POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Simulating Malicious Attacks on VANETs for Connected and Autonomous Vehicles

Supervisor
prof. Antonio Lioy

Candidate
Erasmo Notaro

December 2023

Summary

In recent years, the automotive industry has witnessed remarkable technological advancements. Modern vehicles have become increasingly connected and autonomous, serving as the cornerstone for the development of Cooperative Intelligent Transportation Systems (C-ITS) and the implementation of future Smart Cities. This necessitates the employment of hundreds of Electronic Control Units (ECUs), a growing number of in-vehicle Cyber-Physical Systems (CPS), and the adoption of new wireless communication technologies that support the deployment of Vehicular Ad-Hoc Networks (VANETs).

This phenomenon has garnered the interest of researchers worldwide who have conducted various case studies on unaltered passenger vehicles, demonstrating the feasibility of remotely compromising safety-critical ECUs. The resulting approach highlights how exploiting a chain of vulnerabilities, from a wireless entry point to safety-critical ECUs, might allow remote control of the vehicles.

Indeed, with the advent of inter-vehicle communication (IVC) technologies, today's vehicles are equipped with vehicle-to-everything (V2X) wireless interfaces, such as Dedicated Short-Range Communications (DSRC) for the ETSI ITS-G5 standard, which is responsible for expanding the vehicle's remote attack surface. Additionally, due to cooperative driving automation that grants partial or complete control of the vehicle to V2X Applications, there is an increased impact on the safety of passengers and Vulnerable Road Users (VRUs) in the case of both in-vehicle and V2X security threats.

Originating from these problems, this thesis work aims to demonstrate a possible implementation of realistic malicious attacks on VANETs, compliant with the ETSI ITS-G5 standard and based on the Artery open-source V2X Simulation Framework. The thesis also proposes a feasibility evaluation of the simulated malicious attacks in the presence of VANET with secured ITS communications and validates these attacks on the FEV Hardware in the Loop (HiL) platform, which is equipped with the CohdaWireless MK5 device.

Within this work, I firstly provide a description of the architecture of both in-vehicle and vehicular ad-hoc networks, with an emphasis on related security and safety risks. I detail in-vehicle and V2X threats by analyzing famous case studies on cyber-physical remote attacks. Following this, I outline a detailed description of the European V2X standard, i.e., ETSI ITS-G5, and its security mechanisms, which are necessary for a comprehensive understanding of the implementation of attacks and their execution results.

Then, a brief overview of the state of the art of VANET simulators is offered, with a focus on the Artery framework, the Vanetza open-source project, the SUMO traffic simulator, and the OMNeT++ network simulator.

In the core of the thesis, I describe in detail the process of designing and implementing realistic V2X attack scenarios. The implemented attack scenarios (Sybil, Message Modification, Replay, and Black Hole) are located on the A55 and A4 highways, outside the city of Turin, and they are carried out exploiting both Cooperative Awareness Messages (CAM) and Decentralized Environmental Notification Messages (DENM) of the ETSI ITS-G5 standard. Following this, I detail the execution of attacks in the simulated VANET, and since the ETSI ITS-G5 standard provides security mechanisms, I describe the results of tests conducted on several secured scenarios based on the implemented attacks.

As this thesis was developed at the FEV company, I had the opportunity to validate the simulated malicious attacks on the FEV HiL platform. This allowed me to use the CohdaWireless software products and the MK5 module, which is one of the most commonly used devices in aftermarket deployments and serves as a reference design for automotive production. The thesis work continues with a description of the process of reverse engineering the CohdaWireless device's operation, carrying out attack validation, and reporting a discovered flaw that permits message processing on the device, even with an expired timestamp.

Towards the end of this dissertation, to introduce possible future work in the domain of penetration testing, I present the approach and result of a performed black-box fuzz testing against the CohdaWireless "exampleETSI" binary, which is the official implementation of the European V2X standard for CohdaWireless devices.

Acknowledgements

I would like to express my sincere gratitude to Prof. Lioy for granting me the opportunity to pursue this master's thesis and for overseeing and enriching the entire work with his extensive knowledge.

I wish to convey my heartfelt thanks to Dott. Ing. Cicilloni for generously sharing his expertise, experience, and time during these months at FEV. His support allowed me to define the objectives of this work and successfully carry out a remarkably interesting and innovative project.

Contents

Chapter 1

Introduction

In the modern era, the prevalence of self-driving vehicles, equipped with extensive connectivity, is becoming increasingly widespread. This advancement is facilitated by significant technological progress that has permeated the automotive industry over the last decade. Modern vehicles are equipped with over 100 Electronic Control Units (ECUs) and contain more than 100 million lines of code in their overall architecture. This evolution meets the growing demand for autonomous driving capabilities and infotainment services, which undoubtedly improve road safety, traffic management, and passenger experience. However, this progress also introduces several security challenges. Specifically, the presence of safety-critical ECUs capable of autonomously controlling in-vehicle Cyber-Physical Systems (CPS) and the increasing number of wireless communication interfaces pose a substantial risk of remote safety-critical attacks.

Historically, vehicles lacked external connections, and plausible attacks required physical access to the vehicle's internal network. However, contemporary vehicles increasingly rely on external network connectivity for assistance, navigation, and infotainment purposes. Consequently, this evolution expands the potential remote attack surface that malicious actors could exploit to deliver in-vehicle threats. Various worldwide case studies have demonstrated the feasibility of remotely compromising safety-critical ECUs on unaltered consumer vehicles, showcasing their ability to remotely control vehicles at both low and high speeds. This heightened awareness among OEMs about cybersecurity issues has led to a shift from the previously employed "security through obscurity" approach to adopting regulations, standards, and security best practices. Notably, the ISO/SAE 21434 standard holds considerable significance as it delineates guidelines for fostering a cybersecurity culture, establishing policies and processes, and managing risks for both original equipment manufacturers (OEMs) and their suppliers.

The emergence of Vehicle-to-Everything (V2X) communication technologies plays a pivotal role in the domain of Connected and Autonomous Vehicles (CAVs) and the development of Cooperative Intelligent Transportation Systems (C-ITS). Information exchange between vehicles via periodic V2X messages establishes the basis of V2X networks, commonly known as Vehicular Ad-Hoc Networks (VANETs). Inter-Vehicle Communication (IVC) is crucial for extending sensor-based perception limited by line of sight (LoS) to an enhanced neighbor perception achieved through real-time information sharing among nearby vehicles and infrastructure. This enables the implementation of multiple V2X applications, significantly enhancing autonomous driving capabilities and providing substantial advantages for the future development of smart cities.

However, despite the expansion of V2X communication technologies, which increases the vehicle's remote attack surface and the likelihood of safety-critical in-vehicle remote

attacks, it is crucial also to consider V2X attacks. Although confined to VANETs, these attacks can significantly impact CAVs and their passengers' safety. The enhanced autonomous driving capability involves autonomous decision-making based on neighbor perception achieved through VANETs message exchange. Therefore, V2X malicious attacks, by manipulating the in-transit messages, can directly alter vehicles' perception, leading to traffic disruptions, discomfort, or even safety-critical events.

1.1 Goal of the Thesis

This thesis originates from the growing sensitivity to cybersecurity's importance within the automotive industry. Focused on emerging inter-vehicle communication technology, it aims to raise awareness among readers about the potential risks V2X attacks pose to passenger safety and Vulnerable Road Users (VRUs).

This thesis demonstrates the realistic implementation of malicious attacks on VANETs compliant with the ETSI ITS-G5 standard, executed within a V2X simulation environment. It encompasses Sybil, Message Modification, Replay, and Black Hole Attacks, involving both vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications. These implemented attacks highlight the potential for achieving local and extended execution scopes using Cooperative Awareness Messages (CAMs) and Decentralized Environmental Notification Messages (DENMs).

Over the years, the ETSI TC ITS WG5 working group has addressed security and privacy issues within the ETSI ITS-G5 standard, releasing several technical specifications that outline a security framework for C-ITS. In this context, this thesis aims to conduct an experimental evaluation of attacks' feasibility when utilizing a secure V2X communication channel instead of an unprotected one. Using various test case scenarios, it points out the results of this analysis, illustrating, for example, how Sybil attacks remain possible under certain conditions despite the employment of secured communication.

As the attacks are simulated, their successful execution is confined to the specific software implementation of the ITS-S in the utilized V2X Simulator (i.e., Artery). To overcome this limitation, it was necessary to test the implemented attacks on a real-world device. As this thesis is conducted in collaboration with FEV Italy, access to Cohda Wireless devices and associated software provided by the FEV company, which are also employed in its HiL Platform, has been possible. This access has permitted an in-depth exploration of the most equipped V2X module in both testing and production vehicles globally. An essential aspect of this testing involved integrating the Cohda Wireless device into the V2X simulation scenario, providing virtual GNSS information to align Cohda Wireless On-Board Unit (OBU) behavior with the virtual ego vehicle and simulating the receipt of ETSI ITS-G5 packets as if originating from another legitimate device (e.g., another Cohda Wireless OBU). This interaction achieved between the Cohda Wireless devices and V2X Simulator facilitated the integration of the latter into the XX HiL Platform. Consequently, it becomes possible to validate the developed V2X malicious attacks, demonstrating their successful execution on a common real-world device and emphasizing the associated risks.

The aforementioned case studies on remote compromise of safety-critical ECUs underscore the significant responsibility of ensuring the security of Telecommunication Control Units (TCUs) by both OEMs and their suppliers. Serving as the bridge between the in-vehicle network and the external world, TCUs represent the primary entry point for in-vehicle attack exploit chains. Anticipating potential future developments in vulnerability assessment and penetration testing domains, this thesis delineates the approach and

results of the conducted black-box fuzz testing utilizing both AFL and Radamsa fuzzers. It aims to uncover software flaws that could lead to application crashes or exploitable vulnerabilities. It is conducted against the Cohda Wireless application "exampleETSI", responsible for implementing the European standard of V2X communications (i.e., ETSI ITS-G5) in all Cohda Wireless devices.

Chapter 2

Background

2.1 Vehicular Ad Hoc Network

The emergence of Vehicle-to-Everything (V2X) communication technologies has enabled Inter-Vehicle Communication (IVC), thereby supporting the deployment of Vehicular Ad-Hoc Networks (VANETs), which are pivotal for the advancement of future smart cities. Indeed, IVC plays a crucial role in obtaining a comprehensive understanding of the surrounding environment. This involves the utilization of raw data collected and shared among nearby vehicles and infrastructure, forming the basis for the development of Intelligent Transportation Systems (ITS) applications. With the extensive connectivity among vehicles, it becomes possible to achieve an environmental perception previously confined solely to on-board sensors.

2.1.1 Vehicular Ad Hoc Network Technology

Vehicular Ad Hoc Network (VANET) is a technology that brings together different existing technologies such as **Ad Hoc Networks**, **Wireless LAN**, and **Cellular Networks** to achieve an intelligent Inter-Vehicle Network. VANETs, in some respects, are similar to Mobile Ad Hoc Networks (MANETs) in terms of infrastructure concerns; in fact, they do not rely on a fixed infrastructure for communication and the dissemination of information. On the other hand, it must be said that VANETs have specific requirements, such as high performance, due to the application context involving vehicles moving at high speeds and the exchange of information with highly dynamic content.

The aim of VANETs, as mentioned before, is to achieve ubiquitous vehicle connectivity, rather than being limited to specific locations like service stations or homes. In this way, a variety of applications such as **cooperative traffic monitoring**, **traffic flow control**, **blind crossing**, **collision prevention**, thereby advancing ITS.

Since VANETs are an aggregation of several technologies, as indicated in [1], it is possible to consider three different architectures:

- **pure Cellular/WLAN**. In this network scenario, fixed cellular gateways and WLAN access points are used at traffic intersections to collect information and indirectly connect the network nodes. However, this architecture is very costly because it depends entirely on infrastructure. In fact, since direct vehicle-to-vehicle communications are not possible, each vehicle can communicate using only the cellular gateways or access points available in the area.

- **pure Ad Hoc**. In this network scenario, all the vehicles and roadside wireless devices can form a Mobile Ad Hoc Network (MANET) to perform direct Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communication, thus enabling intelligent applications such as **blind crossings**.
- **Hybrid**. In this hybrid scenario, we have a mixed network where both ad hoc and WLAN/Cellular technology are used. In this case, nodes with both WLAN and Cellular interfaces can serve as gateways for nodes that have only a WLAN interface. This way, in addition to single-hop links, there are also multi-hop links, allowing all nodes of the network to remain connected to the world.

As mentioned earlier, while VANET share many elements with Mobile Ad Hoc Network (MANET), such as the absence of a fixed infrastructure, short radio transmission range, self-organization, self-management, etc., they are considered, for all intents and purposes, a subset of MANET due to the following distinguishing aspects:

- **High dynamic topology**. This is due to the high speed of vehicles, which has a strong effect when vehicles travel in opposite directions, as on highways.
- **Frequently disconnected network**. For similar reasons, especially in low-density network scenarios the connection can be interrupted causing inconvenience, especially for applications that require uninterrupted connectivity.
- **Real-Time constraints**. Most VANET applications do not typically require high data rates, but strict delay constraints are crucial. This is especially true for applications such as blind crossings and collision avoidance.

2.1.2 Vehicular Ad Hoc Network Architecture

As previously established, interconnection and communication among vehicles give rise to the so-called Vehicular Ad Hoc Network (VANET). This technology enables Inter-Vehicle Communication (IVC), also known as **Vehicle-to-Vehicle** (V2V) communication, and extends to **Vehicle-to-Infrastructure** (V2I), **Vehicle-to-Pedestrian** (V2P), and **Vehicle-to-Network** (V2N) communication.

All these communication paradigms serve the primary purpose of enabling the implementation of **safety applications** that enhance road safety by preventing dangerous situations. In addition to basic safety applications, such as **collision avoidance**, which helps avoid traffic collisions, there are other basic and advanced applications that are implemented in VANETs as technologies advance over time. These aspects will be explored in greater detail in the section 2.3.

From an architectural perspective, VANET is composed of Application Units (**AUs**), On-Board Units (**OBUs**), Roadside Units (**RSUs**), and, in the case of C-V2X, LTE/NR base stations (**eNB/gNB**) [2].

- **AUs** are software applications typically integrated into OBUs to facilitate safety or infotainment functions by processing various messages from the VANET according to predefined application logic.
- **OBUs** are devices installed in vehicles and are essential components of VANETs. The primary purpose of OBUs is to exchange relevant information to enable the functioning of implemented vehicular applications. To accomplish this, they are

equipped with one or more wireless communication interfaces, depending on the applicable standard, a GNSS receiver for precise positioning, velocity, and time (PVT) information, and a Central Processing Unit for processing message transmission and reception. To enhance the quality of shared informations, it is possible for OBUs to receive additional relevant data from external in-vehicle sensors.

- **RSUs**, unlike OBUs, are devices situated on the roadside, typically along roadways or at intersections, and are often connected to roadside infrastructure [3]. The primary role of RSUs is to support V2I communications, enabling various application scenarios for safety and infotainment, similar to other communication paradigms. Like OBUs, RSUs are equipped with wireless communication interfaces, a GNSS receiver, and a Central Processing Unit for communication processing.

- **eNB/gNB** stands for Evolved/Next-Generation NodeB and refers to base stations in a vehicular networks using the C-V2X standard as their V2X technology [4]. These base stations use LTE Uu or NR Uu interfaces for communication with devices. As will be clarified in subsection 2.2.2, using a cellular channel for V2X communication presents both advantages and disadvantages.

Based on the devices listed above, which constitute a VANET, we can analyze the different communication paradigms introduced at the beginning of this subsection 2.1.1:

- **V2V (Vehicle-to-Vehicle).** This type of communication requires close proximity between endpoints because it involves direct communication between vehicles. It is enabled through the use of OBUs that connect via short-range communication protocols such as IEEE 802.11p or LTE PC5.

- **V2I (Vehicle-to-Infrastructure).** Similar to V2V, this communication paradigm also requires close proximity between endpoints. In V2I communication, one endpoint is a vehicle equipped with an OBU, and the other endpoint is a roadside infrastructure equipped with an RSU. The exchange of information is bidirectional and is essential for the existence of some VANET applications, which will be discussed in section 2.3.

- **V2P (Vehicle-to-Pedestrian).** The primary objective of this type of communication is to support the development of pedestrian collision mitigation and avoidance systems. It aims to overcome the limitations of conventional pedestrian protection systems, which typically rely on sensors like RADARs, LiDARs, and Computer Vision, all of which have line-of-sight constraints. Jing, P. et al. [5] have pointed out that, although this type of communication is still in development, the main idea would be to use smartphones by exploiting existing wireless interfaces like Wi-Fi or by introducing new ones, such as DSRC.

- **V2N (Vehicle-to-Network).** V2N communication occurs between a vehicle and an Application Server located within the ICT infrastructure. This type of communication uses the cellular network and is possible when C-V2X technology is employed. The benefits and complications of this approach will be explored further in subsection 2.2.2.

- **V2N2V/I/P (Vehicle-to-Network-to-Vehicle/Infrastructure/Pedestrian).**

  Connections through the communication paradigms mentioned above, in the presence of C-V2X VANETs, can be established using the existing mobile network

infrastructure. In cases where direct communication between two endpoints (e.g., Vehicle-to-Vehicle) is not possible due to excessive distance between them, the cellular network can be leveraged to connect these two terminals by routing messages through an application server.

2.2 Technologies for Vehicular Communications

The automotive industry, cooperating with standardization organizations and academic research, has developed various standards that are behind Inter-Vehicular Communication (IVC) to support future ITS applications.These standards are:

- **ETSI ITS-G5**
- **IEEE 1609**
- **C-V2X**

2.2.1 DSRC

The standard ETSI ITS-G5 has been developed since 2007 by the European Telecommunications Standards Institute (ETSI) ITS technical committee in reference to the well-known U.S. project IEEE 1609, also known as Wireless Access in Vehicular Environments (**WAVE**).

The WAVE project defined changes starting from the IEEE 802.11 standard to meet the stringent constraints of vehicular transport systems. It was precisely from these changes that the **IEEE 802.11p** standard was born and subsequently adopted as a reference for both the U.S. project Dedicated Short-Range Communications (**DSRC**) and the European project ETSI ITS-G5.

The DSRC acronym includes "Dedicated" because it operates over reserved radio spectrum bands, which differ in North America, Europe, and Japan. These bands are allocated by the U.S. Federal Communications Commission (FCC) for exclusive use in **DSRC-based** applications. Depending on the type of V2X applications, each DSRC band can be used as a single frequency channel or divided into multiple channels.

One of the major limitations of DSRC is due to **CSMA/CA** (Carrier Sense Multiple Access with Collision Avoidance). CSMA/CA is the principal contention-based medium access control (MAC) scheme employed by DSRC, as it is based on **IEEE 802.11p**. The limitation arises from the fact that, in a high vehicle density scenario, the intensity of channel contention among vehicles significantly increases, resulting in a considerable degradation of IEEE 802.11 performance [6]. This leads to a high transmission collision rate and a large channel access delay, which is not suitable for the ultra-low latency requirements of VANET applications.

The widely investigated performance degradation of the **IEEE 802.11p** standard in highly dense scenarios led to the recent introduction of **quality of service** (QoS) privileged classes through the development of the new **IEEE 802.11ax** amendment [7].

2.2.2 C-V2X

The concerns mentioned earlier regarding pure-DSRC V2X communication solutions have piqued the interest of the research community in the potential of cellular technologies to support reliable V2X communications for various applications [6].

Therefore, in addition to the previously mentioned IEEE and ETSI standards, both of which are based on IEEE 802.11p, in 2015, the **3rd Generation Partnership Project** (3GPP) initiated an adaptation study of the LTE network to support V2X applications (Cellular V2X, **C-V2X**) [8]. The objective was to specify suitable transport mechanisms for both V2V/P/I messages and V2N messages to ensure complete coverage and continuity of service. In fact, one of the key challenges addressed by the 3GPP work was resolving the issue of network presence or absence in the communication areas. This led to the specification of two communication modes:

- **PC5 interface.** This interface is used for V2V/P/I communications, enabling direct communication between two devices without routing through the network.
- **Uu interface.** This interface is used for V2N communications, connecting devices to the LTE radio base station (eNB) and operating similarly to regular communications. It can also facilitate V2N2V/P/I type communications.

The solution based on the **Uu interface** does not differ from the standard communication process between the device and the radio base station. The primary advantage lies in its ability to provide broader communication range, even when direct line-of-sight between terminals is unavailable. However, the main disadvantage is increased latency compared to the PC5 case, as communication must pass through the network.

2.3 Vehicular Network Applications

Over the last 15 years, numerous standardization organizations have formalized V2X messages and their possible related applications. Although there is no universally accepted classification of V2X applications, according to [9], these can be divided into the following types of services:

Road Safety services, aimed at avoiding accidents and improving road safety. Traffic management and efficiency services, designed to facilitate the flow of traffic. Infotainment and Business services, to create value for both the driver and passengers on board.

2.3.1 Road Safety Applications

These applications are primarily employed to reduce the probability of traffic accidents and the death of vehicle passengers. A significant percentage of accidents that occur every year worldwide are associated with intersection vehicle collisions. A research study by the National Bureau of Statistics [10] shows that drivers can avoid at least 60% of rear-end collisions, 30% of head-on collisions, and 50% of road-related accidents as long as they are warned 0.5 seconds before there is a danger of collision; with a 1-second warning time, 90% of accidents can be avoided.

In this regard, these safety applications aim to prevent such collisions with other vehicles by providing information and assistance to drivers. This can be accomplished

by sharing information such as position, speed, altitude, heading, etc., between vehicles and/or infrastructure. Regarding vehicles, this data will be used for real-time prediction of possible collisions or dangerous situations, while with regard to infrastructure, this information is used to identify potential hazardous locations on roads.

Below are some of the main road safety applications:

- **Intersection Collision Warning (ICW).** This safety application aims to avoid lateral collisions for vehicles approaching road intersections. This is accomplished by signaling warnings to drivers of approaching vehicles to reduce impact damage or avoid collisions completely.
- **Overtaking Vehicle Warning.** This safety application aims to prevent collisions between vehicles in an overtaking situation. This is achieved through communication from `vehicle_1` (the first vehicle that intends to overtake `vehicle_3`) indicating its overtaking maneuver to `vehicle_2`. Upon receiving and processing this message, `vehicle_2` immediately stops its overtaking procedure.
- **Forward Collision Warning (FCW).** The goal of this safety application is to prevent rear-end collisions with other vehicles by providing assistance to drivers. According to [10], rear-end collisions are almost always caused by sudden braking and/or driver distractions. Therefore, thanks to this application's logic, when a critical situation is detected (e.g., insufficient safety distance between vehicles), a warning is displayed to the driver.
- **Head-On Collision Warning.** This safety application aims to reduce the risk of head-on collisions by sending early warnings to vehicles traveling in opposite directions that are about to overtake a vehicle ahead. This use case is also referred to as "Do Not Pass Warning" according to [11].
- **Pre-Crash Sensing/Warning.** The purpose of this safety application is different from the previous one, where only a warning is shown. Instead, it enables optimized use of vehicle equipment to mitigate the effects of a crash. Vehicles and available roadside units periodically exchange messages containing a high amount of detailed information. This exchanged data is evaluated in real time and cross-referenced with data from other vehicles to calculate the possibility of a collision. When a collision is highly probable, the vehicle prepares to use equipment such as actuators, airbags, motorized seat belt pre-tensioners, extensible bumpers, etc.
- **Emergency Vehicle Warning.** This use case informs other vehicles in its vicinity to clear an emergency corridor and facilitate the transit of an emergency vehicle, such as an ambulance or police car. As discussed in more detail in chapter 4, this information is included in a type of message that can be rebroadcasted in the vicinity by other vehicles and roadside units.
- **Emergency Electronic Brake Lights (EEBL).** This safety application aims to inform other vehicles, especially those following, that it is hard braking. This safety measure is triggered before the previously mentioned Forward Collision Warning. As in the previous use case, the message is rebroadcasted in the vicinity by other vehicles and roadside units through cooperation.

Additionally, there are other safety applications that differ only in the type of warning and triggering conditions. They all involve the forwarding of messages containing specific warnings that describe the emergency situation. These messages can be rebroadcasted

among nearby vehicles and roadside units. Among these safety applications, we mention: Stationary Vehicle Warning, Traffic Condition Warning, Hazardous Location Notification, and Control Loss Warning [12].

2.3.2 Traffic Management and Efficiency Services

Traffic management and efficiency applications primarily focus on improving vehicle traffic fluidity. However, improving traffic management may offer secondary benefits such as traffic coordination and assistance. According to [10], these types of applications are divided into two groups:

- **Speed Management.** Speed management applications aim to assist the driver in managing their vehicle's speed for smooth driving and to avoid unnecessary stopping.
  - **Regulatory and contextual speed limits**, for example, consist of a capable Roadside Unit broadcasting the current local speed limits (both regulatory and contextual) at a given frequency. This not only improves road safety but also enhances traffic flow and reduces vehicle pollution.
  - **Green Light Optimal Speed Advisory (GLOSA)**, on the other hand, allows traffic lights to broadcast timing data associated with their current state (e.g., how much time remains before the light switches from red to green, etc.).
- **Cooperative Navigation.** This type of application is used to increase traffic efficiency by managing vehicle navigation through cooperation among vehicles and roadside units.
  - **Traffic information and recommended itineraries**, for example, inform approaching vehicles of traffic abnormalities and provide recommendations in case of traffic jams.
  - **Cooperative Flexible Lane Change** enhances mobility efficiency through flexible allocation of a dedicated lane (e.g., reserved for public transport or emergency vehicles) to specific vehicles. They are granted temporary or permanent access rights under specific conditions, such as the absence of buses and emergency vehicles.

2.3.3 Infotainment and Business Applications

These application fields aim to provide on-demand information to passing vehicles on either a commercial or non-commercial basis. These services may include infotainment, comfort, and vehicle or service life cycle management.

- **Cooperative Local Services.** These services focus on infotainment that can be obtained from local services within the ITS network infrastructure, without accessing the global internet.
- **Point of Interest Notification.** It, for example, informs about the presence of locally-based services or points of interest, providing dynamic information such as opening hours and prices. Local Electronic Commerce, on the other hand, consists of a roadside unit capable of processing digital payments for service reservations or goods purchases.

- **Global Internet Services.** These services focus on data that can be obtained from wider internet services. An example could be **Map Download and Update**, where, in the case of an RSU with internet connectivity, vehicles can update their navigation maps for security and efficiency reasons. Another possible application could be **Instant Messaging**, which allows the exchange of messages using a global instant messaging service.

Advanced Applications

In addition to the above examples of vehicular network applications, other more complex ones have emerged with technological advancements. For example, the evolution of the C-V2X standard, specifically in 3GPP Release 15 [13], brings advanced use cases with very stringent requirements not supported by previous technologies. Among these, we find:

- **Vehicles Platooning.** Using periodic data exchange between a lead vehicle and the following ones, this application allows vehicles to dynamically form a traveling group to carry out platoon operations. This type of operation allows following vehicles to be autonomously driven.
- **Extended Sensors.** This is a complex application that involves the exchange of raw or processed data collected from on-board sensors (e.g., lidar and/or cameras) among vehicles, roadside units, pedestrian devices, and V2X application servers. This way, each vehicle has an enhanced perception compared to what would have been achieved with on-board sensors alone.

2.4 In-Vehicle Network

2.4.1 In-Vehicle Network Architecture

Over the years, in-vehicle architecture has changed significantly. Early vehicles used point-to-point connections for communication between various ECUs. However, with the rapid increase of in-vehicle functions and required ECUs, the wiring system became more complex and difficult to manage. As a result, development was required that led to the disappearance of point-to-point connections in favour of a serial bus, a solution that is less complex and more scalable. However, since different vehicle functions require different data rates, the use of a single serial bus for the entire vehicle would not have been the optimal choice for the design of the in-vehicle communication network. In modern vehicles, there are different types of buses used in different areas of vehicle control: **CAN**, **LIN**, **FlexRay**, and **MOST** networks. These buses differ from each other in terms of communication speed and the transmission mode they support [14]. CAN and FlexRay buses are typically used for **critical ECUs** that require high-speed networking, such as those involved in the powertrain. The LIN network is typically used for ECUs that require slower transmission speeds, such as those that control lights, air conditioning, seats and doors, while MOST networks, on the other hand, are mainly used for ECUs that require high transmission speeds, such as infotainment systems, including audio, video and voice.

Based on the characteristics offered by each bus, the vehicle network is divided into different domains [15].

- **Powertrain and Chassis Domain.** The powertrain domain of a vehicle is primarily responsible for managing the engine control and chassis control systems, which include the ECM (Engine Control Module), ABS (Anti-lock Braking System), EPS (Electric Power Steering), and other similar components [16]. These systems require a high level of communication reliability, which is provided by stable high-bandwidth communication capabilities. This domain mainly relies on three communication buses, namely the **high-speed CAN** bus (with a maximum speed of 1 Mbps), **CAN-FD** bus (with a maximum speed of 5 Mbps), and the **FlexRay** bus (with a maximum speed of 10 Mbps). The CAN bus used in powertrain domain is commonly referred to as **CAN-C** (critical). In particular, manufacturers are gradually moving to the use of CAN-FD bus because it has a higher communication rate and a larger data load capacity of up to 64 bytes. FlexRay is a high-speed, critical, fault tolerance bus. It features cyclic timing traffic that includes both a static segment for periodic messages and a dynamic segment for event messages, ensuring critical real-time communication performance. This bus is typically required in applications such as advanced chassis control and communication backbones.

- **Body Domain.** It is used for non-critical component control and information services in the non-powertrain domain, using a **low-speed CAN** bus or **LIN** bus to perform the corresponding function control. Components connected to this bus typically include headlights, electric windows and doors, seats, and HVAC (Heating, Ventilating, and Air Conditioning). In addition, some ECUs with wireless capabilities, such as PKE (Passive Keyless Enter) and PATS (Passive Anti-Theft System), are also connected to this bus. For example, when the keyless entry system ECU receives an unlock command, it sends a CAN message to the BCM (Body Controller), which controls the door functions via the LIN bus [15].

- **High-Speed Information Service Domain.** This domain refers to communication, which can transmit a large data stream but has no control function. In this area, we can identify three subdomain: **Infotainment**, **Telematics**, **ADAS**. The infotainment subdomain uses the MOST bus, which is suitable to connect infotainment peripherals such as displays, audio, video, radio devices, GPS navigation, etc., thanks to its transmission rate of up to 50 Mb/s and its ability to manage up to 64 MOST devices in a ring topology [17]. However, in recent years, the use of Ethernet in vehicles has become more widespread thanks to Broadcom's **BroadR-Reach** Ethernet technology [18], which offers higher bandwidth while significantly reducing connection costs and cabling weight. This is in line with the growing consumer demand for in-vehicle connectivity and advanced driver assistance (ADAS), which is why this technology is widely used in the telematics and ADAS subdomains.

Automotive Gateway

Communication between these different domains takes place via a central gateway. Originally, the structure of the vehicular network was less complex, and the gateway was used only to forward variable-speed CAN bus packets. However, in today's complex network structure, the role of the gateway is no longer limited to forwarding CAN data, but it has become an essential component of the communication system [19]. The gateway interconnects the above heterogeneous vehicular networks and processes the data between them. It provides physical isolation and protocol translation for data transmission between functional domains (powertrain, chassis and safety, body control, infotainment, telematics, ADAS). In addition, the gateway plays a fundamental role in managing the

data that vehicles exchange with external interfaces for safety, entertainment, convenience, and diagnostic purposes.

According to [20], the main function of automotive gateway is to provide secure and seamless communications between ECUs of different domain, acting as a bridge between the internal networks and the extern ones. It has two units that allow the in-vehicle network to communicate with the outside world. These units are the On-Board Diagnostics II (OBD II), which is responsible for the physical connection, and the Telematics Control Unit (TCU or T-Box), which is responsible for the wireless ones.

As aforementioned, the gateway has had an interesting evolution over the years becoming a more and more complex unit which nowadays has a crucial role about security. In the past, the main role was only to forward message between different network segment eventually changing the encapsulation protocol if necessary. However, with the growing evolution of today's vehicles, the gateway capabilities have significantly increased including, in addition to data routing and protocol traslation, OTA Management, Key Management, Diagnostic Routing, Firewall, Intrusion Detection etc. So as of today, it is capable to manage remote Over-The-Air (OTA) updates of ECUs with which it is connected, to process and store network keys and certificates in a secure way and to route and translate diagnostig messages between external diagnostic devices and ECUs. From a security perspective, the Automotive Gateway serves as a dual-purpose system. It acts as a firewall, filtering inbound and outbound network traffic in accordance with predefined rules. This helps to prevent unauthorized access and ensures the security of the in-vehicle networks. Additionally, it acts as an Intrusion Detection System (IDS), continuously monitoring network traffic for any anomalies that may indicate potential intrusions or security breaches.

However, the level of security for all the aforementioned features greatly depends on the specific location in which they are implemented. For this reason, the gateway ECU is the perfect node in the vehicle due to the fact that it offers a high level of security, while also providing the necessary connectivity to all the in-vehicle networks. The high level of security in the Automotive Gateway is ensured through several key aspects. Firstly, the relatively compact software architecture minimizes the likelihood of vulnerabilities or flaws. Secondly, the utilization of trusted Automotive Grade (AG) OEM code which, adheres to standards like A-SPICE II or ISO26262, offers assurance regarding the development process. Lastly, the secure boot process ensures that only firmware signed by the OEM can be executed on the device, adding an additional layer of security. As we will observe in the upcoming section 3.2, despite the implementation of these security measures, various case studies conducted on multiple cars demonstrate that they are usually not enough to limit break-ins, even in the case of remote attacks.

On-Board Diagnostics

As mentioned above, the OBD (On-Board Diagnostics) plays a fundamental role in providing access to the in-vehicle network from the external environment. OBD2 or EOBD (European On-Board Diagnostics) is used to detect and report faults. It is an interface based on the SAE J1962 [21] standard that allows the in-vehicle network to communicate with external devices. The communication is done according to the specification ISO 15765–4 [22], which defines the requirements for CAN-based communication between the in-vehicle network and the diagnostic connector. Unlike the physical interface, the wireless interfaces are multifarious and are located in the TCU.

Telematic Control Unit

The TCU is at the heart of the complex telematics system in modern vehicle architecture. Because it is connected to a multitude of in-vehicle data and control busses and is the first target of an attacker, its security is an important challenge for the automotive industry. It feeds data from the external network into the in-vehicle network to provide functions or entertainment capabilities. Modern T-Boxes have the following wireless interfaces to establish remote communications.

- **Bluetooth**: This technology is designed for short-range wireless communication. Bluetooth Classic Radio (BR /EDR) [23] transmits data over 79 channels in the 2.4 GHz band (ISM) and is optimised for audio streaming, wireless speakers and in-car entertainment systems with a maximum throughput of 3 Mb/s.

- **Cellular**: Automotive service providers such as GM's OnStar [24] offer remote emergency call service, diagnostic data collection, theft protection, and many other features via the cellular network that make the car a connected marvel.

- **WiFi**: In the in-vehicle networking environment, WiFi is used to connect the infotainment subsystem to consumer electronic devices, including smartphones and laptops. When in-vehicle WiFi meets cellular communications [18], its potential is completely unlocked. Specifically, these portable WiFi hotspots provide on-the-go Internet access, allowing drivers to receive real-time traffic status updates via GPS, passengers to watch online videos while driving, etc. With a maximum data rate of 9608 Mb/s [25], the WLAN standard IEEE 802.11ax [26] provides sufficient bandwidth for device-to-device (D2D) video streaming, in-vehicle entertainment and also Over-The-Air (OTA) updates [27].

Recently, vehicles have become increasingly connected to the environment, and to support Intelligent Transportation System (ITS) applications, the telematics box plays a crucial role. As described in subsection 2.1.2, V2X communication technology enables the exchange of information data between two or more vehicles, as well as with pedestrians and infrastructure. The vehicle can exchange information about itself, such as latitude, longitude, elevation and direction, as well as information about the environment, such as events and the related danger level. According to [28] [29], this communication technology is expected to reduce accidents, improve road network capacity, provide value-added convenience applications and so on.

In addition, the T-Box is also used to allow vehicles to exchange V2X messages and the media available are WiFi and Cellular.

- **WiFi** technologies use the 802.11p standard [30] for the PHY and MAC layers to enable short-range transmission of messages according to the IEEE Wave and ETSI ITS -G5 standards.

- **Cellular** technologies use the 3GPP C-V2X standard [31] to enable short and long distance transmission of messages according to the LTE-V2X [32] and 5G-V2X [33] standards. It is possible for both 802.11p and LTE/5G technologies to co-exist in the same vehicle [34] [35] [36]. For example, ITS -G5 can be used for direct V2V and V2I communications over short distances to enable real-time safety applications such as collision avoidance, while LTE-V2X provides broader coverage and connectivity for accessing cloud-based services, advanced traffic management systems and other non-real-time applications.

The presence of external interfaces makes it clear that the on-board network is no longer isolated. Despite all the new features and benefits mentioned above, these new communication channels also bring new risks. For this reason, Gateway must apply various cybersecurity measures to protect the in-vehicle network from external threats. In the next chapter, we analyse some known car hacking, the related exploited access points and cybersecurity countermeasures.

2.4.2 In-Vehicle Network Communication

As discussed in subsection 2.4.1, modern vehicles are equipped with an increasing number of electronics to meet the diverse requirements of drivers and passengers. Numerous sensors, actuators, and ECUs are interconnected, exchanging information and coordinating controls. This enables the implementation of complex functionalities such as Advanced Driver Assistance Systems (ADAS), Electric Power Steering (EPS), Infotainment Systems, etc. In the majority of vehicles, communication between these ECUs occurs through four in-vehicle networking technologies: Controller Area Network (CAN), Local Interconnect Network (LIN), FlexRay, and Media Oriented Systems Transport (MOST).

CAN

CAN Bus is a serial communication protocol used for real-time safety-critical functions in road vehicles and other controlled applications [14] [37]. It is a multi-master protocol developed by Robert Bosch GmbH in 1986 and it was designed for automotive applications that require data rates of up to 1 Mbps and a high level of data integrity. In addition to automotive applications, the protocol CAN is also used as a general embedded communication system for microcontrollers and as a standardised communication network for industrial control systems.

The CAN bus system enables each ECU to communicate with all other ECUs without complex dedicated wiring. Specifically, an ECU can prepare and broadcast information (e.g. sensor data) via the CAN bus. The broadcasted data is accepted by all other ECUs on the CAN network and each ECU can then check the data and decide whether to receive or ignore it.

The CAN bus consists of a single two-wire bus architecture that helps reduces cabling. At the same time, the distributed architecture of the network provides easy-maintenance and decreases the overall system cost. The protocol uses differential wiring mode, represented by `CAN_High` and `CAN_Low`, which enhances the immunity to noise and electrical interference. From a logic point of view, the CAN specifications use the terms 'dominant' bits and 'recessive' bits. In the CAN protocol, a dominant bit represents a logical 0 and is actively driven to a voltage by the transmitter. On the other hand, a recessive bit represents a logical 1 and is passively returned to a voltage by a resistor which represented also the recessive level. When different CAN nodes transmit dominant and recessive bus levels simultaneously, a collision occurs. In such cases, the dominant bit 'wins' according to the `AND-Logic` principle, where the dominant bit takes precedence over the recessive bit. This means there is no delay to the higher-priority message, and the node transmitting the lower priority message automatically attempts to re-transmit six bit clocks after the end of the dominant message. This makes CAN very suitable as a real-time prioritised communications system.

The CAN protocol has message-based communication provided via frames. Standard CAN frame has 11 bits identifier (CAN 2.0A) and it is the type used in most cars. The

extended 29-bit identifier frame (CAN 2.0B) is identical except the longer ID and it has a wide use in the SAE J1939 protocol intended for heavy-duty vehicles. Each frame has a message Id, data field, cyclic redundancy checksum (CRC), and some control bits. Every node listens to each frame and processes the relevant ones based on the message identifier field, which is also used for the arbitration.

It incorporates a set of built-in features that enable robust communication. As mentioned earlier, if two nodes start transmitting at the same time, the non-destructive arbitration mechanism resolves the conflict by allowing the highest priority node to continue transmission without interruption. A node that loses arbitration re-queues its message for later transmission, and the CAN frame bit-stream proceeds without errors until only one node remains transmitting. Therefore, the node that transmits the first recessive bit loses arbitration. Since the 11 (or 29 for CAN 2.0B) bit identifier is transmitted by all nodes at the beginning of the CAN frame, it is the node with the lowest identifier that emerges as the winner in arbitration. Another feature is carrier sense multiple access with collision detection (CSMA/CD), which dictates that the nodes must wait for a certain amount of inactivity before transmission. This helps sense if the bus is idle, ensuring that collisions are avoided.

The CAN bus incorporates bit-level and message-level error checking mechanisms. At the bit-level, the transmitting node monitors the bus for any discrepancies between the transmitted bit and the observed bit on the bus. On the other hand, the message-level error checking in the CAN bus includes frame check over acknowledgment (ACK), cyclic redundancy checksum (CRC), and end of frame (EOF) fields. After transmitting a frame, the transmitter node writes a recessive bit to the ACK field. If a receiving node successfully receives the message, it overwrites the ACK field with a dominant bit. However, if there is a transmission error, the ACK field remains recessive, indicating a transmission error.A CAN frame includes a CRC field of up to 21 bits for data integrity. If any node calculates a different CRC value than the transmitter node, an error flag will be sent. The CRC delimiter, ACK delimiter, and EOF bits have fixed values and must always be recessive. During the frame form check, if these bits are dominant, an error is generated.

LIN

Differently from the CAN-BUS, LIN is a low-cost, low-speed and easy-to-implement [14]. It offers a cost-effective alternative for connecting low refresh rate module. In fact, it is primarily used in simpler and less time-critical tasks as traditional central door lock activation, window lifter control, mirror adjustment, air condition, seats, steering wheel button modules, and various low refresh rate sensors.

The LIN network predominantly utilizes a linear bus topology and follows a master-slave communication pattern. Slave nodes synchronize themselves with the master at every transmission of the message header, and remain synchronized within the required bit rate tolerance throughout the rest of that frame. Within the LIN network, the master node sequentially communicates with each slave node by sending information requests, while the slave nodes respond with their respective data upon being polled. The maximum supported number of nodes is 16, typically one master and up to 15 slaves. As mentioned earlier, the frame structure of the LIN bus consists of a header and a response. The header is transmitted by the master node, while the response is sent by the slave nodes and can have a maximum size of 8 bytes. In the header frame there are three field:

- **Sync Break Field (SBF)** which acts as a ‘start of frame’. It is specified as a

dominant condition (bus low) at least 13 bit times to ensure the bus will see it as a break. This because it is longer than the worst case data pattern of all dominant.

- **Sync field** which has a predefined value of 01010101. It is used to allow Slaves to determine the transmission rate which the Master uses, i.e. the time between two falling edges. This procedure is called auto baud detection.
- **Protected Identifier Field (PID)** which consist of two sub-fields. The Frame ID which acts as an identifier for each LIN message sent and Parity with which Slaves determine the validity of the ID field.

As for the answer, there are Data and Checksum fields.

- **Data** which is generated by the polling of Master. Its length can be customized, but it is typically linked to the ID range, e.g. ID [0–31]: 2 bytes, [32–47]: 4 bytes, [48–63]: 8 bytes. Several signals can be packed into one frame and and the data bytes are transmitted using the Least Significant Bit (LSB) order.
- **Checksum** which is used to ensures the validity of the LIN frame, as in CAN. In the LIN 1.3 protocol, a conventional 8-bit checksum is utilized. In contrast, the LIN 2.0 protocol introduces an enhanced checksum that encompasses not only the data field but also the identifier field.

While the fault-tolerance capabilities of the LIN network are relatively limited compared to other prominent in-vehicle networks, its polling transmission mechanism effectively mitigates arbitration delays and message collision. Delays may arise in communication with low-cost LIN slaves. For this reason, is included a 'response space' between the header and response which allows slave nodes sufficient time to react to the master's header.

FlexRay

With the continuous need to implement even more safety and driver-assistance functions, the speed, quantity, and reliability of data transmitted between the vehicle's several ECUs increase accordingly. Safety systems and advanced control features combine multiple sensors, actuators, and ECUs that require increasingly synchronization and performance which the CAN bus cannot satisfy. This has led to the development of several fault-tolerant and deterministic protocols with far greater data rates than CAN. Among these, FlexRay emerged as the in-vehicle communications bus that, with a data rate of 10Mbit/sec and orientation toward safety- and time-critical applications, meets these new challenges in the next generation of vehicles. Although FlexRay aims to solve mainstream in-vehicle network challenges, it does not displace the other two dominant in-vehicle protocols (i.e., CAN and LIN), which are employed, following a cost-saving approach, respectively for mainstream powertrain communications and low-cost body electronics.

Similar to the CAN protocol, FlexRay is based on a multi-master communication structure, but the FlexRay node cannot access the bus in an uncontrolled way in response to application-related events. Instead, they follow a precisely pre-set communication cycle that dedicates a specific slot to each FlexRay message (Time Division Multiple Access - TDMA). This approach guarantees deterministic data communication and ensures that all nodes can be tested independently since, as detailed below, every FlexRay node has its slot/slots. Every FlexRay node is synchronized to the same clock and waits for its turn to

write on the bus. This is assured by a distributed and fault-tolerant clock synchronization mechanism in which all FlexRay nodes not only continuously correct the beginning of the time slot (offset correction) but also its duration (slope correction). Additionally, to reduce failure risks, FlexRay employs a redundant communication channel which, when necessary, can increase the data rate from 10Mbit/sec to 20Mbit/sec. Thus, for each FlexRay message, it is possible to choose if the additional channel, if available, is used for fault tolerance or additional bandwidth.

The FlexRay communication cycle is the primary element of the media-access scheme in FlexRay. Its duration is fixed at network design and typically lasts around 1–5 ms. Each communication cycle is equally in duration and organized into four main parts: Static Segment, Dynamic Segment, Symbol Window, and Network Idle Time.

- Static Segment: Comprising reserved slots utilized for deterministic data that arrives at a fixed period. Each static slot is assigned to a FlexRay node that has the opportunity to transmit its data. When the time slot passes, the FlexRay node must wait until the next cycle to transmit its data. Since the exact point in time is known in the communication cycle, the data arrives in a deterministic way, and programs know precisely how old the data is.
- Dynamic Segment: Composed of slots called "minislots" wherein their content is not static. To accommodate a wide variety of data that does not always demand high-speed and critical requirements, the dynamic segment allows FlexRay nodes to occasionally transmit data. When a minislot occurs, a FlexRay node has the opportunity to broadcast its frame. If it does not broadcast, it loses its spot in the dynamic frame, and the next minislot occurs. Since the data is sent in broadcast, only one FlexRay node can transmit, thus future minislots must wait until the node completes its data broadcast. When the dynamic frame window ends, the minislots that have not transmitted must wait until the next cycle for another opportunity to broadcast.
- Symbol Window: Serves to check the functionality of the Bus Guardian, an independent control mechanism ensuring that a FlexRay node only gains access to the bus during its turn in the communication cycle. High-level applications do not interact with the symbol window.
- Network Idle Time: A time segment that concludes the communication cycle. During this segment, all FlexRay nodes calculate the correction factors necessary to synchronize their local clocks with the global one.

From a security standpoint, the FlexRay network provides scalable fault-tolerance by allowing single or dual-channel communication. For instance, for security-critical applications, the devices connected to the bus may use both channels for transferring data, enabling redundancy. Additionally, the Bus Guardian protects the channel from interference caused by communication that is not aligned with the communication cycle schedule.

MOST

To accommodate the increasing demands of infotainment devices unmet by existing protocols, the automotive industry designed the Media Oriented System Transport (MOST) protocol. This protocol has been developed with a new and flexible architecture based on a hierarchical communication model. The communication model comprises three

stages: HMI, System Controller, and System Slaves, and is based on the Master-Slave principle. At the top level lies the Human Machine Interface (HMI), a controller that exposes the overall functionality to the user. In the middle, there is the system controller, which covers part of the system functionality and shares partial system knowledge with the HMI. The lowest level consists of the system slaves whose functions are utilized by one or more system controllers. They do not possess full system knowledge; instead, this choice enhances flexibility regarding configuration. Therefore, it is easy to remove or add system slaves from a MOST system since they function as plug-and-play devices.

A MOST network typically implements a ring topology, where all nodes are interconnected in a circular manner. This allows each MOST frame to pass through every MOST device. The MOST bus supports up to 64 nodes. In addition to the ring topology, it is also possible to realize a star topology, although this configuration is not widely used. To ensure high availability and fault tolerance, similar to the FlexRay protocol, the MOST bus also utilizes a double-ring topology.

MOST technology, to avoid unnecessary overhead, enables the transmission of a continuous bit stream. For this purpose, the presence of specially designed MOST devices called Timing Masters is necessary. These Timing Masters send the MOST frame at a fixed frequency, and since all other MOST devices are interconnected in a ring topology, all receive the MOST frame at the same frequency. When the relevant communication partners connect to the same streaming channel, bit streaming begins. Connection and disconnection are managed by a query from the function block Connection Master (CM), which requests that the relevant data source possess the suitable number of streaming channels allocated by the Timing Master. The Timing Master is responsible for managing the channel resource allocation table.

Regarding the Access layer, there are three different channel allocation strategies:

- Time Division Multiplex Access (TDMA) is used to allocate channels in synchronous access. Each node possesses a specific time slot in which it can communicate on the channel.
- Token-based Access is used for asynchronous channels. To access the packet channel, each node must acquire the token circulated on the network.
- Carrier Sense Multiple Access (CSMA) is used to access the control channel. In this case, the node listens to the channel to assess the absence of other traffic before transmitting on the shared medium.

The MOST bus possesses different variants that differ from each other based on the data transfer rates. MOST25, MOST50, and MOST150 offer data transfer rates of 25, 50, or 150 Mbps, respectively, and each is tailored to meet specific requirements of automotive applications. Today, MOST150 is the most widely used protocol in automotive multimedia networks.

2.5 Cyber-Physical Systems

Modern automobiles are controlled by a heterogeneous combination of up to 200 distinct Electronic Control Units (ECUs) [38]. These components oversee a broad range of functionalities, including the drivetrain, brakes, lighting, and entertainment. As mentioned earlier, the interconnection of ECUs permits the implementation of complex features that

improve both safety and convenience. For instance, Automatic Emergency Braking is a complex feature aimed at urgently applying the brakes if the driver fails to respond in case of slow or stopped traffic ahead. This is enabled by the presence of Cyber-Physical Systems (CPS), which are characterized by a tight integration of computation, communication, and physical processes, allowing certain ECUs to control in-vehicle actuators. Therefore, a relevant number of CPS are also considered safety-critical systems since their failures might cause loss of lives, damage, or catastrophic accidents.

Although Functional Safety aims to guarantee that a system operates correctly, ensuring human and environmental safety, it is possible that malicious attacks targeting safety-critical ECUs can exploit cyber-physical systems, leading to human injury or death. Indeed, functional safety standards focus on unintentional errors or faults rather than deliberate malicious actions. Additionally, it is worth noting that the attacker's aim might not be to carry out a complex physical action on the vehicle but instead a simple one that can have a significant impact. Certainly, a malicious actor can compromise a certain safety-critical ECU responsible for controlling the brake actuator and then have the ECU send legitimate messages as if the vehicle requires emergency braking. However, a malicious attacker can forge malicious messages as those in transit on the network, claiming that emergency braking is required. In this way, with less effort, the attacker manages to modify the vehicle's status and then elicit a reaction from the responsible ECU for that response.

2.5.1 Cyber-Physical Features

Recognizing the threat posed by malicious actors and the increasing set of external communication interfaces of modern vehicles, it is interesting to examine the cyber-physical features they offer. Among the various attack scenarios detailed in section 6.1, the most dangerous are those aimed at compromising safety-critical ECUs and thus performing unsafe actions. Hence, it is worthwhile to highlight the following, as each of them could cause the mentioned unsafe actions, potentially leading to human injury or death in the event of a remote cyber-physical attack. Here are the most common cyber-physical features in modern automobiles:

- **Anti-lock Braking System (ABS):** This system is designed to prevent wheel locking during braking, maintaining vehicle stability and steering without losing traction [39].
- **Electronic Stability Control (ESC):** This technology monitors individual wheel speed, steering angle, throttle position, and various accelerometers to automatically modulate engine torque and wheel speed, increasing traction when the vehicle line no longer follows the steering angle (i.e., oversteer or understeer).
- **Lane Keep Assist System (LKAS):** This system prevents cars from unintentionally leaving their lane. A camera detects lane lines, and an ECU calculates whether the car is about to leave its lane. By sending messages to the steering or brakes, the car can adjust its position within the lane [40].
- **Hill Start Assist:** Also known as Hill-holder, this system helps prevent roll-back when starting from a stopped position on an incline. Despite its seemingly harmless nature, this functionality allows an ECU to brake the vehicle, potentially causing unwanted braking in the event of an attack.

- **Automatic Emergency Braking:** This system, also known as a pre-crash system, forward collision warning system (FCW), or collision mitigation system, aims to prevent or reduce the severity of a collision by applying the brakes. Collision calculations are performed by an ECU, and messages are sent to the brakes.
- **Active Cruise Control (ACC):** This system maintains the desired speed of the vehicle by slowing it down or speeding it up depending on the vehicle ahead. A computer controls the vehicle's braking and acceleration based on sensor readings.
- **Park Assist:** Performed by a dedicated ECU that collects data from sensors and calculates how to turn the steering wheel to park in a specific location. The desired steering wheel position is transmitted to the steering wheel ECU, which then turns the steering wheel.

Many of these cyber-physical features fall under the umbrella of Advanced Driving Assistance Systems (ADAS), which are becoming increasingly prevalent in our automobiles. ADAS systems are transferring the control of safety-critical functions, such as braking and steering, to computers, algorithms, and software that operate independently of the driver's input. While it is true that nearly all vehicle accidents result from human error, the integration of ADAS can potentially prevent these errors, subsequently reducing fatalities and injuries. However, it's crucial to acknowledge that these features might also open avenues for vehicle control in the event of a malicious attack.

As elaborated further in the analysis of real-world case studies in section 3.2, the pervasive integration of on-board intelligent systems has rendered the execution of malicious actions on modern vehicles increasingly feasible. This thesis aims to heighten readers' awareness of the imminent risks resulting from inadequate cybersecurity culture, policies, and processes within the automotive industry, which may pose significant threats today and, even more so, in the future.

Chapter 3

Cybersecurity in vehicular communication

Year after year, we witness a constant increase in the number of Electronic Control Units (ECUs) in new automobiles. With the growing demands for safety, security, and comfort, many ECUs are required to control various vehicle features, such as engaging the ABS system, monitoring the steering wheel angle, or simply managing heated seats. In fact, these ECUs have very different responsibilities, resulting in varying levels of privileges and, typically, protection.

Although each ECU has been designed for a specific task, it is often subordinate to certain conditions dictated by a multitude of states from other ECUs or sensors. This means that, in order to function correctly, each ECU needs to exchange a multitude of data with other ECUs to make decisions on how to act. Additionally, some ECUs, to provide innovative services and comfort, also communicate with the outside world, as well as the in-vehicle network. This, of course, makes them cutting-edge automobiles, but at the same time, poses the greatest risk to the OEM (Original Equipment Manufacturer) and passengers.

3.1 Vehicle Attack Surface Analysis

In the past, when the in-vehicle network had no connection with the outside, the only feasible attacks presupposed an attacker's ability to physically connect to the vehicle's internal network. Although it must be underlined that this assumption is certainly unlikely, given the physical limitations, it's worth adding that, according to[41], an attacker with physical access to the automobile can easily carry out non-computerized attacks as well (e.g., cutting the brake lines). For this reason, the presence of ECUs capable of accepting, processing, and taking action based on information from the outside exposes them to possible remote attacks, significantly increasing the likelihood of such an automobile being targeted.

According to Miller and Valasek[42], the evaluation of the likelihood of a safety-critical remote attack against modern automobiles generally involves a combined assessment of three factors: the size of the remote attack surface, the segmentation of the in-vehicle network, and the extent to which features allow ECUs to physically control the vehicle. In fact, concerning safety-critical attacks, a common approach typically requires three stages, which are highly dependent on the aforementioned assessments. First, the attacker needs

to gain remote access to the in-vehicle network and inject malicious messages, thereby taking control of the ECU responsible for handling communication with the outside. Second, due to possible network segmentation, the attacker needs a second attack stage, which involves bridging messages from one network segment to another, where the target safety-critical ECU is assumed to be connected. Finally, the attacker must make the ECU behave in a manner that compromises vehicle safety. Therefore, if the vehicle possesses several cyber-physical features that allow computer-controlled actions on the automobile, it is clear that the success of the attack will be highly probable.

This takes into account the scenario in which the attacker's goal is to carry out a **safety-critical attack**, intending to cause the most significant damage possible, such as human injury or death. If the attacker's goal is solely to steal sensitive data, eavesdrop on in-vehicle microphones, or capture the in-vehicle camera feed, the subsequent attack stages listed above may be unnecessary, and the attack can be considered complete after compromising the first target ECU. This is particularly applicable if the first ECU is responsible for both external communication and microphone management.

As mentioned earlier, since the first step of a safety-critical cyber-attack consists of gaining remote access to the in-vehicle network, we will now analyze the potential attack surface of a typical modern-day car and characterize the threat models under which this surface is exposed.

Considering the full range of input and output channels present in modern automobiles, it is convenient to classify them, as proposed by Checkoway et al. [43], into three categories: **indirect physical access**, **short-range wireless access**, and **long-range wireless access**.

3.1.1 Indirect Physical Access

This category encompasses all the physical interfaces that are accessible to an attacker via a third-party entity, thereby utilizing an indirect approach. This takes into account a more realistic attack scenario, as it assumes a typical situation in which the attacker cannot easily access the target automobile.

- **OBD-II:** It is a system that implements the vehicle's self-diagnostic and reporting capability and is used to access the status of various vehicle sub-systems. To facilitate this status check, it provides direct access to the CAN buses and, as evidenced by [44], it can offer sufficient access to compromise a broad range of automotive systems. Although we assume that the attacker does not have direct access to the car, the OBD-II is an entry point that can be used through the personnel of a car workshop as a third party. In the past, access for diagnostics and ECU programming was achieved using dedicated devices (e.g., Toyota's Diagnostic Tester).

  Today, most manufacturers have adopted an approach based on the use of a personal computer paired with a "PassThru" device plugged into the vehicle's OBD-II port.

  Additionally, W. Yan in [44] has analyzed approximately 20 OBD-II dongles, pointing out that as many as 50% of them had security flaws, such as weak encryption, exposed keys, and communication protocol hijacking. This underscores the fact that, with considerable ease, an attacker can move from a compromised Windows-based computer to the vehicle's internal network, exfiltrating sensitive data, or even worse, reprogramming ECUs [43].

- **CDs, USBs, CarPlay, and AndroidAuto:** Another crucial class of physical entry points focuses on infotainment systems. In this case, exploiting multimedia playback is the most common entry point to gain access to the In-Vehicle Infotainment (IVI) systems. An attacker can deliver malicious input through well-prepared media files, and this can be done using traditional methods (e.g., CDs/DVDs or SD cards) or modern approaches (e.g., Bluetooth or CarPlay/AndroidAuto). In fact, Checkoway et al. in [43] demonstrated how it is possible to turn a song into malware (e.g., a Trojan Horse) by adding extra code to a digital music file. When played, this song can alter the firmware of the car's stereo system, providing attackers with an entry point to launch a more sophisticated attack that affects other in-vehicle subsystems. Since the songs appear unaltered, this type of attack could be distributed on the internet without arousing suspicion.

  In addition to this media-based attack scenario, an application-based attack scenario is possible, exploiting CarPlay and/or AndroidAuto connections. A compromised mobile phone that connects to these in-vehicle services can serve as a vector to transit from the phone to the victim's vehicle Head Unit. This can lead to the exploitation of potential unpatched vulnerabilities, which could grant access to sensitive user information, reveal the vehicle's code, and even tamper with ECUs, disrupting their proper operations.

3.1.2 Short-Range Wireless Access

This category encompasses vehicle wireless interfaces that operate over short ranges. For this attack surface, we assume that the attacker can place a wireless transmitter in proximity to the target vehicle, taking into account the limitations of the considered technology.

- **Passive Anti-Theft System (PATS):** Most modern automobiles are equipped with a Passive Anti-Theft System. It consists of an RFID-based vehicle immobilizer composed of two elements: an RFID tag in the ignition key and a reader in the vehicle's steering column. At ignition time, in less than a second, the onboard computer sends out a radio frequency signal that is picked up by the transponder in the key, which returns a unique RF signal to the sender. This system prevents the vehicle from operating with the incorrect key. In fact, if the RF signal received by the onboard computer is incorrect, certain vehicle components (e.g., the fuel pump) remain disabled.

  Although possible remote attacks are plausible, the only data processed by the onboard computer is the identification code emitted by the transponder, making it hard to imagine a possible vulnerability that would allow remote code execution. The only plausible attack could be a denial of service (DoS) attack, which could prevent the automobile from starting even with the right ignition key or, in case of vulnerabilities, a possible exploitation vector for vehicle theft.

- **Tire Pressure Monitoring System (TPMS):** The Tire Pressure Monitoring System is a mandatory device responsible for alerting drivers to under or overinflated tires by transmitting real-time data to a specific ECU. The sensor module communicates its data via a wireless radio frequency transmitter. The receiving tire pressure control unit, in turn, analyzes the data and can send results or commands to the central car computer over the Controller Area Network (CAN) to trigger a warning message on the cluster instruments (CI).

Researchers in [45] have pointed out that TPMS equipped on vehicles can raise location privacy risks. In fact, they were able to easily reverse the proprietary protocol, revealing the presence of a static 32-bit identifier potentially used to track vehicle movement. In addition, although the attack surface is rather small, researchers have shown that it is possible, in some cases, to remotely disable the ECU responsible for processing data from TPMS. However, most of the time, the TPMS is not connected to the in-vehicle network and is responsible only for managing the ignition light on the IC.

- **Remote Keyless Entry / Start (RKE):** Today, the vast majority of automobiles are capable of allowing keyless entry. It is possible using a key fob and an RKE system, which remotely opens doors, deactivates alarms, and, in some cases, starts the ignition. Key fobs contain a radio transmitter that communicates with a specific in-vehicle ECU. The radio transmitter sends encrypted data containing identifying information, and the responsible ECU determines if the key is valid or not. As in a previous interface, it is possible to cause a denial of service (DoS), not allowing the vehicle to be locked/unlocked.

  In addition, researchers in [46] have demonstrated that they were able to attack the RKE of Volkswagen Group, gaining unauthorized access to the vehicle. They exploited the cryptographic weaknesses of the Hitag2 rolling code system in the RKE context, recovering the cryptographic key with fewer than eight rolling codes and a few minutes of computation.

  With regards to remote code execution, even though the ECU responsible for RKE must have some firmware to handle reading RF signals, encryption/decryption code, some logic to identify data from the key fob, and to be programmed for additional/replacement key fobs, the attack surface is quite small.

- **Bluetooth:** Bluetooth is the de facto standard for supporting hands-free calling in automobiles, and the overwhelming majority of vehicles are equipped with this standard. Ordinarily, pairing a Bluetooth device requires user interaction, but an unsolicited pairing vulnerability is not out of the realm of possibility. Unlike the previously analyzed interfaces, the Bluetooth stack is quite large, resulting in a considerable attack surface. With this interface, two attack scenarios are possible: one that requires a paired phone and another that involves an unpaired phone.

  Researchers in [43] have demonstrated that the software responsible for handling Bluetooth functionality in the telematics box (telematic ECU) contains a copy of a popular embedded implementation of the Bluetooth protocol stack, while the interface of the latter and the rest of the telematics system was custom-built. Analyzing the interface, the researchers discovered over 20 unsafe calls to "strcpy", one of which was easily exploitable. Thus, any paired Bluetooth device can exploit it and execute arbitrary code on the telematics unit.

  Another piece of evidence of the exploiting of this entry point is provided by the Keen Security Lab in [47]. The researchers discovered several security findings in Bluetooth and vehicular diagnostic functions in the car, which chained together to wirelessly take control of the AVN unit (In-Vehicle Infotainment) without any user interaction. They were able to inject malicious CAN messages from the AVN unit into the CAN network, causing the vehicle to perform unexpected physical actions. This last attack scenario is the most dangerous, as any attacker can reach this code without the need to carry out a pairing procedure.

- **Wi-Fi:** Some modern automobiles are also equipped with Wi-Fi technology. This

allows them to create an in-vehicle hotspot, providing internet connection to passengers, while also permitting the vehicle to connect to access points at home or service stations (e.g., when cellular connection is not available or for large over-the-air updates). As mentioned in section 2.4.1, Wi-Fi interfaces are included in the telecommunication box (T-Box), which is in charge of wireless communication. As it is connected to the rest of the in-vehicle network, it requires significant attention concerning safety-critical attacks since it has a longer range compared to the aforementioned Bluetooth interface. Additionally, it must be pointed out that Wi-Fi security can be assessed without advanced knowledge of automotive systems. This significantly increases the probability of discovering possible vulnerabilities, given the frequent documentation assessments from the IT field (e.g., CVE-2014–1635) [48].

In [49], Nie et al. discovered that the SSID "Tesla Guest", the default access point set in every Tesla, has a plaintext saved password that allows creating a malicious hotspot, redirecting all traffic from neighboring Tesla vehicles to a well-crafted endpoint. Details of this attack and all its consequences are analyzed in subsection 3.2.1.

Some years before, Miller and Valasek [50], disassembling the "`WifiSvc`" binary of a Jeep Cherokee, discovered that the algorithm used to construct the random password for in-vehicle WiFi (WPA2) is purely a function of the epoch time. But since the Head Unit cannot get the time at the start, it is set by default to "`00:00:00 Jan 1, 2013 GMT`", meaning a few dozen possible passwords are sufficient to brute force every Jeep Cherokee's access points. Exploiting this entry point, the researchers were able to chain other vulnerabilities and build a complex attack that led to controlling the vehicle remotely.

- **DSRC:** As mentioned above, DSRC is a new emerging technology for Intelligent Transportation Systems (ITS). At the moment, there aren't documented in-vehicle attacks that exploit DSRC as an entry point, as has happened with the above interfaces. However, it must be pointed out that vehicle-to-everything is expanding the attack surface, becoming a future security and safety issue. In fact, DSRC, based on IEEE 802.11p, could be a possible vector for the exploitation of vulnerabilities in the telematic box, acting as an entry point to launch more complex in-vehicle attacks. Additionally, it should be noted that it is still possible to affect the security and safety of connected and autonomous vehicles through V2X attacks. Several of these attacks are analyzed in section 3.3, implemented in chapter 6, and executed in chapter 7.

3.1.3 Long-Range Wireless Access

This category encompasses all the wireless interfaces of automobiles that can operate over long ranges. On one hand, this is an advantage as it guarantees ubiquitous vehicle connectivity. On the other hand, it amplifies the impact of a possible attack because there is no constraint regarding the proximity of the attacker. Therefore, for this attack surface, we assume that the attacker can transmit wireless data without any constraint related to the proximity to the target vehicle.

- **Cellular:** Among the long-range wireless access technologies, such as GPS and FM, Cellular is undoubtedly the most interesting technology for attackers. It is individually addressable and is increasingly connected to more in-vehicle services on a daily basis. It is used to provide a broad range of features like safety, diagnostics,

anti-theft, and convenience, creating a relatively broad attack surface exploitable by malicious entities seeking unauthorized access.

Researchers in [43] conducted a case study on an undisclosed model of sedan in which they discovered a buffer overflow vulnerability and a logic flaw in the unit's authentication system. In particular, they found that the random number generator (RNG) used to generate the random challenge is re-initialized every time the telematics unit starts and is seeded with the same constant. The researchers were able to exploit this vulnerability by replaying a correct response, gaining unauthorized access to the vehicle, and transmitting CAN packets via the cellular network.

Another piece of evidence of the exploitation of this entry point comes from Miller and Valasek in [50]. In their case study on an unaltered passenger vehicle, among several vulnerabilities, some of which were mentioned earlier, they discovered the presence of an open port inside the Sprint cellular network that allows D-Bus (IPC mechanism) interaction over IP. By chaining this vulnerability with others discovered in the vehicle under consideration, they were able to control a vehicle located anywhere in the country.

As mentioned before, the analyzed attack surface is extremely important, as it serves as the first stage of a complex attack chain. The examined in-vehicle entry points differ based on the distance the attack can be performed, whether access is unlocked after any user interaction, and the required equipment for the attack execution.

In the next section, famous case studies of remote attacks against unaltered passenger vehicles are described to assess the attack feasibility and related risks for both passenger security and safety.

3.2 In-Vehicle Security Threats

Since the first widespread hacking of cars in 2010 [51], OEMs have gradually turned their attention to security issues and countermeasures, abandoning the previously used "security through obscurity"approach in favour of regulations, standards, and security best practises [52]. Initially, research focused on physical attacks based on injecting modified CAN messages to reprogram ECUs and/or perform unsafe actions on the vehicle without the driver's will. However, this didn't stop OEMs from not taking security measures, arguing that someone with physical access could perform a simpler action such as cutting a brake cable or similar, which would have equally dangerous consequences such as human injury or death. For this reason, various researchers have decided to switch to remote attacks to analyse a more sophisticated but also more dangerous scenario, since it doesn't require physical and close presence and therefore has a higher feasibility.

In the following sections, recent attacks on modern vehicles are analysed to learn about the modus operandi, understand the vulnerability and weakness, and finally list the adopted or proposed countermeasures.

3.2.1 Tesla Hack 2016 — Tencent

The Keen team focuses its research on Tesla, the most famous connected car. In September 2016, the researchers managed to conduct a remote attack on the Tesla Model

S in both Parking and Driving mode [49]. First, the researchers try to exploit the vulnerabilities found in the vehicle browser using a contactless medium. They analyse the remote attack surface to find the most favourable entry point.

- **WiFi**: It could be exploited because the SSID "Tesla Guest" was stored in many customer vehicles to automatically connect to the Tesla Service Center WiFi. The researchers discovered that its password was saved in plain text within the "Qt-CarNetManager" module. By using a malicious HotSpot with the same SSID and password, all browser traffic could be redirected to another domain where the malicious code is hosted.
- **Cellular**: the attack could be carried out done via phishing or users mistyping with a sufficiently crafted domains.

Both entry points lead to remotely exploit found browser vulnerabilities. They achieve arbitrary code execution by exploiting a vulnerability in the `JSArray::sort()` function and CVE-2011–3928 (UAF vulnerability). By chaining these vulnerabilities, they are able to read and write arbitrary addresses, including the JIT (Just In Time) memory address, into which they write shell code and obtain a shell from Tesla CID. Since the shell is provided by browser hacking, it has low privileges, and due to the presence of AppArmor, they cannot gain arbitrary privileges. However, seeing that the Linux kernel version of CID is very old, the researchers exploit a ARM Linux vulnerability CVE-2013–6282 to escape from AppArmor. This vulnerability consists of a missing access checks in `put_user/get_user` kernel API which allows attackers to read or modify the contents of arbitrary kernel memory locations. In fact, the researchers first change the behaviour of the `setresuid()` system call to gain root privileges, and then replace a system call entry in the system call table to call `reset_security_ops()` and disable AppArmor. At this point, they gain full control of CID and focus on sending messages to other ECUs.

As described in section 2.4, different segments of the on-board network are connected through the gateway, and since it is the access point to reach the target ECUs, it is very interesting. The researchers, reversing the "gw-diag" binary used to diagnose the gateway, discover that they can trigger a function `ENABLE_SHELL` through the gateway's port 3500 that wakes up the gateway's protected backdoor on port 23. By reversing the gateway's firmware, they discover that the token is static and hardcoded, and they manage to bypass the backdoor's token check effortlessly. In this way, they obtained a gateway shell.

Because of they want to reprogram Tesla's gateway, they analyse the contents of the gateway's flash memory finding several files including "`booted.img`" and "`release.tgz`", the latter an ECU software package that contains a manifest, ECU softwares and an overall CRC32 checksum. The researchers focus on the gateway firmware and find that a message with `id=0x08` triggers the update process. Specifically, the upgrade function checks the format and checksum of the image file, renames it "`boot.img`", reboots itself, and then renames it "`booted.img`" to prevent boot into it again. It is possible to modify "`boot.img`" to skip some checks, but since only integrity is checked using a CRC32 checksum, the researchers can recalculate the checksum and pass the verification stage. At this point, they modify the gateway firmware with their customised version and package it with the modified manifest in a new "`release.tgz`" to which they append a recomputed CRC32 value. By starting the upgrade process, the researchers are able to run their code permanently on ECUs, specifically loading modified code on the gateway to open a backdoor that allow them, as detailed below, to send any frame on the CAN bus even when the vehicle is driving.

Since the only way from CID to send messages over the CAN bus is through the gateway, the researchers analyse the tasks running on it and find some vulnerabilities. First of all, the gateway forwards all UDP packets from ports 20100/20101 as CAN messages to the CAN bus. Unfortunately, this is a limited channel, and not all ECUs are reachable. In fact, they can reach only the body ECU and spoof some UDP signals to open the trunk at low speed. Due to this limitation, the researchers find another way to send CAN messages to other ECUs and exploit the second vulnerability in the gateway, a diagnostic function `(0x04)` that injects UDP messages from port 3500 to the CAN bus. They circumvent the unavailability of this function in mode by swapping the handler of the latter with that of another always available function `(0x01)`. Although they can control the lights at high speed, they find that some ECUs do not respond in driving mode and disable dangerous functions such as diagnostics while the vehicle is moving at high speed. To solve this problem, they exploit the third vulnerability in the gateway's tasks, which allows blocking some important messages (e.g., vehicle speed) by changing the `target ID` in the structure stored in the firmware, blocking the forwarding process and allowing the previous operations even at high speed. However, this only affects messages that pass through the gateway. Then the researchers try to exploit a weakness in Unified Diagnostic Services (UDS), a communication protocol used for diagnostic purposes to execute functions on ECUs. They capture CAN messages, get `CAN ID` to send UDS requests and receive UDS responses, and find out the presence of UDS Security Access Service, which is used to unlock ECU. The security access service is used to restrict access for security or safety reasons and is based on a Seed-Key mechanism. After several tests with different ID, the researchers find that ECU sends a fixed seed in the Security Access Service flow, which leads to a fixed key that the updater uses to gain access to ECU. At this point, to affect the real world the researchers decide to target the Electronic Stability Program (ESP). After setting up a diagnostic program session, they can disable the power-assisted steering and power-assisted brakes, causing potential safety issues for drivers.

3.2.2 Tesla Hack 2017 — Tencent

In the previously examined remote attack case study conducted in 2016 [49], the Keen Team exploits two vulnerabilities found in WebKit to execute arbitrary code within the browser's context. Although Tesla developers quickly patch them, other unknown vulnerabilities remain exploitable for a long time due to the unchanged WebKit version. This allows the researchers of the Keen Team to carry out a new case study of a remote attack against Tesla [53], exploiting a chain of multiple zero-days of different in-vehicle components, enabling them to remotely control the car.

The researchers find and exploit the first zero-day vulnerability, which alone makes it possible to achieve arbitrary code execution. Similar to the aforementioned case study, the exploited vulnerability is a Use After Free vulnerability, specifically related to the `SVGTransformList` element of WebKit. During the `initialize()` and `clear()` methods of `SVGTransformList`, the `WTF::fastFree()` function is executed, freeing the buffer of the vector that contains instances of the `WebCore::SVGTransform` class. The security flaw lies in the fact that, although the `SVGTransform` is freed, the reference to the `WebCore::SVGMatrix` structure, which is part of `WebCore::SVGTransform`, remains available, allowing access to the just-freed memory. The researchers discover that the array `m_transform` of `WebCore::SVGMatrix` can be directly read and written to using the get and set methods, which means they can read from or write to any memory location occupied by `WebCore::SVGMatrix`. First of all, using `m_allocBase` of JSArray the researchers

tamper the ArrayStorage structure and, triggering `JSArray::unshiftCount()`, they can free any address. At this point, they focus on reading and writing arbitrary addresses and in particular, executing arbitrary code. To read arbitrary addresses the researchers put a tampered ArrayStorage structure in the memory just freed and, thanks to the big length of fake `WTF::StringImpl`, they can read the whole memory. To write arbitrary addresses they use the same tampered length approach used in the read. They leaked and freed a `Uint32Array` structure and, using a new tampered `Uint32Array` header they can write the whole memory. Finally, to achieve their goal they leak the just-in-time (JIT) memory address from JSCell, as in the previously analyzed case study, and they can write the 'shellcode' to this special address achieving code execution and obtaining a shell of Tesla CID again.

After gaining a CID shell, the researchers need to obtain root privileges. Tesla developers have indeed addressed nearly all well-known vulnerabilities and upgraded the kernel. Therefore, in order to gain arbitrary privileges, it became necessary to discover a new 0-day vulnerability. They discover such a vulnerability (CVE-2017–6261) in the Tegra nvmap kernel module and successfully exploit it to achieve their goal. With AppArmor enabled, QtCarBrowser has limited access. However, it can access to `/dev/nvmap` which is a user interface that permits userland process to send commands via `ioctl()` function. If the `ioctl` command is `NVMAP_IOC_PIN_MULT`, the function `nvmap_pin_ids()` will be executed and, after a certain condition, it triggers the vulnerability. In particular, the researchers discover that through this function, is possible to decrease by 1 arbitrary address. In detail, using an invalid `nvmap_handle` structure, via `nvmap_handle_putfunction` is invoked `atomic_dec_return()`, the one that subtracts 1 at reference count parameter. Since the pointer to this invalid `nvmap_handle` structure is provided by a process that runs in user mode, the researchers can decrease any integer data by one in kernel memory. They exploit this vulnerability to write arbitrary data to arbitrary addresses decreasing the address of a syscall `'accept4'` by `0x10` pointing to an address of a ROP gadget. In this way, calling the syscall with proper arguments, they can disable AppArmor using `APPARMOR_COMPLAIN profile_mode` and patch `setresuid()` allowing any process to elevate privileges. At this point, thanks to the exploit of this 0-day vulnerability, they get full control of CID.

As in the previously analyzed case study, the CID can use `port 3500` of the gateway to send it commands like `'Reboot for update'` etc. On the gateway, there is a script that, among the various tasks, provided a method to upload and rename files but it is forbidden the upload of `'boot.img'` file or rename it using the latter. The researchers discover that `boot.img` is booted directly by the bootloader without verifying if the image is correct, so it is possible to modify the image directly through physical access to the SD card. However, due to this being a less interesting case compared to a remote attack, the researchers choose a different strategy. They find that the function used by the gateway during the update procedure to rename the image from the initial name to `'boot.img'`, includes a signature verification in which the gateway checks if the file is a legal image released by Tesla, refusing the rename if verification fails. To sign the `'.img'`, Tesla uses EdDSA with Curve25519 (Ed25519) as signing algorithm and SHA512 as hashing algorithm. They note that constants and keys are carefully picked, so researchers prefer not to run into cryptographic attacks. They find out the above-mentioned rename function applied no thread lock so there could be a TOCTOU attack by replacing a correct image with a malformed one between the signature verification and the renaming of the file. However, after more reverse engineering jobs, they found out Tesla might be using FatFS r0.09 and that the implementation of `fatfs_rename` strip leading spaces and dots. The researchers are able to bypass Tesla's check thanks to the transformation

of `'x20boot.img'` into `'boot.img'` by FatFs. This allows them to bypass code signing protection and execute their customized code into gateway and ECUs as seen in the previous year's case study.

To prove they are able to affect components directly connected to electromechanical components, the researchers decide to target the Tesla Easter Egg, a special functionality that makes use of different body control ECUs to tailor different shows for each Tesla owner. After a reverse engineering phase, the researchers find out the easter egg has three different stages. First of all, there is the start-up signal launched by CID that is triggered with an unusual combination of buttons. This signal initiates the requirements check, verifying if parking mode is enabled and ensuring that no other show is currently running. Then, the control is passed to the Body Controller Module (BCM), also known as the Central Body Controller (BCCEN). The BCM is responsible for verifying the reception of the key fob signal and ensuring that all passengers have exited the car. In the final phase, the CID executes the `startShow()` function which communicates to BCFRONT to broadcasts commands to synchronize the motions of electromechanical components via body control ECUs. The communication between the BCCEN and CID is done through the gateway using `port 20100` which permits the conversion from UDP to CAN and vice versa. At this point, the researchers try to make their own easter egg to play with electromechanical components. They first patch the CID to bypass certain checks, thereby allowing it to run without requiring parking mode. Subsequently, due to the absence of a validation mechanism, they gain the ability to modify the firmware of ECUs. In fact, the researchers discover that, following the initial signature check at the gateway, which they bypass using the identified flaw, the ECUs lack a secure boot mechanism. As a result, the firmware's signature of the ECUs is not verified, and only an integrity check via CRC checksum is performed. Finally, they patch BCCEN to skip the required key fob signal and patch BCFRONT to customize the sequence of action due to its master role in the coordination of body components. So exploiting these, the researchers are able to control all the body ECUs of the vehicle without any limitations. This could potentially lead to serious safety issues when considering the possibility of activating this customized Easter egg even in drive mode. It would permit attackers to manipulate the direction indicators, turn off lights at will, or open the doors at high speed, posing a significant risk of human injury or even death.

3.2.3 Summary of Attacks and Common Weaknesses

Among the numerous case studies conducted on unaltered passenger vehicles worldwide by several researchers, including the attack analyzed above, a common thread emerges. These studies center on a remote approach, presenting heightened challenges and emphasizing the perils due to the absence of limitations associated with an attacker's physical presence, coupled with a high likelihood of success owing to the expansive array of wireless interfaces.

Initially, researchers concentrate on exploiting a vulnerable wireless entry point, leveraging one or more vulnerabilities to gain remote access to the in-vehicle network. This access allows them to intercept in-vehicle traffic and meticulously reverse engineer all packets in transit to uncover additional vulnerabilities. Subsequently, researchers inject malicious messages into the in-vehicle network, attempting to compromise ECUs. While executing code on the vehicle is concerning, the most perilous phase arises when they can begin transmitting messages to the vehicle's critical ECUs. In the majority of conducted case studies, breaching the safety-critical network segment required supplementary efforts, mandating the attacker to reprogram the gateway ECU. Specifically, through the

exploitation of multiple 0-day vulnerabilities, researchers acquire arbitrary privileges, enabling the execution of arbitrary code and compromising other interconnected ECUs within the network. It is notable that the components responsible for external communication are typically segregated from the network housing safety-critical ECUs. The most challenging aspect of remote attacks, according to the findings of the analyzed case study, appears to be reaching this segment. Subsequently, upon gaining control of the safety-critical ECU, attempts are made to remotely manipulate the vehicle through the interconnected cyber-physical systems.

Several common weaknesses are discernible in these renowned case studies. These weaknesses have facilitated the researchers' objectives, particularly in constructing a viable attack sequence from the remote entry point to the cyber-physical systems, thus enabling remote control of the vehicle.

- Absence of a Secure Boot Mechanism: One of the most dangerous weaknesses lies in the absence of a mechanism that verifies code prior to execution. Specifically, secure boot enables cryptographic verification of code integrity and authenticity through a trusted chain anchored by a key in a write-protected section of the computer. This process involves the BIOS verifying the bootloader, which in turn verifies the kernel and software image.

- Absent Network Segregation: Another prevalent weakness in older case studies is the lack of a gateway equipped with advanced security measures (Security Gateway) such as sophisticated key management schemes, firewalls, and intrusion detection systems, as mentioned in section 2.4.1. In these vulnerable cases, the gateway merely serves as context-aware routing, checking message validity, and permitting all valid messages to reach their designated destinations.

- Presence of Several 0-day Vulnerabilities: This weakness underscores the absence of a robust threat analysis and risk assessment procedure (TARA), as outlined in the ISO/SAE 21434 standard analyzed in section 3.4. It is noteworthy that vulnerabilities are present in both OEMs and TIER1 products, emphasizing the importance of implementing TARA throughout the supply chain and product lifecycle.

- Absence of Message Signing: While this secure mechanism may be rendered ineffectual in the presence of vulnerable ECUs facilitating privilege escalation and subsequent malicious secure communication, it still serves as an important countermeasure against in-vehicle attacks. Message signing ensures confidentiality, integrity, and authentication, complicating the attacker's efforts by impeding the reverse engineering of in-vehicle encrypted traffic, tampering with in-transit packets, or forging malicious messages.

3.3 V2X Security Threats

After analyzing a real-world case study involving cyber-physical attacks against unaltered vehicles, it is worth highlighting the possibility of safety-critical threats also occurring in attacks confined to vehicular ad-hoc networks. Notably, VANET attacks have a greater likelihood of success compared to in-vehicle cyber-physical attacks, as they require less time for reverse engineering, do not necessitate a complex chain of vulnerabilities, and can be used to target multiple vehicles from different companies.

As explained in section 2.3, the primary purpose of V2X communication is to enhance road safety, traffic efficiency, and infotainment services through the exchange of information between vehicles and between vehicles and road infrastructure. This cooperation among vehicles in sharing information leads to the implementation of several Intelligent Transportation System (ITS) applications aimed at addressing the aforementioned issues and introducing a higher level of drive automation. Consequently, this results in the inclusion of on-board logic that can physically operate the vehicle based on received V2X messages (e.g., changing lanes or initiating an emergency brake). This underscores the need for security requirements in the communication systems to prevent, or at the very least, limit malicious attacks that could easily alter the communication, posing security and safety hazards.

Security requirements for V2X communication encompass Authentication and Authorization, Availability, Confidentiality, Integrity, and Privacy. However, as emphasized in [54], messages from most common ITS services are broadcasted to any possible nearby receivers without concerns about confidentiality. Confidentiality is only relevant for a few of the various ITS applications, such as those requiring unicast communication (e.g., media downloads, vehicle software updates, etc.), where messages are encrypted and only the intended recipient can decrypt them. The analyzed attacks are classified based on the previously highlighted security requirements. However, it is worth noting that certain attacks may compromise multiple of these security requirements simultaneously.

3.3.1 Authentication Threats

The authentication property guarantees that only legitimate ITS entities can access VANET and engage in V2X communications. As detailed in section 4.2, every ITS entity can authenticate the sender of V2X communication by verifying if the certificate (Authorization Ticket) used to sign messages is valid. In V2X services, authentication includes user authentication and message authentication. The former guarantees that the entity is legitimate, while the latter ensures that the message is authentic and has not been tampered with.

Attacks in this category can falsify the authentication process, causing several issues in VANET.

Sybil Attack

In a Sybil attack, the attacker is able to forge multiple fake identities to fulfill its intentions and disrupt VANET services. The disruption of services is caused by nodes' perception of the truthfulness of these fake identities. In this way, the attacker can use several forged or stolen authentication tickets (certificates used to exchange V2X messages in VANET) to enhance their reputation to the detriment of legitimate nodes in the case of area hazard warnings. Additionally, the malicious node can exploit the obtained illegitimate certificates to disseminate periodic messages (i.e., Cooperative Awareness Messages), creating a scenario in which a single malicious node can pretend to be several at the same time. The attacker, in fact, by forging different packets with different identities, can claim to be in different positions, thereby creating a confusing traffic situation in VANET (e.g., fake congestion or traffic jams).

Impersonation Attack

The Impersonation Attack can be considered a specialization of the aforementioned Sybil Attack [55]. It aims to use a stolen identity for the purpose of executing malicious activities. In this case, the attacker, using a stolen pseudonym certificate from a compromised legitimate vehicle, changes its identity and presents itself as a different vehicle. Similar to the Sybil attack, this poisons the overall vehicular network architecture and is particularly treacherous if a malicious node is involved in an accident.

Free-Riding Attack

The Free-Riding attack aims to exploit a cooperative network without contributing to cooperation. In this attack, the malicious node selfishly listens to all communications between other nodes without sending its periodic information (i.e., Cooperative Awareness Messages). This attack is only possible in cases where VANET lacks authentication mechanisms (e.g., Certificate-based Authentication).

3.3.2 Availability Threats

Availability guarantees that a network and its services are always functional and accessible to all nodes. However, there are several attacks that aim to disrupt this requirements. Since Intelligent Transportation Systems aim to enhance road safety, any interruption, even for a short duration, can be perilous for passengers and road users.

Denial-of-Service Attack

In a DoS attack, the attacker aims to prevent other nodes from using the network and the deployed services. The attacker can easily compromise the V2X network by sending valid messages at a higher frequency than the system can handle (e.g., beyond the frequency set by the ETSI ITS standard). This results in network unavailability, consequently rendering VANET services unusable for legitimate users. The maximum impact of this attack is achieved when combined with others. Kamel et al. in [56] present a combination of the DoS attack with a Sybil attack, which amplifies the effects and increases the effort required for detection.

Jamming Attack

The jamming attack is a specific implementation of a DoS attack. In this attack, the attacker aims to disrupt communication by interfering with the channel using the same radio frequency but with a strong signal. This tactic is technology-agnostic, relying solely on a particular frequency to disrupt communication. Moreover, it focuses on disrupting communication at a lower layer, making prevention mechanisms ineffective. While this is a relatively straightforward attack common to any wireless communication, in the case of time-critical V2X applications, the introduced delay can cause severe harm to VANET users. Researchers in [57] present a jamming attack that disrupts a platoon of vehicles communicating using the European standard (i.e., ETSI ITS-G5).

Black Hole Attack

While DoS and jamming attacks shut down the network by saturating the channel or degrading wireless transmission, the Black Hole attack achieves unavailability by not cooperating in V2X message routing. In VANET, to enable the dissemination of area hazard warnings (e.g., decentralized environmental notification for the European standard), a forwarding message algorithm is necessary. The attacker exploits this algorithm by announcing itself as having the shortest path to the destination area. Consequently, the attacker becomes the sole node responsible for forwarding warning messages to a specific area and can decide whether to complete the forwarding procedure or not. Since their aim is to disrupt VANET services, they may choose to halt the forwarding procedure, preventing the network from being informed about the area hazard warnings under consideration. A variant of the Black Hole attack is the Gray Hole attack, in which the attacker doesn't always choose to stop the forwarding procedure.

3.3.3 Confidentiality Threats

Threats to confidentiality occur when confidential data is disclosed to unauthorized entities. As mentioned earlier, in ITS applications, the presence of confidentiality requirements is not common because most messages are broadcast to any possible receiver. However, there are a few applications (e.g., Traffic information and recommended itineraries, Media downloading, Vehicle software/data provisioning and updates, etc.) that require unicast addressing and almost always involve confidentiality requirements [54].

Eavesdropping Attack

Eavesdropping is a common threat in wireless communication, as it involves passive listening to the wireless link that enables communication between two legitimate users. Since ITS broadcast messages are transmitted to all network participants, eavesdropping is not particularly useful for an eavesdropper [54]. However, this attack can be exploited to construct a pseudo-profile of an ITS station (ITS-S) by inspecting which services are habitually used, in what situations, etc., in order to leverage this information for more sophisticated attacks.

Man in the Middle Attack

A Man in the Middle attack aims to intercept communication between two legitimate network nodes secretly. In its "Passive Mode", this attack is equivalent to the previously analyzed eavesdropping attack, as it merely involves eavesdropping on traffic. However, there is also an "Active Mode" where the attacker aims to alter the communication by injecting counterfeit information, slowing down messages, or completely dropping them. Given its potential consequences in VANET, it can be considered a generalization of specific attacks like "Black Hole" or "Message Modification" [58]. For this reason, the Man in the Middle attack can be considered an attack that threatens not only confidentiality but also authentication, availability, and integrity.

3.3.4 Integrity Threats

Integrity aims to guarantee the exactness and reliability of information over its entire transmission. This principle ensures the detection of manipulation and/or corruption of

information. In particular, the goal is not to entirely prevent the integrity from being compromised (i.e., tamper-proof), but rather to guarantee evidence in case of compromise (i.e., tamper-evident). Like the previous principles, if this one is compromised, it could have a significant impact on VANET services, as will become evident in the following attacks.

Message Modification Attack

Message Modification, also known as Message Falsification, aims to alter the original content of messages to gain advantages or provoke hazardous events. For example, when used for area hazard warnings, an attacker may change the message information, creating a different situation that degrades the network's reliability. The attacker could easily change the message warning code (e.g., from Weather Condition Warning to Traffic Jam Ahead), triggering rerouting procedures for nearby vehicles and creating empty roads.

Replay Attack

Replay attacks threaten integrity as they aim to retransmit an already transmitted valid message, achieving the same benefits as the genuine sender or, as in previous attacks, causing network confusion. The attack could be location-based or time-based [59]. In a location-based attack, the intention is to retransmit the message as quickly as possible in another location without altering the timing. To carry out this attack, the absence of location references in the replied messages is necessary. Similarly, the time-based attack plans to rebroadcast a valid message at the same location but with a delayed timing. Again, the absence of time references or expiration times in message validity checks is necessary to execute this attack.

Illusion Attack

In an Illusion attack, the attacker can control and mislead its vehicle's sensors, thus broadcasting fake information about unreal events. This is equivalent to injecting fabricated information into the network to fulfill self-interest or malicious intentions. As in previous attacks (e.g., Sybil, Impersonation, and Message Modification), this attack can lead to a broad range of events, thereby causing various hazardous VANET conditions, some of which jeopardize the safety of passengers and road users.

3.3.5 Privacy Threats

Privacy aims to guarantee the absence of any threats that could expose the privacy of VANET users. Since periodically broadcasted messages (CAMs) contain significant status information related to the sending ITS-S, it is necessary to ensure that, although this information is shared, it cannot be linked to any individual. Details on how the leaking of personally identifying information by the CAM services is prevented are described in chapter 4.

Location Tracking Attack

The location tracking attack aims to follow the track of a vehicle over a period of time. As mentioned earlier, this attack exploits the broadcasted ITS-S information, which is

essential for the proper operation of VANET applications described in section 2.3 (e.g., FCW, EEBL, etc.). This attack could be used to gather information that is useful for a more sophisticated attack aimed at targeting a specific ITS-S.

Identity Revealing Attack

In the Identity Revealing attack, the attacker aims to disclose sensitive information about the identity of the owner of the ITS-S in question. Unlike the previous attack, which aims only to marshal ITS-S information, this attack involves user identity information. Researchers in [60] describe how they have been capable of conducting an identity revealing attack by exploiting a developed malicious app which, acting as a Trojan Horse in an Android In-Vehicle Infotainment system, allowed the exfiltration of sensitive information through an opened backdoor.

In chapter 6, some of these attacks will be designed, implemented, and executed in a real-world scenario based on the European standard ETSI ITS-G5.

3.4 ISO/SAE 21434

The continuous demand for on-board electronic and software components to enhance security, safety, and convenience features has indirectly rendered vehicles extremely vulnerable to cyber attacks. As detailed in the preceding section, numerous case studies have aimed to raise awareness about the potential remote compromise of in-vehicle safety-critical ECUs. Consequently, a persistent need for a unified cybersecurity standard in the automotive industry prompted the International Organization for Standardization (ISO) and the Society of Automotive Engineers (SAE) to develop ISO/SAE 21434:2021 "Road Vehicles — Cybersecurity Engineering".

ISO/SAE 21434 delineates engineering requisites for cybersecurity risk management of electrical and electronic (E/E) systems in road vehicles. It aims to furnish guidelines for cultivating cybersecurity culture, establishing policies and processes, and managing risks for both original equipment manufacturers (OEMs) and their suppliers (e.g., Tier 1, Tier 2). This standard does not address specific technical solutions but rather aims to provide a generic framework for managing cybersecurity risks in all types of road vehicles, especially connected and autonomous vehicles (CAVs).

Comprising 15 sections and 8 annexes, the document covers all phases of the engineering process. Sections from the fifth onward, termed "clauses", define requirements (RQ), recommendations (RC), and work products (WP). While all clauses within this document hold immense importance and interest, given the focus of this dissertation, only clause No. 15, **threat analysis and risk assessment (TARA)**, is discussed.

3.4.1 Threat Analysis and Risk Assessment (TARA)

Clause 15 describes a modular approach for assessing cybersecurity risks invoked in activities described in other clauses. This clause introduces a method known as threat analysis and risk assessment (TARA), aiding in evaluating the potential impact of a threat scenario on road users. Employing a modular approach, TARA is executed from the perspective of affected road users and can be initiated at any stage of an item or component's lifecycle. TARA comprises 7 modules:

1. **Asset Identification**: This module aims to identify assets, their cybersecurity properties, and potential damage scenarios. It involves considering the specific item and its cybersecurity specifications as defined in the design phase. Damage scenarios encompass the relationship between the item's proper functionality and potential adverse consequences for road users and relevant assets. For instance, an asset might relate to data communication for the braking function, focusing on ensuring integrity. In this scenario, a potential damage scenario could involve a rear-end collision due to unintended full braking at high speed.

2. **Threat Scenario Identification**: This module aims to identify threat scenarios for a specific item based on the targeted asset, compromised cybersecurity properties, and potential consequences. The method for identifying threat scenarios can involve group discussions and/or systematic approaches (e.g., frameworks such as EVITA, TVRA, and STRIDE). Notably, a threat scenario can lead to multiple damage scenarios and vice versa. For example, a spoofing attack against CAN messages used for braking ECU input could compromise the integrity of these messages, consequently affecting the braking function.

3. **Impact Rating**: This module's objective is to determine the impact rating of damage scenarios. Unlike previous stages, this phase necessitates pre-existing damage scenarios [WP-15-01]. It involves evaluating these scenarios based on potential consequences for road users across categories such as safety, financial, operational, and privacy (S, F, O, P) respectively. Furthermore, it provides a methodology for assessing the impact rating of a damage scenario for each impact category, assigning values of severe, major, moderate, or negligible. Notably, the safety damage criteria outlined in Table F.1 of the ISO/SAE document are derived from **ISO26262–3:2018**, which stands as the international standard for functional safety in the electrical and/or electronic systems of production automobiles.

4. **Attack Path Analysis**: This module aims to identify the attack path leading to threat scenarios. Completion of the threat scenario identification [WP-15-03] is a prerequisite for this module, implying that the related information must already be available. Additionally, this analysis can draw support from information such as potential weaknesses identified during continuous cybersecurity event evaluations or discovered throughout the product development phase. It also considers the outcomes of vulnerability analyses and previously identified attack paths, if accessible. The attack path analysis can follow two approaches:

   (a) Top-Down Approach: This method deduces the attack path by analyzing various ways in which a threat scenario could manifest.

   (b) Bottom-Up Approach: This approach aims to construct the attack path based on identified cybersecurity vulnerabilities.

   An attack path must be linked to the potential threat scenarios it could trigger. [WP-15-05]

   For instance, using a previously discussed threat scenario allows the evaluation of the associated attack path.

   - Threat Scenario: A spoofing attack against CAN messages used to control the braking ECU leads to the loss of integrity of the CAN messages and consequently, compromises the integrity of the braking function.
   - Attack Path:

- The Telematic Control Unit (TCU) is compromised via the cellular interface.
- The Gateway Electronic Control Unit (ECU) is compromised through CAN communication originating from the already compromised TCU.
- Finally, the Gateway ECU transmits malicious braking signals, resulting in unintended braking.

5. **Attack Feasibility Rating**: This module aims to assess the ease with which attack paths can be exploited. Its presence is contingent upon the existence of attack paths. Furthermore, to enhance the evaluation, architectural design and vulnerability analyses are considered. Therefore, each attack path requires an evaluation of the attack feasibility rating, graded inversely proportional to the effort needed to execute the attack (high, medium, low, very low). This rating should be defined based on one of the following approaches:

   (a) Attack Potential-Based Approach

   (b) CVSS-Based Approach

   (c) Attack Vector-Based Approach

   The selection of the appropriate approach depends on the phase in the lifecycle and available information.

   When employing an attack potential-based approach, the determination of the attack feasibility rating should follow core attack potential factors derived from ISO/IEC 18045 (Information security, cybersecurity, and privacy protection). These factors encompass:

   - Elapsed Time: This factor include the time required to identify a vulnerability, develop, and ultimately apply an exploit. The rating is contingent upon the attacker's level of knowledge.
   - Specialist Expertise: This factor relates to the capabilities of the attacker, contingent upon their skills and experience. Table G.2 establishes four rating categories: laymen, proficient, expert, and multiple experts, encompassing possible attacker skill levels.
   - Knowledge of the Item or Component: This factor pertains to the volume of information gathered by the attacker about the item or component. Table G.3 establishes four rating categories: public, restricted, confidential, and strictly confidential information to assess the requisite knowledge grade.
   - Window of Opportunity: This parameter relates to the access conditions necessary to execute an attack, considering access types (logical and physical) and access durations (unlimited and limited). Table G.4 establishes four rating categories: unlimited, easy, moderate, difficult to gauge the window's breadth. For instance, in V2X communication technology, an unlimited window of opportunity might exist due to the high availability of ITS-S via public/untrusted networks without temporal or physical proximity limitations.
   - Equipment: This parameter pertains to the necessary equipment required to identify vulnerabilities and execute the attack. Table G.5 establishes four rating categories: standard, specialized, bespoke, and multiple bespoke, encompassing potential equipment requirements for various attack scenarios.

When employing a Common Vulnerability Scoring System (CVSS)-based approach, the determination of the attack feasibility rating should rely on exploitability metrics such as attack vector, attack complexity, privileges required, and user interaction.

Conversely, a vector-based approach determines the feasibility rating by evaluating the principal attack vector of the considered attack path. [WP-15-06]

6. **Risk Value Determination**: This module aims to ascertain the risk values associated with threat scenarios by evaluating threat scenarios, impact ratings along with their associated impact categories, and attack feasibility ratings. Consequently, this module should be executed when the following are available: [WP-15-03], [WP-15-04], and [WP-15-06]. The determination of risk values should be carried out for each threat scenario. Particularly, if a threat scenario corresponds to multiple damage scenarios and/or if the latter encompasses multiple impact categories, separate risk values should be determined. Moreover, if a threat scenario involves multiple attack paths, the associated attack feasibility ratings can be aggregated appropriately for utilization. The assigned risk values should fall within the range of 1 to 5, where 1 denotes minimal risk. These values are determined using risk matrices or risk formulas. [WP-15-07]

7. **Risk Treatment Decision**: This final module aims to select appropriate risk treatment options for specific threat scenarios. It necessitates the availability of pre-existing threat scenarios and their associated risk values, i.e., [WP-15-03] and [WP-15-07]. Optionally, it may consider cybersecurity specifications, previous risk treatment decisions pertaining to the same or similar items or components, impact ratings and their related categories, attack paths, and attack feasibility ratings if available. For each threat scenario, upon evaluating its risk values, one or more risk treatment options need to be determined from the following choices: avoiding the risk, reducing the risk, sharing the risk, and retaining the risk. Avoiding the risk involves eliminating the source of the risk by relinquishing the activity that gives rise to it. Sharing the risk entails utilizing contracts or insurance mechanisms

Chapter 4

ETSI ITS-G5

4.1 Standard Background

Inter-vehicle communication is a cornerstone of Intelligent Transportation Systems (ITS), also known as Cooperative ITS (C-ITS). C-ITS standards are essential to achieve interoperability among vehicles of different manufacturers and roadside infrastructure. Among the major standards for vehicular communication (IEEE 1609, ETSI ITS-G5, and C-V2X) as described in section 2.2, this thesis focuses on the European standard ETSI ITS-G5.

ETSI ITS-G5 was defined starting in 2007, taking into account its American counterpart, the IEEE 1609 standard (WAVE). In fact, ITS-G5 inherits from WAVE IEEE 802.11p as the access technology, which is a modified version of the IEEE 802.11 standard carried out by the WAVE project to fulfill the stringent constraints of vehicular communication.

4.1.1 ITS-G5 Architecture

The ITS station reference architecture, as specified in [61], follows the principles of the OSI model for layered communication protocols, which is extended to include ITS applications. As shown in Figure 4.1, the access layer specification is outside of ETSI's specifications since it relies on IEEE 802.11 as its kernel. Moving up toward the OSI layers over IEEE's access layer, GeoNetworking [62], Basic Transport Protocol (BTP) [63], and Facilities layer are located. According to ETSI terminology, the Application layer refers to a vehicle feature that operates using shared C-ITS information.

Additionally, the ITS-G5 architecture includes two cross-layers, "management" and "security". The former is responsible for the configuration of an ITS-S, the exchange of information between the different layers, and other tasks. The latter, on the other hand, enforces specific security and privacy services at different layers, such as securing messages, identity and security credential management, and aspects related to security platforms (firewalls, security gateways, tamper-proof hardware) [64].

4.1.2 Applications and Facility Layers

While it may not be valid for a few specific services mentioned in section 2.3, the majority of ITS applications utilize one of the following communication strategies:

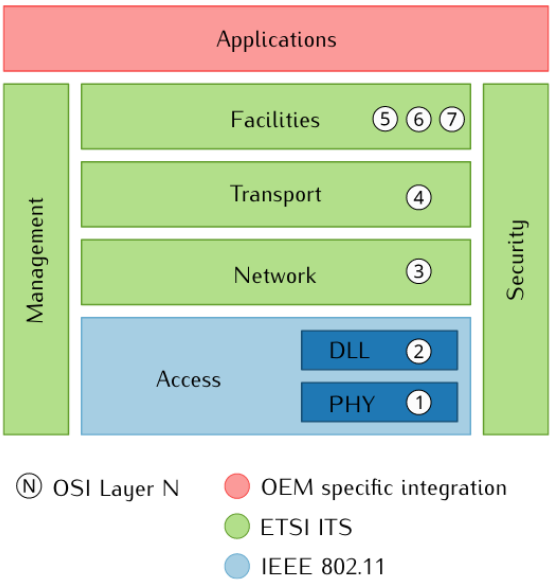

Figure 4.1. Architecture of an ETSI ITS Station.

- **Periodic status exchange.** These types of messages are broadcast periodically by ITS stations and contain information about the status of the specific station sending it. The content of these periodic messages includes information about the station's identifier, location, elevation, speed, heading, etc.

- **Asynchronous notifications.** These kinds of messages are sent asynchronously to inform about a specific situation. They usually carry safety-critical information about hazardous events, so reliable delivery is a key requirement.

According to European regulation [65], a vehicle capable of C-ITS communication shall execute the Cooperative Awareness (CA) and Decentralized Environmental Notification (DEN) service.

Cooperative Awareness Service

The CA basic service is an application support facility provided by the facilities layer, and it is used to generate, manage, and process Cooperative Awareness Messages (CAM). CAMs correspond to periodic status exchange communication strategies and are required to be transmitted periodically, with a maximum period of one second. This unlocks a presence perception based exclusively on wireless communication, without relying on line-of-sight. By receiving CAM messages, the ITS station is aware of other nearby stations as well as their positions, speed, and relevant information. An ITS-S that receives a CAM is expected to evaluate if the information in the message is relevant for the station at hand. This mainly depends on the coordinates and heading since, if the sender is a following vehicle that is going to perform an emergency brake, this information may not be interesting. Depending on the ITS-S type, ETSI standards specify a set of information that every station is allowed to include in their CA messages. In particular, ETSI defines in [66] a so-called "ITS Station Profile" that aims to label a specific ITS-S with a specific profile defining the mandatory and optional values that every ITS-S can use in CAM construction. For example, the "emergencyVehicle" profile, associated with an emergency

vehicle, will allow the insertion of values such as “lightBarInUse” or “sirenInUse” in CAM messages.

According to ETSI Technical Specification [67], the CAM PDU is composed of a common ITS PDU header and a body. The header includes information about the message, e.g., version, messageId, and generationTime. The body contains service information about the ITS station that has sent the message, e.g., ITS-S identifier, type of ITS-S, reference position composed of latitude, longitude, altitude, heading, and other parameters such as speed, yawRate, vehicle length, etc. The body is divided into multiple containers: basic, high frequency, and low frequency. The basic container includes basic information, the high-frequency container contains highly dynamic information, and the low-frequency container contains information specific to the role of the vehicle related to the originating ITS-S within road traffic. There is also a special vehicle container reserved for special vehicles, e.g., police vehicles, special transport, etc.

In addition to the CAM format specification, ETSI standards also define guidelines for the generation of these periodic messages. In [67], the standard specifies rules for calculating the maximum CAM rate and the possible CAM sizes. The generation algorithm evaluates changes in position, changes in direction (heading), and changes in speed to determine whether to trigger the generation of a new CAM. ETSI standard [67] defines the thresholds evaluated by the CAM generation algorithm:

- A change in position by more than 4m
- A change of direction of equal to or more than +/- 4°
- A change of speed equal to or greater than 0.5m/sec

If none of these conditions are fulfilled for 1 second or more, a CAM is generated. However, regardless of the trigger conditions, the smallest period between two consecutive CAMs must not be less than 0.1 second.

This rule applies to all CAM elements except for the Low Frequency and Special Containers. In fact, the low-frequency and special containers, since they contain static information, are usually transmitted no more than twice a second.

Decentralized Environmental Notification Service

The DEN basic service is an application support facility provided by the facilities layer. It constructs, manages, and processes the Decentralized Environmental Notification Message (DENM) [68]. DENMs generation is triggered by an ITS-S application. These messages are asynchronous notifications and contain information about a road hazard or an abnormal traffic condition with related information. Unlike CAM dissemination, which is sent in broadcast in the neighborhood, DENM is disseminated to all ITS-Ss located in a specific geographic area. This is achieved through direct V2V or V2I communication, which, in multiple hops, allows reaching the interested area. At the receiver, the DEN service processes the received DENM and provides its content to the running ITS-S application, which may present the information to the driver if it is relevant.

A DENM contains information related to a dangerous situation that has a potential impact on traffic conditions or road safety. The contained information describes the event type, its position, its detection time, and a time duration. Since the event could vary, these attributes may change over space and over time. In some situations, the ITS-S

transmits a DENM for an event caused by the ITS-S itself, e.g., an electronic brake light event triggered by the EEBL application. In this case, the ITS-S handles the transmission and the termination of the event. In other situations, DENM could be related to a road event, and more than one ITS-S could transmit a DENM for the same dangerous event. It must also be emphasized that the transmission shall continue even after the ITS-S has changed its position due to the persistence of the event, e.g., a pothole on the road. The DENM transmission will be active until a Negation DENM for that event is sent. In this case, assuming that the originating ITS-S has changed its position and is not able to detect the end of the event, another ITS-S can send this type of message as soon as it notices that the event has ended.

DENM could also be sent by roadside units, which play an important role in VANET since their capabilities include delivering important information to vehicles. Indeed, thanks to their connection with infrastructure, RSUs can generate DENMs based on traffic condition information provided by the Traffic Management Center (TMC).

CA and DEN services are facilities that aim to provide application support functionalities for the ITS Basic Set of Applications (ITS BSA). In [69] [70] [71], ETSI has defined three V2X applications dedicated to road safety: RHS, ICRW, LCRW.

- **Road Hazard Signalling (RHS).** This application aims to increase awareness among ITS-S. It operates in two different modes. The originating mode involves the detection and signaling of a road hazard from an ITS-S to other ITS-Ss, generating related DENMs. The receiving mode involves signaling road hazards to the driver of the receiving vehicle when relevant to him. The objective of this mode is to increase driver vigilance, achieving the so-called "driver awareness".

- **Intersection Collision Risk Warning (ICRW).** This application is intended to detect and prevent potential collision risks between vehicles. It relies on CAM and DENM transmission in V2V and V2X, but it may also rely on roadside infrastructure services such as Traffic Light Maneuver (TLM) service, Road and Lane Topology (RLT) service, and Infrastructure to Vehicle Information (IVI) service. These messages enable a receiving vehicle ITS-S to be informed of the movement status of other vehicles in the intersection, as well as the traffic light status, intersection access priority status, and topology of the intersection. This receiving ITS-S is therefore able to estimate the potential collision risk and inform the driver when necessary [70]. Under this class are located several applications described in section 2.4 that reproduce several collision risks.

- **Longitudinal Collision Risk Warning (LCRW).** This application is designed to support a defined set of collision risks such as forward collision, forward/side collision, and frontal collision. The application operates in two different modes. The originating mode involves triggering DENM generation and transmission upon the detection of a longitudinal collision risk. The receiving mode involves analyzing longitudinal collision risks based on received information from DENMs sent by other ITS-Ss. In real-time, with a strong priority, a warning of a possible collision risk shall be shown to the driver, who may act immediately to avoid the impact.

4.1.3 Dissemination of V2X Messages

The ITS Networking and Transport layer includes protocols that are responsible for data delivery among ITS-Ss.

Basic Transport Protocol

Basic Transport Protocol (BTP) is a transport layer protocol defined by ETSI for the ITS-G5 standard. As described in [63], BTP provides an end-to-end, connectionless transport service in the ITS ad-hoc network. Its main purpose is to multiplex messages from different processes at the ITS facilities layer, such as CAM and DENM, for the transmission of packets via the GeoNetworking protocol, as well as the demultiplexing at the destination.

GeoNetworking

The GeoNetworking protocol is a network layer protocol defined by ETSI for the ITS-G5 standard. It provides packet routing in an ad-hoc network and makes use of geographical positions for packet transport. As specified by ETSI in [62], GeoNetworking supports communication among individual ITS stations, as well as the distribution of packets in geographical areas.

In order to ensure security requirements specified by ETSI standards (confidentiality, integrity, availability, accountability, and authenticity), GeoNetworking shall support security mechanisms such as cryptographic protection by digital signature, as defined in [72] and examined in detail in section 4.2.

GeoNetworking supports different forwarding schemes, each with a proper packet header.

- **GeoUnicast (GUC).** In this forwarding scheme, a node sends a unicast packet to another node through a path of other nodes. In this scheme, in fact, the packet reaches the destination by hopping through several nodes that re-forward the packet.
- **Topologically-scoped broadcast (TCB).** In this scheme, a node broadcasts a packet to all its neighboring nodes. All nodes within the communication range of the sender receive the packet, re-broadcast it to their neighbors, and so on.
- **Single-Hop Broadcast (SHB).** This forwarding scheme is similar to the previous one. The only difference is that the receivers within the communication range do not rebroadcast the received packet. So, it is limited to one hop, hence the term "Single-Hop". This scheme is the one used to disseminate messages generated by the CA service.
- **GeoBroadcast (GBC).** In this scheme, communication starts from a single ITS-S (on-board unit or roadside unit) and is destined for a group of ITS-Ss within a geographical area. This scheme is similar to GeoUnicast with the difference that when the packet reaches an ITS-S located in the destination area, the ITS-S rebroadcasts the packet to permit reception by all ITS-Ss in that area. This forwarding scheme is used to disseminate messages generated by the DEN service.
- **GeoAnycast (GAC)** This scheme is similar to GBC, with the difference that, in the destination area, only one ITS-S receives the packet instead of all ITS-Ss.

Packet Forwarding

Since in certain situations, it may be necessary to communicate with ITS-Ss that are not within the communication range of the sender, the previously mentioned GeoUnicast,

GeoBroadcast, and GeoAnycast aim to solve this problem. Taking into account the DEN service, it is interesting to underline the presence of several GeoNetworking forwarding algorithms that control how messages are forwarded to optimize channel congestion.

When GeoNetworking receives a packet that needs to be forwarded because it is a multi-hop packet, it must choose the proper algorithm between "area forwarding" and "non-area forwarding". If the ITS-S is within the destination area, the former is chosen; otherwise, the latter. The destination area is a geographic area represented by a shape (circle, ellipse, or rectangle) and is declared in the Extended Header of GeoNetworking.

Non-Area Forwarding

This mode aims to transport the message as close as possible to the destination area. In the forwarding procedure carried out by the ITS-Ss in intermediate positions, the message will not reach the application levels if the station is not located in the reported destination area. Figure 4.2 describes the forwarding process in which the packet reaches the DEN basic service at the facilities layer and is then passed down again to the forwarding procedure.

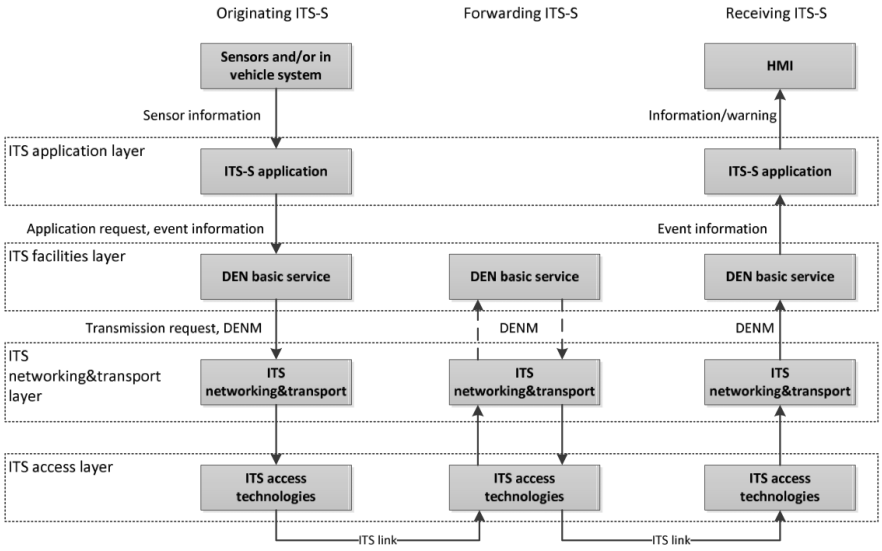

Figure 4.2. Multi-Hop Packet Forwarding Procedure.

For Non-Area Forwarding mode, there are two available algorithms: **Greedy Forwarding (GF)** and **Contention-Based Forwarding (CBF)**. In the Artery V2X Simulation Framework, CBF is the default option.

If **Greedy Forwarding** is used, every router in an ITS-S decides who will be the next hop by looking up in its Location Table (LocT) to find which ITS-S directly connected is located closest to the destination area. If no closer neighbor is available, the algorithm simply broadcasts the packet. The GF algorithm relies on the Location Table (LocT), which every ITS Station must construct by adding Location Table Entries (LocTEs) for every station that announces its information via periodic SHB packets. However, neighbor states stored in LocTEs may not be up-to-date as the stations have moved too far away from the range of the radio's signal.

The **Contention-Based Forwarding** algorithm solves the problem introduced by the GF algorithm by moving the routing decision from senders to receivers. In this

algorithm, every station passes the packet, which should be forwarded, up and puts it in a dedicated packet buffer where each enqueued packet has an associated timer. Once the timer ends, the packet is rebroadcasted. The duration of the timer is set inversely in relation to the progress of the packet toward its destination area compared to the original sender. The greater the distance, the shorter the contention period, ranging from `1 ms` to `100 ms`. Figure 4.3 shows the Contention-Based Forwarding algorithm formula, in which it is underlined that if the progress is more than the maximum distance (usually considered about 1 km due to the radio's coverage limit), the shortest possible timer duration is used [62].

$$TO\_CBF = \begin{cases} TO\_CBF\_MAX + \dfrac{TO\_CBF\_MIN - TO\_CBF\_MAX}{DIST\_MAX} \times PROG & \text{for } PROG \leq DIST\_MAX \\ TO\_CBF\_MIN & \text{for } PROG > DIST\_MAX \end{cases}$$

Figure 4.3. Timeout Calculation for Buffering Packets in CBF.

The goal of **CBF** is to ensure that the stations closer to the packet's destination will transmit it, resulting in a fast rebroadcasting approach. To avoid every ITS-S from rebroadcasting an already rebroadcasted packet and causing unnecessary network saturation, each ITS-S, while waiting for timer expiration, must listen for other rebroadcasting procedures of the same packet. This way, every ITS-S that detects the same packet for which it is waiting can discard the buffered packet and stop the timer, aborting the rebroadcasting procedure. As detailed in chapter 6, the duplicate packet detection proposed by ETSI ITS-G5 in [62] is based on a sequence number, and this is a vulnerable duplicate packet detection mechanism in case of a message modification attack. In the GeoAdhoc router, every ITS-S maintains a Duplicate Packet List (DPL) that contains sequence numbers of packets received from all ITS-Ss. DPL is used to check if a received packet is duplicated by checking if the received sequence number is already included in DPL [62].

Area Forwarding

The following GeoBroadcast forwarding algorithm is used to distribute a data packet within a geographical target area, unlike the previous ones, which are used to distribute packets from the sender to the destination area. Since this section is less relevant to the attack described later chapter 6, it will not be detailed.

4.2 ETSI ITS – Security

As established in section 3.3, there is a considerable number of attacks that can affect inter-vehicle communication (IVC). Authenticity and integrity of the information in transit in communications are important principles for an Intelligent Transportation System (ITS) since there are in-vehicle applications that process exchanged data to perform physical actions on vehicles.

Authenticity means that the claimed identity and the actual identity of a sender are the same, guaranteeing the authentication of the sender in respect to the receiver.

Integrity means that the exchanged messages have not been manipulated in an illegitimate manner by intermediary network nodes. This means that attackers are not able to inject modified messages to affect traffic.

To address these aspects, over the years, several researchers have proposed different mechanisms to secure V2X communication and ensure the cybersecurity principles at hand. All academic researchers are considered by the ETSI TC ITS WG5 working group, which in Europe was tasked with addressing security and privacy issues of the ITS-G5 standard. With the ETSI ITS Release 1 [73], there was a significant achievement of ETSI standards in several fields, including security and privacy, with several base standards. These standards describe a security framework for C-ITS with requirements and technical specifications for secure and privacy-preserving communication, formats of secured messages, and PKI architecture. To develop these standards, the ETSI TC ITS WG5 working group follows a three-step security process: identifying all the ITS security risks, defining security requirements and possible countermeasures to threats, specifying an architecture, and a set of security services that allow for the deployment of secure inter-vehicle communications.

4.2.1 Privacy in ITS

In addition to the security principles mentioned before, other principles are necessary: pseudonymity and unlinkability. These principles deal with the possibility that a vehicle's user can be directly or indirectly deduced, with an impact on the privacy of road users. Indeed, in [74], the following two key requirements related to privacy are identified:

- **Pseudonymity** means that a C-ITS station can use a resource or service without revealing its identity but still remains accountable for that use.
- **Unlinkability** refers to the ability of a C-ITS station to use resources or services multiple times without enabling others to establish connections between those uses.

Therefore, messages in ITS communication require authenticity, integrity, but also pseudonymity and unlinkability. However, there are conflicting requirements, for example, with authenticity and pseudonymity, which cannot be fully guaranteed simultaneously but must be balanced. Authenticity and integrity are ensured by means of a security architecture with the support of a **Public Key Infrastructure** (PKI). At the same time, pseudonymity and unlinkability are also included and balanced with the previous requirements by adopting replaceable pseudonym certificates called "**Authorization Tickets**". Authorization tickets provide pseudonymization since they are not directly bound to the real identity of the vehicle's owner and provide unlinkability since they are changed frequently; in fact, they are called **short-term certificates**.

As the certificate is not directly linked to a real identity, it avoids possible vehicle and user tracking. This introduces the trade-off about the conflict related to the necessity of having authenticity and pseudonymity at the same time. However, if there is a stringent requirement from a specific ITS application about the access to the real identity of the ITS station or its owner, it is possible to reveal it by requesting the long-term certificate behind the specific authorization ticket.

4.2.2 ITS Public Key Infrastructure Design

As mentioned earlier, the ETSI ITS security framework introduces the use of long-term certificates and short-term certificates.

- **Long-term certificates**, named Enrolment Certificates (EC), are used for the identification and accountability of ITS stations (ITS-S).

- **Short-term certificates**, named Authorization Tickets (AT) or pseudonym certificates, are used for V2X communications since they are anonymized certificates.

The privacy concerns already mentioned are driven by the content of safety messages and the fact that they are authenticated messages. This might allow for easy tracking of ITS-Ss if messages are signed with the same cryptographic certificate. Using a pseudonym scheme, tracking of vehicles becomes significantly more difficult or even completely avoided. Therefore, privacy requirements are assured.

However, the public key infrastructure (PKI) has to issue and distribute certificates for every ITS station with high frequency since they are short-lived. This presents two challenges. First, since emitting a new certificate requires an internet connection to interact with the remote responsible authority, it is not feasible in an ad-hoc network context where an internet connection is not guaranteed. Second, due to the time-critical constraints of ITS-S applications, it is not obvious that the responsible authority, upon a certain request, will issue the certificate in time to allow the continuation of communication for the vehicle under consideration. In light of these challenges, as described in [75], the solution is to consider a pool of pre-requested pseudonymous certificates that are available for use without any request. Thus, the PKI must distribute a set of Authorization Tickets to each ITS-S. This is done as the final step of a series of interactions between different entities of the public key infrastructure.

Cooperative-ITS Security Certificate Management System

The Cooperative-ITS Security Certificate Management System (also named C-ITS Trust Model) is composed of functional entities and a list of services provided by the trust model to support the life-cycle management of Trusted C-ITS Stations [76].

The C-ITS Security Management System might allow the operation of one or multiple Root CAs. Several options were evaluated in the context of the European C-ITS Platform WG5 report: Single Root CA, Cross-certification, Bridge CA, Certificate Trust List. In these descriptions, as reported in [76], when multiple Root CAs exist and cooperate within the C-ITS Trust Domain, the C-ITS Trust Model shall follow the Certificate Trust List approach.

ITS Authoritative Hierarchy

In the PKI design, there are the following functional elements and roles:

- **Root Certification Authority.** The Root CA is the highest-level CA in the certification hierarchy. It provides the EA and AA with proof that it may issue enrolment credentials and authorization tickets, respectively.

- **Enrolment Authority.** The EA is responsible for the life cycle management of enrolment certificates. It first authenticates ITS-S and then grants access to ITS communications.

- **Authorization Authority.** The AA is responsible for issuing and monitoring the use of authorization tickets. It provides an ITS-S with authoritative proof that it may use specific ITS services, which is detailed in the "ITS Station Profile" as indicated in section 4.1.2.
- **Sending ITS-S.** Acquired rights to access ITS communications from the Enrolment Authority and negotiates rights to invoke ITS services from the Authorization Authority. At this point, it is able to send single-hop and multi-hop broadcast messages.
- **Relaying ITS-S.** Receives broadcast messages from the sending ITS-S and forwards them to the receiving ITS-S if required.
- **Receiving ITS-S.** Receives broadcast messages from the sending or relaying ITS-S.
- **Manufacturer.** Installs necessary information for security management in ITS-S during production.
- **Operator.** Installs and updates necessary information for security management in ITS-S during operation.
- **Distribution Center.** This role is optional. It provides ITS-S with updated trust information necessary for performing the validation of received messages to check if they are coming from a legitimate and authorized ITS-S. It is in charge of publishing the CTL and CRL. If it is not present, the Root CA stores its security certificates information and the trust list information (CRL, CTL) in a local repository.

Since CAMs and DENMs are sent in broadcast, the trust relationship between ITS stations must support hundreds of millions of nodes and rapid verification given the time-critical V2X applications. To meet these requirements, enrollment and authorization roles are delegated to Trusted Third Parties (TTPs): EA and AA.

Root CA

Within each CA hierarchy, RCA acts as the trust anchor, controlling all subordinate certification authorities (i.e., EA, AA) and end-entities (e.g., ITS-S). It can update the list of trusted sub-CA certificates and revoke a sub-CA certificate. It issues certificate revocation lists for authorities (CRL CA). The primary functions provided by the RCA are as follows:

- Issuing CA certificates for EA and AA.
- Creating, renewing, and distributing Root CA certificates.
- Revoking the EA or AA certificate at their expiration or in case of a compromise of the CA private certificate.
- Generating and issuing the Certificate Revocation List (CRL) and Certificate Trust List (CTL).
- Generating log files for auditing purposes.

Every ITS-S, to verify the trustworthiness of incoming messages, must have secure access to at least the root certificate of the hierarchy for the authorization ticket attached to the message. The RCA certificate is transmitted to the ITS-S during the manufacturing or maintenance lifecycle stages.

Enrollment Authority

The EA is the entity proposed to authenticate the requesting ITS-S and issue a proof of identity in the form of an Enrolment Credential (EC). This certificate does not reveal the canonical identifier to a third party and should be used only to request the issuing of authorization tickets (AT) from AA.

The EA shall provide the following functions:

- Registering ITS-S and managing their EC, indicating the applications, services, and capabilities that the ITS-S is granted to use.
- Issuing EC to ITS-S after authenticating their requests.
- Revoking the ITS-S EC at the end of its life or in case of a compromise of the related private key.
- Creating and renewing EA certificates at RCA.
- Generating log files for auditing purposes.

Authorization Authority

The AA is the entity proposed to handle the requests of ITS-S which have already been authenticated by EA. The proof of granted permission by the AA is in the form of an authorization ticket (AT). Each AT specifies a particular authorization context, which comprises a set of permissions such as the application in which it is possible to use the certificate, application permissions which can limit a certain claim, a time period in the form of an expiration, a geographic region to limit the geographical validity of messages, etc.

The AA shall provide the following functions:

- Issuing Authorization Tickets to the ITS-S under specific requests and valid enrollment procedures at EA.
- Creating and renewing AA certificates at RCA.
- Verifying that the ITS-S has necessary permissions and is trusted when requesting AT.
- Generating log files for auditing purposes.

Together, these entities are sufficient to allow an ITS-S to make full use of the ITS applications and services. To start communicating with other ITS-Ss, every ITS-S requires obtaining specific credentials from AA; the so-called **Authorization tickets** (AT). These credentials ensure that every receiving ITS-S can verify that the sender has the right to send that message and can be trusted.

As mentioned before, these ATs are issued only under a specific procedure and verification steps in the PKI hierarchy. This procedure involves three phases:

1. Initialization: This phase is carried out together with the manufacturer of the vehicles or ITS device. It establishes a set of initialization credentials, including a canonical identity of the ITS-S (which is unique and unchangeable), a canonical cryptographic keypair for the ITS-S, a generic profile of the properties of the ITS-S (e.g., ITS facilities that it supports), and optionally a cryptographic certificate that is self-signed and links the canonical identity with the public key of the ITS-S and its generic profile.

2. Enrollment: In this second phase, the ITS-S interacts with the EA. It uses the canonical credentials (cryptographic keypair) to establish a set of enrollment credentials, one for each application or service that the ITS-S is permitted to use.

3. Authorization: In this last phase, the ITS-S interacts with the AA, submitting a request for authorization credentials after proving the possession of enrollment credentials. The AA will issue one or more cryptographically signed authorization tickets that, while preserving the privacy of ITS-S, will allow sending a specific type of message or information.

4.2.3 Secure Header Specification

When secure ITS communication is enabled, every single message is signed by every ITS-S using its own pseudonym (AT) introduced earlier. The transmitted packet presents a security envelope at the GeoNetworking layer, which consists of new fields such as protocol version, header fields, payload field, and trailer fields, as detailed in [72]. In particular, the Secure Header and Secure Trailer act as envelopes for the remaining part of the packet and allow the presence of multiple instances of header fields and trailer fields, respectively. The header fields are responsible for carrying important information, such as generation time, generation location, request for unrecognized certificates, ITS AID, signer info, etc. The trailer fields are responsible for carrying signatures. Only one payload data is admitted, and it is introduced by the "PayloadType" field (e.g., signed) and the length of the payload data.

Upon receiving secured messages, every ITS-S must verify the authenticity and integrity of the message by verifying the attached signature. This is possible since the Secure Header contains a field called Signer Info, which provides information about the signer (e.g., certificate). This certificate consists of the authorization ticket (Pseudonym) discussed earlier. However, in order to minimize network bandwidth usage, a certificate digest known as `HashedId8` is employed. This digest, which utilizes SHA-256, contains the hash of the pseudonym, resulting in a smaller size of just 8 bytes. Consequently, the Signer Info field is reduced from a size of 130–190 bytes to 11 bytes.

HashedId8 & HashedId3

`HashedId8` is a value used to identify data, such as a certificate. It is obtained by first calculating the SHA-256 hash of the certificate and then truncating the output to the least significant eight bytes. In the hash calculation, a canonical encoding for the EccPoint R (`EccPointType = x_coordinate_only`) must be used to ensure that the same certificate with a different value of `EccPointType` does not result in a different value of `HashedId8`. Highlighting the advantages in terms of network bandwidth usage with the adoption of a hashed representation of the certificate, it is better to use the latter.

However, this is valid only if surrounding ITS-Ss has already received and cached the non-hashed certificate. If the receiving station, upon reading the value of `HashedId8` and looking up in the certificate cache, does not find the related certificate, it discards the packet and, in its next CAM, announces the fact that a specific ITS-S is unknown to it. In this case, `HashedId3` comes into play.

As detailed in [77], ITS-S that does not know a specific `HashedId8` shall, in the next CAM, include a specific header "`request_unrecognized_certificate`" in which it puts all the unknown certificates in the form of `HashedId3`. According to [72], in fact, the real identification (AT) is not relevant. It is sufficient to provide an indication of the unknown certificate through this identifier. It is also a digest computed with SHA-256 on the pseudonym certificate and then truncating the output to the least significant three bytes. It is important to note that since the receiving station does not know the pseudonym certificate, it cannot calculate the relative `HashedId3` to fill the "`request_unrecognized_certificate`" header. Actually, it is possible to get the `HashedId3` by truncating the `HashedId8`, which is sent by the sender station, to the least significant three bytes.

From another perspective, when the sender receives a broadcasted message from the surrounding ITS-S (including the ones that do not know the certificate), it inspects the header field "`request_unrecognized_certificate`" in the secure header. For every `HashedId3`, it checks if it matches its own. If it is the same, the station will use an internal state that forces the use of the pseudonym certificate instead of the hashed version in the next periodic messages (i.e., CAM). With this approach, all stations can almost instantly resolve the lack of knowledge of a certain `HashedId8`. The advantages become appreciable when all stations have cached the `HashedId8` of the surrounding vehicles, resulting in a reduction of network bandwidth usage.

4.2.4 Pseudonym Change Issues

To increase the un-trackability of a vehicle, there are several pseudonym change strategies that aim to guarantee the unlinkability between two different pseudonyms. For example, if the pseudonym change happens quickly, and the ITS-S starts sending CAMs with the new AT immediately, a malicious actor could easily detect the vehicle's trajectory and deduce that the CAM with the new pseudonym coincides with the one that the target vehicle should have communicated. One of the interesting pseudonym change strategies is the **Silent Period**, which proposes that the vehicle remains silent for a certain amount of time after the change of the pseudonym is correctly completed. Tracking in this situation might be more difficult, especially if the change happens when the vehicle has a complex trajectory.

However, although pseudonym change and its change strategies solve the tracking issue, they also raise certain disadvantageous aspects. These are described in the following paragraphs.

ID Change Impacting Sender Behavior

As described in [75], each layer of the V2X communication stack has its own identifier. This can potentially lead to tracking even with a change in the authorization ticket used to sign the messages. There are the StationID at the facility layer, the Geonet address at the network and transport layer, and the MAC address at the access layer. One proposed countermeasure could be changing all IDs in the communication stack when changing the

pseudonym. This would mean stopping communication with the introduction of a delay, which could potentially be dangerous for safety applications. However, when analyzing the frequency of CAM dissemination, the maximum frequency (100ms) appears sufficient to complete the changing of all the IDs without impacting the performance of time-critical safety applications.

Misleading Neighbor Vehicles in Safety Situations

Another potential disadvantageous situation is the change of pseudonym in special situations, such as when the ITS-S is transmitting DENMs to alert road hazards. A pseudonym change may cause uncertainty for other vehicles, especially in cases where the pseudonym change strategy requires a silent period. If this is done in a dangerous situation, it might have safety consequences that cannot be neglected. In [75], two different dangerous situations in which this issue is present are analyzed.

- **Ghost Vehicles:** This scenario describes a situation in which, after the change of pseudonym, the ITS-S starts sending messages without a silent period. Since its old identity remains for a certain amount of time in the Local Dynamic Map (LDM) of its neighboring ITS-S, it is possible that the latter perceive the surrounding environment as incorrect due to the presence of multiple instances of the same vehicle. This can lead to the presence of ghost vehicles and could impact the decision-making of the vehicle in question.
- **Missing Vehicles:** Unlike the previous scenario, if the ITS-S observes a silent period after a pseudonym change, it does not send V2X messages immediately. As a consequence, new entries are not inserted into the LDM, and old entries are removed quickly, resulting in no information about the vehicle in the LDM of neighboring ITS-Ss. This is valid as long as the silent period is active. When the ITS-S restores V2X communication, it appears in the LDM of all neighboring vehicles. This could be dangerous in hazardous scenarios and might generate inappropriate reactions from surrounding ITS-Ss. For example, a vehicle could change its pseudonym just before an overtaking maneuver of a van, as shown in Figure 4.4. The vehicle preceding the van (Ego Vehicle) doesn't possess any information about the vehicle overtaking the van, so it decides to start a lane-change maneuver, assuming there's no hazardous situation. However, this could lead to a "Sideswipe Accident" since the overtaking vehicle acts as a missing vehicle and will reach the Ego Vehicle at speed without the Ego Vehicle realizing it.

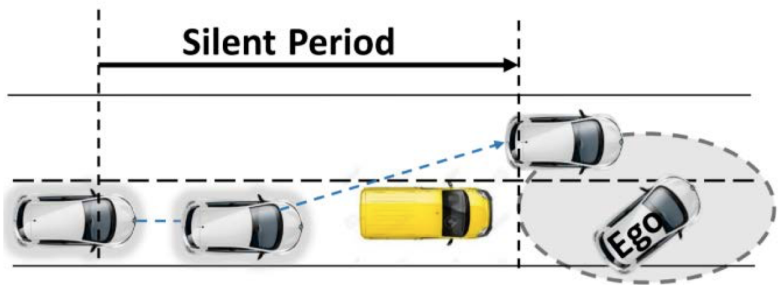

Figure 4.4. Impact of Pseudonym Change on Neighborhood Dynamics.

Chapter 5

Simulation of Vehicular Ad Hoc Networks

To experiment with the development of malicious attacks on VANETs and, thus, sensitize the automotive industry about the feasibility and related risks of V2X threats, the main aim of this thesis work is to conduct a simulation of malicious attacks on VANETs. To achieve this objective in a cost-efficient and flexible manner, enabling the reproducibility of the developed attack without causing harm to humans and vehicles under test (VUTs), the best option is to use simulation frameworks. This choice, as detailed in subsection 8.2.1, proves to be extremely useful for integration with the FEV Hardware-in-the-Loop (HiL) Validation Platform, where simulated real-world conditions are necessary. Therefore, this chapter analyzes the state of the art of Network Simulators, Road Traffic Simulators, and V2X Simulation Frameworks. Then, it evaluates their advantages, limitations, and performance, concluding with the identification of the best options for our needs.

5.1 Network Simulators

A network simulator is a software program that replicates the behavior of a real network. Since communication networks have become too complex for traditional analytical methods to provide an accurate understanding of system behavior, network simulators are employed. They serve as a virtual environment in which users can model, simulate, and evaluate the performance of network protocols, certain configurations, and topologies without the need for physical hardware. This section presents OMNeT++ and ns-3 network simulators.

5.1.1 OMNeT++

OMNeT++ is commonly referred to as a network simulator, but it is actually a generic simulation framework for developing complex distributed systems, as stated in [78]. It simulates discrete events and provides modules to construct real-world scenarios in a simulation environment. Over the years, it has become widely popular as a network simulation tool due to the numerous model frameworks that researchers have developed on top of it. One of the most interesting model frameworks is the INET Framework, which offers a wide range of models for working with communication networks. As mentioned

below, INET serves as the foundation for other simulation frameworks such as Veins, Artery, or SimuLTE.

The OMNeT++ simulator kernel is developed in C++ and provides a component architecture for developed models, using event scheduling, adding and controlling random numbers, etc. It comes with an Integrated Development Environment (IDE) that provides an environment for developing models. Regarding simulation execution, it can run under a graphical runtime environment (Qtenv) or under a console-based runtime environment (CmdEnv).

The OMNeT++ component model is composed of three essential components: modules, connections, and parameters, as shown in Figure 5.1.

1. Modules are self-contained components designed to achieve specific tasks. They can be used as-is or combined with other modules to perform more complex tasks.
   - Simple module: a component designed to achieve a specific task
   - Compound module: a component created by combining multiple simple modules to achieve a more complex task.
2. Connections are defined as part of a compound module and serve to connect two modules, allowing communication and the exchange of messages via the provided link.
   - Messages: elements that allow the exchange of predefined attributes such as timestamps or arbitrary data.
   - Gates: act as gateways, allowing the establishment of connections between two modules
   - Network: a combination of one or more of the described components
3. Parameters are possessed by modules and used to pass configuration data, helping define the model topology. Parameters may have default values, units of measurement, and other attributes attached to them. They can also be volatile.

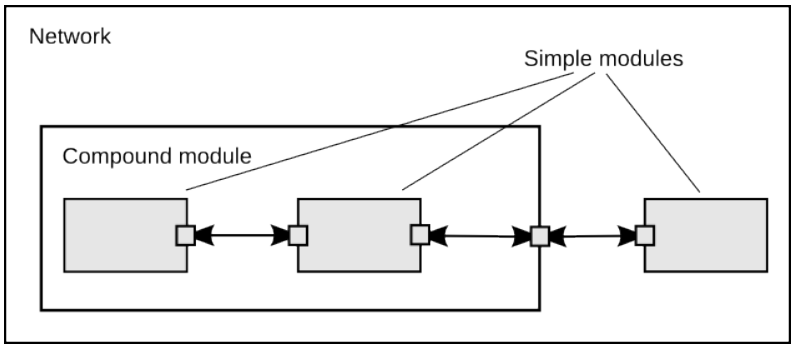

Figure 5.1. OMNeT++: Simple and Compound Modules.

Although using a model framework means that some of the following steps are mostly already done by the developer, we analyze the workflow of an OMNeT++ project.

1. As stated before, an OMNeT++ model is built from components (modules) that communicate by exchanging messages. When creating the model, it is necessary to map the system into a hierarchy of communicating modules.

2. Define the model structure in the NED language. The Network Topology Description (NED) file contains the structure of the network to be simulated.
3. C++ classes that represent protocol headers are described in MSG files, which are then translated into C++ code.
4. Provide a suitable `omnetpp.ini` file to hold OMNeT++ configuration and parameters of the developed model. One INI file may hold several configurations that can build on one another, and may even contain parameter studies.
5. Build the simulation program and run it.
6. Simulation results are written into output vector and output scalar files.

5.1.2 ns-3

ns-3 is a discrete-event simulator and represents a next-generation simulator that aims to improve existing system functionalities and network models of NS2. The main objectives behind the development of ns-3 were to provide a different software core written in C++ and a Python scripting interface to enhance simulation performance. ns-3 is extensible and upgradable by nature, allowing various organizations and researchers to continuously strive to contribute to it. Indeed, ns-3 currently has support from a large community of contributors who work to keep it up-to-date with the implementations of the latest network protocols, telecommunication standards, and network types.

Performance Comparison between OMNeT++ and ns-3

According to the performance study conducted in [79] and [80], it emerged that both ns-3 and OMNeT++ can handle large-scale and complex wireless ad hoc network simulations. Nevertheless, a discrepant result resulted from a comparative analysis of their performances. Regarding the maximum **memory usage** of the network simulators, it emerged that ns-3 uses the lowest amount of memory with an average of approximately 17.4% less than **OMNeT++**. Concerning the **computation time**, once again ns-3 is the best. In fact, with a result of approximately 25.45% less than OMNeT++, for both papers, it is the most efficient.

However, despite the better results obtained by ns-3 in the mentioned performance comparison, OMNeT++ has been chosen as the network simulator to achieve the objectives of this thesis. As detailed below, ns-3 provides modules to simulate vehicular ad-hoc networks (VANETs); however, it also brings with it some important limitations that led to the choice towards OMNeT++ network simulator.

5.2 Road Traffic Simulators

5.2.1 Simulation of Urban Mobility (Eclipse SUMO)

The Eclipse SUMO (Simulation of Urban Mobility) is an open-source, microscopic traffic simulation mainly developed by employees of the Institute of Transportation Systems at the German Aerospace Center. According to [81], it allows modeling and analyzing multimodal traffic simulation, including road vehicles, public transport, cargo logistics, and pedestrians. Road traffic simulation is extremely interesting since it facilitates the

evaluation of new traffic strategies, infrastructure changes, or policy changes before implementing them in a real-world environment. For example, the effectiveness of traffic light control algorithms can be tested and optimized in a traffic simulation before being deployed. Alternatively, it can be essential to evaluate route selection, re-routing algorithms, but also in the field of vehicular communication. Therefore, SUMO has also been proposed by Nico Weber in [82] as a toolchain component for the development and validation of automated driving functions via various X-in-the-Loop (i.e., MiL, SiL, and HiL). This is supported also by the fact that since SUMO is deterministic by default, it meets the requirement of reproducibility required by HiL Testing. However, it is worth noting that if necessary, there are various options for introducing randomness in SUMO simulation.

SUMO is defined as purely microscopic since it models each vehicle explicitly. Every vehicle possesses its own route and moves individually through the network. To specify this modeling, it is necessary to deal with several files described below:

- `.net.xml` file is meant to describe the network topology. This file might contain lots of information such as structures within an intersection, right-of-way logic, etc.
- `.rou.xml` file is meant to describe the routes present in the network. It is used also to define different vehicle types to differentiate their behavior and permissions (e.g., ego_vehicle and attacker_vehicle). Lastly, this file is also used to instantiate vehicles in traffic simulation.

Additionally, if the road network is generated through tools such as netconvert, it might be possible to deal with different extension files. For example:

- `.passenger.trips.xml` file is used instead of `.rou.xml` file and allows controlling the same parameters.
- `.poly.xml` file is generated automatically and reports all the information about 3D polygons of the generated road networks.

In the end, there is the `.sumocfg` file which allows the declaration of all the files that compose the simulation (e.g., `.net.xml` and `.rou.xml`).

To couple the traffic simulator with other components (e.g., network simulators), a set of APIs is provided which allows full control of the traffic simulation. **Tra**ffic **C**ontrol **I**nterface (TraCI), giving access to a running road traffic simulation, allows retrieving values of simulated objects and manipulating their behavior "on-line" [83]. It uses a TCP-based client/server architecture to provide access to SUMO. SUMO acts as a server and, when started with the `--remote-port <INT>` option, it waits for all external applications to connect and take over the control. This is extremely useful because, as will be clarified below, any V2X simulation framework requires the integration of the simulation environment (e.g., network simulator) with the SUMO traffic simulator.

5.3 V2X Simulation Frameworks

Because the objective of this thesis is to address malicious attacks in compliance with the ETSI ITS-G5 standard, only V2X simulation frameworks that support the European V2X standard will be considered. Particularly, delving into the state of the art of V2X simulation frameworks, the only available alternatives seem to be Artery and ms-van3t, which are described below.

5.3.1 Artery

Artery was born in 2014 out of the need for a simulation environment that supports vehicular communication compliant with the European specification (i.e., ETSI ITS-G5). It originated as an extension of Vehicles in Network Simulation (Veins), which at that time was the de facto standard for simulating VANET communication with OMNeT++. It was conceived as an extension since Veins focused on simulating VANET communication using the IEEE Wireless Access in Vehicular Environments (WAVE) standard mentioned in section 2.2. Thus, the idea was to reuse the lower-level model of the 802.11p channel, which was shared by both standards, and provide the implementation following the ETSI specification [84].

Additionally, Artery addresses another issue related to the execution of application types per simulation setup, which in Veins was limited to one. With the evolution of the INET framework over the years, an open-source model library that contains models of several protocols and components, including IEEE 802.11p, Artery is no longer tied to Veins. At present, Artery is considered a completely independent framework that enables V2X simulations based on ITS-G5, while Veins enables V2X simulations based on WAVE.

Architecture of Artery

Artery provides a framework to simplify interaction with the communication stack. Particularly, Artery provides middleware that acts as an abstraction layer and a data provisioning layer for the application. Every vehicle of Artery is equipped with an ITS stack represented in Figure 5.2. It consists of Veins or INET for the access layers, vanetza for the network and transport layer, and Artery for the facilities layer. The middleware is responsible for initializing the services for every ITS station. The services are declared in an XML configuration and can be linked to a single port number. This number is used to multiplex transmitted or received messages to the corresponding services.

Middleware Artery's middleware is also responsible for the life cycle management of the applications. This is extremely relevant for concerns about the insertion and update of vehicles inside the simulation environments. In fact, although SUMO and OMNeT++ adopt simultaneous updates operating on a step signal, Artery's middleware ensures a random offset that guarantees an individual update, which is more coherent with real life. Therefore, as depicted in Figure 5.3, even if two vehicles are updated by the traffic simulator at the same time instant, Artery's middleware guarantees the update is triggered at the application level at different time points, using individual intervals. The random offset that guarantees an individual update is cyclic.

As previously stated, the middleware also acts as an interface for data coming and going to the lower layer. Particularly, it provides `request()` and `indicate()` methods in charge of passing down and pulling up messages. It is worth noting that `request()` and `indicate()` methods are not restricted to the cycle-time scheduling of middleware. Indeed, it is not necessary to wait until the specific service is triggered to execute the cited methods. For example, when messages are received by the ITS-S, the `indicate()` method is immediately executed. It is possible to use two variants of messages, both accepted by Artery's middleware. The first is the most used one and consists of messages based on the Abstract Syntax Notation One (ASN.1), a standardized message format used by real "over-the-air" packets. They are necessary when, as detailed in subsection 8.2.1, Artery is coupled with external devices, e.g., in Hardware-in-the-Loop (HIL) platforms.

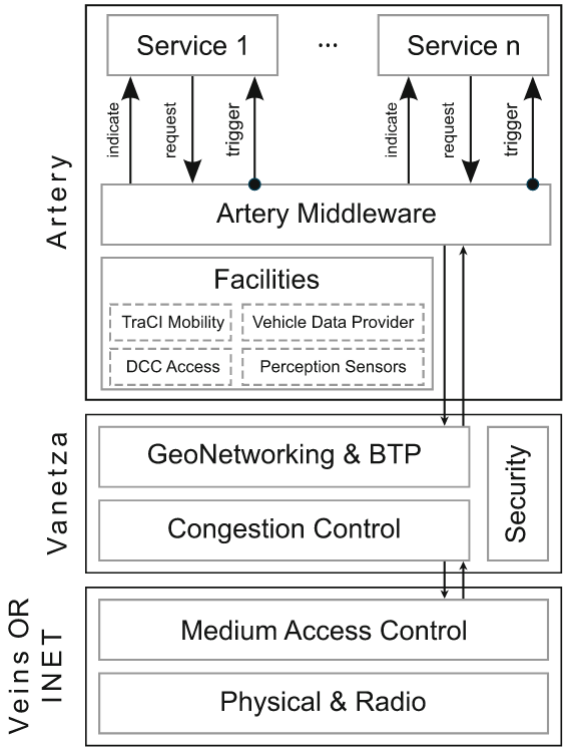

Figure 5.2. Exploring the Artery Architecture.

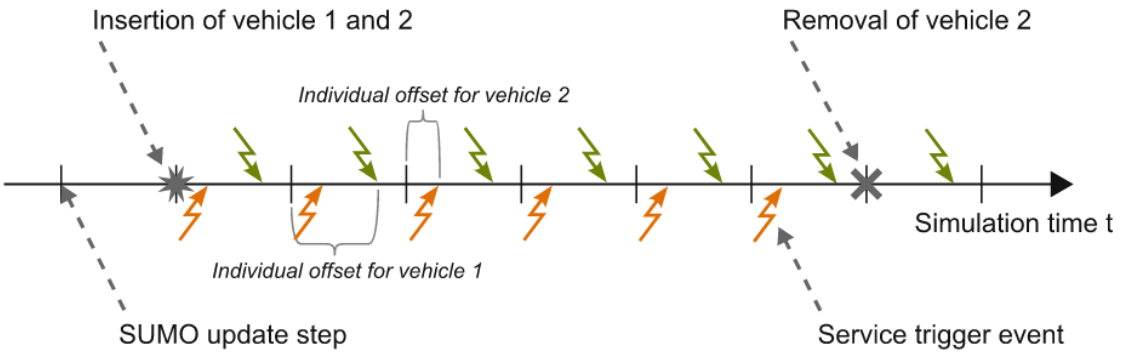

Figure 5.3. Artery: Life Cycle Management.

The second message format serves as rapid prototyping in a pure simulation environment and consists of OMNeT++'s cPacket objects that make no sense outside the OMNeT++ environment.

Services Services in Artery consist of application logics that must be considered as V2X applications or ITS functions in the simulated environment. In fact, every vehicle can be equipped with multiple applications at the same time to provide several functions. Artery is distributed with off-the-shelf services such as CA and DEN services. Every service can run its logic in three different ways. The first is to wait for the trigger scheduling, which as mentioned above consists of a random but cyclic offset from the SUMO simulation step. The second is to react to a received packet using the `indicate()` method that, as stated above, is not dependent on the cyclic scheduling. The last way is to listen to signals emitted by other modules of services.

Vanetza As outlined above and depicted in Figure 5.2, Transport and Network layers of the ITS stack are implemented by Vanetza open-source library. Particularly, Vanetza covers the GeoNetworking (GN), Basic Transportation Protocol (BTP), Security, and Decentralized Congestion Control (DCC). It is a C++ library designed to fit into OMNeT++ and the integration is already dealt with by Artery's build system. For what concerns Security, Vanetza provides a Security entity for every ITS-S used to sign and verify packets according to the ETSI C-ITS security extension of the GeoNetworking protocol based on [72]. The implemented features are the following:

- Security profiles, including the CAM and DENM profile
- Certificate requests for unknown certificates of other stations
- Certificate validation for incoming messages

These features are greatly interesting for a feasibility evaluation of malicious attacks in secured communication scenarios.

5.3.2 ms-van3t

ms-van3t is an ns-3 module to build and simulate ETSI-compliant V2X applications. It is based on the coupling of the SUMO traffic simulator and ns-3 network simulator. Like Artery, it provides the entire ITS-G5 stack, such as Facilities, BTP, and GN, with support for ASN.1 encoding. In particular, unlike what happens in Artery, ms-van3t supports only ASN.1 standard-compliant messages, not allowing for rapid prototyping using custom-defined messages. Also, ms-van3t offers several access technology models such as 802.11p, LTE, and C-V2X, with the possibility of easily switching stack and communication technology. In ms-van3t, the nodes are created in the ns-3 simulator as vehicles enter the SUMO simulation without any random offset to make the simulation more realistic. As stated about the use of exclusively ASN.1 encoded messages, ms-van3t is suitable for use in HiL platforms, too. It is worth noting that, unlike Artery, ms-van3t also offers an off-the-shelf sample V2X emulator application in which the messages are sent through a specified interface, as broadcast packets encapsulated inside BTP and GeoNetworking. It is possible also to specify a UDP mode which enables the transmission of messages after the encapsulation in a UDP datagram. For what concerns ETSI security specification, ms-van3t modules, among the several features, do not provide an implementation of the ETSI C-ITS security extension. That makes this V2X simulation framework quite limiting regarding the objectives of this thesis which, as detailed in subsection 8.1.2, include a feasibility evaluation of malicious attacks in secured communication scenarios.

As became clear through this survey, V2X simulation frameworks leverage on a traffic simulator and a network simulator. The first provides traffic information, while the second enables the simulation of vehicular communication. Regarding the traffic simulator, as is stated in several academic papers, Eclipse Simulation of Urban Mobility (SUMO) is absolutely the best choice in the open-source category. For what concerns the network simulators, although both ns-3 and OMNeT++ are extremely valid, and even though ns-3 has the best performance, reducing about 20% of the memory usage and computation time, OMNeT++ has been chosen. The choice was indirectly dictated by the related V2X framework. In fact, due to the limitation about the security domain for the ns-3 module ms-van3t, Artery has been the unique valid alternative which covers all the requirements defined for this thesis work.

Chapter 6

Malicious Attack Design and Implementation

Before delving into the description of the design and implementation of selected attack scenarios, it is crucial to clarify the potential behavior of attackers in order to have a comprehensive complete understanding of some of the choices made in their design.

6.1 Attacker Model

Since there is no universally adopted model for V2X attack classification, we will consider a model proposed by Raya et al. in [85]. The researchers divide attacker classification into four different dimensions:

- **Attacker Access.** This dimension considers whether the attacker has the right to communicate on the network. An attacker becomes an **insider** when they possess valid credentials to communicate with other nodes on the vehicular network. Such an attacker usually acts according to the underlying system protocol but uses malicious fabricated information. On the other hand, an attacker is considered an **outsider** when they do not possess valid credentials and behave as an intruder. For example, they may mount attacks aimed at misusing network-specific protocols.
- **Attack Objectives.** This dimension considers the objectives of the attack. A **malicious** attacker seeks no personal benefits from the attack, potentially harming network vehicles and causing human injury or death, such as causing accidents or traffic congestion. In this case, it is highly probable that the attacker disregards corresponding costs and consequences, implying their actions. On the other hand, a **rational** attacker follows a so-called "trade-off" between financial costs and advantages, avoiding completing their action at all costs when the trade-off becomes disadvantageous. Consequently, this attacker is more predictable in terms of the attack method and the attack target.
- **Execution Mode.** This aspect takes into consideration the mode in which the attacker executes the attack. An **active** attacker could use an active mode by actively interacting with the system, transmitting malicious packets or signals in V2X channels. A **passive** attacker, on the other hand, does not directly interact with the network. This category includes Trojans that can be launched directly by the vehicle's owner, for example, through a malicious infotainment app, or attackers

aiming to simply listen to unprotected communications to infer critical information (e.g., certificates, private keys, user information) without directly interacting with the system. The gathered information can be used to execute more complex attacks.

- **Execution Scope.** This dimension takes into account the extension of the attacker. An attack is considered **local** if it is limited to a few targeted ITS-Ss. Conversely, if it extends to a broad range of network nodes scattered across the network, it is considered an **extended** attack.

The simulated attacks described in this thesis cover all the above-mentioned attacker models. However, the only model not considered is the "rational attacker". This is due to the fact that, in the considered attack scenarios, personal benefits could, for example, involve clearing the road ahead. Since I have not enabled the rerouting option due to time constraints, I have excluded the possibility of considering a rational attacker who, to clear the road ahead, launches an attack aiming to reroute all vehicles ahead. Despite the lack of a rational attacker, the specular type, i.e., the malicious attacker, is considered more dangerous, as described earlier, since it is not predictable and has no limits in terms of financial and personal trade-offs.

6.2 Exploiting CA Messages

In this section, I describe the simulated attack scenario that exploits Cooperative Awareness Messages, as described in section 4.1.2. Since CAMs are periodic single-hop broadcasted messages, they are not rebroadcasted by every receiving station. This means that the attacker exploiting these messages realizes a local attack without the possibility of extending the threat over the radio-transmission range. Regarding Attacker access, both insiders and outsiders are taken into account for the attack scenarios described below.

6.2.1 Sybil Attack Case

Attack Design

In the Sybil attack scenario, the concept revolves around the presence of an attacker capable of sending malicious messages on the network using multiple identities simultaneously. Since, as mentioned earlier, it was not possible, due to time constraints, to consider a scenario in which the attacker forces the rerouting of other vehicles, I preferred to consider a malicious attacker aiming to disrupt VANET's services, leading to disruptive events or even human injuries.

The primary idea is to consider a traffic scenario with multiple vehicles, one of which is the "Ego Vehicle", the vehicle with the primary interest in the operational scenarios. In this attack, the assumption is that the attacker aims to craft specific malicious packets to annoy or provoke a dangerous reaction in the Ego Vehicle. These malicious packets are perceived as being sent by existing and trusted vehicles ahead of the Ego Vehicle. Specifically, the fake tracks are well-constructed so that the fake vehicles suddenly initiate an emergency brake, triggering a forward collision warning in the Ego Vehicle, which could lead to the Ego Vehicle stopping in the middle of the highway. This could be annoying in the best case scenario, or in the worst case, it could induce hazardous maneuvers that might result in dangerous accidents.

The premise is that the attacker has already forged, stolen, or compromised multiple fake identities and is ready to send malicious messages on the network. While this thesis primarily focuses on the attack methodology and not the detailed events of identity theft or compromise, some considerations about their feasibility are introduced in subsection 7.1.1, which adds realism to the considered scenario. As we will see below, the attack is implemented, considering the assumption that multiple identities have been acquired simultaneously. There are two ways to accomplish this. The first is to disable security in simulation, thus obtaining multiple identities by simply forging messages at the attacker's end with a new StationID at the facility layer, a new GeoNet address at the network layer, and a new MACAddress at the access layer. This has been the most straightforward solution to start implementing the attack with a less complex logic flow. The second approach used to evaluate the complete attack is to enable the use of ETSI ITS-G5 security but provide the attacker's vehicle with multiple available certificates, which are used to sign the malicious packets.

Software Architecture

In the Sybil attack, in addition to the main traffic simulator responsible for initiating the attack, two additional instances of SUMO were required. The first added instance, referred to as SUMO " Ego Perception", as shown in Figure 6.1, is used to display the perception of the Ego Vehicle based on the received CAMs. The second added instance of the traffic simulator is used as a source of information used by the attacker to construct malicious messages for the attack. While a more flexible but less deterministic approach is to forge malicious messages based on the received CAMs of the Ego Vehicle, it introduces variable delays depending on the congestion of the channel and the frequency of Ego's CAMs. Since SUMO is deterministic, the preferred approach is to consider a second instance of SUMO traffic simulator named SUMO "Fake Tracks", which simulates the fake tracks used by the attacker to craft malicious messages.

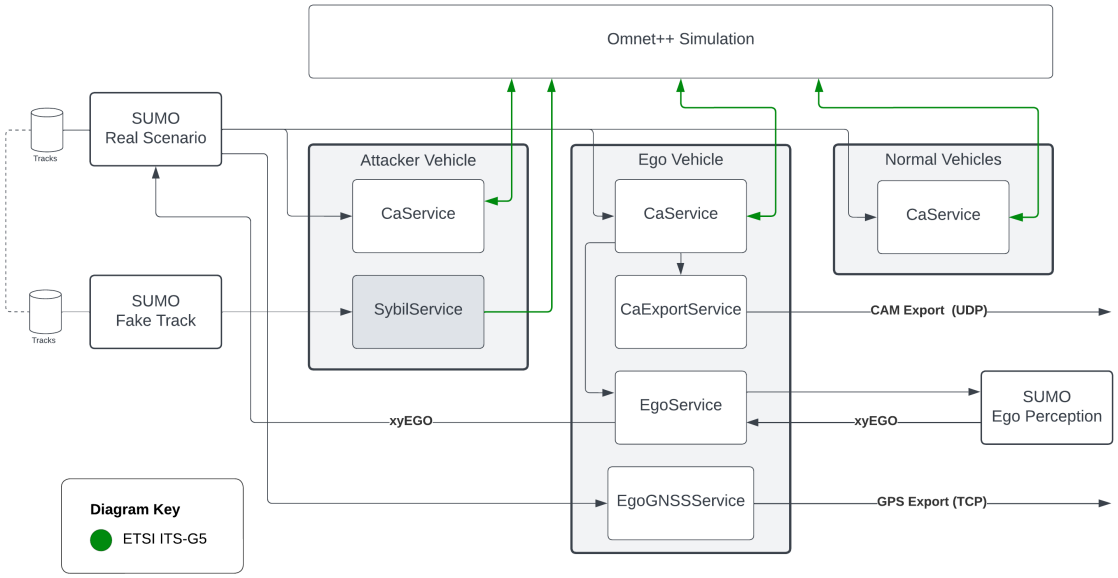

Figure 6.1. Sybil Attack Software Block Diagram.

In every simulated scenario, there are three different types of vehicles that differ based on their onboard active services. Since the focus is on the European standard, i.e., ETSI

ITS-G5, all vehicle types are equipped with Cooperative Awareness Service, which allows the exchange of periodic messages, i.e., CAMs.

The **Attacker Vehicle**, in addition to CaService, possesses a malicious service called **"SybilService"**, responsible for crafting malicious messages and sending them in the same manner as CAService. Specifically, SybilService, as mentioned earlier, takes input data from "SUMO Fake Tracks" and forges specific malicious packets directed at the Ego Vehicle. Simultaneously, the attacker functions as a normal vehicle by broadcasting its CAMs using CAService.

The **Ego Vehicle** possesses a higher number of services since it is the focus of the validation phase. In addition to CAService, which functions as described earlier, I implement CAExportService, EgoService, and EgoGNSSService.

These services are analyzed as follows:

1. **CAExportService** is designed to export ETSI ITS-G5 messages generated inside the simulator, allowing for the validation of simulated attacks against real V2X devices, such as the CohdaWireless MK5. This phase is outlined in the summary and detailed in subsection 8.1.1.

2. **EgoGNSSService** is designed to export information about the Ego Vehicle in the simulated environment. This enables, during the validation phase, the configuration of Cohda Wireless with data consistent with the simulation scenario, including geo-coordinates, speed, heading, etc. This allows the device to operate under specific conditions matching the simulation scenario. Instead of obtaining useful information from the GNSS, the Cohda Wireless device acquires them via a GPS Daemon service, which permits the injection of GPS information through a TCP socket. This is described in more detail in subsection 8.1.1.

3. **EgoService** is designed to update the second instance of SUMO, i.e., "Ego Perception", representing the perception of the Ego Vehicle. With each received CAM, it is necessary to update it by adding new vehicles or updating existing ones. This ensures that Ego Perception updates in real-time to reflect the Ego Vehicle's perception. However, since the Ego Vehicle reacts based on the perceived scenario, it is essential that every physical reaction of the Ego Vehicle is reported in the original traffic simulator, "Real Scenario". As described in section 7.1, this highlights that an emergency brake of the Ego Vehicle upon receiving malicious messages occurs even if there is no braking vehicle ahead.

Services Implementation

The description of service implementations commences with **SybilService** and proceeds to elucidate **EgoService**, **EgoGNSSService**, and, finally, **CaExportService**. In this section, high-level descriptions are employed due to the complexity and size of the code under consideration.

Concerning **SybilService**, it is imperative to underscore that its logic is aimed at constructing the malicious messages used to execute the attack.

This service comprises several operations outlined as follows:

1. An initialization phase establishes a connection between the service and the SUMO "Fake Tracks" instance through a TCP socket using TraCIAPI. This connection provides access to the SUMO scenario, enabling data retrieval and manipulation.

2. Within the periodically executed `trigger()` method in OMNeT++, the IDList of vehicles present in SUMO "Fake Tracks" is retrieved. For each vehicle instance, a new identifier, such as StationID, is created and stored in a `std::map` named `fakeVehiclesMap`. Vehicles no longer present in the SUMO instance are removed from this local map.

3. Subsequently, the service is prepared to transmit malicious messages over the network. For each element in the `fakeVehiclesMap`, the service evaluates the trigger conditions defined by the ETSI ITS-G5 standard for CAMs dissemination. These conditions, described in section 4.1.2, ensure that the dissemination frequency adheres to the limits established by the standardization entity.

4. When CamTriggeringConditions are satisfied, the forging of CAMs takes place. Given that the service is bound to SUMO Fake Tracks during the initialization phase, it can access real-time information about fake vehicles, including their position, speed, heading, and more, to populate the malicious messages.

As a result, vehicles equipped with this service can transmit malicious messages on the network, reflecting a realistic situation in which an attacker can manipulate their own vehicle to execute the described attack.

The second service developed is **EgoService**, responsible for updating the Ego Perception instance of the traffic simulator based on received CAMs from neighboring vehicles. This is essential due to the presence of malicious messages in addition to messages generated by vehicles included in the SUMO Real Scenario. To understand the Ego vehicle's response during an attack, it is vital to observe what it perceives, which is made possible by monitoring SUMO Ego Perception.

This service encompasses several tasks outlined as follows:

1. In the initial task, a connection is established with the SUMO Ego Perception instance via TraCIAPI, which employs a TCP socket. This connection allows access to the simulated vehicles' data and the ability to modify it.

2. In the `indicate()` method, which is activated whenever the node receives an ITS message significant at the application layer, logic for updating the Ego Perception scenario is implemented.

   (a) First, an evaluation is made to determine if security is enabled. This is crucial because, if enabled, vehicles are distinguished using hashedId8 instead of the StationID field in the CAM.

   (b) Subsequently, the service checks the local container to see if a vehicle with that specific identifier already exists. If the vehicle is not present, it is added to the container, and its data is updated using the information transmitted in the message. If the vehicle is already present in the local container, only the updating procedure is performed. Vehicle parameter updates in SUMO are executed using the TraCI API. Specifically, the `moveToXY()` and `setSpeed()` functions allow the manipulation of coordinates, heading, and speed.

3. Moreover, the EgoService, as mentioned earlier, is responsible for updating the Ego Vehicle's parameters in the SUMO Real Scenario based on the Ego Vehicle's reactions in SUMO Ego Perception.

(a) With a frequency matching the transmission of CAMs, the service updates the SUMO Real Scenario by retrieving the Ego Vehicle's parameters from the SUMO Ego Perception.

This synchronization is noteworthy as it highlights that even if there are no interfering vehicles ahead of the Ego Vehicle in the SUMO Real Scenario, it reacts abnormally due to the receipt of malicious messages.

The third service developed is **EgoGNSSService**.

1. It is responsible for real-time exporting vehicle parameters simulated in SUMO Real Scenario at a frequency of 10Hz. Utilizing the TraCI API, EgoGNSSService retrieves the Ego Vehicle's parameters and constructs a specific JSON payload that is exported through a TCP socket. As detailed in subsection 8.1.1, this payload facilitates the provisioning of GNSS parameters to external devices, particularly Cohda Wireless virtual machines, enabling them to operate as if they were in the simulated conditions.
2. Additionally, it exports parameters in the form of NMEA strings (e.g., GPGGA, GPGSA, GPRMC) used by physical Cohda Wireless devices.

For both exportation strategies, the parameters exported include UTC timestamp, longitude, latitude, speed, and heading.

Another indispensable service is **CAExportService**, responsible for facilitating interaction between the Artery V2X Framework and the external world. This service entails the creation of a UDP socket through which all CAMs transmitted over the simulated vehicular ad-hoc network are tunneled. As elaborated in subsection 8.1.1, additional headers, such as Link and Physical, are appended to enable Cohda Wireless devices to accept submitted packets.

This technique, while allowing packet transmission without modification, confines the simulation to the message content, disregarding packet propagation, which may occasionally result in losses due to excessive distance from the sender. In essence, physical devices will receive all packets regardless of potential losses in the OMNeT++ simulation. This ensures minimal latency in the export operation but may slightly affect the simulated scenario since the physical Ego Vehicle will receive packets that the simulated Ego Vehicle does not. Nevertheless, it is important to note that the additional packets received by physical devices will be inconsequential as their content pertains to areas that are not relevant.

Furthermore, to prevent the export of messages generated by the Ego Vehicle in the simulated scenario, a filter based on MAC addresses is applied, excluding these messages from transmission. Inclusion of these messages would introduce significant contamination since the content of messages generated by the virtual Ego Vehicle closely mirrors those generated by the physical devices (i.e., Cohda Wireless MKx).

6.2.2 Replay Attack Case

Attack Design

In a Sybil attack scenario, the idea is to assume the presence of an attacker who is capable of retransmitting sniffed network traffic, with the aim of gaining personal benefits or

causing network disruption. This attack is considered from the perspective of an insider attacker, but it might also be feasible in the presence of an outsider attacker since the primary task is the retransmission of captured traffic. In this attack, we define a scenario that involves three main vehicles: the target vehicle, the attacker vehicle, and the Ego Vehicle (victim). The target vehicle is the one from which the traffic is captured, the attacker vehicle is the one that carries out the described attack tasks, and the Ego Vehicle is the victim, receiving the retransmission of the captured traffic.

The idea behind this attack is to set up a scenario in which, firstly, the target vehicle transits, followed by the attacker, and then the Ego Vehicle. As shown in Figure 6.2, a segment of the highway is selected where the attacker vehicle captures and saves the CAMs transmitted by the target vehicle. The impact of a replay attack is significant if it is launched at a specific moment, for instance when the Ego Vehicle (victim) is passing through that specific area. The attacker will slow down to ensure that the target vehicle goes out of the broadcast range (approximately 900m) and then wait for the victim to arrive near the location of the first saved CAM to initiate the attack. Since all packets are buffered for a few seconds before being retransmitted, it is crucial to ensure that the Ego Vehicle does not also receive the fresh target vehicle's messages. If this were to happen, the Ego Vehicle might detect that the same vehicle continually changes its position from one set of coordinates to another, which is inconsistent with other parameters such as speed and heading, potentially revealing an anomalous phenomenon.

Figure 6.2. Replay Attack Design.

Software Architecture

In the replay attack software architecture, in addition to the already described implemented services, a new specific service, ReplayService, has been developed to enable the attacker vehicle to execute the described attack. Furthermore, an additional instance of SUMO is no longer required for the content of malicious messages, as they are retransmitted instead of forged. As shown in Figure 6.3, the replay attack is developed not only at the application level, as described previously. In the following sections, we describe the necessary software modifications at various levels. There is a clear connection between the "Vanetza Router" block, which identifies the vanetza::geonet::router instance responsible for handling the lower headers, and the ReplayService, which is in charge of controlling the attack at the application level. When the attack mode is active, the attacker vehicle sends not only its own authentic messages but also reconstructed messages belonging to

the target vehicle in the OMNeT++ simulation network. As detailed below, the messages are first buffered and then rebroadcast in a precise manner.

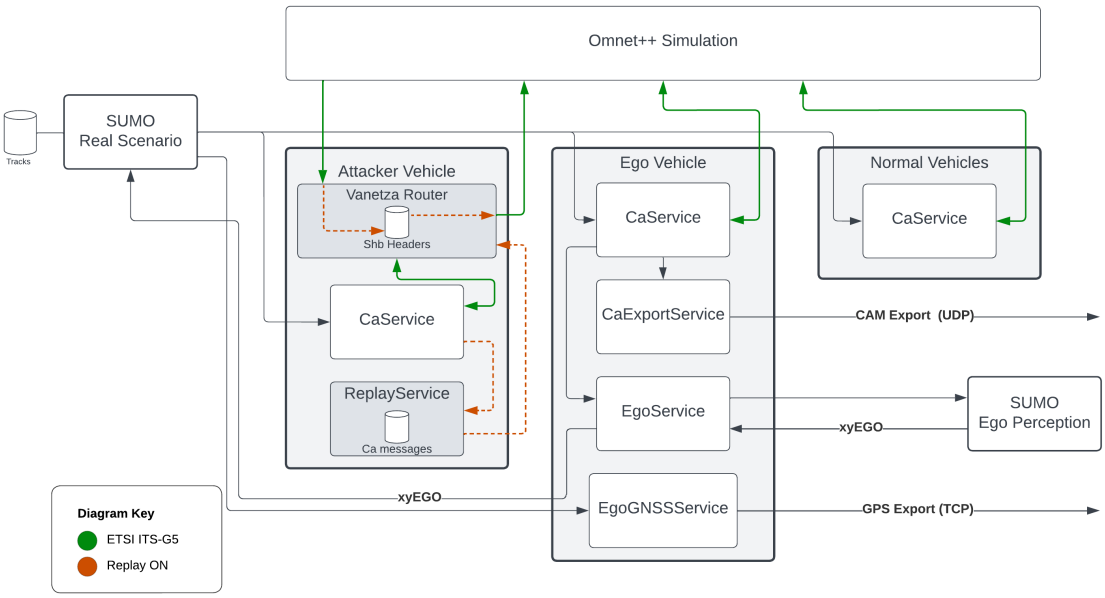

Figure 6.3. Replay Attack Software Block Diagram.

Services Implementation

This section provides a detailed account of the implementation of the ReplayService, with certain modifications made at the facility and network layers. Given the substantial size and complexity of the code under consideration, a high-level description is employed in this section.

ReplayService is exclusively implemented at the application layer, yet it leverages appropriate software modifications to facilitate the attack under analysis. At the application level, it manages the stages of the attack, including the initiation and termination of both buffering and retransmission. The ReplayService comprises the following operations divided into two blocks: **buffering** and **rebroadcasting**.

Buffering

1. The first step involves processing every received CAM at the application level, assessing whether the reference position is included within the radius of the selected point of interest for those with the target vehicle as the sender. If this check is successful, the service initiates the buffering operation.

2. Subsequently, when buffering is triggered, another logic block is responsible for storing all the payload of packets broadcasted by the target vehicle in a vector of unique pointers (`std::vector<std::unique_ptr<geonet::DownPacket>>`).

3. Similarly, when the target vehicle departs from the selected portion of the highway, the service halts the buffering procedure.

Rebroadcasting

1. The initial step in the rebroadcasting logic block is to determine whether the victim vehicle (Ego Vehicle) has reached the selected point of interest or if its reference position falls within the radius of that point. If the check is successful, the service initiates the retransmission operation.

2. Subsequently, at each cadence of the OMNeT++ trigger, the timestamps of buffered messages are evaluated. If the timestamp is equal to or greater than the current timestamp plus the added latency, the packet is sent immediately.

3. Next, the service removes the first element from the buffer using the `.front()` method on the `geonet::DownPacket` vector.

As demonstrated in Figure 6.3, it is important to note that the operation of packet buffering extends beyond the application layer. The aforementioned `geonet::DownPacket` represents only the payload and the application header of the transmitted message. To entirely reconstruct the ITS packet that is retransmitted, it is essential to include the lower header, encompassing the transport, networking and access layers. Consequently, in addition to the aforementioned application logic, there are other logical blocks within the Artery and Vanetza modules that, when appropriately modified, enable the following operations.

1. Initially, it is necessary to buffer not only the payload but also the missing headers in parallel. Within the `vanetza::geonet::router` class, several vectors are defined to accommodate the data that differs from one packet to another and is relevant. These include:

   - `vanetza::geonet::VariantPdu`
   - `vanetza::geonet::IndicationContext::LinkLayer`
   - `vanetza::security::SecuredMessage`

   The `vanetza::geonet::VariantPdu` class encompasses all packet headers (i.e., basic, secured, common, and extended). When the ReplayService triggers the commencement of the buffering operation, a state variable communicates to the modified software block within `vanetza::geonet::router` that the buffering operation may commence. Consequently, in the `indicate()` method of the router, the `vanetza::geonet::VariantPdu` of the incoming packets is appended to the allocated vector for future use.

2. Subsequently, when the ReplayService initiates the rebroadcasting operation, it forwards payloads starting from the buffered payload. These payloads reach the vanetza::geonet::router, and in the `pass_down()` method, the missing headers are added. In particular, instead of constructing lower headers consistent with the sender station (i.e., the attacking vehicle), modified code is employed to use the elements saved in the vectors to reconstruct the original packet transmitted by the target station. This process involves the following steps:

   (a) Updating the extended header, basic header, and common header based on the buffered values.

   (b) Updating the `LinkLayer::sender` and `LinkLayer::destination`.

(c) Finally, the front elements of the vector are removed to accommodate subsequent data in a First-In-First-Out (FIFO) manner.

Furthermore, if security is enabled, the attacker must also reconstruct the security header. Similar to the previous approach, a dedicated vector is used to buffer the security header, and it is populated in the `indicate_secured()` method of `vanetza::geonet::router`. Subsequently, in the `encap_packet()` method, instead of relying on the security entity for constructing the security header, the header is extracted from the `vanetza::security::SecuredMessage` vector.

6.3 Exploiting DEN Messages

In this section, I describe the simulated attack scenario that exploits Decentralized Environmental Notification Messages (DENMs), as described in section 4.1.2. DENMs are asynchronous messages typically disseminated using a multi-hop broadcast approach. This is particularly intriguing because they carry warning messages about hazardous events or locations. Consequently, there are situations in which the destination area exceeds the radio range, or the sender is not located in the target area. To address these challenges and cover a wide area without limitations, various forwarding schemes are utilized. In the following attack, the dissemination mechanism plays a crucial role, as the attacker exploits it to interfere with the propagation of authentic messages and pursue its objectives.

6.3.1 Message Modification Attack Case

Attack Design

In the case of a message modification attack, the concept revolves around the presence of an attacker capable of exploiting the dissemination algorithm. By interfering with this algorithm, the attacker maliciously contributes to the forwarding procedure. The attacker's objective is to modify the message as close as possible to the source to propagate the altered message to a greater number of Intelligent Transportation System Stations (ITS-Ss). In doing so, the attacker aims to modify the warning code and the location of DENMs, causing network confusion.

As previously mentioned in section 6.1, given the inability to reroute vehicles, the only viable objective for the attacker is a malicious one. With this attack, the malicious actor aims to prevent vehicles from detecting the real coordinates of the hazardous situation. Consequently, modifying the location of DENMs and the warning code could lead to the creation of a fake congestion network that disrupts normal road traffic on a common road. The most critical aspect is the potential danger it poses to vehicles traveling on the road associated with the warning. It is important to note that vehicles on the hazardous road will only become aware of the dangerous event within the line-of-sight. This situation poses a significant risk to human safety.

The key strategy is to exploit the **Contention-Based Forwarding** algorithm, as described in section 4.1.3. This algorithm delegates routing decisions to receivers, who are responsible for avoiding packet duplication using a coordination mechanism. To avoid interfering with the forwarding algorithm, the attacker is strategically positioned to transparently ensure that the attacker's router calculates the shortest retransmission timeout

without any tampered logic. However, to prompt vehicles in contention for DENM retransmission to abort the procedure, they must detect a duplicate packet, indicating that retransmission is no longer necessary.

This attack exploits Duplicate Packet Detection (DPD), based on a sequence number evaluation mechanism. As described in [62] and mentioned in section 4.1.3, DPD relies on a sequence number evaluation mechanism. Each GeoAdhoc router maintains a Duplicate Packet List (DPL) for every entry in its Location Table (LocT). When the router processes a packet from the source, the DPL is consulted to identify duplicate packets. The router compares the value of the Sequence Number (SN) field in the packet and checks for a corresponding entry in the DPL. If such an entry exists, the packet is marked as "duplicated"; otherwise, the new sequence number is added to the DPL in a circular buffer approach. Consequently, it is possible to pass off the message as authentic simply by not modifying the SN field.

To create a realistic scenario using the **Non-Area Forwarding** transport mode, it is essential for the sender of the DENM to be far from the location of the notified hazardous event. Specifically, the sender must be at least at a distance equal to or greater than the range of the employed radio transmitters (approximately 1 km). It is worth noting that if a vehicle transmits DENMs for a dangerous event and remains on the spot because it is directly involved, the attack's success is compromised. An approaching vehicle will initially receive the modified messages propagated by an alleged attacker. However, upon nearing the location, it will receive the original message as it comes within the range of the sender's radio station. Therefore, an approach involving a Roadside Unit (RSU) located beyond the reach of vehicles is employed. In this way, the attacker can exploit the propagation algorithm without encountering the limitations mentioned earlier.

As shown in Figure 6.4, the attack scenario is overlaid with numbers indicating different critical locations:

1. Indicates the position of the RSU, which, as mentioned earlier, is situated outside the area of interest, typically at a common service station. The distance between the RSU and the actual hazardous location is over 2000 meters, eliminating the limitation where approaching vehicles receive original messages from the sender station.

2. Marks the attacker's position, which is approximately 900 meters away from the RSU. This strategic placement ensures that the ITS node consistently possesses the lowest retransmission timeout in Contention-Based Forwarding (CBF). This is valid when considering the direction of the event; otherwise, the attacker's location may not be optimal.

3. Shows the authentic location of the simulated hazardous event.

4. Indicates the location of the fake hazardous event. The coordinates of this event replace the coordinates of the real hazardous location in the transmitted DENM by the malicious attacker.

5. Marks the position at which the supposed logic on board the vehicle initiates a slowdown operation upon receiving notifications about the presence of a dangerous event approximately 400 meters ahead. This corresponds to the point where the traffic jam begins.

6. Designates the point at which vehicles will first receive messages about the hazardous location. This point is essential to highlight that vehicles entering the road

associated with the hazardous event will only become aware of the danger within approximately 100 meters (line-of-sight) rather than the 1000 meters defined as the relevant area. This presents a high risk for human injury due to the short distance available for emergency braking, which is further influenced by road and weather conditions.

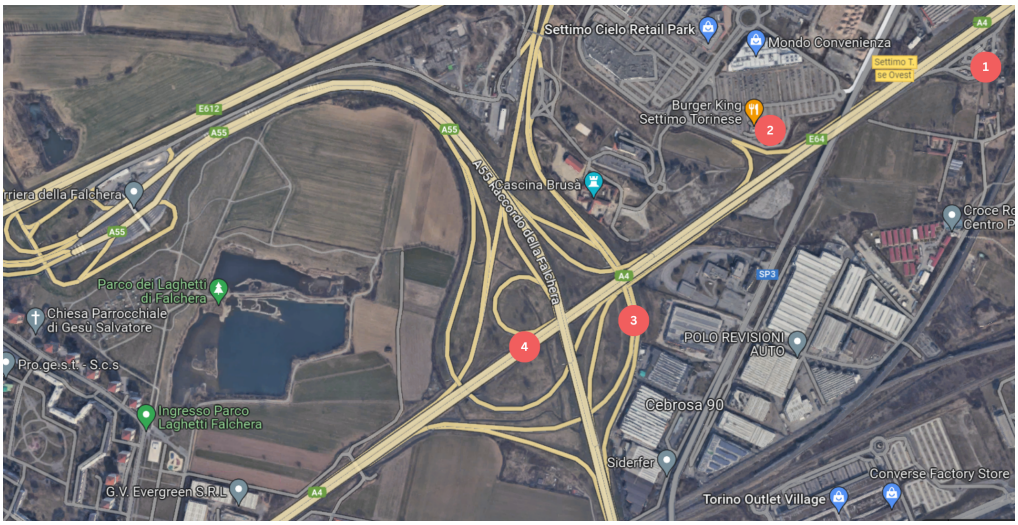

Figure 6.4. Message Modification Attack Design.

Software Architecture

Concerning the software architecture of the message modification attack, as depicted in Figure 6.5, although the vehicle categories are the same as in the previous attack scenario, they are now equipped with new services. The message modification attack is more complex and, like the replay attack, it necessitates modifications not only at the application level but also at the facility and network layers, as elaborated below.

As evident in Figure 6.5, each vehicle is equipped with DenService, responsible for handling the transmission and reception of DENMs. The functionality of this service has been extensively explored in section 4.1.2. However, in the attacker vehicle, an additional service named "MessageModificationServices" is present. This service is responsible for managing the attack at the application level. Specifically, it handles the modification of messages at the application level and oversees changes in message propagation at the router level. When the Message Modification attack is active, the observable flow indicates that instead of propagating the original messages, the attacker vehicle modifies the messages at the router level using information injected by the application service and then propagates the malicious versions.

The HazardousLocationServices is designed for implementing a simple logic that demonstrates how a vehicle reacts upon receiving DENMs. In a real-world context, it would generate warning messages displayed on a dedicated Human Machine Interface (HMI), akin to CohdaWireless Devices.

Furthermore, unlike the software architecture encountered in the sybil and replay attack scenarios, which involved at least two instances of SUMO, in this case, only the real SUMO scenario is employed. This choice stems from the attacker's objective, which

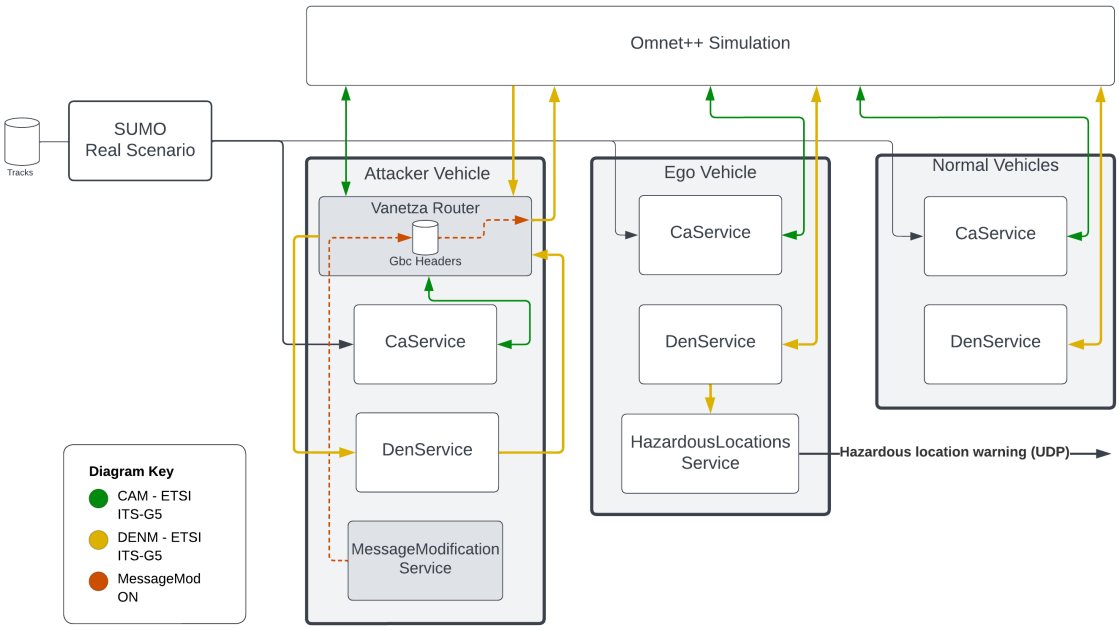

Figure 6.5. Message Modification Attack Software Block Diagram.

is to modify existing messages rather than create or replicate them. Having a single view and appreciating the differences by running the simulation twice, once with and once without the attack, serves to highlight the impact of the attacker and associated risks.

Services Implementation

The following paragraphs detail the implementation of "**MessageModificationService**" and the "**HazardousLocationService**" with a focus on router-level modifications that enable the attacker to interfere with the normal flow of GBC (Geographic Broadcast) packets.

HazardousLocationService

This service is entirely implemented at the application level as its primary role is to process received DENMs and determine their relevance to the vehicle in which it is running. This service is relatively straightforward as it does not involve the complex processing of DENMs, typically the responsibility of OEMs. The service comprises the following operations:

1. In the `indicate()` method, the application-level data of received DENMs is evaluated, specifically calculating the distance between the vehicle's position and the target area of the messages.
2. If this calculated distance becomes less than 0.4 km, a gradual vehicle braking operation is initiated as the vehicle approaches the hazardous location.

MessageModificationService

This service is more complex than the previous one as it interacts with the router level as well. The service's implementation consists of the following operations:

1. Initially, in the `indicate()` method, the service implements the modification of specific parameters selected by the attacker (e.g., `eventPosition.latitude`, and `eventPosition.longitude`).

2. Subsequently, it carries out the construction of new fake BTP (Basic Transport Protocol) and GeoNetworking Headers, aligning them with the newly modified payload. The changes are limited to the modification of `destination.position` in `vanetza::geonet::Area`, in accordance with the design explained above. The `eventType.causeCode` is not included in the network header since it is not relevant for dissemination purposes.

3. Following this, the application service exploits the `sendDenm()` method, which, with a specific parameter list, updates the respective local variables in `geonet::router` with the modified contents.

4. Finally, when the attacker router becomes involved in contention-based forwarding and, for the reasons mentioned earlier, may obtain the lowest timeout, it participates in the re-transmission procedure. In the `process_extended()` method, which is the final step of the indication procedure and is responsible for analyzing the extended packet, the `pass_down()` method is employed to rebroadcast the messages. Just before the `pass_down()` method, an exchange of elements occurs. In particular, the payload is replaced with `m_malicious_payload`, and the PDU is replaced with `m_malicious_gcb_pdu`. This way, in each subsequent DENM forwarding, the attacker router performs this substitution, thereby executing the message modification attack.

6.3.2 Black Hole Attack Case

Attack Design

The concept behind the Black Hole attack case involves an attacker contributing maliciously to the forwarding procedure. Specifically, the premise is that the attacker can block the propagation of Decentralized Environmental Notification Messages (DENMs) by discarding all the messages, rather than rebroadcasting them, as described in the contention-based algorithm used in the non-area forwarding mode. Much like the Message Modification Attack, the attacker is considered a malicious actor because their aim is to disrupt VANET services. Additionally, as in the previous attack scenario, the attacker needs to interfere as closely as possible to maximize the impact of the attack. Hence, the attacker exploits the Contention-Based Forwarding algorithm, as described in section 4.1.3, which has already been discussed in subsection 6.3.1, to achieve the lowest retransmission timeout and be the first to rebroadcast the DENM.

However, there is a challenge. Unlike the Message Modification Attack, where the attacker actually transmits the packet, in this case, the attacker needs to drop it. This action triggers a retransmission by one of the neighboring vehicles. These vehicles do not stop their timers because they do not detect a duplicate packet, so the vehicle with the lowest retransmission timeout rebroadcasts the message. To prevent this from happening and thus avoid the success of the attack, two alternatives exist:

1. It is assumed that the attacker is the only vehicle within the range of the Roadside Unit (RSU) transmitting the DENMs, making it impossible to resolve the drop in any way. Therefore, the attacker succeeds.

2. The attacker can rebroadcast the messages only to the vehicles that are in contention and might rebroadcast the message if the attacker discards it. In this manner, the vehicles will abort the rebroadcasting procedure upon detecting the duplicate packet. In reality, the attacker has sent the DENMs only to these vehicles. Thus, the attacker succeeds.

Much like in the Message Modification Attack, to construct a realistic scenario, it is necessary to place the attacker in a strategic position to guarantee interference with all the transmitted DENMs for the considered hazardous location. Therefore, the considerations mentioned above remain valid.

The attack scenario depicted in Figure 6.6, which is the same as the one used in the message modification attack, is overlaid with numbers indicating different critical locations:

1. Indicates the position of the RSU, as mentioned earlier, situated outside the area of interest at a common service station. The distance between the RSU and the actual hazardous location exceeds 2000 meters, eliminating the limitation where approaching vehicles receive original messages from the sender station.

2. Marks the position of the attacker, approximately 900 meters away from the RSU. In this way, there is a high probability that it will always be the Intelligent Transportation System (ITS) node with the lowest retransmission timeout in contention-based forwarding (CBF). Of course, this is valid when considering the direction of the event at hand; otherwise, the attacker's location may not be optimal.

3. Indicates the genuine position of the simulated hazardous location, which involves a simulated accident reported to the responsible entity, which communicates the hazard through RSUs.

4. Marks the location where a vehicle approaching at high speed might first see the hazard within their line of sight. This occurs about 100 meters before the hazardous location, partly due to the curvature of the road. It translates to a high risk of human injury due to the short distance available for emergency braking, further influenced by road and weather conditions, especially on the curved road.

5. Indicates the point at which vehicles will receive the first message about the hazardous location. This serves to highlight that, in the absence of the attack, they would have been made aware of the dangerous event about 1000 meters before, rather than the 100 meters (line-of-sight). This presents a high risk of human injury.

Software Architecture

The software architecture of the Black Hole attack, shown in Figure 6.7, is largely similar to the previous attack, with a few specific details. Notably, the addition of the "BlackHoleServices" is introduced, which is responsible for executing the attack, and there are modifications in the interaction with the `vanetza::geonet::router`. While all other services and connections remain consistent with the previously described attack, this software architecture emphasizes that some transmissions are blocked by the attack itself. Since the primary objective of the attack is to disrupt the propagation of Decentralized Environmental Notification Messages (DENMs), the attacker vehicle's router

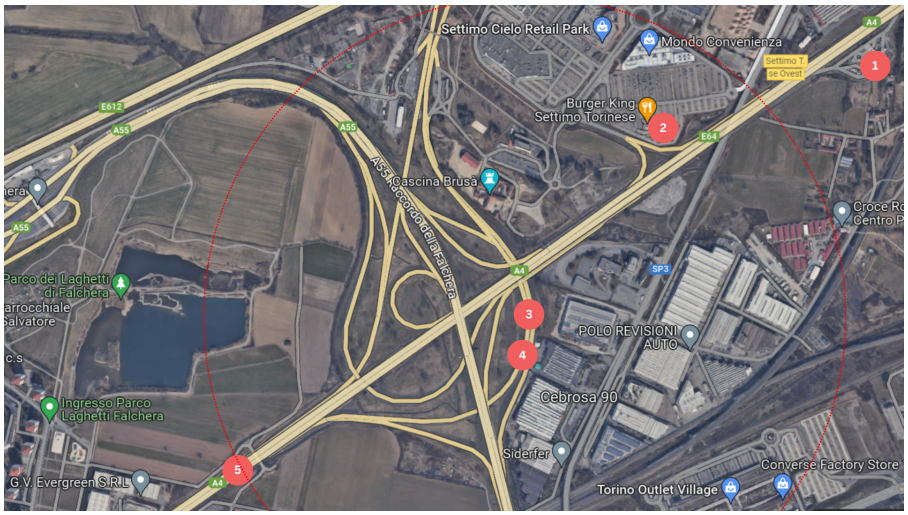

Figure 6.6. BlackHole Attack Design.

will not export received DENMs, thus interfering with the dissemination of hazardous information. Consequently, the "HazardousLocationServices" on the Ego Vehicle will not forward the warning to the Human-Machine Interface (HMI). The detailed implementation is described below.

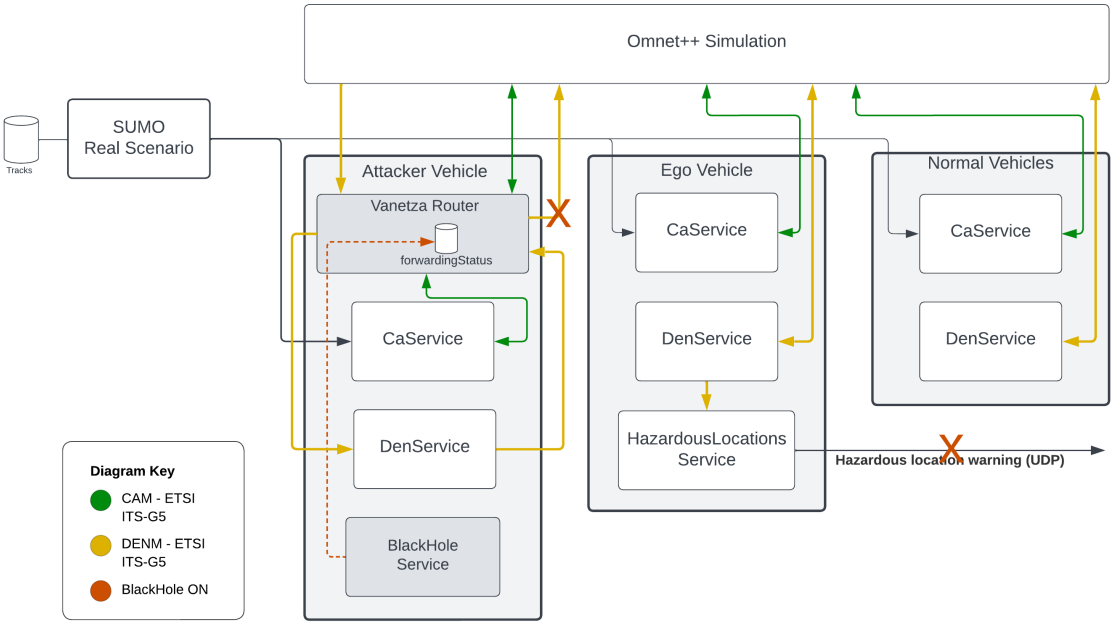

Figure 6.7. BlackHole Attack Software Block Diagram.

Services Implementation

The following outlines the implementation of the **BlackHole Service**.

In contrast to the other services, the BlackHole service is relatively simple and consists of only a few lines of code. It is composed of two logic blocks: the first one, at the application level, controls the initiation of the attack, and the second one implements the

interruption of the forwarding procedure.

1. To initiate the attack, it is sufficient to equip the vehicle with the Black Hole service. In the `initialize()` method, the service updates the local variable named `forwardingStatus` at the router level to signal the termination of cooperation in a multi-hop broadcasting.

2. At the router level, specifically in the `process_extended()` method previously mentioned in the Message Modification Attack, just before the call to `pass_down()`, which is responsible for reintroducing the message into the network, the operation is aborted. In particular, by evaluating the local variable, the router interrupts the normal execution flow of the method and returns to the caller. The approach is analogous to if the router had detected a duplicate packet, but in this case, it is enforced by the attack's objective.

   Unfortunately, since there is no possibility of rebroadcasting selectively in OMNeT++, the message is prevented from being transmitted only to the other cooperating vehicles in the multi-hop forwarding. The chosen approach aligns with the first option mentioned earlier in section 6.3.2.

Chapter 7

Malicious Attack Execution and Security Evaluation

7.1 Malicious Attack Execution

After conducting a comprehensive analysis of the design and implementation of the selected malicious attacks, this section concentrates on their execution in the V2X simulation environment. Furthermore, their feasibility is evaluated within a secured communication scenario, in accordance with ETSI ITS-G5 Security

7.1.1 Result Analysis

Sybil Attack

The execution of the Sybil Attack is correctly carried out with the expected results. In this scenario, the attacker manages to send forged messages using multiple identities, thus influencing VANET services. Particularly, the attacker, focusing on the Ego Vehicle, manages to disrupt its on-board services. With the use of carefully crafted deceptive messages, as described in section 6.2.1, the attacker creates the illusion that there are vehicles ahead of the Ego Vehicle that are about to brake. Upon receiving and processing these messages, the Ego Vehicle perceives the dangerous situation and initiates emergency braking. However, as highlighted in Figure 7.1, which shows the comparison between the real scenario and the one perceived by the Ego Vehicle, there are no preceding vehicles, rendering the braking useless. Furthermore, although the latter event may be merely annoying, attention must be paid to its safety risk. In a real-world situation, especially with vehicles equipped with High or Full Driving Automation (i.e., level 4 or 5), the perception of a vehicle ahead could lead to unnecessary emergency braking on a clear road, as depicted in the simulated scenario. Additionally, if the emergency braking is inadequate to prevent the perceived collision, the autonomous vehicle might execute hazardous maneuvers to avoid the accident, potentially causing events that could result in human injury or death.

Replay Attack

The execution of the Replay Attack within the V2X simulation framework is rendered unfeasible due to the packet discard mechanism triggered by an expired timestamp.

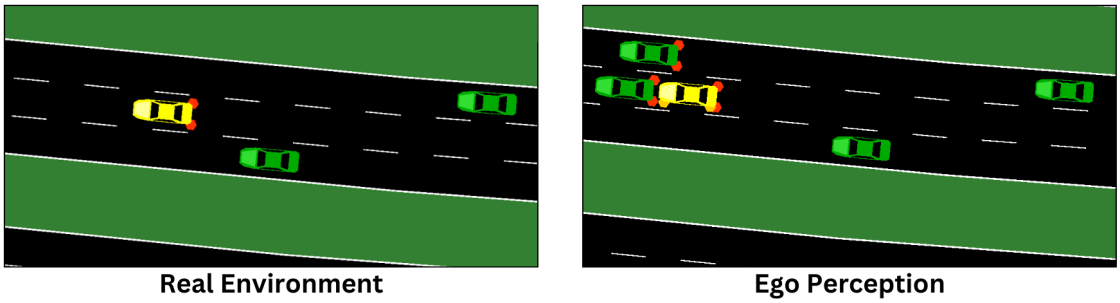

Figure 7.1. Sybil Attack Dynamics: Contrasting Real Environment with Ego-Perceived Scenario

Nonetheless, in order to evaluate the attack's potential execution and underscore the identified software vulnerabilities in CohdaWireless devices, I intentionally activated the `pass_up` procedure even in instances of an expired timestamp. In other words, while it may not be possible to execute the following attack in the Artery V2X Simulation Framework, the mentioned patch could potentially enable its execution, allowing for a comprehensive assessment of the attack's impact. As detailed in section 6.2.2, this attack might be carried out also in the case of an outsider attack since it consists of retransmitting previously captured traffic.

In the executed attack scenario, the malicious attacker pays attention to the retransmission to have the most dangerous impact on the Ego Vehicle. By waiting for the Ego Vehicle to reach the specified location claimed in the initial messages of the sequence to be replayed, the attacker guarantees that the retransmitted messages will affect the victim vehicle. In fact, when the Ego Vehicle receives the replayed messages, it analyzes them and proactively detects the presence of a vehicle ahead. Subsequently, the messages claim that the vehicle is slowing down, causing the Ego Vehicle to initiate emergency braking to prevent a collision with the vehicle ahead. As depicted in Figure 7.2, the Ego Vehicle executes physical actions based on the traffic replayed by the attacker, actions that are inconsistent with the actual environment. Similar to the previously analyzed attack scenario, this approach involves dangerous and unnecessary maneuvers that can lead to hazardous events, including human injury and death.

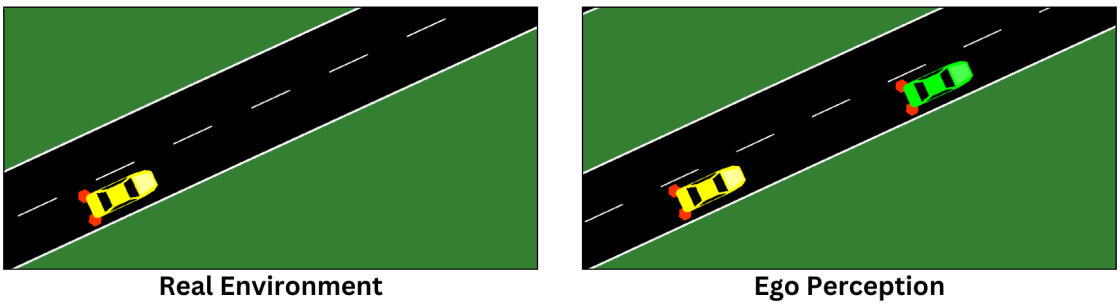

Figure 7.2. Replay Attack Dynamics: Comparing the Real Environment with the Ego-Perceived Scenario

Message Modification Attack

The message modification attack is feasible in the V2X simulation framework, and it reports the expected result. Its execution starts with a simulated accident on a selected motorway junction. Then, the RSU informed about the hazardous event starts to broadcast DENMs, which include several pieces of information, including the hazardous location and the event `cause_code`. These messages might be delivered in the relevance area, and since the sender (in this case, the RSU) is not able to cover the specific area, the messages are sent using a multi-hop broadcast strategy. As detailed in section 6.3.1, the attacker exploits this condition, selecting a strategic position to become the vehicle elected to carry out the first forwarding of multi-hop messages. The modification of the attacker has an effect, and the retransmission of the latter leads to aborting the same procedure in the neighboring vehicles. This is the way with which the attacker delivers the modified packet to the whole network, causing services disruption.

Among the impacts of this attack on VANET, the most critical is the potential lack of service availability. As depicted in Figure 7.3, while message modifications might cause bothersome, unwarranted traffic congestions, they simultaneously result in the absence of notifications for hazardous events. This latter scenario can create dangerous situations wherein vehicle drivers become aware of danger only when they are in close proximity, receiving no prior notifications.

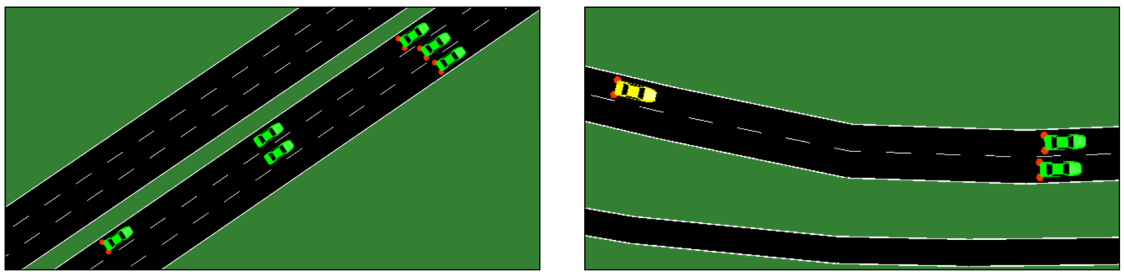

Figure 7.3. Message Modification Attack Dynamics: Distinguishing Effects on Two Resulting Events

BlackHole Attack

The BlackHole attack, akin to the Message Modification Attack, has been successfully executed within the V2X simulation environment. As in the aforementioned analyzed attack, its execution starts with the simulation of an accident and the subsequent forwarding of messages by the RSU about the hazardous event. Considering time limitations and the absence of a straightforward method to selectively transmit Decentralized Environmental Notification Messages (DENMs) solely to vehicles in forwarding contention, the chosen approach aligns with the first option proposed in section 6.3.2. Particularly, it is assumed that the attacker is the only vehicle within the range of the Roadside Unit (RSU) transmitting the DENMs. Consequently, the action of dropping by the attacker becomes unsolvable, irreversibly eliminating the circulated DENMs within the network. Therefore, like the Message Modification attack, this causes an unavailability of the network services, in particular, of the Decentralized Environmental Notification Services, which might be seriously dangerous in certain situations. For example, when the DENMs announce a hazardous location in a motorway junction that the driver will not receive, noticing the dangerous event only a few meters before instead of at the entrance in the

well-defined relevance area. This hazardous scenario, induced by the execution of the BlackHole attack, is evident in the right section of Figure 7.3, similarly triggered by the Message Modification attack.

7.2 Experimental Evaluation of Attack Feasibility in ETSI Security Framework

As detailed in section 4.2, over the years, the ETSI TC ITS WG5 working group has addressed security and privacy issues of the ITS-G5 standard, releasing several technical specifications, most of which are included in [73]. The result analysis discussed above takes into account the execution of malicious attacks in scenarios with unsecured communications. Therefore, it is very interesting to analyze the feasibility of the implemented attacks in the presence of mechanisms to secure V2X communication. This section aims to describe the conducted tests and the related results in the execution of implemented attacks on secured scenarios.

7.2.1 Sybil Attack Analysis

In a secured communication scenario, there are several realistic preconditions that can be considered concerning the execution of the Sybil Attack. Particularly, the following feasibility evaluations of implemented attacks take into account the ability of the attacker to obtain zero, one, or more valid pseudonymous certificates used to communicate in the network. The feasibility evaluations take into account the Artery simulation and Vanetza security extension implementation, based on **ETSI TS 103 097 V1.2.1** standard.

Attacker without valid certificates

The first described scenario considers an attacker that possesses no valid certificate to sign its packets and correctly communicate on the network. However, the malicious attacker can act as an outsider, trying to send messages on the network, even if they are not authorized to do so. In fact, it can send well-constructed ETSI ITS-G5 messages, hoping for software flaws that could lead to an unchecked packet signature, allowing the delivery of the attack. The execution of the implemented Sybil Attack in the described security scenario leads to a complete discarding of the unsigned messages from all the receiving stations. Indeed, the latter implement several checks on what they expect to receive. Specifically, it is first checked for the presence of a specific header and specific value in several fields. For example, the presence of "Secure Header" next to "Basic Header", the field "Payload Type", which must be strictly signed, and so on. Finally, in the "`verify_service`", verifications of digital signatures, validity of the certificate, and certification path are carried out. It is, therefore, possible to conclude that the Sybil Attack under the previously hypothesized scenario is not feasible.

Attacker with only one valid certificate

In this second scenario, it is supposed that the attacker possesses only one valid certificate usable in the considered vehicular network. So, the attacker is no longer acting as an outsider but acts as an insider since it can send signed ETSI ITS-G5 messages that are accepted by the receiving network nodes. However, the anomaly is that the attacker can

only use that Authorization Ticket to sign the broadcasted messages. In fact, since the attack consists of the forging of malicious messages that contain different identities at the application, geonetworking, and access layers, they must sign the latter only using the same pseudonym certificate. The attacker will succeed if the receiving ITS-Ss only check the identities claimed in the previously mentioned layers. Conversely, if the receiving stations evaluate the identity claimed in the "Secure Header", i.e., the pseudonym certificate or its HashedId8, an anomalous situation occurs. In fact, the ITS-Ss might detect that the received messages refer to the same sender, and they update their perceived scenario accordingly. The anomalous situation consists of the discordant coordinates contained in the messages, which lead to instant movement of the detected vehicle from one location to another, as the messages would intrinsically identify different vehicles, as shown in Figure 7.4.

It should be underlined that since the Artery V2X framework does not allow visualization of a vehicle's perception, as detailed in section 6.2.1, a specific service named "EgoServices" is implemented for this task. Indeed, the previously described operation is a logic implemented in "EgoServices" compliant with the ETSI standard. When security is enabled, it requires identification based on the pseudonym certificate, carried out only following a complete verification of digital signatures, certificates, and certification paths, instead of the use of the application identifier as StationId in CAM header. This attack is considered partially successful since its feasibility depends strictly on the implemented prevention mechanisms in every ITS-S logic, for example, the one mentioned above which makes use of the HashedId8.

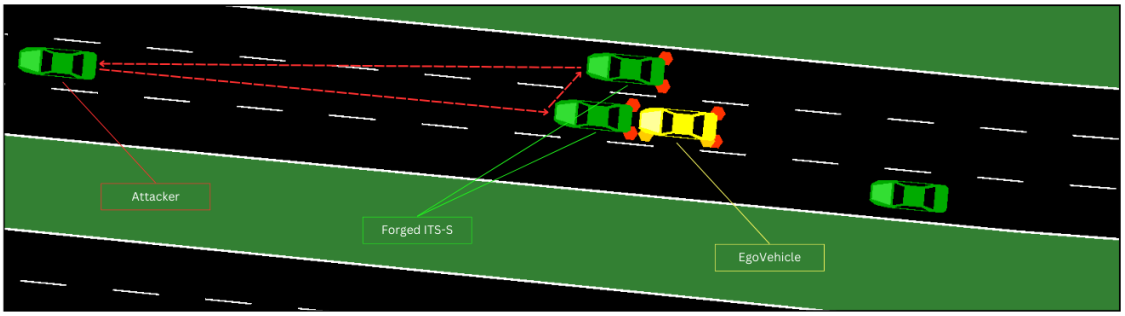

Figure 7.4. Sybil Attack in a Secure Scenario: Attacker with a Singular Pseudonym Certificate

Attacker with N valid certificates

In this third scenario that involves the Sybil attack, it is supposed that the attacker has got multiple valid certificates in a legitimate or illegitimate way, as analyzed below. In this case, the attacker signs transmitted messages using different certificates for every forged application identity, thus making it impossible to detect the attack. In fact, considering the Ego Perception traffic simulator, as it happens when the communication is not secure, every forged track is correctly verified and passed at the application level, then displayed in SUMO. It is possible to conclude that in this case, the attack is feasible.

It is worth noting that the considered hypothesis is not difficult for it to happen. In fact, leaving aside an in-depth analysis of what could be illegitimate ways to obtain a valid certificate (e.g., Stolen Private Keys/Certificates, Certificates issued by a compromised AA, etc.), as pointed out in the [86], it is not remote the scenario in which a malicious vehicle might launch a sybil attack. The technical report underlines that since

the pseudonym certificates must be changed frequently to guarantee unlinkability and avoid tracking, every ITS-S should always have access to new valid authorization tickets. This could be possible considering an on-demand issuing, but this certainly introduces latency and requires a permanent internet connection, which is not always guaranteed because of the uneven coverage of connection with the infrastructure.

The solution that overcame these problems is a pre-issuing of pseudonyms certificates (pseudonym pool) which every ITS-S saves in a secure storage and uses whenever a pseudonym change is requested. This certainly amplifies the probability that a malicious ITS-S will launch a Sybil Attack. In fact, using the authorization tickets available in the mentioned pseudonym pool, the ITS-S can easily simulate multiple fake ITS-Ss. The strength of a Sybil attack is related to the number of valid pseudonyms available at a certain time, particularly if the aim of the malicious node is to disrupt network services, i.e., causing a fake traffic jam. Although a possible countermeasure could be to reduce the number of pre-issued certificates, the dilation of time in which the same pseudonym is employed might damage the guarantee of unlinkability.

To make the evaluation of this scenario possible, the "CertificateProvider" entity, which is in charge of generating certificates for every ITS-Ss at startup, is modified. This is because, although the ETSI standard [86] considers the issuing of a pool of valid certificates, in the Artery V2X simulation framework, to simplify the implementation, every ITS-S is equipped only with one valid pseudonym certificate which is replaced with a fresh one at the moment of expiration. In particular, for the attacker vehicle, the issuing of multiple authorization tickets is forced, as shown in Figure 7.5, each associated with a generated key pair. Then, the pseudonym certificates are stored in a structure `std::vector<vanetza::security::Certificate>` and retrieved respectively from the attacker to sign the various malicious messages.

7.2.2 Replay Attack Analysis

Regarding the Replay Attack, unlike the analysis in the previous attack, in this case, there are no multiple scenarios due to the limitation of the attacker to illegitimately rebroadcast received messages. In fact, in this attack scenario, with the enablement of a secure communication paradigm, there is no need for any logic modification, as the attack simply buffers and replays received messages. The message replay is correctly carried out, as shown in Figure 7.6, which points out that the timestamp of the "Generation Time" field has a gap of about 30 seconds, although the messages are received with a gap of only 10ms from the Ego Vehicle. However, although the attack is carried out, as in the unprotected scenario, there is no impact on the Ego Vehicle. In addition to the application check about the packet expiration mentioned in section 6.2.2, there is another verification at the network layer, particularly in the "`verify_service`". Since the "`verify_service`" is responsible for the verification of digital signatures and the validity of the certificate and certification path, when it evaluates a packet with a signature generated more than 2 seconds before, according to [87], the verification will fail. In this case, the verification procedure will report the following error: "`VerificationReport::Invalid_Timestamp`" referring to the "Generation Time" being too old. However, during the execution of the attack in the described scenario, it is possible to appreciate the presence of another reported error: "`VerificationReport::Signer_Certificate_Not_Found`".

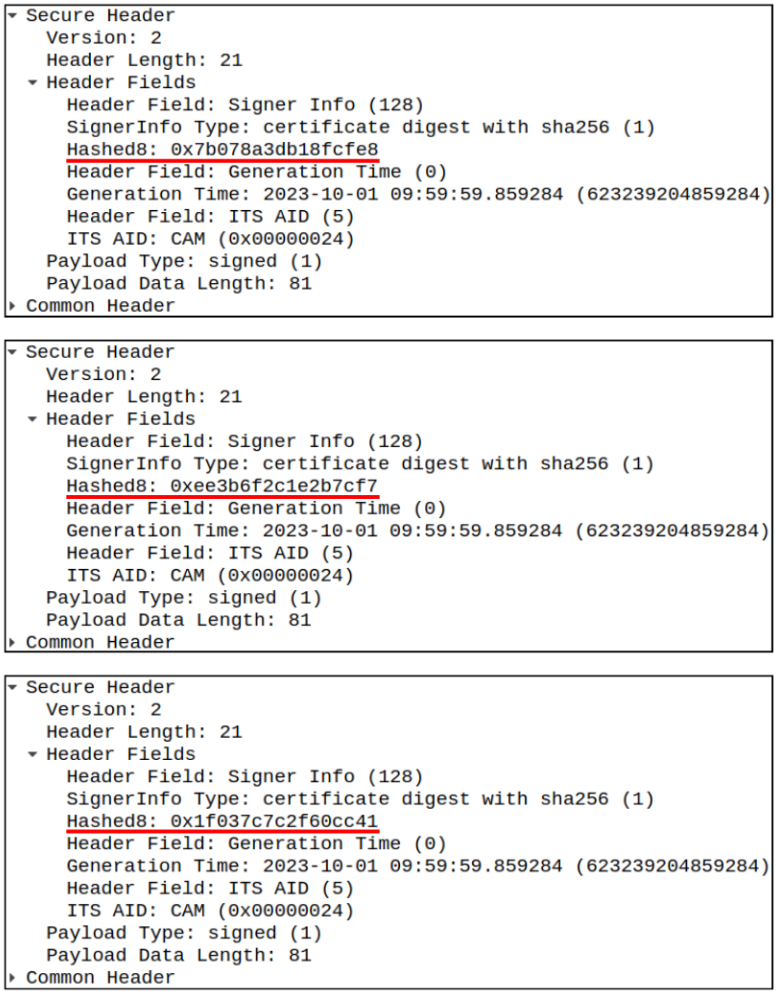

Figure 7.5. Sybil Attack in a Secure Scenario: Attacker with Multiple Pseudonym Certificates

7.2.3 Message Modification Attack Analysis

As mentioned in subsection 4.2.3, to minimize network bandwidth usage, a certificate digest is employed instead of the whole authorization ticket. This results in a smaller packet size, which might be especially useful in the case of high network density. To allow packet signature verification, every ITS-S caches the already verified certificates for only 2 seconds in the form of HashedId8. In this way, every receiving ITS-Ss can first check if the digest has been cached by using the announced HashedId8 in the packet. This ensures that the certificate has already been validated. Then, they can focus on verifying the signature to determine whether the packet has been altered during transit. However, in this specific case, since the "Generation Time" is older than the admitted 2 seconds, the process of signature verification and certificate validation terminates with the first reported error. This leads to an absence of the caching process, causing the second above-mentioned error. In fact, when the packet does not contain the whole authorization ticket but its digest (HashedId8), the "`verify_service`" carries out a cache lookup of the latter, finding the absence of a cached certificate, so communicating the `Signer Certificate is Not Found`. With these considerations, it is possible to conclude that the present attack is not feasible in the simulated secure scenario.

Concerning the Message Modification Attack, the only considered scenario is that

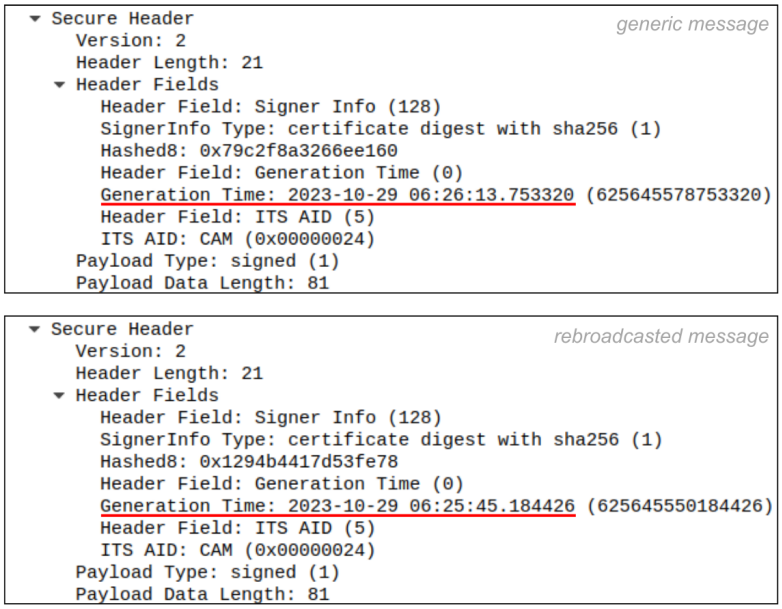

Figure 7.6. Replay Attack in a Secure Scenario: Comparing Network Packets' Timestamps

the malicious attacker acts as an insider (i.e., possessing a valid authorization ticket) and aims to interfere with multi-hop broadcasting, as described in section 6.3.1. To carry out this attack in a secured scenario, it is sufficient to use the logic described for the unsecured one. However, although the attack is carried out, the change made is not sufficient to guarantee its success. In fact, with secured communications, every ITS message is protected by a digital signature that has to be verified at every receiving station. Particularly, as indicated in [88], also intermediate stations involved in multi-hop forwarding have to verify the signature and certificate validity, thus avoiding spreading an invalid packet in the network. This contrasts with the attack at hand, which effectively aims to threaten the integrity of DENMs by delivering a modified hazardous notification. Indeed, the simulated attack is not successful since the intermediate ITS-Ss verify the packet and immediately discard it because of the incorrect signature.

Nevertheless, a possible idea to carry out the attack anyway is to update the signature and consequently also the Secure Header values, guaranteeing that the receiving ITS-Ss do not detect the wrong digital signature, thus not discarding the packet. Particularly, since the attacker is considered an insider, it possesses a valid pseudonym certificate that might be used to sign V2X packets. Within a specific pseudonym certificate are listed the permissions and privileges for that ITS-S under the form of identifiers named ITS-AID and SSP [67]. The ITS-Application Identifier (ITS-AID) indicates the overall type of permission being granted (e.g., the sender is entitled to send CAMs, DENMs, etc.), the Service Specific Permission (SSP) indicate a specific set of permission within the overall permission indicated by the ITS-AID. Whenever the attacker possesses a certificate that enables sending messages as the one which they aim to modify, the attacker might be successful. In fact, the intermediate and receiver ITS-Ss will not notice the modification since there is no trace of the previous attached signature or values. Regarding the ITS-Ss involved in the contention operation, they will abort the procedure, as described in section 6.3.1, because of the presence of a weak packet detection mechanism based only on an explicit sequence number. As shown in Figure 7.7, the attacker updates the DENM with the same change carried out in the insecure scenario. Additionally, it recomputes the signature and updates the related fields in Secure Header. As it is possible to notice,

also the coordinate reported in the Secure Header will update automatically according to the recomputation of the signature, which is no longer referred to the sender (RSU) but to the attacker.

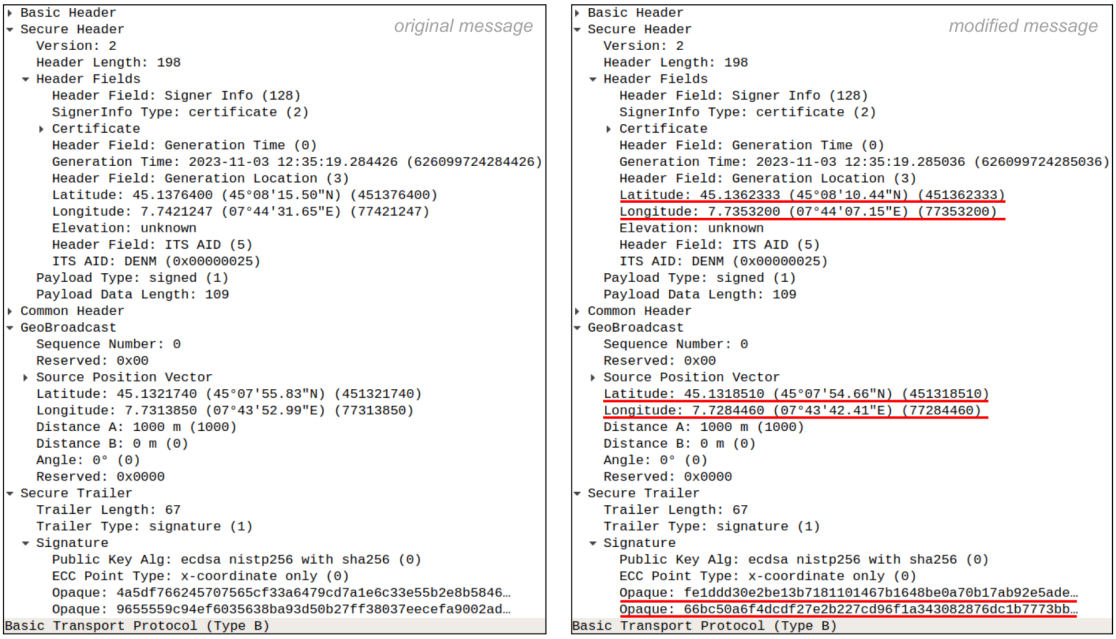

Figure 7.7. Message Modification Attack in a Secure Scenario: Comparing Network Packet Parameters

Obviously, the sender identity of the malicious messages is indirectly indicated via pseudonymization since the Enrollment authority has the possibility to link the AT to a specific EC. However, this is not to be considered a form of containment for this attack. In fact, when a malicious node will be able to obtain valid ATs in an illegitimate way, as mentioned in section 7.2.1, it will carry out the described attack without any worries about the fact that the used pseudonym is related to its identity because of the use of a compromised or illegitimate one.

Chapter 8

CohdaWireless SDK/MKx

As this thesis is being developed in collaboration with the FEV company, one of the objectives is to validate malicious attacks on the FEV HiL platform. Despite executing a malicious attack in a simulated scenario might represent a valid choice, differences in behavior still exist between the simulated ITS-S and the production devices. This chapter aims to describe the process of integrating the V2X Simulation Framework with the FEV HiL Validation Platform. It also provides details about how the V2X Simulation Framework interacts with the device used for V2X communications on the HiL Platform and whether the malicious attack remains feasible against a real device.

8.1 Attacks against Cohda Wireless VM

To better understand how to integrate the V2X simulator with the Cohda Wireless device, I decided to start by using the virtual machine provided by Cohda and made available by FEV. Specifically, Cohda Wireless offers the CohdaMobility MKx SDK, which is a self-contained virtual machine platform that allows the compilation and running of V2X applications for testing purposes before their deployment on the MKx devices. Furthermore, the virtual machine contains several example applications, including Forward Collision Warning (FCW), an application that aims to warn the driver of an impending collision by detecting stopped or slowly moving vehicles ahead, as detailed in section 2.3.

8.1.1 Reverse Engineering of CohdaWireless VM Operations

To interact with the CohdaWireless VM, the first step is to conduct a reverse engineering process of its executed operations. Specifically, by analyzing inbound and outbound network traffic, I noticed that the virtual machine sends JSON objects via a TCP socket on localhost. These objects contain various information such as time, geo-coordinates, speed, etc. In particular, these packets are directed to a specific listening application, the binary responsible for managing V2X communications, and they are thought to emulate the service provided by the so-called GPSD. In fact, GPSD is a daemon that collects data from a Global Positioning System (GPS) receiver and provides the data via an IP network. While GPSD is typically used in MKs devices to provide access to GPS data to the running application (e.g., "exampleETSI"), the CohdaWireless Virtual Machine, lacking a GNSS receiver, emulates this service using a local Traffic Simulator application. The transmitted JSON objects, shown below, are named time-position-velocity (TPV) and are collected from the application responsible for managing V2X communications.

Regarding the European V2X standard, "exampleETSI" is the designed application used for transmitting ETSI messages such as CAM, DENM, etc., which are filled out using the collected TPV objects. Therefore, since our objective was to ensure that the application acts as if it receives real GNSS information, I developed a service named "EgoGNSSService", mentioned in section 6.2.1, which is responsible for solving this problem. Specifically, the service takes, at a frequency of 10Hz, useful information from the SUMO traffic simulator instance, fills out TPV objects, and sends them to a specific port on which exampleETSI application is listening. This approach ensures that the virtual machine is completely immersed in the virtual scenario, enabling it to perceive the environment as entirely authentic.

```
{
    "class":"TPV",
    "device":"/proc/self/fd/0",
    "mode":3,"time":"2023-07-05T09:50:37.660Z",
    "ept":0.005,
    "lat":48.225053833,
    "lon":16.424982167,
    "alt":0.001,
    "epv":46.920,
    "track":317.7000,
    "speed":13.890,
    "sep":43.300
}
```

As evidence of this, executing the exampleETSI application and providing the GNSS information through the designed EgoGNSSService, the virtual machine starts sending CAMs and DENMs as it happens if a physical CohdaWireless device is used. Specifically, the generated messages by CA and DEN services contain information coherent with the virtual Ego Vehicle instance of the simulation environment.

Once the Virtual Machine operates as if it were totally within the simulated scenario, the next step is to deepen a plausible interaction between the latter and the V2X Simulation Framework. Firstly, it is important to consider a specific service responsible for exporting the messages generated within the V2X Simulator. Indeed, this will allow interaction with the Cohda Wireless VM precisely by using the object of the attack, i.e., V2X messages generated in simulation. The designed service is CAExportServices, already described in section 6.2.1.

By dissecting the network packets generated from the CohdaWireless VM, I noticed that the exampleETSI encapsulates the entire V2X PDU inside a UDP datagram, as shown in Figure 8.1. Since our objective is to ensure that the Virtual Machine can properly process the messages generated and exported by the used V2X Simulator, I implemented the exportation of the same exact header sequence encapsulated in a UDP datagram. Then, since the only feedback from the VM about the processing of received packets is the triggering of warnings by running applications, I decided to enable one of these. Particularly, I chose Forward Collision Warning and enabled it by declaring in the ITS-S configuration file "obu.conf" the following: "`Cohda_App_FCW.ENABLE = 1;`".

Regarding the feedback (i.e., warning) sent from the mentioned application, it is dedicated a section of "obu.conf". It is referred to as **Human Machine Interface (HMI)** settings, and it allows choosing the destination IP, destination port, and network interface used to transmit the triggered UDP warning messages. At this point, I properly

```
▸ Frame 49: 464 bytes on wire (3712 bits), 464 bytes captured (3712 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_20:4e:19 (08:00:27:20:4e:19), Dst: IPv4mcast_01:00:01 (01:00:5e:01:00:01)
▸ Internet Protocol Version 4, Src: 192.168.1.17, Dst: 224.1.0.1
▸ User Datagram Protocol, Src Port: 34997, Dst Port: 34997
▸ Cohda Wireless proprietary
▸ IEEE 802.11 QoS Data, Flags: ........
▸ Logical-Link Control (LPD)
▸ GeoNetworking_CW: Secured (GeoBroadcast Circle)
▸ Basic Transport Protocol (Type B)
▸ ETSI ITS (DENM)
```

Figure 8.1. ITS-G5 Packet Headers from CohdaWireless Virtual Machine.

configure the HMI and test if the CohdaWireless process correctly processes the messages coming from the V2X Simulator. I test it using a fixed scenario in which are transmitted:

- Fixed GNSS information via TPV object used by the VM to perceive its geographical information.
- Fixed CAMs generated from the V2X simulator, which claim geo-coordinates, speed, altitude, heading, etc., with values identical to those of a vehicle ahead of the VM.

The result is, as expected, a continuous triggering of warnings by the FCW application via designed UDP datagrams. The warnings are in the form of XML, as shown below, and report important information such as the distance from the vehicle ahead and a related severity. This provides evidence regarding the correct processing of the exported V2X messages done by the exampleETSI application, therefore providing the opportunity to execute the already developed malicious attacks. In this regard, the execution of an attack against the Cohda Wireless ETSI application running on the VM is described below.

```
<ui_request>
    <app_id>1</app_id>
    <app_type>V2V</app_type>
    <severity>95</severity>
    <tti_ms>450</tti_ms>
    <range>13.16</range>
    <icon>./ui/FCW/FCW_high.gif</icon>
    <alt_icon>./ui/FCW/FCW_high.gif</alt_icon>
    <icon_rate_sec>2</icon_rate_sec>
    <text>Forward Collision Ahead</text>
    <audio>./ui/FCW/FCW.wav</audio>
</ui_request>
```

8.1.2 Malicious Attack Feasibility Validation

Once the guarantee is obtained that the VM succeeds in processing fixed-use-case messages exported by the V2X simulator, it is possible to evaluate the reaction of the VM under the developed attack scenarios.

Since the attack scenarios are based on both Cooperative Awareness Messages (CAMs) and Decentralized Environmental Notification Messages (DENMs), and since the focus of this feasibility analysis is confined to the Forward Collision Warning (FCW) application, which processes exclusively CAMs, only the Sybil Attack and Replay Attack are considered.

As detailed in section 6.2, both Sybil and Replay attacks aim to set up a similar scenario in which the Ego Vehicle is targeted via malicious messages claiming the presence of a vehicle ahead involved in emergency braking or traveling at a low speed. Although both aim to induce the same reaction in the Ego Vehicle, the attacks execute different mechanisms. In the Sybil attack, messages are forged, whereas in the replay, they are simply buffered and then rebroadcasted. This highlights the fact that the success of the attacks is not dependent, and it is worth evaluating the feasibility of both.

Regarding the Sybil Attack, the conducted test highlights that the attack is feasible both in a simulated environment, as detailed in section 7.1, and against software used in production (i.e., exampleETSI). During the execution of the developed malicious attack scenario, all virtual vehicles, including the attacker, produce their CAMs, which are exported outside the simulated environment. All the CAMs, as well as the malicious one, are correctly received by the Cohda Wireless Virtual Machine and processed at a high level by the running applications. In particular, the enabled FCW detects, through the malicious CAMs, the presence of an imminent danger and triggers the FCW warning. In this regard, it can be underlined that the simulated Sybil attack can have an effect also against production software.

The Replay attack, instead, as mentioned before, is based on another approach that involves messages with an expired timestamp rather than forged messages. In fact, it is worth noting that, as detailed in section 7.1.1, although the Sybil attack succeeds on the V2X simulator, the Replay attack is not feasible due to the dropping of packets with expired timestamps by receiving ITS-Ss. Conversely, it seems strangely feasible on the Cohda Wireless VM, although the transmitted packets are undoubtedly no longer valid. It is possible to deduce that there is a software flaw in the timestamp evaluation that allows the upward forwarding of expired messages in every Cohda Wireless product (i.e., both VM and MKs devices). To support this finding, it is conduct a test with decreasing timestamp packets, in order to evaluate the expiration gap within which the VM accepts the latter as valid. However, it is possible to conclude that in the Cohda Wireless application logic, there is no limit on how long an expired packet has been, it is processed anyway. This has allowed the successful execution of the Replay Attack with the consequent triggering of warnings from the FCW application.

8.2 Attacks against CohdaWireless MKx

To integrate the V2X Simulation Framework with the **FEV Hardware-in-the-Loop (HiL)** Platform, a transition from the Cohda Wireless VM to the physical device is necessary. In this section, the differences between the Cohda Wireless MKx and the Cohda Wireless VM, the necessary software modifications, and the discovered limitations are detailed.

The first discovered limitation is the impossibility of communicating directly with the MKx devices through a wired channel, thus bypassing the wireless interface (i.e., the IEEE 802.11p radio). The idea is to transmit packets generated from the V2X simulator to the physical device using the MKx Ethernet interface. In this way, it is possible to make the interaction less complicated and avoid the use of Software-Defined Radio (SDR) and dedicated hardware, which may be beyond the scope of this thesis work. Particularly, the idea is to obtain the processing of messages by MKx as if they were coming from the radio interface but instead injecting them through the MKx Ethernet interface.

Although this seems initially not possible, a potential solution emerged, but it introduces some limitations. The MKx, if specifically configured, allows the processing of

messages coming from the wired Ethernet connection but allows only the processing of the Facility layer. This necessarily prevents us from considering the lower layers, especially the GeoNetworking layer, which includes security measures to protect V2X packets, as discussed in section 4.2. Figure 8.2 shows there is a direct transmission of the Facility Layer Protocol Data Unit (PDU) only, which, after proper configuration, allows the MKx to process messages directly. The configuration consists of changing the communication profile from `0xff` values, which indicate the use of ITS radio, to `0x20` to indicate the use of the Ethernet interface (`ItsFacilitiesDefaultCommProfile = 0x20;`).

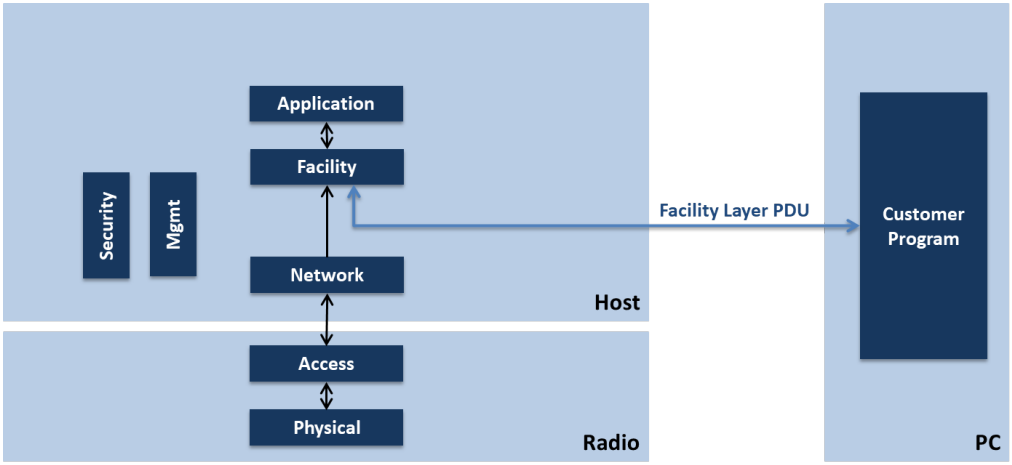

Figure 8.2. Seamless Direct Transmission of Facility Layer PDU to CohdaWireless Device.

Although confining the content to the facility layer PDU may seem limiting, it is worth noting that since the security implemented in Vanetza refers to `ETSI TS 103 097 V1.2.1` [72] while the Cohda Wireless products are updated to V1.4.1, there is no possibility to validate the attack in a secured scenario. Some alternatives have been evaluated to bypass this problem, but they were not carried out due to time constraints. One of these is to develop an update for the Vanetza security implementation porting the implemented standard from `V1.2.1` to the new `ETSI TS 103 097 V1.4.1`. Another possibility, although not very advisable, would have been to use an outdated virtual machine. In this specific case, even if it is possible to validate the attacks, there are no certainties as to whether this would have an effect on current production devices.

Another limitation worth mentioning is the use of self-signed certificates. While these certificates are acceptable for simulated scenarios, they need to be replaced in case of an upgrade of the Vanetza security. In fact, to allow correct signature and certificate verification, certificates issued from an AA belonging to a trusted PKI should be used. The receiving CohdaWireless device, using the European Certificate Trust Lists (ECTL), which is provided by the C-ITS Point of Contact (CPOC), can check if that specific PKI is trusted, so verify the certificate and the whole certification chain.

8.2.1 FEV HiL Validation Platform – Integration Architecture

Hardware-in-the-loop (HiL) testing is a technique employed in the automotive industry and beyond to test complex embedded systems. HiL simulation provides a testing platform that enables reproducible testing of ECUs under simulated real-world conditions. It is extremely necessary since it allows the testing of critical corner cases without safety issues for the device-under-test (DUT) and testing operators. Additionally, it must be

emphasized that it enables the DUT to be tested in myriad scenarios, often not easily implemented and certainly not deterministic in execution.

In the context of the V2X domain, HiL testing allows, in a cost-efficient manner, the execution of high-density and complex traffic scenarios, such as testing traffic jam avoidance applications before deployment. Of particular interest is the execution of realistic V2X malicious attacks, which, falling within the sphere of safety-critical tests, should necessarily be carried out in a protected environment. This forms the foundation of the concept of integration between the V2X simulator, upon which malicious attacks have been developed, and the FEV V2X HiL platform.

Despite the aforementioned limitation regarding the absence of the GeoNetworking layer, it is interesting to consider the integration of the V2X simulator with the FEV HiL Validation Platform. This allows for Malicious Attack Feasibility Validation in an undemanding case, which can be complicated through the consideration of the GeoNetworking layer, after implementing the appropriate changes to the V2X Simulator.

Before the integration, it is necessary to underline the fact that the EgoGNSSServices developed for the CohdaWireless VM must be modified, as the physical devices adopt a different representation of GNSS information. Specifically, to operate the MKx devices require GNSS data in the form of GPGGA, GPGSA, and GPRMC strings, according to the NMEA 0183 standard. Once full interaction between the V2X simulator and the Cohda Wireless device is achieved, it is possible to consider the proposed integration architecture, which allows obtaining a V2X validation platform no longer tied to physical devices, with the exception of the one included in the vehicle under test (VUT). The following architecture, shown in Figure 8.3, presents an interaction between the several involved modules and devices that has already been touched on in the previous sections.

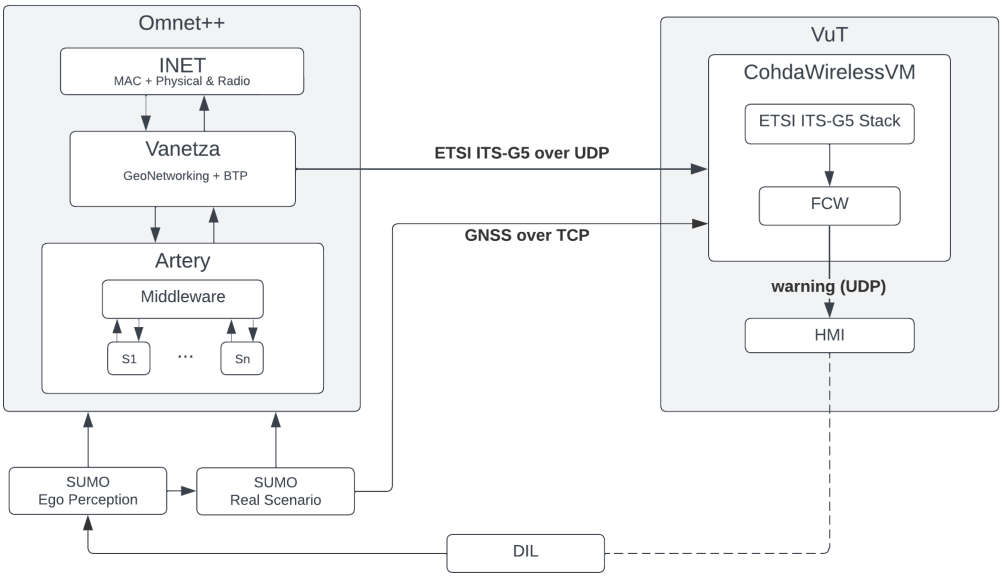

Figure 8.3. Integration Architecture between V2X Simulator and FEV HiL Platform.

Specifically, on the left side, there is the V2X simulation domain, which includes the OMNeT++ network simulator, the Artery V2X simulation framework based on the Vanetza open-source library, and multiple instances of the Sumo Traffic Simulator. These software modules interact in a manner that is detailed both in chapter 5 and in chapter 6. On the right side, there is the so-called vehicle under test (VUT), the hardware that has

to be tested and validated. As one would anticipate, since we have thoroughly described the interaction with Cohda Wireless devices, the VUT we are going to consider will be equipped with a Cohda Wireless MKx, coinciding with the behavior of the virtual machine detailed in subsection 8.1.1. Particularly, the reported HMI module indicates the Human Machine Interface, which is in charge of showing the driver the warnings triggered from FCW. Additionally, in the shown architecture, the so-called Driver-in-the-Loop is reported, which interprets the warning and acts on the traffic simulator, closing the loop.

The interaction between the Simulation domain and the VUT domain consists of a TCP Socket to transmit Virtual Ego Vehicle GNSS data and a UDP Socket to transmit V2X messages exchanged by vehicles on the simulated vehicular ad-hoc network. Both virtual GNSS data and V2X messages serve to achieve a complete belief in the VUT, reacting to real-world driving conditions on the open road. Therefore, it is possible to validate the malicious attack scenarios over production equipment and indirectly obtain an HiL Platform, which, among the various operations, is capable of testing vehicles against real-case scenario attacks.

8.3 Fuzz Testing on CohdaWireless VM

As the last objective of this thesis, the experimentation focuses on aspects that could be part of future works in the domain of vulnerability assessment and penetration testing. In particular, this section aims to describe the approach and obtained results of a black-box fuzz testing conducted against the Cohda Wireless "exampleETSI" application.

As detailed in the first chapter, the continuously increasing on-board connectivity and cyber-physical systems pose a high risk concerning the security and safety of modern vehicles. The million lines of code represent the most significant threat due to the substantial likelihood of vulnerabilities that could enable remote control of vehicles. As learned from reported famous case studies on the remote compromise of safety-critical ECUs in unaltered passenger vehicles, the most targeted entry point is the telecommunication control unit (T-Box). It is typically in charge of handling almost all communication in the in-vehicle network with the outside world. Regarding this, as we are dealing with Cohda Wireless devices responsible for handling V2X communication in this case, we must note that these devices could be targeted in the near future, precisely to exploit the V2X channel as a possible entry point for an in-vehicle attack. Consequently, as it is important to evaluate the security of this device against potentially unknown software flaws or vulnerabilities, black-box fuzz testing is carried out specifically on the application responsible for executing the European variant of V2X communication standard, i.e., ETSI ITS-G5.

Fuzz testing is a dynamic analysis technique that aims to force an application crash by injecting malformed or invalid inputs. Typically, the inputs are automatically generated because of the immense amount necessary to cover the entire application code. Indeed, this is a significant problem because it is not possible to generate all possible inputs, given their number grows exponentially. Hence, under certain conditions, coverage-guided fuzzing (i.e., greybox fuzzing) is adopted, which, using program instrumentation, can evaluate code coverage. This technique involves inserting instrumentation code into a program, which is recompiled and then executed using the fuzzer. In this way, the fuzzer can track execution flow changes with every input and use this information to make finer decisions about how to mutate the input, thus maximizing coverage.

However, this approach is valid only for programs for which the source code is available. In the case of the Cohda Wireless exampleETSI application, the entire source code is not available, and the only plausible option is to conduct black-box fuzzing. In black-box fuzzing, inputs are generated without any knowledge of the internal behavior or control flow, and for these reasons, it has worse performance compared to the instrumented one.

In addition to the coverage-guided approach described above, the fuzzers also differ based on the inputs they generate.

- The generation-based fuzzer generates inputs from scratch.
- The model-based fuzzer generates input starting from a formal representation of the latter.
- The mutational fuzzer requires examples of valid inputs and, starting from them, generates mutated inputs.

Finally, the fuzzers are also categorized based on the complexity of the executed transformation. Particularly, there are dumb fuzzers that execute generic input transformations and smart fuzzers that use specific analysis tool outputs to generate new valid input with an awareness of the underlying data structure.

After this introduction to the objectives of fuzz testing and the parameters that characterize the various fuzzers, let's delve deeper into the operations performed on the Cohda Wireless Virtual Machine.

8.3.1 American Fuzzy Lop

American Fuzzy Lop is a brute-force fuzzer and is considered one of the most valid fuzzers since it is used to discover a huge number of vulnerabilities (more than 300) according to [89]. AFL is coupled with an instrumentation-guided genetic algorithm, which can greatly improve the final result. However, as mentioned above, the unavailability of the source code of the system under test (SUT) makes it not possible to consider an instrumented approach. Regarding the generated input, AFL is considered a dumb fuzzer and uses a mutational-based approach starting from an input example. In particular, the mutations applied by AFL are mostly agnostic to the input format of the target program. It firstly applies a deterministic sequence of mutations to each input, e.g., flipping, overwriting parts of the input with interesting values, or incrementing and decrementing integers. Then, it moves on to havoc, a special stage in which multiple other mutations are applied, such as overwriting bytes with random values, deleting or duplicating blocks, etc.

Since the SUT accepts input sent over the network, to execute the fuzz testing, it was necessary to use a variant of the AFL project [90] that allows the sending of network packets instead of providing them via an input file or stdin. In addition, to avoid the focus on malformed packets at a low level, the tool allows the specification of the transport protocol, thus focusing on fuzzing only the application layer, which is the one processed by the SUT.

The fuzzer correctly executes the exampleETSI application and starts to transmit fuzzed input by mutating the provided example packets. The fuzzing procedures have not revealed any unexpected crashes or abnormal warnings on more than 450k malformed packets.

8.3.2 Radamsa

Radamsa is a brute-force fuzzer and, like AFL, it is considered one of the most used fuzzers since it has already found a slew of important vulnerabilities (more than 90) according to [91]. Radamsa is epithetized as an extremely "black-box" fuzzer since it doesn't need any information about the program input. This is particularly interesting since the highly structured input format of ETSI CAMs and DENMs might make it complex to construct a model. It is intended to be a good general-purpose fuzzer with a focus only on finding issues, without interest in what kind of data the program processes. The characteristics described highlight the fact that Radamsa is also a dumb fuzzer. Like AFL, it uses a mutational-based approach, which aims to apply heuristics and change patterns to the received inputs. In particular, it does not have a predefined schema; in fact, it is possible to make just one change or a slew of them, such as bit flips and more advanced mutations.

As mentioned before for the AFL fuzzer, Radamsa also requires sending the input over the network since the listening SUT waits for packets on a specific multicast address. Therefore, an open-source project [92] has been employed, which, with a man-in-the-middle (MITM) approach, allows the fuzzing of network packets on-the-fly, acting as a proxy. In detail, as shown in Figure 8.4, the idea is to employ two CohdaWireless Virtual Machines. The first acts as a packet generator, and the second represents the SUT. Particularly, it exploits the packet flow generated by the first instance of VM on which the exampleETSI application is running. Then, using an `iptables` rule, the packets become addressed to the `NFQUEUE` target from which they are subsequently collected by the `fuzzor.py` program. The latter is in charge of handling the mutation of packets from valid to invalid ones through the use of the integrated Radamsa fuzzer. In this process, an invalid flow is obtained by starting from a valid flow of ITS packets. Then it is used to test the SUT on which the exampleETSI application is running.

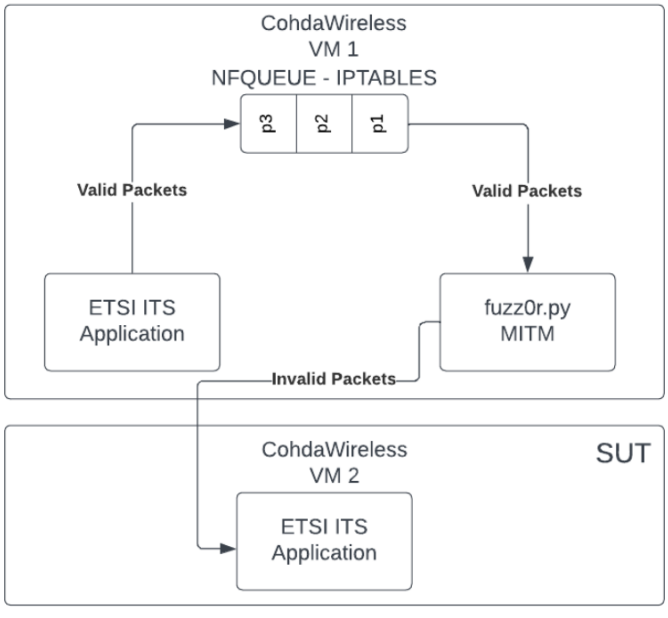

Figure 8.4. MitM Fuzz Testing Architecture Employing Radamsa.

8.3.3 Results Analysis

At the end of this fuzz testing conducted with the fuzzers American Fuzzy Lop and Radamsa, it is possible to analyze the results. An important aspect to take into account is the limited feedback from the SUT. The only available feedback includes the terminal, which reports a possible crash of the application, and, on the other hand, the generated ITS packets which lead us to deduce that the application is running. This, added to the fact that it was not possible to adopt an instrumentation-guided approach, ends up classifying this testing as a completely black-box fuzz testing. It follows that the only viable strategy is to send malformed or invalid input, as shown in Figure 8.5 and Figure 8.6, hoping for a possible application crash which reveals software flaws or vulnerabilities.

449909	18637.311596	172.20.10.3	224.1.0.1	802.11	52	Unrecognized (Reserved frame)[Malformed Packet]
449908	18637.229403	172.20.10.3	224.1.0.1	Cohda	44	[Malformed Packet]
449907	18637.202458	172.20.10.3	224.1.0.1	Cohda	44	[Malformed Packet]
449906	18637.171395	172.20.10.3	224.1.0.1	Cohda	43	[Malformed Packet]
449905	18637.128386	c3:bf:08:00:27…	Harris_bf:c3:bf	802.11	216	Data, SN=3130, FN=10, Flags=op....F.
449904	18637.065198	c3:bf:08:00:27…	Harris_bf:c3:bf	802.11	353	Data, SN=3130, FN=10, Flags=op....F.
449903	18636.975820	c2:a8:c2:a3:20…	23:ba:00:00:80…	802.11	104	Data, SN=0, FN=0, Flags=op....F.
449902	18636.935817	172.20.10.3	224.1.0.1	Cohda	43	[Malformed Packet]

Figure 8.5. AFL-Generated Malformed Packets.

108922	2997.960879	28:c3:6a:c2:96…	c3:bf:7f:00:71…	802.11	174	Association Request, SN=1600, FN=0, Flags=.pmPRMFT
108921	2997.857616	172.20.10.3	224.1.0.1	802.11	221	Unrecognized (Reserved frame), Flags=op....FT
108920	2997.752246	c3:bf:08:00:27…	Harris_bf:c3:bf	802.11	212	Data, SN=3130, FN=10, Flags=op....F.
108919	2997.651951	172.20.10.3	224.1.0.1	Cohda	43	
108918	2997.550368	172.20.10.3	224.1.0.1	Cohda	43	
108917	2997.442404	172.20.10.3	224.1.0.1	802.11	92	Unrecognized (Reserved frame), Flags=op.PRMFT
108916	2997.337759	f5:f5:f5:f5:f5…	47:11:f5:f5:f5…	802.11	218	Association Request, SN=560, FN=0, Flags=........
108914	2997.234556	172.20.10.3	224.1.0.1	802.11	223	Unrecognized (Reserved frame), Flags=op....F.

Figure 8.6. Radamsa-Generated Malformed Packets.

However, in the conducted fuzz testing, after more than 500k malformed/invalid packets sent to the exampleETSI application, no anomalies were revealed. Particularly, no malfunctions or crashes were detected, and since a non-instrumented approach is used, it is not possible to relate the result to the code coverage and therefore understand whether the test has been sufficient.

Nevertheless, it is worth noting that since Fuzz Testing is an incredibly low-effort testing methodology, it is strongly possible to consider that the Cohda Wireless application (i.e., exampleETSI) has already gone through several testing phases that have adopted fuzzed inputs. It follows that the result obtained, which has not detected any anomalies, is consistent with the fact that the SUT is an application in production for several years.

Chapter 9

Conclusion and Future Work

In conclusion, this dissertation has addressed multiple objectives within the domain of inter-vehicle communications cybersecurity, resulting in diverse outcomes and experimental findings summarized below.

The opening chapters delineate how the in-vehicle architecture of modern vehicles has become increasingly vulnerable to remote cyber-physical attacks due to a broad attack surface and the multitude of in-vehicle features. The low priority of cybersecurity within the automotive industry is evident through numerous security weaknesses enabling successful remote attacks on unaltered passenger vehicles, as highlighted by researchers over the last decade. The advent of Vehicle-To-Everything (V2X) communication technologies raises concerns about potential VANET threats that, while confined to the vehicular network, can significantly impact passenger safety.

The core of the thesis focuses on V2X threats, detailing the design and development of several malicious V2X attacks identified as most pertinent from the initial survey. These attacks comply with the ETSI ITS-G5 specification and target both V2V and V2I communication paradigms by exploiting CAMs and DENMs. Subsequently, a discussion is presented on the feasibility of implementing these attacks in both unsecured and secured scenarios, revealing the success of certain attacks. Results highlight the feasibility of Sybil and blackhole attacks independently of the presence of security measures. However, replay and message modification attacks are prevented by the security mechanisms, enabling the ITS-S to respectively detect an expired signature and a tampered signature.

A significant challenge of this thesis has been integrating the Artery V2X simulation framework into the FEV Hardware-in-the-Loop (HiL) Validation Platform. This integration has required a valid architecture to allow platform components (e.g., Cohda Wireless MK5) to properly interact with the newly introduced components. In addition to obtaining an advanced HiL platform, this integration has enabled the validation of simulated malicious attacks on one of the most commonly used V2X OBUs for automotive rapid prototyping. Results indicate the feasibility of both Sybil and replay attacks against Cohda Wireless software, demonstrating vulnerabilities to real-world V2X attacks on vehicles equipped with these devices. These attacks succeed in triggering warnings from the Cohda Wireless Forward Collision Warning (FCW) application by sending both malicious and expired V2X messages. Particularly, expired messages are accepted due to identified flawed software implementation of the `exampleETSI`, the production executable of the European V2X standard for Cohda Wireless devices.

Ultimately, a comprehensive assessment of other software defects and vulnerabilities is conducted through fuzz testing against the `exampleETSI` application, employing two

widely used fuzzers, American Fuzzy Lop, and Radamsa. Utilizing a completely black-box approach due to the impossibility of code instrumentation and limited feedback from the System Under Test (SUT), after generating 500k invalid and malformed packets, no crashes or exceptions were observed in the results.

While this thesis has explored various aspects of inter-vehicle communications cyber-security, several opportunities for future development remain unexplored. One potential avenue involves augmenting the implementation of the Vanetza security module utilized in the Artery V2X Simulation Framework. As outlined in chapters 4 and 7, the current implementation complies with an outdated security standard (ETSI TS 103 097 v1.2.1), confining interaction with real-world devices in unsecured scenarios only, as demonstrated in this thesis. In addition to updating the security implementation to ETSI TS 103 097 v1.4.1 specification, it is also possible to introduce missing features, such as PKI communication between ITS-S and both EA and AA.

Another promising area for future development lies within the Vulnerability Assessment (VA) domain. Consideration can be given to Dynamic Application Security Testing (DAST) employing advanced tools such as Driller proposed by Stephens et al. in [93], which combines fuzzing and concolic execution to discover deep bugs. However, it's important to note potential weaknesses of this approach, such as incompleteness and path explosion. In this regard, another viable option is Static Application Security Testing (SAST) through binary code analysis (BCA) of the Cohda Wireless firmware and the `exampleETSI` application. Although it is a time-consuming testing technique, it may reveal additional vulnerabilities and provide precise information compared to DAST.

The final proposed area for future work falls within the domain of penetration testing. It involves developing a proof-of-concept (PoC) exploit chain on a contemporary vehicle equipped with an already identified vulnerable Cohda Wireless device. While similar case studies have been conducted over the past decade, as mentioned in chapter 3, this proposal aims to exploit a V2X entry point that has not been leveraged in any remote vehicle compromise case studies so far.

It's crucial to emphasize that while only a few consumer vehicles are currently equipped with Telematics Control Unit (TCU) capable of inter-vehicular communication (IVC), there would be no immediate impacts as V2X services are presently inactive. However, with their widespread deployment, the associated risks of potential attacks exploiting this new communication channel cannot be disregarded.

Appendix A

Installation Guide

The following manual is designed to assist users in installing the necessary software to execute the simulated malicious attacks presented in this dissertation. The execution requires a Linux Operating System. Specifically, this manual focuses on Ubuntu 20.04 LTS. It is possible to use a Linux OS on bare metal or employ virtualization on a different OS, e.g., VirtualBox on Windows.

The manual is divided into three sections based on the required dependencies. Since the Artery V2X Simulation Framework employed for the implementation and execution of malicious attacks is based on OMNeT++ and requires interaction with SUMO Traffic Simulator, the complete installation of the mentioned software is detailed.

The installation of the Ubuntu Operating System and/or virtualization software is not detailed herein.

A.1 OMNeT++

As outlined in [94], OMNeT++ necessitates several packages to be installed on the computer. These packages include the C++ compiler (gcc or clang) and several other libraries and programs. Generally, superuser permissions are required to install packages.

Note: It is recommended to adhere to the 5.6 release of OMNeT++ when using Artery for now, as other versions may not be compatible.

A.1.1 Installing the Prerequisite Packages

Before initiating the installation, update the database of available packages by entering the following command in the terminal:

```
$ sudo apt-get -y update
```

Then, to install the required packages, type the following command in the terminal:

```
$ sudo apt-get -y install \
build-essential \
clang \
lld \
gdb \
bison \
```

```
flex \
perl \
python3 \
python3-pip \
qtbase5-dev \
qtchooser \
qt5-qmake \
qtbase5-dev-tools \
libqt5opengl5-dev \
libxml2-dev \
zlib1g-dev \
doxygen \
graphviz \
libwebkit2gtk-4.0-37

$ python3 -m pip install --user --upgrade \
numpy \
pandas \
matplotlib \
scipy \
seaborn \
posix_ipc
```

Additionally, to use Qtenv with 3D visualization support, it is necessary to install the development packages for `Open-SceneGraph (3.4 or later)` and the `osgEarth (2.9 or later)` packages.

```
$ sudo apt-get -y install \
openscenegraph-plugin-osgearth \
libosgearth-dev
```

A.1.2 Downloading and Unpacking

Begin by downloading OMNeT++ from https://omnetpp.org, ensuring the download of the Linux-specific archive, i.e., `omnetpp-5.6.3-src-linux.tgz`. Copy the archive to the directory where you wish to install it, e.g., `/home/<yourDirectory>`. Then, open a terminal and extract the archive using the following command:

```
$ tar xvfz omnetpp-5.6.3-src-linux.tgz
```

This command creates an `omnetpp-5.6.3` subdirectory containing the OMNeT++ files.

A.1.3 Environment Variables

For OMNeT++ to function properly, the `omnetpp-5.6.3/bin` directory needs to be in the PATH. Temporarily add it using the following commands:

```
$ cd omnetpp-5.6.3
$ source setenv
```

To set the environment variables permanently, consider editing .profile or .zprofile in your home directory and adding a line like the following:

```
[ -f "\$HOME/omnetpp-5.6.3/setenv" ] && source
   "\$HOME/omnetpp-5.6.3/setenv"
```

Alternatively, if no shells other than bash are used, modify .bashrc by adding the following line at the end of the file:

```
export PATH=\$HOME/omnetpp-5.6.3/bin:\$PATH
```

Then, restart the machine for the changes to take effect.

A.1.4 Configuring and Building OMNeT++

In the top-level OMNeT++ directory (i.e., `/omnetpp-5.6.3`), enter the following command:

```
$ ./configure
```

The configure script detects installed software and the configuration of your system, writing the results into the Makefile.inc file, which will be read by the makefiles during the build process.

Once `./configure` has finished, compile OMNeT++ by typing in the terminal:

```
$ make
```

A.1.5 Verifying the Installation

To ensure the installation is correct, run a sample simulation as follows:

```
$ cd samples/aloha
$ ./aloha
```

A.1.6 Starting the IDE

To launch the OMNeT++ Simulation IDE, utilize the following command:

```
$ omnetpp
```

A.2 Simulation of Urban Mobility (SUMO)

As the second required software, Eclipse SUMO Traffic Simulator is responsible for simulating the road network used within the Artery Framework. This section details the installation of SUMO on a Linux OS from sources.

Note: A version 1.0 or later of SUMO is necessary, as the TraCI protocol of earlier versions is incompatible with Artery.

A.2.1 Installing Required Tools and Libraries

To install all the necessary tools and libraries, execute the following command:

```
$ sudo apt-get install git \
cmake \
python3 \
g++ \
libxerces-c-dev \
libfox-1.6-dev \
libgdal-dev \
libproj-dev \
libgl2ps-dev \
python3-dev \
swig \
default-jdk \
maven \
libeigen3-dev
```

Additionally, although not necessary, the following packages can be useful-for instance, for speeding up builds or using the experimental 3D GUI:

```
$ sudo apt-get install ccache \
libavformat-dev \
libswscale-dev \
libopenscenegraph-dev \
python3-pip \
python3-setuptools

$sudo apt-get install libgtest-dev \
gettext \
tkdiff \
xvfb \
flake8 \
astyle \
python3-autopep8

$pip3 install texttest
```

For the Python tools, there are specific requirements depending on the tools you intend to use. To install the most common dependencies, use this command:

```
$sudo apt-get install python3-pandas
python3-rtree \
python3-pyproj
```

A.2.2 Obtaining the Source Code

For setting `SUMO_HOME` correctly, the path where SUMO is built is essential. Retrieve the SUMO build path by using the `pwd` command after obtaining the source code:

```
$ git clone --recursive https://github.com/eclipse-sumo/sumo
$ cd sumo
```

```
$ git fetch origin refs/replace/*:refs/replace/*
$ pwd
```

Assuming SUMO is placed in the folder `/home/<user>/sumo-<version>`, to define the variable for the current session only, use:

```
$ export SUMO_HOME="/home/<user>/sumo-<version>"
```

For setting the environment variables permanently, follow the same approach as described in subsection A.1.3.

Tools invoked from netedit require a list of Python packages to generate templates during compilation. Install them as follows:

```
$ sudo apt-get install python3-pyproj
python3-rtree \
python3-pandas \
python3-flake8 \
python3-autopep8 \
python3-scipy \
python3-pulp \
python3-ezdxf

pip install -r tools/requirements.txt
```

A.2.3 Building and Installing the SUMO Binaries

The following describes the compilation of SUMO using cmake. Create a build folder for cmake (in the SUMO root folder) and configure SUMO with the full set of available options, such as GDAL and OpenSceneGraph support, if the libraries are installed:

```
$ cmake -B build .
```

After it finishes, run the following command to start parallel build jobs, significantly speeding up the build process:

```
$ cmake --build build -j $(nproc)
```

Although not necessary, it is possible to install SUMO binaries into your system and then update the `SUMO_HOME` variable accordingly:

```
$ sudo cmake --install build
$ export SUMO_HOME=/usr/local/share/sumo
```

A.3 Artery

Once OMNeT++ and SUMO are installed, we proceed with building Artery. While it is possible to automatically build a virtual machine with Artery through Vagrant, this solution proved to be quite unstable in our tests. Thus, it is preferable to proceed with the manual build process, at least for the moment.

A.3.1 Requirements

Firstly, it's essential to verify if the following requirements are met. Otherwise, utilize the suggested terminal commands to acquire them.

- C++ compiler with C++11 support (GNU GCC 4.9 or later or Clang) If these are not present or your machine lacks them, use the following commands:

  ```
  $ sudo apt -y install clang
  $ clang --version
  ```

  or

  ```
  $ sudo apt -y install build-essential
  $ gcc --version
  ```

- Boost (1.65.1 or later)

  If this is not present or unavailable on your machine, use the following command:

  ```
  $ sudo apt-get -y install libboost-all-dev
  ```

- CMake (3.1 or later)

  If this is not present or unavailable on your machine, use the following commands:

  ```
  $ sudo apt-get -y install cmake
  $ which cmake
  $ cmake --version
  ```

- Python3

  ```
  $ sudo apt -y update
  $ sudo apt -y install python3
  ```

Vanetza, as mentioned in section 5.3.1, is an integral part of Artery, providing the ITS-G5 network stack. Apart from the dependencies mentioned above, Vanetza requires the following:

- GeographicLib (1.37 or later)

  ```
  $ sudo apt-get -y update
  $ sudo apt-get -y install libgeographic-dev
  ```

- Crypto++ (5.6 or later)

  ```
  $ sudo apt-get -y update
  $ sudo apt-get -y install libcrypto++-dev
  ```

Cloning the Artery Repository

The Artery git repository includes compatible versions of Vanetza, Veins, INET, SimuLTE, and some other third-party dependencies as git submodules. These submodules are located in the ***extern*** subdirectory.

When cloning the Artery repository, it's essential to fetch these submodules. The command below clones Artery's ***master*** branch along with matching versions of its submodules onto your machine's file system. Your local clone of the repository will be in the *artery* directory at the location where you invoke the clone command.

```
$ git clone --recurse-submodule https://github.com/riebl/artery.git
```

***Note**: Use the attached folder instead of fetching the resources by cloning the Artery repository to work with the Artery code updated by this thesis development.*

A.3.2 Building Artery from Sources

Bundled third-party dependencies in *extern* subdirectories are built alongside Artery. Artery's build process integrates those dependencies seamlessly without manual intervention. CMake handles the entire build process independently.

From `$ARTERY_PATH` (i.e., the location of your local clone), create a build directory for Artery, configure the build directory with `CMake`, and finally build Artery there.

```
$ mkdir build
$ cd build
$ cmake ..
$ cmake --build .
```

Appendix B

User Manual

Once all the necessary libraries and software listed in subsection A.3.1 have been installed, we can proceed to delve deeper into reproducing the attacks designed and developed in this thesis work.

B.1 Executing the Sybil Attack Scenario

As detailed in chapter 6, executing the Sybil Attack involves using three instances of Sumo Traffic Simulator linked to the Artery Framework. The first represents the real world (Real Scenario), the second reflects the Ego Vehicle's perception (Ego Perception), and the third simulates the fake traffic track used by attackers to generate malicious ITS messages. By default, Artery's execution automatically runs the main SUMO instance (Real Scenario), requiring parallel commands to run SUMO EgoPerception and SUMO FakeTracks.

To initiate the Sybil Attack scenario, use the following commands from Artery's root directory:

```
$ sumo-gui -c ./scenarios/sybil-attack/egoPerception.sumocfg \
--remote-port 9921 --collision.action "warn" & \
sumo-gui -c ./scenarios/sybil-attack/fakeTracks.sumocfg \
--remote-port 9922 & \
cmake --build build --target run_sybil_attack
```

Once Artery runs, it establishes a TCP socket with each SUMO instance. When every SUMO instance detects the established connection with Artery, the simulation can proceed. The simulated attack execution will demonstrate what is described in section 7.1.1.

B.2 Executing the Replay Attack Scenario

Similar to the Sybil Attack, executing the Replay Attack involves using three instances of Sumo Traffic Simulator linked to the Artery Framework. The first represents the real world (Real Scenario), the second reflects the Ego Vehicle's perception (Ego Perception), and the third simulates the fake traffic track used by attackers to generate malicious ITS messages. Use parallel commands to run SUMO EgoPerception and SUMO FakeTracks alongside Artery.

To initiate the Replay Attack scenario, use the following commands from Artery's root directory:

```
$ sumo-gui -c ./scenarios/replay-attack/egoPerception.sumocfg \
--remote-port 9921 --collision.action "warn" & \
cmake --build build --target run_replay_attack
```

The simulated attack execution will demonstrate what is described in section 7.1.1.

B.3 Executing the Message Modification Attack Scenario

Unlike the Sybil and Replay Attacks, for the Message Modification and Black Hole attacks, it is unnecessary to visualize two independent executions in parallel. Therefore, launching two additional SUMO instances to the main one launched by Artery is no longer required. Since SUMO is deterministic unless otherwise requested, relaunching the same simulation with the attack deactivated allows visualizing an ideal scenario where the attacker vehicle does not initiate the attack, potentially emphasizing the consequences of the attacks. To initiate the Message Modification Attack scenario, use the following commands from Artery's root directory:

```
$ cmake --build build --target run_message_modification_attack
```

To evaluate the original scenario where the attack is disabeled, modify the value of `isAttackOn` parameter in the `omnetpp.ini` file. The `omnetpp.ini` file is located in `scenarios/message-modification-attack`.

```
**.isAttackOn = true
```

The simulated attack execution will demonstrate what is described in section 7.1.1.

B.4 Executing the BlackHole Attack Scenario

Similar to the Message Modification attack, the BlackHole Attack leverages only one instance of SUMO. However, evaluating two sequential executions, with and without the attack, allows for a more comprehensive understanding of the attack's impact.

To initiate the Black Hole Attack scenario, use the following commands from Artery's root directory:

```
$ cmake --build build --target run_blackhole_attack
```

To evaluate the scenario where the attack is not enabled, change the value of a specific parameter in the `omnetpp.ini` file located in `scenarios/blackhole-attack`.

```
**.isAttackOn = true
```

The simulated attack execution will demonstrate what is described in section 7.1.1.

***Note**: For each developed attack, it is possible to run the secured scenario by uncommenting a specific line in the `omnetpp.ini` file.*

In the `scenarios/<SPECIFIC_ATTACK>-attack` directory, comment or uncomment the line below to respectively disable or enable secure communication:

```
*.node[*].vanetza[*].security.typename = "SecurityEntity"
```

B.5 Executing Attacks Against a Real Device (Emulation Mode)

To execute attacks presented in this thesis targeting a real-world device (e.g., Cohda Wireless MKx), the following steps are required.

Initially, it's imperative to equip the virtual Ego Vehicle with services designed for external device interaction. While typically inactive in simulation mode, these services play a pivotal role when integrating V2X execution into a Hardware-in-the-Loop (HiL) Platform, as discussed in subsection 8.2.1. Specifically, the EgoGNSSService and CaExportService, detailed in section 6.2.1 and subsection 8.1.1, respectively facilitate the transmission of the virtual Ego Vehicle's GNSS information and the transit packets within the simulated network to the real device.

Enabling these services necessitates equipping the virtual Ego Vehicle instance alone. Within the `services.xml` file in `scenarios/sybil-attack`, insert the following lines:

```
<service type="artery.application.EgoNMEAService">
    <listener port="2003" />
    <filters>
        <name pattern="car0" />
    </filters>
</service>

<service type="artery.application.CaExportService">
    <listener port="2003" />
    <filters>
        <name pattern="car0" />
    </filters>
</service>
```

Moreover, for effective interaction with a real-world device, the adoption of a real-time scheduler is crucial. This synchronization ensures the simulation's execution aligns with real (wall clock) time.

To enable this, include the following in the `scenarios/sybil-attack/omnetpp.ini` file:

```
scheduler-class = "omnetpp::cRealTimeScheduler"
```

Bibliography

[1] F. Li and Y. Wang, "Routing in vehicular ad hoc networks: A survey", IEEE Vehicular Technology Magazine, vol. 2, June 2007, pp. 12–22, DOI 10.1109/MVT.2007.912927

[2] S. Sharma and Nidhi, "Vehicular Ad-Hoc Network: An Overview", International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), Greater Noida (India), February 18-21, 2019, pp. 131–134, DOI 10.1109/ICCCIS48478.2019.8974524

[3] F. Perry, K. Raboy, E. Leslie, Z. Huang, and D. Van Duren, "Dedicated short-range communications roadside unit specifications.", https://rosap.ntl.bts.gov/view/dot/3600, April 2017

[4] 5G Automotive Association (5GAA), "5GAA V2X Terms and Definitions", https://5gaa.org/content/uploads/2017/08/5GAA-V2X-Terms-and-Definitions110917.pdf

[5] P. Jing, W. Huang, and L. Chen, "Car-to-Pedestrian Communication Safety System Based on the Vehicular Ad-Hoc Network Environment: A Systematic Review", Information, vol. 8, October 2017, p. 127, DOI 10.3390/info8040127

[6] K. Abboud, H. A. Omar, and W. Zhuang, "Interworking of DSRC and Cellular Network Technologies for V2X Communications: A Survey", IEEE Transactions on Vehicular Technology, vol. 65, December 2016, pp. 9457–9470, DOI 10.1109/TVT.2016.2591558

[7] D.-J. Deng, S.-Y. Lien, J. Lee, and K.-C. Chen, "On Quality-of-Service Provisioning in IEEE 802.11ax WLANs", IEEE Access, vol. 4, August 2016, pp. 6086–6104, DOI 10.1109/ACCESS.2016.2602281

[8] S. Zeadally, M. A. Javed, and E. B. Hamida, "Vehicular Communications for ITS: Standardization and Challenges", IEEE Communications Standards Magazine, vol. 4, March 2020, pp. 11–17, DOI 10.1109/MCOMSTD.001.1900044

[9] T. Yeferny and S. Hamad, "Vehicular Ad-hoc Networks: Architecture, Applications and Challenges", International Journal of Computer Science and Network Security, vol. 20, 02 2020, DOI 10.48550/arXiv.2101.04539

[10] National Highway Traffic Safety Administration, "Analyses of Rear-End Crashes and Near-Crashes in the 100-Car Naturalistic Driving Study to Support Rear-Signaling Countermeasure Development", https://api.semanticscholar.org/CorpusID:108242920, 2007

[11] U.S. Department of Transportation, "Safety - Do Not Pass Warning", https://www.its.dot.gov/infographs/DoNotPass.htm

[12] ETSI TR 102 638 V1.1.1, "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions", https://www.etsi.org/deliver/etsi_tr/102600_102699/102638/01.01.01_60/tr_102638v010101p.pdf, June 2009

[13] 3GPP TS 22.186 15.4.0, "Service Requirements for Enhanced V2X Scenarios",

https://www.3gpp.org/specifications-technologies/releases/release-16, September 2018

[14] W. Zeng, M. A. S. Khalid, and S. Chowdhury, "In-Vehicle Networks Outlook: Achievements and Challenges", IEEE Communications Surveys & Tutorials, vol. 18, July-September 2016, pp. 1552–1571, DOI 10.1109/COMST.2016.2521642

[15] T. Huang, J. Zhou, Y. Wang, and A. Cheng, "On the Security of In-Vehicle Hybrid Network: Status and Challenges", Information Security Practice and Experience, Melbourne (Australia), December 13-15, 2017, pp. 621–637, DOI 10.1007/978-3-319-72359-4_38

[16] E. Kadiri, E. Fitterer, and B. Manson, "EE Cars Architecture & Linked ECU: Constraints and New Needs", International Symposium on Microelectronics, vol. 2010, 01 2010, pp. 52–58, DOI 10.4071/isom-2010-TA2-Paper4

[17] J. Huang, M. Zhao, Y. Zhou, and C.-C. Xing, "In-Vehicle Networking: Protocols, Challenges, and Solutions", IEEE Network, vol. 33, January-February 2019, pp. 92–98, DOI 10.1109/MNET.2018.1700448

[18] The OPEN Alliance (One-Pair Ether-Net) Special Interest Group (SIG), https://opensig.org/

[19] K. P. Rao and B. Muthukrishnan, "Integrated Automotive Gateway Can Enable Connected Cars", SAE News, November 2014, https://www.sae.org/news/2014/11/integrated-automotive-gateway-can-enable-connected-cars

[20] NXP Semiconductor - Technical Information Center, "Automotive Gateway: A Key Component to Securing the Connected Car", https://www.nxp.com/docs/en/white-paper/AUTOGWDEVWPUS.pdf, February 2018

[21] SAE, "Vehicle E E System Diagnostic Standards Committee", SAE International, July 2016, DOI 10.4271/J1962_201607

[22] ISO 15765-4:2021, "Diagnostic communication over Controller Area Network (DoCAN)", https://www.iso.org/standard/78384.html, July 2021

[23] Bluetooth SIG - Bluetooth Technology Overview, https://www.bluetooth.com/learn-about-bluetooth/tech-overview/

[24] OnStar, LLC, https://www.onstar.com

[25] N. INSTRUMENTS, "Introduction to 802.11ax High-Efficiency Wireless", https://www.ni.com/en/solutions/semiconductor/wireless-connectivity-test/introduction-to-802-11ax-high-efficiency-wireless.html, April 2023

[26] A. Akbilek, F. Pfeiffer, and M. Fuenfer, "Analysis of IEEE 802.11ax High Efficiency WLANs for in-Vehicle Use", 15th Wireless Congress: Systems & Applications, Munich (Germany), November 14-15 2018. https://api.semanticscholar.org/CorpusID:150381295

[27] H. Dakroub and R. Cadena, "Analysis of Software Update in Connected Vehicles", SAE International Journal of Passenger Cars, vol. 7, January 2014, DOI 10.4271/2014-01-0256

[28] J. Wang, Y. Shao, Y. Ge, and R. Yu, "A Survey of Vehicle to Everything (V2X) Testing", Sensors, vol. 19, January 2019, DOI 10.3390/s19020334

[29] K. Yacine, M. Tsukada, J. Santa, C. JinHyeock, and T. Ernst, "A Usage Oriented Analysis of Vehicular Networks: from Technologies to Applications", Journal of Communications, vol. 4, 06 2009, DOI 10.4304/jcm.4.5.357-368

[30] IEEE, "Wireless Access in Vehicular Environments", IEEE Std 802.11p-2010, July 2010, pp. 1–51, DOI 10.1109/IEEESTD.2010.5514475

[31] 5G Automotive Association (5GAA), "C-V2X explained", https://5gaa.org/c-v2x-explained

[32] 3GPP TR 21.914, "Specifications & Technologies - Release 14", https://www.3gpp.org/specifications-technologies/releases/release-14, March 2017

[33] 3GPP TR 21.916, "Specifications & Technologies - Release 16", https://www.3gpp.org/specifications-technologies/releases/release-16, July 2020

[34] P. Roux and V. Mannoni, "Performance evaluation for co-channel coexistence between ITS-G5 and LTE-V2X", IEEE 92nd Vehicular Technology Conference (VTC2020-Fall), Victoria (Canada), 2020, pp. 1–5, DOI 10.1109/VTC2020-Fall49728.2020.9348517

[35] K. Ansari, "Joint use of DSRC and C-V2X for V2X communications in the 5.9 GHz ITS band", IET Intelligent Transport Systems, vol. 15, December 2021, pp. 213–224, DOI 10.1049/itr2.12015

[36] ETSI TR 103-766, "Pre-standardization study on co-channel co-existence between IEEE- and 3GPP- based ITS technologies", https://www.etsi.org/deliver/etsi_tr/103700_103799/103766/01.01.01_60/tr_103766v010101p.pdf, September 2021

[37] ISO 11898-1:2015, "Controller area network (CAN)", https://www.iso.org/standard/63648.html, December 2015

[38] B. Leigh and R. Duwe, "Designing Autonomous Vehicles for a Future of Unknowns", ATZ Electron Worldw 16, vol. 16, March 2021, pp. 44–47, DOI 10.1007/s38314-020-0578-3

[39] D. Burton, A. Delaney, S. Newstead, D. Logan, and B. Fildes, "Effectiveness of ABS and Vehicle Stability Control Systems", Royal Automobile Club of Victoria Ltd, January 2004, ISBN: 1-875963-39-1

[40] Q. Alrousan, S. Matta, and T. Tasky, "Multi-Sensor Fusion in Slow Lanes for Lane Keep Assist System", SAE WCX Digital Summit, Detroit (MI, USA), January, 2021, DOI 10.4271/2021-01-0084

[41] X. Sun, F. R. Yu, and P. Zhang, "A Survey on Cyber-Security of Connected and Autonomous Vehicles (CAVs)", IEEE Transactions on Intelligent Transportation Systems, vol. 23, July 2022, pp. 6240–6259, DOI 10.1109/TITS.2021.3085297

[42] C. Valasek and C. Miller, "A Survey of Remote Automotive Attack Surfaces", IOActive, September 2014, https://ioactive.com/pdfs/IOActive_Remote_Attack_Surfaces.pdf

[43] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive Experimental Analyses of Automotive Attack Surfaces", 20th USENIX Security Symposium (USENIX Security 11), San Francisco (CA, USA), Aug 8-12, 2011, DOI 10.5555/2028067.2028073

[44] W. Yan, "A two-year survey on security challenges in automotive threat landscape", International Conference on Connected Vehicles and Expo (ICCVE), Shenzhen (China), October 19-23, 2015, pp. 185–189, DOI 10.1109/ICCVE.2015.1

[45] I. Rouf, R. Miller, H. Mustafa, T. Taylor, S. Oh, W. Xu, M. Gruteser, W. Trappe, and I. Seskar, "Security and Privacy Vulnerabilities of In-Car Wireless Networks: A Tire Pressure Monitoring System Case Study", 19th USENIX Conference on Security, Washington (DC, USA), August 11-13, 2010, p. 21, DOI 10.5555/1929820.1929848

[46] F. D. Garcia, D. Oswald, T. Kasper, and P. Pavlidès, "Lock It and Still Lose It - on the (in)Security of Automotive Remote Keyless Entry Systems", 25th USENIX Conference on Security Symposium, Austin (TX, USA), August 10-12, 2016, pp. 929–944, DOI 10.5555/3241094.3241166

[47] S. Nie, L. Liu, Y. Du, and W. Zhang, "Experimental Security Assessment on Lexus Cars", Keen Blog, March 2020, https://keenlab.tencent.com/en/2020/03/30/Tencent-Keen-Security-Lab-Experimental-Security-Assessment-on-Lexus-Cars/

[48] I. LABS, "From 0-day to exploit - Buffer overflow in Belkin N750 (CVE-2014-1635)", Devoteam Cyber Trust, November 2014, https://labs.integrity.pt/articles/from-0-day-to-exploit-buffer-overflow-in-belkin-n750-cve-2014-1635/

[49] S. Nie, L. Liu, and Y. Du, "Free-fall: Hacking tesla from wireless to CAN bus", Black Hat USA 2017, Las Vegas (NV, USA), July 26-27, 2017. https://www.blackhat.com/docs/us-17/thursday/us-17-Nie-Free-Fall-Hacking-Tesla-From-Wireless-To-CAN-Bus-wp.pdf

[50] C. Miller and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle", Black Hat USA, August 2015, https://illmatics.com/Remote%20Car%20Hacking.pdf

[51] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental Security Analysis of a Modern Automobile", IEEE Symposium on Security and Privacy, Oakland (CA, USA), May 16-19, 2010, pp. 447–462, DOI 10.1109/SP.2010.34

[52] O. Burkacky, J. Deichmann, B. Klein, K. Pototzky, and G. Scherf, "Cybersecurity in automotive", McKinsey & Company, March 2020. https://www.mckinsey.com/~/media/mckinsey/industries/automotive%20and%20assembly/our%20insights/cybersecurity%20in%20automotive%20mastering%20the%20challenge/cybersecurity-in-automotive-mastering-the-challenge.pdf

[53] S. Nie, L. Liu, Y. Du, and W. Zhang, "Over-The-Air: How We Remotely Compromised the Gateway, BCM, and Autopilot ECUs Of Tesla Cars", Black Hat USA 2018, Las Vegas (NV, USA), August 4-7, 2018. https://i.blackhat.com/us-18/Thu-August-9/us-18-Liu-Over-The-Air-How-We-Remotely-Compromised-The-Gateway-Bcm-And-Autopilot-Ecus-Of-Tesla-Cars-wp.pdf

[54] ETSI TS 102 940 V2.1.1, "Intelligent Transport Systems (ITS); Security; ITS communications security architecture and security management;", https://www.etsi.org/deliver/etsi_ts/102900_102999/102940/02.01.01_60/ts_102940v020101p.pdf, July 2021

[55] F. Sakiz and S. Sen, "A survey of attacks and detection mechanisms on intelligent transportation systems: VANETs and IoV", Ad Hoc Networks, vol. 61, March 2017, pp. 33–50, DOI 10.1016/j.adhoc.2017.03.006

[56] J. Kamel, F. Haidar, I. B. Jemaa, A. Kaiser, B. Lonc, and P. Urien, "A Misbehavior Authority System for Sybil Attack Detection in C-ITS", IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York (NY, USA), October 10-12, 2019, pp. 1117–1123, DOI 10.1109/UEMCON47517.2019.8993045

[57] J. Cassou-Mounat, H. Labiod, and R. Khatoun, "Simulation of Cyberattacks in ITS-G5 Systems", Communication Technologies for Vehicles, Bordeaux (France), November 16-17, 2020, pp. 3–14, DOI 10.1007/978-3-030-66030-7_1

[58] F. Ahmad, A. Adnane, V. N. L. Franqueira, F. Kurugollu, and L. Liu, "Man-In-The-Middle Attacks in Vehicular Ad-Hoc Networks: Evaluating the Impact of Attackers' Strategies", Sensors, vol. 18, November 2018, p. 4040, DOI 10.3390/s18114040

[59] M. Hasan, S. Mohan, T. Shimizu, and H. Lu, "Securing Vehicle-to-Everything (V2X) Communication Platforms", IEEE Transactions on Intelligent Vehicles, vol. 5, December 2020, pp. 693–713, DOI 10.1109/TIV.2020.2987430

[60] G. Costantino, A. La Marra, F. Martinelli, and I. Matteucci, "CANDY: A Social Engineering Attack to Leak Information from Infotainment System", IEEE 87th Vehicular Technology Conference (VTC Spring), Porto (Portugal), June 3-6, 2018, pp. 1–5, DOI 10.1109/VTCSpring.2018.8417879

[61] ETSI EN 302 665 V1.1.1, "Intelligent Transport Systems (ITS); Communications Architecture", https://www.etsi.org/deliver/etsi_en/302600_302699/

302665/01.01.01_60/en_302665v010101p.pdf, September 2010

[62] ETSI EN 302 636-4-1 V1.4.1, "Intelligent Transport Systems (ITS); GeoNetworking; Geographical addressing and forwarding for point-to-point and point-to-multipoint communications", https://www.etsi.org/deliver/etsi_en/302600_302699/3026360401/01.04.01_30/en_3026360401v010401v.pdf, January 2020

[63] ETSI EN 302 636-5-1 V2.2.1, "Intelligent Transport Systems (ITS); GeoNetworking; Transport Protocols", https://www.etsi.org/deliver/etsi_en/302600_302699/3026360501/02.02.01_60/en_3026360501v020201p.pdf, May 2019

[64] ETSI EN 302 636-3 V1.1.2, "Intelligent Transport Systems (ITS); GeoNetworking; Network Architecture", https://www.etsi.org/deliver/etsi_en/302600_302699/30263603/01.01.02_20/en_30263603v010102a.pdf, March 2014

[65] European Commission, Directorate-General for Mobility and Transport, "COMMISSION DELEGATED REGULATION (EU) supplementing Directive 2010/40/EU of the European Parliament and of the Council with regard to the deployment and operational use of cooperative intelligent transport systems", https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=PI_COM%3AC%282019%291789, March 2019

[66] ETSI TS 102 637-2 V1.2.1, "Intelligent Transport Systems (ITS); Basic Set of Applications; Specification of Cooperative Awareness Basic Service", https://www.etsi.org/deliver/etsi_ts/102600_102699/10263702/01.02.01_60/ts_10263702v010201p.pdf, March 2011

[67] ETSI EN 302 637-2 V1.3.1, "Intelligent Transport Systems (ITS); Basic Set of Applications; Specification of Cooperative Awareness Basic Service", https://www.etsi.org/deliver/etsi_en/302600_302699/30263702/01.03.01_30/en_30263702v010301v.pdf, September 2014

[68] ETSI EN 302 637-3 V1.2.1, "Intelligent Transport Systems (ITS); Basic Set of Applications; Specifications of Decentralized Environmental Notification Basic Service", https://www.etsi.org/deliver/etsi_en/302600_302699/30263703/01.02.01_30/en_30263703v010201v.pdf, September 2014

[69] ETSI TS 101 539-1 V1.1.1, "Intelligent Transport Systems (ITS); V2X Applications; Road Hazard Signalling (RHS) application requirements specification", https://www.etsi.org/deliver/etsi_ts/101500_101599/10153901/01.01.01_60/ts_10153901v010101p.pdf, August 2013

[70] ETSI TS 101 539-2 V1.1.1, "Intelligent Transport Systems (ITS); V2X Applications; Intersection Collision Risk Warning (ICRW) application requirements specification", https://www.etsi.org/deliver/etsi_ts/101500_101599/10153902/01.01.01_60/ts_10153902v010101p.pdf, June 2018

[71] ETSI TS 101 539-3 V1.1.1, "Intelligent Transport Systems (ITS); V2X Applications; Longitudinal Collision Risk Warning (LCRW) application requirements specification", https://www.etsi.org/deliver/etsi_ts/101500_101599/10153903/01.01.01_60/ts_10153903v010101p.pdf, November 2013

[72] ETSI TS 103 097 V1.2.1, "Intelligent Transport Systems (ITS); Security; Security header and certificate formats", https://www.etsi.org/deliver/etsi_ts/103000_103099/103097/01.02.01_60/ts_103097v010201p.pdf, June 2015

[73] ETSI TR 101 607 V1.1.1, "Intelligent Transport Systems (ITS); Cooperative ITS (C-ITS); Release 1", https://www.etsi.org/deliver/etsi_tr/101600_101699/101607/01.01.01_60/tr_101607v010101p.pdf, May 2013

[74] ETSI TS 102 941 V1.1.1, "Intelligent Transport Systems (ITS); Security; Trust and Privacy Management", https://www.etsi.org/deliver/etsi_ts/102900_102999/102941/01.01.01_60/ts_102941v010101p.pdf, June 2012

[75] ETSI TR 103 415 V1.1.1, "Intelligent Transport Systems (ITS); Security; Pre-standardization study on pseudonym change management", https://www.etsi.org/deliver/etsi_tr/103400_103499/103415/01.01.01_60/tr_103415v010101p.pdf, April 2018

[76] ETSI TS 102 940 V1.3.1, "Intelligent Transport Systems (ITS); Security; ITS communications security architecture and security management", https://www.etsi.org/deliver/etsi_ts/102900_102999/102940/01.03.01_60/ts_102940v010301p.pdf, April 2018

[77] ETSI TS 103 096-2 V1.2.1, "Intelligent Transport Systems (ITS); Security; Conformance test specifications for ITS Security", https://www.etsi.org/deliver/etsi_ts/103000_103099/10309602/01.02.01_60/ts_10309602v010201p.pdf, September 2015

[78] OMNeT++, "What is OMNeT++?", https://omnetpp.org/intro/

[79] A. ur Rehman Khan, S. M. Bilal, and M. Othman, "A Performance Comparison of Network Simulators for Wireless Networks", CoRR, vol. abs/1307.4129, August 2018, DOI 10.48550/arXiv.1307.4129

[80] E. Weingartner, H. vom Lehn, and K. Wehrle, "A Performance Comparison of Recent Network Simulators", IEEE International Conference on Communications, Dresden (Germany), June 14-18, 2009, pp. 1–5, DOI 10.1109/ICC.2009.5198657

[81] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flotterod, R. Hilbrich, L. Lucken, J. Rummel, P. Wagner, and E. Wiessner, "Microscopic Traffic Simulation using SUMO", 21st International Conference on Intelligent Transportation Systems (ITSC), Maui (HI, USA), November 4-7, 2018, pp. 2575–2582, DOI 10.1109/ITSC.2018.8569938

[82] N. Weber, D. Frerichs, and U. Eberle, "A simulation-based, statistical approach for the derivation of concrete scenarios for the release of highly automated driving functions", Automotive meets Electronics; 11th GMM-Symposium, Dortmund (Germany), March 10-11, 2020, pp. 1–6, DOI 10.13140/RG.2.2.15306.31683/1

[83] Traffic Control Interface - TraCI, https://sumo.dlr.de/docs/TraCI.html

[84] R. Riebl, C. Obermaier, and H.-J. Gunther, "Artery: Large Scale Simulation Environment for ITS Applications", Recent Advances in Network Simulation: The OMNeT++ Environment and ITS Ecosystem (M. K. Antonio Virdis, ed.), pp. 365–406, Springer Cham, 05 2019, DOI 10.1007/978-3-030-12842-5_12

[85] M. Raya and J.-P. Hubaux, "Securing Vehicular Ad Hoc Networks", J. Comput. Secur., vol. 15, January 2007, pp. 39–68, DOI 10.5555/1370616.1370618

[86] ETSI TR 103 415 V1.1.1, "Intelligent Transport Systems (ITS); Security; Pre-standardization study on pseudonym change management", https://www.etsi.org/deliver/etsi_tr/103400_103499/103415/01.01.01_60/tr_103415v010101p.pdf, April 2018

[87] CAR 2 CAR Communication Consortium, "C2C-CC Basic System Standards Profile", https://www.car-2-car.org/fileadmin/documents/Basic_System_Profile/Release_1.1.0/C2CCC_RS_2001_BasicSystemProfile_R110.pdf, December 2015

[88] ETSI TR 102 893 V1.2.1, "Intelligent Transport Systems (ITS); Security; Threat, Vulnerability and Risk Analysis (TVRA)", https://www.etsi.org/deliver/etsi_tr/102800_102899/102893/01.02.01_60/tr_102893v010201p.pdf, March 2017

[89] M. Rash, "afl-cve: Fuzzer for Common Vulnerabilities and Exposures (CVE)", https://github.com/mrash/afl-cve, August 2017

[90] D. Birdwell, "afl: American Fuzzy Lop", https://github.com/jdbirdwell/afl, November 2015

[91] A. Helin, "Radamsa: A Test Case Generator for Fuzz Testing", https://gitlab.com/akihe/radamsa, July 2016

[92] ps1337, "pcap-mitm-fuzz0r", https://github.com/ps1337/pcap-mitm-fuzz0r, February 2020

[93] N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Krügel, and G. Vigna, "Driller: Augmenting Fuzzing Through Selective Symbolic Execution", Network and Distributed System Security Symposium, San Diego (CA, USA), February 21-24, 2016, DOI 10.14722/NDSS.2016.23368

[94] OMNeT++, "Installation Guide - Version 6.0", https://doc.omnetpp.org/omnetpp/InstallGuide.pdf

List of Figures