

A Hardware-based Secure Communication Module to Protect Internet Connected Vehicles

Andreas Schmoll

Information Security, master's level (120 credits)
2018

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

Abstract

Information technology has not only been a driving force of the car industries within the last couple of years but it even seems to be of growing importance for the unforeseeable future. The inclusion of information technology is accompanied with the promise of increased comfort and the prospect that future autonomous and connected cars will become a place to spend our time while being on the road. Thus, car manufacturers strive to equip their next-generation cars with wireless network interfaces (e.g. WiFi, Bluetooth and 3G/4G) and to provide various services based on that. The availability of a wireless interface enables, on the one hand remote maintenance services (such as Over-the-Air (OTA) software updates and OTA calibration) at the passengers convenience and a whole eco-system of smart services based on access to the Internet and car-to-car/infrastructure/road/smartphone communication. On the other hand, the availability of telemetry data is a big chance for the manufacturers to obtain data on the performance of their fleet under real-life conditions.

However, at the same time, a bi-directional interface that is always connected to the Internet opens up the threat of adversarial intrusion and hacked vehicles, which are, in the worst case, remote-controlled by hackers, or even malware - transforming hacked cars into driving botnets. Therefore, the need arises to implement security features to guarantee the passengers safety while maintaining functionality and comfort.

In 2015 two security researchers demonstrated their attack on an unmodified Jeep Cherokee, allowing them to remotely control critical components like the steering or even the breaks of the vehicle. Also other research groups have implemented remote attacks showing the weaknesses of today's internet connected cars. To increase security of the communication link of such vehicles, this thesis focuses on the development of a hardware-based Secure Communication Module (SCM). Such a module should provide a secure way for communication over the Internet. As the vehicle's first layer of defense it should work as a firewall as well as a gateway to the inner-vehicle network. Being exposed to the Internet, the operation system of the SCM also needs to be hardened. Additionally, the SCM should isolate the internal car network, preventing malicious control of in-vehicle components.

After designing a security concept based on well-known security techniques an ARM Cortex-A9 board

running an adopted Linux was used as prototype of an vehicular SCM. A penetration test was performed by an external company, specialized in security audits. The prototype was rated to provide adequate security for connected cars against external attacks. Beside the security, also the functionality was evaluated. Therefore the newly developed SCM was integrated into a car and an OTA firmware update of an internal car component was successfully performed. In contrast to an equivalent update without the security module, the main difference was an increased latency because of the additional device. This cost of a bit slower connection for increased security was accepted for an prototype. Because security is a rather new topic in the automotive industry the main goal of this thesis is to show one approach how a secure communication link can be implemented for connected cars. This can be used as basis for further research and my contribute to more security in a highly connected world. Even though this work is written with the aspect of automotive security, many concepts can be also used in the rapidly growing Internet of Things (IoT) field.

Acknowledgments

First and foremost, I would like to thank my LTU supervisor Ali Ismail Awad for his great support throughout the writing process of this master thesis with helpful comments and feedback. He encouraged me to achieve this final state of my thesis and also assisted with organizational matters.

I am very grateful to my supervisor from AVL, Christian Hanser, although he left the company shortly after I started. Christian provided me with very inspiring ideas how to realize the project. Also many thanks to Wolfgang and Patrick for the many hours of constructive discussions we had together.

During the whole period of the thesis, but mainly in the end, my friends helped me to stay motivated and recharge my batteries with social activities. Here I have to especially thank Ina.

Meinen Eltern einen ganz besonderen Dank für ihre Geduld und Unterstützung die sie mir während meiner gesamten Studienzeit entgegen brachten.

Contents

List of Figures	vii
List of Tables	viii
1 Introduction	1
1.1 Problem definition	4
1.2 Research Questions	4
1.3 Proposed solution	5
1.4 Expected Contribution	5
1.5 Research Delimitation	6
1.6 Thesis Outline	7
2 Background	8
2.1 Required protocols	8
2.1.1 MQTT protocol	8
2.1.2 Transport Layer Security (TLS) protocol	9
2.1.3 Cryptographic Message Syntax (CMS)	11
2.2 Required Linux components	11
2.2.1 Netfilter/iptables	11
2.2.2 Linux Security Modules (LSM)	15
2.2.3 Communication sockets	17
2.2.4 Virtualization technology	17
2.3 Additional security concepts	20
2.3.1 (Distributed) Denial-of-Service attacks	21
2.3.2 Intrusion Detection and Prevention Systems	21

3	Related Work	22
3.1	Summary	23
3.1.1	Research gap	23
4	Research Method	24
4.1	Design Science Guidelines	24
4.1.1	Design as an Artifact	24
4.1.2	Problem Relevance	25
4.1.3	Design Evaluation	25
4.1.4	Research Contributions	25
4.1.5	Research Rigor	25
4.1.6	Design as a Search Process	25
4.1.7	Communication of Research	25
4.2	Design Science Implementation	25
4.2.1	Problem identification and motivation	26
4.2.2	Definition of the objectives for a solution	26
4.2.3	Design and development	27
4.2.4	Demonstration	27
4.2.5	Evaluation	28
4.2.6	Communication	30
5	Security Concept	31
5.1	Defense Mechanisms	31
5.1.1	(Distributed) Denial-of-Service protection	32
5.1.2	Intrusion Detection and Prevention Systems	32
5.2	System Hardening	33
5.2.1	Kernel Hardening: SELinux	33
5.2.2	Isolation: Linux Containers	33
5.2.3	Additional Hardening Measures	34
5.3	Communication via the SCM	34
5.3.1	Isolation of the car's internal control network	35
5.3.2	Message Queue Telemetry Transport protocol	36
5.3.3	Remote-Call-over-Serial protocol	36
5.3.4	Encryption and authentication	42

6	Implementation and Evaluation	45
6.1	Implementation of Security Concept	45
6.2	Operational Evaluation	45
6.2.1	Over-the-Air update test	46
6.2.2	Telemetry data collection test	46
6.3	Performance evaluation	46
6.4	Security Performance	47
7	Discussion	49
7.1	Usability and Practicability	49
7.2	Performance	49
7.3	Summary	50
8	Conclusion and Future Work	51
8.1	Future work	52
	Acronyms	53
	References	61

List of Figures

1.1	Interfaces of connected cars [3]	2
1.2	Secure Communication Module in the overall security architecture	6
2.1	MQTT publish/subscribe communication	9
2.2	Transport Layer Security (TLS) handshake protocol using client authentication	10
2.3	Traversing path of packets through the build-in chains in iptables [31].	14
2.4	LSM architecture (adopted from [35])	16
4.1	Steps of Design Science Research (DSR) methodology	26
5.1	Firewall should only allow communications initiated by the Secure Communication Module (SCM)	32
5.2	Dataflow of sending telemetry data	35
5.3	Example dataflow of an Electronic Control Unit (ECU) firmware update	36

List of Tables

2.1	Correlation of build-in tables and chains [29]	13
4.1	List of evaluation criteria	29
5.1	Required number of bytes for given values	38
5.2	Fields of Remote-Call-over-Serial (RCoS) message	39
5.3	List of currently implemented RCoS commands	40
5.4	RCoS request sent over IPC socket	41
5.5	RCoS request sent over RS-232 connection	42
5.6	RCoS response sent over RS-232 connection	42
5.7	RCoS response sent over IPC socket	42
6.1	Overall time of update process	47
6.2	Activities of the Penetration Test	48

Chapter 1

Introduction

Nowadays, modern cars are much more than mechanical machines. They also contain a complex network of computer systems. Software used in cars runs on so-called ECUs. About 40 years ago, the first small bits of software were used in cars to control the ignition of the engine. At this time ECUs were single, isolated devices working independently. When the car industry added more and more software functionality in cars, they also started to connect the ECUs and let them inter-operate with each other. In today's cars almost everything can be controlled via these digital components. Besides control over entertainment systems or the lighting, also critical systems like drive-train or brakes are nowadays fully controlled by ECUs [1][2][3].

Other than major improvements in safety, usability and comfort, the transition to highly sophisticated communication architectures inside cars also brings new potential threats. Koscher K., Czeskis A., et al. [2] analyzed the practical security of modern cars by performing active experiments against them. They were able to maliciously control highly critical elements within the whole car, even including control of individual brakes independent from the drivers input.

In addition to connecting all the internal components of a vehicle, cars are also getting more and more connected to their environment via wireless communication channels. These wireless channels include Bluetooth, key-less entry systems or WiFi. Furthermore, some car companies introduced remote telematics systems providing connection to cellular voice and data networks. Their wide functionality spectrum ranges from data access for weather or traffic information, over anti-theft remote tracking, to crash reporting in case of an accident. Some manufacturers also use such systems for diagnostic purposes—for example by sending early alerts about mechanical issues [4][2][3].

Checkoway S., McCoy D. et al. [3] draw an overview picture of the communication interfaces of a modern car (see Figure 1.1). This figure illustrates the high number and diversity of different communication points a modern car uses to be connected with its surrounding.

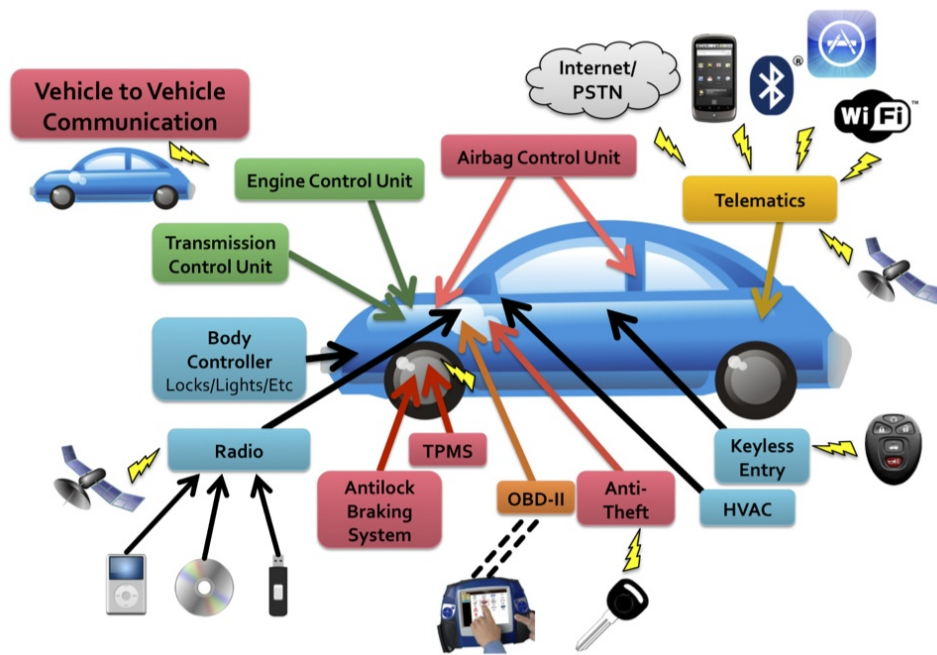


Figure 1.1: Interfaces of connected cars [3]

It is foreseeable that the trend toward connected cars will be further sped up by an initiative of the European Union (EU), namely eCall, which is an in-vehicle emergency call service, aiming at reducing emergency service response time. This system can detect an accident and automatically initiate an emergency call, which means a cellular network connection is required for all eCall equipped cars [5][6]. Based on an EU regulation from April 2018, all new cars in Europe have to be equipped with eCall technology [7].

On the negative side, additional communication interfaces also increase the attack surface. Thus, potential attackers may be able to attack a car without even needing physical access to it. Checkoway S., McCoy D., et al. [3] analyzed the external attack surface of modern automobiles. They were able to access the internal car network via different remote access points and, therefore, having the chance to perform the same kind of attacks than having physical access to the cars network. In 2015, Miller and Valasek [8] managed to remotely control an unmodified Jeep Cherokee, being able to disable the car's transmission and brakes and even take over the steering wheel while the vehicle was in reverse [9]. This forced Chrysler to recall 1.4 million vehicles [10]. Such a recall is very costly and harmful for the reputation of a company. One solution preventing the necessity of such a recall, could be remote Over-the-Air (OTA) updates. Tesla was one of the first car manufacturers using the possibility of OTA updates to automatically fix software bugs in all of its vehicles without the need of visiting a dealer [11]. In 2016 Tesla used OTA update functionality to fix its cars against a vulnerability in the web browser, allowing a Chinese research-group to remote control a Tesla S via its Wifi connection [12]. Anyhow the remote

update feature can be also seen as a potential attack vector, opening a way to modify the cars' firmware without the need of physical access. Therefore OTA updates do have two sides: On the one hand they are a very efficient instrument to close security vulnerabilities in cars, while they may also introduce new attack vectors to the vehicles. Beside security-relevant measures, OTA updates can be also used to fix non-security related bugs or adding/activating new functionality.

Additionally to hacker attacks, which are carried out by humans, also the threat of automated computer attacks does exist. Zhang T., Antunes, H., et al [13] analyze the threat of malware attacks against connected vehicles. They identified 7 possible motivations of attackers to perform malware attacks [13]:

- *Fun and publicity*: Attackers may hack cars just for fun to show they are able to do it or to get publicity.
- *Breach driver privacy*: Using spyware, many private information about the driver can be collected.
- *Ransom*: Ransomware can be used to hinder the car from proper functioning until a ransom is paid.
- *Theft*: Malware can be used to deactivate some of the car's security functionality, like the alarm or the doors' locks. This can be used by thieves to steal a car.
- *Sabotage*: Using malware to disrupt drivers may lead to severe car accidents and also harm the reputation of the car-maker.
- *Harm people and properties*: Another motivation can be to create harm. For example if a malware is able to control the breaks, this can have deadly consequences.
- *Disrupt transportation*: Using malware a big amount of vehicles can be affected, which may lead to transportation chaos.

Next to cars already sold with an integrated Telematic Control Unit (TCU), there are also aftermarket TCUs available. Most of the latter are small dongles, which can be plugged into the mandatory On-board diagnostics (OBD) port (EOBD in Europe, OBD-II in the USA, JOBD in Japan) to provide remote connectivity over the cellular network and, therefore, allow transmitting data from the car to a remote system. Such equipment is used, for example, by insurance companies to provide personalized insurance products based on the driving behavior of a customer. With so-called Pay-As-You-Drive (PAYD) insurance tariffs, a TCU is installed inside the car and sends all relevant data to the insurance company [14][15]. From this data a risk for each driver is calculated and determines the premium. From a security point of view, vulnerabilities in aftermarket TCUs can become even more severe compared to integrated TCUs,

because a huge number of different car types may be affected. Moreover, cars which are not designed to be connected to external networks may be even less secure [16].

1.1 Problem definition

The examples given above, demonstrate the importance of well protected communication interfaces of connected cars. As shown in [3] and [8] providing a secure communication interface is a highly crucial requirement. In both mentioned attacks vulnerabilities in the external communication interfaces allowed direct access to the internal automotive control network. Furthermore, this attacks also emphasize the need of a more holistic approach by combining different security layers. The ultimate goal has to be the highest possible isolation of the in-vehicular control network, while still maintaining full functionality. To make connected cars more secure, even an successful attack on the communication interface shouldn't give direct access to the internal car network. Another challenging issue in automotive security is the role of normal users as potential attackers with physical access and theoretically unlimited amount of time and number of trials to bypass security mechanisms of their car [17].

Anyhow, in contrast to traditional computer systems, cybersecurity is a rather new topic within the automotive industry. According to [18], published in 2016, car manufacturers expect another one to three years, before connected cars will be implemented with full consideration of all security aspects. As shown in Chapter 3, even tough research is addressing secure communication of connected vehicles at the moment there are more theoretical ideas than practical solutions available. Especially the potential of using a hardware-based approach have not been analyzed very deeply.

1.2 Research Questions

Based on Section 1.1, the following two research questions emerge:

- Is it possible to design and implement a hardware-based SCM within an bigger overall vehicular security infrastructure?
- Which security measures can be used to harden such a SCM and how to they perform within a vehicular security infrastructure?

1.3 Proposed solution

The goal of this thesis is the design and development of a hardware-based SCM prototype with the aim to provide a secure way to let cars communicate over the Internet and the ability to persist attacks from the external network.

The decision to use an own physical device was made to provide an optimal isolation of the internal car network. In case an attacker manage to access the SCM, the internal car network is still protected. Only communication related applications are running on the SCM, while everything else is running inside the internal car network, which is only connected to the SCM using a non-routable protocol. Therefore, all data the car wants to send to the outside, has to be sent to the SCM and from there it will be forwarded to its destination. Also all the traffic coming over the car's internet connection is sent to the SCM and only forwarded to the internal network in case it is considered as being legitimate. Consequently, no direct communication between the internal car network and devices outside the car is possible. To ensure authentication of data and for preventing information disclosure, the SCM shall encrypt all data sent over the car's external communication link. Within the setting of this thesis, only outbound communication is permitted, requiring all communication to be initiated by the car. Thus, a potential attacker can't spontaneously send data to the car.

As a first barrier against the outside network it is also crucial to harden the module itself. To minimize the chance of successful attacks on the module, kernel hardening as well as a specially adopted Linux operating system will be used. Only absolutely necessary software is allowed to run on the SCM. In addition to reduction of the attack surface and minimizing the impact of an attack, also firewalling capabilities shall be implemented.

Within this thesis an ARM Cortex-A9 board running an adopted Linux will be used as a platform for the SCM. For ensuring stability and security state-of-the-art technology shall be re-used where appropriate.. After the design and implementation phase, the SCM functionally will be demonstrated within a real car. Here also the operational performance can be measured.

1.4 Expected Contribution

The design and implementation of a hardware-based SCM can improve security of vehicles connected to the Internet, by providing an innovative security concept which uses an own physical device for secure communication and protection of the internal car network. This thesis may also lead to further research in the area of secure communication in internet connected vehicles. Although this thesis is done with focus on automotive security, many ideas may be also applicable within the Internet of Things (IoT)

security field.

Since the thesis is written within the AVL LIST GmbH¹, its outcome has also direct value for the company. First the knowledge gained within this research work can be used as baseline for further developments and secondly the resulting SCM can be used as prototype and proof of concept to demonstrate the applicability of the solution.

1.5 Research Delimitation

The SCM is designed as a small piece of a much bigger overall security architecture shown in Figure 1.2. Typically four layers of security mechanisms are combined to obtain a holistic security approach. Layer 4, the outermost level is directly connected to the Internet communicating via TCP/IP. Its purpose is to perform basic filtering to only allow legitimate traffic to come into the car's network. On this level the type of traffic is more important than the content. Technologies which can be used are for example firewalling, to allow only traffic from/to legitimate communication partners, or intrusion detection by analyzing the traffic for suspiciously looking patterns. More than this, also encryption/decryption takes place here. In Figure 1.2 the "Secure Communication Module" represents Layer 4. Layer 3 has a connection to the internal car network and a non routable connection to Layer 4. On this layer the data gets processed. This layer also represents the separation of the internal and external car network. Within Figure 1.2 the "Internal Security" cloud covers Layer 3. Layer 2 and Layer 1 protect the internal Controller Area Network (CAN) bus and the ECUs by for example using Secure Boot. They fall into the "secure CAN" cloud in Figure 1.2. This work's focus is only on developing a Layer 4 device. Layer 1 to Layer 3 are out of scope and won't be covered within this thesis.

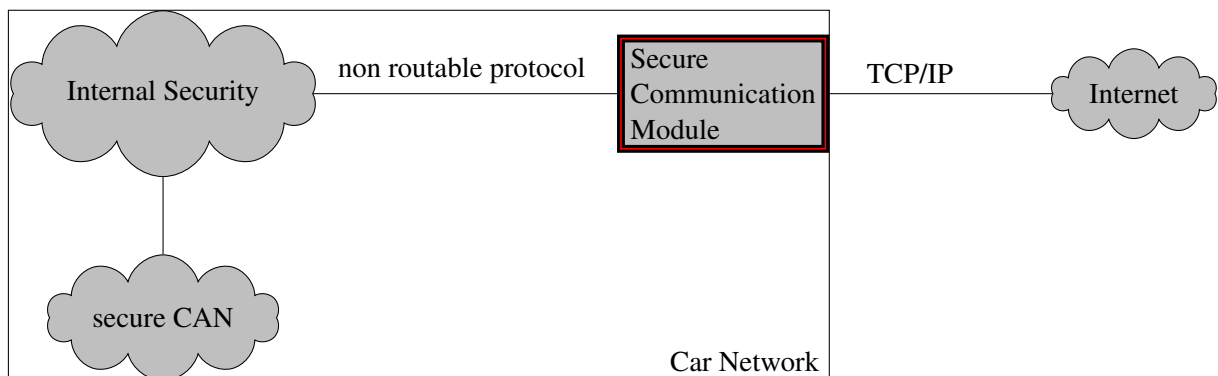


Figure 1.2: Secure Communication Module in the overall security architecture

Beside technical considerations, also privacy aspects have to be taken into account. Driver related data which gets send out from the vehicle shouldn't violate any personal rights of the driver. Especially PAYD

¹<https://www.avl.com/> [Online; last accessed 2017-10-16]

insurance tariffs have to find ways to protect privacy, as they collect huge amounts of privacy-sensitive data [19]. Anyhow, this is out of scope for this thesis, since the SCM doesn't care what data it is sending.

1.6 Thesis Outline

After this introduction, Chapter 2 provides background information of technology used within this thesis. In Chapter 3 an overview of related work will be given. Chapter 4 describes the research method used for this thesis and how it is applied. Afterwards the design is described in Chapter 5. Following to the implementation and evaluation in Chapter 6, Chapter 7 discusses the findings of this thesis. At the end Chapter 8 draws an conclusions and points out some future work of this topic.

Chapter 2

Background

This chapter gives some background information of technology which is used within the thesis project. It is divided into three sections, starting with the required protocols which were used. Afterwards all necessary Linux components which are part of the security concept are explained. In the last section information about relevant technology not fitting in the first two sections is given.

2.1 Required protocols

Different protocols are used by the SCM. This section provides basic information how they work and what they are used for.

2.1.1 MQTT protocol

The Message Queue Telemetry Transport (MQTT) protocol is a Machine-to-Machine (M2M) communication protocol, standardized in ISO/IEC 20922:2016 [20]. It is a very lightweight, asynchronous publish/subscribe protocol running on top of the Transmission Control Protocol (TCP). In normal request/response protocols when client C wants to receive data from server S, C sends a request to S and S answers with a response containing the data. Within publish/subscribe protocols data transmission is done in a different way. MQTT requires three parties for sending data:

- **Publisher:** The client sending the data.
- **Broker:** The server in between to handle communication between the clients.
- **Subscriber:** One or more clients receiving the data.

To send data from a publisher to a subscriber, first the subscriber has to subscribe on the broker. Afterwards the publisher publishes its data to the broker. Lastly the broker sends the data to all subscribed

clients. Because the whole communication is done via the broker, the communicating clients don't even have to know each other. The connection between publisher and subscriber is done via topics. Figure 2.1 illustrates the publish/subscribe communication over MQTT.

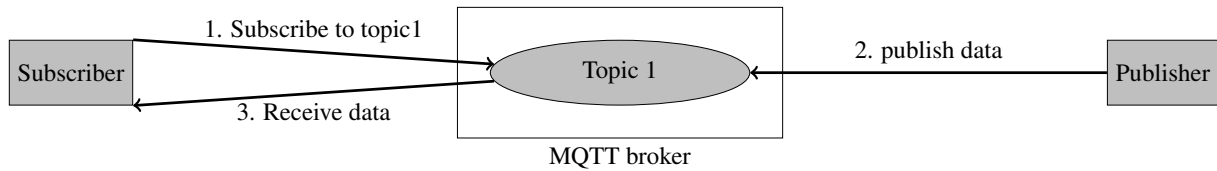


Figure 2.1: MQTT publish/subscribe communication

To allow a bi-directional communication each client can be a publisher as well as a subscriber. It is possible that multiple clients are subscribed to a specific topic on a MQTT broker. Also different clients can publish to the same topic. A broker can host multiple topics [21][22].

Within this thesis Eclipse Mosquitto^{TM1} was used for the MQTT broker. It is a open source implementation of an broker following the MQTT protocol version 3.1.

2.1.2 Transport Layer Security (TLS) protocol

The TLS protocol is a cryptographic protocol, providing communication security for client-server communication over insecure networks. It is the successor of the Secure Sockets Layer (SSL) protocol. Therefore it is often spoken from SSL/TLS encryption. The protocol provide three important cryptographic features:

- **Confidentiality:** Only parties which are allowed to read data can read them. This is ensured by using encryption.
- **Integrity:** The transmitted data have to be the same at the senders and receivers site. During transport no modification of data is allowed. In the TLS protocol data integrity is provided via keyed Message Authentication Codes (MACs).
- **Authenticity:** The receiver can verify the identity of the sender. This means that nobody can send a message in a name of someone else. In TLS authenticity is provided via the use of authenticated encryption.

TLS is a stateful protocol, which means communication is done in two steps: First an connection is established between the two communication partners and afterwards this connection is used to transmit data. For establishing a connection TLS defines an own protocol, the TLS Handshake Protocol. This protocol

¹<https://mosquitto.org/> [Online; last accessed 2018-05-20]

is used to agree on then protocol version and the cryptographic algorithms when starting a connection between a client and a server. Furthermore the communication partners use public-key encryption techniques for the generation of shared secrets and optionally authenticate each other in the handshake phase. For authentication digital certificates are used. Within the thesis project only client-authenticated TLS connections are used. This means that not only the client checks the servers certificate, but also the other way round. Therefore both the client and the server check the identity of their communication partner. Figure 2.2 illustrates the TLS handshake protocol containing client authentication containing the following steps [23]:

1. The client sends a *ClientHello* message to the server.
2. The server responds with a *ServerHello* message. The server sends its certificate to the client. The server sends a *Server Key Exchange Message* to the client. The server sends a *Certificate Request* message to the client. The server sends a *Server Hello Done* message to the client.
3. The client sends its certificate to the server. The client sends a *Client Key Exchange Message* to the server. The client sends a *Certificate Verify* message to the server. The client sends a *ChangeCipherSpec* message to the server. The client sends a *Finished* message to the server.
4. The server sends a *ChangeCipherSpec* message to the client. The server sends a *Finished* message to the client.

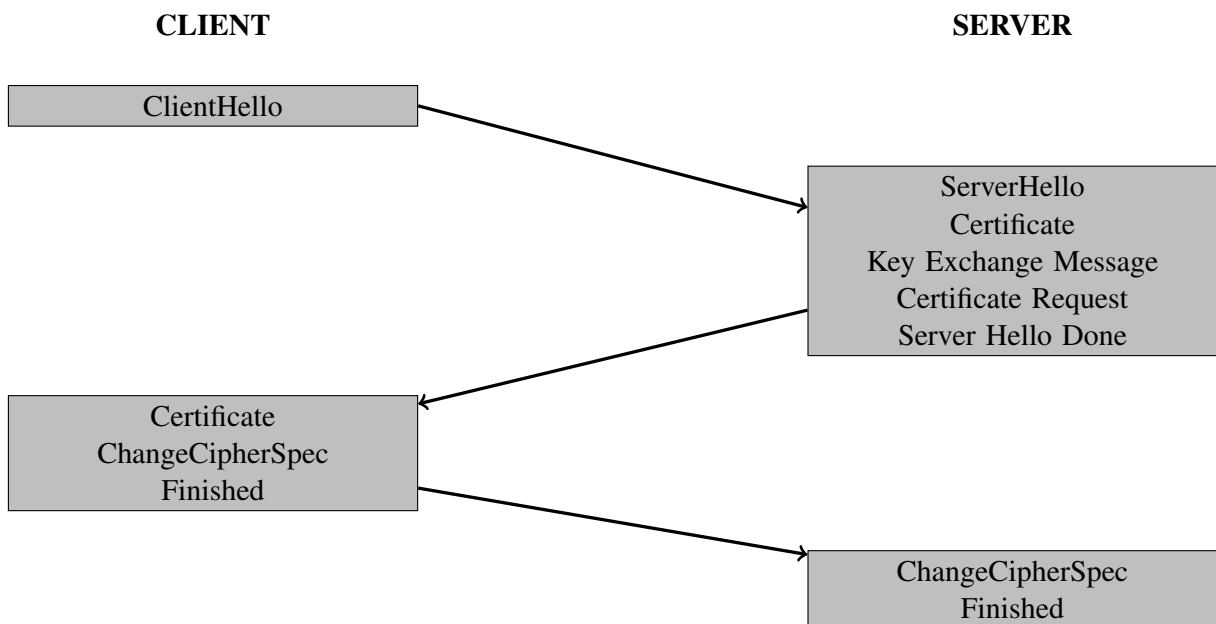


Figure 2.2: TLS handshake protocol using client authentication

2.1.2.1 OpenSSL

OpenSSL² is a software toolkit for the SSL and TLS protocols. Beside other tools it contains an open-source implementation of these security protocols. The core library, implementing basic cryptographic functions and various utility functions, is written in the C programming language. Also wrappers for many other computer languages are available. The OpenSSL library is available for all major operating systems and spread very widely. Within this thesis OpenSSL was used for all cryptographic functions related to TLS [24] [25].

2.1.3 Cryptographic Message Syntax (CMS)

The Cryptographic Message Syntax (CMS) is a standard for cryptographically protected messages. It was written by the Internet Engineering Taskforce (IETF) and published as Request for Comments (RFC) 5652 [26]. The CMS defines the following operations for any form of data:

- **digitally sign:** Proofs that the message comes from a known sender (authentication), that the message wasn't changed (integrity) and the sender can't deny that he sent the message (non-repudiation).
- **digest:** Calculation of a cryptographic hash function.
- **authenticate:** The receiver can verify the source of the message.
- **encrypt:** Modify a message in a form that only authorized parties can read it.

Data protection is reached by using the encapsulation syntax described in the CMS [26]. Also nested encapsulation is possible. For example an already encrypted message can further get signed, which adds two layers of encapsulation.

2.2 Required Linux components

Background information about all relevant Linux components which were used for securing the SCM are described in this section.

2.2.1 Netfilter/iptables

Netfilter is a packet filtering framework inside the Linux kernel. It consists of a set of hooks within the Linux kernel. Therefore kernel modules are able to register callback functions with the kernel's network

²<https://www.openssl.org/> [Online; last accessed 2018-05-20]

stack which are called whenever a packet crosses the particular hook within the network stack. Build on netfilter, iptables provides a management interface for defining firewall rules, which are defined in tables. Every table contains multiple chains. These chains are connected to the different hooks provided by netfilter. Every chain is a list of rules, which are processed one after another. A rule contains a description of the packets it should be applied to, as well as an action which should be performed (mostly accept or drop the packet). Netfilter/iptables is a first-match firewall, meaning that the first matching rule is executed and all subsequent rules are ignored [27][28].

2.2.1.1 Tables and chains

To get a better understanding of the tables and chains including their interaction, an description of the build-in tables and chains follows. This knowledge is also helpful to understand certain decisions made during the design of this security concept. A standard iptables setup consists of four build-in tables [29]:

- *Filter table*: This is the default table. If no table is explicitly defined, the filter table will be used as default. Usually firewall rules whether to drop or accept a packet are defined here.
- *Nat table*: Beside packet filtering, iptables is also capable of doing Network Address Translation (NAT) [30]. The rules for changing either the source or destination address of a packet are placed in this table.
- *Mangle table*: Within this table, rules for modifying specific fields of the ip header can be implemented.
- *Raw table*: The only purpose of this table is to avoid connection tracking for specific packets. Connection tracking within netfilter/iptables is explained in Section 2.2.1.3.

Beside the tables described above, also the chains are important. While tables determine the type of rules (e.g. filter rule, nat rule), the chains define when rules are triggered. There are four build in chains, but users can also add their own chains [29]:

- PREROUTING
- INPUT
- FORWARD
- OUTPUT
- POSTROUTING

Each chain is bound to a table. The correlation of the build-in tables and chains of iptables is illustrated in Table 2.1:

	PREROUTING	INPUT	FORWARD	OUTPUT	POSTROUTING
raw	X			X	
mangle	X	X	X	X	X
nat	X			X	X
filter		X	X	X	

Table 2.1: Correlation of build-in tables and chains [29]

Figure 2.3 illustrates how packets move through the different build-in chains of iptables. First the packet runs through the PREROUTING chain of the raw, mangle and nat table. Afterwards a routing decision is done, whether the packet is for the local machine or has to be forwarded.

- In case of a local packet, the INPUT chains of the mangle and filter table are processed for matching rules. If the packet was able to pass all the rules down to the filter's INPUT chain, it is sent to the local machine.
- When the packet has to be forwarded, the FORWARD chains of the mangle and filter tables get processed.

All outgoing packets, which get sent from the local machine first start with a routing decision (e.g. what source address or which outgoing interface to use). Afterwards the OUTPUT chains of all tables are processed before the next routing decision, where also the external packets which have to be forwarded by the local machine join the same path. As a last step the POSTROUTING chains of the mangle and nat table are considered before sending a packet to the network [31].

2.2.1.2 Stateless vs. stateful firewalls

Packet filtering firewalls can be divided into two main groups: stateless and stateful firewalls. The difference between these two are the different fields that can be used for decision making. Within stateless systems filtering decisions are only based on the packet header.

The major problem of stateless firewalls is that every packet is processed individually. In contrast, stateful firewalls also know about packets which were sent earlier. This allows more complex filtering rules based

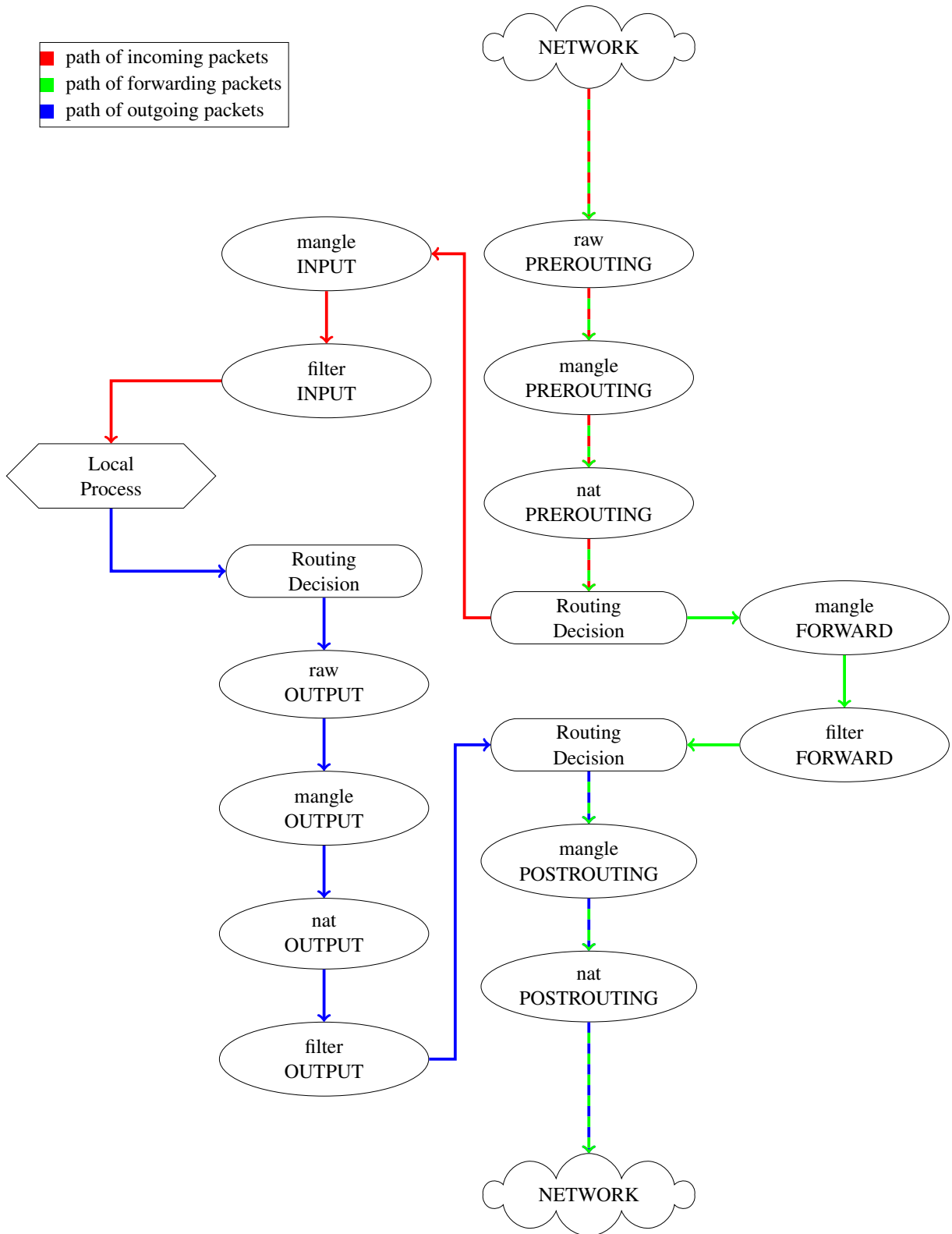


Figure 2.3: Traversing path of packets through the build-in chains in iptables [31].

on previous packets. Such configurations wouldn't be possible with a stateless firewall. Netfilter/iptables can be used as a stateful firewall [32].

2.2.1.3 Connection Tracking

As already mentioned in Section 2.2.1.2, within stateful firewalls, connection based filtering has to be possible. Therefore information about the connections have to be stored. In the Linux kernel, connection tracking is built on the netfilter framework. The stored fields are [33]:

- source IP address and port
- destination IP address and port
- protocol types
- state
- timeout

Whenever a new packet arrives, a lookup for existing connections is done. If there is nothing found having the same attributes than the packet, a new connection will be created. Anyhow a new connection entry only gets inserted in the conntrack table, when the packet successfully leaves the netfilter framework. This means no entry will be created for packets which are dropped by firewall rules. A connection entry gets removed from the conntrack table when the time defined in the *timeout*-field expires without receiving any packets of this connection [32][33][34].

Even though netfilter and iptables are sometimes used as synonyms, the one is the kernel portion, while the second is the userland tool to create the actual rules. Both parts are required to get a working packet filtering firewall.

2.2.2 Linux Security Modules (LSM)

Wright et al. [35] give a good overview of the Linux Security Modules (LSM) framework. This framework provides a very generic way to implement different security models running inside the Linux kernel, by loading them as an kernel module. To achieve this LSM interrupt the access to the kernel's internal objects. Just before the resource is accessed a LSM hook calls the loaded security model whether the access should be permitted or denied. To use a different security model to only thing to do is to load the kernel module and register it with the LSM hook. Figure 2.4 illustrates the LSM architecture. Every time a user level process performs a open system call, to kernel looks up the required inode, performs error checks and does its Discretionary access control (DAC) checks. When access is granted based on the DAC rules, the LSM hook asks the security module for a decision. The policy engine of the current module processes the request and permits or denies the access.

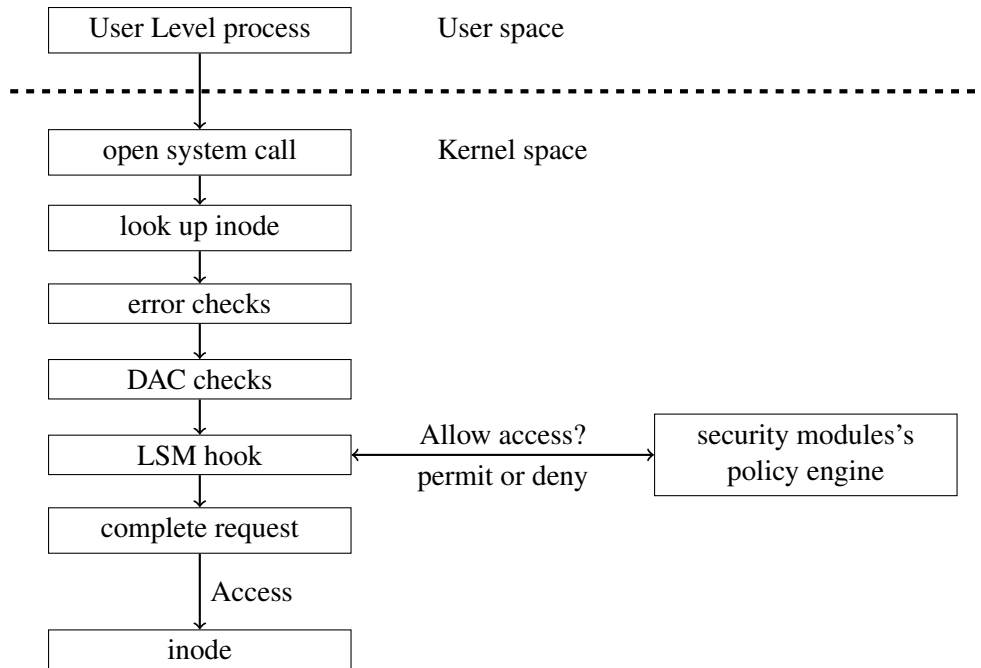


Figure 2.4: LSM architecture (adopted from [35])

2.2.2.1 Security-Enhanced Linux (SELinux)

Security-Enhanced Linux (SELinux) is a Mandatory access control (MAC) system for Linux based on the LSM framework. It was developed by the National Security Agency (NSA) to provide flexible fine grained access controls within the Linux kernel. SELinux was included in the main Linux 2.6 kernel and is part of the core Linux set since then [36, Chapter 1].

Discretionary access control (DAC) vs. Mandatory access control (MAC) DAC and MAC are two different approaches for access control. Their main purpose is to only permit resource access to authorized entities. A good overview on how Mandatory access control mechanisms can solve drawbacks of Discretionary access control systems is given in [36, Chapter 1]. The most used access control type nowadays is DAC, which is also the case for standard Linux security. Thereby an individual user or program (usually the resource owner) defines the access rules. This means every resource owner can specify and change the access rules for its resources. This requires an trusted environment where all users and programs, which are allowed to change access rules, are totally trustworthy. Especially for programs it is unrealistic that there are no flaws which can be exploited. To solve this issue, MAC manages all access rules in a central security policy. Individual programs can not change this security policy.

SELinux type enforcement In SELinux type enforcement is used to control access of subjects to objects [36, Chapter 1]. In Linux subjects are the processes which try to perform the access. Objects are

the resources on which an action is applied. Using type enforcement, every subject and every object has a type identifier. Access rules in SELinux define which types of subjects are allowed to access which types of objects. This allows very flexible and detailed configuration of access controls.

2.2.3 Communication sockets

A communication socket is an endpoint for sending and receiving data. It can be seen as an open file where data can be read and written. Sockets are used to enable communication between different programs or even different devices. To establish a communication channel between two programs, both have to connect to the communication socket and afterwards they can exchange data with each other. There are different types of sockets based on the communication protocol they are built on. Within this thesis two different types are used. Firstly Inter Process Communication (IPC) sockets for communication within the same machine (see Section 2.2.3.1), and secondly network sockets to communicate between multiple machines (see Section 2.2.3.2) [37].

2.2.3.1 Inter Process Communication (IPC) socket

One special type of communication sockets are IPC sockets. They are also known as Unix domain socket. IPC sockets are used for communication between processes on the same host. They provide reliable transmission of byte streams. IPC sockets use the filesystem as their address namespace [38][39].

2.2.3.2 Network socket

In case two processes running on different machines want to communicate, network sockets can be used. In this case all the data between them are sent over a network communication. In the case of Internet Protocol (IP)-networks there are two types of network sockets:

- **Stream sockets:** Reliable transmission of byte stream over TCP.
- **Datagram sockets:** Transmission of individual messages with up to 65,500 bytes using User Datagram protocol (UDP).

Within this thesis project, TCP stream sockets were used for network communication between processes [37].

2.2.4 Virtualization technology

Virtualization can be used as a security mechanism by running sensitive processes on different virtual machines to isolate them. In case an attacker is able to access one virtual system, he still can't access

other virtual systems or even the host system. To provide isolation for different tasks running on the same machine, virtualization can be used. Therefore multiple virtual machines can run on one device and are isolated from each other. There are four different approaches of resource virtualization [40][41]:

- **Full virtualization:** This virtualization technique is also called hardware emulation. A hypervisor is used to run an unmodified operating system on a host machine. The supervisor translates the instructions from the virtual machine to the host. The performance suffers from the high overhead.
- **Paravirtualization:** Similar to the full virtualization a hypervisor is used between the virtual machine and the host system. But for paravirtualization the virtualized operating system has to be changed to be hypervisor-aware. This increases the performance, but reduces the flexibility.
- **OS-level virtualization:** With OS-level virtualization no hypervisor is required. Multiple isolated instances of an operating system are running on a single host machine. Therefore the host operating system has to be modified to provide this isolation. It is very performant but not so stable. Because all instances run on the same kernel, an comprised kernel effects all instances.
- **Native Virtualization:** In native virtualization multiple unmodified operating systems can be running on the host system. In contrast to full virtualization, the virtual machines run directly on the physical processor of the host machine, which supports hardware virtualization. This increases the performance of the virtual machines.

In this thesis Linux Containers (LXC) was chosen to provide OS-level virtualization (see Section 2.2.4.1).

2.2.4.1 Linux Containers (LXC)

LXC is a virtualization method for running multiple isolated Linux systems on a single host device. No hypervisor or emulation is required, making this technique very resource and performance efficient. The single instances of Linux systems are called containers. All processes of the containers are running as normal processes on the host system. Therefore all processes and the whole filesystem of the containers are completely visible and accessible from the host. But the single containers only see their own resources. Resource management is done using control group (cgroup) (see Section 2.2.4.1, control group (cgroup)), while kernel namespaces provide the required isolation (see Kernel Namespaces) [41] [42].

control group (cgroup) cgroup is a Linux kernel feature to organize processes into hierarchical groups. Using this groups different types of resources, like memory, CPU or disk IO, can be monitored and limited. cgroup is organized in form of a tree structure. Every process is connected to exactly one cgroup. When creating a new process it will be in the same cgroup than the parent process. Having

this hierarchical structure resource control for a process and all its subprocesses becomes possible. One practical example could be to limit the memory usage of a program [43][44].

Kernel Namespaces Kernel namespaces are used to isolate global system resources to provide processes inside a namespace only access to the system resources they need. This allows containerization were multiple containers are running on the same system, but see only their own global system resources. Six different namespaces were used in this thesis [45][46]:

- **Mount Namespaces:** Isolates the mount points. Processes in different mount namespaces can have different filesystem hierarchy. Mount namespaces can be used to create a similar environment to chroot jails, were any directory of the host can be used as root directory for the jail. Processes inside this isolated area can't access files outside their file hierarchy. With LXC mount namespaces can be used to mount resources which can't be seen from any other container or even the host system.
- **UNIX Timesharing System (UTS) Namespaces:** Two system identifiers returned by the `uname()` system call are isolated with this namespace. Namely the *nodename* and *domainname*. This enables possibility of having multiple hostnames when using containers.
- **IPC Namespaces:** Isolation of IPC resources including *System V IPC* objects and *POSIX message queues*.
- **Process identifier (PID) Namespaces:** Here the PID number spaces gets isolated. This means that the same PID can be used in different PID namespaces and the same process can have a different PID in different namespaces, for example inside a container and on the host system. Furthermore this isolation only shows the own PIDs within a container.
- **Network Namespaces:** Isolation of all network related system resources, including network devices, IP routing tables, network addresses, port numbers and more. Within containerization this namespace type can be used to provide an own virtual network setup for each container.
- **User Namespaces:** This namespace provides isolation of the user identifier (uid) and group identifier (gid) of processes. Therefore one process can have a different uid/gid inside and outside of an user namespace.

secure computing mode (seccomp) Another security feature of LXC containers are secure computing mode (seccomp) filters to limit available system calls. There are two different approaches of system call

filtering. The first is a whitelisting approach where all system calls which should be allowed are defined. Only these calls are allowed and all others are blocked. All required system calls which are used from all processes within the container have to be known. In contrast blacklisting is also possible by defining a list of forbidden system calls. Only these system calls will be blocked and all others are allowed [47][48].

Unprivileged containers Usually LXC containers are running as the host's root user to have access to the resources of the host machine. This also means that all processes inside the container are running as root which is highly critical in case an attacker is able to break out of the container, having root privileges. Therefore the user namespace was introduced in Linux kernel version 3.12 which allows unprivileged containers. With a mapping of the gid and uid from the users inside the containers to less privileged accounts on the host system even the root user of the container doesn't have critical privileges on the host system. With unprivileged containers no root privileges are needed to run LXC containers and processes of the container run as a user, without privileges on the host system. Even if an attacker is able to start a process as root inside the unprivileged container and break out of the container's isolation, the process can't access any resources on the host system [49][50].

Networking When running in an own network namespace LXC containers can't access the network resources of the host system. In case a container needs network access to communicate with the host or even other machines virtual network interfaces can be used. LXC provides different types of virtual network access [48]:

- **none:** The container shares the network namespace of the host and therefore has direct access to the host's network infrastructure.
- **empty:** Only the loopback interface will be created for the container.
- **veth:** With virtual ethernet pairs the container gets access to a network bridge of the host system. The bridge has to be present and configured on the host before starting the container.
- **vlan:** Using virtual lan interfaces the container's virtual interface is linked to a physical interface of the host.
- **phys:** The container gets directly connected to an physical network interface of the host system.

2.3 Additional security concepts

In addition to the above technologies two more security terms are also relevant for this thesis. They don't fit into any of the other sections and are therefore explained in an own section.

2.3.1 (Distributed) Denial-of-Service attacks

(Distributed) Denial-of-Service ((D)DoS) attacks form an own group of computer attacks, with the goal of hindering a resource to work properly. (D)DoS attacks are commonly used to disrupt network devices. Every network device can only handle a specific number of network requests at a time. When this number is exceeded, the device won't work properly anymore. This is used by (D)DoS attacks, where a resource gets flooded with network requests, until it stops from working. Within a distributed attack, the packet stream is sent from different sources. Therefore it is much harder to detect [51][52].

2.3.2 Intrusion Detection and Prevention Systems

Intrusion Detection and Prevention Systems (IDPS) is a collective term for both Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs), where the first only passively detects intrusions attempts, while the later also responds on detected anomalies.

The purpose of IDSs is to monitor all network traffic and detect malicious traffic. In case of a network anomaly, an alarm is triggered. In comparision to firewalls, not only the packet header but also the payload gets analyzed. Therefore a packet can be inspected in greater detail. In addition to the detection of intrusions, IPSs can also take actions based on the identified thread. The easiest example of such a reaction can be to just drop the malicious packets [53] [54].

Chapter 3

Related Work

Even though security of connected cars is a rather new topic there was already some research done in this area. This chapter provides an overview of the related work and summarizes what was already done. Thereon a research gap is identified which leads to the research questions formulated for this thesis.

One technique to protect the communication with the outside of the car is the usage of firewalls and Intrusion Detection Systems (IDS) or Intrusion Prevention Systems (IPS), which are already well-known concepts in the desktop IT domain [55] [56] [57]. Anyhow, automotive IDS/IPS have some different requirements compared to normal IDS/IPS within the computer world. One big inequality is the difference in computing power since components inside cars are very resource-limited, especially in terms of CPU power and storage size. Also the used networking protocols differ between computer networks and vehicular networks. In contradiction to the hardware limitations, the automotive domain also has very strict real-time requirements, because of the potential safety implications of an attack [56] [57].

Beside software-based security measures also hardware-based approaches came up in the last few years. The advantage of outsourcing security functionality to an own device can be better performance since such a security module will be specially designed for its security tasks. An additional device can also provide better isolation of the network to protect. In case an attacker manages to get access to the security module, he still doesn't have access to any functionality of the car. Wolf and Gundrullis [58] developed a Hardware Security Module (HSM) which provides a holistic protection of all important ECUs and the communication between them. The authors suggest three versions for different use cases. There is a full variant, which is designed to protect connections from external vehicle interfaces. The medium module aims at the protection of the in-vehicle communication, while the light module can be used to secure the communication between ECUs and sensors or actuators. Also Idrees et al. [59] describe a security

concept using these three types of HSMs. The paper presents a protocol for secure OTA updates. These papers are based on the E-Safety Vehicle Intrusion Protected Applications (EVITA) project [17] and describe the general idea of using HSMs to protect the internal vehicular network. It focuses more on the properties of the hardware and theoretical ideas, than on a practical implementation. It is still questionable how practical and cost effective an own HSM for each ECU would be.

Spaan [60] came up with a more secure way to do OTA software updates for cars. The current state of automotive security, with focus on securing communication to external networks outside the car, has been analyzed by the author. This analysis showed that there are quite some theoretical ideas but no real standards covering automotive security on the wireless level [60].

3.1 Summary

The literature review on related work has shown that the scientific community agrees on the importance of a proper security concept to ensure secure communication of connected cars. Anyhow, since cyber security is a rather new topic within the automotive industry there is still a long way to go. There was multiple research done on securing the vehicle's internal network but lesser was worked on securing the communication interface. The very interesting idea of using hardware-based security modules, have been already brought up by some researchers, especially by the EVITA project [17]. But they focused more on the theoretical idea than on the practicability of the solution. Another finding of the literature review is the lack of standards covering secure communication of connected cars. There are lots of standards and regulations for network security within computer systems, but not for connected cars security.

3.1.1 Research gap

The research gap this thesis wants to focus on, is the lack of further research regarding hardware-base security modules for securing the communication interface of a connected car. No research was found on how to practically implement this idea in form of a prototype setup. This leads to the research questions which are formulated in Section 1.2.

Chapter 4

Research Method

The selected research methodology for performing this thesis work is DSR as described in [61]. In Information Systems research there are two main paradigms, which are behavioral science and design science. While behavioral science aims on developing and justify theories which explain or predict human or organizational behavior, the purpose of design science is to generate knowledge and innovation. As an artifact-centered problem approach, within design science a problem gets identified and afterwards an IT-artifact will be developed to solve the problem. The meaning of "IT-artifact" in this context is quite broad. In [61] artifacts are defined as "innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, and use of information systems can be effectively and efficiently accomplished".

DSR seems to be very suitable for this thesis, since the goal is to develop a hardware-based SCM. This module can be seen as an IT-artifact improving the problem of security issues in connected cars. Also this thesis is about designing and implementing an artifact, which doesn't exist in this way.

4.1 Design Science Guidelines

Hevner A., March S., et al. [61] defined seven guidelines for Design Science in Information Systems Research. In the following subsections this guidelines will be discussed in the perspective of this thesis project, which further demonstrates the applicability of DSR for this thesis.

4.1.1 Design as an Artifact

Within design science the goal is to come up with an IT-artifact which is the result of DSR. This artifact has to address an relevant problem and improve it. In the case of this project the artifact is the SCM.

4.1.2 Problem Relevance

Each DSR must address a relevant problem which is of interest to the community. The relevance of this project is illustrated in Chapter 1.

4.1.3 Design Evaluation

An artifact must be also evaluated to proof its utility, quality and efficacy under the usage of well-executed evaluation methods [61]. In this thesis project both the operational and security performance will be tested in Chapter 6.

4.1.4 Research Contributions

The DSR must also provide contribution to the research community. Section 1.4 describes how this thesis provides such a contribution.

4.1.5 Research Rigor

Use of rigorous methods during the construction and evaluation have to be used. The thesis project will use well-established methods both during the construction and evaluation phase.

4.1.6 Design as a Search Process

Design science is an iterative problem solving process. An identified problems is tried to be solved using a new IT-artifact. Using the design evaluation this artifact is tested against its requirements. In case of some new problems within the new artifact another DSR can be used to solve these.

4.1.7 Communication of Research

To provide contribution a DSR also needs to be presented in a appropriate way. This thesis contains two chapters covering it's communication. In Chapter 7 the results are explained and discussed, while Chapter 8 summarizes the whole project and gives an outlook on future work which could be done in this area.

4.2 Design Science Implementation

In [62] a DSR methodology is presented, providing a framework on how to perform a DSR in information systems. The authors identified six steps which should be followed. After the first initialization steps

"Problem identification and motivation" and "Definition of the objectives for a solution", there is a three step development loop containing "Design and development", "Demonstration" and "Evaluation". In case the evaluation fails, the process goes back the "Design and development" step. This loop gets repeated until the evaluation succeeds. After a successful evaluation, the "Communication" step communicates the result to all relevant parties. This process is also depicted in Figure 4.1.

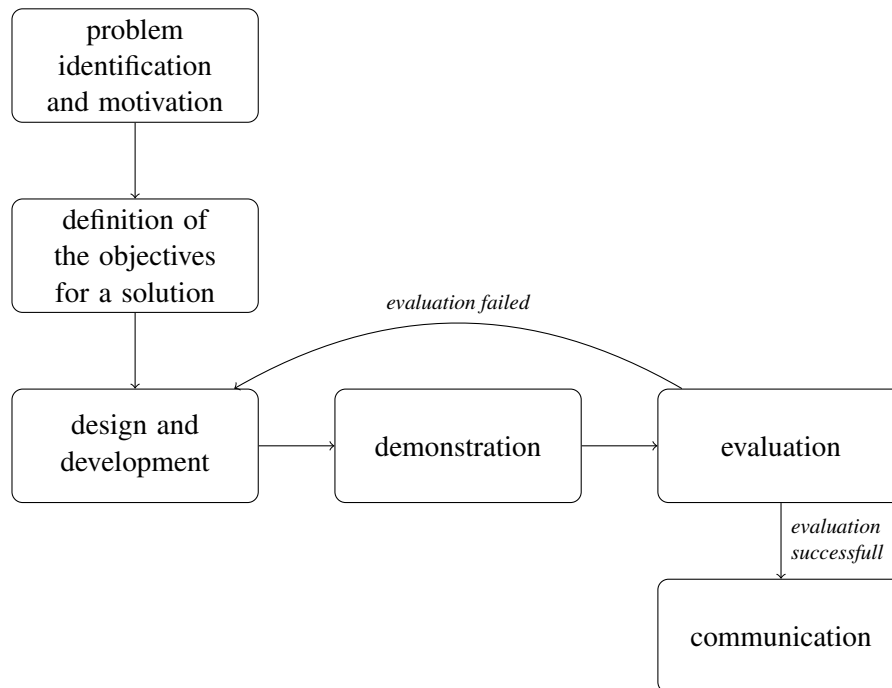


Figure 4.1: Steps of DSR methodology

Afterwards a section for every step describes how this step will be executed in this thesis project:

4.2.1 Problem identification and motivation

As a first step a clear picture of the project is required. The research problem has to be identified and described. The artifact will be build upon this problem definition, to provide a solution. Additionally the relevance and value of such a solution has to be justified. This step is done in Chapter 1 and Chapter 3, where the security problem of connected cars is shown along with a study of related work already done in this area.

4.2.2 Definition of the objectives for a solution

Clear definition of objectives are crucial already in the beginning of the project. This is required to have an exact idea where the work should lead to. The objectives are based on the problem definition. Within this thesis, Section 1.3 points out how the proposed solution should look like.

4.2.3 Design and development

Within this phase the artifact, which should solve the problem covered in the problem definition, gets developed. First the required functionality and architecture of the artifact has to be declared. Afterwards the artifact will get implemented. For the SCM an ARM Cortex-A9 board with a modified Linux running on it will be used. The required security measures can be grouped in two main groups:

- *Defensive measures*: Protecting the internal car network from external threats. Being directly connected to the internet, the SCM has to prevent any malicious communication from entering the car's network. This first defense layer should already block as much external attacks as possible.
- *Hardening measures*: Harden the SCM to protect the module itself. This is crucial, since vulnerabilities within the SCM could be used by attackers to bypass its defense mechanisms. Also in case of an successful attack on the SCM, the attacker shouldn't be able to get access to the internal control network of the car.

In addition to the security measures, the SCM also has to provide a way for the car to communicate with the internet. Having the principle of maximal isolation of the internal car network, only a non-routable connection between the SCM and the internal network is allowed. This requires the development of a communication concept with the two main aspects of security and maximal isolation.

As an overall requirement only well-known standard Linux security techniques should be used. At the beginning research on different applicable techniques for both groups will be performed resulting in an optimal set of security measures (see Chapter 5). Within the next stage the identified security mix will be implemented on the SCM, as described in Chapter 6.

4.2.4 Demonstration

This step shows how the artifact can be used to solve the research problem. It should demonstrate the functionality of the artifact. A problem with demonstrating security features is that security can't really be seen, especially when it's purpose is to only block malicious activities, but in best case don't affect the functionality at all. Therefore the main focus of demonstrating this thesis is based on proving the integration of the SCM within the wider overall security architecture. Within the company there is already a implementation for OTA updates, which can update the firmware of specific ECUs inside a test car. The SCM will be attached to the car and afterwards a software update will be performed. This should verify that the security functionality doesn't interfere the normal operation and integrates well in the overall structure. Further description about the demonstration is given in Chapter 6.

The only way the author can think of presenting the security is a penetration test, which is also used for evaluation. A more detailed description of the setup and the results of such a security demonstration can be found in Section 4.2.5 and in Chapter 6.

4.2.5 Evaluation

To insure the effectiveness of the artifact, the functionality has to be evaluated based on how well it solves the identified research problem. As shown in Figure 4.1, after this step an iteration back to Design and development (see Section 4.2.3) may be necessary for further improvements. In this thesis project the evaluation can be split into two parts, where the operational evaluation checks the functionality and integration of the SCM, while the evaluation of the security performance considers whether the security requirements are met. Both parts are required to prove the applicability of the artifact to solve the research problem.

4.2.5.1 Operational evaluation

The main aspect here is the successful integration of the SCM within the companies overall vehicular test infrastructure. All functionality which was able to be done without the SCM also has to work with the new component in place. The following two activities will be tested:

- *OTA update:* The firmware of a specific ECU gets stored on a server located inside the company. The car opens a connection to this server, downloads the firmware and installs it on its ECU. Afterwards a log file with all relevant information of the installation process gets sent to the server.
- *Collect telemetry data:* On a server inside the company an telemetry collection request can be formulated. It contains a list of values which should be collected, an interval on which the specified values should be read, as well as the duration of the total measurement. When the car gets such an request, it starts the data collection and sends the data back to the server.

Both use cases have to work with a realistic setup using the test vehicle and the internet connection of the SCM. Furthermore also the operational performance will be measured to ensure functional usability. This will be tested by executing an OTA update without the SCM. Afterwards the same update will be performed with all the security mechanisms active. This two rounds will be compared. Possible performance criteria may be:

- CPU usage
- memory usage

- bandwidth
- time from the initiation until completion of the update

From all this criteria the overall run-time of an update was identified as the most important criteria, because it highly affects the usability of the whole system. For all other points it is sufficient to not hinder the functionality of all internet connected communication, which is tested by the functionality tests.

4.2.5.2 Security performance

The second evaluation phase will consider the security performance. This evaluation will be done using a penetration test to evaluate the defense capabilities. The test will be executed by an external company specialized on security assessments. Therefore the security of the SCM against external attacks will be checked.

4.2.5.3 Evaluation criteria

Table 4.1 summarizes the evaluation criteria which were identified in the previous sections. It also shows how they get evaluated and which goal they fulfill. All criteria have to be fulfilled to proceed to the communication phase, as shown in Figure 4.1.

Goal	How to evaluate	Evaluation criteria
Keeping functionality	Performing both use cases in a realistic setup	Both use cases have to work
Keeping usability	Comparing run-time of OTA update with and without the SCM	The SCM shouldn't increase the update time by more than three minutes.
Providing security	Penetration test of external company evaluating the defense capabilities against external attacks	No unprivileged access to SCM should be possible

Table 4.1: List of evaluation criteria

Within Chapter 6 the whole evaluation activity is documented in detail. The results are discussed in Chapter 7 and are also considered in Chapter 8.

4.2.6 Communication

At the end, the work has to be communicated to fulfill the research contribution. The communication has to be done in a way that all relevant audiences are able to understand it. Important topics which should be covered here are the problem, the artifact and how the artifact can be used to solve the problem. In Chapter 7 the results are explained and discussed, while Chapter 8 summarizes the whole project and gives an outlook on future work which could be done in this area.

Chapter 5

Security Concept

As a first step within the development of the SCM a security concept was developed based on standard Linux Security mechanisms. The concept is split into three parts for security. The first two, which are about Defense Mechanisms and System Hardening, are used to prevent attacks on the SCM. The third part contains a concept how the internal car-network can communicate with the outside world, while maintaining maximum isolation of the sensitive internal car network.

5.1 Defense Mechanisms

Since the SCM works as the interface between the car network and the internet, it also presents the first layer of defense against attacks coming over the cellular network. Within this thesis there was a requirement that all connections to the outside have to be initiated by the car (see Figure 5.1). No incoming connections are allowed. Therefore, in case an external server needs to send data to the car, this server has to wait until it gets contacted by the car. Furthermore the SCM knows all its communication partners and doesn't allow connections with any other devices.

Having the principle of only allowing outgoing connections to known hosts already highly increases the security of the car's communication network. An attacker trying to send malicious data to the car, has to take three steps for establishing a connection:

1. Disguise himself as a legitimate communication partner.
2. Wait until the car opens a connection to him.
3. Prevent the car's actual target from answering the request.

To block unauthorized traffic, the SCM requires firewalling capabilities. Due to the requirement of only using standard Linux functionality, netfilter/iptables (see Section 2.2.1) was used for traffic control.

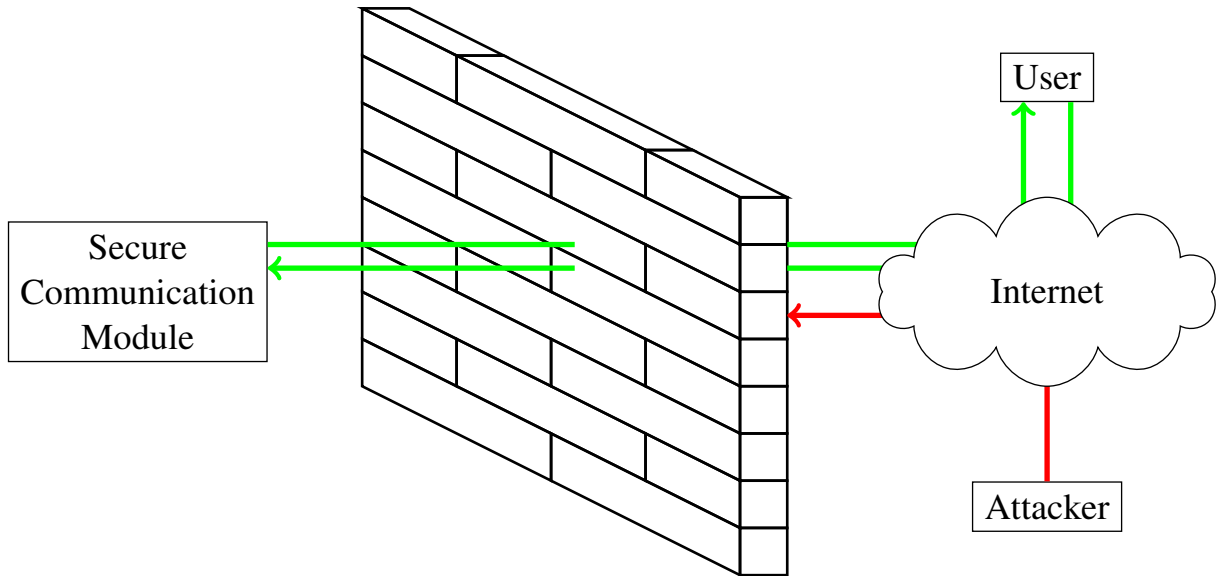


Figure 5.1: Firewall should only allow communications initiated by the SCM

5.1.1 (Distributed) Denial-of-Service protection

A special kind of attack which has to be addressed separately are (D)DoS attacks (see Section 2.3.1). Usually (D)DoS prevention is very difficult, because especially distributed attacks are difficult to differentiate from legitimate traffic. Within this thesis project there is the advantage of only having outgoing connections. Consequently all incoming traffic, which is not related to an outgoing connection can be dropped. This already highly decreases the chance of a successful (D)DoS attack. Anyhow, it is still possible to overload the network interface by sending malicious requests faster as they can be dropped. Therefore the speed of dropping incoming connection attempts has to be increased. As shown in Figure 2.3 all the *PREROUTING* chains are processed and a routing decision is made before the filter rules are reached. To speed up the detection and dropping of illegitimate incoming traffic, already the *mangle* table in the *PREROUTING* chain drops packets, which are not related to an car-initiated connection. Nevertheless, the technique described above only delays an (D)DoS attack and doesn't prevent it. Anyhow, since the main goal of this thesis work is the isolation of the internal car control network, temporary connection losses in consequence of an (D)DoS attack can be seen as acceptable. This is also because no time-critical data is transmitted in the current version.

5.1.2 Intrusion Detection and Prevention Systems

Another powerful defense mechanisms in computer networks are IDPS (see Section 2.3.2) to detect anomalies in network traffic. At the beginning of the design phase the author also planned to build IDPS functionality into the SCM. Unfortunately this isn't possible within the setup of this thesis project,

because all traffic running through the SCM is encrypted. Therefore its content can't be analyzed. Also providing the SCM with the required keys for decryption isn't adequate, since it would contradict the concept of isolation. When using an end-to-end encryption only the endpoints should be able to read the data. Nevertheless, since all internet connections have to be established from the car's side and authenticated encryption is used, only well-known servers can send data to the car. This generates an acceptable level of trust into the data, leading the author to abstain from using IDPS.

5.2 System Hardening

Being a security device also the SCM has to be hardened. A range of different Linux standard security functionality was chosen to protect the SCM against attacks. Within this thesis only the security of the software was considered. Physical security of the SCM was not in the scope of this work. The system security is based on kernel hardening (see Section 5.2.1) and isolation (see Section 5.2.2). Furthermore some additional hardening measures were chosen as described in Section 5.2.3.

5.2.1 Kernel Hardening: SELinux

Also the kernel of the operating system needs to be hardened because it doesn't make much sense to just secure all programs running on a system when the underlying kernel is still vulnerable. Therefore a cleanup of the kernel is necessary. The kernel should only contain modules which are required. For example the kernel doesn't need to provide WiFi access if the SCM doesn't even have a WiFi link. Beside the cleanup SELinux was chosen as MAC system to control access on system resources. SELinux (see Section 2.2.2.1) provides very fine-grained access control. Because the functionality of the SCM is well known and doesn't change a lot, it is possible to define allowed access on system resources. Every resource access beside normal system behavior should be blocked by SELinux.

5.2.2 Isolation: Linux Containers

Within this thesis isolation of sensitive processes is one of the main goals because further isolation increases security by minimizing the effect of a successful attack. Virtualization is an effective method of running multiple isolated systems on a single device (see Section 2.2.4). For this thesis project OS-level virtualization was chosen as the most sufficient variant of isolation. This is because OS-level virtualization is very performant and needs few resources, which makes it suitable for resource limited devices. LXC provides such virtualization for Linux systems (see Section 2.2.4.1). The use of unprivileged LXC containers, with all namespaces mentioned in Section 2.2.4.1, was selected to isolate critical features

on the SCM. To allow communication between containers and the host system, virtual ethernet pairs are used as described in section 2.2.4.1. For the SCM two LXC containers are required:

- **RCoS:** This container contains all required resources for the RCoS communication (see Section 5.3.3). Beside the necessary communication components, also all commands are executed here. This container can communicate with both the inside and the outside of the car. It also has a virtual network connection to other LXC containers on the SCM.
- **Encryption/Decryption:** The whole encryption and decryption functionality of the SCM is isolated in an own container. Only the RCoS container is allowed to have a local communication channel for encrypting/decrypting its data. Because also the encryption keys are stored here, this container only allows local communication on the SCM.

As described in Section 2.2.4.1 a range of Linux security mechanisms can be used in combination with LXC containers. seccomp white-listing is used to only allow system calls which are required. Also resource limitation is part of the security concept using cgroup.

5.2.3 Additional Hardening Measures

Further hardening measures to prevent potential attackers from gaining access to the SCM include:

- **User Management:** Having multiple users for different tasks. Every user is only allowed to access resources which are necessary for him.
- **Cleanup:** Removing all programs which are not required on the system. Especially analysing all automatically started tasks to only run required processes.
- **Secure configuration:** Reviewing the configuration of all programs to close potential security holes.

5.3 Communication via the SCM

To give the car's internal components the ability to communicate with external servers over the internet, an own communication concept was developed for this thesis. Some crucial requirements, the design has to fulfill, were defined as following:

1. *Highest possible isolation of the car's internal control network:* When connecting cars to the internet, the car's internal control network has to be isolated to prevent attackers from getting access to it.

2. *Data Confidentiality*: Data sent or received by the car may contain sensitive data. Only devices which are allowed to read the data should be able to read it.
3. *Data Integrity*: The sent data has to be exactly the same than the received data. Data corruption has to be prevented during the whole communication from the sender to the receiver.
4. *Authenticity*: Within this thesis only communication with well-known servers is allowed. When receiving data, it has to be ensured that this data is really coming from the sender it seems to come from. Furthermore all communication partners have to prove that they really are who they pretend to be.

To fulfill the first point (isolation), devices inside the car's internal network can't access the internet directly. They can only communicate with the SCM. More details can be found in Section 5.3.1. For requirement two to four, authenticated encryption was chosen, as described in Section 5.3.4.

5.3.1 Isolation of the car's internal control network

The worst case in automotive security is an attacker, being able to access the car's internal control network. This shouldn't happen under any circumstances. Therefore no direct communication between the internal network and the internet is allowed. The internal network can only communicate with the SCM, and only the SCM can communicate with the internet. In the other way, the SCM can't directly access the CAN bus, but only communicate with the gateway of the internal car network. This ensures that even after a successful attack of the SCM, an attacker can't access the CAN bus, which is required to control ECUs. Furthermore the gateway of the internal network and the SCM are only connected via a non-routable connection, allowing only direct communication between this two devices. In case of this thesis project a RS232 cable was used.

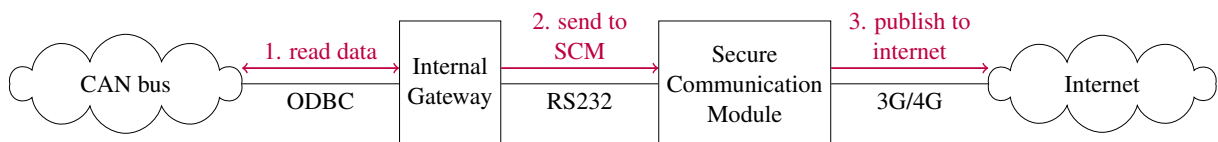


Figure 5.2: Dataflow of sending telemetry data

Figure 5.2 illustrates how the communication flow looks like when sending telemetry data. First the Internal Gateway reads the data from the CAN bus. Afterwards the data are send to the SCM over the RS232 connection. Finally the SCM can publish the data to the internet via 3G or 4G. The communication over the RS232 connection and over 3G/4G are completely independent and use different protocols. Also new packets are formed on the network barrier.

5.3.2 Message Queue Telemetry Transport protocol

Within this thesis project the MQTT protocol (see Section 2.1.1) is used to communicate over the internet. Therefore an MQTT broker is used within the companies network. All communication between the car and the company servers are done via this broker.

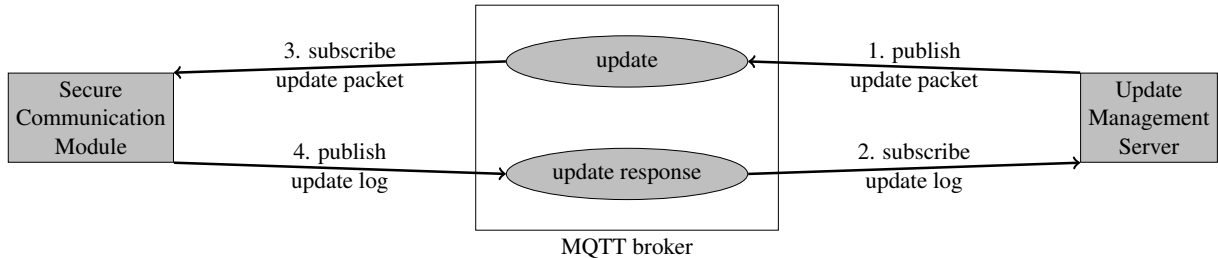


Figure 5.3: Example dataflow of an ECU firmware update

To illustrate how the communication via MQTT works, Figure 5.3 shows the example of performing an ECU firmware update. The communication is done in multiple steps:

1. An software update packet containing the new firmware is generated on an update management server and published to the update topic on the MQTT broker.
2. The update management server subscribes to the update response topic on the MQTT broker, to receive response from the update process.
3. The car regularly subscribes to the update topic of the MQTT broker to check for the existence of a new firmware. If a software update packet is available, the effected ECUs will be flashed with the new firmware. A log file of the update process will be generated.
4. The update log file gets published to the update response topic and processed by the update management server.

5.3.3 Remote-Call-over-Serial protocol

For the communication between the Internal Gateway and the SCM an own protocol, named RCoS, was developed. This protocol allows remote procedure call over a serial connection. It allows to execute predefined commands on a device connected via a serial interface. Before the design process of the protocol, some requirements were identified:

- Lightweight protocol with minimal overhead
- Easy configuration

- High flexibility
- Practicable for big amount of data
- Implementation in Python

5.3.3.1 RCoS components

The RCoS protocol consists of multiple components. First of all at least two devices running RCoS are required. On the one side there is the calling device, on the other side the executing device. Both are connected via an RS232 connection. The calling device sends an RCoS request to the executing device, where it gets executed. Afterwards the executing device sends back the RCoS response. On both devices different components are required. On the calling side, this includes the following:

- **RCoS-Monitor:** This component has to run on the calling device all the time. It opens an IPC socket (see Section 2.2.3.1) and listens for command calls. For every command call an own thread is started. This thread sends the RCoS request over the serial connection to the executing device and waits for a response. Afterwards the response gets send to the IPC socket.
- **RCoS configuration file:** This configuration file contains all RCoS parameters which can be configured on the calling device. This contains for example the serial interface or baudrate as well as the command prefixes.
- **RCoS command scripts:** RCoS commands are called by executing the respective command script. There is an own python script for every RCoS command. This scripts perform an remote call for the particular command using the parameters given to the RCoS command script. The command output is printed to stdout and the script exits with the exit status of the RCoS response.

Required components on the executing side are:

- **RCoS-Listener:** This component has to run on the executing device all the time. It listens for incoming RCoS requests on the serial interface. Every command gets executed in an own thread. After execution an RCoS response containing the output and the exit status gets formed and written to the serial interface.
- **RCoS executables:** For every RCoS command there is an executable which will be called on the respective RCoS command. This can be all kind of executable files or commands.
- **RCoS configuration file:** Similar to the calling device, also the executing device has an config'uration file for its RCoS parameters. Beside information for the serial connection, this file also contains the path to the executables.

5.3.3.2 Representation of a number with variable length

On an RS-232 connection data is written byte by byte. On default there is nothing like a packet of data, but only a stream of bytes. Although the RCoS protocol requires to send packets of data. Therefore an own packet format was defined (see Section 5.3.3.3). When interpreting a byte stream as a stream of packets, the length of the packet has to be known. This is ok for header information with a fix length, but it gets harder with variable length payload. Table 5.1 illustrates the number of bytes required to represent a number. Here it can be seen that all numbers lower than 256 can be represented using 1 byte. For the number 256 already 2 bytes are required. In other protocols, often a fixed number of bytes is used to represent the length of the payload. This either limits the maximum size of the payload in case the number of length bytes is too low, or increases the overhead when having high number of length bytes, but only a short payload. Having the two design requirements of minimal overhead and practicability for big amount of data (see Section 5.3.3), the author decided to use a variable number of length bytes. This is done by adding an additional byte, to represent the number of length bytes required to hold the length of the payload. Therefore for all numbers lower than 256 this additional byte will be set to 1 and the next byte holds the number. For numbers from 256 until 65,535 the additional byte is set to 2 and the following two bytes contain the number.

values	number of bytes
0 - 255	1
256 - 65,535	2
65,536 - 16,777,215	3

Table 5.1: Required number of bytes for given values

5.3.3.3 RCoS packet format

All the fields of an RCoS message are described in Table 5.2. Every message starts with a start byte to indicate that the following bytes will represent the packet. This is required because on the serial connection there is only one data-stream without any separation. Every packet consists of the length of the payload and the payload. The length of the payload is represented in form of a variable length number (see Section 5.3.3.2). RCoS requests and RCoS responses only differ in the payload. In case of requests a three character command prefix identifying the command as well as the parameters are in the payload.

In the response, the payload contains the command output and the exit code. The command output is the output of the executable which run on the executing device. To match a response with the related request, an ID is used, which also uses the variable length number format (see Section 5.3.3.2). This ID is given by the RCoS-Monitor and only used for the communication over the RS-232 connection. For the communication over the IPC sockets inside the calling device no ID is required. This is because a direct connection between the process which sent the call and the respective thread inside the RCoS-Manager is established over the IPC socket.

Field	Length [bytes]	Description
<i>start byte</i>	1	The start byte indicates the start of a new packet. It has the hex value '0xc0'.
<i>ID bytes</i>	1	The number of bytes, which are required to present the ID.
<i>ID</i>	<ID bytes>	A unique ID for every remote call is required for matching RCoS request and the related responses.
<i>length bytes</i>	1	The number of bytes, which are required to present the length of the payload
<i>length</i>	<length bytes>	The length of the payload in bytes. This field can also be empty (length of 0 bytes) in case payload is empty.
<i>command prefix</i>	3	A three character long prefix which uniquely identifies the desired command inside RCoS requests. This field can be also considered as part of the payload.
<i>payload</i>	<length>	Arguments in case of request or exit status and command output in case of response. Payload can also be empty (length of 0 bytes)

Table 5.2: Fields of RCoS message

5.3.3.4 RCoS encryption

The whole RCoS communication is cryptographically protected. Encryption only covers the payload because the id and the length still has to be readable in cleartext. This is acceptable because just from

the id and length of the encrypted payload no conclusions can be made. CMS is used for authenticated encryption. Beside confidentiality, this also ensures integrity and authenticity. On both side, the calling device and the executing device, a certificate for public-key-encryption was generated. Because the encryption is done for a specific destination, only this destination can decrypt the message. For example if the calling device encrypts an RCoS request, it can only be decrypted by the executing device. For more information about the encryption used see Section 2.1.3 and Section 5.3.4.

There is an Encryption-/Decryption-Listener for the encryption and decryption of data. They open an network socket (see Section 2.2.3.2) and wait for enc-/decryption calls. This listeners run in an own LXC container (see Section 5.2.2) to isolate the encryption/decryption and all related files. This is also the reason why network sockets instead of IPC sockets have to be used. Both listeners also have a configuration file to specify the encryption parameters, e.g. the used certificates. To encrypt a text, the cleartext has to be sent to the network socket of the Encryption-Listener, who answers with the encrypted message. The same process is done for decryption.

5.3.3.5 RCoS commands

Table 5.3 illustrates a list of the currently implemented commands including command name, RCoS prefix and description.

Command	Prefix	Description
<i>date</i>	DAT	This executes the Linux date command. It was implemented to test the RCoS communication, having an standard command with a different output every time. Furthermore it can be also used to measure the time which is required for the communication.
<i>mqtt_publish</i>	PUB	Performing MQTT publish based on the given arguments.
<i>mqtt_subscribe</i>	SUB	Performing MQTT subscribe based on the given arguments.

Table 5.3: List of currently implemented RCoS commands

5.3.3.6 Example of an RCoS call

To get a better understanding of the RCoS protocol, an example call gets explained here. To keep it simple, a date call is used. This executes the date command on the executing device. To illustrate the

usage of parameters, a format parameter to only show the year gets passed to the date command. To better understand the data packets, they are presented in cleartext, even though they would be encrypted normally.

1. First the RCoS-Monitor has to be started on the calling device. This opens the IPC sockets and listens for commands. On the executing device, the RCoS-Listener needs to run and listen to its serial interface for incoming commands.
2. To perform the date command, the date command script has to be executed on the calling device. To only get the year out of the date command, the format specifier + "%Y" gets passed as argument.
3. The date command script builds an RCoS request (see Table 5.4) and sends it to the IPC socket. The *length* field is set to 8 because the command prefix needs 3 bytes and the attributes have 5 bytes.
4. The RCoS-Monitor receives the request and opens an own thread. There an ID is assigned and the packet is sent over the serial interface to the executing device. In this example the ID 256 was chosen to have a variable length number. In hexadecimal format the number 256 is 0x0100 and requires two bytes. Table 5.5 shows the packet sent from the RCoS-Monitor to the RCoS-Listener. Because the ID needs two bytes, the *ID bytes* field is set to two. After sending the request, the thread waits for an response.
5. The RCoS-Listener receives the request and opens an own thread, which executes the date command with the format specifier and writes the RCoS response containing the exit code and command output to its serial interface. This response is shown in Table 5.6. The exit code is 0, which means the command was executed successfully. The command output is 2017\n, where \n is a newline character.
6. The RCoS-Monitor receives the response and forwards it to the thread with the related ID. From there the result is written to the IPC socket in form of a RCoS response packet (see Table 5.7).
7. As a last step the command script process prints the command output and exits with the exit code.

start byte	length bytes	length	command prefix	attributes
0xc0	0x01	0x08	DAT	+"%Y"

Table 5.4: RCoS request sent over IPC socket

start byte	ID bytes	ID	length bytes	length	command prefix	attributes
0xc0	0x02	0x0100	0x01	0x08	DAT	+"%Y"

Table 5.5: RCoS request sent over RS-232 connection

start byte	ID bytes	ID	length bytes	length	exit code	command output
0xc0	0x02	0x0100	0x01	0x06	0x00	2017\n

Table 5.6: RCoS response sent over RS-232 connection

start byte	length bytes	length	exit code	command output
0xc0	0x01	0x06	0x00	2017\n

Table 5.7: RCoS response sent over IPC socket

5.3.4 Encryption and authentication

All communication is fully encrypted and also authentication is provided for the whole communication. Therefore data is encrypted at any point of the communication process, both in transit and at rest. Additionally the authenticity of the transmitted data can be checked at any time. Cryptographic protection is done on two layer. First there is an end-to-end encryption and authentication which protects the data from its origin until the final target using the CMS (see Section 2.1.3). This ensures that data is also protected at rest. All devices involved in the communication process only get in touch with encrypted data. This also includes the SCM which never sees transmitted data in cleartext. Furthermore the single communication channels are cryptographically protected. The only exception is the communication over the CAN bus, which is not protected yet. This will be done in a different project. Within this thesis project two communication channels having different security mechanisms were used:

- RCoS communication between the cars internal gateway and the SCM
- MQTT communication between the SCM and the MQTT broker as well as from the MQTT to the backend infrastructure

5.3.4.1 Secure MQTT communication

Within the MQTT protocol no network or application-layer security is provided natively. This is because it was designed to be very lightweight and reduce all unnecessary overhead. But sometimes it is necessary to securely communicate over MQTT. To accomplish such a secure communication channel the ideas of an former project with AVL contribution were used. In [63] a concept for secure MQTT communication, providing confidentiality and authenticity by the usage of TLS (see Section 2.1.2), was

described. Furthermore they also defined two main goals for their architecture [63]:

- **Publisher authorization:** Only authorized clients are allowed to publish data to the MQTT broker.
- **Subscriber authorization:** Only authorized clients are allowed to subscribe to data from the MQTT broker.

These goals are also applicable to the thesis project, because all MQTT clients are known to the MQTT broker, and no other device is allowed to publish or subscribe data. Also data is always published for specific targets and only this targeted device is allowed to subscribe the data. For an authorization mechanism on the MQTT broker, authenticated clients are required. This is done via TLS client authentication (see Section 2.1.2). The MQTT broker retrieves the client identity from the TLS handshake. For all security operations OpenSSL was used (see Section 2.1.2.1). Every SCM needs an own client-certificate which is bound to a specific topic on the broker. All communication from an SCM and the MQTT broker is done via this topic or subtopics. The client-certificates from all SCMs also have to be available on the broker for validation.

5.3.4.2 Example of securely publishing Telemetry data

To better understand the combination of data protection mechanisms, the data flow for publishing telemetry data will be analyzed. Following the steps for publishing telemetry data in a secure manner are listed:

1. Read data from the CAN bus. No data protection for communication over the CAN bus is implemented yet. This is not in the scope of this thesis project.
2. Encapsulate the data in a secure CMS packet. Data gets encrypted and therefore end-to-end data confidentiality and authenticity is provided.
3. Sending a *mqtt_publish* command, containing the protected telemetry data, to the SCM. This is done via the serial connection between the Internal Gateway and the SCM over the RCoS communication protocol. The RCoS packet contains additional CMS protection, so that it can be only read by the SCM.
4. Publish the protected telemetry data to the MQTT broker. This is done over the internet using the MQTT protocol. The whole MQTT communication is TLS protected.
5. Publish the protected telemetry data to the backend server, where it will be processed. This is done over the internal backend infrastructure using the MQTT protocol. The whole MQTT communication is TLS protected.

6. Decrypt and validate the telemetry data. In case validation succeeds the received data can be processed.

Chapter 6

Implementation and Evaluation

After the design phase, a actual prototype of such a SCM was build as described in this chapter. This prototype was then evaluated to fulfill the requirements identified in section 4.2.5.3.

6.1 Implementation of Security Concept

The partner company already implemented an Internal Gateway, which provides a connection to the internal CAN bus, using an ARM Cortex-A9 board. Therefore such a hardware was also used for the SCM. To have full control over the system and having the chance to dive deeply into the operating system or even into the kernel, the use of a Linux based operating system was a basic requirement of the project. As a startpoint the operation system from the Internal Gateway was copied to the SCM. This specially designed Linux Distribution was generated using the Yocto Project¹. This open source project supports generating a custom Linux System for embedded devices [64]. Based on this system the security features described in the design chapter were implemented on the prototype. For all required packages the source code was downloaded and manually build on the SCM. Most packages didn't successfully build on the first try. Therefore research had to be done to figure out and solve the compilation problems. Also many packages have dependencies on other packages which have to be installed first.

6.2 Operational Evaluation

Following the DSR methodology, after the design and implementation also evaluation is an essential part. This ensures that the final artifact fulfils the defined requirements. Being an iterative approach, the implementation and evaluation phases have to be repeated until all requirements are met (see Figure 4.1).

¹<https://www.yoctoproject.org>

The most important criteria the SCM has to fulfil, is functionality. All tasks which were possible to do without the security device also have to work with the newly added module. Using the prototype infrastructure this tasks consists of the two main use cases of OTA software updates and collection of telemetry data. Both use cases were tested within a realistic setup having the SCM attached to a car. Therefore the Internal Gateway was connected to the CAN bus. The SCM was connected to the Internal Gateway using a serial connection and on the other side connected to the internet via a 3G internet stick. Furthermore a laptop with a connection to the backend management server was used.

6.2.1 Over-the-Air update test

The OTA update was tested with the Hybrid Vehicle Control Unit (HVCU) of the car. This ECU is the central control unit within a hybrid car. A new firmware for the HVCU was uploaded to the backend management and also the OTA update process was triggered there. After a view minutes the update was completed successfully and the logfile was available on the backend. To double-check the functionality of the new firmware, a successful test ride with the car completed this first test.

6.2.2 Telemetry data collection test

To test the second use case, first a decision about what data to read was required. To perform the test on a standing vehicle, a counter from the HVCU was selected to show how telemetry data can be collected via the internet connection of the car. From the backend the data collection was triggered and successfully performed by the car. The data was written to the backend and the test was repeated using different parameters for the data collection. All of them worked fine and successfully wrote the collected data into the backend telemetry database.

6.3 Performance evaluation

Beside the functionality of the SCM, also its performance is important. Because the whole traffic has to run over an additional device, the performance suffers. Especially the serial connection slows down the overall communication. Therefore the increase of the overall runtime by the SCM was taken as evaluation criteria for the usability. To make a comparison between the overall runtime with and without the SCM, the exactly same OTA firmware update was performed twice. First by directly connecting the internal gateway to the internet and afterwards using the SCM including all security measures.² For both test rounds a update request including the new firmware of the HVCU were published from the backend

²For the direct connection of the Internal Gateway to the internet the firewall configuration from the SCM was transferred to the Internal Gateway.

to the MQTT broker in advance. The overall update time was defined as the runtime of the update process on the Internal Gateway. This process includes:

1. Subscribe to MQTT broker to check for new updates.
2. Receive update-request, containing the new firmware from the MQTT broker.
3. Decrypt message and perform firmware update.
4. Encrypt and publish logfile to MQTT broker.

To get representative data, the same test was performed three times. Table 6.1 shows the measured times. After the evaluation tests also the logfiles were analyzed. As expected the main bottleneck of the newly developed security design is the serial communication link between the SCM and the internal gateway. Especially with big amount of data the difference in runtime can become substantial.

	without SCM	with SCM	difference
test 1	4min 32sec	6min 12sec	1min 40sec
test 2	4min 30sec	6min 16sec	1min 46sec
test 3	4min 33sec	6min 9sec	1min 36sec

Table 6.1: Overall time of update process

6.4 Security Performance

Additionally to the operational evaluation, within a security related device also the security performance is a crucial evaluation aspect. For such a device it's not enough to just work. Anyhow it is hard to measure how secure a product is. Within this thesis project the evaluation of the security performance was done by performing a penetration test. To provide a independent examination a external company was executing the penetration test. This company is specialized in security audits. For the test setup the SCM was connected to a router with internet access via a 3G data stick. Table 6.2 shows all activities which were performed during the penetration test. Additionally to this activities the security concept was presented and discussed. After the penetration test the security company stated the SCM to be secure against external attacks.

Activity	Result
Performing a port scan to identify all open ports	There are no open ports for incoming connections
Trying to read from an active session	All traffic is encrypted and therefore can't be read
Trying to hijack a active session	Hijacking of an active session was not possible

Table 6.2: Activities of the Penetration Test

Chapter 7

Discussion

Within this chapter the results of Chapter 6 will be discussed and also mapped to the research questions (see Section 1.2). The main focus of this chapter is the usability and performance of the SCM and how it can be used to make connected cars more secure.

7.1 Usability and Practicability

The first research question (see Section 1.2) is about the usability of the proposed solution to use a hardware-based SCM to protect connected cars from external attacks. This idea was already used in the EVITA project [17]. One main difference between the EVITA project and the SCM approach is the number of required security devices. Within this thesis the basic concept of having multiple specialized security modules was replaced by using only one general security module which is responsible for the whole communication between the car and the internet. Having only one device it is easier to equip existing internet connected cars with a secure communication method. Furthermore another important difference is the non-routable connection between the internal gateway and the SCM. Within this thesis even an own communication protocol (see Section 5.3.3) was developed to ensure optimal isolation of the internal vehicular network. Within Section 6.2 the functionality of the SCM was tested in contrast to the already existing vehicular communication infrastructure. After installing the new everything worked exactly the same and all tested use cases worked successfully, proving the usability of the newly developed concept.

7.2 Performance

Beside the usability, also the performance of the SCM is an essential aspect to be covered. As described in Chapter 6, both functional and security performance are covered in this thesis project. The former

considers that the measures to provide a secure communication channel have to be done in addition to the normal functionality. Therefore the performance will differ compared to a setup without any security considerations. Since the security mechanisms are running on an own hardware with adequate resources, the main performance difference concerns the increased time for communication to the internet. This was also seen in the evaluation tests described in Section 6.3, where the serial connection between the SCM and the internal gateway is identified as the main bottleneck of the secure communication channel. Table 6.1 shows the runtime increasing by an average of 1 minute and 40 seconds using the newly developed security concept. This is acceptable for a prototype with the focus of demonstrating the underlying security concept. When putting more focus on time-critical performance further investigation would be necessary to find another non-routable communication channel replacing the RS-232 connection.

An assessment of the security performance was done using a penetration test (see Section 6.4). To ensure objectivity the pentest was performed by an external company, specialized in security audits. The main goal was to test the defense capabilities against an external attacker via the internet connection of the car. The external company rated the combination of conceptual and technical measures to provide adequate security for an internet connected car.

7.3 Summary

Ensuring a holistic view of the whole thesis project, the research questions are answered considering both usability and performance aspects. Both questions are connected while the first one is about the applicability, and the second about adequate security measures including their performance (see Section 1.2). They define the application area of the SCM within an bigger overall security infrastructure. This means that the SCM also has to seamlessly work together with other components. During the evaluation phase it was possible to prove that the developed prototype successfully fulfills the requirements given within this project. Therefore the combined answer for both security questions is the following: Yes, it is possible to design and implement an adequate performing hardware-based SCM within an bigger overall vehicular security infrastructure using the security concept which is described in Chapter 5.

Chapter 8

Conclusion and Future Work

Within a highly connected world, this thesis project contributes to a more secure way of communication. Especially in the automotive sector security had a minor role during the development of internet connected vehicles. Over the past few years this changed a lot and also this thesis proposes a secure communication method for cars.

The fundamental idea of using an own hardware-based communication module was chosen to provide optimal separation between the internal car network and the cellular network. Because the SCM is the only reachable device from outside the car, the internal vehicular network can't be directly accessed. Even if an attacker finds a way to access the SCM, he can't do any harm to the car and even more important, to the passengers. Therefore no external attack should be possible. Within the AVL LIST GmbH a hardware-based SCM prototype was designed and implemented. By combining different Linux security measures a security concept was developed and described in Chapter 5. The main focus was to provide defensive capabilities, but also the SCM itself has to be hardened, since an insecure security device can't provide any security. Afterwards the security concept was implemented on an ARM Cortex-A9 board. As a last step a successful evaluation of the functionality and performance showed that this prototype can be used as a step towards more secure internet connected vehicles. Especially the isolation of the internal network by using a non-routable connection between the internal gateway and the SCM was a major finding of this thesis. Therefore the internal gateway can communicate with the SCM, but no communication between the internet and the internal vehicular network is possible. In combination with allowing only outgoing traffic, the SCM can only reply to requests coming from the internal gateway. This makes external attacks very difficult.

8.1 Future work

Being a prototype, the developed SCM can still be further improved. One important aspect which was not covered in this thesis project is the physical security of the module. This means that an potential attacker shouldn't be able to manipulate the SCM even with physical access to it. On the software side the RCoS protocol has to be reviewed because it was just developed for prototype purposes. Maybe there is a better alternative providing similar functionality but being better developed. In a later stage of development an SCM will also require some kind of update functionality to update its software. This will require an own update concept preventing attackers from installing malicious updates. Furthermore the disk of the SCM could be fully encrypted and read only so that no change can be done on the system. During runtime all data can be stored in memory. Maybe a small permanent storage area can also be writable to store log files, which have to be encrypted as well. Having an encrypted and read-only system disk, the system can't be easily manipulated from an attacker. Additionally a simple reboot of the device can bring back a working system in case it was somehow infected. To support cryptographic operations, an own security chip is planned to become part of the SCM. This chip has all required cryptographic functions hardware-implemented and can perform them in a secure and fast manner. The chip can also save security keys which can't be read but only used for the cryptographic operations of the chip. To increase communication performance the serial connection to the internal gateway has to be replaced by a faster non-routable connection. For developing an usable product out of the prototype further software and hardware optimization has to be done to get secure and performant hardware-based SCM which can be produced on a huge scale for low costs.

Acronyms

(D)DoS (Distributed) Denial-of-Service. 20, 31, 32

CAN Controller Area Network. 6, 35, 41, 43, 45

cgroup control group. 18, 34

CMS Cryptographic Message Syntax. 11, 39, 41, 43

DAC Discretionary access control. 15, 16

DSR Design Science Research. vii, 24, 25, 26, 45

ECU Electronic Control Unit. vii, 1, 6, 22, 27, 28, 35, 36, 46

EU European Union. 1

EVITA E-Safety Vehicle Intrusion Protected Applications. 22, 23, 49

gid group identifier. 19, 20

HSM Hardware Security Module. 22

HVCU Hybrid Vehicle Control Unit. 46

IDPS Intrusion Detection and Prevention Systems. 21, 32

IDS Intrusion Detection System. 21

IETF Internet Engineering Taskforce. 11

IoT Internet of Things. i, 5

IP Internet Protocol. 17, 19

IPC Inter Process Communication. 17, 19, 37, 38, 40, 41

IPS Intrusion Prevention System. 21

LSM Linux Security Modules. 15

LXC Linux Containers. 18, 19, 20, 33, 34, 40

M2M Machine-to-Machine. 8

MAC Message Authentication Code. 9

MAC Mandatory access control. 15, 16, 33

MQTT Message Queue Telemetry Transport. 8, 9, 35, 36, 40, 42, 43, 46, 47

NAT Network Address Translation. 12

NSA National Security Agency. 15

OBD On-board diagnostics. 3

OTA Over-the-Air. i, 2, 22, 23, 27, 28, 29, 45, 46

PAYD Pay-As-You-Drive. 3, 6

PID Process identifier. 19

RCoS Remote-Call-over-Serial. viii, 34, 36, 37, 38, 39, 40, 41, 42, 43, 51

RFC Request for Comments. 11

SCM Secure Communication Module. i, vii, 4, 5, 6, 8, 11, 24, 26, 27, 28, 29, 31, 32, 33, 34, 35, 36, 41, 42, 43, 45, 46, 47, 49, 50, 51

seccomp secure computing mode. 19, 34

SELinux Security-Enhanced Linux. 15, 16, 33

SSL Secure Sockets Layer. 9, 10

TCP Transmission Control Protocol. 8, 17

TCU Telematic Control Unit. 3

TLS Transport Layer Security. vii, 9, 10, 42, 43

UDP User Datagram protocol. 17

uid user identifier. 19, 20

UTS UNIX Timesharing System. 19

Bibliography

- [1] M. Broy, I. H. Kruger, A. Pretschner, and C. Salzmann, “Engineering automotive software,” *PROCEEDINGS-IEEE*, vol. 95, no. 2, p. 356, 2007.
- [2] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, “Experimental security analysis of a modern automobile,” in *2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 447–462.
- [3] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, “Comprehensive Experimental Analyses of Automotive Attack Surfaces.” in *USENIX Security Symposium*. San Francisco, 2011.
- [4] F. Sagstetter, M. Lukasiewicz, S. Steinhorst, M. Wolf, A. Bouard, W. R. Harris, S. Jha, T. Peyrin, A. Poschmann, and S. Chakraborty, “Security challenges in automotive hardware/software architecture design,” in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 458–463.
- [5] O. Iparraguirre and A. Brazalez, “Communication Technologies for Vehicles: eCall,” in *International Workshop on Communication Technologies for Vehicles*. Springer, 2016, pp. 103–110.
- [6] R. Oorni and A. Goulart, “In-Vehicle Emergency Call Services: eCall and Beyond,” *IEEE Communications Magazine*, vol. 55, no. 1, pp. 159–165, 2017.
- [7] European Parliament and Council of European Union, “Regulation (EU) 2015/758 of the European Parliament and of the Council of 29 April 2015,” <http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32015R0758>, 2015, [Online; last accessed 2018-05-19].
- [8] C. Miller and C. Valasek, “Remote exploitation of an unaltered passenger vehicle,” *Black Hat USA*, vol. 2015, 2015.
- [9] A. Greenberg, “Hackers remotely kill a jeep on the highway - with me in it,” <https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/>, 2015, [Online; last accessed 2018-05-19].

- [10] —, “After jeep hack, chrysler recalls 1.4 m vehicles for bug fix,” <https://www.wired.com/2015/07/jeep-hack-chrysler-recalls-1-4m-vehicles-bug-fix/>, 2015, [Online; last accessed 2018-05-19].
- [11] A. Brisbourne, “Tesla’s Over-the-Air Fix: Best Example Yet of the Internet of Things?” <https://www.wired.com/insights/2014/02/teslas-air-fix-best-example-yet-internet-things/>, 2014, [Online; last accessed 2018-05-19].
- [12] A. Greenberg, “Tesla responds to chinese hack with a major security upgrade,” <https://www.wired.com/2016/09/tesla-responds-chinese-hack-major-security-upgrade/>, 2016, [Online; last accessed 2018-05-19].
- [13] T. Zhang, H. Antunes, and S. Aggarwal, “Defending connected vehicles against malware: Challenges and a solution framework,” *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 10–21, 2014.
- [14] D. Yoon, J. Choi, H. Kim, and J. Kim, “Future automotive insurance system based on telematics technology,” in *Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on*, vol. 1. IEEE, 2008, pp. 679–681.
- [15] C. Troncoso, G. Danezis, E. Kosta, J. Balasch, and B. Preneel, “PriPAYD: Privacy-friendly pay-as-you-drive insurance,” *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 5, pp. 742–755, 2011.
- [16] I. Foster, A. Prudhomme, K. Koscher, and S. Savage, “Fast and Vulnerable: A Story of Telematic Failures,” in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*. Washington, D.C.: USENIX Association, 2015.
- [17] O. Henniger, A. Ruddle, H. Seudié, B. Weyl, M. Wolf, and T. Wollinger, “Securing vehicular on-board it systems: The evita project,” in *VDI/VW Automotive Security Conference*, 2009.
- [18] D. Brown, “Responsibility for Vehicle Security and Driver Privacy in the Age of the Connected Car,” <http://www.veracode.com/sites/default/files/Resources/Whitepapers/idc-veracode-connected-car-research-whitepaper.pdf>, 2016, [Online; last accessed 2018-05-20].
- [19] S. Derikx, M. de Reuver, and M. Kroesen, “Can privacy concerns for insurance of connected cars be compensated?” *Electronic Markets*, vol. 26, no. 1, pp. 73–81, 2016.
- [20] ISO, “Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1,” International Organization for Standardization, Standard ISO 20922:2016, June 2016.

- [21] I. Heđi, I. Špeh, and A. Šarabok, "IoT network protocols comparison for the purpose of IoT constrained networks," in *2017 40th International Convention of Information and Communication Technology, Electronics and Microelectronics (PIPRO)*, May 2017, pp. 501–505.
- [22] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A survey on application layer protocols for the internet of things," *Transaction on IoT and Cloud Computing*, vol. 3, no. 1, pp. 11–17, 2015.
- [23] T. Dierks and R. E., "The Transport Layer Security (TLS) Protocol Version 1.2," Internet Request for Comments, RFC Editor, RFC 5246, August 2008, [last accessed 2018-05-20]. [Online]. Available: <https://tools.ietf.org/html/rfc5246>
- [24] M. Khalil-Hani, V. P. Nambiar, and M. N. Marsono, "Hardware Acceleration of OpenSSL Cryptographic Functions for High-Performance Internet Security," in *2010 International Conference on Intelligent Systems, Modelling and Simulation*, January 2010, pp. 374–379.
- [25] J. Viega, M. Messier, and P. Chandra, *Network security with openssl: cryptography for secure communications*. Sebastopol, CA, USA: "O'Reilly Media, Inc.", 2002.
- [26] R. Housley, "Cryptographic Message Syntax (CMS)," Internet Requests for Comments, RFC Editor, RFC 5652, September 2009, [last accessed 2018-05-20]. [Online]. Available: <https://tools.ietf.org/html/rfc5652>
- [27] "netfilter/iptables project homepage - The netfilter.org project," <https://www.netfilter.org/>, [Online; last accessed 2018-05-20].
- [28] R. Spennberg, *Linux-Firewalls mit iptables & Co: Sicherheit mit Kernel 2.4 und 2.6 für Linux-Server und-netzwerke*. Pearson Deutschland GmbH, 2006, vol. 2.
- [29] DigitalOcean, "A Deep Dive into Iptables and Netfilter Architecture," <https://www.digitalocean.com/community/tutorials/a-deep-dive-into-iptables-and-netfilter-architecture>, [Online; last accessed 2018-05-19].
- [30] X. Zhao and Y. Ma, "Linux based NAT-PT gateway implementation," in *Info-tech and Info-net, 2001. Proceedings. ICII 2001-Beijing. 2001 International Conferences on*, vol. 5. IEEE, 2001, pp. 258–263.
- [31] "Structure of Iptables," <http://www.iptables.info/en/structure-of-iptables.html>, [Online; last accessed 2017-10-09].

- [32] N. B. Youssef and A. Bouhoula, "Dealing with stateful firewall checking," in *International Conference on Digital Information and Communication Technology and Its Applications*. Springer, 2011, pp. 493–507.
- [33] P. Ayuso, "Netfilter's connection tracking system," *LOGIN: The USENIX magazine*, vol. 31, no. 3, 2006.
- [34] R. Rosen, "Netfilter," in *Linux Kernel Networking*. Springer, 2014, pp. 247–278.
- [35] C. Wright, C. Cowan, J. Morris, S. Smalley, and G. Kroah-Hartman, "Linux security module framework," in *Ottawa Linux Symposium*, vol. 8032, 2002, pp. 604–617.
- [36] F. Mayer, D. Caplan, and K. MacMillan, *SELinux by example: using security enhanced Linux*. Pearson Education, 2006.
- [37] M. J. Donahoo and K. L. Calvert, *TCP/IP sockets in C: Practical Guide for Programmers*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.
- [38] Computer Hope, "What is Unix Domain Socket?" <https://www.computerhope.com/jargon/u/unix-domain-socket.htm>, April 2017, [Online; last accessed 2018-05-20].
- [39] Linux Programmer's Manual, "unix - sockets for local interprocess communication," <http://man7.org/linux/man-pages/man7/unix.7.html>, April 2018, [Online; last accessed 2018-05-20].
- [40] J. P. Walters, V. Chaudhary, M. Cha, S. G. Jr., and S. Gallo, "A Comparison of Virtualization Technologies for HPC," in *22nd International Conference on Advanced Information Networking and Applications (aina 2008)*, March 2008, pp. 861–868.
- [41] D. Beserra, E. D. Moreno, P. T. Endo, J. Barreto, D. Sadok, and S. Fernandes, "Performance Analysis of LXC for HPC Environments," in *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*, July 2015, pp. 358–363.
- [42] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs Containerization to Support PaaS," in *2014 IEEE International Conference on Cloud Engineering*, March 2014, pp. 610–614.
- [43] Linux Programmer's Manual, "cgroups - Linux control groups," <http://man7.org/linux/man-pages/man7/cgroups.7.html>, February 2018, [Online; last accessed 2018-05-20].
- [44] P. Menage, "CGROUPS," <https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt>, [Online; last accessed 2018-05-20].

- [45] M. Kerrisk, “Namespaces in operation, part 1: namespaces overview,” <https://lwn.net/Articles/531114/>, January 2013, [Online; last accessed 2018-05-20].
- [46] Linux Programmer’s Manual, “namespaces - overview of Linux namespaces,” <http://man7.org/linux/man-pages/man7/namespaces.7.html>, May 2018, [Online; last accessed 2018-05-20].
- [47] “SECure COMPuting with filters,” https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt, [Online, last accessed 2018-05-20].
- [48] D. Lezcano, “Linux Containers - LXC - Manpages - lxc.container.conf.5,” <https://linuxcontainers.org/lxc/manpages/man5/lxc.container.conf.5.html>, May 2018, [Online, last accessed 2018-05-20].
- [49] R. Priedhorsky and T. Randles, “Charliecloud: Unprivileged Containers for User-defined Software Stacks in HPC,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’17. New York, NY, USA: ACM, 2017, pp. 36:1–36:10, [last accessed 2018-05-20]. [Online]. Available: <http://doi.acm.org/10.1145/3126908.3126925>
- [50] Linux Programmer’s Manual, “user_namespaces - overview of Linux user namespaces,” http://man7.org/linux/man-pages/man7/user_namespaces.7.html, February 2018, [Online; last accessed 2018-05-20].
- [51] J. Mirkovic and P. Reiher, “A taxonomy of DDoS attack and DDoS defense mechanisms,” *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.
- [52] C. Douligeris and A. Mitrokotsa, “DDoS attacks and defense mechanisms: classification and state-of-the-art,” *Computer Networks*, vol. 44, no. 5, pp. 643–666, 2004.
- [53] K. Scarfone and P. Mell, “Guide to intrusion detection and prevention systems (idps),” *NIST special publication*, vol. 800, no. 2007, 2007.
- [54] A. B. Palekar and S. S. Dhande, “Complete Study Of Intrusion Detection System,” *International Journal of Innovative Research and Advanced Studies (IJIRAS)*, vol. 4, no. 2, 2017.
- [55] M. D. Pesé, K. Schmidt, and H. Zweck, “Hardware/Software Co-Design of an Automotive Embedded Firewall,” SAE Technical Paper, Tech. Rep., 2017.
- [56] T. Hoppe, S. Kiltz, and J. Dittmann, “Applying intrusion detection to automotive it-early insights and remaining challenges,” *Journal of Information Assurance and Security (JIAS)*, vol. 4, no. 6, pp. 226–235, 2009.

- [57] H. M. Song, H. R. Kim, and H. K. Kim, "Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network," in *2016 International Conference on Information Networking (ICOIN)*. IEEE, 2016, pp. 63–68.
- [58] M. Wolf and T. Gendrullis, "Design, implementation, and evaluation of a vehicular hardware security module," in *International Conference on Information Security and Cryptology*. Springer, 2011, pp. 302–318.
- [59] M. S. Idrees, H. Schweppe, Y. Roudier, M. Wolf, D. Scheuermann, and O. Henniger, "Secure automotive on-board protocols: a case of over-the-air firmware updates," in *International Workshop on Communication Technologies for Vehicles*. Springer, 2011, pp. 224–238.
- [60] R. Spaan, "Secure updates in automotive systems," Master's thesis, Radboud University, The Netherlands, 2016.
- [61] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS quarterly*, vol. 28, no. 1, pp. 75–105, 2004.
- [62] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007.
- [63] C. Lesjak, D. Hein, M. Hofmann, M. Maritsch, A. Aldrian, P. Priller, T. Ebner, T. Ruprechter, and G. Pregartner, "Securing smart maintenance services: Hardware-security and TLS for MQTT," in *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, July 2015, pp. 1243–1250.
- [64] O. Salvador and D. Angolini, *Embedded Linux Development with Yocto Project*. Packt Publishing Ltd, 2014.