POLITECNICO DI TORINO

Master Degree Thesis

Network topology description language and communication policy analysis in the automotive scenario

Supervisors
Prof. Cataldo Basile
Prof. Antonio Lioy

Candidates
Umberto Fierro
matricola: s255434

Anno accademico 2020-2021

To my grandparents

Summary

In recent times, the automotive sector has undergone radical changes as regards to cybersecurity problems, eventually becoming a sector of interest for companies and researchers. Problems concerning automotive industry in fact revolved mainly around the safety of vehicles themselves until few years ago, whereas nowadays cybersecurity has to be also taken into account, since vehicles contain from 30 up to 100 different Electronic Control Units (ECUs). Each one of these ECUs control a specific module (from the brakes to the steering wheel, to the managing of multimedia contents) reading data from the sensors (e.g., wheel speed, tire pressure, etc.) and producing actions through the actuators (e.g., pulling the brakes, turning on/off a lamp, etc.). We can therefore affirm that the safety of a vehicle goes through the proper implementation of cybersecurity of these computing units. A further crucial aspect to underline is that these ECUs are interconnected, thus creating a bigger attack surface that can be exploitable by attackers in order to conduct attacks against vehicles. For these reasons Original Equipment Manufacturer (OEMs) have to setup their objectives in order to deal with this issue and be more competitive, thus applying security by design and being compliant to the latest international regulations with regard to cybersecurity.

This thesis takes the inputs provided by these normatives (e.g., *ISO/SAE 21434* and *UNECE WP.29/R155*) and describes the development of a possible topological representation of structural elements of an automotive architecture, identifying assets of interest based on an ontology. Furthermore a study of an access control model using communication policies follows; this stage involves: High level policies formalization starting from design constraint, through refinement process extract from them Low level policies that should be used in order to configure filtering devices and in the end, with the aim of avoid misconfiguration of filtering element, anomaly analysis and reachability analysis are made.

Conclude this work an analysis of results obtained and how they can be used in future development

Acknowledgements

This is the last act of my academic experience, a very hard path that gave me the chance to grow as a person, enlarge my knowledge, and acquire technical skills. It may seem reductive, but all the gratitude to the people that have surrounded me is contained in a six-letter word that I have used unproperly so many times in the last years: **Ubuntu**, which means *"I am what I am because of who we all are."*

I would like to acknowledge my thesis supervisors: Prof. Basile and Prof. Lioy. They gave me the honor to work with them in a very innovative and interesting field. The virtual door to Prof. Basile's office was always open whenever I ran into a trouble spot or had a question about my research or writing. I am so grateful for allowing me to take part in this project and for your endless patience and willingness.

I would like to acknowledge Giuseppe and all the guys of *Drivesec* team, in particular Nino and Andrea. They make me feel part of the Drivesec family since the first day. I hope that our path together will be full of satisfaction.

My family played the most important role in these years, allowing me to start this journey and being always by my side, no matter the distance between us. Thus I would like to thank my dad, for bequeathing me with the love and the passion for the project development, and my mum, for all the common sense she gave me. Thank you for all the love and trust given, if I am here to write this thesis is basically because of you. To my little sister, thank you for coloring my everyday life with your smile. I would like to thank my grandparents, the roots of my family. This work is dedicated to you, thank you for the future you gave me.

I would thank all of my friends, no one excluded, lost and found, near and far, boring and funny, thank you for all of your advice and the beers shared. Thank you for all the very philosophical discussion above the maximum systems but also the rudest football comments. You allowed me to grow and to overcome the darkest moments and be here now.

Dulcis in fundo, I would thank Sara, my diamond in the rough. Thank

you, baby, for the endless support, all the moments, words, and emotions shared together, and for all who will be.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In an open and connected world, the automotive scenario is a thriving area for companies and researchers involved in cybersecurity. If the problems in the automotive industry turned around the safety of vehicles themselves until a few years ago, these aspects are now accompanied by cybersecurity issues since the vehicles contain from 30 up to 100 different Electronic Control Units (ECUs) [1]. Each one of them controls a specific module [2] from the brakes to the streaming of multimedia content through a set of sensors and actuators, that respectively read data from the environment (e.g., wheel speed, tire pressure, steering angle, etc.) and produce a specific action (e.g., pulling the brake, turning on/off a lamp, etc.). We can threfore affirm that nowadays the safety of vehicle goes through the proper implementation of cybersecurity of these computing units. A further crucial aspect to underline is that these ECUs are interconnected, which means a bigger attack surface that can be exploitable by attackers in order to perform attacks against vehicles. For these reasons Automotive OEMs[1], i.e. companies that produce parts and equipments that may be marketed by another manufacturer [3], have to set up their objectives to fix this issue and be more competitive. In particular, they have to:

- *Be compliant with standards and regulations* [4] to address investment in the proper direction

- *Increase reliability of vehicles*, security by design can in fact avoid the detection of vulnerability after the vehicles are sold, thus preventing

[1]Original Equipment Manufacturer

a potential negative impact generated by negative press coverage and resulting recall campaigns [2]

- *Apply cybersecurity principles to the entire production chain* [5], it must be included from the design phase to the software development and maintenance until the end of the vehicle's life. It shall also be integrated into Hardware, Software, and protocols used for intra-vehicle and inter-vehicle communications.

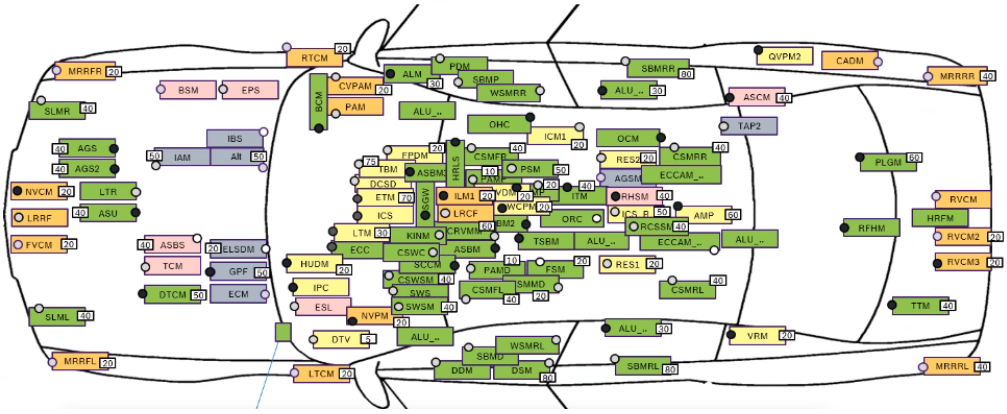


Figure 1.1: Example of Automotive Architecture.

1.1 Open problem

From a cybersecurity point of view, modern automotive architectures are not so different from a computer network of the beginning of the century. In this scenario, the main goal is in fact to keep the computing unit extremely simple in order not to compromise their performances. This fact highlights the open problems automotive cybersecurity is facing today, which will described in the following sections.

Lack of security property in communication protocols

Messages in the automotive network are exchanged using different protocols, but regarding security properties [6], none of them provides authentication of packets, that is to say that anyone can write data on the bus pretending to be a proper ECUs. Messages also lack of confidentiality because they are sent in clear on the bus, making a sniffing activity of an attacker easy to conduct.

Lack of access control policies or misconfiguration

In Section 2.1.3 at page 7 we will focus on the structure and the role of the Security Gateway in an Automotive infrastructure, but it is necessary to state from now how this is a crucial element. Since communication protocols do not guarantee security properties and most of them are broadcast, there is the need for a filtering element that can assure a correct flow of admitted packets, discarding the others. For this reason, a correct analysis of policy and a deep refinement activity represent a fundamental step to assure the safety of the vehicle through the correct implementation of cybersecurity

1.2 Thesis Outline

After this merely introductive chapter, the topic of my final work will be presented, analyzing the background and then highlighting the problem and its solution. In particular:

- In Chapter 2 I will introduce the basic concepts of the Automotive world focusing on Architectural and normative aspects.
- In Chapter 3 I will introduce the research field of my thesis, explaining which are the possible attacks an attacker can mount, underlining possible attack surfaces, attack vectors, and typologies of attacks.
- In Chapter 4 I will discuss works and studies related to this Master Degree Thesis and describe the theoretical background the solution to the problem relies on
- In Chapter 5, I will concisely describe the problem by defining the requirements and the scenarios in which it finds application.
- In Chapter 6, I will give details about the design of the solution, describing in detail all the sub-parts it is made of and which project choices are considered.
- In Chapter 7, I will further analyse in the refinement algorithms and policy analyzing functions, describing their implementation.
- In Chapter 8, I will propose a proof of concepts to test some use cases scenarios, in order to demonstrate that the requirements have been met

- In Chapter 9, I will evaluate the outcome obtained from this final work, describing the results achieved and discussing possible future development

1.3 Thesis Results

The goal that this thesis aims to reach is mainly to propose a description language that allows to identify items of interest, from a cybersecurity point of view, inside an automotive architecture and methods demonstrates that the policy analysis used in a normal computer network is suitable for automotive network as well, with minor changes imposed by the context. It tries to give basic components to develop tools useful to be compliant with the most recent automotive cybersecurity certification.

Chapter 2

Automotive Scenario

The goal of this chapter is to contextualize the problems highlighted in Chapter 1 presenting the technical aspects that characterize a vehicle and more in general the Automotive world. In this way, the reader can receive all the necessary knowledge to understand the following chapters. In particular, I will focus on two main aspects:*Architecture* (Section 2.1) and *Network Protocol* (Section 2.2). At the end of the chapter a presentation of the most recent standards about Automotive Cybersecurity can be found (Section 2.3).

2.1 Automotive architecture

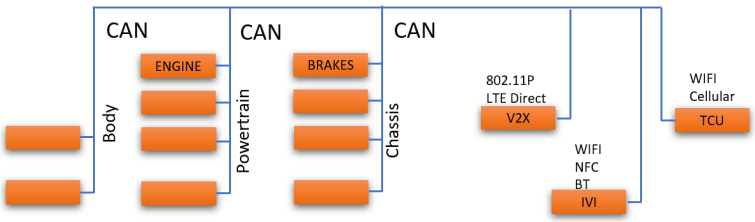
In the automotive industry, there are no real design standards and some architectural aspects can differ from OEM to OEM. In this section I will try to describe them conceptually. First of all, we have to distinguish three main models of automotive architecture (Figure 2.1):

1. **Flat Architecture**, it is the very first typology of architecture created and it is characterized by a single level of ECU: each one of them are interconnected to a Hub that propagated the CAN frame and there is no filtering element (Figure 2.1a).

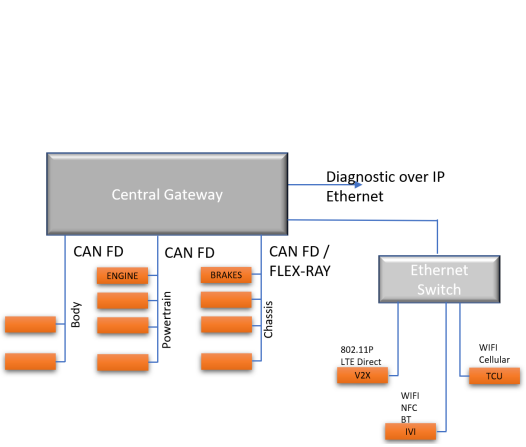
2. **Gateway Architecture**, the first evolution requires the presence of a central gateway that filters packets that cross them, providing a network separation (Figure 2.1b).

3. **Domain Controller Architecture**, it is the next-generation architecture that can observed in the design of vehicles from 2022. It is based

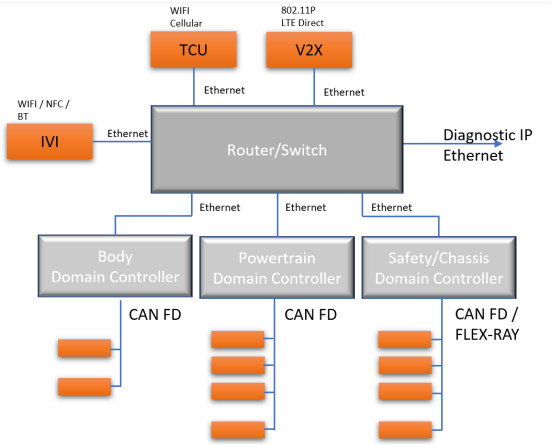
on a new component, i.e. the *Domain Controller*, which physically integrates more than a control function in the same ECU, reducing the space needed to allocate more different ECU, but introducing Cybersecurity issues that must be taken into account to protect safety-critical operation (Figure 2.1c)



(a) *Flat Architecture*



(b) *Gateway Architecture*



(c) *Domain Controller Architecture*

Figure 2.1: Evolution of Automotive Architecture

In the following part, we will concentrate only on the second model, since it is the currently most used architecture.

2.1.1 Network

Since Electronics components were introduced, different communication areas were defined to make a more rational design and a network segregation. Historically the main networks are [7]:

- **Powertrain**, here the safety-critical ECUs used for engine, transmission, emissions control, fuel economy, and performance control can be found
- **Chassis**, another set of safety-critical ECU responsible for braking function, Airbag activation, steering lock, etc. belongs to this network
- **Body**, this area is filled for the most part with not safety-critical components, that control for example door or lamp.

Sometimes a fourth network is introduced to classify other particular ECUs like Telematic Unit, V2X, or Head Unit. In other cases, the Designer chooses to attach them directly to a Gateway or rarely to insert them in one of the previously presented networks.

2.1.2 ECU

After defining the macrostructure of the vehicle architecture, we will now concentrate on the active elements that compose them, namely the ECUs. Their structure is similiar [8], they have some sensors, actuators, various interfaces, and microprocessors or a microcontroller. At the software level, they have at least a firmware, some of them have a real Operating System that runs on application [9]. ECU can be divided into two main categories: *Safety Critical and Not Safety Critical* [7]. The former controls basic vehicle functions, like braking and engine control, have a real impact on the safety of the vehicle and on the health of the passengers, in case of impairment. The latter control accessory functions instead, like Infotainment or light control. Table 2.1 summarizes the functions of the main ECUs indicating the network to which they belong.

2.1.3 Security Gateway

The Central element of Gateway Architecture is the *Security Gateway* (also called *Automotive Gateway*), it is a particular ECU, that differs from the others because it does not have any sensors or actuators and does not make any "visible" action (like braking or accelerate). It securely and reliably interconnects and transfers data across many different networks inside the vehicle [10], providing physical isolation and protocol translation [11] to route signals between functional domains [12]. For these reasons it has various network interfaces like *CAN, FlexRay, Ethernet, OBD-II*. It has the same role as a packet filter in a modern network because it contains access policies, allowing only admitted communication between ECUs and networks.

Table 2.1: Automotive architecture basic component

ECU	Network	Description
ECM	Powertrain	Engine Control Module controls the engine using information from sensors to determine the amount of fuel, ignition timing, and other engine parameters.
TPMS	Powertrain	Tire Pressure Monitoring System sends tire pressure data to other ECUs
EBCM	Chassis	Electronic Brake Control Module controls motor pump and valves, preventing brakes from locking up and skidding by regulating hydraulic pressure.
TCM	Powertrain	Transmission Control Module controls electronic transmissionusing data from sensors and the ECM to determine when and how to change gears.
BCM	Body	Body Control Module controls various vehicle functions, provides information to occupants, and acts as a firewall between the two subnets.
HVAC	Body	Heating, Ventilation, Air Conditioning controls cabin environment
ADAS	Body	Advance Driver Assistance System is an ECU that groups different functionality designed to assist drivers using sensors and cameras (e.g., obstacles detection, lane assistant, ecc.)
TCU	Other	Telematic Control Unit enables data communication via LTE, BT, and Wi-Fi.
Infotainment	Other	In addition to regular radio functions, it allows BT/NFC/Wi-Fi connections in order to stream multimedia content and shows some vehicle status
V2X	Other	Vehicle-to-everything, allows communication with external entity of interest, like infrastructure (V2I) or vehicle (V2V)

2.2 Network Protocol

Regarding network protocol used in automotive, there are many different standards. The most used is CAN, however OEMs use various network protocols to diversify network domains.

CAN

Controller Area Network (CAN bus) is the most common vehicle bus standard designed to allow microcontrollers and devices to communicate with each other's applications without a host computer [13]. Communication is made by sending in broadcast messages that can be received by every ECUs in the network but processed only by the interested ones and eventually filtered by Security Gateway. A CAN message [14] has an identifier of 11 bits and at most an 8 Bytes long payload. There is no authenticator fields, no encryption, and just a CRC code to retrieve possible error in transmission. Another peculiarity is that it does not include addresses in the traditional sense and the CAN ID is used to indicate the packet type. There is also an evolution of the standards called CAN-FD as an extension of ISO 11898, which has an extended header of 29 bits.

FlexRay

FlexRay is a high-speed bus that can communicate at speeds of up to 10Mbps. It's geared for real-time communication, such as drive-by-wire, steer-by-wire, brake-by-wire, and so on. FlexRay is more expensive to implement than CAN, which is the reason why, it is used only for high-end systems [9].

Automotive Ethernet

Since MOST and FlexRay are expensive to implement the majority of newer vehicles are moving towards Ethernet. The implementations can vary, but substantially they are very similar to the one that can be found in a standard computer network. Often, CAN packets are encapsulated as UDP. One of the advantages of Ethernet is the possibility to transmit data at speeds up to 10Gbps, without using proprietary protocols and choosing arbitrary topology [9].

2.3 Standards for Automotive

As it has been already stated at the beginning of Chapter 1, the automotive world was mainly focused on the safety of the vehicles in the past, but now we are currently witnessing a radical change. In the last two years, international organizations have in fact formalized the first draft of standards going in the direction of cybersecurity.

2.3.1 ISO/SAE 21434

It is a very innovative standard [15] proposed in 2016 and finally approved in 2021. It was elaborated by the *ISO*[1] jointly with *SAE*[2]. It provides a series of cybersecurity measures that must be followed and applied during the entire development of a road vehicle's lifecycle. Officially it will enter into force for new vehicles from 2022 and for new homologation from 2024 (for vehicles enrolled before 2022). So from that moment onwards, carmakers have to be compliant with ISO/SAE 21434 to place their vehicles on the market.

ISO/SAE 21434 was born with three specific purposes:

[1]International Organization for Standardization

[2]Society of Automotive Engineers

1. Defining cybersecurity policies and processes inside automotive industry for road vehicles

2. Managing cybersecurity risks

3. Speeding up the dissemination of cybersecurity culture among OEMs in the automotive sector

and it is threfore addressed to OEMs, managers and all those responsible for the design, the development, and the implementation of hardware and software systems in motor vehicles, and managers.

The main consideration regarding this normative is that it aims to be applied on every Cybersecurity relevant item and component inside the vehicle perimeter including aftermarket and service parts. Moreover, it provides a security in-depth approach, as the only way to mitigate threat scenarios and risks. For these reasons the standards describe a multi-layered security architecture, that must be implemented to improve the security controls and mitigate eventual attacks to the entire automotive infrastructure.

2.3.2 UNECE WP.29/R155

It is a normative [16] produced by *UNECE*[3], an international organization. They jointly with EU published in 2019 this document intended to give support to the ISO 21434, providing a possible interpretation and developing a certification for a "Cyber Security Management System" (CSMS), which is to be mandatory for the type approval of vehicles [17].

It has the objective to:

1. Focus on Cybersecurity, defining security aspects related to data protection and software updates

2. Guarantee vehicle safety in case of cyberattacks, addressing to vulnerability assessment and cyber threat identifications

Unlike ISO 21434, *UNECE WP.29/R155* has a more pragmatical approach, trying to design an entire process to assess the key threats and vulnerabilities inside the vehicle development perimeter and to propose possible mitigations. Another vital aspect is that it provides a threat analysis model based on the state of the art. The last important consideration regarding this normative

[3]United Nations Economic Commission for Europe

is that it refers to more concrete aspects of cybersecurity like the use of cybersecurity control, asset identification, the discovery of potential target of attacks and potential vulnerabilities, and possible mitigation. However, both normative are focused on the proper configuration of the productive process and on the principle of security in-depth as a golden rule to develop a standard security framework for automotive cybersecurity.

Chapter 3

Automotive cyber attacks

The problems highlighted in chapter 1 are exploited by malicious users because they know how to use the various attack surfaces available to mount dangerous attacks. In the following sections, I will describe the most studied in the literature (these concepts are summarized in Table 3.1).

3.1 Attack surface

In their work Checkoway et al. [18] experimentally defined a threat model of a car highlighting the attack surfaces that exposes. Although cars do not have a command-line prompt or a keyboard, there is still the possibility to interact with them through the huge variety of exposed interfaces.

- **OBD II**, it is the diagnostic port, used by car manufacturers or car maintainers to update software or to analyze the vehicle status. It also represents a privileged way for a malicious user that can directly "enter" into vehicle network, reading/writing information.

- **Multimedia connectivity**, nowadays a lot of cars have a rich user interface panel, used mainly to stream multimedia and entertainment contents. To enrich the offer they have several other communication interfaces, like Bluetooth, Wi-Fi, NFC, USB, in addition to the most common cd player, which can be other exploitable points of access.

- **V2X**, it is used for communications which involve the vehicle and a second party that can be other vehicles (V2V) or infrastructure (V2I). Since it is the door to the external world it can catch the attention of

malicious hackers. For these reasons, it requires security control to check Authentication and Integrity.

- **Telematic Unit**, it is a fundamental component in a vehicle, since it controls every kind of wireless communication, in particular Cellular network. They expose a GSM and an LTE antenna, so the vehicle should be reachable worldwide.
- **TPMS**, the ECU controlling the tire pressure contains a radio-frequency sensor, that sends data to the other ECUs.

3.2 Attack vector

After the definition of attack surfaces, we have to illustrate how an attacker can access these surfaces, in particular which mediums are needed. The following list contains the most common ones:

- **OTA Update**, since we are describing vehicle architecture like a "normal" computer network, with hardware devices that run operating system and apps, Over-The-Air updates are essential to correct a bug or to fix vulnerabilities. On the other hand, wrong management of this process, can make it extremely dangerous: software shall be signed in order to install only legitimate software, otherwise a malicious user can release fake updates and take control of the vehicle.
- **External Infrastructure**, this point is linked to the previous one, because aftermarket products, third partied products, or the car maintainer can be sources of insecurity just like a not proper update.
- **External Devices**, attack surfaces can also be exploited by using external devices like OBD-II connectors or BT-receiver, as well as the cheapest USB drive or even the old CD-ROM. An attacker can use these instruments connected to a PC or another programmable board to read and write data on the CAN bus, even remotely.

3.3 Typologies of attacks

Common classification of different typologies of attacks is made considering the distance between the attacker and the target vehicle, namely distinguishing if the attack is is conducted through an indirect access [18] or a direct access to the vehicle [8].

3.3.1 Direct physical access attack

As it is stated in [8] the majority of the direct access attacks are conducted by connecting to the CAN bus or the OBD-II port, highlighting how they can easily attack a vehicle from inside and reach safety-critical ECUs [19] causing severe damages.

- **Message injection**, it consists in writing on the bus forged messages to the desired target ECU in order to force it to execute some action or to alter the messages transiting on it.
- **Message sniffing**, it consists in reading the messages travelling on the network, in order to discover what kind of messages a certain ECU can send or receive, which are the answers of the destination target, which are the possible structures of the payload, etc. In general, This phase can generally be useful for reverse-engineering the automotive system and maybe acquiring the knowledge needed to perform another kind of attack.
- **Denial of Service (DoS)**, like every normal network, it is possible to make it to make it partially or totally unusable by sending a huge number of messages, creating delays in receiving the correct ones, and performing the correct actions. Sometimes this attack takes place using some tool like CAN frame generator or fuzzer.
- **Malware Injection**, an attacker can try to exploit the vulnerability of the network and also the lack of authentication and integrity check to load malware on ECU. Moreover, a malicious user could flash a modified version of firmware on this component, jeopardizing or taking control of the functionality.

3.3.2 Indirect attack

The same research group that evaluated the direct access also investigated the attack mounted using different attack surfaces so that it can be performed through wireless interfaces or remotely [18].

Short-range wireless access

The attack in this section are mounted mainly using Bluetooth communication and therefore they are linked to the Telematic Unit and exploit the

vulnerability of the Bluetooth stack. We can consider that an attacker can use his smartphone in order to try to pair it with the vehicle in case of short distances (about 10 meters). First of all, they sniff the car's Bluetooth MAC address and then they pair their devices trying to use Brute-Force in order to retrieve the secret PIN used by Bluetooth in order to grant connection. At this point, they could mount an attack against the car's network by sending malware.

Long-range wireless access

Even in this case the attack exploits the Telematics Unit in particular its capabilities to be reachable from the outside. By using a signal analyzer it is possible to exploit vulnerabilities in cellular *Gateway* and its *Authentication system* demonstrated by Checkoway et al. [18] in their work :

- **Authentication system**, it is been observed that the random authentication challenge uses a random generator that has been re-initialized each time the Telematic Unit starts and uses always the same seed thus observing multiple calls and properly forging the message it is possible for an attacker to replay a response each time.
- **Gateway**, a buffer overflow vulnerability in protocol stack in particular in Command Program is exploitable; the only gap following this path is that authentication code is required. However, with a huge effort it is possible to succeed.

Once one of these vulnerabilities has been exploited an attacker can then inject messages on the Automotive network (e.g., 3.4).

Remote Control

Previous attacks can also be lead remotely by using receiver attached to the following interfaces:

- **Bluetooth**, it can be conducted using two accomplices and a paired device as a vector. In this way, the one inside the car pairs his device so that the second one can remotely attack the vehicle through the connected device thus mounting an attack against the car's network by sending malware.
- **TPMS**, its RFID can be exlpoited in order to control it via Radio; in this way, the code of TPMS can be reflashed or used in order to indirectly reflash another ECU.

- **OBD-II**, as stated in Subsection 3.3.1, OBD-II port allows direct access to the vehicle network; for this reason, an attacker can compromise it by attaching by attaching a remote controller to this port that admits to reading/writing CAN Frame remotely.

Table 3.1: Attack classification

Attack Surface	Tipology	Goal
TPMS	Message Injection	Send a arbitrary messages
TPMS	DoS	Causing fault in network
TPMS	Near-range attack	Track a vehicle based on the TPMS unique IDs
Cellular	Near-range attack	Access the internal vehicle network from anywhere
Cellular	DoS	Jam distress calls
Cellular	Long-range attack	Track the vehicle
Cellular	Remote Control	Use a cellular network to connect g to the remote diagnostic system
Wi-Fi	Long-range attack	Access the internal vehicle network from up to 250m
Wi-Fi	Malware Injection	Install malicious code
Wi-Fi	Message Sniffing	Sniff communications
Wi-Fi	Long-range attack	Track the vehicle
USB	Malware Injection	Install malware on the infotainment unit
USB	Malware Injection	Install modified update software on the vehicle
Bluetooth	Near-range attack	Access the vehicle from close ranges (less than 90m)
Bluetooth	Malware Injection	Execute code on the infotainment unit
Bluetooth	DoS	Jam the Bluetooth device
OBD-II	Remote Control	Plug directly to the port a receiver in order to control the vehicle from remote
OBD-II	Message Injection	Send a arbitrary messages
OBD-II	Message Sniffing	Read transit messages
V2X	Message Injection	Retransmit the V2X signal, delaying it, and broadcasting the signal in the same frequency
V2X	Message Sniffing	Spoof V2X signal
V2X	DoS	Jam the signal

3.4 Famous attacks

In this section I will report some of the most famous attacks conducted against vehicles, describing the evolution and real applications of the typologies of attacks described above.

Jeep Cherokee attack

Perhaps the most known demonstration of Automotive System insecurity is the famous attack conducted by two American computer security researchers, Charlie Miller and Chris Valasek, who are involved in Automotive Penetration Testing [20] since 2009. The most famous experiment took place in 2015, when they remotely hacked a 2014 Jeep Cherokee and took control of the brakes, the steering wheel, and the accelerator [21]. They exploited the FCA software installed in the Infotainment System of their model (Jeep Cherokee included) using the cellular network: all the people who know the car's IP address can in fact hijack the car, regardless of where they are. In the demonstration, they assumed control of the vehicle by sending CAN frame through the cellular network directed to different ECU, compromising the brakes, transmission, and other safety-critical components[1] from more than 10 miles away. In the end, Miller and Valasek signaled the flaws to FCA, which was obliged to release a security update.

Tesla attack

Maybe the most recent attack was conducted by two German cybersecurity researchers, who took control of a Tesla Model 3 using via Wi-Fi dongle connected to a drone in October 2020 take control of . They exploited a vulnerability in the function of the internet connection manager of the vehicle. They demonstrated that they could take control of the infotainment system and and could do anything possible by simply pressing the button on it, for example, locking and unlocking the doors, playing music, controlling the air conditioning[2]. Researchers informed Tesla about the outcome of the experiment, and the flaws were promptly fixed with a software update. However, they also affirmed that other OEMs might have the same vulnerability in their operating systems.

[1] https://www.dailymotion.com/video/x2yv2dl

[2] https://kunnamon.io/tbone/tbone-v1.0-redacted.pdf

Chapter 4

Background and related works

The objective of this chapter is to introduce the concepts used to develop the solution of my work, presenting technology used and discussing works and studies related to this Master Degree Thesis (Section 4.4).

4.1 Ontologies and libraries

4.1.1 Ontology

An ontology [22] (in its purely computer science sense), means a way to specify a domain of knowledge through an abstract model of it. This type of model is obtained by highlighting its properties and how they are related with each other. This process is done by defining concepts and categories subject-specific. The constitutional elements of an ontology are:

- *Classes or Concepts*, they are used to describe the principal entities of the application domain
- *Roles or Relationships*, they are used to describe the way classes and individuals can be related to each other.
- *Instances or Individuals*, they are the concrete representation of what is described in classes
- *Attributes*, An attribute of an individual can relate the individual to aspects or parts. TThese aspects or parts are typically represented by classes or instances.

However, the most innovative aspect of Ontology language is that starting from classes and their relationships there is the possibility to retrieve knowledge inferring new association automatically. This is possible using a *reasoner*, a component that runs parallel to the model definition and uses logic and semantical operations to deduce correlated concepts. As stated in Section 4.4 ontology allows to categorize things belonging to a particular domain, improving problem-solving; the major disadvantages deal with some limitation in property constructs and in the way OWL employs constraints, namely adding inconsistent data, which will not prevent from constraints, due to the philosophy besides the ontology that relies on a-priori principle.

4.1.2 Owlready2: a library for ontology manipulation

To manage Ontology concepts, individuals, and properties inside my model, Owlready2, a Python open-source library is used, developed by Lamy [23] for biomedical ontologies. It takes advantage of the point in common between ontologies and objects of an Object-Oriented Programming language and tries to connect them. The result is a *Ontology-Oriented Programming* language that dynamically translate at runtime the former into the latter. This approach is faster and more flexible compared to a statical generation of entities in code but has the drawback of avoiding type checking. The developer has the essential to make CRUD[1] operation on the model and construct a powerful application. As far as it concerns the architecture shown in Fig. 4.1, it relies on an optimized RDF quadstore, implemented a SQL Database stored in memory and disk, a *lazy parser*, for dynamically loading the entities from the quadstore on-demand and *automatic update* of RDF quadstore when a modification is done.

4.2 Communication Policy e Access Control

A communication policy states whether two entities inside a network can communicate or not. They are enforced in filtering devices and govern the access. It is possible to formalize it in many languages and using a different granularity. We can produce *High-level policies or Low-level policies*, the former is the result of a specification design phase and they describe rules that do not take into account physical details of the network upon which

[1]Create, Retrieve, Update and Delete

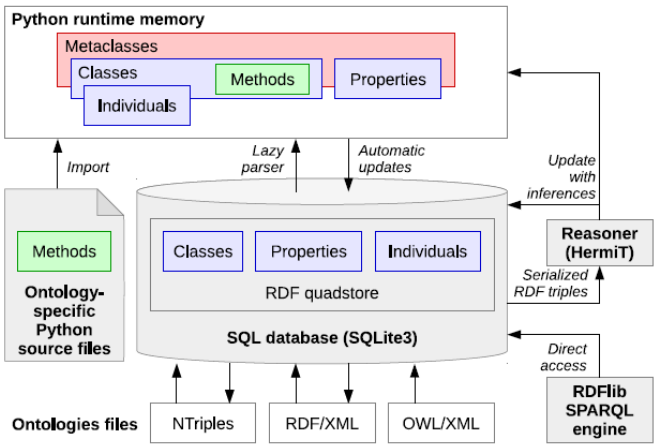


Figure 4.1: OwlReady2:Architecture

they are used, but they underline admitted communication and specific attributes and resources linked to it. The latter comes from the High-Level Policies declined on the real topology network. The *refinement* is a process that makes communication policies "more concrete and fitted" on the real interconnection.

4.2.1 High-Level policy and XACML

XACML [24]("eXtensible Access Control Markup Language") is a framework made of a declarative attribute-based access control policy language, an architecture, and a way to process access requests and evaluate the permission according to the rules, and eventually to their attributes. It is a standard and it was born to make uniform the terminology for access control policy. Regarding the language, it is an XML-based language and the schema is open source. The syntax is very powerful and easy to understand, providing the basic blocks to write policy [25]. The root element can be a single *Policy* or a group of them, *PolicySet*. Each policy can describe the set of requests to which it is applied, defining a *Target* and contains *Rules*, that is boolean expressions evaluated in isolation and then combined among them in order to extract the aggregate result. It is carried out using one of the various *Rule-combining Algorithms* like Deny overrides, Permit overrides, First applicable rule, etc.

4.3 Policy analysis in computer network

Packet filters contain an ordered set of rules and since the cardinality of this set is commonly very high, relations among rules bringing to misconfiguration are very difficult to discover. The most common issues are:

- Rules that are exactly matching, when each selector of R_1 is equal to the homologous selector of R_2
- Rules that inclusively matching, if each selector of R_1 is contained into a subset of the selector of R_2
- Rules correlated, if they are not *inclusively matching*, but their selectors are partially or completely overlapping
- Disjointness, when two rules are not *inclusively matching*, *exactly matching* or *correlated*. It can happen completely or partially, if there is at least a selector of R_1 that is equal, a subset or correlated to a selector of R_2.

For these reasons *policy analysis*, examining the rules inside one or more policies and makes an anomaly detection, is needed.

4.3.1 Anomaly Analysis

Al-Shaer et al. [26], in their works, define a possible taxonomy of policy anomalies, in particular, the first distinction in two macro groups considers if an anomaly occurs in the same filtering element (*Intra-policy anomalies*) or in different ones (*Inter-policy anomalies*).

Intra-policy anomalies

We have this kind of anomaly when there are at least two rules that are matched by a packet in the same filtering points. They can be of different tipologies [27]:

- Exception anomaly, it occurs when two rules R_1 and R_2, with priority of R_1 > priority of R_2, action of R_1 != action of R_2 and $R_2 \subset R_1$. If a packet matches R_1, then it will match R_2 too. In this case R_2 is more specific and therefore should be applied.

- Redundancy anomaly, it occurs when two rules R_1 and R_2, with priority of R_1 > priority of R_2, action of R_1 = action of R_2 and $R_2 \subset R_1$. In this case R_2 will match all the packets matched by R_1, so R_2 is redudant and can be removed.
- Duplication anomaly, it occurs if two rules generate a *Redundancy anomaly* and have an *exact matching* on each of their selectors.
- Irrelevance anomaly, it occurs regarding packets that can not exist on the devices under test.
- Shadowing anomaly, it occurs when two rules R_1 and R_2, with priority of R_1 > priority of R_2, action of R_1 != action. If R_1 matches every packet which will match R_2 too, we can state that R_1 shadows R_2.
- Correlation anomaly, it occurs when two rules R_1 and R_2, with action of R_1 != action of R_2. If a packet matches both of them, the outcome becomes ambiguous, and the resolution strategy has to determine which is the most specific among them.

Inter-policy anomalies

In a real network, where we have more than a domain and more than a filtering point, anomalies can occur between different policies, so an inter-policy anomaly analysis is needed. Also in this case we can find *Redundancy and Correlation anomaly* and other kinds of anomalies [27]:

- Shadowing anomaly, it assumes a different meaning in this context we discover it when a policy of an upstream device discards packets admitted by a downstream device.
- Spurious anomaly, it is the opposite case of the previous typology, that is a policy of an upstream device allows packets rejected by a downstream device.

4.3.2 Reachability analysis

Reachability analysis [28] in a distributed system context deals with the solution of the reachability problem, which is the process of computing the set of reachable states for a system. It consists of on-the-fly state graph construction and then, starting with the initial state, it explores the next states using a suitable graph-traversal strategy. The goal is retrieving which

global states can be reached by a distributed system made of a certain number of nodes that exchange messages between them.

4.4 Related Works

In this section I will discuss the related work and the extensive research literature on the matter.

Regarding the first aspect, namely the formalization of a reference model for the classification of topological elements contained into an Automotive architecture, it starts from the study of the work of various research groups that devoted their energy to the formalization of ontology for the automotive industry during the years. In particular Klotz et al. [29] approach the problem underlining the importance of a common semantics for studying ECU signals and messages, point out different formats and reference architectures to facilitate the development of application and connectivity of modern automotive standards. These concepts are shared with Feld and Müller [30] that develop an ontology oriented to the design of complex HMI[2], linking therefore the user's action to the vehicle context, producing personalization in Graphical interface and a better user experience. Maier, Schnurr, and Sure-Vetter [31] instead, demonstrate the high expressiveness of the ontology regarding the organization of vehicle data and how easy the integration without changing into the existing IT environment can be. Finally, a work, with a more practical approach is [32]. This paper deals with a description of automotive diagnostic, emphasizing however the aspect linked to the troubleshooting process, describing possible issues and their solution, in order to make easier the development of an automatic diagnostic tool.

The second aspect related to the access control model using communication policies, trying to supply to the necessity of stronger security controls expressed in stardards [15] and [16], follows the work of Rumez et al. [33]. In fact, they use ABAC policy in a secure gateway, underlining the need for well-organized protection against cyberattacks, defining a model of Authorization based on attributes typical of automotive infrastructure. My work follows this path but exploits the model developed in the first part and makes further analysis to give robustness to the rules enforced into filtering devices, avoiding misconfiguration.

[2]Human Machine Interaction

Chapter 5

Problem Statement

The objective of the chapter is to define the problems that this thesis tries to solve, by describing the main requirements. There are three main aspects to take into account:

1. Defining a language to describe the network topology of an automotive architecture
2. Defining an Authorization language for allowed communication inside the vehicle network
3. Reducing error in the configuration of filtering elements

Each one of them can be seen as an independent task to be solved properly, but at the same time, they are tightly correlated, thus the output of a task becomes the input of the following one.

5.1 Requirements

The first problem is directly linked to the certification process, in fact, in order to be compliant with *ISO 21434* [15] (sections 8: Risk assessment methods and 9: Concept Phase) and *UNECE WP.29 / R155* [16], OEMs have to produce an Item definition and an Asset identification. In these documents, they have to identify architectural elements and functions that are cybersecurity relevant. The distinction between internal components and the outside environment shall be included in the description language since the interaction between the vehicle and the external world shall be taken into account in this analysis. Item definition is followed by an Asset Identification, so a classification of assets and a way to mark structural elements with them shall

be done. A formal description of possible assets and structural elements can ease this task and pave the way to the development of new tools designed to make automatic the certification process.
Figure 5.1 shows the documents to produce to be compliant with *ISO 21434*, those to which this thesis refers are in the red dotted box.

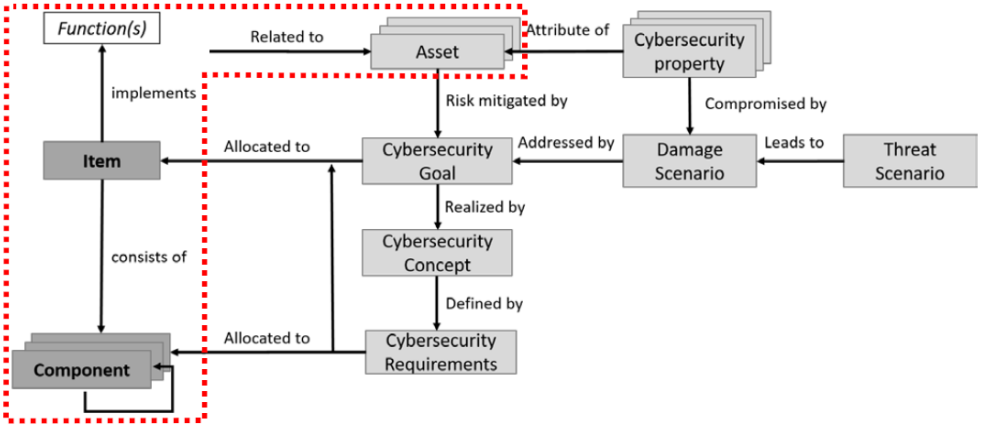


Figure 5.1: ISO 21434: Requirement generation for cybersecurity relevant items or components

As far as it concerns the second and the third problems, they catch the challenge thrown by the open problem in the automotive world. Vehicles require a correct implementation of protocols and the introduction of more security controls. So I try to give a partial solution to this task by implementing access control for messages exchanged inside the vehicle, defining Authorization policies for the Security Gateway. It shall implement filtering functions since it has the same role as the normal firewall.

Finally, the last problem is linked to the configuration of the filtering point: in this step, misconfigurations shall be avoided. In this phase, the solutions produced by the previous task become the input of this one. Low-level policies and the network topology shall be used to perform analysis useful to reduce the wrong configuration due to human error or wrong assumptions in the design phase.

5.2 An overview of the proposed solution

The proposed solution, which meets these requirements, relies on the following choices:

1. **Analyses of Standards and Regulamentations**, studying deeply these documents inspired the topic to develop in this final work, helping mainly in the elaboration of the first part of the solution, namely in the Topology and Asset Definition.

2. **Define an Ontology to describe automotive scenario**, after the study of much automotive architecture, the right choice to make deals with a suitable language that could describe properly the scenario. In the end, we decide to explore an Ontology because of its power to describe reality in a precise and expressive way

3. **Use XACML as Authorization Language to define High-Level Policy**, to design policies using positive authorization specification, so that only admitted communication is allowed, denied all the others

4. **Define a Refinement algorithm to traduce the High-Level Policy into Low Level to be used to configure devices**, this choice arises from the consideration that very often Design phase shall be distinguished from Device configuration, since these two phases are done by different people, with different skills, so an algorithm that could allow this differentiation shall be done.

5. **Perform Anomaly analysis and reachability analysis on Low-Level Policy**, a library developed by **TORSEC** group is used in this phase. It was born to analyze computer networks, but it was designed to be agnostic, so with some customization, it has proved a very useful tool to solve this part of the problem

All of these points shall be discussed more in detail in the following section.

Chapter 6

Design of the solution

In this chapter I will describe in detail the design choice made to solve the problem stated in the previous chapter, describing all the workflow followed and all the activities performed.

6.1 Model definition

The formalization of a description language for automotive topology is the first outcome expected by this final work because it is the basic step upon which the following steps rely on. Before the formalization, many architectures have been analyzed, because there is not a real standard since every OEM has its own network topology. It has been very challenging research because they are protected by an industrial secret, so they are not easy to retrieve, anyway opensource projects[1] and some constructor whitepapers helped in this projectual phase. What I have obtained from this analysis is a possible generalization of components related to automotive so different interfaces, ECUs, and protocols. This taxonomy is been validated by a company involved in automotive cybersecurity. They gave their feedback about the concepts followed by a fine-tuning phase in which ontology has been improved.

After this period of data collection, I studied many languages used to model data, in particular, I have considered UML, XML, and Ontology. All of them are suitable to describe reality and are fully compatible with the

[1] https://github.com/commaai/opendbc

majority of programming languages, so they are directly usable for tool development. However, they have some peculiarities which have tended towards the choice of Ontology rather than the others. UML[2] is mainly used for communicating ideas to developers, using diagram for software design, instead, XML[3] is a language for storing and exchanging data in software, like passing messages between applications. So no one of them can be used to have a real formal taxonomy of components inside the Automotive architecture, in fact, Ontologies try to remove, or at least reduce, conceptual and terminological confusion in order to have a shared interpretation.

Regarding the formal language used to construct the Ontology, OWL has been chosen. It is a declarative language, so the ontology is specified without providing an algorithm that describes how the ontology should be constructed or how the reasoning should happen. OWL is built on top of the RDF[4] and RDF Schema. RDF provides a way to express statements about resources in the form of subject-predicate-object expressions (triples). RDF Schema provides mechanisms to describe RDF resources and the relationships between them in terms of classes, properties, and values. OWL is a successor of RDF Schema; it is a stronger language and has greater machine interpretability.

The main decisions taken to describe the application domain are shown below. The design of the core components of ontology represents the critical point of this phase. In fact, the main goal of this activity is to produce a highly expressive model that can fit in the best way on the real domain. We have followed a bottom-up approach, we have analyzed every single component and then we have discussed and modeled their relationships.

The first, and the most crucial, task was how to model the single node, that is to say the different types of hosts that can be present inside an automotive architecture. The first consideration we made was to include two kinds of elements: *Network* and *PhysicalNode*, the former represents the logical area or domain of which the vehicle is made. The latter, instead, depicted in Figure 6.1 is the class containing the set of components which are the active parts of the vehicle, that runs applications and does the principal functions. We can observe that some general-purpose devices like hub or switch belongs to this *Concepts*, which are used in networking, especially domain-specific units like Domain Controller, Gateway, and ECU. For this last typology,

[2]Unified Modeling Language

[3]eXtensible markup language

[4]Resource Description Framework

further classification is necessary in order to distinguish the safety-critical ECUs from the non-safety-critical ones.

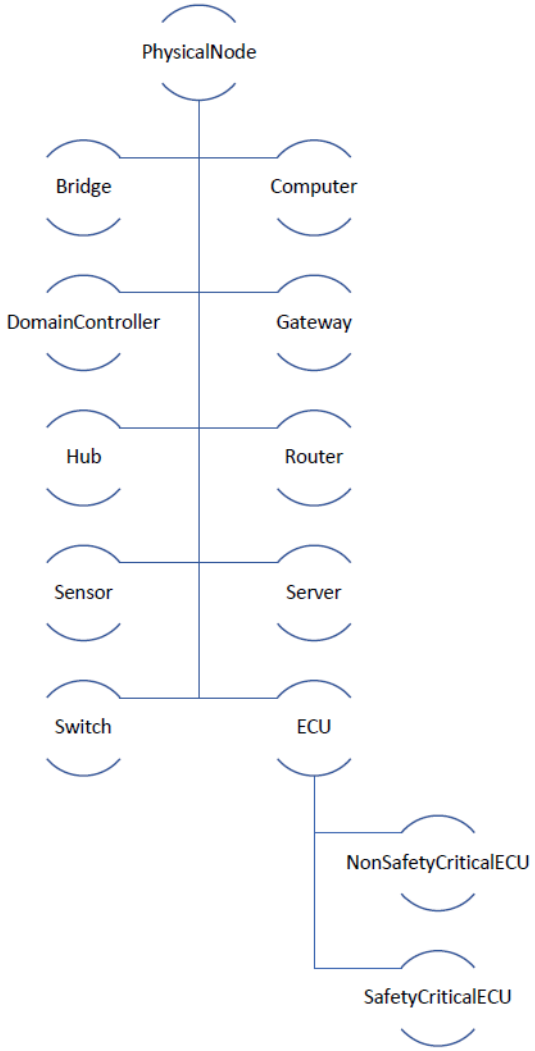


Figure 6.1: Automotive Ontology: Node

Since the ECUs and the other devices are interconnected, the taxonomy of different interfaces is needed to better qualify them and to create a very expressive relationship between components. Figure 6.2 and Figure 6.3 shows the "branch" of the ontology that describes all the possible communication interfaces that are available on the automotive component. The principle distinction to make is between *CommunicationInterface* and *NetworkInterface*, the former is used to transfer information inside/outside the vehicle, like the interface on the Infotainment system(USB, NFC, etc) or the information

about the position that comes from the navigator; the latter deals instead with the network, since they are more domain-specific (e.g., CAN, MOST, etc.). Behind these two macro-groups, we can find a further division in the category, in particular between *PhysicalInterface* and *WirelessInterfaces*.

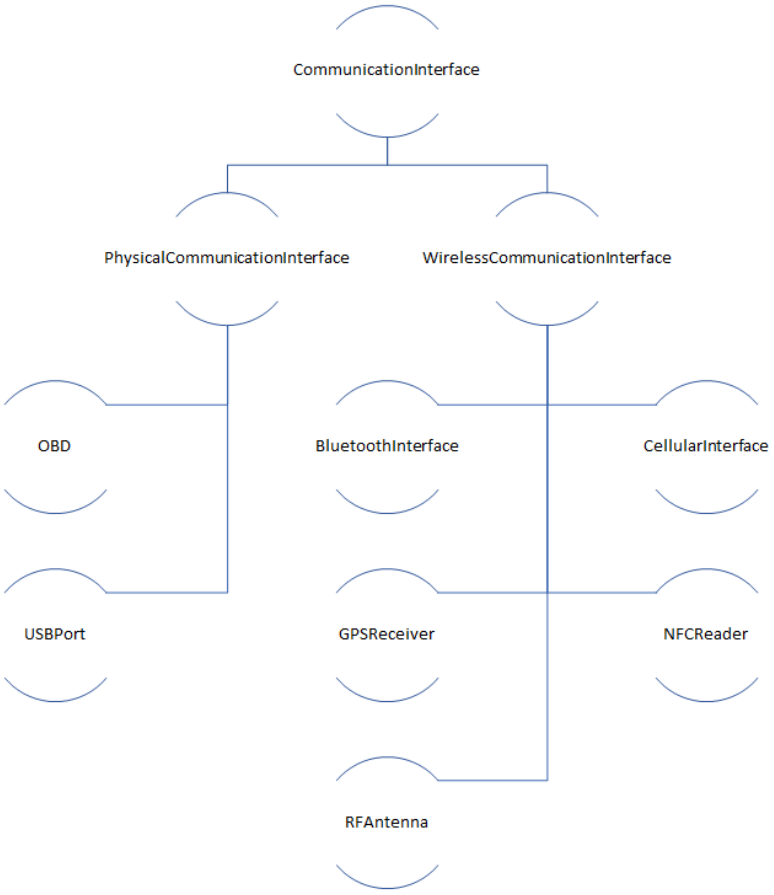


Figure 6.2: Automotive Ontology: Communication Interface

As regards to the Automotive infrastructure, another specific aspect to model deals with the message exchanged by the internal/external components. In Figure 6.4 the categorization of these concepts is depicted, it descends from the decision taken from the definition of the previous components. *Messages* are in fact the information that flows through the interfaces from a network domain to another, and it will be the core aspect of the following design steps regarding the Authorization policy(Section 6.3 and Section 6.2).

The most important aspect to consider from a cybersecurity point of view deals with an asset definition. The design of *Asset* class descends to the input

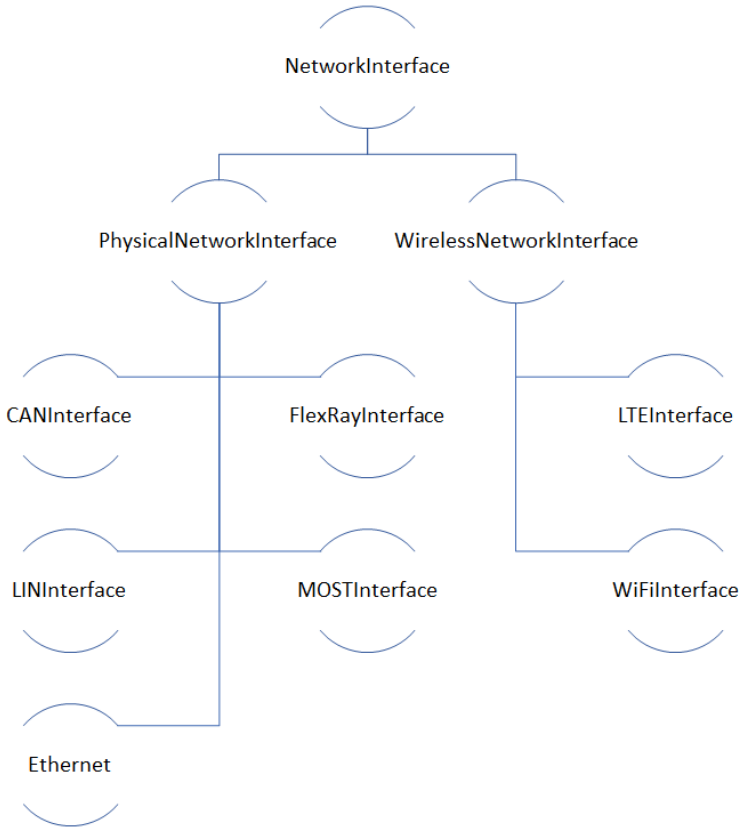


Figure 6.3: Automotive Ontology: Network Interface

given by *UNECE WP.29/R155*, where a basic taxonomy of components and function is done. Figure 6.5 describes the results of the normative analysis, we have focused on the main characteristic of automotive components that can have an economical value or can be significant for safety reason. So the main subclasses that we have analyzed are *Code*, *Function* and *Data*.

6.2 High-level policy design

The study of different automotive architecture also helped in this phase, in fact, the design activity that characterizes the formulation of High-Level Policies starts from an analysis of components inside the network topologies and messages they exchange. To analyze them, DBC files were studied. They are text files that contain information for decoding raw CAN bus data, converting signals into human-readable messages, each network domain has its own DBC files, so with them, the architecture and data flow can be

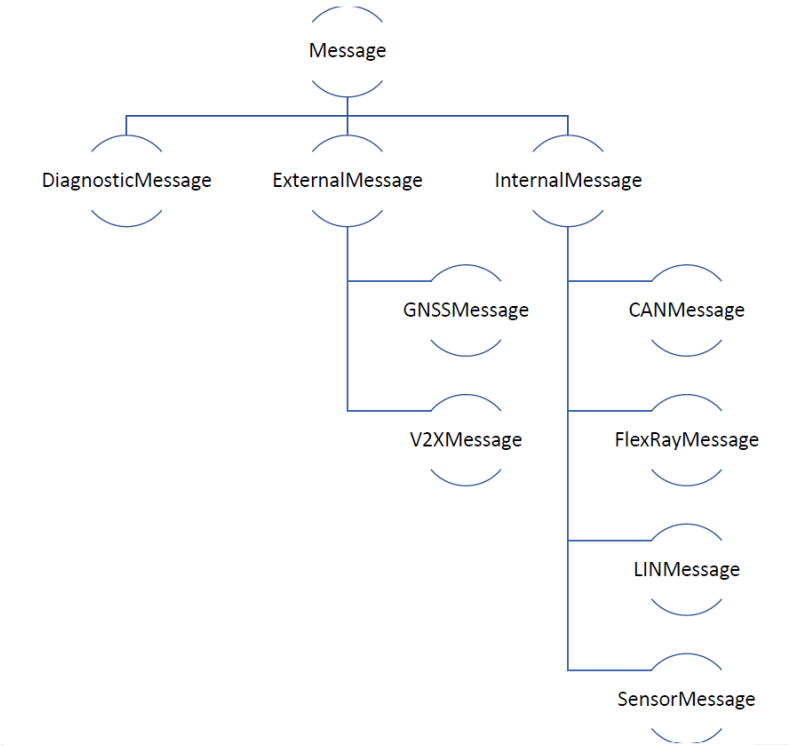


Figure 6.4: Automotive Ontology: Message

retrieved.

So at this point, we have to decide how to design the policy and how we can link them to the description language, so the main hypothesis was:

1. Writing policy as an element inside the ontology, so formalizing them as if they were entity

2. Using a specific language for policy definition, including information about the structural element

Both of them solve the problem, but they bring a slightly different result. The design of the first solution involves the creation of policy directly inside the OWL ontology, modeling subject and resource with Class or Individuals, and applying action-creating Relationships. The evaluation of authorization request is in charge of the Reasoner that analyzing the formalized statement can determine whether a request is legit or not. The second possibility deals with the use of another paradigm so that the policy has to be written in a specific Authorization language, that relies on a data structure to be application-specific. Comparing the two solutions we observed that the

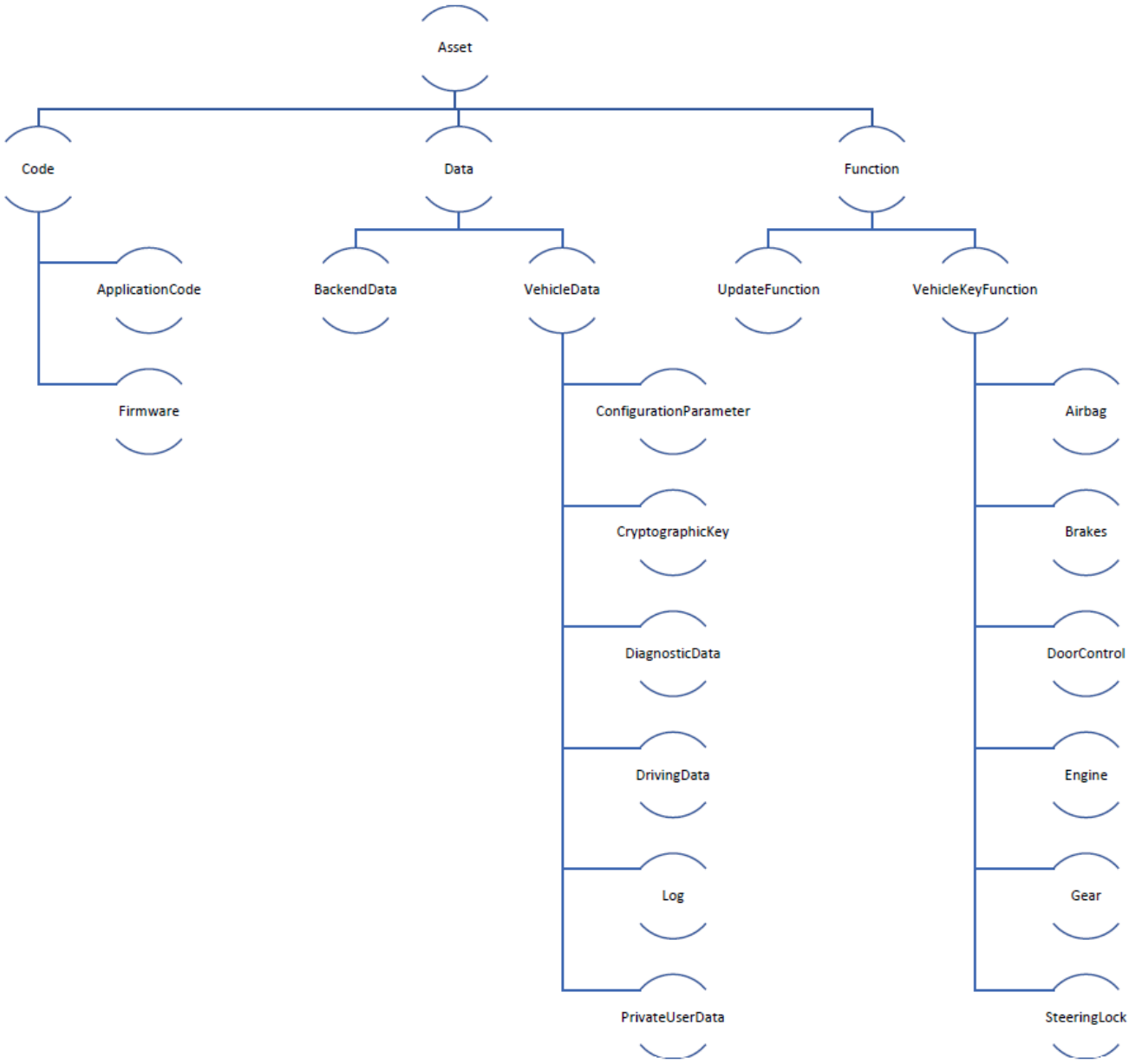


Figure 6.5: Automotive Ontology: Asset

former is more efficient because inherits the strength of ontology, resulting however less scalable for the same reason. Regarding implementation, the second is more straightforward because the specific language has a framework useful to implement very powerful policies and it is extensible, including inside them an XML schema that describes the domain of application. For these reasons, the latter was chosen.

As I described in Chapter 2, XACML includes many resolution algorithms and allows the possibility to insert not only policy definition but also restrict the applicability of a policy defining the characteristics of Request. Before starting to design policy we have converted the Ontology into an XML schema, in this way we can obtain a more compact representation, with some minor loss linked to the ontology feature, but not indispensable for policy development. This schema has been included directly inside the XACML model schema[5], in this way a policy can be validated against XACML schema and Ontology schema.

In Appendix A I report the code of the template used to design policy, it considers the most general situation for the automotive context and can be customized to adapt to different needs. Now a description of the main element of that template will follow. The approach used is based on the description of a policy for each ECU and a rule for each message exchanged with another entity. XACML allows to specify *Subject-Action-Resource* and of optional *Attributes*, which qualify in a better way the policy or eventually the rule itself. In the design of the template, both strategies have been used, so that it can be very specific. In particular, in the first part, I have designed the target of the entire policy, then the target of each rule.

The root element XACML is `<Policy>`, that contains namespace linked with the policy itself and the *RuleCombiningAlgId*, that in our scenario is *Deny Overrides*. Then the `<Target>` to which the policy refers has been specified by means of the triad Subject-Action-Resource, in particular a list of elements for each of them. In our scenario, the subject is the ECU under consideration, so in the `<Subjects>` are specified two different `<Subject>` representing its features. The first one specifies the *subject-id*, a unique identifier for the component and the other is *subject-type* that represents the class defined into the ontology designed for the ECU. The resource instead addresses the destination ECUs, that will receive the messages sent by the subject, so in the `<Resources>` are specified two different `<Resource>` definitions. Since the source ECU can have multiple distinct destination ECUs, here we have just set the basic requirement for them, so it has been defined the resource type by means its structure inside the ontology `xpath-node-equal` and the relative position inside the XACML request `resource:xpath`. In this case the *xpath* notation has been used to refer to *node* inside another *XML schema*. Finally, a custom Action has been chosen: *Authorize*, to specify that the

[5]http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd

communication between Subject-resource is allowed.

The most specific part of the policy, regards the different `<Rules>`, each policy can contain an arbitrary number of them, but each `<Rule>` is defined always by means two attributes: the name of the Destination ECU, defined by means of `xpath-node-equal` inside the request, and the `resource:message-id` that contains the unique identifier of the CAN message.

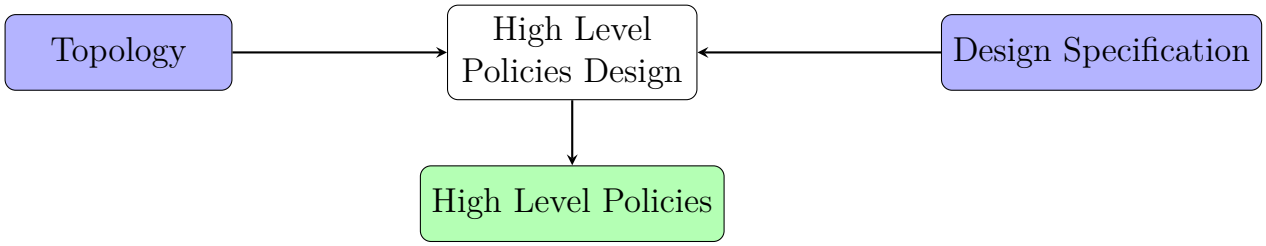


Figure 6.6: Policy Design workflow

6.3 Policy refinement

Policy refinement is the transformation process that converts a more abstract high level policy into a low-level, operational policy that a system can enforce. The goal of this operation is to determine the resources that are needed to satisfy the requirements of the policy, to analyze the underlying system upon which the policies should be enforced, to extract a graph and compute the solution. Verification of the correctness of the process is needed to validate that the lower-level policies are compliant with the design specification of high-level policy. After this process, a policy can be enforced on filtering devices.

The workflow of this design phase is depicted in Figure 6.7, all the choices taken are described in this chapter instead the implementative details are in the following chapter (Chapter 7). This algorithm was designed to receive as INPUT:

1. High-Level Policies, they are the result of the previous design phase and they contain Subject-Action-Resource of the policy

2. the landscape, it is the representation of Individuals described through the ontology

3. DBC files, one for each subnet inside the automotive network. They

contain the list of nodes inside a particular subnet and the list of messages that can be received or sent by them. Each message is described by a tuple with:*messageId messageName sourceNode destinationNode*. For messages from/to different network source/destination nodes assume a fictitious value.

The algorithm takes this INPUT and, as it is described in the pseudocode Algorithm 1, it transforms the landscape in a graph, then parses DBC files and High-Level Policies. It elaborates these data structures and extracts the list of the path for each couple *(source, destination)*, at this point every single path is analyzed and if it contains one or more filtering elements, a rule is created and inserted in a configuration file. At the end of the process, each one of them has a configuration file, containing the *Low-level policy of the specific filtering element*, composed by a set of rules. Each rule contains the admitted message, the source network, the destination network, and the destination node.

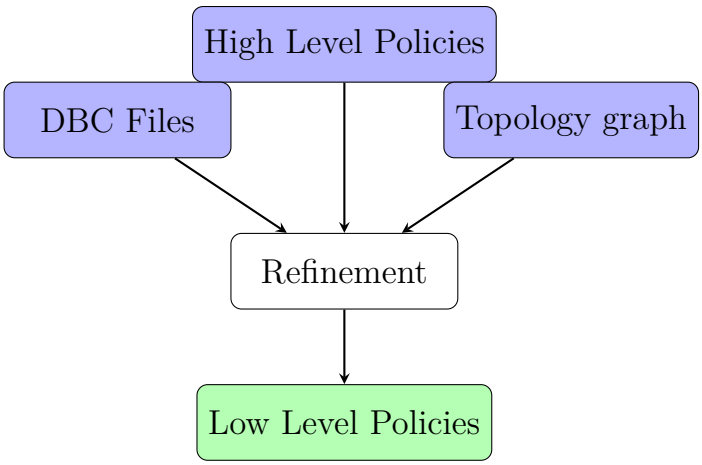


Figure 6.7: Policy Refinement workflow

6.4 Policy analysis

6.4.1 Anomaly analysis

In order to perform this kind of analysis the *"Policy tool library"*, developed by *TORSEC* group, is used. It is integrated to detect all the inconsistencies and rule anomalies. The security problems faced by the design are intra-policy anomalies, which means there are misconfigurations in the same filtering

Algorithm 1 Refinement algorithm pseudocode

```
1: Input: O                                                    ▷ Ontology
2: Input: DBC                                                  ▷ List of DBC files
3: Input: HP                                                   ▷ High-Level Policies files
4:
5: Output: Configuration                                       ▷ FilteringElement→ [rule]
6:
7: G=generateGraph(O)
8: FilteringElements=[ ]
9: for all e instances in O do
10:     if e = 'Gateway' then
11:       FilteringElements.append(e)
12:     end if
13: end for
14:
15: MessageSourceDest=DBCParse(DBC)    ▷ msg → [source], [destination]
16: HLp=XACMLParse(HP)                              ▷ source → [destination]
17:
18: for all source,destination in HLp do
19:     PathList=g.getShortestPaths(source, dest)
20:     for all path in PathList do
21:         if path contains a FilteringElement then
22:             r=createRule(msg, source, destination)
23:             Configuration.put(FilteringElement, r)
24:         end if
25:     end for
26: end for
```

point and inter-policy anomalies that represent a more complex issue because conflicting rules can be on different filtering devices.

Figure 6.8 and Figure 6.9 show the input and the output of the anomaly analysis of security policies. the input are:

- Low-level policy
- Landscape

The Low-level communication policies are defined to be fully compatible with the schema provided by the tool in order to reuse the library's modules. Moreover, XSD language was chosen instead of DTD one, since it is more expressive and it is possible to define and establish more specific constraints and references. In particular, all the policies are nested in a more general node named EntityElement, which could express further information. In my case, it is characterized by:

- *Label*: it is the uniquely identifier of the entity among all the entities;
- *PolicyElement*: it is the entity's policy.

```
<complexType name="EntityElement">
        <sequence>
                <element name="Policy" type="PolicyElement" />
        </sequence>
        <attribute name="Label" type="string" use="required" />
</complexType>
```

Each policy is modeled as a "PolicyElement", it expresses the security behaviour that must be followed by all its instances. It is characterized by:

- *PolicyName*: it is the name of the policy;
- *PolicyType*: it is the type of the policy, in this thesis work the only useful policy type is PolicyTypeElement = FILTERING, but this feature is kept for compatibility and for additional future work;
- *DefAction*: it is the default action performed by this policy;
- *Rule*: it is the ordered filtering rule list.

```
<complexType name="PolicyElement">
        <sequence>
                <element name="PolicyName" type="string" />
```

```
4               <element name="PolicyType" type="PolicyTypeElement" />
5               <element name="DefAction" type="string" />
6               <element name="Rule" type="RuleElement" minOccurs="0"
                    maxOccurs="unbounded" />
7       </sequence>
8 </complexType>
```

The "rule" item is the last one modelled, it has the task to describe what kind of traffic pattern it needs to be matched by this filtering rules and what action has to be performed on it. It is defined as:

- *Priority*: it sets the rule priority in the policy;
- *Selector list*: it is the list of selectors, that is to say the fields in a packet that need to be checked in order to be matched with the rules;
- *Action*: it represents the action to be performed on the matched traffic;
- *Label*: it is the unique identifier of the rule.

```
1 <xsd:complexType name="RuleElement">
2       <xsd:sequence>
3               <xsd:element name="Priority" type="PriorityElement" />
4               <xsd:element name="Selector" type="SelectorElement"
                    minOccurs="1" maxOccurs="unbounded" />
5       </xsd:sequence>
6       <xsd:attribute name="Action" use="required">
7               <xsd:simpleType>
8                       <xsd:restriction base="xsd:string">
9                               <xsd:enumeration value="Authorize" />
10                              <xsd:enumeration value="DENY" />
11                      </xsd:restriction>
12              </xsd:simpleType>
13      </xsd:attribute>
14      <xsd:attribute name="Label" type="xsd:string" />
15 </xsd:complexType>
```

As regards to the landscape, it is described through another XML schema and it is characterized by:

```
1 <element name="Landscape">
2               <complexType>
3                       <sequence>
4                               <element name="Firewall" type="
                                    FirewallElement" minOccurs="1" maxOccurs=
                                    "unbounded"/>
```

```
5                               <element name="FilteringZone" type="
                                    FilteringZoneElement" minOccurs="1"
                                    maxOccurs="unbounded"/>
6                               <element name="Host" type="HostElement"
                                    minOccurs="1" maxOccurs="unbounded">
7                               <element name="Link" type="LinkElement"
                                    minOccurs="0" maxOccurs="unbounded"/>
8                       </sequence>
9               </complexType>
10 </element>
```

- *Firewall*: it represents the filtering element where the policy will be enforced, in our case they are the security gateway;
- *FilteringZone*: it represents a specific network domain;
- *Host*: it represents the different active component of the automotive architecture, mainly ECUs;
- *Link*: it models the connection between *Hosts* and *Firewall*.

The Policy Tool Library has been designed to take these INPUTS and produce as OUTPUT the list of single or distributed anomalies. These analyses come from the representation of policies in a format that guarantees policy semantics, but it abstractly transforms them, so that they can be easily managed by the tool. It is the concept of *Semantics-Preserving Policy Morphism*, this translation of policy representation allows analyzing anomalies independently from the resolution chosen by the administrator. From a design point of view, the steps to follow are:

1. translating the policy in a generic intermediate representation called *Canonical Form*;
2. translating the canonical form into any target resolution strategy.

Namely, the*Canonical Form* is a policy representation, useful for policy manipulation and based on the operation set to solve conflicts. It is formally equivalent to the original one, but it has more rules, this permits easier processing because all the possible combinations of conflict cases are precomputed with the resolution strategy. Now all the algorithms that work on the Canonical Form of a policy do not have to consider the original, because all the differences between the resolution strategies are removed.

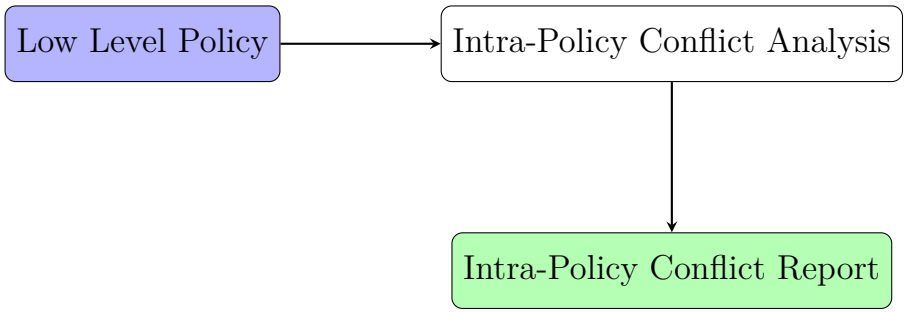


Figure 6.8: Intra-Policy Anomalies workflow

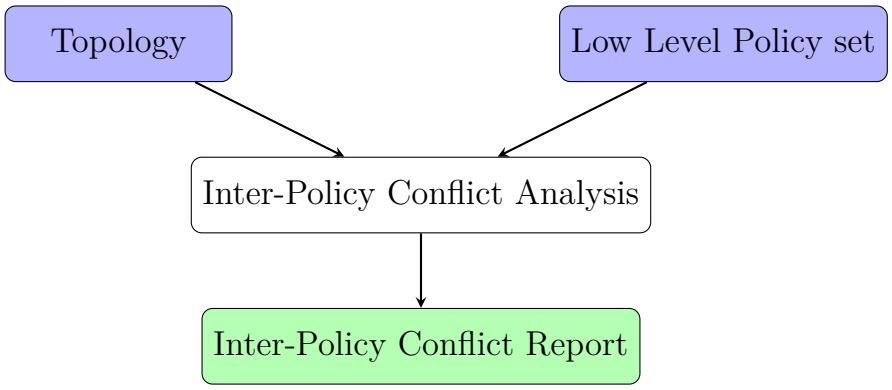


Figure 6.9: Inter-Policy Anomalies workflow

6.4.2 Reachability analysis

The library used in the previous step is also suitable to perform *Reachability analysis* (Figure 6.10), which allows elaborating all the paths or a subset inside the landscape to verify whether two nodes are reachable or not. The functions in charge of these work take the same input of reachability analysis and produce in output an *Equivalent firewall*, which is an abstract representation of a firewall that aggregates in a single point all the policies of several firewall making some optimizations, where possible. In this way a complex network can be simplified, particularly the path from a domain to another can be summarized using these logical components, making easier the discovery of admitted communication between entities belonging to the domain under consideration. After this representation is produced, there is the possibility to check its correctness by running anomaly analyses over it.

In order to do this all the firewall that are contained in the different analyzed zone have to be collected. After that all the policies contained have to be extracted and transformed. The result of this step is a data structure

that represents the network analyzed and contains all the reconcilied policy.

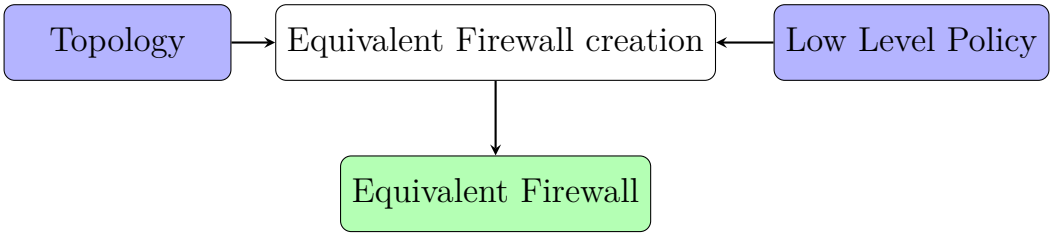


Figure 6.10: Equivalent Firewall creation workflow

Chapter 7

Implementation

This chapter aims to show the real implementation of the algorithm presented in the previous section, describing the technology used and all the choices that have characterized this step.

7.1 Refinement algorithm

The implementation of the refinement algorithm will follow the pseudocode described in the previous chapter (the code is available in Appendix B). The language used is *Python*, because of the high expressivity and the availability of libraries. Our scenario is quite heterogeneous since it is made of different kinds of input files (e.g., ontology, DBC, and XACML files) and it has to handle graphs.

The main libraries used are:

- *Owlready2*, to manage the ontology in OWL format. As I have described in Chapter 4, it is very efficient also with a very huge ontology and it suits perfectly in our context. It offers the basic functions to transform the ontology in a python object with all the necessary member methods, in this way the transformation in a graph becomes easier.

- *cantools*, it is a specific python library used to handle CAN messages, and it allows to extract them from file and elaborate directly in the code.

- *xmlschema*, it is a general-purpose library used to extract all the elements contained in an XML file from the root element to every sub-node. I have decided to include it in my implementation since XACML policies are based on XML.

- *igraph*, it is one of the most used library for graphs handling. It allows to construct a graph object and then to apply all the basic operations and algorithms used to retrieve paths to it or to make other kinds of computations.

So after a setup phase, consisting in parsing input, the main operations described in the previous chapter have been implemented step by step. Thus a graph representing the topology is extracted from the ontology and all the chains from destination to source have been extracted using the method to extract *the shortest path*. The time complexity of this operation is $O(|V|+|E|)$, so it is linear in the number of vertices and edges in the graph. Once I have retrieved all the paths for each couple (source, destination), the discovery of filtering elements inside them has been conducted. Finally, the configuration file of every single element has been computed and written into the proper configuration. The format used in our implementation reflects the model used by many Security Gateway manufacturers, so a policy for each of them has been created. They contain the list of rules with all the fields described in Section 6.4.1. Since in our scenario the rules are not correlated, because they regard distinct messages, each one of them has the priority set to the same value.

7.2 Anomaly analysis

In this section, I have collected all the Data structures and the methods of the library involved in this analysis. As already introduced in the previous Chapter 6, policy tool library has been used in my project. It has been developed by *"TORSEC"* security group in Politecnico di Torino for the past EC-funded project *SECURED*. Originally it has been used to handle and manage security policies, to resolve rules conflicts, and to analyze reachability properties in a given landscape with servers, hosts, and firewalls, but with proper changes and adaptations, it can be suitable also for other computer network scenarios like the automotive one.

The core of the library is the module that implements what Section 6.4.1 explains, which is to say it bases its policy representation through the *Canonical Form*. The library also provides all the tools to generate a *Semilattice* representation and the possibility to translate a given policy into a *Morphism form*. Different modules of the library were exploited to solve anomalies and conflicts of the input policies.

- *Intra-policy conflict analysis*: each policy is represented by the Java object *PolicyImpl* that offers a "resolution strategy", a "default action" and a list of "filtering rules". Starting from this class, the correspondent *closure* is generated, and from those results, the canonical form is created to have easy processing of the policy itself. After this operation the class *SemiLatticeGenerator* allows the library to analyze the policy and to discover all the rule anomalies which are solved by the *FMRMorphism* object. It generates a morphism representation of the policy from which a new policy could be defined with no conflicts.

- *Inter-policy conflict analysis*: this process is called "reconciliation" because it is done between policies. The first operation is to create a *ComposedPolicy* that contains an ordered list of the policies to be reconciled. After this step, the procedure to find all the anomalies and to remove the conflicts is the same as the "intra-policy conflict analysis" described in the previous point.

To apply the library methods to our scenario, *SelectorType*, a Java Class that represents the field that characterizes each rule (e.g., Source/Destination ECU, Message-ID, etc.) has been modified. For each one of them the proper data type shall be defined and all the set operations useful to combine them.

7.3 Reachability analysis

As regards to the reachability analysis described in Section 6.4.2, the library offers a method that allowing to retrieve a *ComposedPolicy*, and collecting all the policies contained in the *Filtering points* that belongs to two different Networks. A composition between rules is generated when two rules overlap, it is made by the condition clause intersection of the two rules and the action resulting by the application of the resolution strategy. Given this policy, it is possible to define its closure as the set of all the possible composition of the policy rules. In this way, it is possible to summarize them and to use them as they were a single firewall. These operations simplify the analysis of anomaly and it is used in the testing phase in order to verify whether a path between two nodes exists or not. A proper method was implemented to perform these operations. It takes the SourceNode and the Destination node as input and it retrieves from them the network they belong to. So the method *getEquivalentFW* is called and the path is verified. This method returns TRUE or FALSE, respectively if the path is feasible or not.

Chapter 8

Evaluation

The objective of this chapter is to exploit solutions defined in chapter 6 and 7, thus implementing a Proof-of-Concept that aims to adapt to the use case suggested in Chapter 5.

8.1 Proof-of-Concept

As the last task of the final work I have developed a tool that allows:

- To perform item definition and asset identification, using the description language;
- To compute the configuration of the security gateways, if they are described in the item definition.

It exploits the ontology and allows to draw the automotive architecture under test by choosing the needed elements from a catalog containing all the available items. At this point, it is possible to connect the different components with the proper network interfaces (e.g., CAN, FlexRay, etc.) or communication interfaces (e.g., USB, Bluetooth, NFC, etc.). So it is possible to retrieve a graph representation of the architecture, in order to have a complete visual of it and perform the asset identification as required by the standard.

In the same window it is then possible to upload DBC files related to the networks described and the High-Level policies specifically designed in order to apply the refinement algorithm. In this way, the proper configuration file is computed for each filtering element retrieved in the topology and associated with the security gateway it belongs to.

It is possible from the tool to export the entire topology or the configuration files, in order to use them as input for the Policy analysis developed with the *PolicyToolLibrary*.

8.1.1 Architecture

The tool is implemented as a Web Application, thus the development is results quick, powerful, and platform-independent. The development stack used is Django+Angular+MySQL, so:

- *Backend*, it has been done in Python, exploiting the Django framework. In order to manage the ontology, the *owlready2* has been used as internal components
- *Database*, the platform data are stored in a MySQL database, exploiting an ORM[1] to perform the association between the backend and the DB.
- *Frontend*, it has been used the latest version of Angular framework, because of its flexibility and for the complete and clear documentation.

In Figure 8.1 is depicted the software architecture of the tool.

8.1.2 Design

Once the goal of this application has become clear, the main functional requirements and the database design have been defined.

Functional requirements

The application shall show all the existing classes of items and present them as they were in a catalog.

For each class, it shall be possible to create new individuals representing the Node of the architecture under test.

Each one of them can be defined through a name and a set of interfaces and shall be possible to update it or delete it.

The list of all created items shall be shown on the main page of the application.

[1]Object–relational mapping

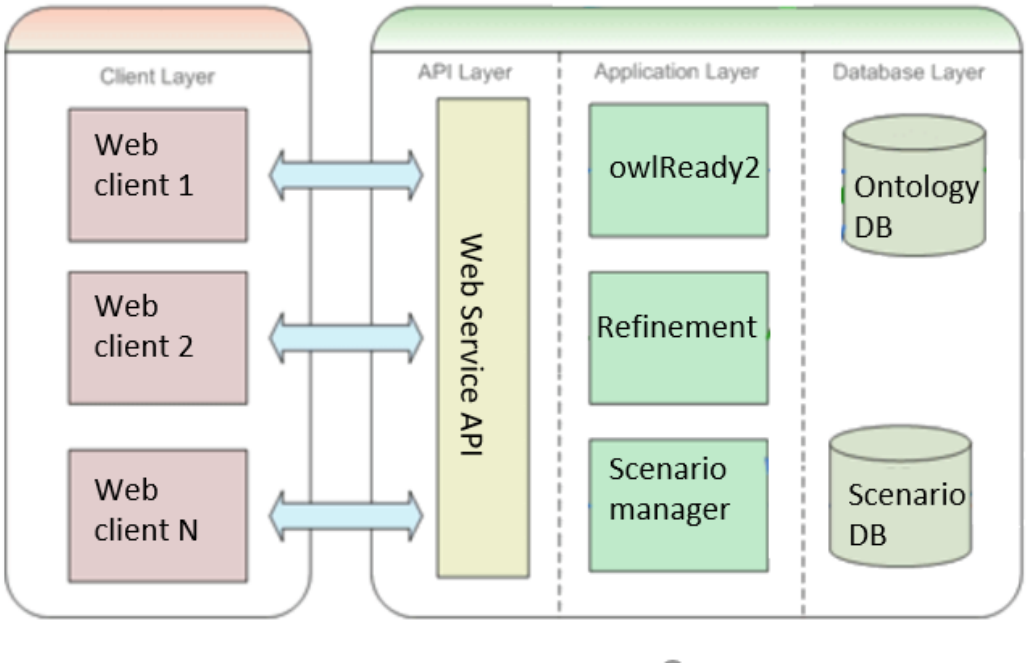


Figure 8.1: Tool for Item definition and Asset Identification: Architecture

For each couple of individuals shall be possible to create a channel in order to define the connection among them.

Each one of them can be defined through a name, the two endpoints, and the type of connection and shall be possible to update it or delete it.

The list of all created channels shall be shown on the main page of the application.

It shall be possible to extract a graph structure from the ontology representation that represents the architecture under test.

The application shall show all the existing assets and present them as they were in a catalog.

For each class of assets, it shall be possible to create new individuals representing assets and use them to mark an element of the architecture under test.

Each one of them can be defined through a name and shall be possible to update it or delete it.

The list of all created assets shall be shown on the main page of the application.

It shall be possible to create different scenarios, in order to describe multiple architectures separately.

Each one of them can be defined by a name and they can be listed, modified, or deleted.

The deletion of a scenario shall involve the deletion of all the elements and assets that it contains.

It shall be possible to upload the High-Level policy and the DBC files for each scenario, thus performing the refinement of High-level policy in order to retrieve the configuration of the filtering points.

It shall be possible to export a file containing the landscape.

It shall be possible to export a file containing the configuration of the single filtering points.

All of these requirements have been implemented through REST endpoints inside the Django backend. The entire list of all endpoints is available in Appendix C.

Database design

As regard to the database design, the Web application uses two different data sources (Figure 8.2):

1. *Platform dependent database*
2. *Ontology database*

This choice comes from the requirements definition. The former database is used by the backend to store information about the different Scenarios and to associate to each one of them the proper elements. The latter is managed by *owlready2* independently by the former and it is completely transparent to the developer that interacts with it directly through the API provided by the library. As regard to the *Platform dependent database*, it has two Tables *Scenario* and *ScenarioElement*. The former contains the information related to every single Scenario, the latter on the other hand contains the association between the ontology element and the Scenario they belong to. The Ontology database instead has three tables *Resource, PropFts, Objs*. They are mantained in order to make more efficient the internal operations. In particular, *Objs*, contains the individuals and *Resource* mantains the association between a single element of the ontology with its iri.

8.2 Test scenarios

To verify if the functional requirements have been met, three different scenarios have been designed, they are depicted in Figure 8.3, Figure 8.4 and

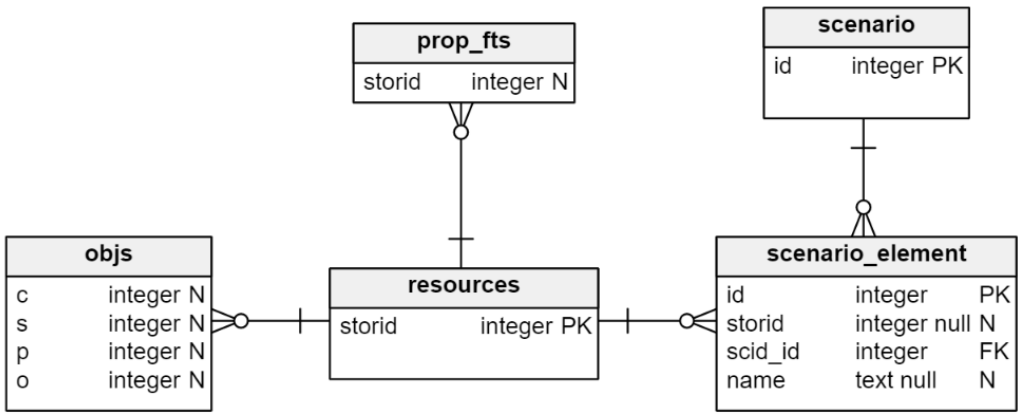


Figure 8.2: Tool for Item definition and Asset Identification: Database Design

Figure 8.5. They contain a reduct set of all the ECUs contained in a real vehicle, but the focus is on the main architectural aspects in this phase.

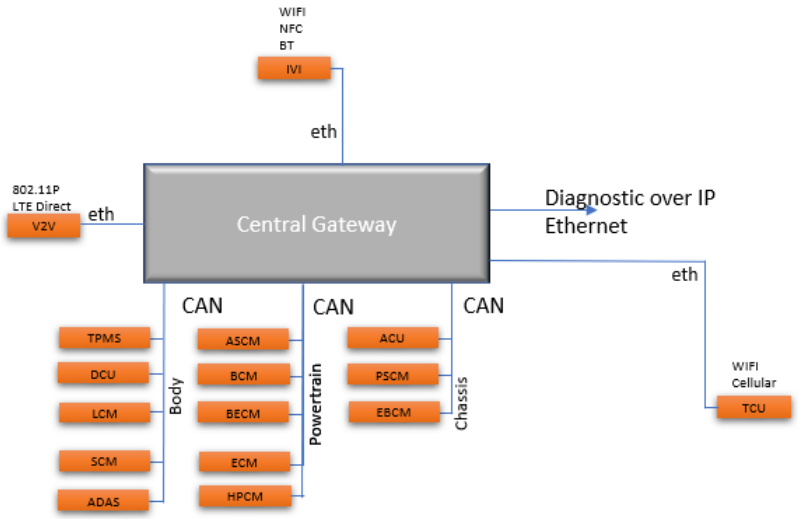


Figure 8.3: Complete Automotive architecture: Scenario 1

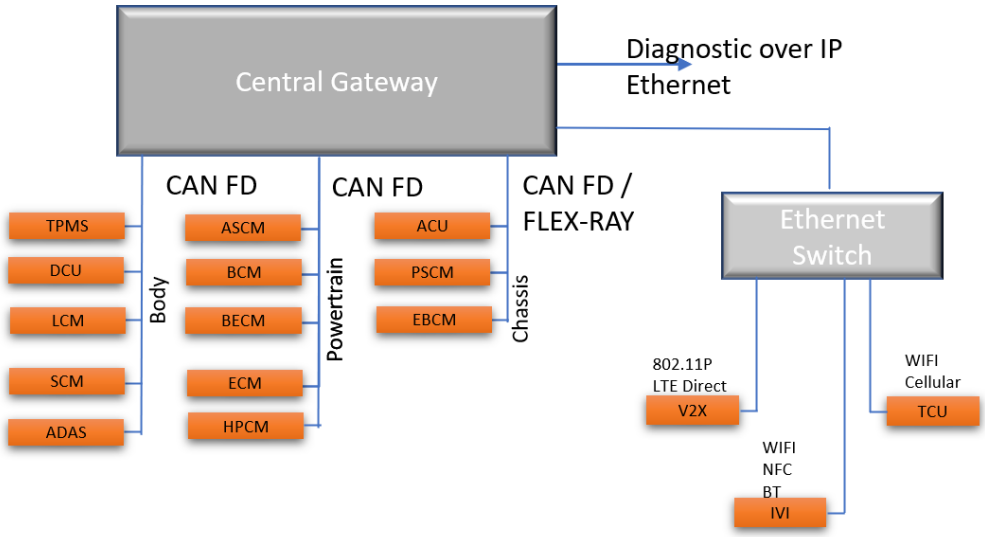


Figure 8.4: Complete Automotive architecture: Scenario 2

They were described using the web app and then the refinement algorithm was used above them. Thus the configuration of single filtering points was made for each of them. The specific architectures are taken from the opensource project and they allow to highlight the different levels of complexity: in fact, the one in Figure 8.3 is the simplest because it has only a security gateway and has complete segregation between different networks, so the refinement algorithm has to discover and to configure rules for a single firewall. The second architecture (Figure 8.4) instead has two interconnected security gateways and disjoint networks, so the messages travelling from a network to another have to cross both of them and thus the specific rule has to be configured in both devices. The last scenario in Figure 8.5 is the most complex because it is the evolution of the scenario in Figure 8.4 with the introduction of another component such as the *Body Computer*. It is another filtering point that is used for redundancy of the security gateway, so they have a similar configuration in order to increase the performance avoiding bottleneck in communications.

To perform the refinement phase and then the policy analysis, a set of DBC files is used: they contain more than 100 messages each, exchanged among all the different ECUs.

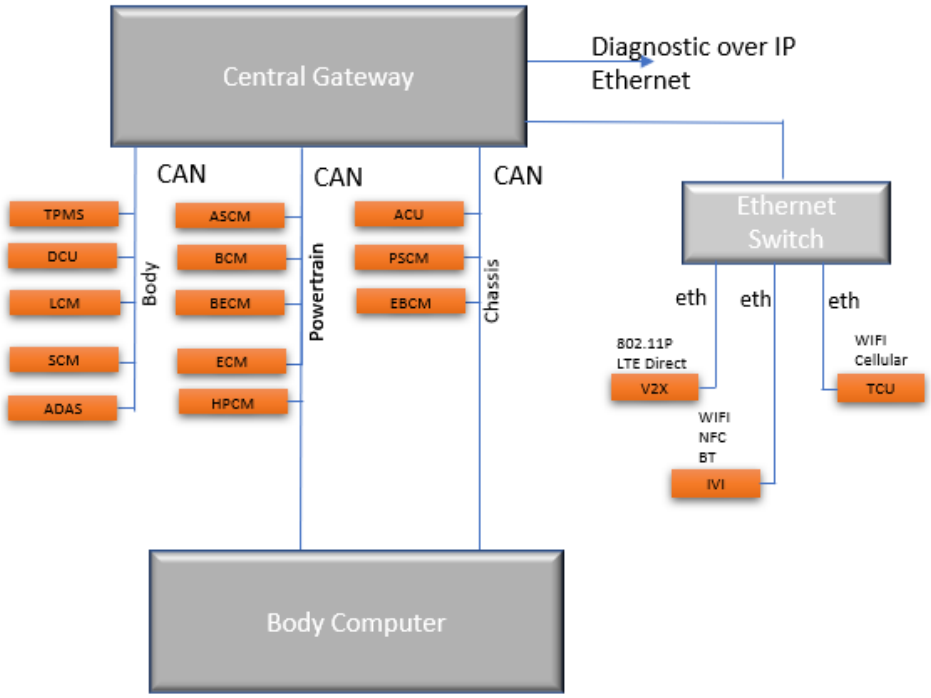


Figure 8.5: Complete Automotive architecture: Scenario 3

8.3 Policy Analysis test

Policy anomalies test

The *Intra-policy analyses and Inter-policy analyses* has been carried out on the architecture depicted in Figure 8.3, Figure 8.4 and Figure 8.5. In this section the results obtained will be discussed.

The *Intra-policy analyses*, which deal with the analysis of a single policy, demonstrates for all of the scenarios under test that the configuration taken in isolation is well-formed. This witnesses an intense design activity, that excludes any misconfiguration.

The *Inter-policy analyses*, that deals with policy analyses in relation to the landscape, shows different outcomes for the three scenarios. In fact, as regards the scenario with a single filtering point (Figure 8.3), the outcome of the analyses is trivial, because the result is the same as for the *Intra-policy*

analyses, since there is no other firewall. The scenarios depicted in Figure 8.4 and Figure 8.5 are more interesting because they contain more than a single filtering point, thus two specific anomalies have been revealed. In both scenarios, I have observed *Shadow anomalies and Spurious anomalies*. The former deals with a rule that is accepted from the downstream device and discarded from the upstream one, instead of the latter revealing the opposite situation. At first these outcomes looked like a misconfiguration, but focusing on the data analyses they resulted expected because these anomalies affect the rules that refer to the networks directly attached to the filtering device. In fact, in the Automotive scenario, the messages can be sent only from a proper set of ECUs and sometimes they do not have to cross the filtering zone they belong, thus the traffic of data can not be influenced by these anomalies, and for some aspects, this represents also a sort of protection.

Reachability test

In order to test the reachability I created a function that has the following prototype: `public Boolean checkReachability(String NodeS, String NodeD , String MsgId, String MsgName)`. It takes as parameters:

- the source Node;
- the destination Node;
- the message identification code;
- the message name.

and it recomputes the equivalent firewall between source and destination and creates a rule with the given field. Then it checks whether if it is contained or not inside the equivalent firewall and, in positive case, the function returns `true`, otherwise `false` In our context it is also useful to add another function to extract the reachability path: `public List<String> getReachabilityPath( String NodeS, String NodeD, String MsgId, String MsgName)`. It has the same parameters as the previous function but it computes the list of firewalls between the source and the destination instead of the equivalent firewall. Then if the list is not empty, it is iterated checking whether the rule is contained by all the firewalls in the list or not. If it is true the path is created and returned, otherwise, the function returns `null`.

This function is been used to test if the refinement algorithm has been properly designed and implemented, by launching it for each admitted message between two nodes, described in the XACML files checking whether

they are reachable or not. In the testing phase, I reuse the Test Scenario of the previous section, with the results that for the 100% of the messages they are reachable.

Chapter 9

Conclusion

This thesis can be seen as a starting point for the development of a bigger framework for automotive cybersecurity, it contains concepts taken from modern technology and design scenarios and it has been conducted to retrieve the basic knowledge to perform more complex analysis and projects.

The first work that has been assigned to me dealt with the study of software documentation and normative, useful to understand which are the needed background notions. It has been an important and interesting task because it allows to focus on the state-of-the-art of automotive cybersecurity field. It also makes me to reflect on a real business case and the market perspective that affects the field nowadays. Thus the research has not been only theoretical, but it has involved also the development of a tool, as a final product of this thesis, including all the solutions to the problems stated in Chapter 5. In this perspective, every single workproducts has been validated by cybersecurity companies, which have evaluated the expressivity of the description language and the usability of the application developed for the proof-of-concept. However, system testing has been carried out using the test scenario provided in Chapter 8. They have to be considered as a simplified example, but exhaustive at the same time because it contains the basic set of ECUs that can be retrieved on an OEM testbed. However, it represents the actual architecture and they do not cover other particular models made with different network protocols and other typologies of nodes.

The software proposed in Chapter 8 has to be intended just as a proof-of-concept, not as a complete and definitive tool, many upgrades can be made, from the usability to the improvement of the core functions in order to better adapt to the companies' needs.

Since the requirements of this Application has come from discussion with

these companies, the tool can be considered ready to be integrated into a bigger framework that can find applications in two main fields:

1. ISO/SAE 21434 Certification process, thus easing the task of a certification entity providing a framework to conduct all the steps stated by the standard

2. Enhancing the performance of security controllers, thus simplifying the management of *Automotive Security Gateway* providing an automatic configuration system starting from the landscape and Policy design requirements

A framework for Certification process

The definition of a language that formally describes assets and structural elements inside a vehicle can ease the task of people involved in the certification process because it provides a common terminology for item description and asset identification.

It paves the way to the development of new tools designed to perform the TARA[1]. The language developed for this final work can be the base upon which software that makes this task automatic relies on.

In addition, the analysis done in this final work can be used in the subsequent steps described by the ISO/SAE 21434 related to the risk response phase where the proper remediations and mitigations have to be evaluated in order to resolve cybersecurity risk arising with the TARA. In particular, the definition of a High-level design policy and the anomaly analysis can be used in relation to the threat introduced by the misconfiguration of the filtering device.

The policy and the Ontology have been written using very flexible languages so they can be updated and extended to describe new automotive architecture or to make the policy design more complex.

A tool for Security Gateway configuration

Another possible scenario in which this thesis finds application is strictly related to the configuration process of filtering devices. It is linked to two main problems:

[1]Threat Analysis and Risk Assesment

1. Mishandling of configurations, due to the company internal organization which sees different people involved in the design and deployment of policies.

2. Misconfigurations of device, thus the writing of wrong rules in filtering devices mounted in automotive architecture

The refinement algorithm proposed in this thesis tries to solve both problems, because it automatically converts the policy designed in the abstract with the real low-level representation and deployment. However, it describes a very general scenario and it could not be directly applied by the OEMs. So a future development starting from the refinement algorithm could be its customization to adapt to carmaker's needs. It can also be extended in order to receive as input not only the DBC files which are strictly related to the CAN network, but also other sources coming from FlexRay or Automotive Ethernet. It is the same for policy language, it is designed to parse and compute policies in XACML, but with minor changes, it can be modified for another policy format.

The policy tool library can be further extended in order to suit to the evolution of policy design. Since the model is XML-based it can be modified and adapted to different cases. In particular, the selector type, linked to the policy field can be customized in many different ways using the many classes included in the library or even by creating new classes to model properly the selectors.

Appendix A

Template of a communication Policy

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" xmlns:ont="
       http://www.example.com/onto" xmlns:xsi="http://www.w3.org/2001/
       XMLSchema-instance" PolicyId="policy_alg" RuleCombiningAlgId="urn:
       oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides" xsi:
       schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os␣
       xacml_os_schema2.xsd">
3    <PolicyDefaults>
4      <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</
           XPathVersion>
5    </PolicyDefaults>
6    <Target>
7      <Subjects>
8        <Subject>
9          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string
               -equal">
10           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string
                 ">ECU_Source</AttributeValue>
11           <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml
                 :1.0:subject:subject-id" DataType="http://www.w3.org/2001/
                 XMLSchema#string" MustBePresent="true" />
12         </SubjectMatch>
13       </Subject>
14       <Subject>
15         <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string
               -equal">
16           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string
                 ">type(ECU)</AttributeValue>
```

```
17          <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml
                :1.0:subject:subject-type" DataType="http://www.w3.org/2001/
                XMLSchema#string" />
18        </SubjectMatch>
19      </Subject>
20    </Subjects>
21    <Resources>
22      <Resource>
23        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath
              -node-match">
24          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string
                ">xpath_request(/ont:Ontology/ont:Node)</AttributeValue>
25          <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:
                xacml:1.0:resource:xpath" DataType="http://www.w3.org/2001/
                XMLSchema#string" />
26        </ResourceMatch>
27        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath
              -node-equal">
28          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string
                ">type(ECU)</AttributeValue>
29          <AttributeSelector DataType="http://www.w3.org/2001/XMLSchema#
                string" MustBePresent="false" RequestContextPath="//ont:
                Ontology/ont:Node/@type" />
30        </ResourceMatch>
31      </Resource>
32    </Resources>
33    <Actions>
34      <Action>
35        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
              equal">
36          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string
                ">Standard_Action(Authorize)</AttributeValue>
37          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml
                :1.0:action:action-id" DataType="http://www.w3.org/2001/
                XMLSchema#string" />
38        </ActionMatch>
39      </Action>
40    </Actions>
41  </Target>
42  <Rule Effect="Permit" RuleId="alg_rule">
43    <Target>
44      <Resources>
45        <Resource>
46          <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
                xpath-node-equal">
```

```
47            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#
                  string">ECU_dest</AttributeValue>
48            <AttributeSelector DataType="http://www.w3.org/2001/XMLSchema#
                  string" MustBePresent="false" RequestContextPath="//ont:
                  Ontology/ont:Node/@name" />
49          </ResourceMatch>
50          <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
                string-equal">
51            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#
                  string">msg_id</AttributeValue>
52            <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:
                  xacml:1.0:resource:message-id" DataType="http://www.w3.org
                  /2001/XMLSchema#string" />
53          </ResourceMatch>
54          <ResourceMatch>
55          </ResourceMatch>
56        </Resource>
57        <Resource>
58        </Resource>
59      </Resources>
60    </Target>
61  </Rule>
62  ...
63 </Policy>
```

Appendix B

Refinement algorithm implementation

```
import igraph
import json
import sys
from owlready2 import *
from policy_parse import *
from dbc_parse import *
from record import record
from graph_lib import callGraph, confGeneration


nome_file, nome_scenario, topos, policy_schema, policy_dir, dbc_dir,
    out_dir=sys.argv

onto = get_ontology(topos).load()

####1####Graph Creation
g=callGraph(onto)

####2####Filtering Element discovery

filtering_element=[]
conf={}
for e in onto.Gateway.instances():
    filtering_element.append(e)
    conf[e.name]=[]

####3####HL Policy

```

```
28 #from_to={'from':[to]}
29 from_to=XACMLpolicy(policy_dir, policy_schema)
30
31 ####4####DBC Parsing
32 #msg_SourceDest={'msg_id':[['from'],['to']]}
33 msg_SourceDest=parseNew(dbc_dir)
34 #ecu_net={'ecu':'net'}
35 ecu_net=DBC_ECU_to_net(dbc_dir)
36 ####5####Path Extraction
37
38 for source in from_to.keys():
39     f=onto.search(iri="*"+source, is_a=onto.Node)
40
41     for dest in from_to[source]:
42         t=onto.search(iri="*"+dest, is_a=onto.Node)
43         path_l=g.get_shortest_paths(g.vs['obj'].index(f[0]), to=g.vs['obj'
              ].index(t[0]), weights=None, mode='out', output='vpath')
44
45         #path_list
46         for path in path_l:
47
48             sub_path=[]
49             pos_start=0
50             pos_end=0
51
52             #single path
53             for n1 in path:
54
55                 if g.vs['obj'][n1] in filtering_element:
56
57                     pos_end=path.index(n1)
58                     sub_path.append(path[pos_start:pos_end+1])
59                     pos_start=pos_end
60             sub_path.append(path[pos_start:len(path)])
61
62
63             if len(sub_path)>1:
64                 for k in msg_SourceDest.keys():
65                     if source in msg_SourceDest[k][0] and dest in
                          msg_SourceDest[k][1]:
66                         x=k.split("/")
67
68                         #N chain to test
69                         node_d=g.vs['obj'][sub_path[len(sub_path)-1][-1]].
                              name
70                         innet=ecu_net[g.vs['obj'][sub_path[0][0]].name]
```

```
71                      i=0
72                      for sp in sub_path:
73                          i=i+1
74                          next_sp=sub_path.index(sp)+1
75                          if next_sp<len(sub_path):
76                              ef=g.vs['obj'][sp[-1]].name
77                              if g.vs['obj'][sub_path[next_sp][-1]].name
                                    not in conf.keys():
78                                  outnet=ecu_net[g.vs['obj'][sub_path[
                                        next_sp][-1]].name]
79                              else:
80                                  outnet=g.vs['obj'][sub_path[next_sp
                                        ][-1]].name
81                              e=record(ide=x[0], name=x[1], dest=node_d,
                                    nettin=innet, nettout=outnet)
82                              conf[ef].append(e)
83
84 ####6####Export JSON
85 key=['id_messaggio', 'label_messaggio', 'nodo_dest', 'net_source', '
       net_dest']
86 for x in conf.keys():
87     f=open(out_dir+"/"+str(nome_scenario)+"_"+str(x)+"1.json","w")
88     c_l=[]
89     pointer=g.vs['label'].index(x)
90     for y in conf[x]:
91     # print(y)
92         value=[]
93         value.append(y.identifier)
94         value.append(y.message_name)
95         value.append(y.node_dest)
96         value.append(y.net_in)
97         value.append(y.net_out)
98         c_l.append(dict(zip(key, value)))
99
100    g.vs['conf'][pointer].append(c_l)
101    json.dump(c_l, f, indent=1)
102    f.close()
103
104 print(g.vs['conf'][pointer][0])
```

Appendix C

WebApp: list of API

All the resources are under the root `poc_api/`

Table C.1: WebApp: list of API

Resource	Method	Path	Description
Class	GET	/classes	Get all classes
	GET	/classes/{chid}	Get info about a specific class
Node	GET	/individuals	Get all Nodes
	POST	/individuals	Create a new Node
	GET	/individuals/{iid}	Retrieve information related to a specific Node
	PUT	/individuals/{iid}	Update the information related to a Node
	DELETE	/individuals/{iid}	Delete a specific Node
Channel	GET	/channels	Get all channels
	POST	/channels	Create a new Channel
	GET	/channels/{chid}	Retrieve information related to a specific Channel
	PUT	/channels/{chid}	Update the information related to a Channel
	DELETE	/channels/{chid}	Delete a specific Channel
Asset	GET	/assets	Get all Assets class
	GET	/assets/{chid}	Retrieve information related to a specific Asset class
	PUT	/assets/{aid}	Update the information related to a specific Asset
	DELETE	/assets/{aid}	Delete a specific Asset
Graph	GET	/graph	Retrieve the graph representation of the landscape
Scenario	GET	/scenarios	Get all Scenario
	POST	/scenarios	Create a new Scenario
	GET	/scenarios/{sid}	Retrieve information related to a specific Scenario
	PUT	/scenarios/{sid}	Update the information related to a Scenario
	DELETE	/scenarios/{sid}	Delete a specific Scenario

Bibliography

[1] Stefan Seifert and Roman Obermaisser. «Secure automotive gateway — Secure communication for future cars». In: *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. 2014, pp. 213–220. DOI: 10.1109/INDIN.2014.6945510.

[2] Robert N Charette. «This car runs on code». In: *IEEE Spectrum* 46.3 (2009), p. 3.

[5] Jasmin Brückmann, Tobias Madl, and Hans-Joachim Hof. «An Analysis of Automotive Security Based on a Reference Model for Automotive Cyber Systems». In: Sept. 2017.

[6] Marko Wolf, Andre Weimerskirch, and Thomas Wollinger. «State of the Art: Embedding Security in Vehicles.» In: *EURASIP J. Emb. Sys.* 2007 (Jan. 2007).

[7] D. Crolla et al. *Automotive Engineering: Powertrain, Chassis System and Vehicle Body*. Elsevier Science, 2009. ISBN: 9781856175777. URL: https://books.google.it/books?id=EwcRnwEACAAJ.

[8] Karl Koscher et al. «Experimental security analysis of a modern automobile». In: *Security and Privacy (SP), 2010 IEEE Symposium on*. 2010, pp. 447–462.

[9] C. Smith. *The Car Hacker's Handbook: A Guide for the Penetration Tester*. No Starch Press, 2016. ISBN: 9781593277031. URL: https://books.google.it/books?id=Ao%5C_QCwAAQBAJ.

[11] Lee, Lin, and Liao. «Design of a FlexRay/Ethernet Gateway and Security Mechanism for In-Vehicle Networks». In: *Sensors* 20 (Jan. 2020), p. 641. DOI: 10.3390/s20030641.

[14] ISO. «ISO 11898-1:2003 - Road vehicles - Controller area network». In: *International Organization for Standardization* (2003).

[15] ISO. «ISO 21434:2019 - Road vehicles - CyberSecurity Engineering». In: *International Organization for Standardization* (2019).

[16] UNECE. «World Forum for Harmonization of Vehicle Regulations (WP.29)». In: (2019), vi, 129 p. : URL: http://digitallibrary.un.org/record/3824138.

[18] Stephen Checkoway et al. «Comprehensive Experimental Analyses of Automotive Attack Surfaces.» In: *USENIX Security Symposium*. 2011.

[19] A. Weimerskirch M. Wolf and C. Paar. «Security in automotive bus systems». In: *Proceedings of the Workshop on Embedded Security in Cars*. 2004.

[23] Jean-Baptiste Lamy. «Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies». In: *Artificial Intelligence in Medicine* 80 (2017), pp. 11–28. ISSN: 0933-3657. DOI: https://doi.org/10.1016/j.artmed.2017.07.002. URL: https://www.sciencedirect.com/science/article/pii/S0933365717300271.

[26] E. Al-Shaer et al. «Conflict classification and analysis of distributed firewall policies». In: *IEEE Journal on Selected Areas in Communications* 23.10 (2005), pp. 2069–2084. DOI: 10.1109/JSAC.2005.854119.

[27] Cataldo Basile, Alberto Cappadonia, and Antonio Lioy. «Network-Level Access Control Policy Analysis and Transformation». In: *IEEE/ACM Transactions on Networking* 20.4 (2012), pp. 985–998. DOI: 10.1109/TNET.2011.2178431.

[29] Benjamin Klotz et al. «VSSo: A Vehicle Signal and Attribute Ontology». In: Oct. 2018.

[30] Michael Feld and Christian Müller. «The automotive ontology: managing knowledge inside the vehicle and sharing it between cars». In: Nov. 2011, pp. 79–86. DOI: 10.1145/2381416.2381429.

[31] Andreas Maier, Hans-Peter Schnurr, and York Sure-Vetter. «Ontology-Based Information Integration in the Automotive Industry». In: vol. 2870. Oct. 2003, pp. 897–912. ISBN: 978-3-540-20362-9. DOI: 10.1007/978-3-540-39718-2_57.

[32] Axel Reymonet, Jérôme Thomas, and Nathalie Aussenac-Gilles. «Ontology Based Information Retrieval: an application to automotive diagnosis». In: June 2009.

[33] Marcel Rumez et al. «Integration of Attribute-based Access Control into Automotive Architectures». In: June 2019. DOI: 10.1109/IVS.2019.8814265.

Sitography

[3] *OEM*. URL: https://en.wikipedia.org/wiki/Original_equipment_manufacturer. (accessed: December 6, 2021).

[4] Daniel Kolb and Rafael Schmid. *Cyber security- the next big challenge for automotive OEMs*. URL: https://accilium.com/en/cyber-security-the-next-big-challenge-for-automotive-oems. (accessed: December 6, 2021).

[10] STMicroelectronics. *Automotive Gateway*. URL: https://www.st.com/en/applications/body-and-convenience/automotive-gateway.html#key-products. (accessed: December 6, 2021).

[12] Timo van Roermund. *Secure connected cars for a smarter world*. URL: https://www.nxp.com/docs/en/white-paper/SECURE-CONNECTED-CARS-WP.pdf. (accessed: December 6, 2021).

[13] *CAN bus*. URL: https://en.wikipedia.org/wiki/CAN_bus. (accessed: December 6, 2021).

[17] *Cybersecurity standard*. URL: https://en.wikipedia.org/wiki/Cybersecurity_standards#ISO/SAE_21434. (accessed: December 6, 2021).

[20] C. Valasek and C. Miller. *A Survey of Remote Automotive Attack Surfaces*. URL: https://ioactive.com/wpcontent/uploads/2018/05/IOActive_Remote_Attack_Surfaces.pdf. (accessed: December 6, 2021).

[21] Andy Greenberg. *Hacker remotely Kill A Jeep on the Highway- with me in it*. URL: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/. (accessed: December 6, 2021).

[22] *Ontology*. URL: https://en.wikipedia.org/wiki/Ontology_(information_science). (accessed: December 6, 2021).

[24] *XACML*. URL: https://en.wikipedia.org/wiki/XACML. (accessed: December 6, 2021).

[25] *A Brief Introduction to XACML*. URL: https://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html. (accessed: December 6, 2021).

[28] *Reachability analysis*. URL: https://en.wikipedia.org/wiki/Reachability_analysis. (accessed: December 6, 2021).