



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Framework for Benchmarking the Intrusion Detection Systems of the Controller Area Networks

Thesis of the Master degree in Telecommunication engineering

Politecnico di Milano, DEIB
and
Universitat Politècnica de Catalunya, ETSETB

Author:

Teresa Costa Cañones

Advisors:

Stefano Zanero
Oscar Esparza

MAY 2021

Abstract

As in many other ecosystems, in the transportation of persons or goods, the three basic goals of the journey are security, safety, and low costs. Modernizing the manner of traveling has forced the trade-off between the three goals to be challenging to achieve. The value of human lives is essential, so the terms of security and safety should have major significance, but the automotive market still requires vehicles to be affordable. The development of new functionalities and the increase of connectivity in the vehicle provoke an increase of the attack vectors, which enable attackers to access the vehicle's internal networks, and in particular, the Controller Area Network (CAN), where the majority of safety measures are connected. For this reason, researchers analyze the threats of the automotive world, and design new countermeasures for them. To ensure that such countermeasures are efficient, safe, and secure, they need to be carefully tested and analyzed. To aid researchers towards this objective, in this thesis we develop a framework to benchmark Intrusion Detection Systems (IDS) for CAN. We propose eight different IDS designs from the state of the art to test attacks against. Alongside, we designed a tool capable of synthesizing the majority of CAN attacks in the state of the art, and two further evasion attacks, in order to study their capability of bypassing the IDSs. To propose a benchmark of the capability of the IDSs, we built and test them against crafted attacks, and we compare their performance results. Our results highlight multiple insights on the capability of CAN IDSs and attacks, such as the stronger detection capability of IDSs that are based on time sequences and the effectiveness of the evasion attacks against part of the IDSs.

Abstract (Spanish)

Como en muchos otros contextos, en el transporte de personas o mercancías, los tres objetivos básicos del sector son la seguridad, la protección y los bajos costes. La modernización de la forma de viajar ha obligado a que el compromiso entre los tres objetivos sea un desafío. El valor de las vidas humanas es esencial, por lo que los términos de seguridad y protección deberían tener una gran importancia, pero el mercado automovilístico aún requiere que los vehículos sean asequibles. El desarrollo de nuevas funcionalidades y el aumento de la conectividad en el vehículo provocan un aumento de los vectores de ataque, que permiten a los atacantes acceder a las redes internas del vehículo y, en particular, a la Controller Area Network (CAN), donde se encuentran conectadas la mayoría de las medidas de seguridad. Por esta razón, los investigadores analizan las amenazas del sector del automóvil y diseñan nuevas contramedidas para ellas. Para garantizar que tales contramedidas sean eficientes, seguras y protegidas, deben probarse y analizarse cuidadosamente. Para ayudar a los investigadores a alcanzar este objetivo, en esta tesis desarrollamos un marco para comparar los sistemas de detección de intrusiones (IDS) para CAN. Proponemos ocho diseños de IDS diferentes extraídos del estado del arte para probar los ataques. Paralelamente, diseñamos una herramienta capaz de sintetizar la mayoría de los ataques CAN del estado del arte, y dos ataques más de evasión, con el fin de estudiar su capacidad de eludir los IDS. Para desarrollar un marco de comparación de la capacidad de los IDS, construimos los IDS, los probamos contra ataques diseñados, y comparamos los resultados de su rendimiento. En nuestros resultados destacan múltiples conocimientos sobre la capacidad de los IDS CAN y los ataques, como la capacidad de detección más sólida de los IDS que se basan en secuencias de tiempo y la eficacia de los ataques de evasión contra una parte de los IDS.

Abstract (Catalan)

Com en molts altres contextos, en el transport de persones o mercaderies, els tres objectius bàsics del sector són la seguretat, la protecció i els baixos costos. La modernització de la forma de viatjar ha obligat que el compromís entre els tres objectius sigui un desafiament. El valor de les vides humanes és essencial, de manera que els termes de seguretat i protecció haurien de tenir una gran importància, però el mercat automobilístic encara requereix que els vehicles siguin assequibles. El desenvolupament de noves funcionalitats i l'augment de la connectivitat en el vehicle provoca un augment dels vectors d'atac, que permet als atacants accedir a les xarxes internes de el vehicle, i en particular, a l'Controller Area Network (CAN), on es troben connectades la majoria de les mesures de seguretat. Per aquesta raó, els investigadors analitzen les amenaces de el sector de l'automòbil i dissenyen noves contramesures per a combatre-les. Per garantir que aquests contramesures siguin eficients, segures i protegides, s'han de provar i analitzar acuradament. Per ajudar als investigadors a assolir aquest objectiu, en aquesta tesi desenvolupem un marc per a comparar els sistemes de detecció d'intrusions (IDS) dissenyats per CAN. Proposem 8 dissenys d'IDS diferents extrets de l'estat de l'art per provar els atacs. Paral·lelament, dissenyem una eina capaç de sintetitzar la majoria dels atacs CAN de l'estat de l'art, i dos atacs més d'evasió, per tal d'estudiar la seva capacitat d'eludir els IDS. Per desenvolupar un marc de comparació de la capacitat dels IDS, construïm els IDS, els provem contra atacs dissenyats, i comparem els resultats del seu rendiment. En els nostres resultats destaquen múltiples coneixements sobre la capacitat dels IDS CAN i els atacs, com la capacitat de detecció més sòlida dels IDS que es basen en seqüències de temps i l'eficàcia dels atacs d'evasió contra una part dels IDS.

Acknowledgments

I would like to express my gratitude to my advisors S. Zanero, M. Carminati, S. Longari, and O. Esparza.

I would like to thank also to the contributors of a the CANTack, which is a part of the thesis, C. A. Pozzoli and A. Nichelini.

Last but not least, I would like to thank my family for their encouragement and support during my studies.

Contents

Abstract	3
Acknowledgments	9
1 Introduction	19
2 Background and motivation	21
2.1 Controller Area Network bus	21
2.1.1 CAN properties	21
2.1.2 Security CAN vulnerabilities and threats	24
2.1.3 Countermeasures against security CAN threats	25
2.2 Machine Learning for CAN IDS	28
2.2.1 Algorithms for CAN anomaly IDS	28
2.2.2 CANnolo	30
2.2.3 Evaluation of the IDS	30
2.3 Reverse Engineering of Automotive Data frames	31
2.4 Adversarial Machine Learning	33
2.4.1 Generative Adversarial Network	33
2.4.2 Heuristics	35
2.5 State of the art	36
2.6 Motivation	36
3 Threat Model	39
3.1 Attacker’s motivation and goals	39
3.2 CAN Security Violations	40
3.3 Types of CAN attacks	40
3.4 Attacking the IDS	41
3.5 IDS Attacker’s knowledge	42
3.6 Attacker’s capabilities	42

4	Approach and implementation	45
4.1	Normal CAN dataset	45
4.2	Synthesize basic CAN attacks	46
4.2.1	Crafting attacks	46
4.3	Intrusion Detection System design	47
4.4	Synthesize sophisticated attacks	52
4.4.1	Design of WGAN attack	52
4.4.2	Design of Bit BIM Variant attack together with WGAN	53
4.5	Approach summary	55
5	Results of the experiments	57
5.1	IDS performance with basic CAN attacks	57
5.1.1	Experiments with Fuzzy Attack	57
5.1.2	Experiments with Replay Attack	58
5.1.3	Experiments with Multiple Attack	59
5.2	IDS performance with sophisticated attacks	59
5.2.1	IDS performance with WGAN attack	59
5.2.2	IDS performance with WGAN and Bit BIM variant attack	60
5.3	Experimental evaluation summary	61
6	Conclusions	63

List of Figures

2.1	CAN structure	22
2.2	CAN data frame standard format. Figure extracted from [55, 8].	23
2.3	CAN data frame in extended format. Figure extracted from [55, 8].	23
2.4	Graphical explanation of reactive and proactive countermeasures, extracted from [5].	25
2.5	Location of IDS in CAN.	26
2.6	Types of CAN IDS, extracted from [1].	27
2.7	Graphical performance of a Isolation Forest.	29
2.8	Graphical performance of a OCSVM.	29
2.9	Schematic performance of a generic auto-encoder.	30
2.10	Specifications of the CANnolo IDS.	30
2.11	Schematic performance of a generic GAN.	34
4.1	IDS design process.	48
4.2	Working flow of the created IDS.	50
4.3	Graphical general workflow of Bit BIM variant.	53

List of Tables

4.1	Parameters of models used for IDS.	49
4.2	Relation between input and output auto-encoders.	50
4.3	Parameters of network models used for IDS.	51
4.4	WGAN hyper-parameters.	52
4.5	Time-series WGAN hyper-parameters different from WGAN hyper-parameters.	53
5.1	Performance results of the IDS with Fuzzy attack.	58
5.2	Performance results of the IDS with of Replay attack (2-15 packets for the pattern).	58
5.3	Performance results of the IDS with of Replay attack (35-50 packets for the pattern).	59
5.4	Performance results of the IDS with of Multiple Attack.	59
5.5	Percentage of packets bypassed through the IDS with WGAN.	60
5.6	Percentage of packets bypassed through the IDS with Bit BIM variant together with non-time-series WGAN.	60
5.7	Percentage of packets bypassed through the IDS with Bit BIM variant together with time-series WGAN.	61

Acronyms

ABS	Anti-lock Bracking System
ACK	Acknowledgement
BIM	Basic Iterative Method
CAN	Controller Area Network
CRC	Cyclic Redundancy Checksum
D	Discriminator
DBC	Data Base Communication
DLC	Data Length Code
DoS	Denial-of-Service
ECU	Electronic Control Unit
EOF	End Of Frame
FGSM	Fast Gradient Sign Method
FID	Fréchet Inception Distance
FN	False Negative
FP	False Positive
G	Generator
GAN	Generative Adversarial Network
GRU	Gated Recurrent Unit
ID	CAN Identification
IDS	Intrusion Detection System
IF	Isolation Forest
IS	Inception Score
LIN	Local Interconnection Network
LSTM	Long-Short Term Memory
MI	Message Injection Attack
ML	Machine Learning
MOST	Media Oriented Systems Transport
NN	Neural Network
OBD	On Board Diagnostics
OCSVM	One-Class Support Vector Machine
READ	Revers Engineering of Automotive Data frames
RPM	Revolutions Per Minute
RTR	Remote Transmission Request
SOA	Start Of Frame
SRR	Substitute Remote Request
TN	True Negative
TP	True Positive
WGAN	Wassertein Generative Adversarial Network

Chapter 1

Introduction

When K.Benz invented cars in the XIX century, they were only mechanical engines. The evolution of technology conveyed modern vehicles to not having mechanical components only. Vehicles contain electronic devices that the engineers craft to fulfill different new functionalities. These recent utilities help the driver to have a safer and comfortable journey (e.g., auto park, airbag). An Electronic Control Unit (ECU) is a microcontroller that has sensors and actuators. Each ECU performs a specific simple functionality, and in the case of more complex functions, the functionality needs cooperation between different ECUs. We can classify the ECUs into five classes depending on their functionality: power-train, vehicle safety, comfort, infotainment, and telematics[37].

One example of such specific functionality could be the Anti-lock Braking System (ABS) that we categorize in the type of safety ECU. The information used for taking a concrete action can be acquired by its sensors or by data sent by other ECUs. For that reason, ECUs need to be connected and coordinated to achieve good performance of the vehicle. The easiest solution is to build one or more networks to connect the ECUs and then connect the networks between them. There are five types of in-vehicular networks: Local Interconnection Network (LIN), Controller Area Network (CAN), FlexRay, Ethernet, and Media Oriented Systems Transport (MOST). The network most used is the CAN due to the cheapness and real-time capabilities[56].

At the time of the invention of CAN, 1983, the inventor did not expect that such a network could be exterior accessible. Therefore the design of CAN protocol does not cover security issues. The reality is that an attacker can reach CAN. Various attacks were demonstrated in [30], where the evidence shows that it is possible to enter into an in-vehicle network physically and remotely. When an attacker compromises an ECU, the intruder can control the performance of the function of which that ECU was in charge. There are some types of ECU that, in the situation when an attacker can bypass them, could cause more damage than others. It is not the same to compromise the ECU in charge of the airbag or the one in control of the music. Furthermore, there are some ECUs that are more exposed to attackers than others. E.g., the ECUs related to infotainment or telematics are connected to the exterior (Bluetooth, Wifi). A CAN network connects all the ECUs, making thus the less exposed ECU also vulnerable to attacks. The cause of that is the vulnerabilities of the CAN bus[30].

In conclusion, adding more ECUs to the vehicle does not mean that we have a more secure environment[20]. Once known the possibility of having a car compromised, some countermeasures have been investigated and applied. There are two types of countermeasures: proactive and reactive ones. Although both are not mutually exclusive, our thesis focuses on the second one because of its simpleness. One of the most used reactive countermeasures is the Intrusion Detection System (IDS), which detects the attacks once they occur in CAN.

Before launching an IDS, the vendor needs to test the countermeasure for assuring a satisfactory performance of it. For testing the IDS, the researchers check that it can detect the attacks, and then researchers test methods for bypassing the countermeasure.

We have not found other general frameworks for benchmarking various types of IDS in the automotive sector. Furthermore, the seen methods for testing the performance of the IDS only focus on detecting some attacks. In the current work, we aim to manufacture an attack database for using it to test future developed IDS. To the best of our knowledge, there is no free tool to synthesize the attacks in the CAN bus neither to prove that the techniques used for building IDS are good enough. For that reason, the contributions of this thesis to the scientific community are:

- The construction of a framework to assess the detection capabilities of CAN IDS, by comparing them with the most common detection techniques.
- The development of a framework to craft CAN attack datasets from real-world CAN logs. We call the previous mechanism the attack tool (CANtack).
- The development of traditional and adversarial attacks as a mean of testing the IDSs proposed above.

In Chapter 2 we explain the necessary information background for understanding our work and our motivation. Then, we continue with the Threat Model in Chapter 3. In Chapter 4 we explain the approach and the implementation of the thesis. In Chapter 5 we show the results of the experiments made. Finally, in Chapter 6, we explain the limitations, the conclusions, and the future work.

Chapter 2

Background and motivation

In the current chapter, we expose the background information appropriate for understanding the thesis. Furthermore, we describe the state of the art and the motivation of the thesis.

2.1 Controller Area Network bus

In chapter 1, we have introduced the automotive world and the in-vehicle networks. CAN is the most suitable network for vehicle communications[56] and in the section 2.1.1 we explain why. The automotive sector is not the only one that uses CAN, the industrial control systems also operate with CAN because of its properties.

CAN is a serial protocol that allows the connection between different ECU in a vehicle. It was invented in 1983 by BOSCH[8]. The ISO 11898 standards[43] defines the standard protocol of this communication. The standards describe which are the features of the network and how the ECUs communicate between them. We explain the specification of CAN in the section 2.1.1.

Usually, inside a vehicle, there is more than one CAN network. The different CAN buses can operate at different data rates. Gateways connect various CAN buses. Furthermore, the gateways can act as a firewall in case needed[54].

2.1.1 CAN properties

Before the invention of CAN, all the ECUs used a point-to-point communication structure. Such a type of structure was not scalable. Every time the designer introduced extra ECUs, it was more challenging to connect all the ECUs between them. Furthermore, the designer needed a lot of wires, which increased the cost of the structure. CAN simplify this structure using the bus structure[36], and made it with lower cost. We have depicted the CAN bus in Figure 2.1, where we can observe that CAN connects the ECUs more simply.

CAN is a message oriented communication. We define these messages as frames. CAN also makes the communication in a fully centralized manner and efficiently by using a broadcast environment. When an ECU wants to send data, it sends a frame to the bus, and then the other ECUs in the network decide to process it or not. In this manner, the ECU only needs to send the message once. The CAN frame does not have a specific destination in such a way ECU individually filters the frames depending on the identification (ID) of the packet. We describe the communication protocol and the CAN frames in section 2.1.1.1

We can define CAN as a robust protocol because BOSCH designed the CAN bus to operate in

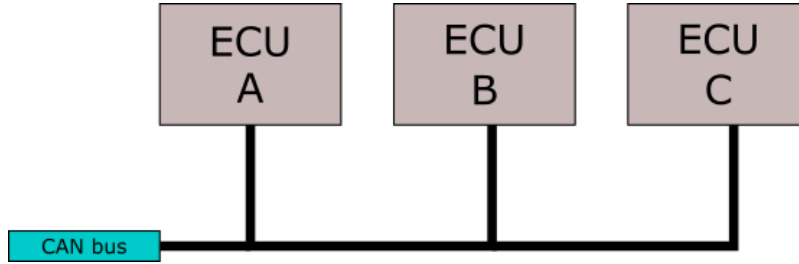


Figure 2.1: CAN structure

noisy environments. To achieve robustness in a broadcast environment, CAN has a non-destructive contention-based arbitration network. CAN performs the arbitration by using priorities. The priorities depend on the CAN ID number of the packet, which refers to the specific ECU sending the information. The lower ID has more preference than the higher ID. Furthermore, CAN uses error detection and error signaling. CAN performs error detection by writing different checks in the packet's transmission (Cyclic Redundancy Checksum). In addition, during the communication, the transmitter listens to the message that is sending to be aware of possible errors. After, in case the receivers find an error, they answer the received frame rewriting the acknowledgment field (see section 2.1.1.1). In case of having an error in a frame, the system automatically retransmits the packet damaged. CAN can distinguish between temporary errors and permanent failures, and in the case of the second one, the system disconnects the target ECU.

Summarizing, the general properties of CAN are: simple, low cost, fully centralized, extremely robust and efficient.

It is worth mentioning that the bit encoding uses the zero dominant and the one recessive. It is to say, when the node wants to transmit a 0 it puts voltage, whereas when the node transmits a 1, it does not transmit voltage. From the procedure, we can also deduct the ID priority schema.

2.1.1.1 Types of frames

In all the existing communications, there are protocols and rules for sending the information. There is a common need to fix the manner of sending the information through the bus. For that reason, the standards define the different frames of the message oriented communication protocol. The ECUS connected in CAN communicates between them using CAN frames. The types of CAN frames described in [8] are:

- **Data frames:** The Data frames carry the data from the ECUs.
- **Remote frames:** The Remote frames ask for data of a specific ECU. The ECU sends the Data frame after receiving the Remote frame with its ID.
- **Error frames:** An ECU sends an Error frame when it detects an error in the bus. The ECUs that have received the error send the Error frame together. The Error frame is an announcement to the other ECUs of the error.
- **Overload frames:** To avoid reaching the maximum bus speed, CAN use Overload frames to delay other Data frames or Remote frames. The ECU sends the Overload frame after the Error frame.

It is worth mentioning that before an ECU transmits a Data frame or a Remote frame, it needs to wait for an interframe space. CAN uses the interframe space to avoid collisions.

In our work, we focus only on the Data frames because those are the ones that the ECUs use

during normal driving. When Bosch invented CAN, the vehicle did not need to support a lot of different ECUs. For that reason, the field which determines the identification of the ECU was 11 bit. When the number of ECU increased, they extended the bits identification and also the frame. For that reason, there are two types of data frames depending on the identification extension: the standard format and the extended one. As the difference between standard and extended is only the inclusion of new fields in the extended format, we explain only the second format, but we depict both in Figure 2.2 and Figure 2.3. We can observe the extended CAN data frame in Figure 2.3.

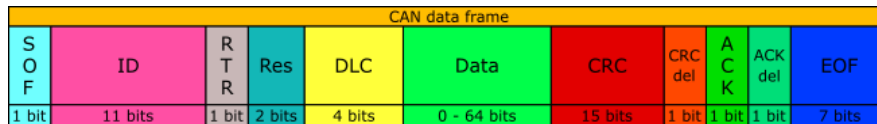


Figure 2.2: CAN data frame standard format. Figure extracted from [55, 8].

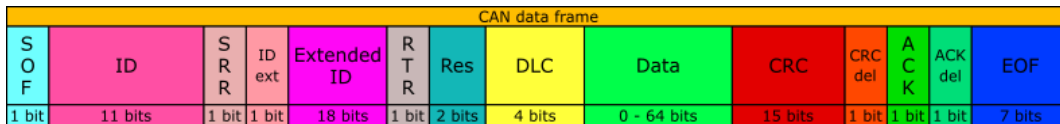


Figure 2.3: CAN data frame in extended format. Figure extracted from [55, 8].

The fields of the extended CAN data frames are:

- **Start Of Frame (SOA)**: The start of the frame is a dominant bit used for starting the communication. The ECUs can only start transmitting when the bus is free during the interframe space.
- **ID**: The ID represents the type of the message, i.e., which functionality is transmitting. The ID represents a name or direction of the ECU, and must be unique. The size is 11 bits in the standard format. CAN establishes the priority message by this field: a lower value indicates a higher priority.
- **Identifier extension bit (ID ext)**: In the standard frame it is transmitted dominant whereas in the extended is transmitted recessively.
- **Substitute remote request (SRR)**: The SRR is a recessive bit and substitutes the RTR in extended IDs. This field does not exist in the standard frame.
- **Extended ID**: This is the extension of the ID in case of the extended frame. The extension is 18 bits. This field does not exist in the standard frame.
- **Remote Transmission Request (RTR)**: CAN protocol uses this bit to distinguish between Remote frame and Data frame. This bit is dominant in Data frame. In the standard frame, the RTR follows the ID in transmission.
- **Reserved bits (Res)**: CAN reserves this field for further applications. It contains 2 bits that should be dominant. The ECUs can validate the message in case of being recessive.
- **Data Length Code (DLC)**: establishes the length of the bytes inside the data field. DLC is a 4-bit field, and due to that can range between 0 to 8. Thus, it is a control field.
- **Data**: The payload is the data that the functionality wants to convey. The data of the frame can be between 0 bits and 64 bits. Because of the length variability, the CAN protocol uses the data length code (DLC).

- **Cyclic Redundant Code (CRC):** CAN uses CRC. ECUs use CRC to detect the error regarding the message transmission. CRC field is 16 bits, and it contains the CRC sequence from the SOF to the Data Field.
- **CRC delimiter (CRC del):** CRC delimiter is a recessive bit.
- **Acknowledge (ACK):** ACK is used to get the confirmation from the receiver node regarding the proper reception of the CAN message. The ECU that transmits the frame sends the field as a recessive bit. The ECUs that receive the frame correctly, i.e., the CRC is correct, superscribe a recessive bit being then dominant. Otherwise, the transmitter sends the CAN message again.
- **ACK delimiter (ACK del):** ACK delimiter is a recessive bit.
- **End of Frame (EOF):** This is a flag field. It consists of 7 recessive bits.

In our work, we are going to focus on the ID and payload fields. The reason is that those fields are the ones that characterize communication. For knowing the payload length, we also use the DLC.

The Communication Database, a.k.a., the DBC file, specifies the format of the data. The DBC file also describes other features like the time cycle of a specific message or the ID of each ECU. With the DBC file, the ECUs decide from which ID frames they filter. The vendors try to maintain secret the DBC file to make it more difficult for an attacker to inject meaningful messages, but, It is not possible to defend a system only by having the protocol in secret or not being public.

2.1.2 Security CAN vulnerabilities and threats

The CAN properties make such protocol the most suitable technology for vehicle communication. The problem is that such properties lead to security vulnerabilities.

The first vulnerability is that it is easy to reach the CAN bus. There are different manners to access the CAN bus: physically or remotely. One physical example is using On-Board Diagnostics. On-Board Diagnostics (OBD) is a computer-based system that can access CAN by a physical method. The main goal of OBD is to show emissions and some troubleshooting. The OBD is compulsory in some countries. The attacker could perform her attack in two different manners: attaching a device in the OBD-II simulating another ECU or inserting malware that affects some ECU. On the other hand, remote accessing examples are via Wi-Fi or Bluetooth between others. As the technology progresses, exists more ECU connected to the exterior, which causes an increase of potential threads.

Another vulnerability is that CAN does not use encryption in its communication[57]. If CAN had adapted encryption in its communication, the protocol would not fulfill the principles of simplicity and low cost. Once the attacker reaches the CAN bus, she can sniff the traffic of the network. Such vulnerability can generate other potential threats. Combining the previous vulnerability with the property of a broadcast environment, the attacker can read the entire communication once in the network. For that reason, CAN lacks confidentiality.

In addition, CAN bus does not have authentication either authorization[54]. It is easy to understand that these two characteristics make it an attacker easy to inject messages on the network or spoof the ID of a specific ECU. Furthermore, there are no data integrity checks. The ECUs can ignore invalid messages. In case the invalid frame does not have an incorrect CRC, there is no error in the network derived from an invalid frame. The [8] standard explains how to compute the CRC, so it is easy that the attacker sends messages without being detected. The vendors try to make it more challenging to know the data structure by hiding the DBC file, but this measure is not enough for avoiding attacks.

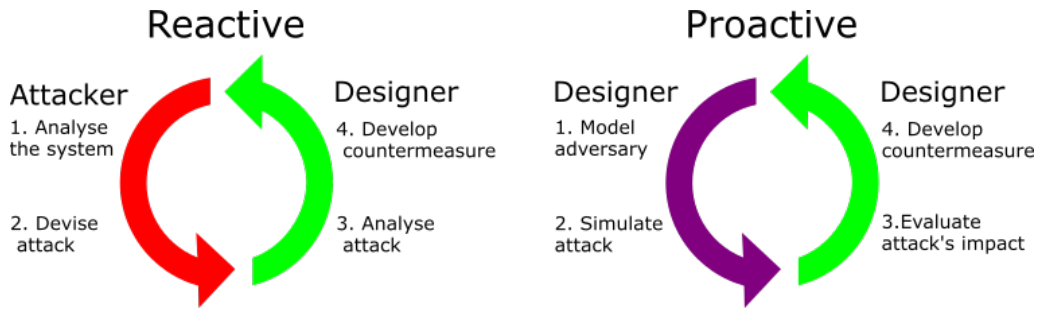
Finally, combining all the previous lacks, it is easy to perform a Denial of Service attack[20]. As there is no authentication, neither authorization, and combined with the broadcast environment and the priority arbitration, the attacker can constantly inject messages with the higher priority ID. In this manner, the other ECUs would not be able to transmit any message.

It is worth mentioning that the attacker cannot manipulate all the vehicles' functionalities by the CAN bus. There is no certainty that in case an attacker spoofs a packet, it has some effect on the behavior of the vehicle. It depends on where are located the sensors and the actuators. If the actuator performs the action depending on the sensor data, and both are located in the same ECU, there is no need to send packets about sensor data acquired. Thus, the ECU sends only readable packets.

2.1.3 Countermeasures against security CAN threats

In order to defend against the threats, researchers have built different countermeasures. The countermeasures can be classified between proactive countermeasures and reactive countermeasures. In the Figure 2.4 can be seen a graphic which summarizes both types.

Figure 2.4: Graphical explanation of reactive and proactive countermeasures, extracted from [5].



The proactive countermeasures are those that are done in order to avoid the attack (a.k.a, active countermeasures). For such type of countermeasures, the designer of the system tries to attack her own system and then finds the best solution to defend against the attacks. Examples of the current type of measures are: securing remote end points, change the network architecture, use message cryptography, etc. On the other hand, the reactive countermeasures are those actions done once the attack is detected (a.k.a, passive countermeasures). The work-flow of the aforementioned countermeasures is to detect the attack, and develop the defenses after detecting it. In this case the most typical measure is the Intrusion Detector System (IDS).

The measures most used in the CAN are the reactive countermeasures[1]. The reason is because these measures are easier to implement. Proactive measures require more time and power to develop them, because the system needs to be analyzed before trying to attack it. Furthermore, the proactive countermeasure could mean a change of the whole system as in the case of encryption. In addition, it could happen that the designer does not find some attack, so it is also needed to have reactive measures in the system. It is not feasible to spend such amount of power and time into proactive countermeasure, if the reactive ones would need to be developed anyways. Following the previous reasoning, in the current work we use the IDS as a countermeasure.

2.1.3.1 CAN Intrusion Detection Systems

The CAN Intrusion Detection Systems (IDSs) are structures that aim to detect when an attack occurs on the in-vehicle network. Once the IDS detects an intrusion, it sends a message to the user to alert him. We can locate the IDS in various parts of the CAN structure, categorizing them

into centralized or distributed IDS structures. We can compare the two location options in Figure 2.5.

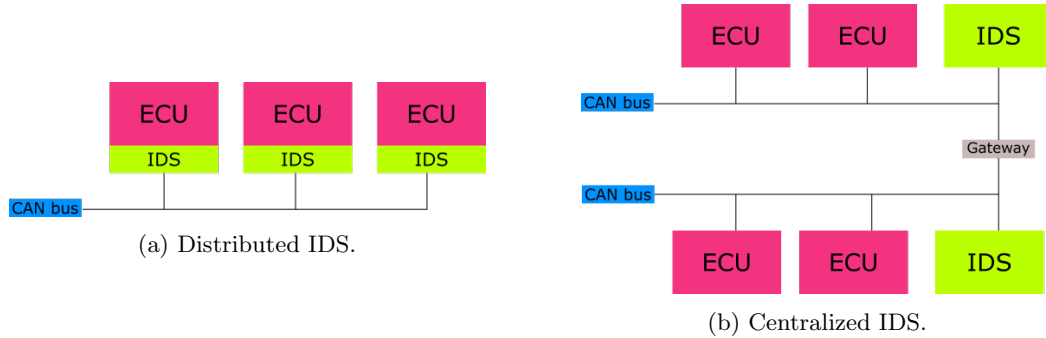


Figure 2.5: Location of IDS in CAN.

In the decentralized structure, we place the IDS inside each ECU. In this manner, the IDS would analyze only the packets received by the specific ECU. It is worth mentioning that the ECU has low computational resources, and we need to adapt the IDS to this requirement. The previous condition makes the building of an IDS a challenging task. We represent distributed structure in Figure 2.5a. Having the IDS in a distributed manner has advantages and disadvantages. The advantage is that we can know which ECU the attacker has compromised. Furthermore, the structure is scalable from the IDS point of view because we put as IDS as ECUs you want to protect in the network. The disadvantage is that each ECU performs all the operations, duplicating thus the same actions. Furthermore, we should also implement an IDS manager. For that reason, the distributed IDS structure is computationally expensive. On the other hand, the IDS can be another different ECU located on the same bus. Thus, the IDS can analyze all the transmitted frames of the bus. In the circumstance of having more than one bus connected through a gateway, we would put one IDS per bus. Otherwise, the IDS can only analyze the frames of one bus. We represented an IDS centralized structure in Figure 2.5b. The centralized IDS structure has the advantage of using only the resources once per bus, not once per ECU. The disadvantage is that the IDS cannot detect how many ECUS is affecting the attack. Furthermore, in case the attacker bypasses the IDS, it is not possible to detect any attack. Another disadvantage is that it is more difficult to scale from the IDS point of view. The reason is that as more ECUs, the more information the IDS needs to analyze.

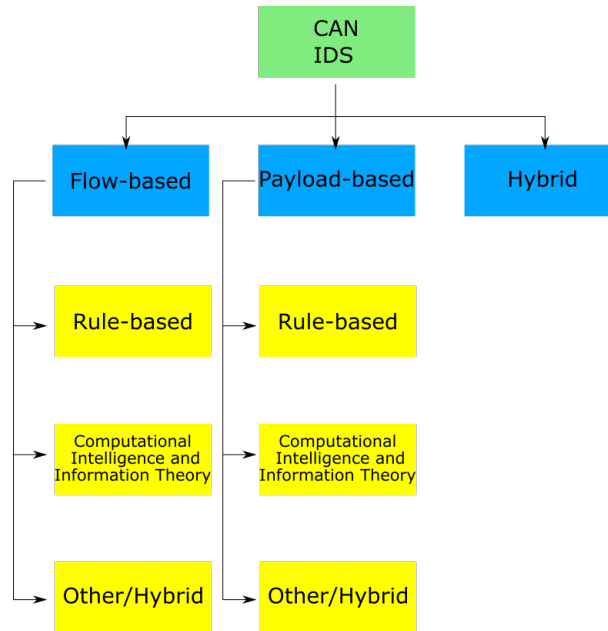
We can categorize the IDS depending on which features analyze and the methods for analyzing them. We can see the classification type structure of the CAN IDS in Figure 2.6.

The first division in Figure 2.6 categorizes the IDS depending on which type of features analyzes. We can distinguish three big groups of IDS:

- **Flow-based:** The flow-based IDS considers the behavior of the frames. The main feature that Flow-based IDS analyzes is the time-frequency of packets, but it could also analyze other physical features. The flow-based IDS does not analyze the payload content.
- **Payload-based:** The payload-based IDS takes the decision only depending on the data content of the payload. It does not consider the behavior of the frames.
- **Hybrid:** The hybrid mixes the two types of IDS. Hybrids are more able to detect more types of attacks.

The following categorization depends on how the IDS analyzes the features, and the algorithms used for detecting an intrusion. The methods are:

Figure 2.6: Types of CAN IDS, extracted from [1].



- **Rule-Based:** IDS can detect anomalies by using statistical techniques and rule-based techniques. Usually, there are a lot of different rules, and the designer describes them manually. For that reason, the implementation of such IDS is time-consuming. Furthermore, it is unlikely that a Rule-based IDS can detect novel attacks. In the case of being flow-based, the IDS can identify only the intrusions that affect periodic messages. One different example of a rule-based IDS related to payload-based is [32], where only uses the Mahalanobis distance between data frame payloads for detecting the intrusion. Rule-based IDS have limiting capabilities because it is not able to identify all the known attacks. Furthermore, once the attacker knows the rules of the IDS, the attacker can bypass it easily.
- **Computational Intelligence:** Those are the IDS that use Machine Learning (ML) for detecting intrusions. The Computational Intelligence IDS are the most used in actual times because of the considerable ability to learn patterns.
- **Information theory:** Those are based on classifying the dataframes with its bits uncertainty i.e., on the entropy (uncertainty of data). They are not commonly used.
- **Others or Hybrid:** Those IDS use two or more of before types. Hybrids are more able to detect more types of attacks.

All the previous IDS have different pros and cons. Some of them have limited capabilities for some type of attack, whether others need a lot of computational resources. We need to consider a wide range of different thread models that include all the attack possibilities. The IDS type that has a higher accuracy taking into account the diverse attacks is the Computational Intelligence type. Furthermore, we are going to do a payload-based IDS to detect more complex attacks.

It is worth mentioning that the IDSs are passive countermeasures. After detecting the attack, we should make an active countermeasure to protect the vehicle.

2.2 Machine Learning for CAN IDS

We can categorize the Machine Learning (ML) IDS into two categories depending on the detection technique: the signature-based and the anomaly-based. The signature-based compares the CAN behavior with the previously stored knowledge. For that reason, it can only identify the already seen attacks. The advantage of signature-based IDS is that the detection accuracy is high. It is worth mentioning that the designers of signature-based IDS need to update them with all the known attacks. On the contrary, the anomaly-based IDS studies the normal operation of the system. In case there is an outlier or deviation of the behavior, it detects an attack. Anomaly-based IDS can detect novel attacks. Another advantage of the Anomaly-based IDS is that it needs less memory than the signature-based IDS. The disadvantage of the anomaly-based IDS is that it has lower accuracy than signature-based IDS. The algorithms commonly used for building IDS can be supervised or unsupervised models. Supervised models are those that train the model by using the two categories: normal and attack data. Whereas unsupervised models use only normal data for training the models. Following the same rationing as before, the signature-based IDS use supervised algorithms whereas anomaly-based uses un-supervised algorithms. There is also the possibility of having a hybrid IDS which uses both methods. In our work, we focus on anomaly IDS because it is difficult to find an authentic CAN attack database[35].

2.2.1 Algorithms for CAN anomaly IDS

The IDS aims to detect intrusions in the CAN system. An intrusion is an occasion when someone goes into a place or situation where they are not wanted or expected to be [52]. Usually, an intrusion in CAN implies a deviation of the usual behavior of the system. The issue is that the opposite is not always correct. Not all the anomalies are intrusions. During the vehicle's journey, it can occur, e.g., the driver brake abruptly when appearing suddenly a pedestrian. The IDS should face the challenge of distinguishing between an anomaly and an intrusion.

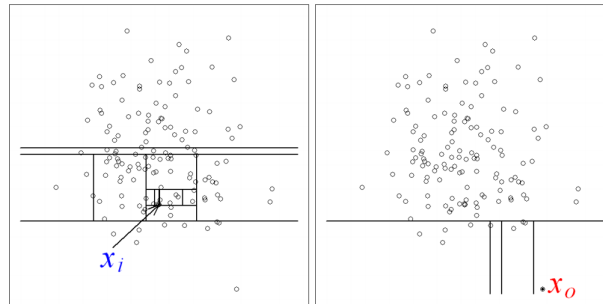
The final goal of the anomaly IDS is to know when there is a peculiarity inside the normal data, also known as an anomaly. For achieving the objective, it is essential to acquire the data and identify the normal behavior using some computations. We explain the models used further in chapter 4: one-class support vector machine, isolation forests, auto-encoders, and CANnolo. The first three are general algorithms that we can use for other environments, whereas the last one is an algorithm created just for CAN IDS.

2.2.1.1 Isolation Forests

The Isolation Forest (IF) is an anomaly isolation model based on isolation trees[25]. The method consists of dividing the samples until all are isolated, and then the samples which need fewer partitions to be isolated (shorter path) are classified as anomalies. IF makes the partitions by choosing a threshold in a feature. The method randomly determines the feature to split the group of samples. We can see a graphical view of what does the Isolation Forest algorithm in Figure 2.7 extracted from [25]. In Figure 2.7 the sample x_i is classified as a normal sample. Whereas the sample x_0 is classified as an anomaly. We can observe that the sample x_0 needs fewer partitions to be isolated than the sample x_i .

Isolation Forest is a suitable method for classifying large datasets with a vast amount of features. In case that we want to have a model able to detect anomalies in time series, we need to transform the input data.

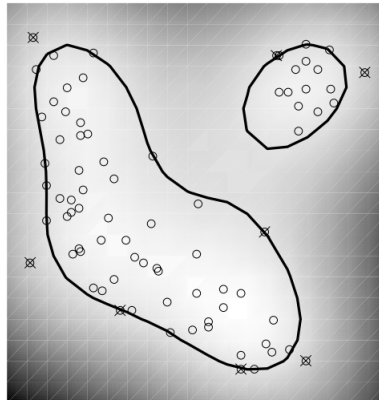
Figure 2.7: Graphical performance of a Isolation Forest.



2.2.1.2 One-Class Support Vector Machine

One-Class Support Vector Machines (OCSVM) are an unsupervised classification method between two classes. It classifies the data between a normal class, that is inside a set of samples S , or outside this set by approximating a function f [46]. In other words, the function f tries to define a boundary between the training data-set and the outliers.

Figure 2.8: Graphical performance of a OCSVM.



OCSVM are linear classifiers but can use non-linear kernels to have a non-linear f . In Figure 2.8 extracted from [46] it can be seen the performance of an OCSVM with a non-linear kernel. In Figure, it can be seen the estimated function f as the line. The samples classified as normal are the circles inside the boundaries. The anomalies are the crossed circles outside the boundaries of the line f .

2.2.1.3 Auto-encoders

Auto-encoders are unsupervised neural networks that learn how to encode and then decode data [3]. During the training, the auto-encoder learns which are the features that summarize better the data (compression). Then, it tries to reconstruct the compression with a sample that is similar to the input (decompress). In 2.9 we can see a general scheme of the auto-encoders adapted to the automotive world.

Usually, the encoder and the decoder are both neural networks. To have a meaningful representation of data, the hidden layers of the encoder need to be of a lower dimension than the data. Otherwise, the auto-encoder learns to copy the input into the output (the identity function). There is a trade-off when choosing the dimensions of the encoder because, on the one hand, we want a

Figure 2.9: Schematic performance of a generic auto-encoder.

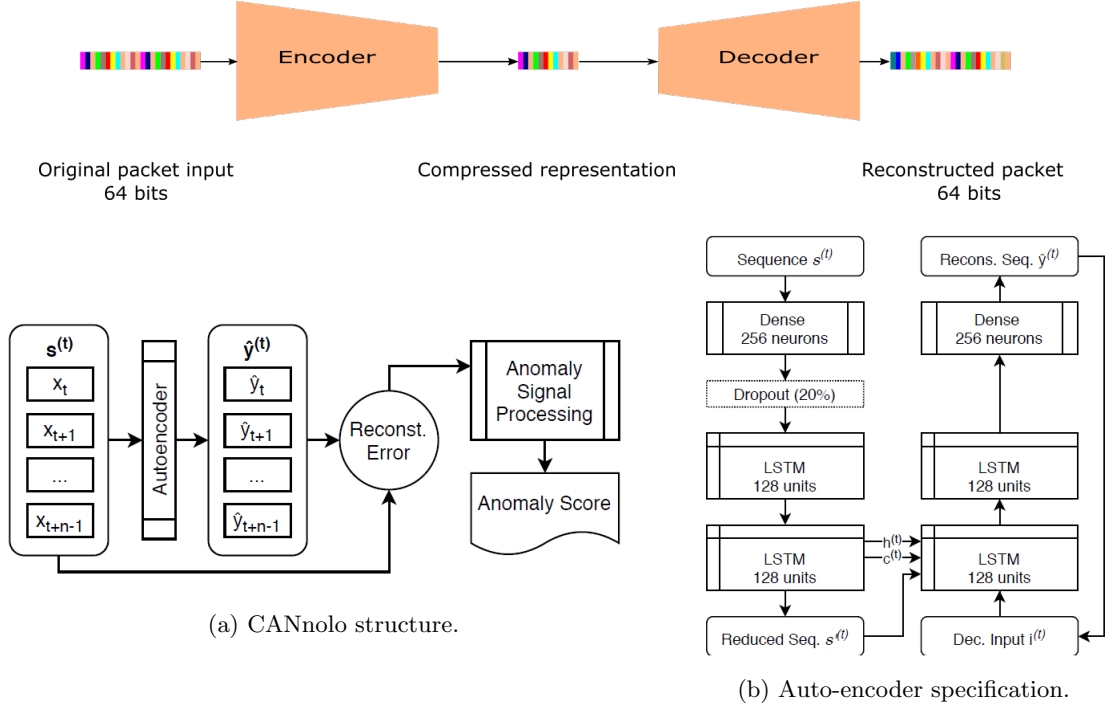


Figure 2.10: Specifications of the CANnolo IDS.

low dimension to learning only the most relevant features. On the other hand, we need that the structure manages to reconstruct the data.

2.2.2 CANnolo

CANnolo is a CAN IDS based on an LSTM auto-encoder[26]. CANnolo is an unsupervised anomaly detector. Once we have trained CANnolo, it takes the last 40 packets as the input and tries to reconstruct it. After, it computes the distance between the input and the output. In case the distance is greater than a threshold, it marks the last packet as an intrusion. We can see the aforementioned structure in Figure 2.10a. We can see the specifications of CANnolo auto-encoder in Figure 2.10b.

In addition to specifications explained on Figure 2.10b, it uses the ADAM optimization algorithm instead of the classical stochastic gradient descent procedure to update the network weights. Thus, avoiding the vanishing problem. Moreover, it also uses binary cross-entropy.

The problem with the CANnolo IDS is that it exceeds the computational requirements that the ECU has, making it not feasible in a real environment.

2.2.3 Evaluation of the IDS

There are different metrics to evaluate the performance of the ML models. In this section, we are going to explain only those that we have used in the thesis. Those metrics are extracted from [41].

In the current work, we build a model which can classify between two classes: the intrusion sample (positive) and the normal sample (negative). The samples classified as true positive (TP) and true

negative (TN) are those well-classified, whereas the false positive (FP) and false-negative (FN) are those miss-classified. The used metrics are related to TP, TN, FP, and FN.

The metrics are:

- Accuracy: This is the metric that tells you how well the model can classify the samples. We cannot check the model performance with only that metric because accuracy assumes the same cost to miss-classify a sample, whereas it is positive or negative in reality[42]. The formula of the accuracy is in the Equation 2.1.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.1)$$

- Precision: Is the metric that tells which is the proportion of well predicted positive concerning all positive predicted. It is also called confidence. We can observe the precision equation in the Equation 2.2.

$$Precision = \frac{TP}{TP + FP} \quad (2.2)$$

- Recall: This is the metric that tells which is the proportion of positive well classified. It is also called sensitivity. We can see the formula of recall in the Equation 2.3.

$$Recall = \frac{TP}{TP + FN} \quad (2.3)$$

- F-measure: This is the mean between the precision and the recall. We can see the formula of the F-measure in the Equation 2.4.

$$F_{measure} = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (2.4)$$

2.3 Reverse Engineering of Automotive Data frames

Building an IDS is not an easy task, and one of the most decisive parts is to know which features to use. The choosing of the features depends on the type of IDS that we want to build. In our case, we focus on one which prevails the data integrity. For that reason, it is necessary to know which is the meaningful information contained in the payload. The structure information inside the payload follows the rules of the DBC file. The added difficulty is that the DBC file is hidden. Knowing the structure of the data frames is necessary for bounding the features for training the IDS. The Reverse Engineering of Automotive Data frames (READ)[29] is a method used for extracting the signals of the payload. The signals are the parts of the payload which has significant meaning, e.g., the steering angle. READ does not need any prior knowledge for extracting the signals. Moreover, READ is also qualified to classify the signals between three classes: Physical Value, Counter, and CRC (Checksum Redundancy Control). The Physical Value is the signal in charge of checking if the data frame is valid or not.

Before starting with the method for extracting the signals, READ needs pre-processing. We can see the in the pre-processing pseudocode in the algorithm 1. During the pre-processing, we compute the bit-flip rate and the magnitude. The bit-flip is the number of times that the bit changes from 1 to 0 or vice versa. The first step of the pre-processing algorithm is to compute the bit-flip. Then, divide the bit-flip by the total number of the analyzed packets. The result is the flip-bit rate. Then the magnitude is calculated following the formula 2.5 where MG_k is the magnitude of the k est bit, and BF_k is the bit-flip rate on the k est bit. It is worth mentioning that the magnitude of a constant bit is infinite. Otherwise, when it occurs bit-flip in all the sequences, the magnitude will be 0.

Algorithm 1 Pre-processing of READ extracted from [29].

```

1: procedure 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:      end if
11:    end for
12:  end while
13:  for ix = 0; ix < DLC; ix ++ do
14:    bitFlip[ix]  $\leftarrow$  bitFlip[ix]/payloadLen
15:    magnitude[ix]  $\leftarrow$   $\log$  bitFlip[ix]
16:  end for return bitFlip, magnitude
17: end procedure

```

$$Mg_k = \lceil \log 10BF_k \rceil \quad (2.5)$$

After the pre-processing, we perform the definition of the signals and the classification of them in two different phases. We can see the procedure for extracting the signals in the algorithm 2. The previous procedure is called the *phase1*. In the *phase1*, the bits are grouped depending on their magnitude.

Algorithm 2 Phase 1 READ extracted from [29].

```

1: procedure PHASE1(magnitude, DLC)
2:   signals  $\leftarrow$  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:       signals.add((ixS, ix - 1))
8:       ixS  $\leftarrow$  ix
9:     end if
10:    prevMagnitude  $\leftarrow$  magnitude[ix]
11:  end for
12:  signals.add((ixS, DLC - 1)) return signals
13: end procedure

```

In the *phase2*, READ can classify the previous signals between Counter, CRC, and Physical value (for more info see [29]). Although the READ classification is an interesting feature, and we could use it for deceive the IDS, we only focus on the extraction of the signals of READ.

2.4 Adversarial Machine Learning

An adversarial example is an input that is crafted in order to be misclassified by a ML model. It could be also explained as a blind spot in the model. These samples could be small variations of

the original examples, imperceptible to the human eye. On the other hand, the samples could be very different from the real examples, but could be able to deceive also the IDS. It is demonstrated in [16, 49, 6] that the majority of the models are vulnerable to adversarial samples.

The adversarial examples can be categorized depending on which step of the time process the attacker introduces the manipulated samples to the system[5]. We can classify them as evasion attacks or poisoning attacks. The evasion attacks are samples that the attacker introduces into the classifier during the test time. Those attacks only affects on the decision of the classifier. Poisoning attacks changes the training samples in order to have more missclassifications during the test time. The attacker would try to inject only few samples in order to not alert the designer. This attack is done during the training phase of the classifier. In our work we suppose that the IDS is already trained, and does not need to be re-trained.

The adversarial examples can be classified also depending on how are misclassified into two sub-groups: the targeted attacks and the untargeted attacks. The targeted attacks are those that the attacker wants to misclassify the sample with a concrete output. Whereas the untargeted attacks are those which are misclassified. In the current thesis the attack is targeted because it is wanted to be of the class normal traffic.

The adversarial examples can be created with different methods. Those methods could craft samples to be misclassified by only one model or by different models. It is to say, It is typical to attempt if the examples crafted with one model are also misclassified with another model. The aforementioned feature is called transferability [39].

In the following sections we are going to explain some evasion attacks algorithms that are able to build targeted adversarial samples.

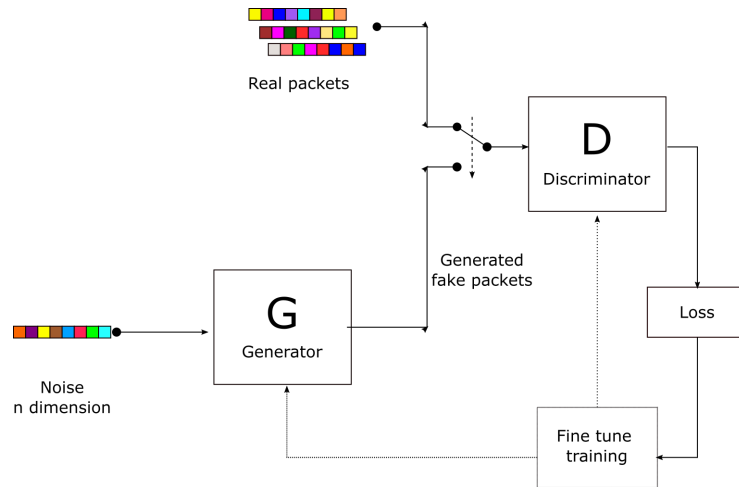
2.4.1 Generative Adversarial Network

A Generative Adversarial Network (GAN) is a generative model that learns the density probability distribution of data by a sampling mechanism. Generally speaking, a GAN is a contra-position of two models: a generative model (G) and a discriminative model (D). The discriminative model classifies samples between two classes, examples that proceed from the real dataset and fake samples crafted by the generator. On the other hand, the generative model tries to deceive the discriminative model by creating samples that are similar to the original data set[17]. It is to say, from a noise vector (random vector), it can generate an example with a probability distribution similar to the original.

We can see in Figure 2.11 a general schematic of GAN. We have adapted the GAN in Figure 2.11 to the automotive world. The training phase is compounded by first generating the samples using noise and G, and then D classifies between the authentic data and the crafted one. In the final part, we compute the loss between the real label and the output of D. The generator updates its weights for deceiving the discriminator. On the other side, the discriminator does the opposite. It is worth mentioning that the two models of D and G need to be of similar computation power (i.e., similar layers and neurons) to have a well-trained system. Training GAN is not a simple task, and it could happen different problems[45]. The possible complications during training GANS are:

- **Mode Collapse:** Synchronize well the discriminator and the generator[17] is a challenging task. G needs to be less updated than D. Otherwise, it could happen the mode collapse. The mode collapse is when G does not have diversity on the outputs, only can generate few different samples.
- **Vanishing gradients:** In case D learns fastly to distinguish between authentic samples and fake samples, G is not able to learn from D.

Figure 2.11: Schematic performance of a generic GAN.



- **Failure to converge:** There are cases when G and D parameters oscillate, making an unstable solution.

Researchers have found solutions to the challenges of training GANS:

- **Wassertein GAN:** This GAN variation put limits when D updates the weights. We explain it in deep in section 2.4.1.2.
- **Unrolled GAN:** The unrolled GAN trains in the same way the D but G is updated every k steps[31]. Unrolled GAN solves the problem of mode collapse.
- **Change loss:** For avoiding vanishing gradients and failure to converge, researchers proposed to change the training loss to the Wassertein loss or min-max modified [45].

GAN was not created for crafting adversarial examples, indeed it was thought for enlarging the databases that do not have a wide range of samples. Eventhought, are other authors that also used the GAN for deceiving intrusion detector systems [11, 24, 34]. The GAN technique is used later in the experimental evaluation as a generative model to make intrusion samples. The previous model creates samples that we modify them later for bypassing the IDS.

Furthermore, it is challenging to know when to stop training the GAN. In case of being over-fitted, we could have the collapse mode. We explain the previous problem in section 2.4.1.1.

2.4.1.1 Metrics GANs

To know the quality of the GAN or to check it during training, it is needed to have evaluation metrics. There are different manners to evaluate the GANs. We can group the methods in qualitative metrics and quantitative metrics. The qualitative metrics are those in which data is descriptive, i.e., the user can differentiate between generated images and the real ones or not. The quantitative metrics are those where the information is measurable and has numerical values. As it is difficult to evaluate the samples generated by the automotive GAN by the human eye, we decided to use a quantitative metric,

In this section, we will only explain the two quantitative metrics more used[7]:

- **Inception Score (IS):** It is necessary to pre-train a neural network with the images or samples before computing the score to classify the generated samples. Then, the model

predicts the generated images. The result is the probability of the sample being a concrete class. The score captures two features: the quality and the diversity of the samples generated. The drawback of IS is that the method does not compare the statistics of the real samples with the statistics of the generated samples[44].

- **Fréchet Inception Distance (FID)**: This method is an extension of the Inception Score. We can see the Fréchet Inception Distance formula in Equation 2.6. Where m is the mean and C is the covariance of the group of samples. The lower is the distance, the more similar are the two groups of samples. The problem of FID is that we assume that the distribution of the data set is Gaussian, which is not always assured.

$$d^2((m, C), (m_w, C_w)) = \|m - m_w\|_2^2 + \text{Tr}(C + C_w - 2(CC_w)^{1/2}) \quad (2.6)$$

2.4.1.2 Wasserstein GAN

One of the main differences between Wasserstein GAN (WGAN) and GAN is loss function. The loss function derives from the Wasserstein distance[2]. The Wasserstein loss formula is shown in equation 2.7 and 2.8.

$$loss_{discriminator} = averagescore_{realimage} - averagescore_{fakeimages} \quad (2.7)$$

$$loss_{generator} = -averagescore_{fakeimages} \quad (2.8)$$

The other difference with GAN is that the updates of the D weights are clipped to a maximum variation. Furthermore, the [2] authors to use RMSprop optimizer.

2.4.2 Heuristics

In section 2.4.2, we will explain the simplest and most used mechanisms for crafting adversarial samples by heuristics. Heuristics is a manner for solving problems that implies practical methods for achieving a specific goal [40]. The heuristics do not entail optimally solving the problem. Heuristics only find a simple manner of solving the problem. Thus, these samples would not be optimal. In the current section, we will explain the Fast Gradient Sign Method and the Basic Iterative Method. There are a lot of others attacks made by heuristics as could be the Jacobian Based[38] or Carlini and Wagner[10]. Unfortunately, we need to focus only on some of them.

2.4.2.1 Fast Gradient Sign Method

The Fast Gradient Sign Method (FGSM) [16, 23] is it based on adding noise to the original sample along the gradient direction.

$$adv_{sample} = x + \epsilon \cdot sign(\Delta_x J(\theta, x, y)) \quad (2.9)$$

The method can be expressed in 2.9, where ϵ is a small number, Δ_x is the gradient with respect to the input, x is the input sample, and y is the wanted label. The samples crafted with FSGM are not an optimal adversarial sample. Moreover, as FSGM is a one single step attack, it is easy to develop and it is also easy to defend.

2.4.2.2 Basic Iterative Method

There are some cases where we cannot feed IDS directly with the samples. Before the packet arrives at the IDS, we need to pass it through a physical device. In the current case, the samples navigate through the bus as frames. The Basic Iterative Method is an extension of FGSM to bypass those cases. The BIM[22] runs multiple times the FGSM until the sample is miss-classified. In each iteration, we clip the sample or the bit. The reasoning for doing the clipping is to avoid being out of the range of the bit $[0,1]$. We can see the algorithm on equation 2.11.

$$adv_{sample}^0 = x \quad (2.10)$$

$$adv_{sample}^N + 1 = Clip_{x,e} \{adv_{sample}^N + \epsilon \cdot sign(\Delta_x J(\theta, adv_{sample}^N, y))\} \quad (2.11)$$

2.5 State of the art

As explained in section 2.1.3.1, there are various types of CAN IDS. Furthermore, talking about ML IDS, there are different algorithms to build them. Before deciding which is the most suitable IDS to use, we need to benchmark and compare their performance.

There is little information available about benchmarking the CAN IDS. ATG[18] tests the CAN network by applying Denial Of service attack, Fuzzing attack, and target spoofing attack. ATG is also able to capture data from CAN. An essential characteristic of ATG is that researchers tested it in a real environment. The problem with ATG is that it is not focused on testing IDS, it only tests the network security. CANsec [57] is an in-vehicle evaluation security tool. It evaluates the possibility of having an attack, but again, it is not able to test the countermeasures of the vehicle. CANsec, cannot either craft an attack dataset. In HCRL[47], researchers produced an attack dataset that includes DoS, Fuzzy attack, Spoofing attack (RPM, gear). The dataset has only some of the known attacks and does not benchmark the IDS. In [24], researchers tested different classification models by crafting samples with a WGAN. The problem with this work is that it is not performed for the automotive sector, and they only test WGAN attacks. In [58], the researchers created a testbed for remote IDS. The issue of [58] is that for remote IDS the environment is different. In our work, we aim to build a testbed but for the in-vehicle IDS.

2.6 Motivation

We aim to design a framework capable of testing ML IDS. The IDS alerts an attacker when it identifies a failure in the integrity of the CAN data. The need to design a framework for benchmarking these IDS arises from the variety of ML IDS and the lack of a similar tool.

The goals of the benchmarking framework are:

- Check that the IDSs analyzed can detect general attacks.
- Find which IDS from the analyzed is superior.
- Check that the IDSs analyzed can detect unknown attacks.
- Check that the IDSs are not easy to bypass.

Chapter 3

Threat Model

In the current chapter, we are going to explain the attacker's motivation and goals, CAN security violations, the type of CAN attacks, the attacker's knowledge, and the attacker's capabilities.

Before starting with the attacker's motivations and goals it is worth mentioning the relationship between security and safety in the automotive sector. From [15] we can extract two definitions:

- **Safety** is the degree to which accidental harm is prevented, detected, and properly reacted to. Safety can be further categorized into health safety, property safety, and environmental safety.
- **Security** is the degree to which malicious harm to a valuable asset is prevented, detected, and properly reacted to. Security can be further categorized into communications security, data security, emissions security, personal security, and physical security.

The main difference between them is that security aims to avoid intentional damage whereas safety focuses on preventing unintended damages. It is important to link both concepts because in the automotive sector it is very probable that a security issue leads to an accident[19]. The aforementioned statement could lead to considering that security is a bullet point on safety evaluation.

3.1 Attacker's motivation and goals

The most significant attacks are those that carry safety risks for vehicle's occupants[37] and other road users. Other attacks could not have an effect on the driving performance but could deactivate some functionalities of the vehicle. It is important to identify the final goals of the attacker in order to understand the range of the attackers willing to compromise the system. Thus, we can classify the attackers depending on their final goal. The most common ones in this context are:

- Damage the reputation of a vendor. The attacker who aims to damage the prestige of a vendor is usually another vendor. This type of attacker commonly has the motivation of earning more money by attacking the competence. The attacker usually has abundant resources considering the automotive vendors.
- Physically injure the vehicle's occupants or the road users. The attacker who wants to attempt to human life has, generally, a personal motivation.
- Violate the privacy of the user by stealing confidential information. Depending on the type of information that the attacker wants to steal, the motivation of this attacker would be

personal or economical. In actual times there are various companies that sell the user's data.

- Steal the vehicle or prevent starting the engine affecting thus the vehicle's availability. The type of attacker that aims to steal a car is usually a delinquent. The motivation of the attacker is criminal purposes.

Another, less obvious attacker is the vehicle owner itself, which could be interested in implementing intentional modifications to the vehicle that are obtainable through software. For example, change the maximum cruise velocity.

3.2 CAN Security Violations

In general we can categorize attacks and attackers also depending on the means through which the attacker implements the attack. For that reason, we define which parts of the general system the attacker can harm with the types of the security violations:

- Damage the **integrity** of the system: by changing the behavior of a functionality of the vehicle. This security violation is the one which implies more safety risks. A countermeasure for this security violation is an Intrusion Detector System.
- Affect to the system **availability**: by making the user not able to use some functionality or even to get into the vehicle. Depending on the type of affected functionality, the security violation implies or not a safety risk. If the user cannot use the radio while driving, it does not affect her safety. Differently, if the user has an accident and her airbag is unavailable, the user could be injured.
- Compromise **confidentiality** of the users: by sniffing users' information. This security violation does not imply any safety risks. Furthermore, this security violation is the most challenging to prevent because the possible countermeasure is using cryptography or access control. Both countermeasures are challenging measures to implement in CAN.

In order to secure the vehicle, it is necessary to prevent the system from the previous security violations. For preventing attacks, there are different countermeasures that impede each security violation. Protecting availability and confidentiality in the CAN system is challenging, and to assure them we need to make considerable changes on the protocol. Furthermore, it is not feasible to focus on different security issues at the same time, it is better to tackle one by one. For that reason, our work is focused on protecting the integrity of the system. In order to secure the integrity of the vehicle (as we explained in section 2.1.3) we use as a countermeasure the Intrusion Detection Systems (IDS).

3.3 Types of CAN attacks

The IDS countermeasure aims to detect the most common CAN attacks. In the current section, we are going to explain the most typical CAN attacks. The attacks are extracted from [20, 1]:

- **Message Injection attack(MI)**: Message injection is one of the most basics attacks. It consists of injecting the packets on the CAN bus. It refers to an attack where the attacker modifies ID and/or payload to accommodate her goal.
- **Tampering attack**: Type of attack which the attacker writes on the bus at the same time while another ECU is also transmitting, making thus the resultant message not valid.

- **Dropping attack:** Dropping attack is another basic attack that consists of erasing single or various packets of the network. The attacker can further only erase a part of the packet. This attack simulates the case when one (or various) ECU cease its transmission as a cause of the attacker[50].
- **Denial-of-Service attack(DoS):** Denial of service attack is an attack that affects the availability of the CAN bus. The attack takes advantage of the vulnerability of CAN related to priorities. The attacker injects a flood of messages in the CAN with zero ID or low ID. The zero ID is the one with the most priority, and such flood of messages provokes that the other ECUs are not able to send packets through CAN, affecting thus to the operation of the vehicle.
- **Fuzzy attack:** Fuzzy attack is used to do reverse-engineering on the DBC file, and consists in testing iteratively random packets or changing a part of a packet, and then see which is the effect on the CAN bus.
- **Reply Attack:** The replay attack has two phases. The first one consist in sniffing the messages on the CAN during a period. In the second phase, the attacker sends a single packet or a set of packets which the attacker has already seen on the network. The replay attack assures that the message injected is valid because it is a copy of a real message.
- **Spoofing attack:** the attacker has compromised one ECU, and she transmits using the ECU's physical features (i.e frequency, ID, etc...). Spoofing attack is also a type of a masquerade attack more sophisticated. Masquerade attack is when an attacker intends to impersonate an ECU by sending the messages with the ID of the victim.
- **Others:** There are other type of attacks but there are out of the scope (e.g attack bus on error handler)[50, 53, 21, 20].

The IDS should be able to detect the previous attacks because those are well-known. The first step that an IDS vendor should check before launching her product is to know if her IDSs are capable to detect at least the previous attacks. The most difficult attacks to detect between those are: reply attack and spoofing attack. A replay attack is difficult to detect because the attacker is using valid messages, and we need a context to detect when the attack occurs. The aforementioned context can be categorized as temporal context or ID context. An example of temporal context is that a car is not able to change its velocity from 0 km/h to 200 in one second. On the other hand, ID context is the context of combining different functionalities (or ECUs), e.g., the vehicle is not going to change the steering angle 90 degrees while it is driving at 120km/h because the vehicle would get out of the road. The spoofing attack is very dangerous in case the attacker knows the data structure of such ECU because she would also use valid packets. The attacker can extract the data structure information from the DBC file or by performing a fuzzy attack.

3.4 Attacking the IDS

Although in case the IDS can detect the previous attacks, the attacker can bypass the IDS. In case the IDS is built with machine learning models, the attacker could exploit the vulnerabilities explained in section 2.4. We can further categorize the attacks for bypassing the IDS as evasion attacks and poison attacks (section 2.4). In our thesis, we perform some tests on evasion attacks. Poison attacks are out of the scope because they occur during training (or retraining) of the models, and we want to test the IDS once trained.

Furthermore, we did not consider the process of compromising the IDS. The attack specificity and the error specificity encompass how the attacker can compromise the IDS.

Attack specificity means to classify the aim of the attacker between a targeted or indiscriminate attack. The attacker could perform the attack within only a target packet or craft multiple of them indiscriminately [5]. In our thesis, we focus on indiscriminate attacks because usually the attacker needs different attacks to achieve her goals, e.g., to change the steering angle, you need a progression of CAN messages.

On the other hand, error specificity categorizes the attacker's procedure for crafting samples. When the attacker crafts a sample for being misclassified as a concrete class, we call it a specific attack. On the other hand, if she does not care to which class is classified, we call the attack generic. In the current environment, the attacker wants that the IDS classifies her packet as normal behavior. Accordingly, we define that in our settings, the attacks are specific. In the current environment, the attack influence is exploratory because the attacker does not affect the building or training of the IDS. The attacker can only manipulate the decision of the IDS.

3.5 IDS Attacker's knowledge

Depending on what the attacker knows of the system, she would implement the intrusion differently. The more information she has, the easier she can find the vulnerabilities and exploit them. First of all, we describe the type of information the attacker can use to fool a CAN IDS. The profitable information in the current system is formed by: the training data-set of the IDS; the feature set, which is composed of the features used for training the IDS; the learning algorithm used to build the IDS; the hyper-parameters of the learning algorithm used; and finally the optimized objective function. We can describe different attack scenarios depending on the attacker's knowledge[5]:

- Perfect Knowledge: The attacker knows everything about the system. It is also known as the white box attack.
- Limited Knowledge: The attacker has some degree of knowledge. It is also known as the gray box attack.
- Zero-Knowledge: The attacker does not know anything about the system. It is also known as the black box attack.

In our work, we suppose the attacker has perfect knowledge. More accurately, the attacker has in her power the data-set for training and testing and knows which is the exact sub-set. Furthermore, the attacker has the IDS models and knows which are the input features. We do those suppositions because it is the worst-case evaluation for the IDS. Therefore, if the IDS passes the test with success, it means that it is secure for such kinds of attacks at least. It is worth mentioning that in our thesis, we have explained READ as a mechanism to extract information. The attacker can further use that mechanism to implement her attack.

3.6 Attacker's capabilities

It is significant to define which are the capabilities of the attacker to delimit the attack influence. The attacker is already on the network through a physical device (e.g., OBD-II port) or compromising one ECU (i.e., physically or remotely). She can read and write on the CAN bus. Furthermore, the attacker can only decide the ID and the payload of the frame.

In case the attacker is not compromising one ECU, the attacker needs to decide or know which CAN ID wants to masquerade. The attacker cannot always choose the ID because potentially, the

chosen ID does not exist. Depending on the case, the attacker spoofs the ECU that performs a specific functionality.

Furthermore, the attacker needs to decide the payload's frame. Moreover, the data format needs to follow the norms written in the DBC file. Otherwise, the frame will not be valid. As we said in section 2.1.1.1, the DBC file is not public. Furthermore, although the injected messages are not detected by the IDS, it is possible that the compromised ECU only gives information of some sensor, and any ECU performs an action with this information, making thus not effective attacks.

Chapter 4

Approach and implementation

To understand the working flow, we explain the system architecture of our IDS benchmarking. The process to follow for building our IDS benchmarking is:

1. **Select the normal CAN dataset:** It is crucial to select a real normal CAN data set to have realistic testing on the IDS. In our case, we use ReCan[55].
2. **Synthesize basic CAN attacks:** We craft the basics attacks. The basic attacks are those already explained in section 3.3.
3. **Provide(or design) and train the IDS:** the user trains the IDS model in the case she already has it and can proceed to the next step. Otherwise, the user builds and trains the IDS in this step. In this chapter, we explain how to assemble IDS with different algorithms
4. **Test the IDS with basics attacks:** The current step is necessary for going further or regress to the previous, i.e., to check the correct performance of the IDS. In case the IDS has not a good performance related to the basics attacks, we need to refine it.
5. **Synthesize sophisticated attacks:** In the current step is when we craft more sophisticated attacks. Those attacks are the evasion attacks that could bypass the IDS. In our thesis, we divide them into two parts
 - (a) **WGAN** attack.
 - (b) **Bit BIM variant** attack.
6. **Test the IDS with sophisticated attacks:** The final step is to test the sophisticated attacks on the IDS, i.e., test the attacks that could bypass the IDS. In case the attacks are not detected, we should perform another step to refine the IDS. The countermeasures of the sophisticated attacks are out of the scope.

In chapter 4, we explain only the steps related to the approach and implementation. We expose all the related tests and results in chapter 5.

4.1 Normal CAN dataset

In the current work, we use the ReCan dataset[55]. ReCan contains real data set extracted from 3 cars and 4 trucks. For each vehicle there are various experiments done. Each experiment is composed by two parts:

- **RAW data:** a CSV file with time stamp, CAN line, ECU identifier and binary data extracted from the CAN frames.
- **Decoded data:** a CSV file with time stamp, CAN line, ECU identifier, variable and values extracted from the CAN frames.

In the current work, we use the raw data file because we preferred to do our own processing of the data. Furthermore, it is a more realistic environment to have the raw data of the CAN bus at the beginning. On the other hand, we have used the dataset of *C-1-AlfaRomeo-Giulia* because of the considerable amount of data frames. Moreover, the dataset comprises the behavior of the CAN bus of a car driving in the city and on the highway. Thus, the dataset includes general situations. More accurately, we have used the first experiment but could have selected any of them.

4.2 Synthesize basic CAN attacks

A complete, authentic attack database does not exist, so we need to synthesize it. It is not usual to create attacks on the car because it is time-consuming and not efficient. Moreover, it requires specific knowledge of the vehicle, and usually, it is not transferable to another car model. On the other hand, it has considerable risks for the researcher to try to make attacks on a vehicle and then test which are the consequences.

With the attack tool, we can perform the attacks: basics attack, replay attack, fuzzy attack, progressive attack, drop attack and, DoS attack. The only attack that we have not explained in section 3.3 is the progressive attack. The progressive attack is a similar attack to the fuzzy attack, but in the first one, the payloads neither the signals are not random. The tool increases or decreases the packet's payload or the concrete signal one by one or by a specific number. The attack wants to resemble a counter. There is also a multiple attack that combines different types of the previous ones. Multiple attack is not a realistic attack because the probability of having two attacks at the same time is very low. Nonetheless, we have crafted it to check the IDS performance.

As we are interested in the data comprised in the frames and their pattern along time, we use the basics attack, the replay attack, and the fuzzy attack. We can also use the multiple attack to check the performance of an IDS simulating the situation in which we do not know the intrusion type.

The attack tool crafted does not fulfill some properties of the CAN bus for simplicity. One of the properties of CAN that does not accomplish is in the situation of injecting messages. It does not delay the current ones (contrary to what is supposed to happen on the CAN bus). Another unfulfilled property is the treatment of a flood of messages. The tool cannot make recovery management of the frames, i.e., the CANTack does not erase the messages that do not fit in the bus. Those unfulfilled properties are not a problem for the current project because we are focusing only on the payload contained in the data frames. We are not interested in the inter-arrival frequency of packets.

We can find the CANTack in [51].

4.2.1 Crafting attacks

In the current section, we are going to explain how to synthesize the attacks with the CANTack. First of all, we created the most basics attacks that we have already exposed in section 3.3 although they could have different names in CANTack. One of the differences between the attacks of the tool and the ones explained in section 3.3 is that there is almost no distinction between an injection attack and a spoofing attack. The user specifies when she wants to inject new messages on the data

set (inject) or use the already packets frequency in the data set (masquerade). In case the user chooses injection, the attack tool puts the frames specified inside the data set with an injection rate that the user can choose. The CANtack adds a little randomness to the injection rate trying to create a data set more similar to reality. Contrarily, the masquerade case only changes the payloads of the frames being already in the data set. The tool can craft the attacks in both manners, although the user would create a not realistic attack. For the creation of the attacks we have followed the specific procedure for each attack:

- **Basics attack:** The user tells which is the payload or payloads to insert or masquerade. The user also specifies the ID that could be valid or not and when to insert it (i.e., a time delta from the initial timestamp of the data set).
- **Dropping attack:** To delete a number of packets specified taking into account the time delta from the beginning.
- **DoS:** During an amount of time specified, inject the maximum number of packets per second considering the throughput of the bus. The user can decide which ID to use, but the default ID is the 0 ID. The default payload contains all the bits 1, but the user can customize it.
- **Reply Attack:** The user needs to specify the time delta for starting sniffing and the time delta for starting injecting or masquerading. The attack follows the procedure that we have explained in section 3.3. The tool picks one either a specific number of consecutive packets from the subset sniffed. The chosen packets are injected or masqueraded in the dataset. We can combine this attack with others to build a forceful attack.
- **Fuzzy attack:** To inject or to masquerade a random packet or signal. The user needs to specify the maximum and the minimum range. In case the user wants, we can combine it with the replay attack. Take the sniffing packets and replace a part of the payload with the fuzzy attack.
- **Progressive attack:** The tool takes the signal provided by the user, and it increases or decreases it between a minimum and a maximum. We can combine the progressive attack with the replay attack.
- **Multiple attack:** To mix randomly all the previous attacks.

Moreover, the user can invent other attacks if she knows the payload to insert or masquerade. We can create new attacks using the basics attack, and then putting the messages to insert. One example is developing the WGAN and the Bit BIM variant attack.

Finally, the output of the attack tool is a data set with an extra column that says which frames are the normal data frames and which are modified or created.

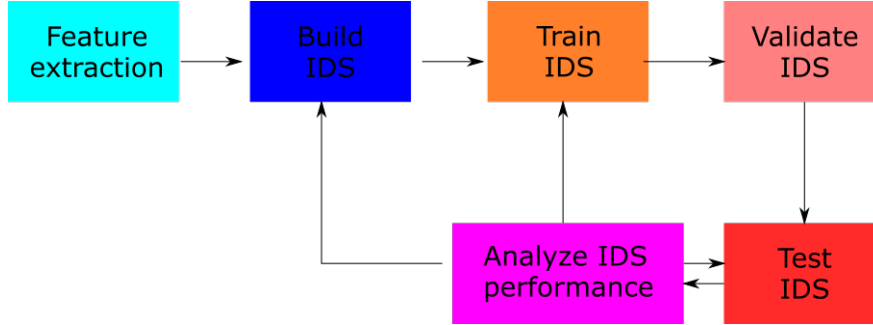
4.3 Intrusion Detection System design

In the current section, we are going to explain how we build our IDSs. In case the user provides her IDS, the user would need to train the IDS in its specific manner and with the wanted data set.

In our work, we have built one IDS for each ID. We do not join the information between IDS of different IDs, so the IDSs crafted are not contextual IDS. It is worth mentioning that it is not necessary to build an IDS for all the existing IDs. Some IDs have constant data frames or almost invariable (from 2 to 10 different payloads). In case it is also wanted to detect the attacks for such kind of ID, we suggest using a distinct IDS type, as it could be a rule IDS. Our IDS is supposed to detect when an intrusion occurs only by learning the ordinary operation of the CAN bus. It

identifies an intrusion when there is different behavior than the expected (unsupervised IDS). We can see the IDS design process in Figure 4.1.

Figure 4.1: IDS design process.



The IDS designing process has the following general workflow:

1. **Feature extraction:** For feature extraction, we have used the READ method that we explained in section 2.3. From the READ method, we can extract all the signals from the payload flow of the different IDs. After obtaining the signals, they are arranged together in the same order that we have found them. From now on, every time we mention a packet, we are referring to this rearrangement of bits. It is worth mentioning that each bit is a feature. For that reason, there are as many inputs as the total sum of the signals' bit.
2. **Build the IDS:** To build the IDS, we need to decide which algorithm and its hyperparameters to use. The algorithms used are:
 - Isolation forest (IF)
 - One Class Support Vector Machine (OCSVM).
 - Auto-encoder crafted with Neural Network (NN).
 - Auto-encoder crafted with Long-Short Term Memory (LSTM).
 - Auto-encoder crafted with Gated Recurrent Unit (GRU).
 - CANnolo[26].

We have built two sizes of the LSTM and GRU auto-encoders for understanding which has better performance. We have chosen those algorithms because we want to test the attacks on different models (similarly as in [4]). Some algorithms consider the pattern a long time (LSTM, GRU, and CANnolo), and others that do not (IF, OCSVM, and NN). The algorithms that do not consider the time sequences cannot detect attacks with valid payload, as could be the replay attack. Each algorithm for crafting IDS is trained, validated, and tested differently.

3. **Train the IDS:** The model learns the pattern of payloads in the normal situation in the current step.
4. **Validate the IDS:** In the validation step, we measure some features of the model that we use in the test step. Those features are the logarithmic error mean and covariance.
5. **Test the IDS:** Predict when there are attack frames, and compare the prediction with the real label. There are some models which also show the confidence mark of its decision. The current step is the longest part because it is when we test all the attacks.

6. **Analyze the IDS performance:** In the current step, we analyze the performance of the IDS with the testing results. In case the IDS does not have good results, we need to decide if it is necessary to rebuild or retrain the model. Otherwise, we can do more tests on the IDS.

The percentage of the data set used for training is 42%, for validating 18%, and for testing 40%. We decide which amount needs to be for each step by random search method, seeing which has the best performance. It is worth mentioning that the data set is not disordered during the training process, neither in the validating nor testing process. We determine in that way because otherwise, the models would not be able to comprehend the time-series pattern. Furthermore, we divide the data set in subsets (for training, validating, and testing) considering the order of arrival of packets, not the number of frames that appeared for each ID. We perform that division because the sniffing attack performs similarly. This type of division induce problems. The problems are that such division and can make the subsets unbalanced. Not all the IDs have the same frequency, neither the same emergence (for example the ABS is only used in case of emergency, this is an anomaly but not an attack, and the IDS should catalog it as normal). The problems are that there are not enough packets to do good training, good validation, or good testing. In an authentic environment, in case the vendor has not sufficient packets should extract more data. An attacker could be in a similar situation and would sniff more CAN frames. For testing the IDS, we need to have a dataset with attacks on it.

Either the user provides the IDS or builds it, the prediction output should be between 0 and 1. The reason is to have the same criteria for all the IDS. The prediction output should mean the probability of that payload to be an attack. Therefore, some IDS have only the possibility of having it in integers $[0, 1]$ output where the 0 means normal, and 1 means intrusion.

After having explained all the general aspects between all the built IDSs, we divide the previous algorithms into two groups: ones already made for the scikit-learn library (table 4.1, IF and OCSVM)[9] and the others provided by the research group (table 4.3, NN, LSTM, GRU, and CANnolo).

Design of IDS with algorithms from scikit-learn[9]

The general workflow of the algorithms done by [9] is to create the model and then change slightly the output. The prediction output of IF and OCSVM is 1 in the event of being a normal sample, and -1 in case of being an outlier (i.e., an intrusion). As it is said previously, we need to build the IDS similarly, thus the output needs to be between 0 as a normal and 1 as an intrusion. We changed the prediction output of IF and OCSVM with a simple rule: we convert the output -1 to 1, and the output 1 to 0. Furthermore, IF and OCSVM IDS do not have the validation phase. We present the hyper-parameters of IS and OCSVM on table 4.1.

Table 4.1: Parameters of models used for IDS.

Parameters	OCSVM
kernel	linear
ν	0.01
Parameters	Isolation Forests
# estimators	10
Warm start	True

OCSVM

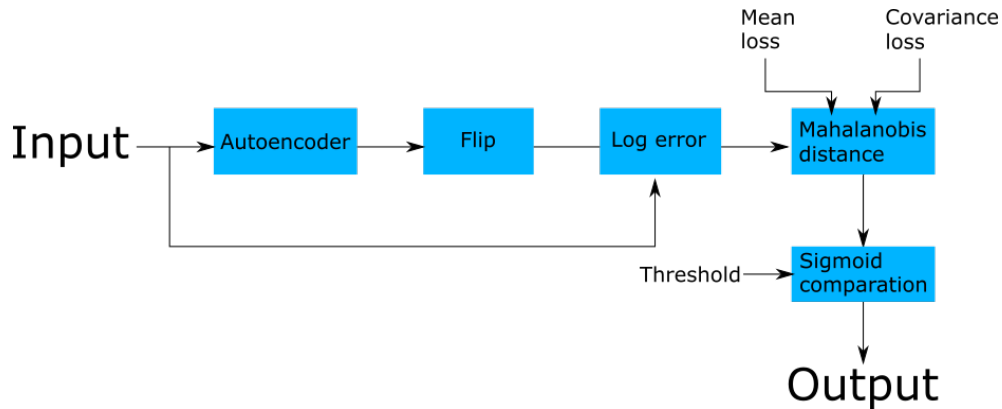
The hyper-parameters used for OCSVM are the ones that perform better in [12].

IF

The hyper-parameters used in IS are found by random hyper-parameter search methodology, i.e., by casually take the hyper-parameters and then choose the ones that perform better in terms of accuracy, detection rate, precision, and f-measure. It is worth mentioning that during training we need to specify that all the training data belongs to normal data (i.e., no contamination). Otherwise, IF model cannot learn the normal pattern, neither discriminate between a normal sample and an outlier.

Design of IDS with particular algorithms

Figure 4.2: Working flow of the created IDS.



At this moment, we are going to explain the operation of the second group algorithms. In the figure 4.2 we can see the workflow of the crafted IDS. The general idea is that when the auto-encoders can regenerate the source (i.e., low reconstruction error), the input follows a similar pattern to the dataset trained. Otherwise, in case it is not well regenerated, the IDS labels the packet as an intrusion. The architecture is analogous to the one in [28, 26]. We describe in more details each step of the process:

- **Auto-encoder:** The auto-encoder tries to regenerate the source. Depending on the algorithm used for the initial auto-encoder the source (input) and the target (output) of the auto-encoder model be different. In the table 4.2 there are summarized the input-output of the different auto-encoders. In [48] was evidenced that reversing the order of the source gives better prediction in the auto-encoders. Later, in [26] was tested that has the same result for reversing the source or the target. We use the reverse order in the target, to feed the auto-encoders. After, we flip the output in the prediction.

Model	Input	Output
NN	#features one packet	#features one packet
LSTM	#features one packet + 39 previous	#features one packet
GRU	#features one packet + 39 previous	#features one packet
CANnolo	#features one packet + 39 previous	#features one packet + 39 previous

Table 4.2: Relation between input and output auto-encoders.

- **Flip:** In this step, we flip the output of the Auto-encoder. The actual step is necessary to be able to compare the input and the output.

- **Logarithmic error:** To know if the auto-encoder can manage to regenerate the source or not, we use the logarithmic error for computing the reconstructed error. We expressed the formula used for computing the logarithmic error in 4.1. We extracted the formula from [26].

$$\Delta_k = -\left(b_{o_k} \log(b_{i_k} + \epsilon) + (1 - b_{o_k}) \log(1 - b_{i_k}) + \epsilon\right) \quad (4.1)$$

In Equation 4.1, Δ_k means the logarithmic error of the k est bit, b_{i_k} is the k est input bit, b_{o_k} is the k est output bit (or prediction bit), and ϵ is a margin error introduced. In the current work ϵ is equal to $10e^{-15}$. In the validation part, we compute the mean and the covariance of the logarithmic error produced by the subset.

- **Mahalanobis distance:** With the mean and the covariance of the loss, we can compute the Mahalanobis distance.

$$D_M(x) = \sqrt{(x - \mu)^T C^{-1} (x - \mu)} \quad (4.2)$$

We expressed Mahalanobis distance in Equation 4.2. We extracted the Equation 4.2 from [14] where μ is the logarithmic error mean, C^{-1} is the inverse of the covariance of the logarithmic error, and x is the logarithmic error of the concrete sample. Besides, in the validation part is computed the threshold of the Mahalanobis distance by computing the percentile. Depending on the model, the percentile is greater or lower. We found the percentile by the random search method.

- **Sigmoid comparator:**

$$\delta = \frac{D_M(\Delta)}{Threshold} - 1 \quad (4.3)$$

$$sigmoid(\delta) = \frac{1}{1 + e^\delta} \quad (4.4)$$

In the last step, we take the distance of the actual sample (D_M) and divide it by the threshold. Then we subtract 1 (seen in equation 4.3). We pass the result (δ) by a sigmoid function, seen in Equation 4.4. The previous result is the final prediction output of the IDS.

We present the parameters for implementing the auto-encoders in table 4.3. Moreover, similar to [26], we trained all models with ADAM optimizer, 128 batches, binary cross-entropy loss, early stopping with 10 or 20 epochs, depending on the satisfactory performance of the model. In all the above models, the activation function is the *tanh* in all layers except the last one. The final output layer has the activation function of the sigmoid.

Parameters	Models					
	Neural Network	Small LSTM	Large LSTM	Small GRU	Large GRU	CANnolo
Hidden Layers	2	2	4	2	4	5
Number of neurons per layer	16, 16	256, 256	512, 128, 128, 512	256, 256	512, 128, 128, 512	128, 128, 128, 128, 256

Table 4.3: Parameters of network models used for IDS.

After explaining the general characteristics of the built models, in the Tables 4.3 and 4.2 we can compare the models. The main differences are:

- **Auto-encoder NN:** The auto-encoder NN is not a model that recognizes the time sequences. For that reason, we train it with only one packet source and output. We extract the parameters of auto-encoder NN from [4]
- **Auto-encoder LSTM:** The LSTM models have as the source the actual packet and the 39 previous ones the output is the current packet. We extract the parameters of auto-encoder LSTM from [4]
- **Auto-encoder GRU:** The GRU models have as the source the actual packet and the 39 previous ones the output is only the current packet. We extract the parameters of auto-encoder GRU from [4]
- **CANnolo:** The CANnolo model has as the source and the output, the actual packet and the 39 previous ones. We extract the hyper-parameters of CANnolo are from [26].

4.4 Synthesize sophisticated attacks

4.4.1 Design of WGAN attack

In section 2.4.1.2, we have explained what is a WGAN and why we can use it for deceiving the IDS. To summarize, the WGAN can generate samples with a similar distribution that an IDS could miss-classify those created examples. The attack with a WGAN could be the evolution of the replay attack. Moreover, although in section 3.5 we specified to focus on white-box attacks, we can use WGAN attack for implementing the black-box attacks. We perform the pre-processing of the original dataset with a similar method to when we trained the IDS: rearranging the bits of the symbols found with the READ method. The difference is that for training the WGAN, we disorder all the packets. The reason is that otherwise, the WGAN could have a mode collapse problem. Besides, in each iteration of the training process (in each epoch), we disorder the dataset again.

We show the hyper-parameters used for building the WGAN in the table 4.4. We have found those parameters by the random research method.

Table 4.4: WGAN hyper-parameters.

Models	Parameters	
Generator	layers	Noise dimension,64,64,Packet bits
	Activation function	Leaky-Relu
	Loss	Binary cross entropy
	Optimizer	RMSprop
	Learning rate	0.00005
	Unrolling steps	10
Discriminator	Layers	Packet bits, 64, 8, 1
	Activation function	Leaky-Relu
	Loss	Wassertein loss
	Clip constraint	0.01
	Optimizer	RMSprop
	Learning rate	0.00005
	Error value	20

The batch size of the training is 64 samples. To know when to stop training, we have introduced an early stopping related to the FID measure. Every time the epoch finishes, we generate a set of samples with the generator of the WGAN. We compute the FID between the generated sub-set

and the original data sub-set of the original data set. In case the FID does not vary more than 0.005 during 20 epochs, we stop the training. The noise dimension of the generator is 200.

To consider the payload among time, we have also implemented a time-series WGAN. The only difference between the WGAN and the time-series WGAN is that in the second one instead of having the original data set with one packet per sample, we build the dataset with groups of 40 ordered packets per sample. The hyper-parameters are similar to the WGAN. In the table 4.5, we can see only the hyper-parameters that are different. We trained the time-series WGAN equally as in the WGAN. Differently, we stop the training when the FID varies less than 0.4 during the last 20 epochs. In some cases, we can reduce the variation to 0.25, depending on the total number of features. The fewer the features, the lower is the variation. Furthermore, the noise dimension in the time-series WGAN is 1000.

Table 4.5: Time-series WGAN hyper-parameters different from WGAN hyper-parameters.

Models	Parameters	
Generator	layers	Noise dimension,64,64,Packet bits*(LSTM dimension+1)
Discriminator	Layers	Packet bits*(LSTM dimension+1), 64, 8, 1
	Clip constraint	0.01

4.4.2 Design of Bit BIM Variant attack together with WGAN

In this section, we present the algorithm used for crafting adversarial samples. The algorithm is a variant of the typical BIM (explained in section 2.4.2.2) which tries to find which bit has a higher contribution to the loss and then changes it. Before crafting the attack, we first need a generator of packets capable of originating packets similar to the authentic frame flow. It is worth mentioning that the current method is a white box attack, and due to that, it is also necessary to have the IDS which the user wants to deceive. We have taken the NN auto-encoder IDS, and then we have tested the transferability of the attack into the other models. We show the algorithm

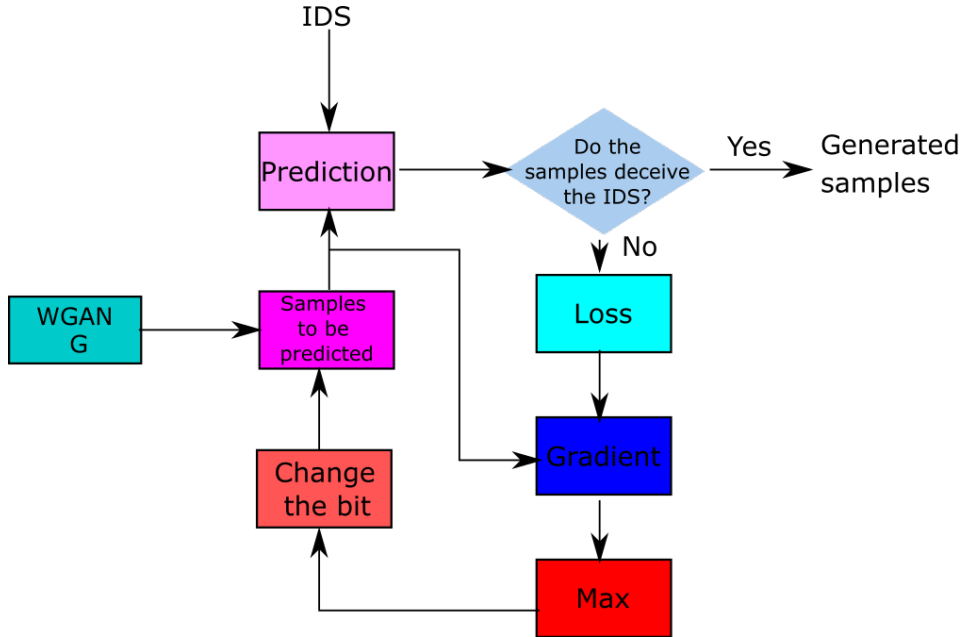


Figure 4.3: Graphical general workflow of Bit BIM variant.

on pseudocode 3 and graphically in Figure 4.3. It takes the packet generated and predicts the

output of the IDS. Then it computes the loss between the prediction and the wanted label. The wanted label is 0 because it is the case when the IDS categorizes the normal sample. Formerly, it computes the gradient of the loss concerning the sample, and it finds which is the bit that has more contribution to that loss. Finally, it changes the most contributing bit. The algorithm does the process iteratively until the IDS classifies the sample as a normal one. The algorithm stops when it cannot compute the gradient or after reaching the maximum iterations (epochs). It is worth mentioning that we have included a mechanism to avoid a worse prediction after changing the most contribution bit. In that situation, we return the bit to the previous value. To avoid having a closed-loop (i.e., to change constantly the same bit or the same group of bits), we store the changed bits during n iterations. During those iterations, we do not change those bits. We decided the number of iterations n depending on the number of bits of the packet. We took as the generator of packets the WGAN trained in the section 4.4.1. In case we take a generator of random packets instead of the trained generator of WGAN, the algorithm is not going to work because the gradient would be 0. The reason is that the prediction is 1, and the loss would be maximum at such point (i.e., taking into account the sigmoid function shape).

Algorithm 3 Bit BIM variant. Change each time only one bit

```

1: System Initialization: generate the fake samples, decide the number of epochs and pass the
  IDS model.
2: procedure BIT-FGSM(fakeSamples, epochs, model)                                ▷ Initialize parameters
3:   prediction = 1
4:   lastbit = []                                                                    ▷ Make a list of the bits changed previously
5:   while gradients ≠ 0 AND prediction ≥ 0.5 AND epoch < epochs do
6:     prediction = model(fakeSamples)                                             ▷ Make the prediction of the fake samples
7:     if predictionprev < prediction then fakeSamples ∨ eindexmax ▷ If prediction is worse,
  change again the bit
8:     end if
9:     loss(prediction, 0) ▷ Compute the loss between the expected label and the predicted
10:    gradient = grad(loss, fakeSamples) ▷ Make the gradient of the loss with respect to
  the fake samples
11:    gradient = dontchange(gradient, lastbits) ▷ Put 0 to the bits in the gradient that has
  already changed during the last n epochs
12:    indexmax = max(abs(gradient)) ▷ Find the bit in the gradient with the maximum
  absolute value
13:    lastbits = save(indexmax) ▷ Save the already changed bits during n epochs
14:    fakeSamples ∨ eindexmax ▷ Change only the bit of the maximum
15:    predictionprev = prediction ▷ Save the previous prediction
16:    epoch + = 1
17:  end while
18: end procedure

```

When someone tries to deceive an IDS with time series, she could think that would be the same procedure but with a different IDS (e.g., LSTM). The previous statement is not correct because when someone tries to deceive an IDS that recognizes the time sequences, it also needs to change all the previous packets, not only the current one. Time-series complicates a little bit the manner of crafting the samples. Once the attacker deceives the IDS with one sample (i.e., 40 packets per sample), it needs to perform the algorithm one by one by changing only the last packet of the sample. The time-series Bit BIM variant is more time-consuming because the tool generates the attack one by one. Whereas for the non-time-series algorithm the tool could generate more than one attack packet at the same time.

4.4.2.1 Post processing WGAN and the Bit BIM variant attacks

We designed the WGAN attack and Bit BIM variant attack to have a similar input of our IDS. The tool should craft the attacks in a general manner for all types of IDS. For that reason, the attacks need to be post-processed. Post-processing is the opposite operation done in the feature selection of the IDS.

We can summarize the process with:

1. **RE-READ**: Do the same operation of the READ method, but on this occasion, save the bits which are not a signal. I.e., save the bits of the packet that are constant in time.
2. **Re-arrange** the attack payload and the RE-READ payload: Put together all the payload taking into account the order of the bits. The bits to put together are the bits extracted from the previous operation and the bits generated from the attack.

After doing the previous operation, we can insert the payloads in the data set with the basics attack as explained in section 4.2.1.

4.5 Approach summary

In the current chapter, we proposed eight different models for building the IDS. In case the reader wants to reproduce our work, we showed the used parameters for building and training the IDS. Moreover, we specified which dataset was used, and extract the features by using the READ method. Furthermore, we explained how we developed the attack tool and the manner to use it. In this manner, the reader can perform the attacks and know if its IDS detects them. We explained a manner to craft new attacks with the CANTack, and how to introduce them in the dataset with a postprocessing part. In addition, we introduced two attacks for deceiving the IDS: WGAN and Bit BIM variant. With both attacks, the reader can test if its IDS is easy to bypass.

Chapter 5

Results of the experiments

We have performed the experiments using a Windows computer with Intel Core 7, a RAM of 16GB, and a GeForce-GTX-1660. We have written the program in python 3.8 with the help of PyCharm. The libraries used are Keras, Pandas, Numpy, and Scikit-learn. The user can find the code in [13].

In chapter 5 we expose the results of the test experiments performed with the IDSs. We first perform the tests related to common CAN attacks. In these tests, we want to analyze the general performance of the IDS. Finally, we develop the tests with more sophisticated attacks with WGAN and bit BIM variant. We analyze the performance of sophisticated attacks to know if the attacker can bypass the IDS easily. It is worth mentioning that the results are expressed as a mean between all the analyzed CAN IDs.

5.1 IDS performance with basic CAN attacks

For checking that the IDS can detect basic attacks, we analyze the performance of our IDSs when there are happening different situations:

- When the packet is **not valid**. We use the Fuzzy Attack with the whole payload and without considering the signals. The goal of a fuzzy attack is to do reverse engineering of CAN, but we use the previous attack for simulating a non-valid payload.
- When the packet is **valid** but does not have the correct time sequence pattern. We operate the Replay attack using a different number of pattern packets (3, 5, and 10 pattern packets).
- When we do not know the exact attack that the attacker is doing, which we simulate by **mixing** different attacks. We use the Multiple Attack.

5.1.1 Experiments with Fuzzy Attack

The first experiment that we perform is the test of the IDS with Fuzzy Attack. Executing the previous attack, we check that the IDS can detect when the payload in CAN is not valid. This attack should be the most effortless to detect. For the experiment, we introduced various random messages in the CAN database. In Table 5.1 we can see the performance of the IDS with the Fuzzy Attack.

We can observe in Table 5.1 that the OCSVM and the IF do not have an acceptable performance because the recall is very low. Furthermore, the OCSVM, IF, LSTM, GRU, and CANnolo do

<i>Fuzzy Attack</i>	Isolation Forest	OCSVM	NN	Small LSTM	Large LSTM	Small GRU	Large GRU	CANnolo
Accuracy	0.99836	0.79761	0.95576	0.95613	0.93746	0.96281	0.94888	0.85034
Recall	0.00578	0.33678	0.89801	0.91914	0.88321	0.92037	0.91617	0.99971
Precision	0.00969	0.02934	0.28932	0.29428	0.21948	0.33130	0.26214	0.11366
F-measure	0.00724	0.05398	0.43764	0.44582	0.35159	0.48722	0.40764	0.20412

Table 5.1: Performance results of the IDS with Fuzzy attack.

not have as excellent performance as the models from the papers where are extracted [12, 4, 28, 26]. The reasons could be the different implementation and that the attacks applied are distinct. The IF cannot detect the most uncomplicated attacks, and the OCSVM detects only 33% of the attacks. The auto-encoder models have similar results between them. From that group of models, the model with better performance is the Small GRU auto-encoder. Furthermore, the large auto-encoders have worse accuracy and recall than the small ones, which is an opposite behavior than the one in [4]. The model that detects better the Fuzzy attack is the CANnolo, but it does not have the best accuracy. The reason is that it has a lot of false negatives and false positives.

The f-measure and the precision are low although having an acceptable accuracy and recall. The reason is that the ratio between the number of attacks and the normal data is unbalanced. As the FP rate is high compared to the TP rate, the precision is low. Having a low Precision implies having a low F-Measure.

5.1.2 Experiments with Replay Attack

The second experiment with the basics attacks is the one performed with Replay Attack. In this attack all the payloads are valid, and this is the reason why it is challenging to detect them. Only the IDS which consider the time-sequence be able to detect them. We can observe the performance of the IDS with the Replay attack in Table 5.2.

<i>Replay Attack</i>	Isolation Forest	OCSVM	NN	Small LSTM	Large LSTM	Small GRU	Large GRU	CANnolo
Accuracy	0.99848	0.79186	0.93587	0.94933	0.92802	0.95493	0.94291	0.84474
Recall	0.0	0.12524	0.03080	0.609	0.4416	0.5572	0.64348	0.73304
Precision	0.0	0.01313	0.01623	0.24813	0.14430	0.26487	0.22926	0.10015
F-measure	0.0	0.02378	0.02126	0.35260	0.21752	0.35906	0.33807	0.17623

Table 5.2: Performance results of the IDS with of Replay attack (2-15 packets for the pattern).

The first conclusion that we can extract from Table 5.2 is the reaffirmation that only the IDS that consider the time-sequence can detect correctly this attack. For that reason, IF, OCSVM, and NN models have a low Recall. The GRU and LSTM models have similar performance between them. The model with better performance is the CANnolo, the reason is that such a model has more consistency taking the last 39 packets. The results are different if we train the models considering fewer previous packets. Differently, if we test the IDS with a higher pattern of packets, we have a similar behavior of the IDS and the Recall would be lower. In Table 5.3 we can observe the results with a higher pattern (from 35 to 50 packets).

Comparing the Tables 5.2 and 5.3, we can observe that the performance is worse in the second one. The reason is that in the second attack we increase the number of packets in the pattern. It is worth mentioning that the models that do not consider the time-sequences have similar performance.

<i>Replay Attack</i>	Isolation Forest	OCSVM	NN	Small LSTM	Large LSTM	Small GRU	Large GRU	CANnolo
Accuracy	0.87090	0.79310	0.93266	0.94249	0.92617	0.94778	0.93864	0.84524
Recall	0.0	0.14013	0.07881	0.41476	0.46754	0.36	0.54006	0.73304
Precision	0.0	0.01426	0.04878	0.21546	0.17808	0.22183	0.23317	0.10015
F-measure	0.0	0.02589	0.06026	0.28359	0.25793	0.27451	0.32571	0.17623

Table 5.3: Performance results of the IDS with of Replay attack (35-50 packets for the pattern).

5.1.3 Experiments with Multiple Attack

The last experiment, to check the performance of the IDS with basics attacks, is the one performed with Multiple attacks. In the current experiment, we use an attack database with random mix attacks, where there are attacks with valid and non-valid payloads. We can see the performance results of the IDS in Table 5.4.

<i>Multiple Attack</i>	Isolation Forest	OCSVM	NN	Small LSTM	Large LSTM	Small GRU	Large GRU	CANnolo
Accuracy	0.87573	0.79582	0.94389	0.93550	0.91758	0.94212	0.92973	0.83737
Recall	0.20099	0.38621	0.38799	0.39130	0.39203	0.32018	0.46399	0.82294
Precision	0.04936	0.05187	0.18239	0.15759	0.12118	0.15514	0.16166	0.11038
F-measure	0.07926	0.09147	0.24813	0.22469	0.18513	0.20901	0.23978	0.19466

Table 5.4: Performance results of the IDS with of Multiple Attack.

Comparing Table 5.4, Table 5.2, and Table 5.1 the performance with Multiple attack is better than in the Replay Attacks but worse than the Fuzzy attacks. We expected similar results because of the diverse nature of the attack. Again the model with better detection is CANnolo.

5.2 IDS performance with sophisticated attacks

5.2.1 IDS performance with WGAN attack

In the current section, we want to show that WGAN attacks can evade some IDS. For performing this experiment, we feed the IDS directly with the samples generated by the WGAN generator. It is worth mentioning that we could also insert the messages in the dataset using the RE-READ but it is time-consuming, and the conclusions extracted from the experiment would be similar. We have inserted 16,000 packets in each IDS for having a consistent result. In case the user provides her IDS, she should use the RE-READ for inserting the attacks in the database and then use her feature extraction method for testing this attack.

We divide the experiment results with the samples extracted from the non-time series and the time-series WGAN. It is worth mentioning that the time-series WGAN takes more time to learn than the non-time-series WGAN because it has more features to predict. In Table 5.5 we can see the percentage of the injected packets, which came from non-time-series WGAN and time-series WGAN, that have been able to deceive the IDS.

The results in Table 5.5 show again that the IDS that does not consider the time-sequences are easier to bypass. The reason is that if WGAN learns to craft packets that have valid payloads. Thus, the WGAN attack has similar behavior to the Replay attack. The models that consider time-series can detect the non-time-series WGAN attack. On the other hand, the time-series WGAN can deceive the models that consider time-series, except CANnolo.

Another observation is that the time-series WGAN has worse performance than the non-time-series

	WGAN	Time-series WGAN
IF	99,99%	89,86%
OCSVM	94,53%	99,98%
NN	82,84%	72,83%
Small LSTM	9,04%	37,61%
Large LSTM	43,55%	44,74%
Small GRU	17,24%	36,99%
Large GRU	19,98%	33,08%
CANnolo	8,33%	5,00%

Table 5.5: Percentage of packets bypassed through the IDS with WGAN.

WGAN in the IF, OCSVM, and NN. The reason is that the time-series WGAN needs more time to be trained than the non-time-series. As the time-series WGAN has more features to predict, it is challenging to have a similar performance.

Although the WGAN attack bypass tampered packets in almost all the IDS, we are crafting an attack without a final goal. We do not want that the vehicle changes the steering angle or to turn off the stereo. For that reason, we cannot realize the effect of such an attack in an authentic environment.

5.2.2 IDS performance with WGAN and Bit BIM variant attack

We want to show in the current section that by taking WGAN as a generator together with the Bit BIM algorithm, we can bypass some IDS with better performance than only with WGAN. In a similar manner to section 5.2.1, we feed the IDS directly with the attack samples crafted with the Bit BIM variant in NN. For performing the evaluation tests, we have used 16,000 samples, and the maximum number of iterations for the Bit BIM variant method is 1,500. In Tables 5.6 and 5.7, we can see the percentage of packets that evade the IDS with the no-time-series Bit BIM method.

<i>Non-time-series WGAN</i>	WGAN	WGAN+VBIM
IF	99,99%	94,53%
OCSVM	94,53%	99,99%
NN	82,84%	92,99%
Small LSTM	9,04%	9,04%
Large LSTM	43,55%	43,55%
Small GRU	17,24%	17,24%
Large GRU	19,98%	19,98%
CANnolo	8,33%	4,88%

Table 5.6: Percentage of packets bypassed through the IDS with Bit BIM variant together with non-time-series WGAN.

The goal of the Bit BIM variant is to modify the packets that were correctly classified by the NN model. After the modification, the packets bypass the IDS. For that reason, we can highlight the increase of deceived packets in that model. The transferability of the samples to other models is not decisive. We can observe in Tables 5.7 and 5.6 that there is not a noticeable increase of the percentage with and without the Bit BIM variant in LSTM, GRU, and CANnolo. The reason is that the Bit BIM variant does not take into consideration the time sequences. Furthermore, we perform the method only in the NN model, and then we transfer the same samples to the other model. For that reason, we can see in NN but no in the other models.

<i>Time-series WGAN</i>	WGAN	WGAN+VBIM
IF	89,86%	91,69%
OCSVM	99,98%	99,99%
NN	72,83%	97,51%
Small LSTM	37,61%	40,38%
Large LSTM	44,74%	47,37%
Small GRU	36,99%	38,04%
Large GRU	33,08%	34,91%
CANnolo	5,00%	4,88%

Table 5.7: Percentage of packets bypassed through the IDS with Bit BIM variant together with time-series WGAN.

Although we thought that the time-series Bit BIM variant would have better performances, the result is the opposite. After passing the WGAN samples by the time-series Bit BIM variant, OCS, IF, and NN had poorer results. In addition, LSM, Gru, and CANnolo have a percentage of 0% deceived frames. The results are not valid, and we need to redesign the method for time series.

5.3 Experimental evaluation summary

In the current chapter, we have exposed the results of the test experiments with basics CAN attacks and with sophisticated attacks.

Testing the IDS with basics attacks, we showed that the IDS with OCSVM and IF does not have an acceptable accuracy either recall. On the other side, the IDS build with auto-encoders has sufficient accuracy and recall results. We highlight the excellent recall of CANnolo with all the basics attacks but with less accuracy than the others auto-encoder IDS. We showed that the models which consider time-sequences are able to detect attacks with valid payloads whereas the other models only detect the non-valid payload attacks.

From the sophisticated attacks tests, we can conclude that the models that do not consider the time-sequences are easier to bypass. The last outcome extracted is that the crafted sophisticated attacks have been able to deceive, to some sort of degree, the majority of models, except CANnolo.

Chapter 6

Conclusions

In our work, we presented a framework for benchmarking the IDS. Our framework evaluates the IDSs capability to detect the basics attacks and the ease of bypassing them. Furthermore, the user can compare distinct IDSs by the metrics accuracy, recall, precision, and f-measure.

We developed eight different IDS models for testing our framework, all trained in an unsupervised manner. We successfully built a tool for crafting a database with the basics attacks on CAN (CANtack). We tested those IDS using attacks with non-valid payload, valid payload, and a mix of them. We compared the performance results between the analyzed IDS and we comprehend that the most suitable models for building the IDS are the models that consider the time-sequences because they can detect attacks with valid payloads. Although the performance results with basic attacks are not excellent, they are acceptable for performing experiments that aim to bypass the IDS.

We have successfully bypassed various IDS with WGAN attacks and time-series WGAN attacks. We showed that Time-series WGAN increases attack performance in IDS models that consider the time sequences. Furthermore, joining the WGAN with the bit BIM variant, we showed that we can perform a forceful attack in the models that do not consider the time sequences. The IDS which we could not deceive is CANnolo IDS.

The most substantial limitation of our work is that we have not used real IDS in our benchmarking. This limitation is critical because there exist better IDSs than the ones crafted in our work. In future works, we should test IDS with better performance to know if they are easy to bypass or not. In addition, we could adapt the existent IDS models to consider the time-sequences, i.e., exist some OCSV and IF that are adapted to time-sequences[27].

Another limitation is that we have only build few types of attacks for bypassing the IDS. In future works, we could use other evasion attacks as Carlini and Wagner[10] or adapt the Universal Adversarial Perturbations[33] to the automotive sector. Furthermore, we could use poisoning attacks to bypass the IDS when IDS is training.

In actual times we are in a technology race, and all the companies want to have the most innovative products. The automotive sector is integrating the newest technology in its ecosystems, and one of the hottest topics is autonomous cars. Autonomous cars need to be secure before being launched. For that reason, we think that such a line of investigation will be further explored.

Bibliography

- [1] O. Y. Al-Jarrah et al. “Intrusion Detection Systems for Intra-Vehicle Networks: A Review”. In: *IEEE Access* 7 (2019), pp. 21266–21289. DOI: [10.1109/ACCESS.2019.2894183](https://doi.org/10.1109/ACCESS.2019.2894183).
- [2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein GAN”. In: (Jan. 2017). URL: <https://arxiv.org/abs/1701.07875>.
- [3] Dor Bank, Noam Koenigstein, and Raja Giryes. *Autoencoders*. 2020. arXiv: [2003.05991](https://arxiv.org/abs/2003.05991) [cs.LG]. URL: <https://arxiv.org/abs/2003.05991>.
- [4] Ivo Berger et al. “Comparative Study of Machine Learning Methods for In-Vehicle Intrusion Detection”. In: *Computer Security*. Ed. by Sokratis K. Katsikas et al. Springer International Publishing, 2019, pp. 85–101. ISBN: 978-3-030-12786-2. URL: https://link.springer.com/chapter/10.1007/978-3-030-12786-2_6.
- [5] Battista Biggio and Fabio Roli. “Wild patterns: Ten years after the rise of adversarial machine learning”. In: *Pattern Recognition* 84 (Dec. 2018), 317–331. ISSN: 0031-3203. DOI: [10.1016/j.patcog.2018.07.023](https://doi.org/10.1016/j.patcog.2018.07.023). URL: <http://dx.doi.org/10.1016/j.patcog.2018.07.023>.
- [6] Battista Biggio et al. “Evasion Attacks against Machine Learning at Test Time”. In: Jan. 2013, pp. 387–402. ISBN: 978-3-642-38708-1. DOI: [10.1007/978-3-642-40994-3_25](https://doi.org/10.1007/978-3-642-40994-3_25). URL: <https://arxiv.org/abs/1708.06131>.
- [7] Ali Borji. “Pros and Cons of GAN Evaluation Measures”. In: *Computer Vision and Image Understanding* 179 (Feb. 2018). DOI: [10.1016/j.cviu.2018.10.009](https://doi.org/10.1016/j.cviu.2018.10.009).
- [8] ROBERT BOSCH. “CAN Specification 2.0: Protocol and Implementations”. In: (1991). DOI: <https://doi.org/10.4271/921603>. URL: <http://esd.cs.ucr.edu/webres/can20.pdf>.
- [9] Lars Buitinck et al. “API design for machine learning software: experiences from the scikit-learn project”. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.
- [10] Nicholas Carlini and David Wagner. “Towards Evaluating the Robustness of Neural Networks”. In: May 2017, pp. 39–57. DOI: [10.1109/SP.2017.49](https://doi.org/10.1109/SP.2017.49). URL: <https://arxiv.org/abs/1608.04644>.
- [11] Feng Chen et al. *Few-Features Attack to Fool Machine Learning Models through Mask-Based GAN*. 2019. arXiv: [1911.06269](https://arxiv.org/abs/1911.06269) [cs.LG].
- [12] Valliappa Chockalingam et al. “Detecting Attacks on the CAN Protocol With Machine Learning”. In: 2016.
- [13] T. Costa. *Benchmarking framework for CAN IDS*. 2021. URL: https://bitbucket.org/necst/costa_thesis_code/src/master/.
- [14] R. De Maesschalck, D. Jouan-Rimbaud, and D.L. Massart. “The Mahalanobis distance”. In: *Chemometrics and Intelligent Laboratory Systems* 50.1 (2000), pp. 1–18. ISSN: 0169-7439. DOI: [https://doi.org/10.1016/S0169-7439\(99\)00047-7](https://doi.org/10.1016/S0169-7439(99)00047-7). URL: <https://www.sciencedirect.com/science/article/pii/S0169743999000477>.

- [15] Donald Firesmith. *Common Concepts Underlying Safety, Security, and Survivability Engineering*. CMU/SEI-2003-TN-033. Pittsburgh, PA, 2003. URL: <http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=6553>.
- [16] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *International Conference on Learning Representations*. 2015. URL: <http://arxiv.org/abs/1412.6572>.
- [17] Ian Goodfellow et al. “Generative Adversarial Networks”. In: *Advances in Neural Information Processing Systems* 3 (June 2014). DOI: [10.1145/3422622](https://doi.org/10.1145/3422622).
- [18] Tianxiang Huang, Jianying Zhou, and Andrei Bytes. “ATG: An Attack Traffic Generation Tool for Security Testing of In-vehicle CAN Bus”. In: Aug. 2018, pp. 1–6. ISBN: 978-1-4503-6448-5. DOI: [10.1145/3230833.3230843](https://doi.org/10.1145/3230833.3230843). URL: <https://dl.acm.org/doi/pdf/10.1145/3230833.3230843>.
- [19] Yasuyuki Kawanishi et al. *Detailed Analysis of Security Evaluation of Automotive Systems Based on JASO TP15002*. Springer International Publishing, 2017, pp. 211–224. ISBN: 978-3-319-66284-8.
- [20] K.Koscher et al. “Experimental Security Analysis of a Modern Automobile”. In: (2010), pp. 447–462. DOI: [10.1109/SP.2010.34](https://doi.org/10.1109/SP.2010.34).
- [21] Pierre Kleberger, T. Olovsson, and E. Jonsson. “Security aspects of the in-vehicle network in the connected car”. In: *2011 IEEE Intelligent Vehicles Symposium (IV)* (2011), pp. 528–533.
- [22] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. *Adversarial examples in the physical world*. 2017. arXiv: [1607.02533](https://arxiv.org/abs/1607.02533) [cs.CV]. URL: <https://arxiv.org/abs/1607.02533>.
- [23] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. “Adversarial Machine Learning at Scale”. In: (Nov. 2016). URL: <https://arxiv.org/abs/1611.01236>.
- [24] Zilong Lin, Yong Shi, and Zhi Xue. *IDSGAN: Generative Adversarial Networks for Attack Generation against Intrusion Detection*. 2019. arXiv: [1809.02077](https://arxiv.org/abs/1809.02077) [cs.CR].
- [25] Fei Tony Liu, Kai Ming Ting, and Zhi hua Zhou. “Isolation Forest”. In: *In ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining. IEEE Computer Society*, pp. 413–422. URL: <https://cs.nju.edu.cn/zhoush/zhoush.files/publication/icdm08b.pdf?q=isolation-forest>.
- [26] S. Longari et al. “CANnolo: An Anomaly Detection System based on LSTM Autoencoders for Controller Area Network”. In: *IEEE Transactions on Network and Service Management* (2020), pp. 1–1. DOI: [10.1109/TNSM.2020.3038991](https://doi.org/10.1109/TNSM.2020.3038991).
- [27] Junshui Ma and S. Perkins. “Time-series novelty detection using one-class support vector machines”. In: vol. 3. Aug. 2003, 1741–1745 vol.3. ISBN: 0-7803-7898-9. DOI: [10.1109/IJCNN.2003.1223670](https://doi.org/10.1109/IJCNN.2003.1223670).
- [28] Pankaj Malhotra et al. “LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection”. In: *CoRR* abs/1607.00148 (2016). arXiv: [1607.00148](https://arxiv.org/abs/1607.00148). URL: <http://arxiv.org/abs/1607.00148>.
- [29] M. Marchetti and D. Stabili. “READ: Reverse Engineering of Automotive Data Frames”. In: *IEEE Transactions on Information Forensics and Security* 14.4 (2019), pp. 1083–1097. DOI: [10.1109/TIFS.2018.2870826](https://doi.org/10.1109/TIFS.2018.2870826).
- [30] Stephen Checkoway Damon McCoy et al. “Comprehensive experimental analysis of automotive attack surfaces”. English (US). In: (Jan. 2011). Conference date: 08-08-2011 Through 12-08-2011, ”77–92.
- [31] Luke Metz et al. *Unrolled Generative Adversarial Networks*. 2017. arXiv: [1611.02163](https://arxiv.org/abs/1611.02163) [cs.LG].
- [32] Xiuliang Mo et al. “Anomaly Detection of Vehicle CAN Network Based on Message Content”. In: June 2019. DOI: [10.1007/978-3-030-21373-2_9](https://doi.org/10.1007/978-3-030-21373-2_9).

- [33] S. Moosavi-Dezfooli et al. “Universal Adversarial Perturbations”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 86–94. DOI: [10.1109/CVPR.2017.17](https://doi.org/10.1109/CVPR.2017.17).
- [34] Simon Msika, Alejandro Quintero, and Foutse Khomh. *SIGMA : Strengthening IDS with GAN and Metaheuristics Attacks*. 2019. arXiv: [1912.09303](https://arxiv.org/abs/1912.09303) [cs.CR].
- [35] Michael Müter, André Groll, and F. Freiling. “A structured approach to anomaly detection for in-vehicle networks”. In: *2010 Sixth International Conference on Information Assurance and Security* (2010), pp. 92–98.
- [36] N. Navet et al. “Trends in Automotive Communication Systems”. In: *Proceedings of the IEEE* 93.6 (2005), pp. 1204–1223. DOI: [10.1109/JPROC.2005.849725](https://doi.org/10.1109/JPROC.2005.849725).
- [37] D. Nilsson, Phu H. Phung, and Ulf Larson. “Vehicle ECU classification based on safety-security characteristics”. In: 2008.
- [38] N. Papernot et al. “The Limitations of Deep Learning in Adversarial Settings”. In: *2016 IEEE European Symposium on Security and Privacy (EuroS P)*. 2016, pp. 372–387. DOI: [10.1109/EuroSP.2016.36](https://doi.org/10.1109/EuroSP.2016.36). URL: <https://arxiv.org/abs/1511.07528>.
- [39] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. “Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples”. In: (May 2016). URL: <https://arxiv.org/abs/1605.07277>.
- [40] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. USA: Addison-Wesley Longman Publishing Co., Inc., 1984. ISBN: 0201055945.
- [41] David Powers. “Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness and Correlation”. In: *Mach. Learn. Technol.* 2 (Jan. 2008). URL: https://web.archive.org/web/20191114213255/https://www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf.
- [42] Foster Provost, Tom Fawcett, and Ron Kohavi. “The Case Against Accuracy Estimation for Comparing Induction Algorithms”. In: *Proceedings of the Fifteenth International Conference on Machine Learning* (Apr. 2001). URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.49.5218&rep=rep1&type=pdf>.
- [43] *Road vehicles — Controller area network (CAN)*. Mar. 2015.
- [44] Tim Salimans et al. *Improved Techniques for Training GANs*. 2016. arXiv: [1606.03498](https://arxiv.org/abs/1606.03498) [cs.LG]. URL: <https://arxiv.org/abs/1606.03498>.
- [45] Divya Saxena and Jiannong Cao. “Generative Adversarial Networks (GANs): Challenges, Solutions, and Future Directions”. In: *ArXiv abs/2005.00065* (2020).
- [46] Bernhard Schölkopf et al. “Estimating Support of a High-Dimensional Distribution”. In: *Neural Computation* 13 (July 2001), pp. 1443–1471. DOI: [10.1162/089976601750264965](https://doi.org/10.1162/089976601750264965). URL: <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-99-87.pdf>.
- [47] E. Seo, H. M. Song, and H. K. Kim. *Car-Hacking Dataset for the intrusion detection*. 2018.
- [48] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. arXiv: [1409.3215](https://arxiv.org/abs/1409.3215) [cs.CL].
- [49] Christian Szegedy et al. “Intriguing properties of neural networks”. In: (Dec. 2013). URL: <https://arxiv.org/abs/1312.6199>.
- [50] A. Taylor, S. Leblanc, and N. Japkowicz. “Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks”. In: *2016 IEEE 3rd International Conference on Data Science and Advanced Analytics (DSAA)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2016, pp. 130–139. DOI: [10.1109/DSAA.2016.20](https://doi.org/10.1109/DSAA.2016.20). URL: <https://doi.ieeecomputersociety.org/10.1109/DSAA.2016.20>.

- [51] A.Nichelini T.Costa C.A Pozzoli. *CANtack, synthetic CAN attacks generator*. 2021. URL: https://bitbucket.org/necst/attack_tool_code.
- [52] Elizabeth Walter. *The Cambridge Dictionary*. Cambridge University Press, 1999.
- [53] M. Khalid Weiyng Zeng and Sazzadur Chowdhury. “In-Vehicle Networks Outlook: Achievements and Challenges”. In: *IEEE Communications Surveys and Tutorials* (2016).
- [54] Marko Wolf, André Weimerskirch, and Christof Paar. “Security in automotive bus systems”. In: *IN: PROCEEDINGS OF THE WORKSHOP ON EMBEDDED SECURITY IN CARS (ESCAR)'04*. 2004.
- [55] Mattia Zago et al. “ReCAN - Dataset for Reverse engineering of Controller Area Networks”. In: *Data in Brief* 29 (Jan. 2020), p. 105149. DOI: [10.1016/j.dib.2020.105149](https://doi.org/10.1016/j.dib.2020.105149).
- [56] W. Zeng, M. A. S. Khalid, and S. Chowdhury. “In-Vehicle Networks Outlook: Achievements and Challenges”. In: *IEEE Communications Surveys Tutorials* 18.3 (2016), pp. 1552–1571. DOI: [10.1109/COMST.2016.2521642](https://doi.org/10.1109/COMST.2016.2521642).
- [57] Haichun Zhang et al. “CANsec: A Practical in-Vehicle Controller Area Network Security Evaluation Tool”. In: *Sensors* 20.17 (2020). ISSN: 1424-8220. DOI: [10.3390/s20174900](https://doi.org/10.3390/s20174900). URL: <https://www.mdpi.com/1424-8220/20/17/4900>.
- [58] Valentin Zieglmeier et al. “A real-time remote IDS testbed for connected vehicles”. In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing* (Apr. 2019). DOI: [10.1145/3297280.3297465](https://doi.org/10.1145/3297280.3297465). URL: <http://dx.doi.org/10.1145/3297280.3297465>.