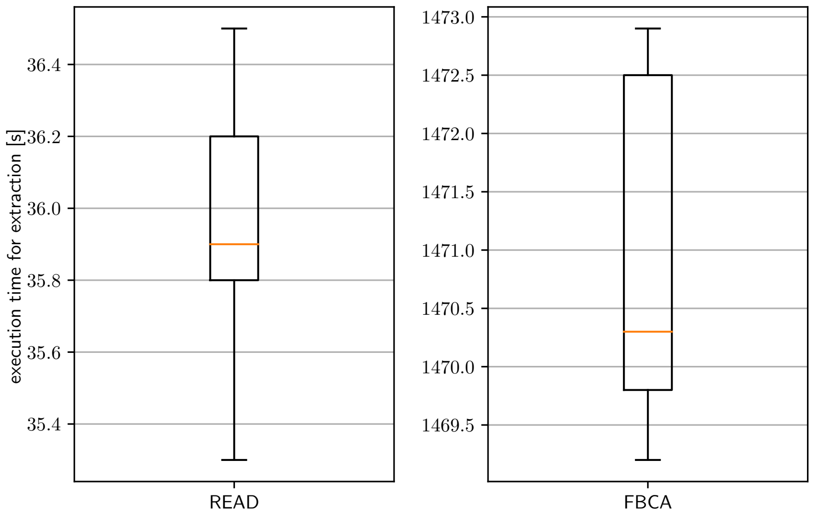


Figure 6.9: Execution time of READ and FBCA over all CAN traffic traces

final considerations in mind, it is possible to notice that READ is better with respect to the state of the art [56] at identifying signals within the payload of CAN data frames by determining their exact boundaries. Moreover, for all messages extracted correctly, READ performs labeling with a very high accuracy. These results hold for both the *synthetic* and the *real* datasets.

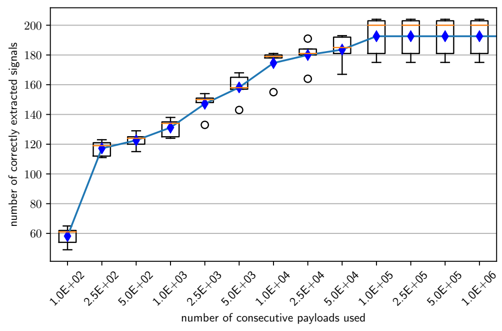
Execution time

The time required to execute both algorithms on the traffic traces of the *real* dataset (see Table 6.4) are compared. These evaluations are carried out on a server equipped with an Intel® Core™ i7–7700HQ CPU @3.8 GHz and with 16 GB of RAM running Fedora 24 $x64$.

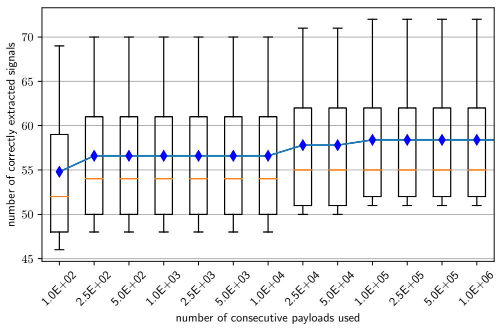
For this experiment, the *running time* is evaluated as the time an algorithm requires to generate its final output starting from a raw CAN bus traffic log. Both algorithms have been implemented in the Python programming language and leverage the same boilerplate code for low-level operations (such as reading files from memory and parsing the fields of a CAN message). Execution times of both READ and FBCA are shown in Figure 6.9, where the two box plots represent the time (expressed in seconds) needed for the complete signal extraction and labeling over the 25 traffic traces by the two different implementations. Please note that for the sake of readability the two box plots refer to a different scale on the y-axis.

The execution times of READ are two orders of magnitude lower with

Figure 6.10: Convergence of READ and FBCA



(a) READ Convergence



(b) FBCA Convergence

respect to the execution times of FBCA. The minimum time required for the complete extraction of FBCA is 1469.2 seconds (24 minutes) for a traffic trace of 5.5 million messages, equivalent to approximately 38 minutes of driving. On the other hand, the maximum execution time of READ on the same traffic traces equals to 36.5 seconds.

Convergence requirement

The final metric evaluated for the comparison of these two algorithm is the number of consecutive messages they need to analyze to produce their best results. Since both algorithms learn their message boundaries by observing the evolution of payloads over time, it is important to provide a sufficient number of messages in the training process to achieve good and stable classification results. To perform an unbiased and realistic evaluation these experiments refer to the *real* dataset. This number is evaluated by extracting the first N messages for each ID from the available CAN traffic traces, applying the extraction algorithms to them and evaluating the number of correctly extracted signals. The convergence results for READ and FBCA are shown in Figure 6.10. Figure 6.10a and Figure 6.10b represent convergence results of READ and FBCA, respectively. In both figures the x-axis represents the number N of consecutive payloads analyzed by the reverse engineering algorithm and the y-axis represents the number of correctly extracted signals through the gathered traces. Please note that, for the sake of readability, the y-axis of Figures 6.10a and 6.10b refer to different scales.

Results show that both READ and FBCA converge to their best results with $100,000$ or more consecutive payloads. The authors of FBCA claim that their algorithm only requires about 100 messages for each ID in order to produce stable results, and this experimental evaluation verified that FBCA is capable to achieve near-optimal results after the analysis of 100 messages. However, to achieve its peak performance over real CAN traffic, the FBCA algorithm requires at least $100,000$ messages for each ID. Moreover, even with as few as 100 messages in the training set, READ is able to correctly extract a median of 61 correct messages, with respect to the 52 extracted with FBCA. To summarize, the experimental evaluation shows that READ achieve better performance than FBCA, with lower execution times and comparable convergence requirements.

6.2 Analysis of signals extracted with READ

This Section provides an example of the different signals extracted with the READ algorithm. The signals are grouped according to the labels assigned by READ: Counter, CRC, or Physical signals. Moreover, an inspection of the variability over time of the signals classified as "Physical signals" by READ is provided, proposing two different detection metrics based on two different typologies of signals.

6.2.1 Counter inspection

For the inspection of a counter-labeled signal an analysis of the values of the signal evolution over time is provided. Figure 6.11 shows one of the signals labeled as a Counter by READ, where the value of the counter (*y-axis*) is compared with its evolution over time (*x-axis*). The depicted counter signal is a 4 bit counter, with a minimum and maximum values of 0 and 15, respectively.

From the inspection of Figure 6.11 it is possible to notice that the counter signal follows the expected trend: the next value of the sequence is the previous value plus 1, with an overflow taking place after the value 15 since the next value requires at least one extra bit for its encoding. An anomaly detector focused on the inspection of the counter of the messages only requires to check that the counter value follows the trend described previously and depicted in Figure 6.11. If the counter signal encoded in a message does not follow the expected trend, it is possible to flag it as an anomaly.

6.2.2 CRC inspection

For the inspection of a CRC-labeled signal an analysis of the values of the bits composing the CRC over time is provided. Figure 6.12 shows the bit-flip

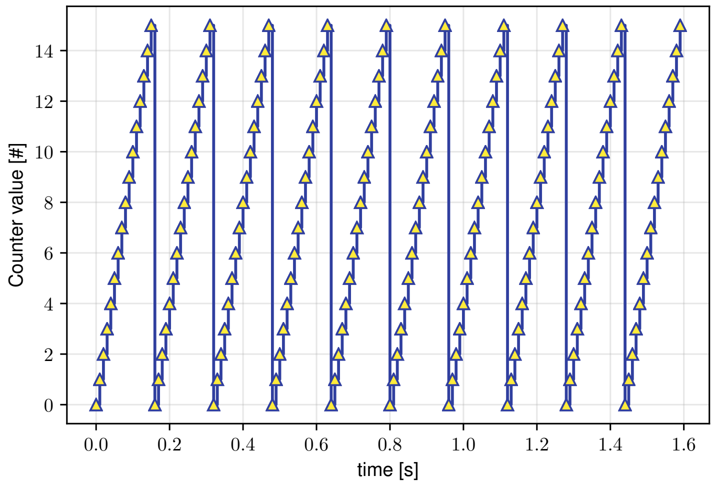


Figure 6.11: Counter signal value extracted with READ

probability (*y-axis*) over time for each bit of the signal (depicted on the *x-axis*) for one of the signals labeled as a CRC by READ. From the inspection of Figure 6.12 it is possible to notice that the values of the bit-flip probability for each bit composing the signal is centered around 0.5 with little deviation. As already depicted for many detection methods in Chapter 5, it is possible to leverage this behavior of the bit-flip probability to define the normal values as the values in the range $[\mu - k * \sigma, \mu + k * \sigma]$, in which the evaluated values of μ and σ are equal to 0.5 and 0.01, respectively. In Figure 6.12 the boundaries are defined with a value of $k = 3$, since the distribution of the bits of the CRC signals follows the normal distribution. An anomaly in any of the bits is detected if its bit-flip probability is evaluated to be outside of the normal range. However, for this type of analysis it is necessary to define the size of the window on which evaluate the bit-flip probability periodically.

6.2.3 Physical signals inspection

Two different typologies of signals labeled by READ are inspected in this Section. Each signal type depicts a different evolution over time, for which an anomaly detection algorithm is proposed.

Step signal

The evolution over time of the step signal follows a discrete step function, meaning that the next value of the signal is either the previous value or in

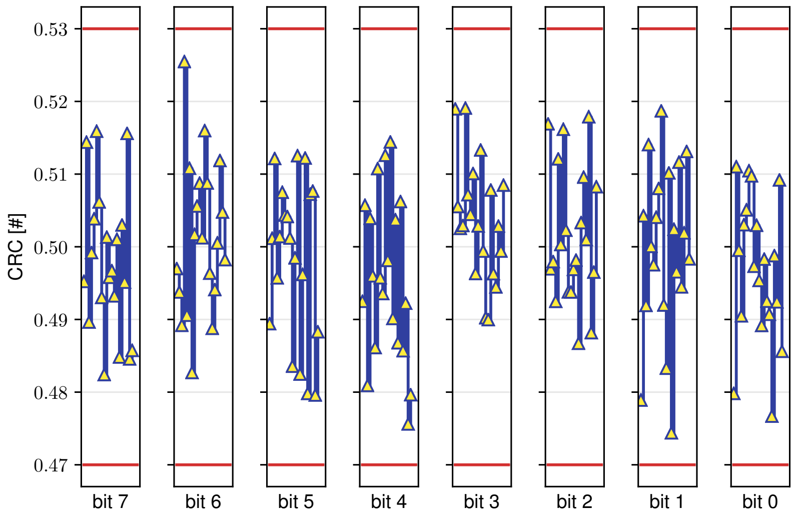


Figure 6.12: CRC signal value extracted with READ

within a fixed offset. In the example depicted in Figure 6.13, the next value of the signal is the previous value plus $-1, 0$, or 1. From a manual analysis of the signal depicted in Figure 6.13 it was possible to recognize the signal as the value of the gear used by the vehicle (which has an automatic gear mechanism). Signals like the one depicted in Figure 6.13 follow a particular trend, thus it is possible to leverage this trend for the detection of anomalies on this typology of signals. Since each value is the previous plus a value in a defined range, after the identification of the valid range (in this case $[-1, +1]$), it is possible to detect anomalies by checking the value of the signal. If the difference of the current signal value with its previous value is outside of the valid range, an anomaly is raised.

Generic signal

The evolution over time of a generic signal does not follows a particular trend, thus it is extremely hard to define a generic rule for the definition of the normal behavior of this kind of signals without inspecting its references in the DBC. Figure 6.14 depicts the evolution over time (x-*axis*) of the *vehicle speed* signal extracted using the references encoded in the DBC. The vehicle speed signal is encoded in a signal with a length of 16 bits, thus with a maximum encodable value of 8191. The specifications for this signal define the maximum raw value equals to 5440, the scaling of the signal is 0.0625, and the measurement unit as Km/h. Figure 6.14 shows the signal value decoded

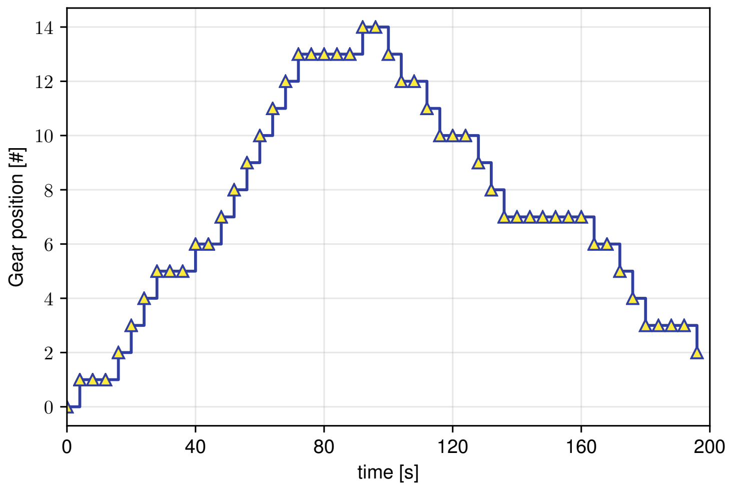


Figure 6.13: Step signal value extracted with READ

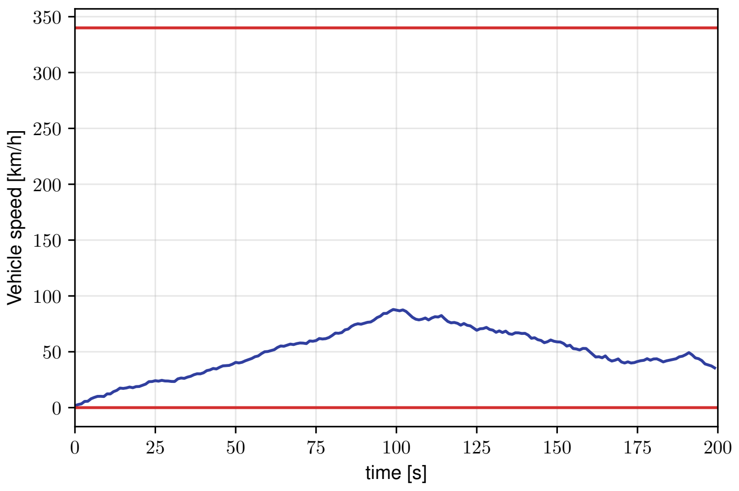


Figure 6.14: Generic signal value extracted with READ

by applying the scale to the raw values. The two horizontal lines depict the minimum and maximum value of the signal. Using the specifications encoded in the DBC file, it is possible to define multiple features to inspect for the detection of anomalies. The first feature is to check that the raw signal value is not outside of its boundaries. As an example, the vehicle speed signal depicted in Figure 6.14 has a maximum raw value of 5440, meaning that any value from 5441 to 8191 (which is its real maximum due to the size of the signal) is to be considered anomalous. Another possible feature to inspect is the difference between consecutive values of the signal. Since this signal represents the vehicle speed expressed in Km/h, it is possible to apply detection metric similar to the ones inspected in case of the *step signal*, checking that the difference between consecutive readings of the vehicle speed are withing a fixed offset, thus detecting sudden increases/decreases of the vehicle speed as anomalies.

Table 6.4: Number of CAN messages and duration of the 25 CAN traffic traces that compose the *real* dataset

Trace	Number of CAN Messages	Duration (HH:MM:ss)
1	4,646,605	00:32:25
2	4,715,886	00:32:54
3	4,759,489	00:33:11
4	4,589,270	00:32:01
5	4,959,564	00:34:36
6	4,909,393	00:34:15
7	5,532,928	00:38:36
8	5,224,743	00:36:27
9	5,444,530	00:37:59
10	5,598,760	00:38:51
11	4,634,650	00:32:20
12	5,348,991	00:37:19
13	4,558,212	00:31:48
14	4,993,051	00:34:50
15	5,262,967	00:36:43
16	5,145,906	00:35:54
17	5,045,538	00:35:12
18	4,617,935	00:32:13
19	4,942,846	00:34:29
20	4,866,383	00:33:57
21	5,294,014	00:36:56
22	5,497,079	00:38:21
23	5,253,401	00:36:39
24	5,315,515	00:37:05
25	5,399,129	00:37:40
Total	126,523,785	14:42:41

Table 6.5: Correct labeling of signals extracted from the *real* dataset

	Label	Extracted	Correctly labeled	Percentage
Worst Case	Physical	115	104	90.43%
	Counter	28	19	85.71%
	CRC	25	21	96.00%
Best Case	Physical	151	149	98.01%
	Counter	28	24	85.71%
	CRC	25	25	96.00%

Chapter 7

Anomaly reaction based on control-based approaches

Since the introduction of microcontrollers in modern vehicles, automotive industries began to deploy novel features to the vehicle, aiming to increase the physical safety of all the passengers and to enhance the driving experience. These microcontrollers are electronic devices connected with sensors and actuators to the mechanical parts of the vehicle, that are programmed to react to the different driving situations to increase the safety of the vehicle, thus limiting potential injuries and road fatalities. These microcontrollers are usually deployed in different parts of the vehicle, and are connected to one or more sensors to collect data from the inspected mechanical system. Some of these microcontrollers are also connected to actuators, allowing the microcontroller to actively control the desired mechanical system. These microcontrollers allow to deploy electronic-controlled features on the vehicle, ranging from simple functions (such as the power windows or the power seats), to more complex features such as the anti-braking system (ABS), the engine control, or the electronic stability control (ESC). Although simple feature require simple microcontrollers for their activation, more complex and safety-related features require microcontrollers able to compute data generated by heterogeneous sensors to drive actuators in real-time, that in most of the cases require complete autonomy and do not allow for active intervention from the driver. These microcontrollers are connected with each other via a different communication network, including the CAN bus which has already been depicted in Section 2.2 and its security issues have been described in Section 2.3. The security issues described in Section 2.3 are a consequence of the *drive-by-wire* capabilities of modern vehicles, where the driving system (composed by the engine, brakes, and steer) is controlled by the values of the signals encoded in CAN messages. One of the first features implemented in modern vehicles that provided drive-by-wire capabilities is the cruise control, which drivers usually activate when road and traffic conditions allow to proceed at a fixed-speed, for more efficient fuel consumption and driver comfort. Since the combination of drive-by-wire capabilities and internet connectivity without proper security countermeasures has severe consequences on the safety of passengers and road users [45, 59], many security researchers already proposed different detection algorithms that inspect the data and the characteristics of the microcontrollers to detect anomalies in the CAN bus [46, 64]. The aforementioned detection techniques identify anomalies on data transmitted over the communication network without inspecting the associated information, thus preventing any possible detection in the case the attacker is able to send malicious data by pretending to be a valid component of the system [70, 86]. To provide more information about the context of the attack and to preventing them from being unnoticed, a combination of the detection methodologies from Computer Science suited for the automotive networks with cyber security solutions developed in the research field of the control theory are inspected in this Chapter.

7.1 Control System

To provide a solution of the cruise control problem in a generic vehicle, it is necessary to describe the vehicle system in a generic form:

$$\dot{x}(t) = f(x(t), u(t)) \tag{7.1}$$

$$y(t) = Cx(t) \tag{7.2}$$

where $x \in \mathbb{R}^n$ is the state space, $u, y \in \mathbb{R}$ are the input and the measured output [33]. Since the car speed is required to be fixed at a desired value, it is common to study (7.1) around an equilibrium point x_{eq} and input u_{eq}. Hence,(7.1) can be considered linear:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{7.3}$$

where

$$A \triangleq \left. \frac{\partial f(x,u)}{\partial x} \right|_{\substack{x = x_{eq} \\ u = u_{eq}}} \qquad B \triangleq \left. \frac{\partial f(x,u)}{\partial u} \right|_{\substack{x = x_{eq} \\ u = u_{eq}}} \tag{7.4}$$

For analysis and controller design purposes it is also useful the transfer function representation in the Laplace's domain:

$$y(s) = F(s)u(s) \tag{7.5}$$

where $F(s) = C(sI - A)^{-1}B$. In this context a proportional-integral-derivative (PID) controller is used:

$$u(s) = K\left(1 + \frac{1}{sT_i} + \cdot sT_d\right)e(s). \tag{7.6}$$

where K, T_i and T_d are the proportional gain, integral time and derivative time respectively. For the real implementation, Equation (7.6) is digitalized.

7.2 Powertrain Model

This Section presents the equations for modeling the powertrain and the controller modules used for simulating a generic sparkle-ignited engine and defines the content of the related CAN messages used in the simulated network.

7.2.1 Engine and Controller Models

The internal combustion (IC) engine has been modeled as a mean-value model (MVMs) [33], in which the sparkle-ignited (SI) engine is described by the models of eight interconnected subsystems. To solve cruise-control problem the main blocks considered for the composition of the SI engine are the throttle body, the intake manifold, the gas exchange, the combustion and torque generation, and the engine inertia. The nonlinear form of the model is described as:

$$\frac{dp_m(t)}{dt} = \frac{R\theta_m}{V_d}(\dot{m}_\alpha(t) - \dot{m}_\beta(t)) \tag{7.7}$$

$$\frac{d\omega_e(t)}{dt} = \frac{1}{\theta_e}[T_e(t) - T_l(t)] \tag{7.8}$$

where p_m is the intake manifold pressure, $\dot{m}_\alpha$ is the air-flow mass entering from the throttle valve, $\dot{m}_\beta$ is the air-flow mass going out from the intake manifold to the cylinders, w_e is the engine speed, T_e the torque generated by the engine, and T_l the load torque. Assuming that the air is a perfect gas and the throttle is isenthalpic, $\dot{m}_\alpha$ is given by:

$$\dot{m}_\alpha(t) = \begin{cases} A_\alpha(t)\frac{p_a}{\sqrt{R\theta_a}}\frac{1}{\sqrt{2}} & \frac{p_m(t)}{p_a} \leqslant 0.5 \\ A_\alpha(t)\frac{p_a}{\sqrt{R\theta_a}}\sqrt{\frac{p_m(t)}{p_a}[1 - \frac{p_m(t)}{p_a}]} & else \end{cases} \tag{7.9}$$

where $\dot{m}_{out} = \dot{m}_\alpha$ and $p_{out} = p_m$ is the intake manifold pressure.
The throttle valve open area A_α is computed as follows:

$$A_\alpha(\alpha_{th}) = \frac{\pi d_{th}^2}{4}(\frac{\cos(\alpha_{th})}{\cos(\alpha_{th,0})}) + A_{th,leak} \tag{7.10}$$

$$\alpha_{th} = \alpha_{th,0} + (\frac{\pi}{2} - \alpha_{th,0}) \cdot u_\alpha \tag{7.11}$$

where α_{th} is the throttle angle, $u_\alpha \in [0,1]$ is the control input. In (7.10) and (7.11) it is assumed that the throttle actuation is neglected.

Neglecting the wall-wetting phenomena and the injectors dynamics,

$$\dot{m}_\beta(t) = \frac{\dot{m}(t)}{1 + \frac{1}{\lambda\sigma_0}} \tag{7.12}$$

where the air/fuel ratio is assumed constant and λ_l is the volumetric efficiency. Then the air/fuel ratio $\lambda(t)$ can be use to describe the fuel and air mass flows ratio into the cylinder respect to the constant stoichiometric ratio σ_0:

$$\lambda(t) = \frac{1}{\sigma_0} \cdot \frac{\dot{m}_\beta(t)}{\dot{m}_\phi(t)} \tag{7.13}$$

then, under the hypothesis that $\lambda(t)$ is known, (e.g $\lambda(t) \approx \lambda$):

$$\dot{m}_\phi = \frac{\dot{m}_\beta(t)}{\sigma_0 \lambda(t)} \tag{7.14}$$

which represents the mean value fuel mass flow. Then $\dot{m}(t) = \dot{m}_\beta + \dot{m}_\phi$ represents the gas-mixture mass flow aspired in the cylinder which is given by:

$$\dot{m}(t) \quad = \quad \frac{p_m(t)}{R\theta_m} \cdot \lambda_l(\omega_e(t), p_m(t)) \cdot V_d \cdot \frac{\omega_e(t)}{4\pi} \tag{7.15}$$

The volumetric efficiency λ_l describes how far the engine differs from a perfect volumetric device that can be approximated as:

$$\lambda_l(\omega_e, p_m) = \lambda_{l\omega}(\omega_e) \cdot \lambda_{lp}(p_m) \tag{7.16}$$

where

$$\lambda_{l\omega}(\omega_e) \quad = \quad \gamma_0 + \gamma_1\omega_e + \gamma_2\omega_e^2 \tag{7.17}$$

$$\lambda_{lp}(p_m) \quad = \quad \frac{V_c + V_d}{V_d} - \frac{V_c}{V_d}\left(\frac{p_{out}}{p_m}\right)^{\frac{1}{\kappa}} \tag{7.18}$$

V_c is the compression volume, V_d is the volume displacement, κ is the ratio of the specific heat, $\kappa \approx 1.4$, p_{out} is the pressure at the engine's exhaust side.

Note that, (7.14) is also related to the engine speed w_e

$$\dot{m}_\phi(t) = m_\phi(t)\frac{w_e(t)}{4\pi}, \tag{7.19}$$

thanks to (7.14) and (7.19) T_e is related to $\dot{m}_\phi$. The break mean effective pressure is given by:

$$p_{me} = \frac{T_e \cdot 4\pi}{V_d}. \tag{7.20}$$

and the fuel mean effective pressure:

$$p_{m_\phi} = \frac{m_\phi \cdot H_f}{V_d}. \tag{7.21}$$

The effective efficiency is:

$$\eta_e = \frac{p_{me}}{p_{m_\phi}} = \frac{T_e \cdot 4\pi}{m_\phi \cdot H_f} \tag{7.22}$$

then:

$$p_{me} = \eta_e(\cdots) \cdot p_{m_\phi} \tag{7.23}$$

the value of $\eta_e(\cdots)$ can be evaluated in different ways, and a possible approximation uses the indicated mean pressure:

$$p_{me} \approx p_{mi}(\omega_e) - (p_{m0f}(\omega_e) + p_{m0g}(\omega_e)) \tag{7.24}$$

where:

$$p_{mi} = \frac{w_i}{V_d} = \eta_i \frac{m_\phi \cdot H_f}{V_d} \tag{7.25}$$

where the indicated thermodynamic efficiency $\eta_i(\omega_e) \approx \eta_0 + \eta_1 \omega_e$ refers to the Williams approximation. The coefficients p_{m0f} and p_{m0g} represent the loss due the friction and gas exchange. From (7.20), (7.23), and (7.24):

$$T_e(t) = p_{me}(t) \cdot \frac{V_d}{4\pi} = (\eta_i(\omega_e) p_{m_\phi} - p_{m0f}(\omega_e) - p_{m0g}(\omega_e)) \frac{V_d}{4\pi} \tag{7.26}$$

Using (7.22):

$$T_e(t) = ((\eta_0 + \eta_1 \omega_e) \frac{H_f \cdot m_\phi(t)}{V_d} - p_{m0f}(\omega_e) - p_{m0g}(\omega_e)) \frac{V_d}{4\pi} \tag{7.27}$$

From (7.19), $m_\phi \to \dot{m}_\phi$ and from (7.14):

$$m_\phi(t) = \frac{\dot{m}_\beta \cdot 4\pi}{\alpha \cdot \omega_e} \tag{7.28}$$

where $\alpha = \lambda \cdot \sigma_0$. Note that, gas-mixing transportation delays need to be considered in (7.14), $\dot{m}_\phi(t - \delta)$ and $\dot{m}_\beta(t - \delta)$, that are extremely important

$$\begin{aligned}\frac{dp_m(t)}{dt} &= \frac{R\theta_m}{V_d}\left(A_\alpha(t)\frac{p_a}{\sqrt{R\theta_a}}\frac{1}{\sqrt{(2)}} - \left(\frac{p_m(t)}{R\theta_m}\left(\gamma_0+\gamma_1\omega_e(t)+\gamma_2\omega_e^2(t)\right)\right.\right.\\ &\left.\left.\left(\frac{V_c+V_d}{V_d}-\frac{V_c}{V_d}\left(\frac{p_{out}}{p_m}\right)^{\frac{1}{\kappa}}\right)\frac{V_d\omega_e(t)}{4\pi}\frac{\alpha}{\alpha+1}\right)\right) \qquad (7.29)\\ \frac{d\omega_e(t)}{dt} &= \frac{1}{\theta_e}\left[\left((\eta_0+\eta_1\omega_e(t))\frac{H_f\cdot p_m(t)}{R\theta_m}\left(\gamma_0+\gamma_1\omega_e(t)+\gamma_2\omega_e^2(t)\right)\left(\frac{V_c+V_d}{V_d}-\frac{V_c}{V_d}\left(\frac{p_{out}}{p_m}\right)^{\frac{1}{\kappa}}\right)\right.\right.\\ &\left.\left.\cdot\frac{V_d}{\alpha+1}-\left(\beta_0+\beta_2\omega_e^2(t)+(p_{out}-p_m(t))\right)\frac{V_d}{4\pi}\right)-T_l(t)\right] \qquad (7.30)\end{aligned}$$

Figure 7.1: Nonlinear model of an SI-engine for cruise-control problem.

param	**value**	**units**	**param**	**value**	**units**
R	287	[J/KgK]	$\gamma 1$	3.42e-3	[s]
θ_a	298	[K]	$\gamma 2$	-7.7e-6	[s^2]
θ_m	340	[K]	$\eta 0$	0.16	[J/Kg]
α_{th0}	7.9	[deg]	$\eta 1$	2.21e-3	[Js/Kg]
d_{th}	58.7e-3	[m]	$\beta 0$	15.6	[Nm]
$A_{th,leak}$	5.6e-6	[m^2]	$\beta 2$	0.175e-3	[Nms^2]
Vd	2.77e-3	[m^3]	θ_e	0.2	[kg/m^2]
Vc	0.277e-3	[m^3]	H_f	45.8e6	[-]
p_a	1e5	[Pa]	$kappa$	1.35	[-]
p_{out}	1e5	[Pa]	α	14.70	[-]
$\gamma 0$	0.45	[-]			

Table 7.1: Values of the parameters used in the model

for the idle control. For the case discussed in this scenario, those delays can be neglected.

Assuming sonic condition and isothermal gas, (7.7) can be rewritten using (7.9) and (7.12). Neglecting transportation delays, (7.8) is rewritten using (7.27) and (7.28). See Figure 7.1 (7.29), and (7.30) respectively. The values of the parameters used in our model are provided in Table 7.1, while the meaning of these could be found in [33]

The state variables are p_m and w_e

$$x(t) \triangleq \begin{bmatrix} p_m(t) \\ w_e(t) \end{bmatrix} \tag{7.31}$$

For not overloading the representation of the final system, $A_\alpha(t)$ is considered as the system input, but using (7.10) and (7.11) it is possible consider directly u_α. The output of the system is w_e, which express the angular velocity of the engine. The considered cruise-control problem assumes that the vehicle speed is constant with the ratio between wheel speed and engine speed. This scenario is very plausible for a replay attack.

The equilibrium point for a given vehicle speed $\bar{v}$ is $\bar{x} = [\bar{p}_m, \bar{w}_e]^T$ and $\bar{u} = \bar{A}_\alpha$. These values allow to linearize the system following the procedure described in 7.1.

7.2.2 CAN message design

To propose a realistic model for the simulation of the powertrain system based on the equations previously described, it is necessary to define the messages used for data exchange over the simulated CAN bus. The following signals are selected for being included in the messages:

1. **Throttle request**: the throttle request signal is sent from the controller to the model and represents the throttle requested by the driver (i.e. by increasing or decreasing the push on the acceleration pedal);

2. **Engine speed**: the engine speed signal (the model output) is sent from the engine ECU to the cruise controller to compute the optimal throttle for keeping the speed constant;

3. **Controller reference**: the controller reference signal is sent from an ECU external to the inspected system and carries the reference speed value for the controller (i.e. the fixed speed requested by the driver upon activation of the cruise control).

Each message has a fixed payload length of 64 bits, which encodes the value of the corresponding signal as read from the sensor.

7.3 Threat Model

The adversary can physically or remotely compromise one or more ECUs via numerous attack surfaces and means, as already inspected in [11, 48]. Since the powertrain model simulated in this Chapter is created by implementing the different equations describing its subsystems, the threat model considered for the security analysis of the model uses a modified version of the threat model described in Section 2.3. The main difference between the threat model described in 2.3 and the threat model considered in this scenario is that only the attack scenario of the *message injection* attack (specifically only the single ID replay attack) is inspected. This different scope of applicability is explained by the analysis of the definition of the model itself: since the model relies on the messages sent from the different sensors for its functioning, removing the messages from the network would result in preventing the model from functioning, thus it is not possible to inspect the consequences of the message removal attack as a real-case attack scenario.

In the injection attack scenario the attacker is considered able to fabricate and send on the CAN bus semantically valid messages. The objective of this attack is to inject a maliciously forged message after the valid message is sent, to subvert the behavior of the vehicle model. Three different types of message injection attack are inspected:

1. **Inject Speed attack:** in this attack scenario a message with a fixed value of the engine rotational speed lower than the reference speed is injected;
2. **Inject Reference attack:** in this attack scenario a message with a higher fixed value of the reference used by the cruise controller is injected;
3. **Inject Throttle attack:** in this attack scenario a message with a fixed value of the requested throttle higher than the one generated by the cruise controller is injected.

The final goal of all the simulated injection attacks is to increase the speed of the engine's model. For simulation purposes, the *reference speed* is set at 4200 RPM.

7.4 Consequences of cyber-attacks

In this Section the consequences of the different cyber attacks described in Section 7.3 are evaluated. All the results are depicted by comparing the values of the signals sent on the CAN bus with the actual engine speed generated by the model. The attacks have a duration of 5 seconds and are triggered after 20 seconds of normal conditions. The consequences of the injection of messages on the model are inspected. The messages are injected after the valid one is sent on the network.

7.4.1 Injection speed attack

In case of an **injection speed attack** the injected message encodes an engine speed value equals to 4190 RPM, and the consequences of this attack on the powertrain model are depicted in Figure 7.2, in which the engine speed (y-axis, expressed in RPMs) is compared with the time (x-axis, expressed in seconds). The solid line of Figure 7.2 represents the engine speed signal extracted from its respective CAN message, while the dashed line represents the real engine speed generated by the model. The vertical red line highlights the start of the attack.

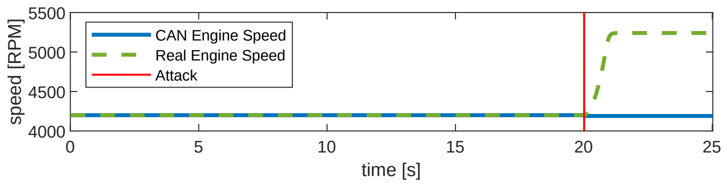


Figure 7.2: Injection engine speed attack

7.4.2 Injection reference attack

In case of an **injection reference attack** the injected message encodes a reference value equals to 4500 RPM, and the consequences of this attack on the powertrain model are depicted in Figure 7.4, in which the engine speed (y-axis, expressed in RPMs) is compared with the time (x-axis, expressed in seconds). The solid line of Figure 7.4 represents the engine speed signal extracted from its respective CAN message, while the dashed line represents the real engine speed generated by the model. The vertical red line highlights the start of the attack.

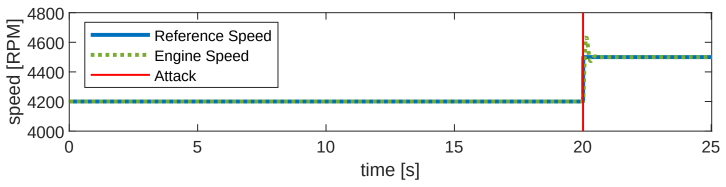


Figure 7.3: Injection reference attack

7.4.3 Injection throttle attack

In case of an **injection throttle attack** the injected message encodes a throttle value equals to $2.5 * 10^{-4}$, and the consequences of this attack on the powertrain model are depicted in Figure 7.3, in which both the throttle (left y-axis) and the engine speed (right y-axis, expressed in RPMs) signals are compared with the time (x-axis, expressed in seconds). The solid line of Figure 7.3 represents the value of the throttle signal extracted from its respective CAN message, while the dashed line represents the real engine speed generated by the model. The vertical red line highlights the start of the attack.

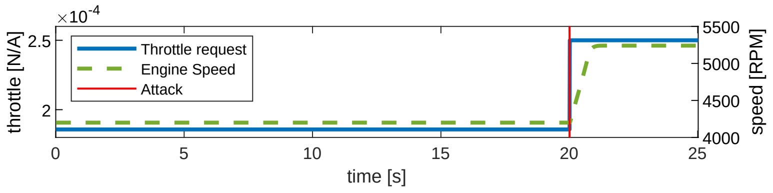


Figure 7.4: Injection throttle request attack

As clearly depicted in Figures 7.2, 7.3, and 7.4, all the inspected cyber-attacks achieve their goal of increasing the engine speed, either via a direct attack (e.g. the *inject throttle attack*) or as consequence of indirect attacks (e.g. the *inject speed attack* and *inject reference attack*).

7.5 Possible solutions

Two different techniques from the control theory are inspected as possible solutions for the mitigation of the consequences of cyber-attacks on the powertrain model.

The first solution requires to apply watermarking [60] for the detection of attacks on the system. The model with the watermarking technique is deployed and simulated for 20 seconds, thus without any attack on the system. The model is implemented and simulated in MATLAB on a laptop computer running Windows 10.

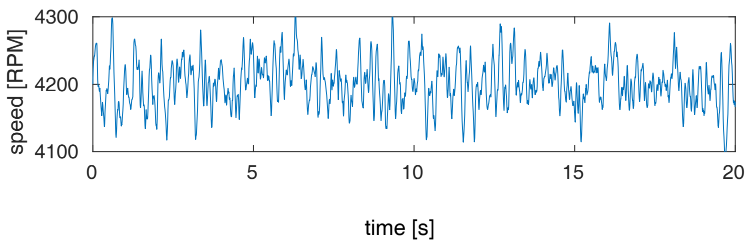


Figure 7.5: Model output with watermarking technique applied on the input signal.

Figure 7.5 shows the engine speed signal value (y-axis) while applying the watermarking technique to the powertrain model. As clearly depicted in Figure 7.5 it is possible to notice that the engine speed value is oscillating around the speed reference values (4200 RPMs). Such behavior may not be acceptable from both mechanical stress and driver perspective, since it may result in over-stressing mechanical parts of the powertrain section and it can introduce severe discomfort while activating the cruise control system [23]. Hence watermarking is not applicable in this context.

The second solution is the software rejuvenation [76]. This method is tested and applied to the system, allowing it to be periodically restored in a safe state. As for the previous solution, the model with the rejuvenation technique is simulated in conjunction with the anomaly detection algorithm described in 5.3. The model is implemented and simulated in MATLAB on a laptop computer running Windows 10, with an attack-free simulation for the first 20 seconds, and the attack simulated for the following 10 seconds.

The results of the software rejuvenation process applied to the powertrain model of a generic internal combustion engine are shown in Figure 7.6, where the rejuvenation process is triggered every second while simulating the

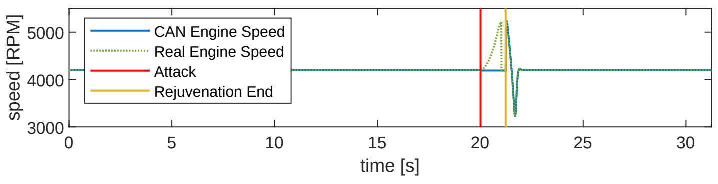


Figure 7.6: Software Rejuvenation triggered after the *injection speed* attack is detected.

injection speed attack, as described in Section 7.4.1. Figure 7.6 compares the value of the *engine_speed* signal received on the CAN bus (continuous line) with the real speed of the engine (dashed line) as evaluated by the model. The left-most vertical line represents the instant the attacks is started, while the right-most vertical line represents the instant the rejuvenation process is concluded and where the system is recovered in a steady and observable state. By the analysis of the results depicted in Figure 7.6 it is clear that the software rejuvenation method is able to restore the engine model to a safe and observable state. It is necessary to remark however, that the depicted case is only one of the attacks described in 7.3 and represents only the promising results achievable by means of Software Rejuvenation following the detection of anomalies.

Chapter 8

Anomaly reaction exploiting existing mechanisms

In this Chapter a solution designed for the reaction to the detection of a cyber-attack is proposed. The proposed reaction mechanism leverages a mechanical prevention mechanism deployed in modern vehicles that allows ECUs to enter a safe-mode, preventing further mechanical and/or electronic failures. This system, called *limp back home mode*, is usually triggered by safety control units to avoid irreversible damages to the mechanical parts of the vehicles. This reaction mechanism proposed in this Chapter is called *vehicle safe-mode*, and it is designed to mitigate the consequences of identified cyber-attacks. Two different modes for the *vehicle safe-mode* are introduced, to provide a more flexible integration with future reaction mechanisms.

8.1 Limp back home mode

The *limp back home mode* is a mechanism designed to limit the potential damage to either mechanical or electrical malfunction, allowing the driver to operate the vehicle with limited functionality without forcing it to a complete stop, preventing further damages. In modern vehicles, the *limp back home mode* is activated automatically after an ECU detects a malfunction in one or more vehicle subsystems. It is possible to distinguish between two different types of Limp-modes: a *local limp-mode* that is limited to the operation of a single ECU, and a *global limp-mode* affecting the global state of the vehicle.

The **local limp-mode** is a physical pin that, when activated by applying the proper voltage, allows to override the normal behavior of the microcontroller and drive the output pins directly to pre-configured settings (e.g. the technical documentation of the DRV8305-Q1 automotive micro-controller [87]).

Local limp-mode can be easily deactivated by restoring the normal voltage to the limp-mode pin, thus restoring the normal operation of the microcontroller.

Global limp-mode is activated when one of the central ECUs connected to the in-vehicle network detects possible fail conditions by analyzing the values of the messages received from the CAN bus (e.g., the Central BCM produced by Infineon [40]). As an example, the global limp-mode may be activated if the coolant temperature rises above safety values [34] or if the powertrain control module detects a failure (or near-failure) condition in the transmission [89]. Depending by the type and the severity of the failure, the central ECU triggers a set of operations that restrict the vehicle to a limited set of fail-proof and safe states. When the global limp-mode is triggered, the vehicle speed might be electronically limited to a maximum value, the transmission might be fixed in a specific low gear and, if an issue related to the engine is detected, the limp-mode can be used for shutting down the engine and gradually forcing the vehicle to a complete stop. The exact countermeasures deployed when the limp-mode is activated depend on the specific settings defined by the OEM. The global limp-mode may be implemented by activating the local limp-mode of some peripheral ECUs, allowing the main ECU to directly control their behavior. The deactivation of the global limp-mode also depends from the nature and severity of the detected failure. As an example, consider a limp-mode activated following the detection of transient failure conditions, which is usually automatically reset after the ignition of the vehicle or after a predefined amount of time. In some cases, the car owner can perform a sequence of operations that resets the limp-mode for non-severe failures, such as switching the car ignition on, and pressing and releasing the throttle pedal for a given number of times [3]. On the other hand, more severe failures may require a manual reset of the limp-mode, which is usually performed by operators of authorized car services by physically connecting to the OBD-II port and executing proprietary diagnostic software procedures. In the remainder of the Chapter the term limp-mode will be used for referring to the global limp-mode.

8.2 Adversary Model

The adversary model is analyzed by assuming that an attacker is able to obtain access to the CAN bus of a modern vehicle and to inject forged CAN messages. The number and the typology of injected messages is different depending from the final goal of the attacker. Two different attack injection scenarios are inspected: the *internal-injection* and the *external-injection.*

- **Internal-injection:** In this attack scenario the adversary is able to obtain full control over one or more ECUs, hence it has the ability to directly control some functions of the vehicle by altering the logic of the compromised ECUs. The attacker is also able to exploit the compromised ECUs to inject arbitrary messages over the CAN bus. This attack mode is depicted in Figure 8.1a.

- **External-injection**: In this scenario the adversary is not able to directly control any ECU but it is able to inject arbitrary CAN messages over the CAN bus either by physical access (such as a direct connection through the OBD-II diagnostic port) or remote access (e.g., by transmitting malicious messages over a legitimate wireless channel). This attack mode is shown in Figure 8.1b.

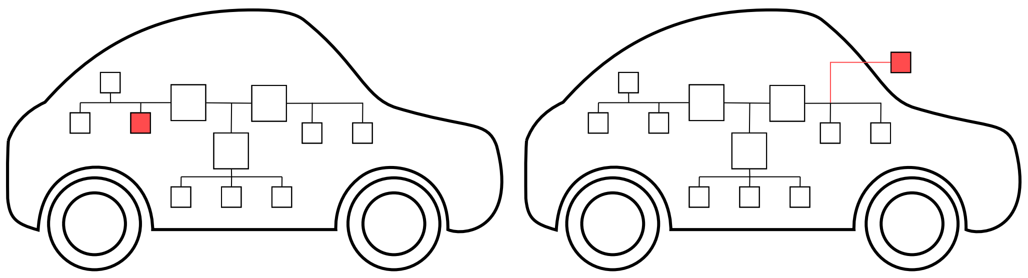

(a) Internal-injection (b) External-injection

Figure 8.1: Adversary models considered for this analysis. The attacker position is highlighted in red.

8.3 Vehicle Safe-mode architecture

The concept of vehicle *safe-mode* proposed in this Chapter is similar in principle to the limp-mode mechanism described in Section 8.1: the *safe-mode* mechanism is activated to allow the driver to safely stop the vehicle in case a cyber-breach is detected, while reducing the potential damage of these attacks to the vehicle system and to mitigate possible safety-relevant accidents.

The vehicle safe-mode system operates as follows: when a cyber-attack is detected, a safe-mode manager (*SMManager*) triggers the vehicle into a safe-mode condition in which several operations are limited or disabled, by sending an alert triggering message (*TMessage*) to other ECUs. The SMManager evaluates the vehicle state by analyzing the output of any existing intrusion detection system. The output of the SMManager also includes the recommended alert level and the chosen reaction mechanism, which are both

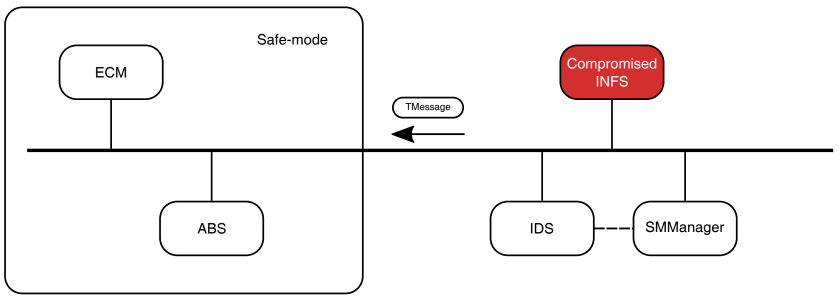


Figure 8.2: The system overview. Note that the *SMManager* can be connected directly to the IDS system, or alternatively, get its feedback over the bus.

encoded into the broadcast TMessages. The system overview is depicted in Figure 8.2.

Two different operational modes are presented for the implementation of the system: the *Transparent-mode* and the *Extended Mode*. In the former operational mode the SMManager only causes the ECUs in its same network segment to enter into their pre-configured limp-mode state, thus limiting the vehicle's functionality and reducing potential dangers. The main advantage of this mode is that it is deployable on all modern vehicles, since the introduction of the SMManager does not affect the other ECUs. In particular, the system may be deployed by adding a single OBD-connected entity to include the SMManager, with optional IDS capabilities.

The latter mode requires the addition of a novel software component, called the safe-mode client (*SMClient*) to the chosen ECUs. The main purpose of the SMClient is to process and react to the TMessages sent by the SMManager. The extended mode allows to increase the flexibility of the system, allowing different implementation of customized reactions for each individual ECU.

It is necessary to define the recovery options that allow the vehicle to exit from the safe-mode and to restore its complete functionality. This step is required for the prevention of unwanted deactivation of the safe-mode and for ensuring the driver with the ability to restore the vehicle when necessary.

8.3.1 Operation modes

Transparent-mode

In the transparent-mode of operation, the goal of the SMManager is to trigger the limp-mode state on the target ECUs to reduce the potential damage of an identified attack. This reaction mechanism requires to be effective for the reduction of potential damages to both mechanical and electronic components of the vehicle, while also ensuring the safety of the

Table 8.1: Table of TMessages that triggers the Limp-mode on different ECUs

ECU	**Msg ID**	**Data**
ECM	014	"Dangerous high engine temperature"
ECM	014	"Major engine malfunction"
ABS	004	"Dangerous low oil pressure"
TPM	020	"Dangerous low air pressure"

passengers.

For this purpose, the SMManager is required to maintain a list of all relevant CAN bus messages that typically cause each ECU to enter limp-mode. This list can be maintained by in a table that consists of all the relevant TMessages for each different ECU (Table 8.1). This table may include several different lines per ECU, in case multiple TMessages for each ECU are available.

This operational mode allows to apply the safe-mode mechanism to any existing vehicle, e.g., by connecting an after-market device (to include the *SMManager* and some anomaly-detection component) to its OBD-II port. A more sophisticated after-market device based on mobile devices allows to include more sophisticated notification and recovery options to the vehicle. However, the drawback of this operational mode is that the *safe-mode* reaction mechanism is only limited in the activation of the *limp-mode*, thus countermeasures that are designed specifically against cyber-breaches cannot be implemented.

Extended-mode

In the extended operational mode the SMManager is able to trigger selected ECUs into a particular safe-mode, instead of trigger their pre-configured limp-mode. The extended mode offers higher flexibility at the cost of increasing the requirements (thus the costs) of some software components (such the *SMClient*). This operational mode provides high freedom in the decision of the triggered reaction for each ECU, thus reducing the potential damage of a cyber-breach to the overall safety of the vehicle and passengers. Furthermore, this operational mode allows different reaction mechanisms according to the type and severity of the identified attack. The extended-mode also increases the flexibility for the definition of the notification and recovery options to deploy. Moreover, these notification could also allow the driver to override the suggested countermeasures with more strict ones, if desired.

In the extended mode the *SMManager* maintains a table of all the relevant messages triggering the safe-mode *TMessages*, for each ECU and for each different alert-level, including the type of desired reaction. It is necessary

TMessage ID	Alert Level	Reaction Level	[Counter]	[MAC]
11	3	5	8	48

Figure 8.3: candidate structure of an *Extended-mode TMessage*. The numbers represent the field length in bits. Note that the ID field is a regular CAN-ID-field, while the other fields fit into the CAN 8 byte data-field; Both the counter and the MAC fields are optional; *Transparent-mode TMessages* are regular (*Limp-mode* triggering) CAN messages.

to highlight that this table can be used for both modes of operation, even though the *transparent-mode* only requires a simplified version of the one required by the *extended-mode*.

A generic *TMessage* is based on the underlying protocol (typically the CAN protocol). Unlike the *transparent-mode TMessage*, the extended-TMessage includes the vehicle's Alert-level AL, the required Reaction-level RL, a replay counter and a truncated MAC evaluated with a robust algorithm. AThe candidate structure for an extended-TMessage is depicted in Figure 8.3.

The *SMClient* is a software component required to be included in all the participating ECUs, thus allowing proper identification, processing and reaction to the *safe-mode TMessages*. The *SMManager* is responsible for the required management and distribution of cryptographic material for the evaluation of the truncated MAC in the *TMessages*. Several solutions for key management have already been proposed in literature, from factory serialization to specialized solutions, and have already been covered in Chapter 4.

Safe-mode Manager

The SMManager is responsible for processing the outputs of the intrusion detection systems deployed in the network, evaluating the vehicle alert-level (AL), the corresponding reaction-level (RL), and for triggering the vehicle safe-mode by sending the appropriate *TMessages*.

The *SMManager* is also responsible for the management of cryptographic materials required for the evaluation of the truncated MAC of the *TMessages*. It is necessary to remark that the *SMManager* can implementable in both hardware and software. Despite having a dedicated hardware component for the SMManager tasks implies higher costs for the vehicle manufacturer, this solution also increases the cyber-resistance of the suggested mechanism, thus improving the security guarantees offered by the mechanism.

Topology

The SMManager can be implemented in two different ways according to the topology of the internal networks and the computational load of each

ECU: the *Independent SMManager*, and the *Incorporated SMManager*.

The **Independent SMManager** implementation allows to deploy the logical operation performed by the *SMManager* on a dedicated hardware module, as either an internal or external component. The internal *SMManager* is a dedicated ECU responsible for collecting the different notifications generated by the IDS deployed in the internal network to trigger the vehicle *safe-mode* when necessary. The external *SMManager* can be implemented as a dedicated OBD-II dongle (Figure 8.4a) and allows to provide an aftermarket solution for the implementation of the vehicle safe-mode. The external solution requires that the relevant internal networks are exposed on the OBD-II port, thus allowing the SMManager to access the content of the powertrain section and to inject the required TMessages.

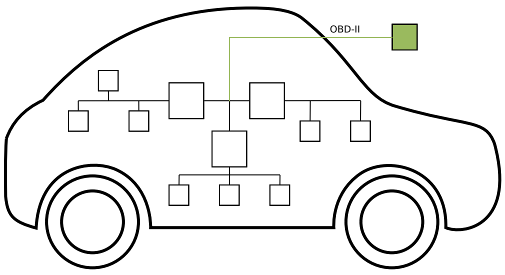
The **Incorporated SMManager** implementation allows to deploy the logical operation performed by the *SMManager* on an existing ECU of the internal network (as shown in Figure 8.4b). This option implements the *SMManager* as part of the vehicle system by design. An *Incorporated SMManager* allows three different topologies for its implementation:

- *Centralized SMM:* the logic for the *SMManager* is embedded on a centralized ECU (e.g., the ECM or BCM)
- *Distributed SMM:* the logic for the *SMManager* is embedded on multiple ECUs of the network, each one with its specific set of operations allowing both for monitoring and triggering of the vehicle *safe-mode*.
- *Hybrid SMM:* a composition of the two previous topologies: different instances of the same *SMManager* are responsible for the monitoring and the collection of different information, that are forwarded to the centralized *SMManager*. The vehicle safe-mode is activated only by the centralized SMManager.

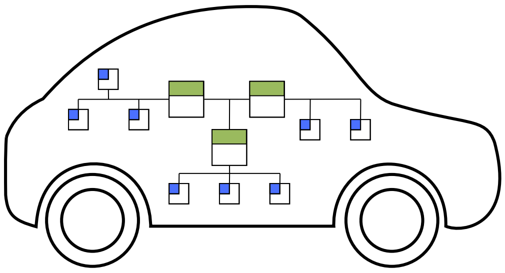
8.3.2 The safe-mode client

To provide the necessary support required by the *Extended-mode* it is necessary to deploy a *SMClient* allowing proper identification, processing and reaction to the custom *SMManager TMessages* for the participating ECUs.

The reaction list for each reaction-level (*RL*) encoded in the *TMessage* is maintained by the client. The reactions are defined by the manufacturer to limit the potential damage of the possible attacks to the vehicle. To prevent adversarial manipulation, the *TMessages* are authenticated and the *SMClient* is responsible for the validation of the authentication tag embedded in the



(a) *External independent SMM*



(b) *Distributed incorporated SMM*

Figure 8.4: *Safe-mode Manager* suggested topologies. The *SMManagers* are marked in green, while the *SMClients* are marked in blue. Note that Figure (*a*) shows one example for a system in *Transparent-mode*, while Figure (*b*) shows an example for a system in *Extended-mode*.

TMessage. After a valid *TMessage* is received, the *SMClient* triggers the *safe-mode* on the hosting ECU, thus performing the relevant pre-configured actions encoded in his reaction table. The *SMClient* requires to support the recovery mechanism, allowing proper recovery of the hosting ECU when required. Recovery could be either triggered automatically depending on the system configuration or upon reception of a special recovery message.

8.3.3 The vehicle alert-level

The *SMManager* is responsible for the evaluation of the vehicle alert-lever (AL), independently from its implementation. Different alert level denotes a different threat level, thus requiring to deploy appropriate reactions. For the evaluation of the current AL the *SMManager* requires the deployment of Intrusion Detection Systems designed for the detection of anomalies in the inspected vehicle network. The IDS generates the main source of information required for the evaluation of the AL. Intrusion Detection Systems designed for the internal vehicle network could be either based on the inspection of the messages (thus based on open standards 5), or based on the inspection of the signals encoded in the messages (thus based on closed standards 6). The *SMManager* collects and analyzes the security alerts and evaluates the current *AL*. For the purposes of this Section, five different levels of AL are inspected, each one representing the increasing severity of the alert.

8.4 Possible reaction, recovery plans, and potential problems

8.4.1 Reaction

In this paragraph the different actions triggered by the SMManager after the detection of a cyber-breach are described. The two reaction steps of the *SMManager* are described as follows:

- Notification: feedback to the driver and the vehicle surroundings about the identified attack and the chosen reaction;
- Action: countermeasures deployed by the SMManager to mitigate the consequences of the attack.

The trigger order for notifications and actions are defined in the relative alert-level.

Reaction-Matrix

The reaction-matrix is used by the *SMManager* for the evaluation of the the reaction-level (RL), and it encodes the steps required for the mitigation of a cyber-attack. The RL is evaluated by using the current alert-level and the current vehicle-condition (VC). The current VC represents the condition of the vehicle dynamics (including speed, yaw, roll, pitch, lateral acceleration and outputs of the ABS and ESP systems) evaluated by the SMManager. It is important to consider the current vehicle conditions to allow proper evaluation of the reaction level by the *SMManager*, preventing the vehicle state to be forced in more dangerous state with respect to the state in which the attack is detected. In the scenario in which countermeasures are deployed without inspecting the current vehicle conditions, the decision of the SMManager might be able to exclude the ESP or any of the ADAS systems. While this decision might be appropriate for a vehicle running at low speed on a straight road, it can increase the safety risks associated in case the vehicle state is considered safety-critical (such as at high speed, or

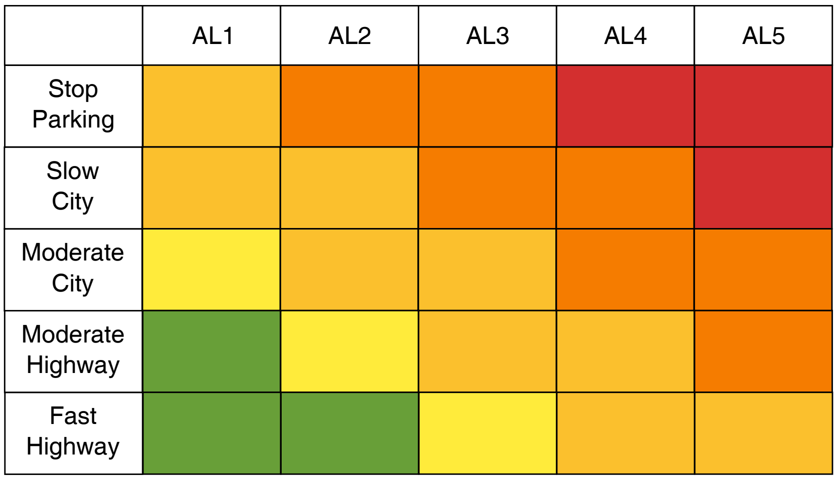

AL1
AL2
AL3
AL4
AL5
Stop Parking
Slow City
Moderate City
Moderate Highway
Fast Highway

Figure 8.5: Example of a reaction matrix based on a 5 × 5 AL/VC structure.

under high lateral accelerations). To prevent similar situations the reaction-matrix requires different countermeasures associated with the same AL, but in different vehicle conditions. The evaluated RL is used to determine the most appropriate reaction, aiming to secure the vehicle state, mitigating the safety-risks of any detected cyber-breach.

An example for a RM is depicted in Figure 8.5, which includes different *RL* matching the combinations of a given 5 × 5 *AL/VC* structure, where the columns represent the different alert level and different colors are used to highlight different reaction levels.

Actions

Following the activation of the *safe-mode*, the SMManager is responsible for the activation of different actions for deploy the most appropriate reaction mechanism, depending from the reaction-matrix. These actions could take place before, after or during the notification phase, thus it is necessary to define the timing sequence.

Some actions might differ depending from the triggered reaction-level, hence requiring multiple intensity levels according to the generated AL. While lower value of the RL only trigger limited actions to recover from not safety-critical situations, higher RL values require more invasive solutions, reflecting higher safety risks.

Triggered actions range from the already existing *Limp-mode* operations (e.g., limit the vehicle speed) to custom actions such as those presented below:

- **Ignore all non-critical messages**: ECUs are allowed to ignore attacks that leverage non-critical messages, reducing the computational power required by ECUs to operate the vehicle, thus increasing the computational power required for the safe-mode;

- **Shutdown specific ECUs**: the ECU classified as responsible for a cyber-attack is removed from the network operation, avoiding further impact of the attack on the vehicle system.

- **Reset specific ECUs**: an extension of the previous action, allowing the controller to reboot the ECU after the shutdown procedure is complete.

- **Authentication on some (CAN) messages**: forces the usage of cryptographic primitives for the authentication of the messages to mitigate spoofing attacks.

- **Encryption on some (CAN) messages:** forces the usage of cryptographic primitives for the encryption of the messages to mitigate a wide scenario of cyber-attacks.

- **Segment isolation:** isolates a target network segment from the network, containing the consequences of attacks.

- **Parallel network:** a secondary network implementation with limited capabilities is used instead of the normal network, enabling communication between only critical ECUs on a different interface. It is necessary to remark that an extra interface is required by critical ECUs.

Notification

Notifications are considered to be both internal or external. Internal notifications are used to notify the driver that the *SMManager* is performing different actions to react to the evaluated AL. These notifications could be either acoustic, visual, or even include haptic feedback on different parts of the cockpit, like the steering wheel or the pedals. A more articulated schema for vehicle internal-notifications has already been proposed in [38]. External notifications are mostly used to notify other drivers, vehicles, and nearby pedestrians of a potentially dangerous situation. External notifications might be implemented through visual feedback for fast reaction by all the involved parties, but also includes external messages sent using vehicle-to-vehicle (V2V) or vehicle-to-infrastructure (V2I) technologies.

8.4.2 Recovery phase

The *Recovery* phase is the last phase of the vehicle *safe-mode*, and it is started after the reaction phase is terminated. The recovery procedure aims to restore the vehicle normal behavior under safe circumstances. The *SMManager* evaluates when the recovery operations are started. Different recovery modes are inspected: self recovery, driver-initiated recovery, and authorized-garage recovery.

- **Self-recovery** allows the SMManager to trigger the recovery procedure autonomously. The self recovery procedure should be applied only if non-critical parts of the network are involved in the cyber-attack. An example of a self-recovery procedure has already proposed in Section 7.5.
- **Driver-initiated recovery** allows the driver to trigger the recovery procedure by requesting his authorization through the HMI of the vehicle. This option allows to increase the robustness of the mechanism, preventing the attacker to trigger the recovery procedure during the attack.
- **Authorized-Garage recovery** allows to start the recovery procedure only by authorized garages, following major attacks to the network or in case the attack has not been mitigated properly by the deployed reactions.

A proposal of the metrics used by the SMManager for triggering the required recovery mode is provided as follows:

- **iAL:** the initial Alert-level, computed before the reaction phase;
- **RL:** the previously evaluated Reaction-level, computed before the reaction phase;
- **aAL:** the actual Alert-level, computed after the reaction phase;
- **aVC:** the actual Vehicle-condition, computed after the reaction phase.

8.4.3 Potential problems

Limitations and side effects of the safe-mode mechanism are inspected in this Section.

False positives

The IDS or anomaly detection component is a critical, yet external, part of the *safe-mode* system, which is responsible for the input provided to the SMManager to trigger the *safe-mode*. This implies that the limitations of the IDS affect the safe-mode mechanism, thus it is necessary to consider that the safe-mode could be triggered when unnecessary in case of false positives. However, it is necessary to remark that the false positive rate of any deployed IDS might be previously considered while deploying the vehicle safe-state. Moreover, a detection framework as the one proposed in Section 5.7 might mitigate the false positive issue.

Adversarial triggering

Adaptive adversaries might be able to trigger the *safe-mode* mechanism by sending maliciously forged TMessages while the system is deployed in *Transparent-mode* or in a non-secured *extended-mode*. However, this scenario could be similar to the false positive described above.

Transparent-mode TMessage collisions

It is necessary to ensure that no collisions between TMessages and legit messages occurs while the SMManager is configured in transparent-mode. To prevent message collisions, it is necessary to include the TMessages as part of the design of the vehicle. This scenario only applies if the SMManager is deployed in extended-mode, thus the collision issue in the transparent-mode is not relevant.

Transparent-mode, TMessage overriding

Another issue related to the transparent-mode SMManager is that legit CAN messages could override the content of TMessages sent by the SMManager. For the prevention of this scenario, it is necessary to deploy the same countermeasures deployed for the TMessage collision scenario.

8.5 Safe mode summary

This Chapter described a concept for the vehicle *safe-mode*, designed to mitigate the damages of cyber-attacks to the vehicle network. Differently from other defense mechanisms designed to block attacks or to simply notify their existence, the mechanism proposed in this Chapter allows to actively respond to the detection of attacks by limiting the functionality of the vehicle and triggering security countermeasures. The proposed mechanism exploits the existing mechanism of the *limp-mode*, originally designed to limit the potential damage of either mechanical and/or electrical malfunctions. The proposed mechanism introduces two different modes for its *safe-mode* operation: the *transparent-mode*, in which the pre-configured *limp-mode* is activated upon detection of cyber attacks; and the *extended-mode*, in which custom messages are used to increase the flexibility of both reaction and recovery phases. The *extended-mode* requires heavy modifications to the participating ECUs, either via software or hardware, while the *transparent-mode* is designed to be deployable on existing vehicles by connecting a dedicate OBD-II dongle to the OBD-II port of the vehicle, without requiring any modification of the internal network.

Chapter 9

Conclusions

The increasing adoption of automatic and autonomous controls in modern vehicles, together with vulnerabilities of automotive protocols and microcontrollers to a wide range of cyber attacks, motivates the current research efforts aiming to improve the cybersecurity of the automotive ecosystem. Within this scenario, this thesis improves the state of the art by proposing original solutions designed to improve the security of the internal network of modern vehicles.

Chapter 4 proposes an analysis of the current state-of-the-art solutions for secure communications applied to intra-vehicular networks and Electronic Control Units connected to the CAN network. This analysis considers the full life-cycle of a modern vehicle (from design to disposal) and shows that many solutions that may appear efficient and secure when deployed on a single vehicle, present severe disadvantages when deployed at scale. Results of this analysis also shows that solutions based on pre-shared symmetric secrets complicate the management and maintenance of the vehicles. Moreover, solutions not based on a centralized point of control managed by the car manufacturer are not able to offer the desired security guarantees.

Chapter 5 presents five novel algorithms for the detection of attacks and anomalies by analyzing CAN data frames. Each algorithm inspects a different feature available from the analysis of network communications gathered from a modern, licensed and unmodified vehicle. The detection performance of the proposed algorithms are evaluated experimentally against the same dataset. Moreover, this thesis designs a detection framework that leverages the detection capabilities of the proposed attack detection algorithms, highlighting that complete coverage of all realistic threats can be achieved through a combination of two of the proposed algorithms, while improved detection rate can be achieved by combining all of them.

To overcome the limitation posed by the lack of public specifications describing the content of CAN messages, this thesis also proposes READ (see Chapter 6), a reverse engineering algorithm designed for the extraction of signal boundaries from generic CAN traffic traces. Extensive experimental evaluations against the ground truth represented by the full specification of a modern, unmodified, licensed vehicle show that READ extracts more than twice correct signals with respect to previous work. Moreover, READ is characterized by lower execution times and comparable convergence requirements. The results of READ enable the analysis of the behavior of CAN signals, and different detection model are proposed for the different signal types extracted by READ: counters, CRCs, and signals representing the evolution over time of physical measures.

Chapter 7 proposes a novel detection and reaction approach that fuses intrusion detection techniques with solutions borrowed from the domain of control theory. This mixed approach is motivated by cyber-attacks that can affect the cyber-physical model of the vehicle without being detected by control-theory techniques alone. To overcome this critical issue, a novel solution based on the software rejuvenation method is proposed and evaluated through simulations carried out over a mathematical model representing the powertrain section of a vehicle equipped with cruise control.

The last contribution of this thesis is the proposal of a safe-mode concept for modern vehicles (see Chapter 8). The proposed reaction strategy exploits the limp-home mode mechanisms to mitigate the physical effect of a cyber attack after its detection.

Future research directions will expand on the significant results presented in this thesis to improve the security of future, cooperative Intelligent Transportation Systems (ITS).

At first it is necessary to adapt all the algorithms proposed in this thesis for future bus technologies that will eventually replace CAN, such as CAN-FD and automotive Ethernet. Moreover, since future bus technologies are designed to include security solutions, it is necessary to integrate possible flaws with the solutions proposed in literature. The lack of standardization for the technologies used in the development of current in-vehicle communication protocols and embedded devices exposed the necessity of a security test bed, enabling a baseline for the definition of the security characteristics of future automotive-grade microcontrollers.

Another possible expansion of the work proposed in this thesis is focused on the inspection of the secure protocols used for vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications. These solutions are inspected as part of the future generation of ITS, that will focus on the cooperation between connected devices to increase the efficiency of the road management. Despite the future ITS scenario paves the way to many different and interesting solutions, it is also necessary to grant authenticity, non repudiation, and confidentiality of the communications. These security guarantees are well established solutions in classical IT communication, but the ITS scenario raises novel and unaddressed issues due to its composition of multiple dynamic objects, with limited computational capabilities, and operational time constraints. Moreover, ITS are based on beacon-broadcast communication between the nodes, with a maximum allowed latency for each communication. Current ITS standard defines the maximum density for each communication beacon of more than 150 devices. In this scenario, each device requires to validate the authenticity of more than 150 messages sent from all the other devices (in the worst-case scenario) with very strict time constraints.

To address these problems, future work will focus on the analysis, the development, and the deployment of security protocols designed to be compliant with the ITS standards.

Bibliography

[1] F. Angiulli, L. Argento, and A. Furfaro. Exploiting n-gram location for intrusion detection. In *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1093–1098, Nov 2015.

[2] F. J. Anscombe and I. Guttman. Rejection of outliers. *Technometrics*, 2(2):123–147, 1960.

[3] AnthonyJ350. Nissan 350z/ infiniti g35 ecu reset (check engine light). `https://www.youtube.com/watch?v=5tQm28K32SQ`, 2016.

[4] S. Basu, M. Bilenko, and R. J. Mooney. A probabilistic framework for semi-supervised clustering. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 04, page 5968, New York, NY, USA, 2004. Association for Computing Machinery.

[5] G. Bella, P. Biondi, G. Costantino, and I. Matteucci. Toucan: A protocol to secure controller area network. In *Proceedings of the ACM Workshop on Automotive Cybersecurity*, AutoSec 19, page 38, New York, NY, USA, 2019. Association for Computing Machinery.

[6] P. Biondi, G. Bella, G. Costantino, and I. Matteucci. Implementing can bus security by toucan. In *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Mobihoc 19, page 399400, New York, NY, USA, 2019. Association for Computing Machinery.

[7] S. Byers and A. E. Raftery. Nearest-neighbor clutter removal for estimating features in spatial point processes. *Journal of the American Statistical Association*, 93(442):577–584, 1998.

[8] A. Callado, C. Kamienski, G. Szabo, B. P. Gero, J. Kelner, S. Fernandes, and D. Sadok. A survey on internet traffic identification. *IEEE Communications Surveys Tutorials*, 11(3):37–52, rd 2009.

[9] W. B. Cavnar and J. M. Trenkle. N-gram-based text categorization. In *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

[10] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computer Surveys*, 41(3):15:1–15:58, July 2009.

[11] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *Proceedings of the 20th USENIX Conference on Security (SEC'2011)*, Aug 2011.

[12] A. L. M. Chiu and Ada Wai-chee Fu. Enhancements on local outlier detection. In *Seventh International Database Engineering and Applications Symposium, 2003. Proceedings.*, pages 298–307, July 2003.

[13] D. Cho and J. K. Hedrick. Automotive powertrain modeling for control. *Journal of dynamic systems, measurement, and control*, 111(4):568–576, 1989.

[14] K. Cho and K. G. Shin. Viden: Attacker identification on in-vehicle networks. *ACM Conference on Computer and Communications Security*, abs/1708.08414, 2017.

[15] K.-T. Cho and K. G. Shin. Error handling of in-vehicle networks makes them vulnerable. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1044–1055, New York, NY, USA, 2016. ACM.

[16] K.-T. Cho and K. G. Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, pages 911–927, Berkeley, CA, USA, 2016. USENIX Association.

[17] Comma.AI. Comma cabana, 2018.

[18] Comma.AI. Comma.ai - ghostriding for the masses, 2018.

[19] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley-Interscience, New York, NY, USA, 1991.

[20] M. J. Desforges, P. J. Jacob, and J. E. Cooper. Applications of probability density estimation to the detection of abnormal conditions in engineering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 212(8):687–703, 1998.

[21] S. Di Cairano, D. Yanakiev, A. Bemporad, I. V. Kolmanovsky, and D. Hrovat. Model predictive idle speed control: Design, analysis, and experimental evaluation. *IEEE Transactions on Control Systems Technology*, 20(1):84–97, 2012.

[22] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition).* Wiley-Interscience, USA, 2000.

[23] W. El Falou, J. Duchne, M. Grabisch, D. Hewson, Y. Langeron, and F. Lino. Evaluation of driver discomfort during long-duration car driving. *Applied Ergonomics*, 34(3):249 – 255, 2003.

[24] E. Eskin. Anomaly detection over noisy data using learned probability distributions. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML 00, page 255262, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[25] A. Francillon, B. Danev, and S. Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *Proc. of the 18th Annual Network & Distributed System Security Symposium (NDSS 2011)*, Feb 2011.

[26] R. B. GmbH. Can specification version 2.0, 1991.

[27] M. Gmiden, M. H. Gmiden, and H. Trabelsi. An intrusion detection method for securing in-vehicle CAN bus. *2016 17th International Conference on*

Sciences and Techniques of Automatic Control and Computer Engineering (STA), pages 176–180, 2016.

[28] R. M. Gray. *Entropy and information theory.* Springer Science & Business Media, 2011.

[29] A. Greenberg. Hackers remotely kill a jeep on the highway - with me in it!, 2015.

[30] B. Groza, S. Murvay, A. Van Herrewege, and I. Verbauwhede. Libra CAN: a Lightweight Broadcast Authentication Protocol for Controller Area Networks. In *Proc. 11th Int'l Conf. Cryptology and Network Security*, 2012.

[31] B. Groza, S. Murvay, A. van Herrewege, and I. Verbauwhede. Libra-can: A lightweight broadcast authentication protocol for controller area networks. In J. Pieprzyk, A.-R. Sadeghi, and M. Manulis, editors, *Cryptology and Network Security*, pages 185–200, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[32] Y. Gu, A. McCallum, and D. Towsley. Detecting anomalies in network traffic using maximum entropy estimation. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement (IMC '05)*, 2005.

[33] L. Guzzella and C. Onder. *Introduction to modeling and control of internal combustion engine systems.* Springer Science & Business Media, 2009.

[34] Hagens Berman Sobol Shapiro LLP. Ford shelby gt350 mustang overheating. `https://www.hbsslaw.com/cases/ford-shelby-gt-mustang-overheating`, 2005.

[35] R. W. Hamming. Error detecting and error correcting codes. *Bell Syst. Tech. J*, 29(2):147–160, 1950.

[36] O. Hartkopp, C. Reuber, and R. Schilling. MaCAN - Message Authenticated CAN. *10th International Conference on Embedded Security in Cars*, 2012.

[37] T. Hoppe and J. Dittman. Sniffing/replay attacks on can buses: A simulated attack on the electric window lift classified using an adapted cert taxonomy. In *Proc. of the 2nd Workshop on Embedded Systems Security (WESS)*, Oct 2007.

[38] T. Hoppe, S. Kiltz, and J. Dittmann. Adaptive dynamic reaction to automotive IT security incidents using multimedia car environment. In *2008 The Fourth International Conference on Information Assurance and Security*, pages 295–298, Sept 2008.

[39] N. Hubballi, S. Biswas, and S. Nandi. Layered higher order n-grams for hardening payload based anomaly intrusion detection. In *2010 International Conference on Availability, Reliability and Security*, pages 321–326, Feb 2010.

[40] Infineon Technologies AG. Driving the future of automotive electronics, automotive application guide. `http://www.infineon.com/dgdl/Infineon-Automotive_Application_Guide-ABR-v00_00-EN.pdf?fileId=db3a30431c48a312011c6696b47402cc`, 2014.

[41] S. Jafarnejad, L. Codeca, W. Bronzi, R. Frank, and T. Engel. A car hacking experiment: When connectivity meets vulnerability. In *Proc. of the 2015 IEEE Globecom Workshops (GC Wkshps)*, Dec 2015.

[42] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice-Hall, Inc., USA, 1988.

[43] M. J. Kang and J. W. Kang. A novel intrusion detection method using deep neural network for in-vehicle network security. In *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, pages 1–5, May 2016.

[44] Keen Security Lab of Tencent. Car hacking research: Remote attack tesla motors, 2016.

[45] Keen Security Lab of Tencent. New car hacking research: 2017, remote attack tesla motors again, 2017.

[46] M. Kneib and C. Huth. Scission: Signal characteristic-based sender identification and intrusion detection in automotive networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 787–800, New York, NY, USA, 2018. ACM.

[47] J. Z. Kolter and M. A. Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 7(Dec):2721–2744, 2006.

[48] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Snachám, and S. Savage. Experimental security analysis of a modern automobile. *Proceedings of IEEE Symposium on Security and Privacy*, pages 447–462, 2010.

[49] V. Kumar. Parallel and distributed computing for cybersecurity. *IEEE Distributed Systems Online*, 6(10), 2005.

[50] P. Laskov and N. Šrndić. Static detection of malicious javascript-bearing pdf documents. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 373–382. ACM, 2011.

[51] H. Lee, K. Choi, K. Chung, J. Kim, and K. Yim. Fuzzing can packets into automobiles. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pages 817–821, March 2015.

[52] C.-W. Lin and A. Sangiovanni-Vincentelli. Cyber-Security for the Controller Area Network (CAN) Communication Protocol. In *Proc. 2012 Int'l Conf. Cyber Security.*

[53] C. Ling and D. Feng. An algorithm for detection of malicious messages on can buses. In *2012 National Conference on Information Technology and Computer Science. Atlantis Press*, 2012.

[54] F. E. M.A. Xli. on discordant observations. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 23(143):364–375, 1887.

[55] H. Mansor, K. Markantonakis, R. N. Akram, K. Mayes, and I. Gurulian. Log your car: The non-invasive vehicle forensics. *2016 IEEE Trustcom*, pages 974–982, Aug 2016.

[56] M. Markovitz and A. Wool. Field classificatin, modeling and anomaly detection in unknown can bus networks. In *Proceedings of the 14th Conference on Embedded Security in Cars (ESCAR 2015)*, Oct 2015.

[57] M. Markovitz and A. Wool. Field classification, modeling and anomaly detection in unknown can bus networks. *Vehicular Communications*, 9:43 – 52, 2017.

[58] C. Miller and C. Valasek. Adventures in automotive networks and control units, 2014.

[59] C. Miller and C. Valasek. Remote exploitation of an unaltered passenger vehicle, 2015.

[60] Y. Mo, R. Chabukswar, and B. Sinopoli. Detecting integrity attacks on SCADA systems. *IEEE Transactions on Control Systems Technology*, 22(4):1396–1407, 2014.

[61] A. W. Moore and K. Papagiannaki. Toward the accurate identification of network applications. In *Proceedings of the 6th International Conference on Passive and Active Network Measurement*, PAM'05, pages 41–54, Berlin, Heidelberg, 2005. Springer-Verlag.

[62] M. R. Moore, R. A. Bridges, F. L. Combs, M. S. Starr, and S. J. Prowell. Modeling inter-signal arrival times for accurate detection of can bus signal injection attacks: A data-driven approach to in-vehicle intrusion detection. In *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*, CISRC '17, pages 11:1–11:4, New York, NY, USA, 2017. ACM.

[63] P. Mundhenk, A. Paverd, A. Mrowca, S. Steinhorst, M. Lukasiewycz, S. A. Fahmy, and S. Chakraborty. Security in Automotive Networks: Lightweight Authentication and Authorization. *ACM Transaction on Design Automation of Electronic Systems*, 22(2), 2017.

[64] M. Müter and N. Asaj. Entropy-based anomaly detection for in-vehicle networks. *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 1110–1115, June 2011.

[65] M. Müter, A. Groll, and F. Freiling. A structured approach to anomaly detection for in-vehicle networks. In *Proc. of the Sixth International Conference on Information Assurance and Security (IAS 2010)*, Aug 2010.

[66] D. K. Nilsson and U. E. Larson. Conducting forensic investigations of cyber attacks on automobile in-vehicle networks. *Proceedings of the 1st International Conference on Forensic Applications and Techniques in Telecommunications, Information, and Multimedia and Workshop*, pages 8:1–8:6, 2008.

[67] D. K. Nilsson, U. E. Larson, and E. Jonsson. Efficient in-vehicle delayed data authentication based on compound message authentication codes. *Proceedings IEEE 68th Conference on Vehicular Technology*, 2008.

[68] H. Oguma, A. Yoshioka, M. Nishikawa, R. Shigetomi, A. Otsuka, and H. Imai. New Attestation-Based Security Architecture for In-vehicle Communication. In *Proc. 2008 IEEE Int'l Conf. Global Communications.*

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

[70] F. Pasqualetti, F. Dörfler, and F. Bullo. Attack detection and identification in cyber-physical systems. *IEEE transactions on automatic control*, 58(11):2715–2729, 2013.

[71] PEAK-System. Pcan-usb.

[72] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee. Mcpad: A multiple classifier system for accurate payload-based anomaly detection. *Computer networks*, 53(6):864–881, 2009.

[73] M. D. Pesé, T. Stacer, C. A. Campos, E. Newberry, D. Chen, and K. G. Shin. Librecan: Automated can message translator. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2283–2300. ACM, 2019.

[74] A.-I. Radu and F. D. Garcia. Leia: A lightweight authentication protocol for can. In I. Askoxylakis, S. Ioannidis, S. Katsikas, and C. Meadows, editors, *Computer Security – ESORICS 2016*, pages 283–300, Cham, 2016. Springer International Publishing.

[75] K. Rieck and P. Laskov. Detecting unknown network attacks using language models. *Detection of Intrusions and Malware & Vulnerability Assessment*, pages 74–90, 2006.

[76] R. Romagnoli, B. H. Krogh, and B. Sinopoli. Design of software rejuvenation for cps security using invariant sets. *To appear on the Proceedings of the 2019 IEEE American Control Conference*, 2019.

[77] I. Rouf, R. Miller, H. Mustafa, T. Taylor, S. Oh, W. Xu, M. Gruteser, W. Trappe, and I. Seskar. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In *Proc. of the 19th USENIX Conference on Security (USENIX Security'10)*, Aug 2010.

[78] C. Smith. The car hacker's handbook, 2016.

[79] C. Spence, L. Parra, and P. Sajda. Detection, synthesis and compression in mammographic image analysis with a hierarchical image probability model. In *Proceedings IEEE Workshop on Mathematical Methods in Biomedical Image Analysis (MMBIA 2001)*, pages 3–10, Dec 2001.

[80] D. Stabili and M. Marchetti. VTC2019Fall Dataset, 2019.

[81] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, K. Mohamed, and Y. Laarouchi. Survey on security threats and protection mechanisms in embedded automotive networks. In *Proc. of the 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W 2013)*, June 2013.

[82] C. Y. Suen. N-gram statistics for natural language understanding and text processing. *IEEE transactions on pattern analysis and machine intelligence*, (2):164–172, 1979.

[83] P. Syverson. A taxonomy of replay attacks [cryptographic protocols]. In *Proceedings The Computer Security Foundations Workshop VII*, pages 187–191, June 1994.

[84] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining, (First Edition).* Addison-Wesley Longman Publishing Co., Inc., USA, 2005.

[85] A. Taylor, N. Japkowicz, and S. Leblanc. Frequency-based anomaly detection for the automotive can bus. In *2015 World Congress on Industrial Control Systems Security (WCICSS)*, pages 45–49, Dec 2015.

[86] A. Teixeira, I. Shames, H. Sandberg, and K. H. Johansson. A secure control framework for resource-limited adversaries. *Automatica*, 51:135–148, 2015.

[87] Texas Instruments Incorporated. DRV8305-Q1 three-phase automotive smart gate driver with three integrated current shunt amplifiers and voltage regulator. `http://www.ti.com/lit/ds/symlink/drv8305-q1.pdf`, 2016.

[88] A. Theissler. Anomaly detection in recordings from in-vehicle networks. In *Proceedings of the 1st International Workshop on Big Data Applications and Principles (BIGDAP 2014)*, Sep 2014.

[89] Transmission Repair Cost Guide. What is limp mode? causes and what to do. `https://www.transmissionrepaircostguide.com/limp-mode/`, 2015.

[90] C. Valasek and C. Miller. Car Hacking: For Poories, 2014.

[91] A. Van Herrewege, D. Singelee, and I. Verbauwhede. CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus. In *ECRYPT Workshop on Lightweight Cryptography*, volume 2011, 2011.

[92] A. Wagner and B. Plattner. Entropy based worm and anomaly detection in fast ip networks. In *Proc. of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05)*, pages 172–177, Jun 2005.

[93] K. Wang, J. Parekh, and S. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Advances in Intrusion Detection*, pages 226–248. Springer, 2006.

[94] C. Wressnegger, G. Schwenk, D. Arp, and K. Rieck. A close look on n-grams in intrusion detection: Anomaly detection vs. classification. In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, AISec '13, pages 67–76, New York, NY, USA, 2013. ACM.

[95] S. Zander, T. Nguyen, and G. Armitage. Automated traffic classification and application identification using machine learning. In *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)l*, pages 250–257, Nov 2005.

[96] L. Zhuowei, A. Das, and S. Nandi. Utilizing statistical characteristics of n-grams for intrusion detection. In *Proceedings. 2003 International Conference on Cyberworlds*, pages 486–493, Dec 2003.

[97] T. Ziermann, S. Wildermann, and J. Teich. Can+: A new backward-compatible controller area network (can) protocol with up to 16 higher data rates. In *2009 Design, Automation Test in Europe Conference Exhibition*, pages 1088–1093, April 2009.

[98] A. Zimek and E. Schubert. Outlier detection. *Encyclopedia of Database Systems*, pages 1–5, 2017.