Thesis for The Degree of Licentiate of Engineering

Towards a Secure and Resilient Vehicle Design: Methodologies, Principles and Guidelines

Kim Strandberg

Division of Computer and Network Systems
Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2022

Towards a Secure and Resilient Vehicle Design: Methodologies, Principles and Guidelines

Kim Strandberg

ISSN 1652-876X
Department of Computer Science & Engineering
Division of Computer and Network Systems
Chalmers University of Technology
Gothenburg, Sweden

Author e-mail: kim.strandberg@chalmers.se

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden, 2022.

Towards a Secure and Resilient Vehicle Design: Methodologies, Principles and Guidelines

Kim Strandberg
Department of Computer Science and Engineering, Chalmers University of Technology

Abstract

The advent of autonomous and connected vehicles has brought new cyber security challenges to the automotive industry. It requires vehicles to be designed to remain dependable in the occurrence of cyber-attacks. A modern vehicle can contain over 150 computers, over 100 million lines of code, and various connection interfaces such as USB ports, WiFi, Bluetooth, and 4G/5G. The continuous technological advancements within the automotive industry allow safety enhancements due to increased control of, e.g., brakes, steering, and the engine. Although the technology is beneficial, its complexity has the side-effect to give rise to a multitude of vulnerabilities that might leverage the potential for cyber-attacks. Consequently, there is an increase in regulations that demand compliance with vehicle cyber security and resilience requirements that state vehicles should be designed to be resilient to cyber-attacks with the capability to detect and appropriately respond to these attacks. Moreover, increasing requirements for automotive digital forensic capabilities are beginning to emerge. Failures in automated driving functions can be caused by hardware and software failures as well as cyber security issues. It is imperative to investigate the cause of these failures. However, there is currently no clear guidance on how to comply with these regulations from a technical perspective.

In this thesis, we propose a methodology to predict and mitigate vulnerabilities in vehicles using a systematic approach for security analysis; a methodology further used to develop a framework ensuring a resilient and secure vehicle design concerning a multitude of analyzed vehicle cyber-attacks. Moreover, we review and analyze scientific literature on resilience techniques, fault tolerance, and dependability for attack detection, mitigation, recovery, and resilience endurance. These techniques are then further incorporated into the above-mentioned framework. Finally, to meet requirements to hastily and securely patch the increasing number of bugs in vehicle software, we propose a versatile framework for vehicle software updates.

Keywords: Vehicle Security, Vehicle Resilience, Vehicle Attacks, Vehicle Software Updates

Acknowledgment

First of all, I would like to thank my family, especially my wife Lisa and my four kids, for their love, understanding, and support. Special thanks go to our dog Tove for keeping me company and warming my feet all late nights while working in front of the computer.

I would like to thank my supervisors, Tomas Olovsson, Nasser Nowdehi, Magnus Almgren, and my examiner Per Larsson-Edefors, for continuous and constructive feedback and all co-authors for fruitful discussions and feedback. I would like to thank all industrial partners involved in the CyReV project, my former and current group manager and product manager at Volvo Cars, Ulf Edvardsson, Karolina Hill, and Hans Alminger, respectively, and all my colleagues at Volvo Cars and Chalmers for their support and encouragement during my studies.

I would also like to thank Volvo Cars and VINNOVA, the Swedish Governmental Agency for Innovation Systems, for funding my research within the CyReV project (2019-03071).

Kim Strandberg
Gothenburg, January 2022

List of Publications

Appended publications

This thesis is based on the following publications:

[A] **K. Strandberg**, T. Olovsson, E. Jonsson, "Securing the Connected Car: A Security-Enhancement Methodology" *IEEE Vehicular Technology Magazine, 2018.*

[B] T. Rosenstatter, **K. Strandberg**, R. Jolak, R. Scandariato, T. Olovsson "REMIND: A Framework for the Resilient Design of Automotive Systems" *IEEE Secure Development, 2020.*

[C] **K. Strandberg**, T. Rosenstatter, R. Jolak, N. Nowdehi, T. Olovsson "Resilient Shield: Reinforcing the Resilience of Vehicles Against Security Threats" *IEEE 93rd Vehicular Technology Conference, 2021.*

[D] **K. Strandberg**, D.K. Oka, T. Olovsson "UniSUF: a unified software update framework for vehicles utilizing isolation techniques and trusted execution environments" *19th escar Europe : The World's Leading Automotive Cyber Security Conference, 2021.*

Contents

Chapter 1

Introduction

1.1 The Changing Automotive Landscape

Figure 1.1 shows the Volvo ÖV4 (Open Carriage), referred to as Jakob from the name day the model was ready on 25 July 1926. It was the first car built by Volvo.

Figure 1.1: The Volvo car model referred to as Jakob [1]

Today, a modern vehicle has moved from merely a transportation vessel to become a *computer on wheels*, a complex *Cyber-Physical System (CPS)* containing over 150 computers and over 100M lines of software code controlling various functionalities such as safety-critical steering, brake, and engine control.

A simplification of the principles of a modern vehicle architecture is shown in Figure 1.2. Vehicle electronics can be broken down into four main categories: *internal and external communication*, *hardware*, *software*, and *data storage* [2]. The first category consists of communication busses, such as CAN, FlexRay, MOST, and LIN. These are used in different network segments in correlation to their specific characteristics. For example, FlexRay is commonly used for safety-critical systems, since it is faster and more reliable than, e.g. CAN,

and MOST is specifically designed for media-oriented data. Each bus has a main gateway that translates and relays data between the different segments. Additionally, a vehicle has multiple connection points and communication interfaces, such as USB ports, WiFi, Bluetooth, and 4G/5G.

The second category consists of *Electronic Control Modules (ECUs)*, *sensors*, and *actuators*. The mechanical linkage is being replaced by *Drive-by-Wire (DbW)* systems that use these sensors, actuators, and ECUs to control safety-critical functions. Sensors give information about, e.g., speed, temperature, distance, and identification of obstacles such as pedestrians and animals. Sensors consists of, e.g., laser and ultrasonic devices, and cameras. The actuators turn input from these sensors (via ECUs) into actions, such as braking, steering, and engine control.

The third category includes *software* installed or running in ECUs, as well as software update systems, e.g., over-the-air or workshops updates. Finally, the fourth category consists of various data, such as logs for forensics, fault codes, reports from software updates, previously executed diagnostics, and cryptographic keys.

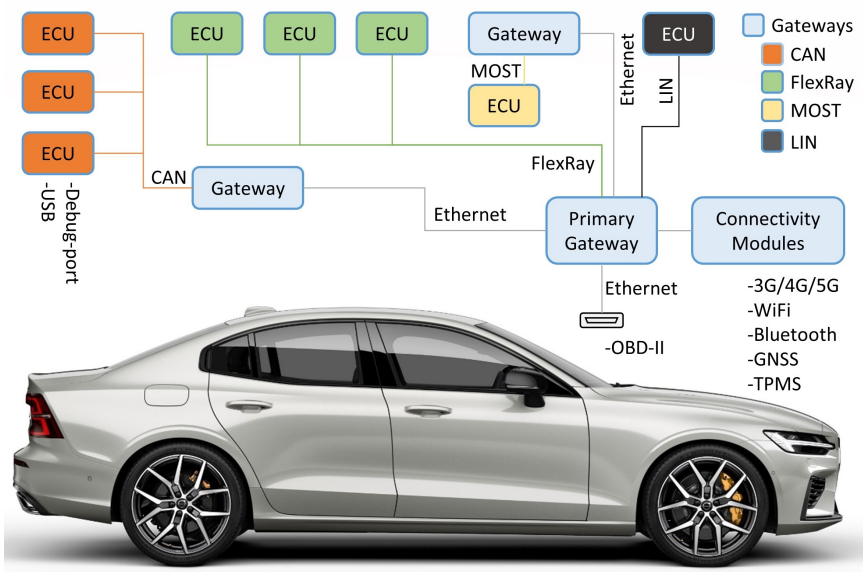

Figure 1.2: An example of vehicle architecture (adapted from [1])

The development of *Cooperative Intelligent Transport Systems (C-ITS)* has enabled *vehicle-to-everything (V2X) communication*, such as communication to other vehicles, cloud, infrastructure as well as other road users (e.g., pedestrians and cyclists). As shown in Figure 1.3, vehicles will become part of a much larger system (e.g., smart cities/infrastructure), where a fleet of connected vehicles will share various information, such as the location for approaching vehicles and traffic conditions (e.g., traffic jam and accidents). In addition to physical objects such as traffic lights and road signs, virtual entities (e.g., *Road Side Units (RSUs)* and virtual traffic lights) can be used to supply supplementary information. In this way, e.g., travel routes and vehicle speed can be better adapted for various situations.

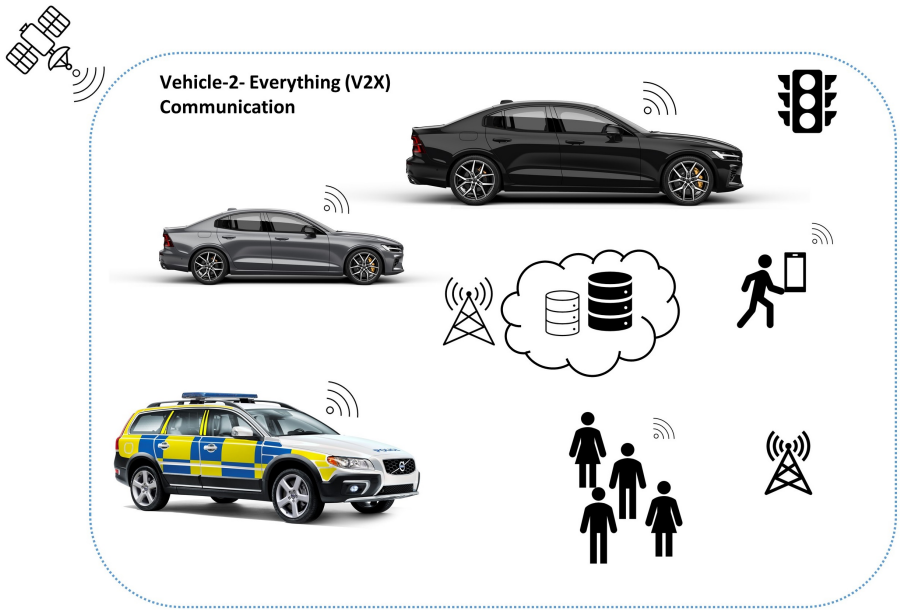

Figure 1.3: An example of V2X communication (adapted from [1])

1.2 Challenges and motivation

We are moving towards a virtual world that requires various interactions with physical entities. These advancements give rise to great opportunities but also many security concerns. For example, none of the aforementioned bus technologies (cf. Chapter 1.1) have been developed with cyber security in mind; instead, the emphasis concerning vehicle design have been on comfort, safety, and reliability. For instance, CAN nodes use broadcasting for communication with no encryption or authentication of messages, which gives the potential for malicious devices to record, manipulate and spoof messages (i.e., masquerading as a trusted entity). Moreover, CAN is priority-based, thus, vulnerable to Denial of Service (DoS) attacks. Additionally, many of the ECUs used in modern vehicles have limited performance and cannot use security techniques such as cryptographic operations. Limited network bandwidth in these protocols is another issue. Thus, the main challenges are that previous designs have not been developed with security in mind, and adding security as an afterthought is cumbersome.

The increased complexity in modern vehicles increases the risk for vulnerabilities. Adding the increased connectivity, the attack surface is further broadened with a higher potential to exploit these vulnerabilities. The associated cost for more advanced hardware in the new design is another issue. Thus, securing vehicles is a complex and challenging task.

Nevertheless, as reflected in recent regulations, the trend has begun to shift, and security and resilience requirements within the automotive cyber-security are continuously being established [3–6]. Along with these requirements, and as shown in Figure 1.4, there is a tendency within the automotive industry to

move towards a more centralized architecture using mainly Ethernet communication where high-performance core computers have the potential to virtualize hardware. Consequently, the potential for better use of security techniques emerges [2, 7].

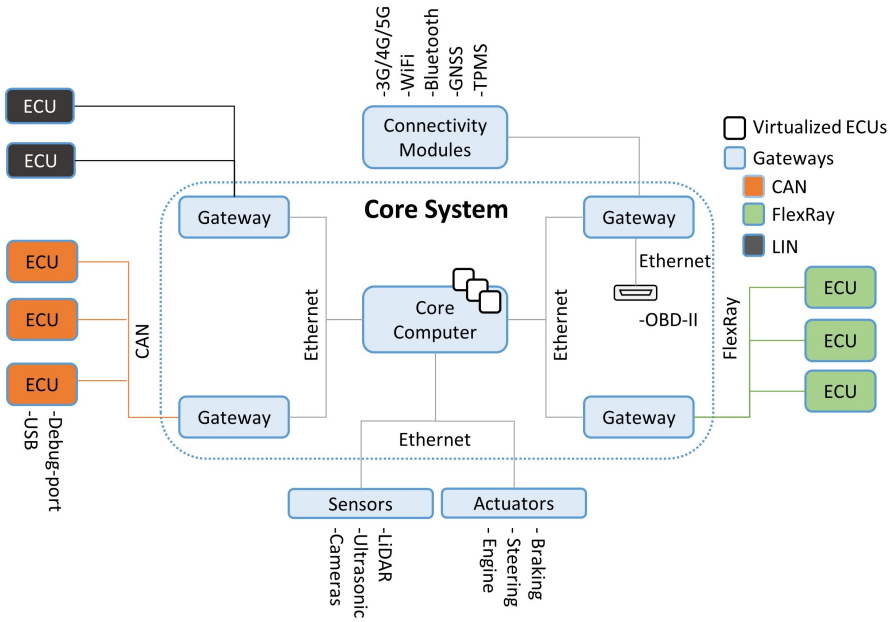

Figure 1.4: An example of a core vehicle architecture

Technological advancements along with the increased complexity within the automotive industry call for novel approaches to find, analyze and mitigate vulnerabilities. Although it is not realistic to find and mitigate all vulnerabilities, strategies towards a baseline for protection against security threats and attacks are imperative. However, the unfortunate truth is that cyber-attacks will occasionally succeed, and cyber-security must be extended with a resilience approach to ensure a safe operation even in the occurrence of a successful breach.

According to NIST, cyber-security can be defined as *the ability to protect or defend the use of cyberspace from cyber attacks* [8]. Resilience can be defined as the *property of a system with the ability to maintain its intended operation in a dependable and secure way, possibly with degraded functionality, in the presence of faults and attacks* [7]. Therefore, cyber-security should detect and prevent cyber-attacks, while cyber-resilience additionally should respond and recover after attacks have occurred. Consequently, the core research question and main emphasis in this thesis are:

- How can we ensure a secure and resilient vehicle design?

In the remainder of this thesis, we present an overview of the papers, research questions, approach, results, conclusions, future work, and finally, we append the publications.

1.3 Papers Overview

In this section we provide a summary over the publications included in this thesis, including research questions and contributions. Figure 1.5 shows an overview of the included publications as well as planned work.

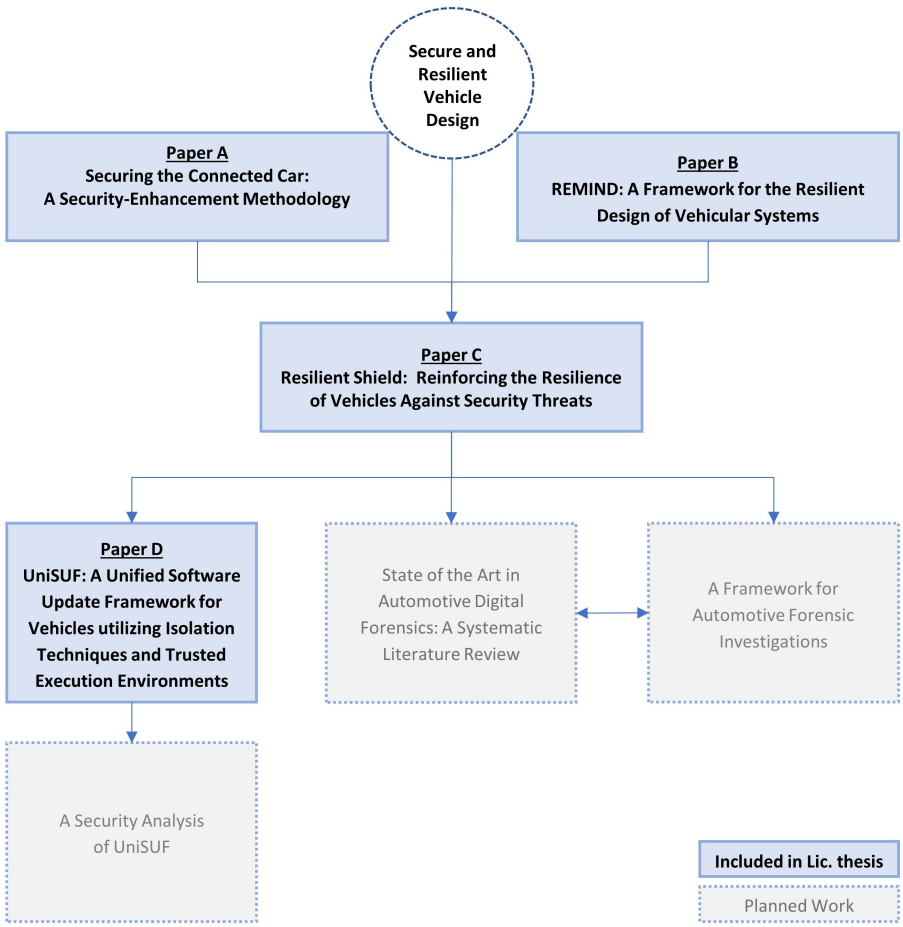

Figure 1.5: Research publications and planned work

Paper A. Securing the Connected Car: A Security-Enhancement Methodology [9]

K. Strandberg. T. Olovsson, E. Jonsson

There is a need for approaches specifically adapted for the automotive domain concerning cyber-security analysis, such as threat modeling, risk assessment, mitigation, and security testing. However, previous approaches have been connected to areas other than automotive or limited to certain phases in a vehicle life. Thus, in this paper we investigate:

- What existing methods and models within the cyber-security area are applicable for the automotive industry?
- Can these methods and models be adapted and used for vehicles?

The result is a security-enhancing methodology named the *Start Predict Mitigate Test (SPMT)* with the purpose to identify and mitigate vulnerabilities in vehicles during their entire life cycle, from development to market use. The methodology was developed by first studying various methods and models in different areas and selecting and adapting relevant parts applicable to the automotive industry. Second, a theoretical and empirical study of attacks was conducted to validate that vulnerabilities related to these attacks can be detected. The *SPMT methodology* was then developed based on the conclusions made from these two steps. The core contributions and benefits are:

- An innovative, comprehensive, and systematic methodology, where existing models and methods have been studied, and components have been connected and adapted by incorporating new ideas suitable for vehicles.
- High coverage for mitigation is given against high priority threats on an operational, safety, privacy, and financial aspect by following the proposed methodology.
- Security as an entirety is considered by performing the analysis both on a device and system-level aligned with vehicles development, production, and market phases.
- The proposed methodology is time and cost-effective, and adaptable to meet different situations by adapting tools and attacks depending on the evaluated asset. Moreover, it is possible to start in different methodology phases to align with other development processes within the automotive industry.

In summary, a security and safety-enhancing methodology is proposed and achieved through a comprehensive and systematic approach to security analysis, specifically adapted for vehicles. This methodology covers security analysis for the entire vehicle life-cycle, which is essential for the automotive industry's efforts to improve security and safety.

Statement of contributions. This is a joint work with my supervisor Tomas Olovsson and Erland Jonsson. I contributed as the lead for the idea, performed the development of the methodology and the writing of the manuscript.

Appeared in: *IEEE Vehicular Technology Magazine, 2018*

Paper B. REMIND: A Framework for the Resilient Design of Automotive Systems [7]

T. Rosenstatter, K. Strandberg, R. Jolak, R. Scandariato, T. Olovsson

A changing automotive landscape with increased complexity and connectivity aligns with the difficulties of ensuring security in these systems; although, the automotive journey towards a more centralized architecture facilitates the realization of security and resilience techniques. Still, it is not realistic to assume protection against all cyber-security incidents; thus, an additional approach is required to endure security breaches.

In essence, detection and prevention occur at a first layer, and if this fails, resilience to endure the breach and an attempt to recover from the breach is required. For example, switching to a safe operational state with limited functionality to provide a safe stop of the vehicle. Thus, ensuring resilience for vehicles is imperative. In this paper, we investigate:

- What work concerning resilience techniques, fault tolerance, and dependability exist in the literature?
- How can these techniques be used to detect, mitigate, recover and endure cyber-attacks in vehicles?

We perform a literature study on resilience techniques, fault tolerance, and dependability. The result is the REMIND resilience framework providing techniques for attack detection, mitigation, recovery, and resilience endurance. Moreover, we provide guidelines on how the REMIND framework can be used against common security threats and attacks and further discuss the trade-offs when applying these guidelines. Thus, the core contributions are:

- A framework for the design of resilient automotive systems.
- Techniques are categorized into four groups: detection, mitigation, recovery, and endurance, representing their purpose.
- Techniques of different categories can be combined to create layers of security.
- Techniques are mapped to classes of automotive assets, and pros and cons are provided for each technique to further support design decisions.

In summary, we provide a framework to guide architects on the selection of appropriate resilience techniques in the design of automotive systems.

Statement of contributions. This is a joint work with the main and co-authors. Thomas Rosenstatter was the lead for the idea, the literature review, and the design of the taxonomy for resilient techniques. I have written and contributed particularly with the attack model, assets identification, and the associated table. Rodi Jolak was primarily responsible for the REMIND resilience guidelines.

Appeared in: *IEEE Secure Development, 2020*

Paper C. Resilient Shield: Reinforcing the Resilience of Vehicles Against Security Threats [2]

K. Strandberg, T. Rosenstatter, R. Jolak, N. Nowdehi, T. Olovsson

Various vehicle cyber-attacks have occurred in the past. To investigate the risk of these attacks and ensure that future vehicles can withstand these types of attacks, we have performed a comprehensive threat and risk analysis on published attacks on vehicles from the past 10 years. Thus, in this work, we investigate:

- What types of cyber-attacks have been performed on vehicles for the last 10 years?

- How can we achieve security and resilience to mitigate such attacks?

We have performed the *SPMT methodology* proposed in paper A for the complete vehicle as produced by the manufacture. In the first phase of the *SPMT*, the *Start Phase*, we define vehicle assets, threat actors, their motivations, and objectives, which gives rise to a threat model. In the second phase, the *Predict Phase*, we study attacks that might impact the defined assets from the previous phase. We analyze these attacks based on an attack probability and consequence criteria and appoint a risk value for each attack. Each attack is further categorized into a category based on the attack type. This phase gives rise to an attack model. In the next phase, the *Mitigate Phase*, we define required security and resilience enhancements against all threats as security goals. Moreover, we define detailed directives on how to fulfill these security goals. The resilience techniques defined in paper B are further incorporated into these detailed directives. The *Mitigate Phase* gives rise to the *Resilient Shield*, a framework for designing resilient automotive systems. Thus, the core contributions are:

- By applying the *SPMT methodology*, we performed a comprehensive threat and risk analysis of 52 published attacks against vehicles from the past 10 years.

- We have developed a threat model for securing vehicles by identifying vital vehicle assets and the related potential threat actors, their motivations and objectives.

- We have developed a comprehensive attack model created from the analysis of the identified threats and attacks, further filtered and categorized based on attack type and risk criteria related to the probability and consequences of the attack.

- We present an exhaustive mapping between asset, attack, threat actor, threat category and resilience mechanism for each attack.

- We define necessary security and resilience enhancements for vehicles, which also validates the effectiveness of the *SPMT methodology*.

In summary, we propose *Resilient Shield*, a comprehensive resilience framework that considers actual cyber-attacks against defined automotive assets. Mitigation techniques are proposed giving rise to a vital baseline of protection against common security threats and attacks.

Statement of contributions. This is a joint work with the co-authors. I contributed as the lead for the idea and performed the writing of the manuscript. I also contributed to the identification of the assets, threat actors and their motivations and objectives, and the high-level security goals. The identification of the detailed directives and the work to find and review attacks has been a joint effort between me and the secondary author. The assignment and mapping of mitigation techniques, STRIDE categories, automotive assets, and threat actors have been a collaborative effort between the authors.

Appeared in: *IEEE 93rd Vehicular Technology Conference, 2021*

Paper D. UniSUF: a unified software update framework for vehicles utilizing isolation techniques and trusted execution environments [10]

K. Strandberg, D.K. Oka, T. Olovsson

A modern vehicle can contain over 100M lines of code, and although there are different code quality, we can assume there is at least one bug per 1000 lines of code; thus, there will be more than 100k bugs in a modern vehicle. We can further assume that at least 1% (1k) of these bugs are exploitable vulnerabilities [11]. The severity can vary from causing a lamp to blink to affecting safety-critical systems such as braking or steering with potentially hazardous outcomes. Thus, the ability to hastily and securely patch vulnerabilities is imperative and a prerequisite when securing modern cars. However, current solutions lack necessary details for a versatile, unified, and secure approach covering various update scenarios, e.g., over-the-air, with a workshop computer, at factory production, or using a diagnostic update tool. Thus, in this paper, we investigate:

- How can we ensure the ability to hastily and securely patch software vulnerabilities in vehicle software for various required scenarios within the automotive domain?

Consequently, one of the stated security goals from paper C is investigated more in-depth: *SG4: Secure Software Techniques*. The core contributions are:

- We have investigated several software update use cases and identified constraints and conditions for a unified and versatile approach.
- Considering these constraints and conditions, we propose an approach for vehicle software updates, where all data needed for a complete software update is securely encapsulated into one single file.
- This file can be processed in several update scenarios and executed without any external connectivity since all data is inherently secured.
- We provide a comprehensive overview for a possible secure implementation covering the whole software chain from producer to receiver.
- Our approach has been reviewed with automotive software update architects to ensure that the proposed approach can be practically deployed and efficiently adopted for vehicle software updates.

In summary, we propose the *Unified Software Update Framework for Vehicles (UniSUF)* that utilizes isolation techniques and trusted execution environments to ensure a secure software update process.

Statement of contributions. This is a joint work with my supervisor Tomas Olovsson and Dennis Kengo Oka. I contributed as the lead for the idea, performed the development of the framework and the writing of the manuscript.

Appeared in: *19th escar Europe : The Worlds Leading Automotive Cyber Security Conference, 2021*

1.4 Summary

The lack of methods and techniques specific to the automotive industry to ensure a secure and resilient vehicle design is troublesome. We are moving towards a future where almost all devices, vehicles included, have some sort of connection to the outside world. The ever-increasing complexity aligned with increased connectivity gives the potential for malicious actors to compromise these devices with the risk of hazardous outcomes.

The possibility to *detect*, *prevent*, *respond* and *predict* potentially malicious events is imperative to ensure the safety of the vehicle and its passengers. Detection is required for an informed decision around how to manage events (i.e., the response), e.g., prevent, allow or limit impact of events. One way to predict the future is to know the past. Thus, the possibility to securely trace back previous events is a prerequisite for vehicle safety that enables fixing the detected vulnerabilities e.g, software bugs, and the potential to find the origin and differentiate between cyber-attacks and hardware and software failures. Knowledge of existing and potential future threats aligned with the corresponding mitigation techniques is required to ensure a vehicle design that fulfills required security and resilience properties. However, research around cyber security and resilience for modern vehicles is a relatively new area; thus, more work is needed.

We have contributed with a systematic methodology to find and mitigate vulnerabilities in vehicles (paper A). We have shown how the methodology can be practically used to strengthen and enhance vehicle security, resilience, and safety (Paper C). We have provided a framework for resilience techniques (Paper B) and incorporated these techniques practically (Paper C). Finally, to meet the increasing number of software bugs in vehicles software, we have provided a framework to hastily and securely deploy vehicle software (Paper D).

1.5 Conclusion and Future Work

Conclusion. Chapter 2 defines the *SPMT methodology* further performed in Chapter 4 to derive the *Resilient Shield*, a framework for resilient vehicle design. Chapter 3 defines and categorizes resilience techniques, i.e., the *REMIND* framework, from where techniques are incorporated into the detailed directives in *Resilient Shield*. Thus, *SPMT* and *REMIND* lay the foundation for *Resilient Shield*. *Resilient Shield* defines security goals and detailed directives to ensure a secure and resilient vehicle design. *Resilient Shield* further lays the base for *UniSUF* as it goes in-depth into *SG4: Secure Software Techniques*. Upcoming papers consider the security goals and detailed directives defined in *Resilient Shield* as a base towards a secure and resilient vehicle design.

Future work. One of the security goals stated in *Resilient Shield* is *SG8: Forensics*. It is imperative to investigate the root cause of incidents and failures and be able to differentiate between, e.g., software failures and cyber security issues. However, not much work has been done within the automotive digital forensic area. Thus, as a first step and shown in Figure 1.5, we aim to perform a systematic literature review and identify the current literature on automotive digital forensics, including research gaps, issues, and challenges. We aim to

identify surveys, technical solutions, and the type of digital forensics data relevant to forensics investigations. As a second step, we aim to propose a framework for the design of automotive digital forensics. Furthermore, we aim to extend *UniSUF* with in-depth requirements, a security evaluation, and a comparison with other approaches.

Chapter 2

Securing the Connected Car: A Security Enhancement Methodology

Adapted version that appeared in IEEE Vehicular Technology Magazine 2018

K. Strandberg, T. Olovsson, E. Jonsson

Abstract. A new era is upon us, an era where Internet connectivity is available everywhere and at all times. Cars have become very complex computer systems with about 100 million lines of code and more than 100 electronic control units (ECUs) interconnected to control everything, including steering, acceleration, brakes, and other safety-critical systems. However, cars were never created with Internet connectivity in mind, and adding this connectivity as an afterthought raises many security concerns. As a result, a security-enhancing approach that considers the entire process from product development to market introduction is required.

This article suggests using a methodology known as start, predict, mitigate, and test (SPMT). Its purpose is to predict and mitigate vulnerabilities in vehicles using a systematic approach for security analysis specifically adapted for vehicles. The SPMT methodology builds on existing methodologies and models that are applicable to different phases in a vehicles life cycle as well as on new ideas. Unlike other methods, however, the SPMT methodology covers a vehicles entire life cycle, which results in security and safety enhancements, something that cannot be achieved by existing methodologies.

2.1 Introduction

The Internet of Things (IoT) has moved society into a new era where a growing number of devices have Internet capabilities and are behaving more like computers (e.g., smart TVs and washing machines). Access possibilities are provided via USB sticks, Bluetooth devices, or Wi-Fi/cellular connections. Modern cars can have more than 100 ECUs and contain roughly 100 million lines of code [12]. Today, a car is not just a car, it is a computer on wheels. ECUs are responsible for various safety functions such as steering and brakes, and new functionality is constantly being introduced in the automotive industry, which calls for an increase in the number of ECUs and the amount of code. As a result, this increases the likelihood of attacks by hackers. Electrical systems in vehicles are no longer isolated systems, but, rather, they are vulnerable to cyber-attacks.

2.1.1 Context

A vehicle is a safety-critical system, which means that vulnerabilities can potentially lead to life-threatening hazards. Koscher et al. [13] discuss security implications in vehicles, such as vulnerabilities in the controller area network (CAN) protocol. CAN nodes communicate with broadcast messages, which enables the possibilities for malicious devices to detect, manipulate, and inject messages anywhere on the CAN bus. Furthermore, CAN is a priority-based protocol with no authenticator field and is therefore vulnerable to both denial of service (DoS) attacks and spoofed packets. Access control, which enables firmware updates, is usually performed using obsolete proprietary encryption algorithms. In many cases, these algorithms are publicly known [13] and use short cryptographic keys indicating that the access control is vulnerable to various kinds of attacks (e.g., a bruteforce attack followed by malicious code injection through a firmware update). Koescher et al. managed to disable CAN communications and update the firmware while a test subject was driving. They have also found imperfect network segmentation, thereby using the infotainment unit in the vehicle as a bridge to attack other vehicle ECUs. Furthermore, they found that reverse engineering of the firmware is usually not necessary for attacks, since a relatively small range of CAN packets is valid. This makes it possible to simply fuzz the network with various packets and observe the response.

By exploiting these and other vulnerabilities, the commandeering of many vehicle functions, including safety-critical functions both in driving mode at high speed and while at a standstill, is an unfortunate reality. Considerable vulnerability is demonstrated in vehicles when physical access is obtained. However, the Koscher et al. research was met with resistance from the automotive industry for not being relevant; the industry argued that, with physical access, an individual could just as well cut cables or destroy other components in the vehicle. In response, Checkoway et al. [12] provided evidence that external attacks are also possible via a wide range of entry points and categorized these attacks into three groupings: *indirect physical access*, *short-range wireless access*, and *long-range wireless access*.

2.1.2 Indirect Physical Access

Indirect physical access can refer to the vehicles media player, since music can be crafted with malicious content. The media player can also be used to bridge an attack on other components in the vehicle if it is not adequately isolated from the vehicles main network. This was demonstrated when a standard International Organization for Standardization 9660-formatted compact disk that contained a specific filename was inserted into a vehicles media player, and the content of the file was automatically used to re-flash the unit [12]. Additionally, since the media player can parse complex files, they managed to add content to Windows Media audio files, that played normally on a PC, but, in a vehicle, had the side-effect of also sending arbitrary CAN packets on the vehicle network.

2.1.3 Short-range Wireless Access

Short-range wireless access refers to Bluetooth devices, remote keyless entry, and the tire pressure monitor system. Checkoway et al. demonstrated successful attempts to compromise a vulnerable Bluetooth implementation in a telematics unit. They planted a Trojan horse in an application for an Android 2.1 operating system that could be uploaded to the Android market. For example, when a device that contains this application is paired with the vehicle, it would exploit a buffer-overflow vulnerability enabling the execution of arbitrary code in the telematics unit. Checkoway et al. also successfully paired a laptop to the vehicle with no user interaction.

2.1.4 Long-range Wireless Access

Long-range wireless access refers to global positioning systems, digital audio broadcasting, and remote telematic systems. Checkoway and his colleagues used a cellular connection to first exploit an authentication vulnerability, and then a buffer-overflow vulnerability in the telematics unit. Following this, the telematic unit was forced to download additional malicious content from the Internet. Moreover, for each vulnerability they demonstrated, they were able to obtain complete control over the vehicles systems. Remote attacks received considerable attention when Charlie Miller and Chris Valasek performed a successful attack on an automobile by using the Internet to gain control of its vital systems [14]. Similar to Checkoway et al.s experiments, Miller and Valasek used a cellular channel in the vehicle as leverage, but, in this case, an open port (6667) made this attack possible. As a result, 1.4 million vehicles were recalled by the manufacturer. Another attack occurred when Samy Kamkar managed to remotely unlock an OnStar-enabled General Motors car [15].

2.1.5 Goal and Approach

The goal of this article is to introduce a practical, security-enhancing methodology for identifying and mitigating vulnerabilities in vehicles throughout their life cycle (i.e., from the product development phase to the market introduction phase). The approach used for creating the SPMT methodology was divided into three parts. First, a study of various security methods and models used in

different areas was conducted, and methods or parts of methods relevant to the automotive industry were selected. Secondly, a theoretical and empirical study of attacks against vehicles that have occurred was conducted. The understanding of how and why attacks are successful is necessary to develop a methodology capable of finding vulnerabilities related to these attacks, as well as to help identify other vulnerabilities. Lastly, the methodology was defined and implemented based on the conclusions from the two previous parts.

2.2 Models, concepts, and tools

The definition of a security model varies depending on the category. Generally, it specifies how to enforce a security policy and defines how to maintain security for a system or entity. The following models in particular have been effective as they pertain to the development of the SPMT methodology.

- *Spoofing of user identity tampering repudiation information disclosure DoS elevation of privilege (STRIDE).* A threat model proposed by Microsoft as a scheme for categorizing and identifying known threats according to different vulnerabilities or the malicious intents of the attacker [16].
- *Damage reproducibility exploitability affected users discoverability (DREAD).* A model also proposed by Microsoft as the next step after threat modeling. It is used to evaluate the risk for each threat by quantifying, comparing, and prioritizing the risk [16].
- *E-safety vehicle intrusion protected applications (EVITA).* A European Union project whose objective was to provide the basis for secure release of applications based on vehicle-to-infrastructure and vehicle-to-vehicle communication. EVITA proposed a method for security and safety/risk analysis for a vehicles network and also proposed a secure architecture and communication protocol [17].
- *Healing vulnerabilities to enhance software security and safety (HEAVENS).* A project that was conducted between 2013 and 2016 [18]. The HEAVENS security model policy is divided into two categories: security objectives and security attributes. The security objectives are taken from the EVITA model, and the security attributes are taken from the STRIDE model. The main workflow from top to bottom for HEAVENS is shown in Table 2.1.

Table 2.1: The workflow of HEAVENS

TOE	Define Target of Evaluation
Threat Analysis	Define the possible threats against the TOE
Risk Assessment	Grade the severity of those threats
Security Requirements	Define needed mitigations against those threats

- *Threat agent risk assessment (TARA)*: a predictive methodology developed by Intel to define the security risks that are most likely to occur. This methodology is based on the presumption that it is too expensive

and impractical to defend against all possible vulnerabilities; therefore, by choosing the most important ones, results are maximized and costs are minimized. TARA identifies all possible threats by assessing different lists of known threats. Those threats are then filtered so that only the most serious ones remain [19].

The following concepts and tools are used in this article:

- *Target of Evaluation (TOE).* This is the product or system that is the subject of evaluation.

- *Attack Tree.* A diagram visualizing the steps needed for a realized threat on an asset.

- *Threat Modelling.* The first step when creating a threat model is to evaluate which assets need protection and how they are threatened. Known vulnerabilities and attacks (i.e., searching known vulnerability databases) are then identified.

- *Risk Assessment.* This is a continuation of the previous step when the risks of the threats are evaluated. The threats are prioritized with respect to the probability of occurrence and its consequences; these are weighted against the cost of mitigation. After a threat modeling and a risk assessment are performed, it is important to evaluate the possible mitigations.

- *Attack Attempts.* A thorough understanding of attacks related to the TOE and its corresponding hacker tools is imperative. Hence, studies of attacks and adaptations of these attacks as tests should be a part of a security evaluation verifying mitigations. To verify this concept we performed attacks against the vehicle Wi-Fi network and the Volvo On Call service in a Volvo XC90 as shown in Figure 2.1 [20].

Figure 2.1: Attack attempts against Volvo XC90

2.3 The SPMT Methodology

The SPMT methodology is divided into two main stages. First, the procedure is applied to a specific TOE, i.e., an ECU or other unit. The TOE in question is then integrated into the vehicle and becomes a part of the vehicles network. The procedure will then be repeated and adapted to the integration and performed again. By considering the devices, the integration, and system tests as a whole, a broad security perspective is achieved.

Figure 2.2, shows the procedures and concepts that have inspired the development of the SPMT methodology. The start phase defines the TOE, security policies, and a brief threat modeling based on common security and safety concepts. The predict phase consists of threat modeling by using the STRIDE threat model for categorizing vulnerabilities and threats in different lists. The idea of filtering comes from the TARA methodology (although the filtering of these lists is based on a qualitative assessment inspired by the DREAD methodology), the EVITA, and the HEAVENS projects, and by the establishment and analysis of attack trees.

The mitigate phase is based on brainstorming and further analysis of countermeasures related to the attack trees from the predict phase, as well as a quantitative assessment. Hence, a hybrid approach of a qualitative and quantitative assessment is considered. The test phase consists of practical security-related tests in conjunction with automated fuzz and vulnerability scanning tools. Penetration testing inspired by hacker attacks and their tools results in the creation of tests and tools that are adapted for the TOE. Figure 2.3 shows an illustration of the input and output to each phase.

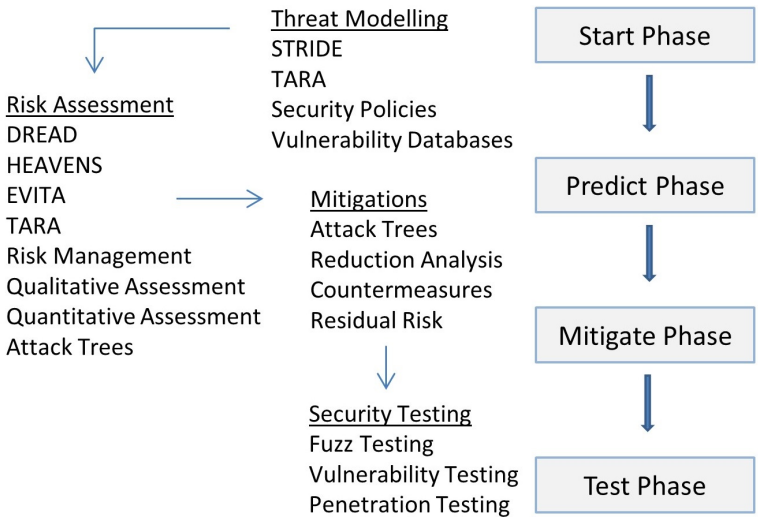

Figure 2.2: Illustration of the SPMT phases

2.3.1 Start Phase

The start phase is conducted in a workshop-like manner with discussion and brainstorming sessions on how to best implement this idea. This phase is

divided into two stages:

1. The establishment of what needs protection, i.e., what is the TOE? Which assets are affected? How are those assets interconnected? How can a compromised asset affect other assets? What are the possible entry points for threats to the vehicle?

2. Defining security policies, i.e., what does it mean for the system to be secure/insecure? How are security and safety-related attributes achieved for the TOE, such as those listed in Table 2.2?

Table 2.2: Security and Safety attributes

Confidentiality	Authorization	Privacy	Reliability
Integrity	Authenticity	Isolation	Least privilege
Availability	Freshness	Maintainability	

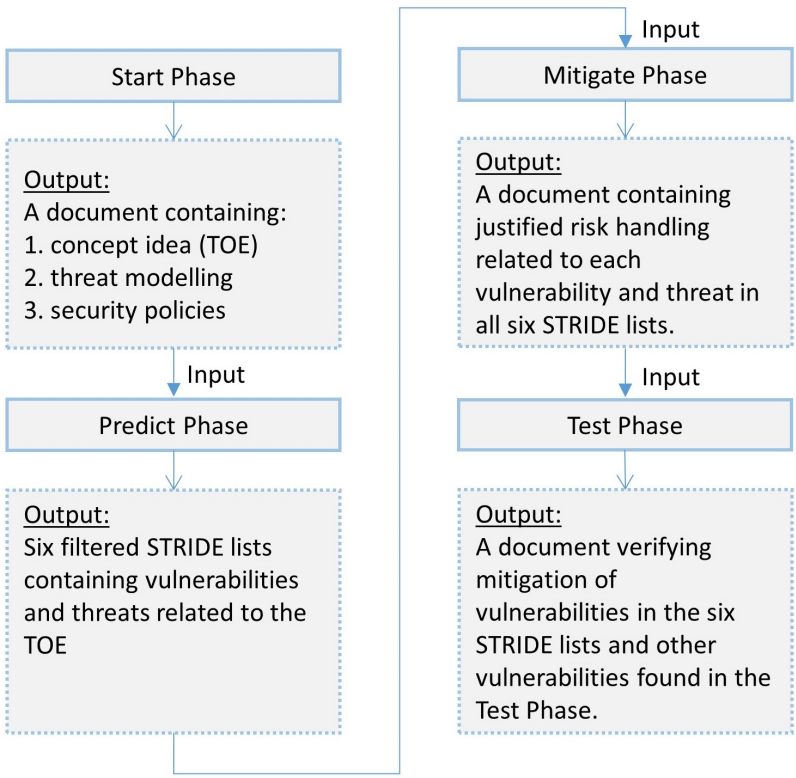

Figure 2.3: Illustration of the input and output to SPMT phases

This phase produces a document containing the concept idea, a brief threat model for the TOE, and security policies, including high-level directives for the enforcement of security attributes.

2.3.2 Predict Phase

This phase consists of predicting threats against potential vulnerabilities in the TOE and its related assets. This is partly accomplished by searching known vulnerability databases, e.g., the common vulnerability enumeration (CVE) database.

Table 2.3: The six STRIDE lists

Spoofing	Authenticity/Freshness	S-LIST
Tampering	Integrity	T-LIST
Repudiation	Non-Repudiation/Freshness	R-LIST
Information disclosure	Confidentiality/Privacy	I-LIST
Denial of Service	Availability	D-LIST
Elevation of Privilege	Authorization	E-LIST

As shown in Table 2.3, this list is filtered by keywords and divided into six lists based on the STRIDE threat model. It is then managed by applying four steps:

1. Compose six lists based on the STRIDE model centered on the CVE identifier.
2. Automate the filtering of these lists based on keywords, i.e., excluding threats not relevant to the TOE.
3. Evaluate the vulnerabilities in each list, and remove those that are not relevant (the filtering from the previous step makes possible a more manually filtered approach).
4. Perform a qualitative assessment of the remaining vulnerabilities based on the calculated risk value (i.e., risk = probability for a realized threat x consequences):

   - Grade the probability of a realized threat on a scale from 1 to 3 (i.e., 1 = low, 2 = middle, 3 = high). Base the probability for a realized threat on how easy it is to exploit a vulnerability in the following manner:
     - Where, when, and in what situation can the attack be carried out?
     - What expertise is required of the attacker? What tools are needed and how difficult is it to acquire these tools?
     - What time is needed to perform the attack?

     Based on the answers to these questions, the probability of a realized threat is graded and the highest value (low, middle, or high) is chosen.
   - Grade the consequences on a scale from 1 to 3 (i.e., 1 = low, 2 = middle, 3 = high). If the attack is successful, what are the consequences? The consequences of the following four attributes (the objectives from EVITA) are graded and the highest value (low, middle, or high) is chosen.

  - Operational (graded on a scale from 1 to 3): Are any operational factors affected?
  - Safety (graded on a scale from 1 to 3): Is safety affected?
  - Privacy (graded on a scale from 1 to 3): Is any personal information compromised?
  - Financial (graded on a scale from 1 to 3): How are financial factors affected?
- Calculate the risk by multiplying the result from the probability for a realized threat with its corresponding consequences, as shown in Figure 2.4. These values determine the risk severity graded on a scale from 1 to 9.

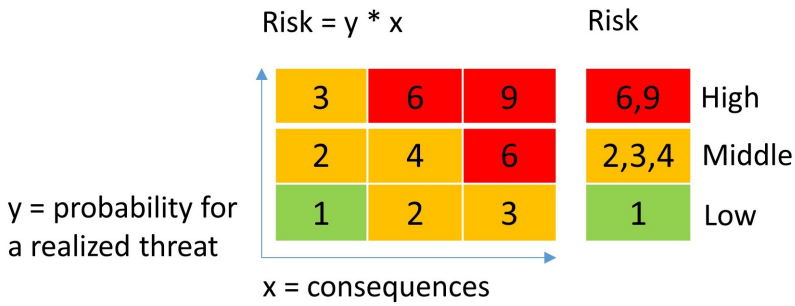

Figure 2.4: Visualizing the risk calculation process

- Remove all vulnerabilities that result in a risk = 1 from the list (acceptable risk).
- Add other relevant threats which are missing from each category by brainstorming. Consider both inside and outside threats, i.e., employees with privileges and attackers without privileges.
- Sort the remaining threats in the lists based on risk value, with the highest value first.
- Create attack trees for the threats with a risk value of 6 or 9 (prioritized threats) to connect the vulnerability with the threat, i.e., the conditions (leaf nodes) that need to be met for a successful attack (i.e., an attacker traverses from a leaf node to the root of the tree).
- Assess these vulnerabilities and threats in the next phase.

This phase produces a document containing six filtered STRIDE lists containing vulnerabilities and threats to the TOE. Note that each TOE needs its own set of lists. This requires considerable effort initially, but, once it is done, it only needs to be assessed when new potential vulnerabilities or threats are discovered.

2.3.3 Mitigate Phase

The filtered STRIDE lists are assessed from the previous phase in a workshop-like manner through brainstorming. The vulnerabilities with the highest

risk are considered first, and mitigation techniques that prevent attacks are discussed and implemented if possible. This is done by analyzing the conditions (leaf nodes) in the attack trees from the predict phase, e.g., by placing a countermeasure at each leaf node. The closer to the root a countermeasure is situated, the more leaf nodes are covered. Some leaf nodes can be attained by more than one attack; hence, a countermeasure can mitigate more than one attack. Figure 2.5 shows an example of an attack tree visualizing different attacks affecting the brakes in a vehicle. Countermeasures at the node labeled Tamper with the ECU software provide protection against all three attacks in the attack tree. We can also see that we need other countermeasures relating to the node labeled DoS attack. Finding these commonalities is termed a reduction analysis and can be very effective.

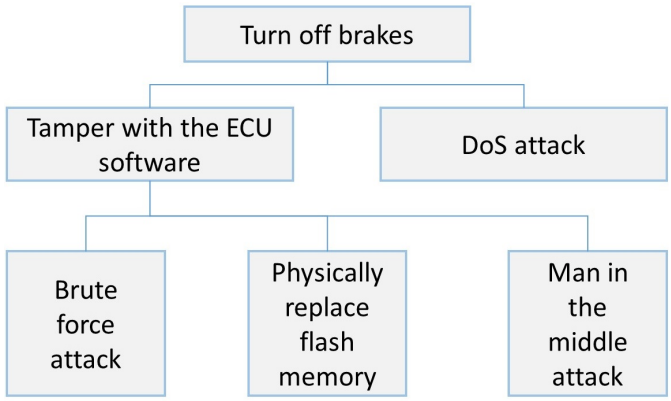

Figure 2.5: A simplified example of an attack tree

In this process, the cost for mitigation versus the value of the asset (TOE) is considered (since this enables cost-efficient mechanisms). This value is estimated by a quantitative assessment calculation, i.e., Annualized Loss Expectancy (ALE) = Annual Rate of Occurrence (ARO) x Single Loss Expectancy (SLE). In turn, the SLE is the product of the asset value (AV) and the exposure factor (EF). The exposure factor describes the monetary asset loss for a realized threat expressed in a percentage [21].

For example, if AV equals 1,000 units and EF equals 25% and the probability of the realized threat (ARO) is once in every ten years, the calculation is expressed as ALE = ARO x SLE = ARO x AV x EF = 0.1 x (1,000 x 0.25) = 25. Hence, the mitigation cost per year should not exceed the ALE value of 25 units. However, this is a very rough estimation since it might be difficult to estimate the costs for potential damage. Still, it is not feasible if mitigation costs are higher than the monetary value of the asset that is being protected; and it should not exceed the financial loss for a realized threat.

The residual risk, i.e., the remaining risk after a countermeasure has been applied, must also be considered. It could be acceptable to decrease the mitigation cost by increasing the risk. However, risk management is usually determined by the mitigation, transference, avoidance, or acceptance of the risk. Transferring the risk to an insurance company is also an option. Avoiding risk can be done by terminating the activity, which introduces the risk or by

accepting that the risk might be necessary if mitigation is not feasible. This phase produces a document that contains justified risk handling related to each vulnerability and threat in all six STRIDE lists.

2.3.4 Test Phase

This phase consists of practical security testing and verifying the output from the predict and mitigate phases (i.e., the filtered STRIDE list and mitigations therein). These tests could also reveal more vulnerability. When possible, the use of automated software and hardware tools is recommended. The following three tests are used: fuzz testing, vulnerability testing, and penetration testing.

Fuzz testing considers mostly negative testing, i.e., testing inputs that are unexpected. Positive testing (valid, expected output testing) is assumed to be a part of normal function testing and not necessarily part of security testing; however, security and normal function testing should be integrated. The difference between vulnerability testing and penetration testing is subtle, and in many cases they overlap. Vulnerability testing is defined more as an automated approach with scanning tools used to find vulnerabilities, whereas penetration testing is a manual approach wherein vulnerabilities that are difficult to automate (e.g., TOE-adapted attacks as tests) are tested. Penetration testing considers both black- and white-box testing. Also considered for tests are internal and external perspectives. An example of white-box testing against a vehicles wireless local area network (WLAN) router as a TOE can be a vulnerability scan of the private internet protocol (IP) address (i.e., test access via the vehicles internal Wi-Fi network). An example of black-box testing of the TOE might instead be a vulnerability scan of the official IP address (i.e., to test access from the Internet). An example of an operating system specialized in penetration testing is Kali Linux, which contains many useful tools [22]. An example of a hardware tool, applicable for evaluation of an access point, is Pineapple [23]. The output from this phase results in a document verifying mitigations toward vulnerabilities found in this phase and the six STRIDE lists.

2.4 Integration into the vehicle

At this point, the test phase has mainly considered the testing of a device (e.g., a single ECU). As previously mentioned, it is important to consider how the TOE affects, and is affected by, other assets (e.g., ECUs) in the vehicle. The TOE is not changed in this step; rather, the focus shifts toward a broadening of the scope to consider the complete vehicle relative to the TOE. Hence, when the test phase passes the device tests, the focus shifts and turns once more toward the integration tests. A compromised ECU (e.g., the vehicle infotainment system) might be used as a platform to send critical CAN messages to other safety-critical ECUs, if not correctly isolated or if the communication is not filtered. An example of tests that could be performed against the vehicle network as a whole (including the TOE) is an attack that exploits the error handling at the CAN bus by injecting messages, eventually forcing one or many ECUs to shut down [24]. A test such as this can be automated with scripting languages (e.g., Python or Communication Access Programming Language).

Scripting and tools are typically adapted to the TOE; an example of an effective analysis tool for the total vehicle network is CANoe from Vector [25].

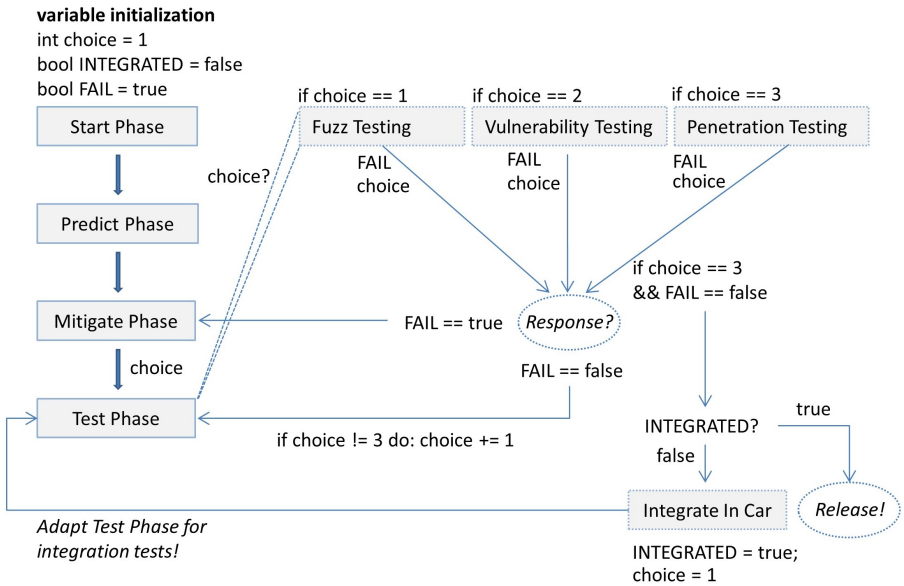

Figure 2.6: Flowchart of SPMT phases

2.5 Flowchart and Pseudocode for the whole process

In this section, all phases are visualized in a complete manner by combining stated diagrams, as shown in Figure 2.6, with the pseudocode shown in Figure 2.7. In the pseudocode, the usability of the SPMT methodology is displayed with the vehicles WLAN router as a TOE; it is assumed that the start and predict phases have been completed. It is also assumed that the tools used in the test phase are Defensics, OpenVAS, and Kali Linux together with certain hardware tools. Hence, at row 1, the mitigate phase is entered. All identified vulnerabilities are corrected at row 2. The test phase is presented at row 3, with the variable choice set to fuzz testing (the first test to be performed). At rows 4 and 5, fuzz testing is entered. At row 6, the TOE is scanned with the Defensics tool. At rows 7-11, the response is handled. If the test fails, the mitigate phase is re-assessed, the vulnerabilities are addressed, and fuzz testing is performed again. If the fuzz testing is successful (row 11), the vulnerability testing (row 12) can begin. The vulnerability testing (rows 12-18) works in the same manner as the fuzz testing, except that the OpenVAS tool is used. Penetration testing, the last test, differs from the others because, if this test is successful, integration of the TOE to the car is performed (row 25); this can only be done once (row 24). At row 27, all tests are repeated once more (although they are adapted for integration testing) after the TOE is integrated into the vehicle. If all tests are successful, the product can be released to

```
1:  Mitigate Phase(choice):
2:    do:   mitigate vulnerabilities for choice;
3:    goto: Test Phase(choice);
4:  Test Phase(choice):
5:      1:   //***Fuzz Testing***//
6:        do: scan TOE (use: Defensics)
7:           Response?
8:             CASE(FAIL):
9:                True ? goto: Mitigate Phase(1);
10:                  else
11:                  goto: Test Phase(2);
12:     2:   //***Vulnerability Testing***//
13:       do: scan TOE (use: OpenVAS)
14:          Response?
15:            CASE(FAIL):
16:               True ? goto: Mitigate Phase(2);
17:                  else
18:                  goto: Test Phase(3);
19:     3:   //***Penetration Testing***//
20:       do: attacks against TOE (use: Kali Linux
              hw Tools)
21:          Response?
22:            CASE(FAIL):
23:               True ? goto: Mitigate Phase(3);
24:                  else if !INTEGRATED
25:                     do: Integrate In Car and
                            INTEGRATED = true;
26:                     //***Repeat all tests***//
27:                     goto: Test Phase(1);
28:                        else
29:                        do: Release;
```

Figure 2.7: Pseudo code to exemplify the usability for the SPMT Methodology

market (row 29). Within the industry, components are ordered by external suppliers in a more-or-less developed status. In these cases, it is possible to enter an appropriate phase, e.g., the last phase (test phase) to begin practical tests.

Considering a released product, it is possible to assess SPMT as shown in Figure 2.8 by continuously monitoring for new threats or system changes.

For the former, new threats need to be assessed and placed into the appropriate list (predict phase), and, for the latter, system changes need to be reassessed with practical tests (test phase). Hence, the SPMT is adaptable and can meet the requirements to cover both development and after-market security analysis.

2.6 Discussion and Contributions

In this chapter we highlight and discuss some of the advantages and benefits of the proposed methodology.

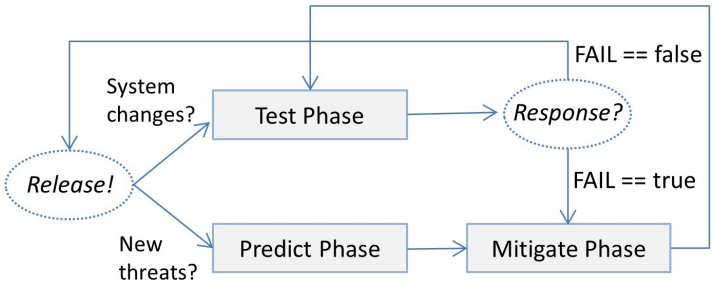

Figure 2.8: Possible scenarios after product is released to market

- *Innovative and important.* The SPMT methodology shows how components of existing models and methods of security can be applied successfully when securing connected cars. This approach is both innovative and important for the automotive industry.
- *Comprehensive and systematic.* SPMT methodology offers a comprehensive systematic approach to security analysis specifically adapted for vehicles.
- *High coverage.* By following the SPMT methodology, comprehensive threat lists with mitigations are created for high-priority threats. High coverage is given for mitigations against threats with evaluated and prioritized risks considering the operational, safety, privacy, and financial factors that relate to vehicles. However, it still remains to be seen how large the coverage is in quantitative terms.
- *Security as entirety.* The SPMT methodology considers both the TOE from a device perspective and when the device has been integrated into the vehicle as a whole.
- *Cost and time effective.* The SPMT methodology simplifies the process of reassessing security after system changes and new threats. This is done by making use of existing documentation from earlier assessments and conducting practical tests concerning the changes in the test phase. Therefore, the SPMT methodology is both cost- and time-effective.
- *Adaptable.* The SPMT methodology is applicable for adaptation to different situations by selecting different tools and adapted attacks for different TOEs. It is possible to start the adaptation in different phases depending on the situation, as shown in Figure 2.8. Integrating the SPMT methodology to current development processes for the automotive industry is a straightforward approach, as exemplified in the Software V-model in Figure 2.9.

2.7 Conclusion

This article defines a security-enhancing, and thus safety-enhancing, methodology that identifies and mitigates vulnerabilities in vehicles. This is achieved

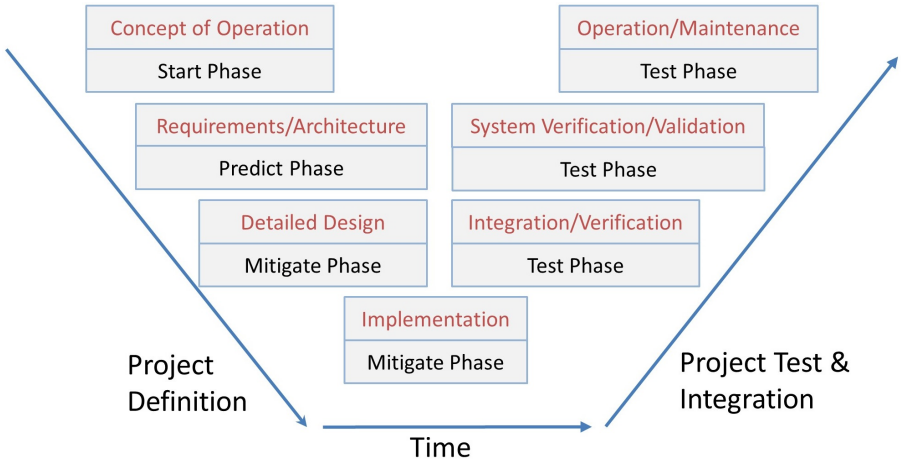

Figure 2.9: The SPMT methodology integrated into the software V-model process

through a comprehensive, systematic approach to security analysis, specifically adapted for vehicles. This methodology covers security analysis for the entire process, from product development to market introduction, by adapting and integrating relevant parts of existing security methods and models and by incorporating new ideas suitable for vehicular domain. This methodology, named SPMT, is essential for the automotive industrys efforts to improve security and safety.

Chapter 3

REMIND: A Framework for the Resilient Design of Automotive Systems

---

Adapted version that appeared in IEEE Secure Development 2020

T. Rosenstatter, K. Strandberg, R. Jolak, R. Scandariato, T. Olovsson

Abstract. In the past years, great effort has been spent on enhancing the security and safety of vehicular systems. Current advances in information and communication technology have increased the complexity of these systems and lead to extended functionalities towards self-driving and more connectivity. Unfortunately, these advances open the door for diverse and newly emerging attacks that hamper the security and, thus, the safety of vehicular systems. In this paper, we contribute to supporting the design of resilient automotive systems. We review and analyze scientific literature on resilience techniques, fault tolerance, and dependability. As a result, we present the REMIND resilience framework providing techniques for attack detection, mitigation, recovery, and resilience endurance. Moreover, we provide guidelines on how the REMIND framework can be used against common security threats and attacks and further discuss the trade-offs when applying these guidelines.

3.1 Introduction

In the past years great effort has been spent in publishing guidelines and standards for security frameworks specific to their domains and in identifying security principles. Examples range from the NIST guideline for cybersecurity in smart grids [26], the cybersecurity guideline for ships [27], cybersecurity guidelines for the automotive domain [28–30] and the upcoming ISO/SAE standard for cybersecurity engineering for road vehicles, namely ISO 21434 [5].

Resilience is the next step towards reliable, dependable and secure vehicular systems. Vehicles need to be able to mitigate faults, errors, attacks and intrusions that would ultimately result in failures in order to withstand safety and security threats from their environment. We define automotive resilience as the *"property of a system with the ability to maintain its intended operation in a dependable and secure way, possibly with degraded functionality, in the presence of faults and attacks."* This definition is inspired by Laprie's definition [31] and the definition of network resilience by Sterbenz et al. [32]; however, the chosen definition highlights that faults or changes, e.g., functional and environmental (see [31]), can also be originated by an attacker whose aim is to disrupt the system.

Resilience can be obtained in many different ways and on different levels, i. e., hardware, software or (sub)-system level. Today's internal architecture of vehicles is quite complex and can be distributed over more than hundred so-called Electronic Control Units (ECUs). However, we are currently in a transition towards a more centralized architecture where functions will be concentrated on much fewer and more powerful ECUs [33]. These central ECUs are connected to sensors, actuators, external communication media and to some extent to smaller legacy subsystems. Such a centralized architecture enables vehicle OEMs not only to perform more resource intensive operations needed for autonomous driving, but also allows to introduce new designs and technologies needed to secure and protect these highly connected and autonomous vehicles. Virtualization is seen as one key technology enabling the isolation of vehicle functions from each other along with the possibility to dynamically assign hardware resources. Introducing resilience to such a centralized automotive system requires the deployment of techniques and principles in all layers and components of the system, ranging from the vehicle itself, the connected IT infrastructure, road infrastructure and the communication to other vehicles.

Motivation. The increasing complexity towards autonomous driving combined with the interconnectedness of vehicles, e. g., vehicle-to-vehicle and vehicle-to-infrastructure communication, and the continuous development of functions require vehicles to react and adapt to changes and attacks independently. The automotive domain is distinct from other domains as it is a safety and real-time critical system operated by millions of individuals each day. Furthermore, security and safety techniques need to be aligned and extended with resilience techniques in order to strengthen vehicles' capabilities to withstand impending threats.

Contributions. This paper provides a framework to design resilient automotive systems. First, we systematically identify relevant automotive resilience techniques proposed in the literature with the goal to provide a full picture of available tools and techniques. We also organize these techniques into a

taxonomy, which comprises the categories of Detection, Mitigation, Recovery, and Endurance (REMIND). These categories represent high-level strategies that can help designers understand the *purpose* of each technique. Further, it can be beneficial to combine techniques from different strategies to achieve multiple layers of security. The selection of the right technique for the task at hand is further supported by associating the resilience techniques to the classes of *automotive assets* they are appropriate for. Additionally, we elaborate on the *trade-offs* (i.e., pros and cons) that are associated with each of the techniques, e.g., with respect to performance and other qualities. In summary, we provide a multi-dimensional decision support framework (built in a bottom-up fashion from the analysis of the literature) that can lead designers to the informed and optimal selection of a suitable set of resilience techniques to be implemented in an automotive system.

3.2 Methodology

By means of a systematic literature survey, we identify research papers that discuss techniques that are suitable to provide automotive resilience. We consider existing work related to resilience, fault tolerance and dependability. We also analyze the papers describing each technique to understand (i) the assets that can benefit from the technique, (ii) the risks that are mentioned as being mitigated by the techniques, and (iii) any pros/cons associated with the use of such technique.

We identified relevant research papers by searching the Scopus database [1]. A search string was intended to find relevant publications that carried out a review of suitable techniques. Therefore, we formulated the search string to *include* survey or literature review, and relevant topics, such as resilience, survivability, attack recovery, error handling or fault tolerance, as well as the keywords software, system or network. We *excluded* the keywords FPGA, memory, wireless, SDN and hardware to limit the search result to publications focusing on system architecture, software design or physical networks. Furthermore, we considered only publications written in English and published after 2010 in the areas of computer science and engineering. We manually screened the 200 most relevant publications returned by Scopus and found eight additional research publications, which were added to our result set. Ultimately, we retained and analyzed 12 publications which are shown in Table 3.1.

3.3 Attack Model and Assets

The four *strategies* in the REMIND framework are, as shown in Figure 3.1, further refined in *patterns* and *techniques*. A collection of these techniques specific for automotive systems is described in Section 3.4 and has been identified based on existing research in other domains and areas (see Table 3.1). We additionally describe the trade-offs of these techniques in Appendix 3.7 and point to relevant publications in Appendix 3.8. In the remainder of this section we describe the assets, security threats and attacks of automotive systems.

[1] https://www.scopus.com/

Table 3.1: Publications that provide an overview or collection of relevant techniques.

Discipline	Existing Work	Domain
Resilience	Chang2015 [34]	Fog Computing
	Hukerikar2017 [35]	High Performance Computing
	NIST 800-160v2 [36]	Systems Engineering
	Ratasich2019 [37]	Cyber-Physical Systems
	Sterbenz2010 [32, 38]	Networks
Security	Segovia2019 [39]	SCADA systems
Dependability	Bakhshi2019 [40]	Fog Computing
Fault Tolerance	Egwutuoha2013 [41]	High Performance Computing
	Kumari2018 [42]	Cloud Computing
	Mukwevho2018 [43]	Cloud Computing
	Sltten2013 [44]	Software Engineering
	Wanner2012 [45]	Vehicle Controller

We consider four asset types, namely *Hardware*, *Software*, *Network/Communication* and *Data Storage*. The attacker aims to compromise these assets via various attack vectors, whereas the defender, i.e. the vehicle, aims to cope with these attacks via resilience techniques. We consider skilled attackers as well as novice hackers (e. g., script kiddies) and further give examples from an asset, threat and attacker perspective.

Hardware. Can be broken down to *ECUs*, *Sensors* and *Actuators*. An *ECU* can vary in complexity depending on its objective, from a specific limited task to a multitude of tasks. The former can relate to the processing of a sensor signal and the latter an infotainment-system with lots of applications. *Sensors* can give information about speed, temperature and obstacle distance and identification where the *Actuators* turn input from these sensors (via an ECU) into actions, such as braking, steering and engine control.

Attack example. Tampering with existing hardware or installing malicious hardware into the vehicle can act as mediators to gain complete vehicle control. Input signals from sensors may be manipulated to cause an unwanted behavior.

Software. Can be *in transit*, *at rest* or *running*. *In transit* can relate to software provisioning systems, such as over-the-air or workshop updates and the latter two to software installed or running in ECUs.

Attack example. Software vulnerabilities might be exploited, e. g., via a privilege escalation attack which enables ECUs to be re-programmed with additional functionalities, such as adding remote access to the system.

Network/Communication. Can be broken down to *internal* and *external communication*. Examples for internal communication are CAN, FlexRay, LIN, MOST and Automotive Ethernet and for external communication Wi-Fi, Bluetooth, and V2X as well as external interfaces such as OBD-II, debug ports (e. g., JTAG) and CD player.

Attack example. The attacker can try to inject malicious data, through a device connected to an in-vehicle bus affecting the internal communication. Furthermore, modification of V2X data from other vehicles as well as malicious roadside units (e. g., vehicle positioning or traffic condition data) could affect system functions.

Data Storage. Can potentially be sensitive data, such as cryptographic keys, forensics logs, system information (e. g., from software libraries, OS and applications) and reports about the vehicle and the driver.
Attack example. The attacker can exploit secret keys used for sensitive diagnostics to disable firewalls. Logs and report data might be manipulated or removed to hide forensic evidence of the crime. Furthermore, information about the system can reveal vulnerabilities which might be exploited. Attackers typically exploit the above-mentioned assets in any order to achieve their goal, e. g., uploading malicious software to the vehicle by first compromising the cryptographic keys to get access to the memory and consequently upload a modified firmware containing malicious code. This can give elevated privileges and extended functionality which could cause inconsistencies or disruption of the system.

More examples of assets and related security threats and attacks can be found in Table 3.2.

Table 3.2: Automotive assets and related security threats and attacks

Asset	**Asset Examples**	**Security Threat**	**Attack Examples**
Hardware	ECU (hardware) Sensors Actuators	Disruption or direct intervention. Availability and Integrity.	*Fault Injection:* fuzzing, DoS, microprobing, malicious hardware as well as environmental injections (e. g., voltage and temperature) can disrupt or disable components or system resources. *Information Leakage:* side channel parameters, such as timing information or power consumption (e. g., differential power analysis) to extract secret keys.
Software	ECU (software) Libraries OS Virtualization	Manipulation of software, measurements or control signals. Availability and Integrity.	*Malware/Manipulated software:* indirectly affecting storage through alteration, deletion or blocking data, or indirect affecting the communication by read, manipulate or replay of messages, hence causing disruption and deviations from normal system operation.
Network/ Communication	CAN LIN MOST FlexRay Automotive Ethernet Mobile Network Wi-Fi Bluetooth OBD-II CD player	Communication failure or protocol vulnerabilities. Confidentiality, Integrity, Availability and Privacy.	*Fabrication/Jamming attack:* introducing fake traffic, e. g., sending high priority messages, to block legitimate low priority messages. *Masquerading/Spoofing attack:* masquerading as a legitimate node, e. g., by suspending the authentic ECU and send fabricated messages which seems to origin from the same. *Collision:* spoofing a message to induce a bit error/collision and then potentially spoof additional messages which get accepted. *Eavesdropping/hijacking:* intercept to read, block, manipulate or replay messages. *Suspension/DoS attack:* disable an ECU, such as inducing programming mode causing an ECU to not transmit or relay messages, potentially causing other ECUs to malfunction.
Data Storage	User Data Logs/Reports/Events Checkpoints Backups Forensics data Cryptographic material	Malicious handling of data storage. Confidentiality, Integrity, Availability and Privacy.	*Unauthorized read:* acquire sensitive data, such as privacy related user data e. g., previous locations or driving behavior. *Manipulation*: malicious alteration of data, e. g., replacing the software validation key enables potential alteration of memory data. *Removal:* data deletion of sensitive information, such as forensics data. *Reverse engineering:* extraction and analysis of firmware to deduce design features, vulnerabilities or secret keys.

3.4 REMIND Automotive Resilience Framework

We have developed the REMIND framework shown in Figure 3.1 to provide system designers and developers with a categorization of suitable resilience mechanisms including the identification of the assets they protect. The structure of the layers is chosen similarly to the work in Hukerikar et al. [35], where the bottom layer is divided into *strategies* and the mid layer is split into *patterns* that provide more details about the way the strategies can be realized. We refer to relevant solutions for automotive systems in the top layer and further link to the survey papers and reviews that identify specific *techniques* for their domain in the description listed below.

The four REMIND strategies for providing resilience for vehicular systems are:

- **Detection.** Faults, attacks and other anomalies need to be detected by the system in order to take reactive measures to avoid a failure.
- **Mitigation.** Once an anomaly is detected and located, mitigation techniques need to be triggered to keep the system operational. These techniques may result in a non-optimal system state.
- **Recovery.** Transitioning back to the desired, i. e., optimum state, is the aim of recovery.
- **Endurance.** The focus is set on lasting resilience in contrast to recovery & mitigation strategies which aim at taking immediate measures.

The remaining part of this section details the strategies and describes the patterns and corresponding techniques.

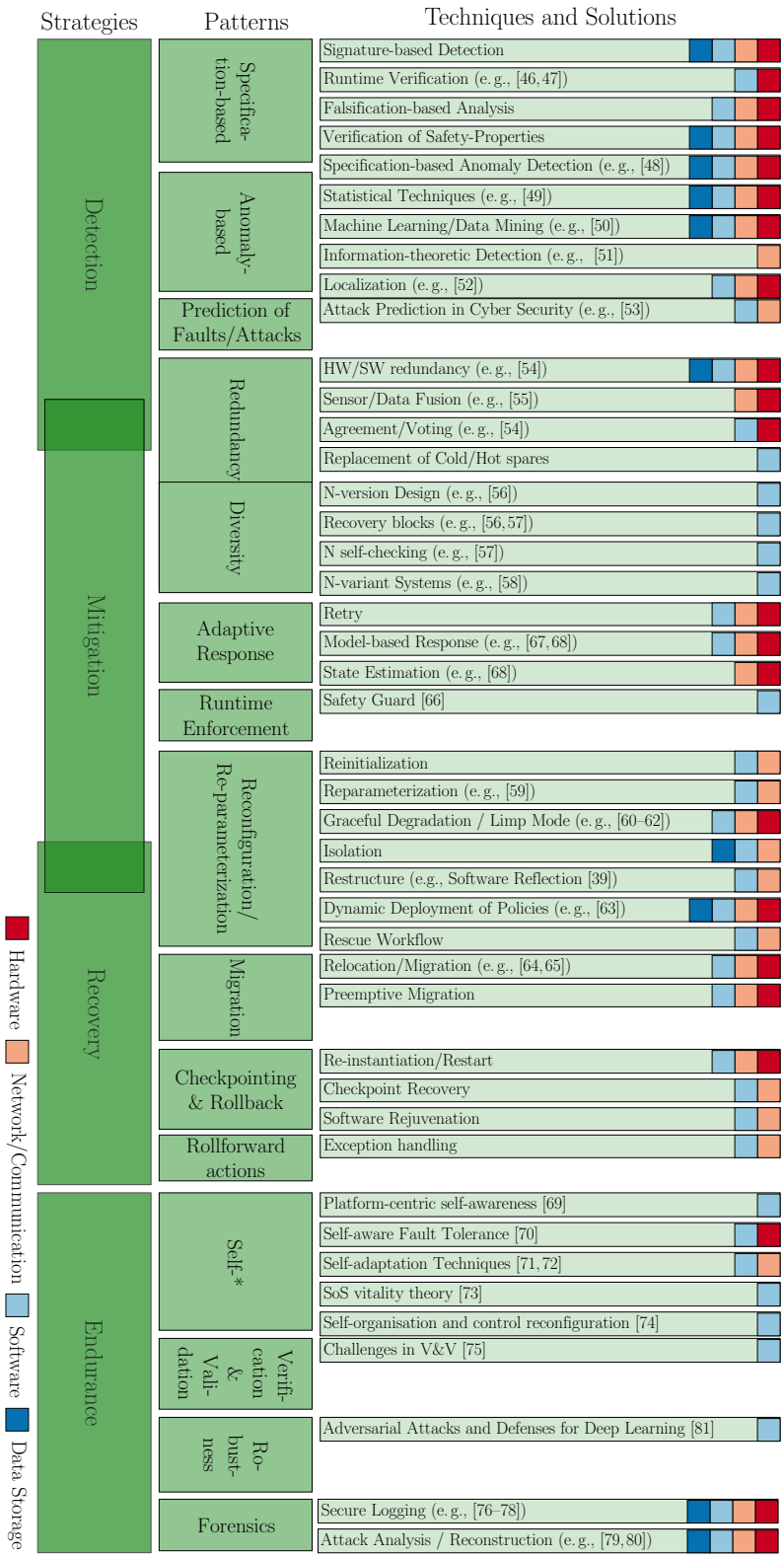

Figure 3.1: REMIND resilience techniques and solutions including a mapping to the assets for each technique. The overlap of the *Mitigation* strategy highlights that some patterns also contribute to *Detection* respectively *Recovery*.

3.4.1 Detection

The monitoring and detection capabilities of a system can be limited due to various factors, such as computational resources, energy consumption, and the complexity of functions and network architecture. The move to a more centralized architecture, however, paves the way for more extensive monitoring.

Specification-based Detection. Malicious or abnormal behavior is detected using a specification that describes the behavior of signals or communication patterns. Domain knowledge is needed to create the specifications.

– *Signature-based Detection [37].* Signatures are constructed to describe known attack behavior. By design, these techniques suffer from detecting new attacks and zero-day vulnerabilities. However, they typically achieve a low false positive rate [48].

– *Runtime Verification [37, 44].* A monitor observes the system at runtime to verify the correctness of the execution. Formal specification languages, e. g., Signal Temporal Logic (STL) [82], have been developed to describe the normal system behavior which is matched against a trace during execution.

– *Falsification-based Analysis [37].* It extends STL by including a quantitative semantics allowing the return of real values rather than Boolean values.

– *Verification of Safety-Properties [37].* The formal verification of safety properties has become increasingly complex due to the added functionality in modern vehicles. Exhaustive verification techniques, as listed and argued by Ratasich et al. [37], are currently limited to small scale models.

– *Specification-based Anomaly Detection.* Normal behavior, according to a set of rules, is defined using this technique. An alert is sent when a violation of these rules is detected [48].

Anomaly-based Detection. Anomaly- or behavior-based detection techniques are based on comparing behavior with a model of normal behavior. Alerts are raised when a deviation is detected [83].

– *Statistical Techniques [37].* A statistical model describing the system or a specific process is designed in order to detect anomalies. Events are considered anomalies when the probability of their occurrence is below a certain threshold according to the model.

– *Machine Learning/Data Mining [37].* These techniques typically do not require domain knowledge. A model, such as Bayesian networks, neural networks and support vector machines, learns through training data how to classify observations in normal and abnormal classes.

– *Information-theoretic Detection [37].* The entropy of information can be used to detect anomalies, as a change of the entropy above a certain threshold may be caused by an attack, e. g., masquerading attack [37, 51].

– *Localization.* Finding the source of the attack may be required to take appropriate actions. Network-based Intrusion Detection Systems (IDSs) can be used to limit the location to a specific subnet, however, solutions identifying the particular ECU are needed (e. g., [52]).

Prediction of Faults and Attacks. First, the system needs to identify the presence of an attacker. The next actions are *attack projection* and *attack intention recognition* which aim at identifying the next steps and the ultimate goal of the attacker. *Attack or intrusion prediction* can be used to foresee when and where an attack will take place [53].

Adversaries mounting simpler attacks on a single vehicle, such as DoS attacks on the CAN bus, may be difficult to predict as the attack consists of fewer steps. However, large-scale attacks requiring the attacker to go through several stages may be predicted by this technique.

Redundancy. Redundancy is twofold, as it can support both detection and mitigation. It is important to highlight that purely redundant systems suffer from the same design faults and vulnerabilities. Thus, diversity is combined with redundancy to overcome this issue.

– *HW/SW Redundancy [35–37, 39, 41–44].* Redundancy combined with a voter allows to mask system failures. The voter compares the results of a number of independently executed software and/or hardware modules and selects, for instance, the majority [54]. Repeating the computation n times on the same hardware can be used to detect random faults.

– *Sensor/Data Fusion [37].* Data from different origins may be fused to compensate inaccuracies or temporary sensor failures. Sensor fusion, e. g., extended Kalman filter [84] and particle filter [55], can be used to describe the non-linear relationship between sensors. For example, the motion of a vehicle can be described with measurements from the wheel speed sensor, GPS location and data received from other vehicles.

– *Agreement/Voting [35, 37, 41, 44].* Redundant components are required for this technique. Voting can be realized in two ways, i. e., exact voting and inexact voting, where the latter allows a variation of the result within a certain range [54].

– *N-version Design [35–37, 41, 43].* N versions of a software with the same requirements are developed by N independent teams resulting in a diverse set of functionally equivalent software components that fulfill the same specification. These versions are executed concurrently and a voter decides based on the majority or calculates, for instance, the median or average of the results [56].

– *Recovery Blocks [35, 43, 44].* Similar to n-version design, n versions of a software component exist; however, only one version is executed at a time. After the active version is executed, a common acceptance test decides whether the result is accepted. In case the result is rejected, the subsequent version is executed and evaluated [56, 57].

– *N self-checking [41].* This technique is a combination of n-version design and recovery blocks. It requires at least two diverse versions with their own acceptance test. When the active component fails its acceptance test, the subsequent component takes over [57].

– *N-variant Systems.* Multi-variant execution automatically diversifies software and monitors the output of at least two variants to detect and mitigate attacks [58].

– *Replacement of Cold/Hot Spares [37].* Concurrent and sequential execution of redundant software components is costly in terms of energy consumption and computational resources. Therefore, the introduction of cold or hot spares, such as in N self-checking, have been found to be a viable alternative [37].

3.4.2 Mitigation

After detecting an attack or anomaly, the system needs to react to reduce the impact of the attack. Some mitigation techniques may require the transition

to a non-optimal state.

Adaptive Response. We focus on techniques that adapt the response of a function or sub-system in order to maintain its intended functionality.
– *Retry [42, 43].* Performing the same computation with new measurements if the first computation resulted in an undesired system state or in an error. Retry can mitigate a replay attack.
– *Model-based Response and State Estimation [39, 45].* System models, e. g., Kalman filter for state estimation [84, 85], or parameter estimation techniques, like regression analysis, are not only a temporary solution to mitigate attacks, such as replay and masquerading attacks, they can also be used to alert the system and log important information for forensics [68].

Runtime Enforcement. Runtime enforcement is an extension of runtime verification where the system also reacts to violations [46].

Reconfiguration and Reparameterization. The system protects itself by adapting parameters when an attack is detected. We distinguish between reconfiguration and migration in the way that migration focuses on relocating functionality whereas reconfiguration changes system or application parameters.
– *Reinitialization [35].* Temporary faults and attacks can be addressed with this technique. However, permanent faults or reoccurring attacks cannot be mitigated by restoring the system or a function to its initial state. Reinitialization can be seen as checkpoint recovery with the checkpoint being the initial state of the system or function.
– *Reparameterization [37].* Is similar to reinitialization, however, the system configuration is dynamically adjusted to the situation. As Ratasich et al. [37] point out, reparameterization typically results in a non-optimal state.
– *Graceful Degradation / Limp Mode [37, 39].* Given the extended automated driving functions of future vehicles, it is of utmost importance to implement more sophisticated solutions that ensure the passengers safety when key components in the vehicle fail or are subject to attacks. These techniques are similar to reparameterization, but focus on safety and should be seen as a last resort. Modern vehicles already have a so-called limp mode implemented, which is triggered when the vehicle detects major technical problems [86].
– *Isolation [35,37].* Restricting access or completely isolating system components in the presence of an error or intrusion can limit the impact on the entire system and its performance.
– *Restructure [35].* Restructuring components within a sub-system aims at providing resilience through reconfiguration of affected components. Segovia et al. [39] explore software reflection as means to mitigate attacks.
– *Dynamic Deployment of Policies [39].* Security or other policies can be applied dynamically based on the type of attack, e. g., DoS or masquerading, that is detected.
– *Rescue Workflow [42,43].* A workflow can be used to describe tasks with their dependencies to each other. The idea behind rescue workflows is to dynamically adjust the structure of the workflow when an error or intrusion affects a specific task. Existing cloud solutions may need to be adapted for automotive systems.

3.4.3 Recovery

Recovery techniques intend to bring the system back to an optimal state.

Migration. These techniques are mainly originating from high performance computing and cloud systems. As future automotive systems move towards a centralized architecture, virtualization and service-oriented architectures are becoming more relevant.

– *Relocation/Migration [37,43].* Virtualization such as hypervisor and container-based solutions allow a fast migration and relocalization to other nodes in the vehicular network.

– *Preemptive Migration [42,43].* Continuous monitoring and analysis of the system can be used to relocate software functions or services before a fault occurs.

Checkpointing & Rollback. A checkpoint or snapshot describes the system state at a specific point in time. By design, recovery does not prevent the same attacks from happening again.

– *Re-instantiation/Restart [35, 37, 41, 43].* When an intrusion is detected, the affected component can be re-instantiated or restarted to recover to a known, error and attack free, state. This technique can be combined with reparameterization to avoid the same anomaly to happen again [37].

– *Checkpoint Recovery [35, 41–44].* Snapshots can be created in two ways: checkpoint-based and log-based. Egwutuoha et al. [41] highlight the complexity of taking checkpoints in a distributed system, as these checkpoints need to be consistent.

– *Software Rejuvenation [35,43].* This technique carries out periodic restarts or reinitializations of the system to maintain a known, error-free state.

Rollforward actions. These techniques aim at bringing the system to a stable state immediately before the error or attack was detected. As in rollback, the recovery is based on using checkpoint-based or log-based recovery [35].

– *Exception Handling [35].* From a model-driven engineering view, *Rollforward* can be performed using exception handling. Sltten et al. [44] highlight that this solution can be only applied to anticipated events.

3.4.4 Endurance

Resilience needs to be ensured over the entire lifetime of a vehicle. The preceding techniques center around providing immediate response when anomalies are detected.

Self-*. Self-* or self-X techniques cover solutions and research directions focusing on how to introduce autonomy into the system. This pattern is especially important for future vehicles as the environment is and will change frequently, new vulnerabilities will be found, new attempts to attack vehicles and their infrastructure will be developed, and new technologies will appear. Also, considering the lifetime of cars, which is around 10–15 years, it is evident that automotive systems need to adapt to a certain extent autonomously.

Verification and Validation. Due to the increasing functionality and interconnectedness of modern vehicles it is required to update software components via over-the-air updates in order to fix vulnerabilities and bugs or upgrade vehicle functions. This is especially challenging as each vehicle model can be further configured, resulting in a manifold of possible vehicle configurations.

Robustness. Artificial intelligence, especially machine learning, is a key technology for autonomous driving and decision making, as the system needs

to be able to handle previously unseen situations [37].

Forensics. Providing evidence of intrusions even after a crash is important for taking appropriate countermeasures.

– *Secure Logging.* Hoppe et al. [76] express the need for forensic solutions in vehicles. Non-safety-critical events, such as updates, component failures and other malfunctions, need to be logged and stored securely for a prospective analysis. The authors also discuss in great detail which information and how this information can be stored in vehicles.

– *Attack Analysis.* Nilsson and Larson [79] specify requirements for forensic analyses of the in-vehicle network. It is also important to analyze attacks disclosed by researchers, such as Checkoway et al. [87] and Miller and Valasek [88], as well as attacks logged by the vehicle manufacturers in order to take appropriate actions.

3.5 Related Work

Making vehicles safe and secure has traditionally been the main focus in research. For instance, methods to combine safety and security [89] and how to assess an automotive system and/or derive security requirements and mechanisms have been proposed [18, 90, 91]. Le et al. [92] provide a survey on security and privacy in automotive systems and further provide an overview of suitable security mechanisms.

One of the first structured collections of principles for cyber resilience is the Cyber Resiliency Engineering Framework [93] by MITRE in 2011 which got further incorporated in NIST SP 800-160v2 [36]. Other work describing principles for resilience have been either concentrating on other domains, i. e., high performance computing, cyber-physical systems, or networks, or they focused particularly on dependability or fault tolerance. Table 3.1 provides an overview of relevant publications, which provide a comprehensive overview or collection of techniques, and categorizes them according to their discipline and the area they are focusing on.

The reviewed publications classify the identified techniques in different ways. Hukerikar et al. [35] divide them into strategies, i. e., fault treatment, recovery, and compensation, whereas Ratasich et al. [37] organize them according to their ability, i. e., detection and diagnosis, recovery or mitigation, and long-term dependability and security. Work focusing on fault tolerance either split the identified techniques in reactive and proactive measures [42, 43] or classify them according to their ability, e. g., error handling and recovery [41, 44].

With the developed REMIND framework, we contribute to supporting the resilience of automotive systems by: (i) identifying techniques for attack detection, mitigation, recovery, and resilience endurance; (ii) organizing the techniques into a taxonomy to guide designers when selecting resilience techniques; (iii) providing guidelines on how the REMIND framework can be used against common security threats and attacks; and (iv) discussing the trade-offs when applying the techniques that are highlighted in this framework.

In addition to the identified techniques in Figure 3.1, we point to implementations relevant for or specific to the automotive domain in Appendix 3.8.

3.6 Conclusion

The reviewed work shows the current research efforts towards making systems resilient to attacks and faults in related domains. We present a novel structure for categorizing resilience techniques in the form of the REMIND framework with the aim to lead designers in making informed decisions when choosing resilience techniques. We build upon the existing work and set the focus on the limitations of automotive systems and their challenges. The REMIND techniques have been chosen considering automotive assets and related attacks which are described in Section 3.3 and further linked to the guidelines and trade-off analysis in Appendix 3.7.

Future work includes the validation of the REMIND framework in regard to studying its applicability in industry in more depth. Furthermore, specific solutions for the identified techniques that consider the unique properties of automotive vehicles can be explored. Especially, the role of software-defined networking and its contribution to resilience can be investigated.

3.7 REMIND Resilience Guidelines

In this section, we report in Table 3.3 resilience techniques that can be used against common security threats and attacks. We also describe the trade-offs when implementing these techniques.

Table 3.3: REMIND Resilience Guidelines

Asset Hardware	**Attack** Fault Injection	
Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
DETECTION		
• Statistical Techniques [37] • Machine Learning/Data Mining [37] • Localization (e. g., [52]) • Sensor/Data Fusion [37]	• Less computation is required. • No domain knowledge is needed. It handles multivariate and non-linear data. • Identifies the exclusive part causing the fault or attack. • Calculates a value of trust of the data sources derived from the normalization factor.	• Very sensitive to outliers, imprecise detection, and increased complexity when modelling non-linear data. • Requires training. Imprecise prediction: false positives and false negatives. Time penalty and resource consumption (power, processing, and storage). • Often applied offline. The precision of the localization is dependent on both, the number of observed parameters and the set frequency for probing monitored resources. • Imprecise detection: false positives and negatives. It also introduces time penalty (increase in execution time) and space penalty (increase in resource usage).
MITIGATION		
• Hardware Redundancy [35–37, 39, 41–44]	• Enables offsetting the effects of faults and attacks, and allows the progress of the system without loss of functionality.	• Time penalty (increase in execution time) and resource consumption (increase in required resources). Hardware costs independent of whether attacks occur. Also, the design and verification of replicas requires an effort.
RECOVERY		
• Relocation/Migration [37, 43]	• Maintain system functionality in an operational state as it was before the fault or attack.	• May cause a degraded system, with less functionality, resources, and performance.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
ENDURANCE		
• Self-aware Fault Tolerance [70]	• Enables systems to adapt their behavior when a fault or attack occurs in their environment, thus allowing a continuous operation of these systems.	• Complexity and resource consumption.
Asset	**Attack**	
Software	Malware/Manipulated Software	
DETECTION		
• Signature-based Detection [37] • Runtime Verification [37, 44]	• A precisely calibrated signature effectively identifies abnormal events during software execution. • Well-established and efficient technique to verify the correctness of software execution and monitor the behavior of the system.	• Does not work when designers and 3rd party suppliers (e.g., intellectual property providers) are not trusted. It cannot handle zero-day attacks and, thus, often used in combination with anomaly-based techniques leading to an increased resource consumption and time penalty. • Limited coverage. The used monitoring algorithms usually handle a single execution trace which limits the scope of the verification.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
MITIGATION		
• Software Redundancy [35–37, 39, 41–44] • N-Version Design [35–37, 41, 43] • Agreement/Voting [35, 37, 41, 44] • Recovery Blocks [35, 43, 44] • N self-checking [41]	• Helps to contain and exclude malicious behavior (i.e., reduces likelihood of harm). Enable restoration in case of disruption. Enhances the availability of critical capabilities. • Helps to mitigate the impact of failures when a risk is introduced to system design or configuration. • Typically combined with redundancy. Can be used to select, for instance, the average or median of the results provided by the redundant sources. • Uses different implementations of the same design specification to provide tolerance of design faults. • Provides mitigation by creating N versions of the same software, each with its own acceptance test. The version that passes its own acceptance test is selected through an acceptance voting system.	• Resource consumption. It demands the protection of redundant resources. It can degrade over time as configurations are updated or connectivity changes. It is often applied with diversity techniques which increases complexity and leads to scalability issues. • Requires much effort for designing, implementation, testing, and validation of the N independent versions. • Attackers may exploit the voting process in order to force the system to a degraded mode. • Requires extra verification and validation effort and, thus, more resource consumption. It might be difficult to create alternative software implementations without any correlation between the various versions. • Causes an increase in required resources and execution time.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
RECOVERY		
• Preemptive Migration [42, 43] • Checkpoint Recovery [35, 41–44] • Software Rejuvenation [35, 43]	• Prevents failures from impacting running parallel applications by enabling the migration of running software from one virtual machine to another in real time. • Helps the system to resume its operation in a state free of the effects of the fault or attack. Frequent checkpointing reduces the amount of lost work. • Helps avoiding the costs of failures from software degradation, as periodic (graceful) restarts of the software component allow the release and re-allocation of memory, thus, operation in a clean state.	• Lack of standardized metrics for measuring and evaluating the health and interfaces between system components. • Overhead in relation to the size and frequency of created checkpoints. Creating a checkpoint, for instance, requires interrupting the normal operation of a system to record the checkpoint. Moreover, it requires storage resources to store the checkpoint. The created checkpoints might potentially contain an error or intrusion that has not been detected yet. Globally consistent checkpoints are not trivial to obtain in a distributed system, due to e. g., variation of the local clock, parallel computation and possible different system states. • Requires shutting the software down and restarting it periodically which causes the software to be unavailable for the duration of the restart. It is often a slow process requiring an extra overhead.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy / Technique	**Trade-off**	
	Pros	Cons
ENDURANCE		
• Platform-centric Self-aware-ness [69] • Secure Logging (e. g., [76–78])	• Enables systems to recognize their own state and to continuously adapt to change, evolution, system interference, environment dynamics, and uncertainty. It optimizes resilience, quality of service, and supports system dynamics and openness. It also helps to reduce uncertainties and identify inconsistencies. • Prevents modifying the logs by using e.g., chained hashes. It enables storing security-related events containing information about e.g., flash operations, external interactions, and power downtime. This information helps to reconstruct events, detect intrusions and identify problems.	• Automatically maintaining coherent specifications that capture and monitor security is a challenging task. Complexity, scalability, and difficulty in dealing with uncertainties and inaccuracies. The determination of relevant dependencies in a complex system is also challenging. • Resource consumption and time penalty. Moreover, missing authentication and lack of cryptographic means to ensure data integrity can limit the potential of the logging.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
Asset Network/Communication	**Attack** Fabrication/Jamming	
DETECTION		
• Specification-based Anomaly Detection (e. g., [48]) ◦ Localization (e. g., [52]) • Verification of Safety-Proper-ties [37]	• Helps detecting anomalies in the system's behavior by reporting the specific deviation that has been observed. ◦ Identifies the exclusive part causing the fault or attack. • Ensures that the system does not evolve in unsafe state starting from some initial conditions.	• Needs of resources for detection and processing of collected information (e.g., costly intelligent sensors). Domain knowledge is required to specify normal behavior. Specifications need to be adapted for each specific vehicle configuration otherwise risk of high false positives or negatives. ◦ Requires additional resources. • It is limited to small scale systems.
MITIGATION		
• Isolation [35, 37] ◦ Restructure [35]	• It provides a remedy to enable the system to continue its operation by offsetting the effect of the attack. Also, it prevents loss of functionality. ◦ Helps to mitigate incorrectness in the interactions between the components or subsystems by excluding the affected part from interacting with the rest of the system, and maintaining system functionality.	• Introduces a time penalty and an increase in required resources (e.g., replica modules that are used to compensate for isolating the affected component of the system). ◦ May cause an operation of the system in a degraded condition which influences its performance and incurs additional time overhead to the system.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
RECOVERY		
• Relocation/Migration [37, 43] • Re-instantiation/Restart [35, 37, 41, 43]	• Maintain system functionality in an operational state as it was before the fault or attack. • Helps to restore the system to its initial state when the impact of the attack can not be handled in another manner. It guarantees that the impact of the attack is completely removed.	• May cause a degradation in the operation of the system which influences the performance and functionality thereof. • Restoring the system to its initial state causes lost data, such as privacy related data (e. g., location, speed, driving behavior) and workshop data (e. g., vehicle health, engine data and emissions). The impact of the lost data depends on the type of data and the current need for it. In addition, the re-instantiation of safety-critical functions may require the vehicle to be in standstill.
ENDURANCE		
• Self-adaptation [71, 72]	• Ensures a secure, reliable, and predictable communication between system components and between the system and its environment. Supports and maintains an acceptable level of service despite the occurrence of faults and other factors that affect normal operations. Seamlessly adapts to different network loads and reacts to security threats and other disturbances in the environment.	• Complexity and resource consumption.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
Asset Network/Communication	**Attack** Masquerading/Spoofing/Collision	
DETECTION • Information-theoretic Detection [37] • Falsification-based Analysis [37]	• Helps to detect anomalies by analyzing available audit logs and records (e.g., entropy measures) and comparing these records with defined normal behaviors. More records enhance the precision of the detection. • Provides an indication (i.e., a robustness degree) to what extent temporal logic properties are from satisfying or violating a specification.	• Time penalty for processing audit records. More records at disposal increases the processing time and complexity. On the other hand, a low number of records leads to an imprecise detection with more false positives and false negatives. • Imprecise detection: false positives and false negatives
MITIGATION • Rescue Workflow [42, 43] (adaptation may be necessary) • Dynamic Deployment of Policies [39]	• Enables the system to continue operation after the failure of the task until it is unable to proceed without amending the fault or attack. Already finished tasks do not need re-execution, thus saving time and resources. • Takes the dynamic and changing nature of attacks into account. Deploys different defense policies depending on the attack, for example, it can modify the executed actions while the attack is going on.	• It may lead to a decrease in the quality of service. Time penalty might be caused by recomputing and migrating the tasks which cause the problem. • Leads to performance overhead. Moreover, it always requires runtime permissions which may not be present when running normally. Complexity.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
RECOVERY		
• Checkpoint Recovery [35, 41–44] • Re-instantiation/Restart [35, 37, 41, 43]	• Helps the system to resume its operation in a state free of the effects of the fault or attack. Frequent checkpointing reduces the amount of lost work. • Helps to restore the system to its initial state when the impact of the attack can not be handled in another manner. It guarantees that the impact of the attack is completely removed.	• Overhead in relation to the size and frequency of created checkpoints. Creating a checkpoint, for instance, requires interrupting the normal operation of a system to record the checkpoint. Moreover, it requires storage resources to store the checkpoint. The created checkpoints might potentially contain an error or intrusion that has not been detected yet. Globally consistent checkpoints are not trivial to obtain in a distributed system, due to e. g., variation of the local clock, parallel computation and possible different system states. • Restoring the system to its initial state causes lost data, such as privacy related data (e. g., location, speed, driving behavior) and workshop data (e. g., vehicle health, engine data and emissions). The impact of the lost data depends on the type of data and the current need for it. In addition, the re-instantiation of safety-critical functions may require the vehicle to be in standstill.
ENDURANCE		
• Secure Logging (e. g., [76–78])	• Prevents modifying the logs by using e.g., chained hashes. It enables storing security-related events containing information about e.g., flash operations, external interactions, and power downtime. This information helps to reconstruct events, detect intrusions and identify problems.	• Resource consumption and time penalty. Moreover, it requires authentication and cryptographic means to ensure data integrity and confidentiality.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
Asset Network/Communication	**Attack** Hijacking/Replay/Suspension/DoS	
DETECTION • Signature-based Detection [37] • Verification of Safety-Proper-ties [37]	• A precisely calibrated signature effectively identifies abnormal events during software execution. • Ensures that the system does not evolve in unsafe state starting from some initial conditions.	• Does not work when designers and intellectual property providers are not trusted. It cannot handle zero-day attacks and, thus, often used with Anomaly-based techniques leading to a increased resource consumption and time penalty. • It is limited to small scale systems.
MITIGATION • Reparameterization [37] • Isolation [35, 37] • Graceful Degradation [37, 39]	• Enables adaptation by switching the configuration parameters of the compromised component to another configuration. • It provides a remedy to enable the system to continue its operation by offsetting the effect of the attack. Also, it prevents loss of functionality. • Prevents a catastrophic failure of the system. It enables a system to continue functioning even after parts of the system have been compromised. It shuts down less critical functions to allocate the resources to more critical functions to maintain availability.	• Decreases the quality of service. • Introduces a time penalty and an increase in required resources (e.g., replica modules that are used to compensate for isolating the affected component of the system). • Causes a degradation in the performance of the operations and services of the system.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy / Technique	**Trade-off**	
	Pros	Cons
RECOVERY		
• Relocation/Migration [37, 43] • Software Rejuvenation [35, 43] • Reinitialization [35]	• Maintain system functionality in an operational state as it was before the fault or attack. • Helps avoiding the costs of failures from software degradation, as periodic (graceful) restarts of the software component allow the release and re-allocation of memory, thus, operation in a clean state. • Applied in conditions in which the mitigation is deemed impossible. Restores or pristine resets the system to its initial state.	• May cause an operation of the system in a degraded condition which influences its performance. • Requires shutting the software down and restarting it periodically which causes the software to be unavailable for the duration of the restart. It is often a slow process requiring an extra overhead. • Causes loss of work, and accordingly leads to a waste of resources.
ENDURANCE		
• Attack Analysis / Reconstruction (e. g., [79, 80])	• Helps to enhance resilience by systematically and empirically analyzing attacks as well as used technologies (potential entry point, e.g., Bluetooth and WiFi) that interact with the external environment.	• Resource consumption and analysis effort.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
Asset Data Storage	**Attack** Unauthorized Read/Manipulation	
DETECTION		
• Signature-based Detection [37] • Specification-based Anomaly Detection (e. g., [48])	• A precisely calibrated signature effectively identifies abnormal events during software execution. • Helps detecting anomalies in the system's behavior by reporting the specific deviation that has been observed.	• Does not work when designers and intellectual property providers are not trusted. It cannot handle zero-day attacks and, thus, often used with Anomaly-based techniques leading to a increased resource consumption and time penalty. • Needs of resources for detection and processing of collected information (e.g., costly intelligent sensors). Domain knowledge is required to specify normal behavior. Specifications need to be adapted for each specific vehicle configuration otherwise risk of high false positives or negatives.
MITIGATION		
• Redundancy [35–37,39,41–44] • Isolation [35, 37]	• It enables data backup and restore by replicating information and data sources. • It provides a remedy to enable the system to continue its operation by offsetting the effect of the attack. Also, it prevents loss of functionality.	• Requires extra resources for data storage. • Introduces a time penalty and an increase in required resources (e.g., replica modules that are used to compensate for isolating the affected component of the system).
RECOVERY		
• Dynamic Deployment of Policies [39]	• Takes the dynamic and changing nature of attacks into account. Deploys different defense policies depending on the attack, for example, it can modify the executed actions while the attack is going on.	• Leads to performance overhead. Requires runtime permissions which may not be present when running normally. Complexity.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
ENDURANCE • Secure Logging (e. g., [76]) • Attack Analysis / Reconstruction (e. g., [79, 80])	• Prevents modifying the logs by using e.g., chained hashes. It enables storing security-related events containing information about e.g., flash operations, external interactions, and power downtime. This information helps to reconstruct events, detect intrusions and identify problems. • Helps to enhance resilience by systematically and empirically analyzing attacks as well as used technologies (potential entry point, e.g., Bluetooth and WiFi) that interact with the external environment.	• Resource consumption and time penalty. Moreover, missing authentication and lack of cryptographic means to ensure data integrity can limit the potential of the logging. • resource consumption and analysis effort.

3.8 Proposed Automotive Solutions

In Table 3.4 we provide a description of the solutions referred to in Figure 3.1. This overview of specific solutions should be considered as a starting point for interested readers and is by no means complete.

Table 3.4: Techniques and solutions relevant for the automotive domain.

DETECTION

Pattern	**Technique**	**Solution**
Specification-based	Runtime Verification	Heffernan et al. [47] use the automotive functional safety standard ISO 26262 as a guide to derive logical formulae. They demonstrate the feasibility of their proposed runtime verification monitor with an automotive gearbox control system as use case.
	Specification-based Anomaly Detection	Müter et al. [48] describe eight detection sensors that are applicable for the internal network of automotive systems. Six of these sensors are specification-based, e. g., the frequency of specific message types and the range of transmitted values like speed.

Table 3.4 – *Continued on next page*

Table 3.4 – *Continued from previous page*

Pattern	Technique	Solution
Anomaly-Based	Statistical Techniques	Nowdehi et al. [49] propose an IDS that learns about the automotive system by learning from samples of normal traffic without requiring a model definition.
	Machine Learning	Hanselmann et al. [50] propose CANet an unsupervised IDS for the automotive CAN bus. The anomaly score is calculated using the error between the reconstructed signal and the true signal value.
	Information-theoretic	Müter et al. [51] design an entropy-based IDSs for automotive systems with experimental results using data from a vehicle's CAN-Body network.
	Localization	Cho and Shin [52] present a scheme identifying the attacking ECU based on fingerprinting the voltage measurements on the CAN bus for each ECU. We see great opportunities in the localization of attacks when considering a centralized vehicle architecture combined with virtualisation techniques. This allows us to get detailed performance metrics of virtualized vehicle functions.
Predicting Faults and Attacks	Attack Prediction	Husk et al [53] perform a survey about current attack projection and prediction techniques in cybersecurity.
Redundancy	Diversity Techniques	Baudry and Monperrus provide in their survey [94] an overview of different software diversity techniques.
	Adaptive Software Diversity	Höller et al. [59] introduce an adaptive dynamic software diversity method. The diversification control receives error information from the decision mechanism and randomizes specific parameters during execution. Their experimental use cases demonstrate the dynamic reconfiguration of ASLR parameters, respectively, random memory gaps.
MITIGATION		
Adaptive Response	Model-based Response	Cómbita et al. provide a survey on response and reconfiguration techniques for cyber-physical control systems. Controllers or other systems that can be modelled as a control loop can be, for instance, adjusted to have another module in the feedback loop that compares the actual feedback from the control loop with a simulated/modelled response of what is expected.
Runtime Enforcement	Safety Guard	Wu et al. [66] show how so-called safety guards can be applied to safety-critical Cyber-Physical Systems (CPSs).

Table 3.4 – *Continued on next page*

Table 3.4 – *Continued from previous page*

Pattern	Technique	Solution
Reconfiguration and Reparametrisation	Graceful Degradation	Dagan et al. [60] provide an architectural design on how to extend limp modes so that they can be additionally used in a cyber security context. A safe-mode manager sends out triggering messages that cause the ECUs to transition to a limp mode when cyber-breaches are detected.
		Ishigooka et al. [61] propose a graceful degradation design process for autonomous vehicles with focus on safety.
		Reschkka et al. [62] explore how skills and ability graphs can be used for modelling, on-line monitoring and supporting decision making of driving functions.
	Restructure	Segovia et al. [39] set the focus of their survey on software reflection as mitigation technique for SCADA systems. Software reflection enables the system itself to examine and change its execution behaviour at runtime, which allows, for instance, the system to take actions when an attack is detected. The drawbacks currently seen in software reflection are the performance overhead, the increased execution time and the extended permissions required by software reflection.
	Dynamic Deployment of Policies	Rubio-Hernan et al. [63] propose an architecture for CPS that combines feedback control loops with programmable networking in order to mitigate attacks by re-routing traffic or applying security rules.
RECOVERY		
Migration	Relocation/Migration	Jiang et al. [64] propose a hypervisor that meets real-time requirements.
		Other relocation techniques are microservices [95]. Pekka and Mattila [65] propose a service-oriented architecture for real-time CPSs.
	Pre-emptive Migration	Engelmann et al. [96] describe a pre-emptive migration technique which uses a feedback-loop for observing health parameters to detect behaviour indicating a fault. This solution was developed for high performance computing and its applicability for the automotive domain needs to be further investigated.
Checkpointing and Rollback	Software Rejuvenation	Romangnoli et al. [97] describe a method to decide when it is safe to reload the software of a CPS.

Table 3.4 – *Continued on next page*

Table 3.4 – *Continued from previous page*

Pattern	Technique	Solution
ENDURANCE		
Self-*	Continuous Change	Möstl et al. [69] identify in their work the challenges of continuous change and evolution of CPS and propose two frameworks for self-aware systems centring around self-modelling, self-configuration and self-monitoring. The controlling concurrent change (CCC) framework is concerned with how to deal with changes in software components during the lifetime of a CPS. The authors highlight that the well-established V-model currently used is not designed for continuous change and therefore parts of the integration testing and system validation and verification need to be moved to the system itself. The proposed framework includes an automated integration process for new or updated functions that addresses safety, security, availability and real-time requirements. The structure and workflow of the proposed framework is further described using an automotive use case. The second framework concentrates on optimising performance, power consumption and resilience of CPS by using self-organisation and self-awareness techniques.
Verificaton & Validation	Challenges in V&V	De Lemos [75] discuss research challenges of verification and validation for self-adaptive systems at runtime.
Robustness	Adversarial Attacks on DNN	Yuan et al. [81] give an overview of current adversarial attack and defence techniques for deep learning.
Forensics	Secure Logging	Lee et al. [77] describe T-Box a secure logging solution for automotive systems that makes use of the trusted execution environment in ARM TrustZone.
		Mansor et al. [78] propose a framework to log vehicle data, such as diagnostic transmission codes, via the mobile phone and store it on a secure cloud storage.
	Attack Analysis / Reconstruction	Nilsson and Larson [79] discuss the requirements for conducting forensic investigations on the in-vehicle network.
		Bortles et al. [80] present which types of data may be retained from current infotainment and telematic systems.

Chapter 4

Resilient Shield: Reinforcing the Resilience of Vehicles Against Security Threats

Adapted version that appeared in IEEE 93rd Vehicular Technology Conference 2021

K. Strandberg, T. Rosenstatter, R. Jolak, N. Nowdehi, T. Olovsson

Abstract. Vehicles have become complex computer systems with multiple communication interfaces. In the future, vehicles will have even more connections to e.g., infrastructure, pedestrian smartphones, cloud, road-side-units and the Internet. External and physical interfaces, as well as internal communication buses have shown to have potential to be exploited for attack purposes. As a consequence, there is an increase in regulations which demand compliance with vehicle cyber resilience requirements. However, there is currently no clear guidance on how to comply with these regulations from a technical perspective. To address this issue, we have performed a comprehensive threat and risk analysis based on published attacks against vehicles from the past 10 years, from which we further derive necessary security and resilience techniques. The work is done using the *SPMT* methodology where we identify vital vehicle assets, threat actors, their motivations and objectives, and develop a comprehensive *threat model*. Moreover, we develop a comprehensive *attack model* by analyzing the identified threats and attacks. These attacks are filtered and categorized based on attack type, probability, and consequence criteria. Additionally, we perform an exhaustive mapping between asset, attack, threat actor, threat category, and required mitigation mechanism for each attack, resulting in a presentation of a secure and resilient vehicle design. Ultimately, we present the *Resilient Shield* a novel and imperative framework to justify and ensure security and resilience within the automotive domain.

4.1 Introduction

The complexity of vehicles is increasing. Consequently, vulnerabilities which might be exploited increase as well. Attacks to vehicular systems can be realized: (i) *indirectly* via compromised devices e.g., phones, dongles, or workshop computers connected to vehicle interfaces; (ii) *directly* via physical interfaces e.g., debug ports and the OBD-II connector; and (iii) *remotely* via various malicious sources, such as rogue access points and compromised servers. It has been demonstrated that vehicle cyber-attacks e.g., physical attacks [12] and remote attacks [13] are potential threats that have to be taken seriously. As a case in point, Miller and Valasek [98] performed a successful remote attack on a Jeep Cherokee via the Internet taking control of its primary functions by exploiting an open port via a cellular channel, an attack that led to a recall of 1.4 million vehicles. In [99], researchers managed to get remote access to the CAN bus of a BMW by compromising its infotainment system, allowing them to execute arbitrary diagnostic requests. Vulnerabilities in phone applications paired to vehicles have been exploited by adversaries to track vehicles, unlock the doors and to start their ignitions [100–102].

Motivation. Securing a vehicle as an afterthought is cumbersome, considering both the complexity which constantly increases and the existing dependencies on current architectural design. Hence, it is imperative to consider security during the vehicle's complete life cycle from idea to cessation.

There are increased requirements towards ensuring a resilient vehicle design, in a way that a vehicle should be able to withstand various types of cyber-attacks, malfunctioning units, and other external disturbances. Consequently, the resilient design should be able to *prevent*, *detect*, and *respond* to cyber-attacks, something which is also in line with the UNECE regulation [4] and the upcoming standard for automotive cyber security ISO 21434 [103]. In short, *prevention* is accomplished with security controls, *detection* by identifying faults and attacks, and *response* are mechanisms related to handling the detected anomalies with the ability to restore and maintain operation. However, there is currently no clear guidance how to comply with the aforementioned regulations and standards from a technical perspective. The *start*, *predict*, *mitigate*, and *test* (*SPMT*) is a systematic approach for identification and mitigation of vulnerabilities in vehicles [9]. The aim of *SPMT* is to ultimately enhance the security of vehicles through their entire life cycle. In this paper, we use and extend the *SPMT* methodology to establish an in-depth resilient design model with imperative mitigation mechanisms.

Contributions. By applying the *SPMT* methodology, we performed a comprehensive threat and risk analysis of 52 published attacks against vehicles from the past 10 years. 37 of these attacks were considered significant due to their high risk and were thus further mitigated with imperative security and resilience techniques. In this process, we have developed a *threat model* for securing vehicles by identifying vital vehicle assets and the related potential threat actors, their motivations and objectives. Moreover, we have developed a comprehensive *attack model* created from the analysis of the identified threats and attacks, further filtered and categorized based on attack type and risk criteria related to the probability and consequences of the attack. We present a comprehensive summary of the result from applying the *SPMT* methodology,

an exhaustive mapping between asset, attack, threat actor, threat category and resilience mechanism for each attack. Ultimately, we define necessary security and resilience enhancements for vehicles, the *Resilient Shield*, which also validates the effectiveness of the methodology. To the best of our knowledge, our result is both novel and imperative to justify and ensure security and resilience within the automotive domain.

4.2 Related Work

Good practices for security of smart cars [104], *Cyber security and Resilience of smart cars* [105], and *The Cyber security guidebook for cyber physical vehicle systems, SAE J3061* [106], provide guidelines regarding threat and risk assessment. *EVITA* [107] proposed a method for security, safety, and risk analysis of in-vehicle networks, whereas *HEAVENS* [108] proposed a security model based on security objectives from EVITA and security attributes from Microsoft *STRIDE* [109]. Rosenstatter et al. [110] continue with the result from an analysis such as HEAVENS and map the identified security demands to security mechanisms. However, this mapping focuses only on securing the in-vehicle network.
The *SPMT* methodology builds on existing methods, models and security principles that are applicable to different phases in a vehicles life cycle. By adapting and incorporating relevant parts suitable for the vehicular domain, a comprehensive security and safety enhancement is achieved. Consequently, the *SPMT* methodology covers the vehicles entire life cycle, something which cannot be achieved with existing methodologies [9]. *SPMT* adopts Microsoft's *STRIDE* categorization [109] which enables a mapping of attacks to a category with associated security attributes. Thus, mitigation mechanisms can be considered for the attribute and consequently mitigate more than one attack. Additionally in *SPMT*, a reduction analysis is performed for critical threats by creating attack trees to connect the vulnerability with the threat, i.e., an attacker wanders from a leaf node (condition) to the root of the tree (attacker objective). Consequently, the closer to the root a countermeasure is placed, the more conditions are mitigated. Moreover, some conditions can be attained by more than one attack, hence a countermeasure can mitigate several attacks. The REMIND framework [7] for vehicular systems provides a taxonomy for resilience techniques identified from a review of existing work. In this paper we take advantage of previous knowledge and new results by applying the *SPMT* methodology. In the next sections we present the detailed approach followed by the results.

4.3 Approach

We use the aforementioned *SPMT* model to perform a comprehensive threat modelling and risk assessment of published attacks to further map these threats and attacks to imperative security and resilience mechanisms.

The *SPMT* methodology has 4 phases: *Start*, *Predict*, *Mitigate* and *Test*. In this paper, we perform the first three phases on a Target Of Evaluation

(ToE) and analyze security threats and attacks as well as provide mechanisms for the mitigation thereof (see Figure 4.1).

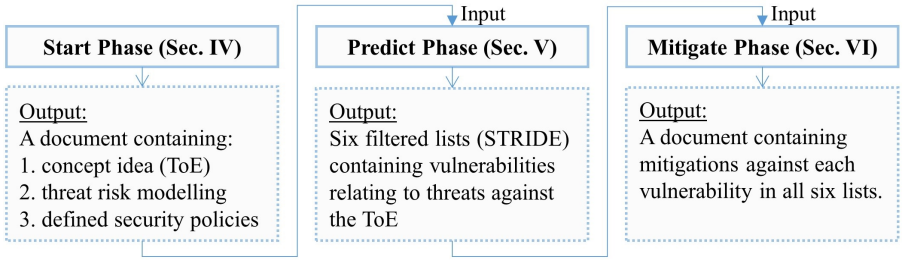

Figure 4.1: The first three phases of the SPMT methodology

In the *Start Phase*, we address the following questions. *What are the threats requiring a resilient design? What are the entry points to the vehicle? Who are the actors, their motivators, and their objectives?* The outcome of the *Start Phase* is a threat model and high-level goals for the enforcement of security and safety attributes.

In the *Predict Phase*, we address the following question. *What are the potential attacks?* The outcome of the *Predict Phase* is an *attack model* which contains relevant attacks categorized and filtered according to a stated criteria.

In the *Mitigate Phase*, we address the following question. *What are the needed mechanisms to ensure a resilient design?* The outcome of the *Mitigate Phase* is a resilient design framework i.e., the *Resilient Shield*, which provides mechanisms and goals for detecting, preventing, and responding to security threats and attacks.

The *Test Phase* includes the implementation of the mitigation mechanisms followed by an execution of different security tests, such as fuzz, vulnerability, and penetration testing. In this paper, we do not perform the *Test Phase*; however, we plan to test the identified mitigation mechanisms within an industrial context in the future.

In the following sections, we perform and provide the outcomes of the first three phases of the *SPMT* methodology (see Figure 4.1) that are used to establish the *Resilient Shield*.

4.4 Threat Model

A threat model is created by considering: (i) the target of evaluation (ToE), and (ii) attackers as well as their motivators and objectives. First, our ToE is stated as the complete vehicle provided by the manufacturer, where we propose to include the following assets. As shown in Table 4.1, the relevance of these assets is verified by the mapping to attacks.

Internal and external communication: *Automotive Bus technologies*, e.g., CAN, FlexRay, LIN, MOST and Ethernet. *Connection interfaces*, e.g., OBD-II, USB, debug ports, Wi-Fi and Bluetooth.

Hardware: *ECUs*, e.g., sensor signal processing. *Sensors*, related to speed, position, temperature, airbag and object detection. *Actuators*, translate signals from ECUs into actions, e.g., braking, steering and engine control.

Software in transit, rest or running: *Software update systems*, e.g., over-the-air or workshop updates. *Software installed or running* in ECUs.
Data Storage: *Sensitive data*, e.g., cryptographic keys, forensics logs and reports.

Second, we propose a simplification of threat actors (i.e., attackers) inspired by the work of Karahasanovic et al. [111] in relation to motivators and objectives.
Actors and Motivators. *The Financial Actor* is driven by financial gain in relation to a company (intellectual property), organization or individual. This actor can be the owner who wants to make unauthorised modifications (e.g., chip tuning) or criminals who install ransomware. *The Foreign Country* is driven by power through cyber warfare, with the intent to disable viable assets within infrastructure (e.g., transportation). *The Cyber Terrorist* is driven by ideological, political or religious objectives. *The Insider* is motivated by retaliation or other personal gains, has knowledge of sensitive information and may plant malicious code into the vehicle. *The Hacktivist* is driven by publicity or adrenaline (i.e., the rush) and can have an agenda for political or social change. *The Script Kiddie* has usually no clear objective, possess limited knowledge and is often using already available tools and scripts. However, the reality is usually a combinations of the mentioned categories and objectives, and actors can be *black hat*, *gray hat*, or *white hat* hackers in relation to society's interpretations of the hackers' intentions. *White hat*, are assumed to be the good guys, *black hats* are the bad guys, and *grey hat* are somewhere in the middle.

Furthermore, in Section 4.6 we adopt the security and safety attributes used in *SPMT*. These attributes are imperative to uphold to ensure a secure and resilient vehicle. On the other hand, the actors are driven by stated *motivators* (e.g., financial, ideological, publicity) with the goal of compromising these attributes. A discussion and a brainstorming about fulfilment of these attributes is part of the *Start Phase*, however we have chosen to include it in Section 4.6 to have all considerations for mitigation in one section. Stated assets and actors are applied to Table 4.1 and used in the following section.

4.5 Attack Model

We perform a qualitative risk assessment of published attacks covered in news media and research publications by estimating (i) the probability and (ii) the consequences of the attacks based on the following criteria. As shown in Table 4.1, the affected assets, the threat actors and the STRIDE categories for each attack are considered during this assessment.
Attack Probability. The first step in this phase is to define attack probability where the three following estimates should be used:
E1: *Where, when, and in what situation can the attack be carried out?*
E2: *What expertise is required of the attacker?*
E3: *How much time does it take to perform the attack?*
The resulting probability is on a scale of 1 to 3, where 3 indicates that an attack is more probable to take place. The highest value in E1-E3 is chosen.
Attack Consequence. In the second step, the consequences are defined

by assessing the effect of the attack on the operational, safety, privacy, and financial aspects. The resulting consequence is on a scale from 1 to 3, where 3 indicates that the consequence is more severe. The highest value is chosen.

Risk Assessment. Once we get the estimates of the attack probability and consequences, we estimate the overall risk by calculating the product of the probability and the consequence, which gives a risk value between 1 and 9 (see Figure 4.2). To achieve a realistic balance between the financial cost for mitigation and its related complexity versus the risk and asset value, we consider only the most significant threats. These threats have a risk value of 6 or 9, which is in line with ISO 26262 and ASIL [112] and corresponds to high and critical risk.

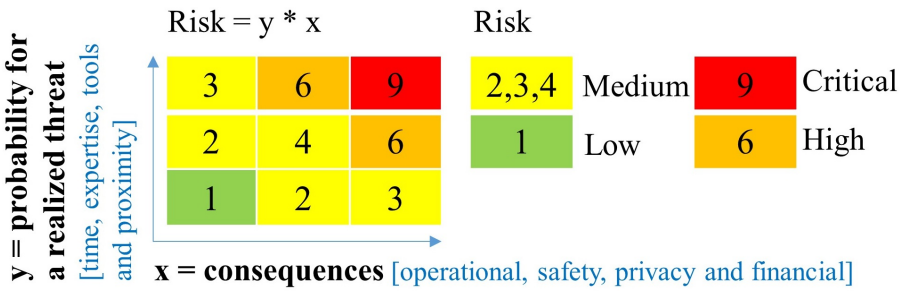

Figure 4.2: Adapted table for the risk calculation from the SPMT methodology.

4.5.1 Disclosed Attacks

To create the *attack model*, we follow the *SPMT* recommendation for search criteria and query scopus[1] and Google scholar for academic work, and common vulnerability databases (NVD, CVE) with keywords related to vehicle, attack and STRIDE categories (e.g., spoofing) or related terms (e.g., mitm). Moreover, we do query the Google search engine for media reports on attacks. Next, we classify the attacks according to STRIDE categories, followed by some examples. Attacks are considered and analyzed with respect to probability, consequence and risk within their respective category. Out of a total of 52 published attacks, we have identified 37 high and critical risk attacks which are further considered in this work.

1) Spoofing Attacks - Authenticity, Freshness [100,113–130]. The goal of the attacker is to intercept, hijack, manipulate or replay the communication with a potential remote access persistence. *Security flaws in mobile software*, such as demonstrated in the OwnStar attack [100]. OwnStar intercepts communication after the OnStar user opens the application, whereas the OwnStar device gains the user's credentials. Relay attacks, as in compromise of remote keyless entry systems as well as breaking poor authentication mechanisms [113–115]. GNSS spoofing considers broadcasting fake signals over authentic in order to to trick a receiver, with the intention to get a vehicle off course [116]. *In-vehicle protocol spoofing*, can affect safety critical actuators, such as brake, steering and engine control. Protocols themselves might lack inherent mechanisms for security

[1] `https://www.scopus.com/`

which makes active attacks possible such as malicious drop, modify, spoof, flood and replay of messages.

2) Tampering Attacks - Integrity [13,99,128,130–133]. Vulnerable USB/OBD-II dongles or compromised in-vehicle devices can potentially enable a hacker to control the communication. Devices can be compromised in various ways e.g., vulnerabilities in proprietary authentication mechanisms can enable the right to run sensitive diagnostics commands. Brute-force attacks can be used to retrieve cryptographic keys, with potential to upload exploits to ECUs. Physical tampering of ECUs or other connected devices. Manipulated firmware in current ECUs, such as malicious code injection via firmware update. Replacement of ECUs or new devices to eavesdrop/inject messages or to manipulate software, modify or compromise vehicle functions. Vulnerable connected devices such as OBD and USB dongles can potentially provide remote access to individual cars and vehicle fleets [132]. Moreover, in [13] firmware was extracted and reverse engineered, manipulated and injected directly into ECU firmware facilitating persistent and bridging capabilities for attacks.

3) Repudiation Attacks - Non-repudiation, Freshness. An attacker manipulates or removes forensic in-vehicle data, such as GPS coordinates, speed, acceleration and brake patterns, with the intention to hide traces of the attack. Despite our best effort, we did not find attacks which can be clearly mapped to this category; however, this type of attacks will likely be more frequent in the future due to both increased number of attacks and digital forensic investigations.

4) Information Disclosure Attacks - Confidentiality, Privacy [102, 130, 131, 134–137]. An attacker may be able to exploit cryptographic keys and consequently decrypt sensitive data by e.g., reverse engineering software with hard-coded keys. Bad routines for handling of replaced unit led to leaked sensitive data such as owners home and work address, calendar and call entries and Wi-Fi passwords [134]. A mobile application for vehicle control contained hard-coded credentials, thus an attacker may be able to retrieve sensitive data remotely by recovering the key from the application [102]. A vulnerability in an OBD-II dongle exposed all transferred data to the public [135]. Vulnerabilities in automotive bus technologies make various attacks possible, such as sniffing of CAN traffic due to its broadcast transmission and lack of encryption [136].

5) Denial of Service (DoS) Attacks - Availability [126–129, 138–141]. Many attacks focus on the in-vehicle network that uses CAN as this technology suffers from fundamental vulnerabilities with respect to security (e.g., broadcast communication, lack of encryption/authentication). Other attacks range from sending an indefinite amount of data to ECUs to make them unresponsive or crash, exploiting error handling mechanisms, or flooding the network with high priority messages in order to block lower priority messages. A vulnerability in the Bluetooth functionality supported unrestricted pairing without a PIN, thus enabled the potential for sending remote CAN commands affecting safety critical assets [140]. The Bus-off attack made ECUs unresponsive or crash [141]. Murvay et al. [139] managed to disable FlexRay nodes by exploitation of the bus guardian, power saving functionality and by causing loss of synchronization.

6) Elevation of Privilege Attacks - Authorization [98, 102, 128, 130, 131, 133, 142–144]. In [128] two Bluetooth vulnerabilities allowed remote code execution with root privileges. Moreover, manipulation of the firmware of the infotainment

unit enabled injection of arbitrary CAN messages. In [142], they were able to release the airbag by message injection due to a vulnerable authentication mechanism. Lack of authentication in the NissanConnect app allowed to retrieve personal data by entering an URL with the vehicle identification number [144]. The outcome of this phase is applied to Table 4.1 and used in the next phase in the following section.

4.6 Resilient Shield

In this section we present the *Resilient Shield* which consists of high-level security goals emphasizing the overall design requirements resulting from an analysis of the threat model (Section 4.4). We further provide in Section 4.6.2 detailed directives for fulfilling the high-level security goals for resilient vehicles which are based on these goals and the *attack model* (Section 4.5). Table 4.1 summarizes the *Resilient Shield*. We list automotive assets, associate them with high risk attacks, potential threat actors and STRIDE threat categories, and link these to suitable security and resilience techniques to show how *Resilient Shield* can be used to mitigate these attacks.

4.6.1 High-level Security Goals (SGs)

The following high-level goals are the result of an analysis of the *threat model* detailed in Section 4.4. Each SG is associated with the relevant safety and security attributes they enforce.

SG.1 Secure Communication. *Integrity*, *authenticity* and, in specific cases, *confidentiality* need to be ensured for communication. *Integrity* and *authenticity* allow to verify the origin of the message and protect the message from being altered during transmission. *Confidentiality* can be achieved through encryption of the message to prevent unauthorized read access. *Freshness*, e.g., via counters or timestamps, can be used to mitigate replay attacks.

SG.2 Readiness. *Availability* to authorized entities under normal circumstances as well as disturbances. Even if an adversary tries to disrupt the information flow, the *integrity* and *availability* of correct information needs to be guaranteed.

SG.3 Separation of Duties is needed to limit access to resources for *authorized* entities only. *Authorization* should be combined with the principle of *least privilege* to limit the number of entities having access to a resource to the minimum.

SG.4 Secure Software Techniques need to provide security features to ensure that the executed software has not been modified by an unauthorized entity (*authenticity*) and that the software does not contain disclosed vulnerabilities.

SG.5 Separation/Segmentation on an architectural or process level is necessary in order to limit access and reduce the severity in case of an intrusion (*availability*). *Isolation* techniques, e.g., process isolation, should be considered where possible.

SG.6 Attack Detection and Mitigation is of utmost importance to enable the system to react and ideally prevent further damage to the system.

SG.7 State Awareness should be ensured with the ability to switch between various operational states, thus providing *reliability* and *maintainability*.

SG.8 Forensics is necessary for post analysis of detected malicious events and accordingly updating access control policies and other preventive measures. Physical security, such as vehicle locks, alarm system, and protecting infrastructure server rooms should be considered. Components must be extensively tested against requirements separately and when integrated into the vehicle, such as stated in the *SPMT Test Phase*. *SPMT* suggests to use both a qualitative and quantitative assessment; however, we focus on the qualitative assessment as the aim of *Resilient Shield* is to guide the resilient design of automotive systems. Moreover, a reduction analysis of attack trees is suggested to find commonalities in countermeasures; however this is not considered and is thus left as future work.

4.6.2 Detailed Directives

In this section, we list detailed techniques and patterns that contribute to the security and resilience of automotive systems based on the identified security goals, *threat* and *attack model* presented in this paper. First, we incorporate the identified patterns from the REMIND framework [7] in *Resilient Shield* and further extend them with security techniques to provide a comprehensive collection of both, security and resilience techniques for automotive systems. Second, we further discuss the security aspects of the identified resilience techniques. Next, we detail these techniques.

Authentication: Message authentication can be achieved through Message Authentication Codes (MACs) or signatures which ensure that the message: (i) is created by the claimed source and (ii) has not been altered during transmission. The authentication of devices can verify that the hardware, e.g., the head unit or a diagnostic device, is legit.

Encryption: Encryption of data ensures the protection of intellectual property, makes it more difficult to reverse engineer software, protects cryptographic material and the privacy of users and forensics data.

Redundancy/Diversity: A voting mechanism is used when comparing the output of two or more redundant systems or software functions. Redundancy increases the resilience against anomalies; however, from a security perspective it must be ensured that the voting process cannot be exploited by an attacker to perform DoS or spoofing attacks.

Access Control: Gateways with firewall capabilities allow filtering of messages between different networks in the vehicle. In addition, host-based firewalls on the ECUs can limit the exposure of open communication ports. Securing physical debug ports is vital to protect against unauthorized exploitation. Access control to resources such as files, computation, and diagnostic commands can be provided by the operating system or by e.g., challenge-response authentication.

Runtime Enforcement: Runtime verification is combined with reactive measures when safety properties are violated [7, 66].

Secure Storage: Cryptographic material needs to be protected against unauthorized modifications and read access. Data can be either stored encrypted in the regular file system or in a protected memory partition.

Secure Boot: A validation of the authenticity and integrity of the firmware to be loaded during the boot process [145].

Secure Programming: Secure programming guidelines such as MISRA

C [146] are important to avoid common programming errors. Additionally, trusted execution environments may be necessary for isolating and securing applications.

Secure Software Update: The ability to update software is not only a necessity to improve and extend functionality, it is also essential for security, e.g., to mitigate vulnerabilities. In addition, the update process itself needs to be secure [147], during the distribution and installation process.

Verification & Validation: The *Test Phase* in *SPMT* focuses on the need for security testing and verification of each asset by doing fuzz, vulnerability and penetration testing. In addition to security testing, the verification and validation of functionality and safety is required [7, 9].

Separation: Architectural separation can be achieved through physical separation into smaller networks or through virtualization techniques allowing to allocate resources to specific functions or systems.

Specification-based Detection: Knowledge about abnormal behavior is used to detect anomalies and attempts to exploit known vulnerabilities. It also requires domain knowledge and needs to be updated regularly [7, 148].

Anomaly-based Detection: Is based on defining normal behavior and deviations trigger alerts and has the potential to detect unknown attacks. Anomaly-based detection can be categorized in statistical, information-theoretic, machine learning and localization techniques [7, 148].

Prediction of Faults/Attacks: Predicting the next step or the ultimate goal of an ongoing attack.

Adaptive Response: The function response may be temporarily adapted, e.g., through a model, while under attack [7].

Reconfiguration: Graceful degradation can be used to limit the impact of an attack when preventive measures failed.

Migration: The ability to migrate services to other nodes in order to maintain system functions when under attack [7].

Checkpoint & Rollback: Used to recover the system to a desired state. The state needs to be secured, e.g., through secure logging, to defend against possible attacks that aim at modifying a saved system state [7].

Rollforward Actions: Upon detecting an anomaly or error the system transitions back to the state immediately before the event happened. Similarly to rollback it needs to be ensured that this mechanism cannot be exploited [7].

Self-X: The system needs to be aware of its state and able to switch to other states when anomalies occur [7, 149].

Robustness: Employed mechanisms and functions need to be robust against anomalies [7].

Forensics: Secure logging is used to record events, e.g., detection of an ongoing attack, use of specific services or diagnostics. In addition, events with non-repudiation claims can be used as evidence of a crime.

Table 4.1 presents the *Resilient Shield*. Assets with high or critical risk threats are associated with appropriate security and resilience techniques demonstrating the ability of *Resilient Shield* to defend against these threats. For example, hacktivists and insiders are the main threat actors for *communication:external:debugport*, such as JTAG, and needs to be protected with authentication mechanisms combined with access control or, if not possible otherwise, with physical protection (e.g., deactivation).

Table 4.1: Resilient Shield. A mapping from automotive assets to identified attacks, potential threat actors, STRIDE threat categories and ultimately to appropriate security and resilience techniques, and Security Goals (SGs).

Assets targeted by attacks with high or critical risk.
ToE category:subcategory reference

■ Resilience patterns identified in REMIND [7]

Potential Threat Actors
Financial Actor (FA)
Foreign Country (FC)
Cyber Terrorist (CT)
Insider (IN)
Hacktivist (HA)
Script Kiddie (SK)

STRIDE categories
(S)*poofing*
(T)*ampering*
(R)epudiation
(I)nformation Disclosure
(D)*enial of service*
(E)*levation of privilege*

			(SG.1,8) Authentication	(SG.1) Encryption	(SG.2) Redundancy/Diversity ■	(SG.3) Access Control	(SG.3) Runtime Enforcement ■	(SG.4,8) Secure Storage	(SG.4) Secure Boot	(SG.4) Secure Programming	(SG.4) Secure Software Update	(SG.4) Verification & Validation ■	(SG.5) Separation ■	(SG.6) Specification-based Detection ■	(SG.6) Anomaly-based Detection ■	(SG.6) Prediction of Faults/Attacks ■	(SG.6) Adaptive Response ■	(SG.6) Reconfiguration ■	(SG.6) Migration ■	(SG.6) Checkpoint & Rollback ■	(SG.6) Rollforward actions ■	(SG.7) Self-X ■	(SG.7) Robustness ■	(SG.8) Forensics ■
Hardware																									
sensor:camera [126, 127]	FC, CT, HA	S, D			•												•	•				•	•		
sensor:GNSS [116, 118, 121, 122, 124]	FC, CT, HA	S	•		•										•		•	•				•	•		
sensor:lidar [120, 126]	FC, CT, HA	S, D			•												•	•				•	•		
sensor:ultrasonic [127]	FC, CT, HA	S, D			•												•	•				•	•		
Communication																									
internal:can [132, 136, 138, 139, 141]	FA, FC, CT, IN, HA	S, T, I, D	•	•	•	•	•						•	•	•	•		•	•			•			•
internal:flexray [129]	FA, FC, CT, HA FC,	S, D	•			•	•						•	•	•			•				•			•
external:bluetooth [99, 128]	CT, HA	S, T, D, E	•			•							•												
external:usb [99]	FC, CT, HA	S, T, E	•			•							•												
external:keyfob [114, 115]	HA, SK	S	•			•									•								•		
external:wifi [100, 125]	HA, SK	S, I	•	•		•				•			•												
external:cellular [98, 99, 133, 137, 143, 144]	FC, CT, HA, SK	S, T, I, D, E	•			•							•												
external:obdII [102, 119, 123, 130, 132, 135, 138, 140]	CT, HA	S, T, I, D, E	•			•	•						•	•	•			•			•		•		•
external:debugport [98, 133]	HA, IN	I, E	•			•																			
Software																									
running:state [117]	FC, CT, HA	S, D				•				•				•	•						•	•			•
running:firmware [98–100, 125, 128, 131, 133, 137, 143, 144]	FC, CT, HA	S, T, E				•				•	•	•	•	•	•			•			•				•
instorage:update [99, 128, 133]	HA, SK	S, T, E	•	•		•		•	•		•		•	•	•	•	•	•	•						•
instorage:weakcrypto [113, 142, 144]	FC, CT, HA, SK	S, E	•							•														•	
Data Storage																									
crypto:certificates [133]	FC, CT, HA	I		•		•		•	•																
hw:replaced [134]	HA, SK	I	•	•		•																			

4.7 Conclusion

We have performed a comprehensive threat and risk analysis of published attacks against vehicles and derived imperative security and resilience mechanisms by applying the *SPMT* methodology. A *threat model* with vital vehicle assets and related potential threat actors, their motivations and objectives was developed. By an extensive analysis of threats and attacks, further filtered and categorized based on attack type, probability and consequence criteria, an *attack model* was developed based on the remaining high risk attacks. Based on the developed models, a comprehensive mapping between asset, attack, threat actor, threat category, and defense mechanisms was performed for all attacks and is presented in Table 4.1. Table 4.1 summarizes the outcomes by applying *SPMT*, i.e. the *Resilient Shield*, a novel framework both justifying and defining imperative security and resilient mechanisms needed in a modern vehicle. Consequently, the *Resilient Shield* can be used as a vital baseline for protection against common security threats and attacks.
We believe our work is imperative for facilitating and guiding the design of resilient automotive systems; however, it still remains to be seen how large the coverage is in relation to future attacks. Moreover, testing and validation of the *Resilient Shield* within an industrial context is left as a future work.

Chapter 5

UniSUF: a unified software update framework for vehicles utilizing isolation techniques

Adapted version that appeared in 19th escar Europe 2021

K. Strandberg, D.K. Oka, T. Olovsson

Abstract. Today's vehicles depend more and more on software, and can contain over 100M lines of code controlling many safety-critical functions, such as steering and brakes. Increased complexity in software inherently increases the number of bugs affecting vehicle safety-critical functions. Consequently, software updates need to be applied regularly. Current research around vehicle software update solutions is lacking necessary details for a versatile, unified and secure approach that covers various update scenarios, e.g., over-the-air, with a workshop computer, at factory production or using a diagnostic update tool. We propose UniSUF, a Unified Software Update Framework for Vehicles, well aligned with automotive industry stakeholders. All data needed for a complete software update is securely encapsulated into one single file. This vehicle unique file can be processed in multitudes of update scenarios and executed without any external connectivity since all data is inherently secured. To the best of our knowledge, this comprehensive, versatile and unified approach cannot be found in previous research and is a contribution to an essential requirement within the industry for handling the increasing complexity related to vehicle software updates.

5.1 Introduction

A vehicle can contain more than 150 ECUs (Electronic Control Units) and over 100M lines of code. The complexity of software within the automotive domain is increasing and with it the risk for vulnerabilities. To address this, there are ongoing activities for vehicle software updates, such as ISO/CD 24089 [6] and UN Regulation No. 156 regarding vehicle software update requirements [3]. The latter states, among other, requirements for vehicle manufactures to have a secure software update process. While standards and regulations typically focus on high-level requirements, technical design and implementation requirements are left up to the automotive organizations. There is a risk that if the software update process is vulnerable, it can be exploited by attackers who could potentially introduce malicious code at some stage into the software update process that finally reaches in-vehicle systems causing life-threatening hazards such as manipulated brakes, steering, or engine control.

A secure software update framework that can support numerous different update scenarios, such as over-the-air, in workshops, and in factories, with or without Internet access, is required for automotive organizations in order to apply software updates to address vulnerabilities in a timely and regular manner. Our approach is to provide a cost-effective, open architecture, with increased security through isolation and separation of duties that is comprehensive to support numerous use cases. Thus, we propose UniSUF, a versatile and unified approach for secure vehicle software updates. By using multiple signing and encryption keys, all data needed for a complete software update is securely encapsulated into one single file, the *Vehicle Unique Update Package (VUUP)*. This vehicle unique file can be processed by a vehicle ECU, using a workshop computer, at factory production or with a diagnostic update tool, hence considerably simplifying software management processes. At the receiving vehicle side, this file is decapsulated and validated layer by layer, where cryptographic material and sensitive operations are isolated within a trusted execution environment to ensure both the integrity and the confidentiality of the data. The main contributions of this paper are:

- We have analyzed and reviewed several software update use cases in the automotive industry and as a result, defined a number of constraints and conditions for a unified and versatile approach.

- Considering these constraints and conditions, we suggest an approach for vehicle software updates, well aligned with automotive industry stakeholders. In-depth details give a comprehensive overview for a possible secure implementation covering the whole software chain from producer to receiver.

- We have reviewed the suggested approach with automotive software update architects to ensure that the proposed approach can be practically deployed and efficiently adopted for vehicle software updates.

5.2 Problem Statement

Considering the different existing use cases for vehicle software updates, such as over-the-air, using a workshop computer, at factory production, or with a diagnostic update tool, each use case typically has its own approach which causes complexity. Moreover, new use cases for software updates need to be considered with future demands to support 3rd party component updates ([150], [151]). Therefore, to simplify, reduce costs, allow flexibility, and to make the update process manageable, all while considering security aspects, we propose a unified and versatile approach to handle all the use cases.

After reviewing the above-mentioned use cases, the following constraints and conditions are defined for a unified software update framework:

- Support for online updates (software update files and/or cryptographic credentials/operations require online access).
- Support for offline updates (software update files and cryptographic credentials/operations are accessible offline).
- Should not rely on additional input for cryptographic keys or installation instructions, e.g., from a diagnostic update tool (i.e., all data needed for a complete software update is securely encapsulated into one single file and no additional input is required).
- No dependency on the data distribution model (i.e., software update files can be provided through different means and it does not matter how they are distributed to the vehicle).
- No dependency on software update storage location (i.e., software update files should be independently protected regardless of where they are stored).
- Flexible and modular to support 3rd party component updates.

We have taken these constraints and conditions into consideration when designing a software update framework to allow for a unified and versatile approach to support different use cases. Our proposed software update framework is described in the next section.

5.3 UniSUF: A Unified Software Update Framework

In this section, we present the Unified Software Update Framework (UniSUF). First, an overview of the involved entities in the framework is presented, followed by a brief explanation on how to secure data distribution and data execution, and finally, the procedure for preparing software update files is given.

5.3.1 Entities

There are three main entities involved in the software update process: the producer, the consumer, and the repository. The producer is responsible for

producing the software. The consumer is responsible for the download and installation process of the software, and the repository is a storage point for software preferably located in various cloud sources, enabling both proximity and redundancy for data in relation to the vehicle.

An overview of the data distribution in the backend handled by the ***Producer Agent (PA***) is shown in Figure 5.1. The main entities it contains are:

- ***Producer Security Agent (PSA)*** facilitates functionalities for secure key generation using ***Secure Key Generator (SKG)***, secure storage for cryptographic material using ***Cryptographic Material Storage (CMS)*** and signing of data using ***Producer Signing Service (PSS)***.
- ***Version Control Manager (VCM)*** has control over available software versions w.r.t. current vehicle status to create both download instructions using ***Producer Download Agent (PDA)*** and installation instructions using the ***Producer Installation Agent (PIA)***.

On the receiving side, Figure 5.4 shows the ***Consumer Agent (CA***) handling data distribution to the vehicle. The main entities it contains are:

- ***Consumer Download Agent (CDA)*** downloads required data, e.g., instructions and software files, verifies the authenticity of the data and initiates installation using the ***Consumer Installation Agent (CIA)***.
- ***CDA*** and ***CIA*** uses the ***Consumer Security Agent (CSA)*** which requires a Trusted Execution Environment (TEE) in order to support secure operations and store cryptographic keys securely.

By using isolation mechanisms and implementing each entity as a module according to the principles of *least privilege* and *separation of duties* a potentially compromised entity cause the least possible harm to the complete system. These modules can be secured either locally or in the cloud.

5.3.2 Securing Data Distribution and Data Execution

To be able to secure the *data distribution* and *data execution*, we propose using signed asymmetric and symmetric keys in conjunction with key wrapping mechanisms. Symmetric session keys are used to encrypt sensitive cryptographic material needed for the update processes, such as keys for unlocking ECUs and keys for decryption of software. The symmetric session keys are encrypted with a public vehicle unique asymmetric key, ensuring the secure storage and transfer of key material. Using an asymmetric key for key wrapping ensures that only the vehicle with the corresponding private key can decrypt the encrypted session keys.

Policies dictate rules for each individual encrypted session key, where policies and keys in conjunction are signed i.e., giving rise to a *Key Manifest (KM). KMs* are securely processed at the receiving side, where session keys are appointed to certain trusted applications according to the stated policies. The functionality of trusted applications can be decryption of software files, unlock ECUs for software updates, and signing of installation reports and logs.

5.3.3 Preparation of Software Update Files

The individual files that contain the actual software need to be secured, ensuring both confidentiality and authenticity. Considering the entities in the framework the procedure to secure software files is as follows.
1. The producer of software signs software files with a supplier specific signing certificate to provide authenticity. If supported, this signature is later validated by end receivers (e.g., ECUs) before installation. Software files are uploaded to the *Producer Local Secure Storage*, shown in Figure 5.1.
2. *VCM* receives the software files and validates the software supplier's signature.
3. *VCM* requests a symmetric encryption key (hereafter called sw_key) from *PSA* and encrypts the software file with this key to provide confidentiality.
4. *VCM* requests a signature of the hash of the encrypted software file from *PSS*. The signature is added to the encrypted file metadata to provide authenticity.
5. *VCM* performs mutual authentication towards the cloud software repository and uploads the signed encrypted file to the cloud and stores the url to this file in a database.
6. *VCM* securely stores the symmetric encryption key (i.e., the corresponding sw_key) in *CMS* to be retrieved later, and encrypted and included into a *Secure Key Array (SKA)* for a future software update (cf. Step 6. in Section 4.1).

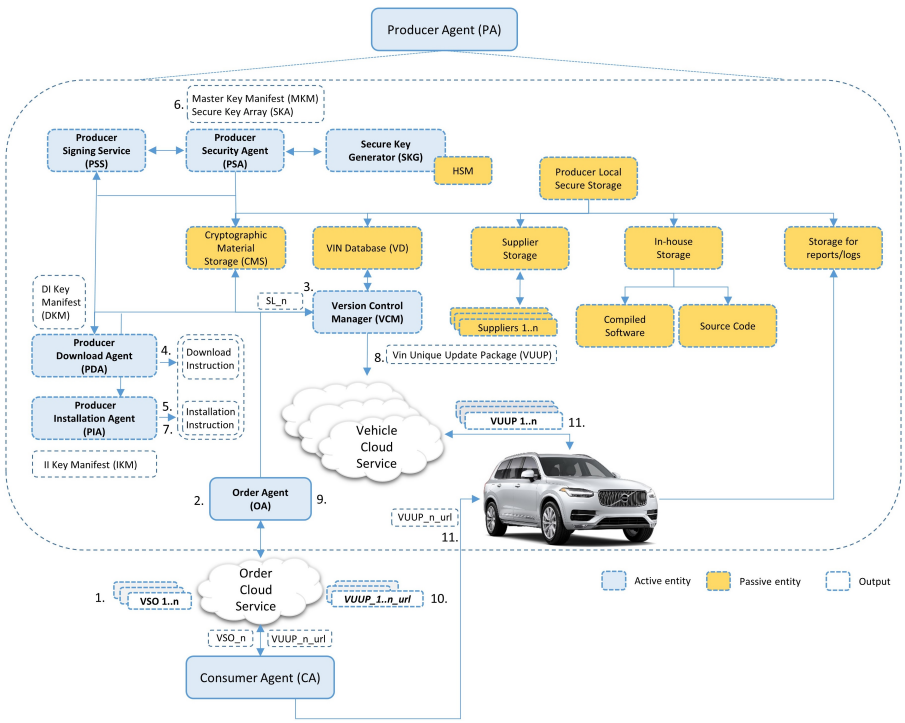

Figure 5.1: Data distribution in the backend

5.4 The Software Update Process

In this section, we dive into the details of the complete software update process in UniSUF. Explanations of the abbreviations used can be found in Table 5.1.

Table 5.1: Abbreviations

Abbreviations	
Vehicle Identification Number (VIN)	The VIN number is a vehicle unique fingerprint, and is composed of 17 characters.
Producer Agent (PA)	Parent entity consisting of many children entities covering backend requirements.
Producer Security Agent (PSA)	Responsible for handling cryptographic material in the backend systems.
Producer Signing Service (PSS)	Executes signing requests i.e., returns signatures of hash values requested by authenticated entities.
Order Agent (OA)	Responsible for managing software requests from consumers.
Secure Key Generator (SKG)	PSA uses this module for the secure generation of key material.
Secure Key Array (SKA)	An array that PSA creates with cryptographic material related to a VIN unique software package.
Version Control Manager (VCM)	Responsible for management of software versions related to unique vehicles and for repackaging of data into the final VUUP file.
Producer Download Agent (PDA)	Creates the instructions for the download of software for a certain VIN.
Producer Installation Agent (PIA)	Creates the diagnostic instructions for installation of software for a certain VIN, including retrieving necessary cryptographic material.
VIN Database (VD)	Stores VIN unique data related to software.
Cryptographic Material Storage (CMS)	Secure storage of cryptographic material.
Download Key Manifest (DKM)	The manifest that contains the DKM session key with the policy for decryption of the download instruction.
Installation Instruction Key Manifest (IKM)	Contains the IKM session key with policy for decryption of the installation instruction.
Master Key Manifest (MKM)	Contains MKM session keys with policies for decryption of cryptographic data.
Vehicle Unique Update Package (VUUP)	The update package that includes information to perform a complete vehicle software update, e.g., software download instructions, installation instructions and cryptographic material.
Consumer Agent (CA)	The parent entity to the children entities covering vehicle requirements for the software installation process. The localization for children entities can be adapted to accommodate various use cases, e.g., OTA, workshop, and factory.
Consumer Download Agent (CDA)	Executes download instructions and retrieves required software files to local storage.
Consumer Installation Agent (CIA)	A diagnostic client responsible for the execution of installation instructions and requests to CSA for the execution of cryptographic material.
Consumer Security Agent (CSA)	A trusted execution environment (TEE), with prestored certificates between vehicle manufacture and CSA; which enables secure transfer and execution of cryptographic material from the backend to the vehicle.
Key Wrapping (KW)	The process of encrypting one key with the use of another symmetric or asymmetric key, to securely store or transmit it over an untrusted channel.
Key Manifest (KM)	Used to define policies and relations for certain keys. Keys are secured with KW, where encrypted keys and policies are signed, giving rise to a KM.

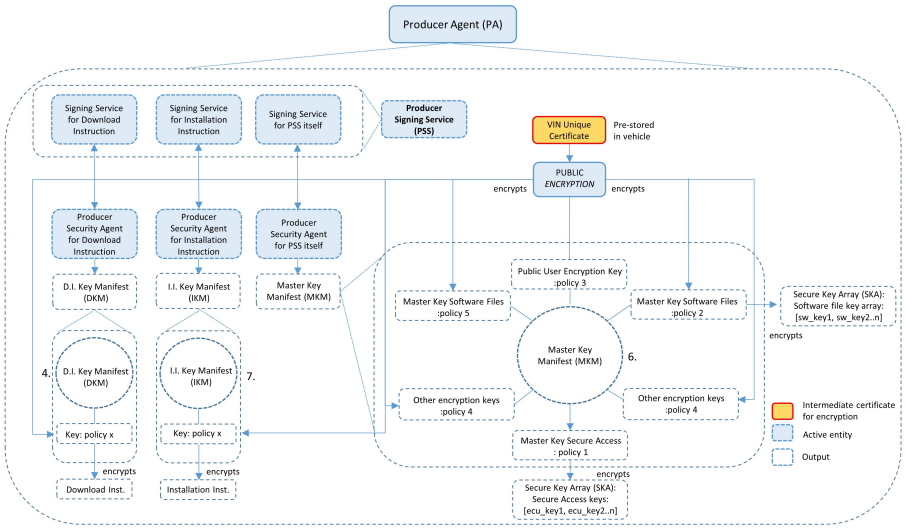

Figure 5.2: Data distribution in the backend in relation to cryptographic material

5.4.1 Encapsulating Data into a VUUP file

Producer Agent (PA): data distribution in backend. Figures 5.1, 5.2 and 5.3 describe the process of creating a complete *VUUP* file. The 11 steps described below are indicated by numbers where relevant in the Figures 5.1, 5.2 and 5.3.

1. Order request. The *Consumer Agent (CA)* in the vehicle, local workshop, or any other consumer, places a signed order on behalf of a *Vehicle Identification Number (VIN)* (i.e., a *Vehicle Signed Order (VSO)*). A *VSO* should contain a complete vehicle readout and be signed by the entity which creates the order. *VSOs* are placed in the *Order Cloud Service* queue. *Output: VSO_n.signed;*

2. Initiate VCM with VSO file. The *Order Agent (OA)* pulls *VSOs* from the *Order Cloud Service* queue, verifies the *CA* signature of the *VSO*, and requests initiation by *VCM* with this *VSO*.

3. VCM creates an SL file with VIN unique software information. *VCM* receives a *VSO* from *OA* for a certain *VIN*. *VCM* validates the signature of the *VSO* and retrieves the latest available software versions and *VIN* vehicle data from the *VIN Database (VD)*. VIN data in *VD* is compared with actual vehicle software readout in the *VSO*. Software deviations are handled, and a signed *Software List (SL)* is created from information in the *VSO* and *VD* and is sent to *PDA*, *PIA*, and *PSA*. *Output: SL.signed;*

4. PDA creates download instructions. *PDA* verifies the *SL* and creates download instructions (list of software urls) for all ECUs based on the *SL*. *PDA* requests a DKM (*Download Instruction Key Manifest*) session key from *PSA* and encrypts the download instructions with this key. Next, this key is encrypted with a vehicle unique public certificate retrieved from *CMS*, where the certificate is validated for authenticity towards the *Root CA* and *OCSP* (Online Certificate Status Protocol). The encrypted DKM session key and a policy that dictates the association to the download instructions give rise to the *DKM*.

A hash is calculated of the encrypted download instructions and the *DKM* separately, and signature requests are sent to *PSS* on behalf of *PDA*, which replies with two separate signatures. *Output: download_instruction.signed; DKM.signed; PDA_cert;*

5. PIA creates installation instructions. *PIA* verifies the signature of the *SL* and creates installation instructions for all ECUs based on the *SL*. *Output: installation_instruction;*

6. PSA requests cryptographic material. *PSA* verifies the *SL* and retrieves the required cryptographic material for software related to the received *SL* from *CMS*, such as keys for unlocking ECUs, privileged diagnostic requests, and software decryption keys. For each category of cryptographic material, *PSA* generates an MKM (*Master Key Manifest*) session key, where each key is associated with that specific category policy. MKM keys are in turn encrypted separately with a vehicle unique public certificate retrieved from *CMS* (same certificate as in Step 4), where the vehicle unique certificate from *CMS* is validated for authenticity towards *Root CA* and *OCSP*. The encrypted MKM keys with each respective category policy give rise to the *MKM*. A key array named *SKA* is created, which includes a sub-array for each category with separately encrypted key data, encrypted with the MKM key which belongs to that specific category. For example, *SKA* can include an array of encrypted symmetric keys used to encrypt/decrypt the relevant software update files, so called sw_keys (cf. Section 5.3.3), and an array of encrypted security access keys used for unlocking relevant ECUs. A hash is calculated of the *SKA* and *MKM*, where after signature requests are sent to *PSS* which replies with two separate signatures. *Output: MKM.signed; SKA.signed;*

7. PIA retrieves the signed MKM and SKA from the PSA, and encrypts/signs the installation instruction. *PIA* request the signed *MKM* and the signed *SKA* from *PSA*. *MKM* and *SKA* signatures are validated where after *MKM* and *SKA* are included as part of the installation instructions. *PIA* requests an IKM (*Installation Instruction Key Manifest*) session key from *PSA* and encrypts the installation instructions with this key. The IKM session key is then encrypted with a vehicle unique public certificate retrieved from *CMS* (same certificate as in Step 4.), where the certificate is validated for authenticity towards the *Root CA* and *OCSP*. The encrypted IKM session key and a policy that dictates the association to the installation instructions give rise to the *IKM*. A hash is calculated of the encrypted installation instructions and the *IKM* separately, and signature requests are sent to *PSS* on behalf of *PIA*, which replies with two separate signatures.
Output: installation_instruction.signed; IKM.signed; PIA_cert;

8. VCM creates the VUUP file.
Input: download_instruction.signed; DKM.signed; installation_instruction.signed; IKM.signed; PDA_cert; PIA_cert;
VCM retrieves the generated data from *PDA* and *PIA*. Certificates are fetched from CMS and validated for authenticity towards the *Root CA* and *OCSP*, signatures are validated with the respective certificate and all the data is repackaged into *VUUP* content. A hash is calculated of the *VUUP* content, and a signature request is sent to *PSS* on behalf of *VCM*, which replies with a signature. The signed VUUP is uploaded to the *Vehicle Cloud Service* together with its VCM certificate. *Output: VUPP_n.signed; VCM_cert;*

9. VCM notifies OA. *VCM* notifies *OA*, that the order is ready and supplies a signed URL to the VUUP file. *Output: VUUP_1..n_url.signed;*

10. OA adds the url to the VIN unique VUUP in the Order Cloud Service. The *OA* validates the signature of url and thereafter adds the url in the *Order Cloud Service*.

11. CDA requests status. The *CDA* pulls status from the *Order Cloud Service* (via the *CA*) to indicate that updates are available for download via the signed VUUP_n_url. If no updates yet are available, the signed url will be empty.

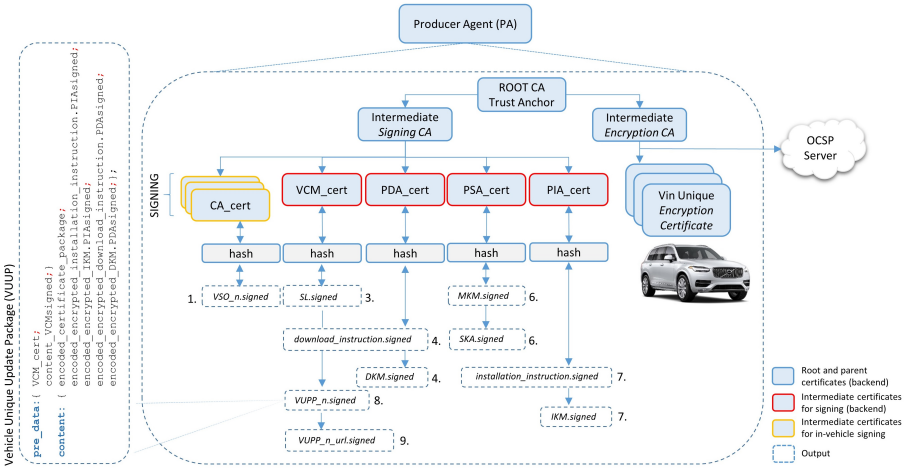

Figure 5.3: Data distribution in the backend in relation to signing

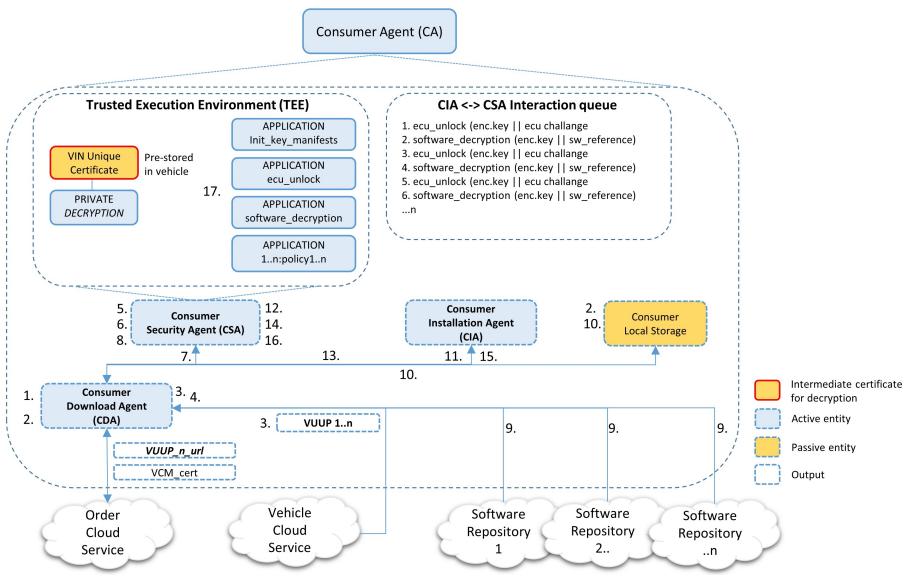

Figure 5.4: Data distribution to the vehicle

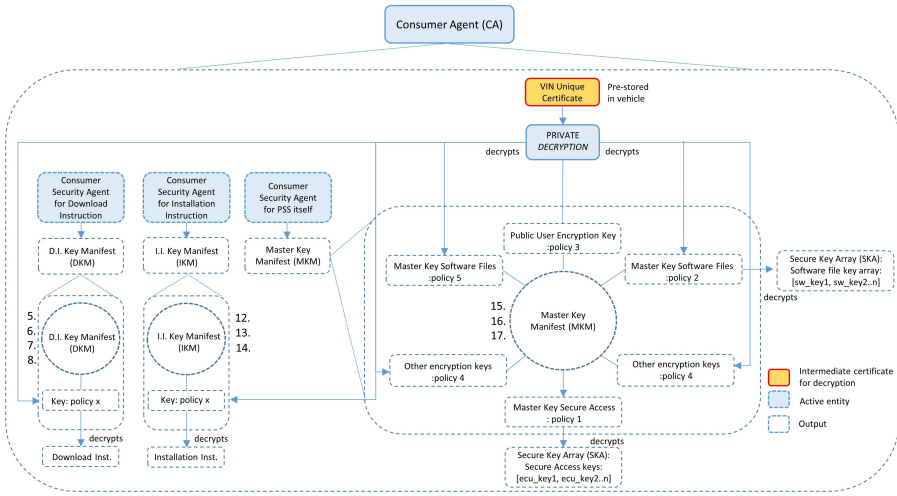

Figure 5.5: Data distribution to the vehicle in relation to cryptographic material

5.4.2 Decapsulating the VUUP file

Consumer Agent (CA): data distribution to vehicle. For the *CA*, the process can be considered as the *PA* process reversed. The 17 steps described below are indicated by numbers where relevant in the Figures 5.4, 5.5 and 5.6.

1. The CDA requests software updates.
Input: VUUP_n_url.signed; *VCM_cert;*
If there are updates available, the CDA receives a signed VUUP_n_url and VCM_cert, where the certificate is validated for authenticity towards the *Root CA* and *OCSP*, where after the signature of VUUP_n_url is validated using the received VCM_cert.

2. Download of VUUP. Mutual authentication is performed towards the *Vehicle Cloud Service* and the signed VUUP_n is downloaded to *Consumer Local Storage*. *Output: VUPP_n.signed;*

3. Validate VUUP. The signature of VUUP_n is validated with VCM_cert, and VUUP_n is decapsulated to produce the signed contents of download instructions, DKM, installation instructions, IKM as well as the included PDA_cert and PIA_cert. *Output:*
download_instruction.signed; DKM.signed; installation_instruction.signed; IKM.signed; PDA_cert; PIA_cert;

4. Validate data within VUPP. Certificates are fetched for online cases or retrieved from the VUUP file for offline cases. The signatures for the download instructions and the *DKM* are validated with the PDA_cert, and the signatures for the installation instructions and the *IKM* are validated with the PIA_cert.

5. DKM Key manifest initiation. The *CDA* requests the *CSA* to initialize the *DKM*, by providing the *DKM.signed* and PDA_cert.
Output: DKM.signed; PDA_cert;

6. CSA associates DKM with TEE application. The signature of *DKM* is validated with PDA_cert. The *DKM* session key is decrypted with the pre-stored vehicle unique private certificate and associated with the TEE application according to the policy in the *DKM* manifest, i.e., for decrypting

download instructions.

7. Request decryption of download instruction. The *CDA* provides the signed download instructions to the *CSA* and requests decryption.
Output: download_instruction.signed;

8. Perform decryption of download instruction. The *CSA* validates signature of the download instruction with PDA_cert, decrypts with the DKM session key from the *DKM* in accordance with policy (i.e., decrypting download instructions) and returns the decrypted download instructions to the *CDA*.
Output: download_instruction;

9. Download of software files. The *CDA* performs mutual authentication towards various software repository sources and downloads encrypted signed software files to *Consumer Local Storage* using the download instructions. The *CDA* validates signatures of all encrypted software files with the VCM_cert.

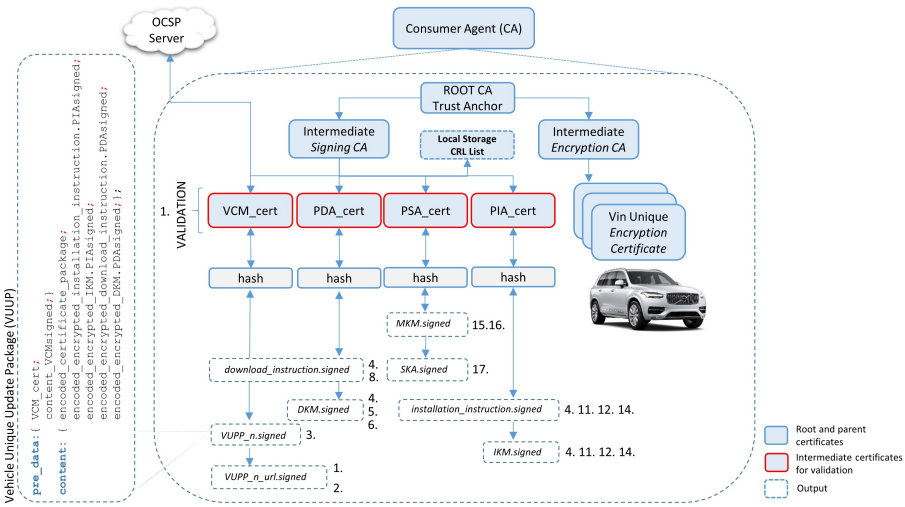

Figure 5.6: Data distribution to the vehicle in relation to validation

Consumer Agent (CA): data execution in vehicle. After data distribution to the vehicle has been completed, the following steps describe the installation of the software update through data execution in the vehicle. These steps can be performed completely offline.

10. Initiation of pre-state phase. The *CDA* requests to start installation of software by sending the signed installation instructions, signed IKM, and the PIA_cert to the *CIA*.
Output: installation_instruction.signed; IKM.signed; PIA_cert;

11. Reboot to secure state. The *CIA* validates the PIA_cert for authenticity towards *Root CA* and *OCSP*. The signatures of the installation instructions and *IKM* are then validated with the PIA_cert. *CIA* then reboots to an offline secure state; ready for pre-state installation processes. PIA_cert is validated again after reboot, against *Root CA* and an offline CRL list; and the signature of IKM is validated with PIA_cert, where after the *CIA* requests *IKM* initialization by sending the signed IKM and PIA_cert to the *CSA*. *Output: IKM.signed; PIA_cert;*

12. IKM key manifest initiation. The *CSA* validates the PIA_cert for

authenticity towards *Root CA* and an offline *CRL*. The *CSA* then validates the *IKM* signature with PIA_cert, where after the IKM session key within *IKM* is decrypted with the pre-stored private asymmetric unique key and associated according to policy in the *IKM*, i.e., to be used for decrypting installation instructions.

13. Request of decryption of installation instruction. The *CIA* provides the signed installation instructions to the *CSA* and requests decryption. *Output: installation_instruction.signed;*

14. Decryption of installation instruction. The *CSA* validates the signature of the installation instructions with PIA_cert, decrypts with the IKM session key from the *IKM* in accordance with policy (i.e., decrypting installation instructions) and returns the decrypted installation instructions to the *CIA*. *Output: installation_instruction;*

15. Request MKM key manifest initiation. The *CIA* retrieves the encapsulated signed *MKM* and *SKA*, and PSA_cert from the decrypted installation instructions. The *CIA* validates the PSA_cert for authenticity towards *Root CA* and an offline *CRL*, and verifies signatures of the *MKM* and the *SKA* with the PSA_cert. *CIA* then requests *MKM* initialization by sending the signed *MKM* and PSA_cert to the *CSA*. *Output: MKM.signed; PSA_cert;*

16. MKM key manifest initiation. The *CSA* validates the *MKM* signature with the PSA_cert. MKM session keys within the *MKM* are decrypted with the pre-stored private asymmetric unique key and are associated with applications according to policy in the *MKM*.

17. Secure CIA - CSA interface established. Peri-state. The *CIA* - *CSA* communication interface is now initialized. The *CIA* can request decryption of software from the *CSA* by sending the encrypted file (or path/link) together with the corresponding encrypted sw_key retrieved from the *SKA*. Before decryption can start, the *CSA* validates the authenticity of software files with the VCM_cert and aborts the decryption request from the *CIA* if this fails. On the other hand, if it is successful, the *CSA* then decrypts the encrypted sw_key with the MKM session key for software files from the *MKM* and uses the sw_key to decrypt the software file. This interface is also used for unlocking ECUs using, e.g., security access to authorize the update. In this case, the challenge from the ECU is sent to the *CSA* together with the corresponding encrypted security access key from the array in *SKA*. *CSA* decrypts the encrypted security access key and processes the challenge from the ECU and can provide the results to the *CIA*. This approach allows the *CIA* to perform ECU unlock without exposing the security access key outside of *CSA*. The *CIA* is after this step ready to stream out software to the ECU.

5.4.3 Post-State Activities

CSA needs to have the possibility to sign post-state installation data, such as installation reports and logs potentially affecting upcoming software updates. A unique session signing key can be transported via *MKM* to *CSA* which can handle signing requests within a trusted application isolated within the *TEE*. The corresponding validation key can be stored in *CMS*. Part of post-state is to perform a complete vehicle software version request (readout). To provide authenticity, the readout can be signed by supported ECUs (e.g., if

they contain pre-stored private keys) and validated by *CSA* with the help of the corresponding validation keys attached to the *SKA*. These responses are then attached to the installation report. To provide confidentiality, *CSA* can encrypt installation reports and logs by using keys in *SKA*.

5.5 Implementation Considerations

The *CA* (all consumer entities) can as shown in Figure 5.4 be located in an ECU in the vehicle used for over-the-air updates or in a client workshop computer with a separated *CSA*. *CSA* in this case can be implemented in a hardware security device such as a Yubico key [152] or a smart card, with a pre-stored encryption/decryption certificate and a pre-stored *Root CA* acting as a trust anchor for validating certificates. Vehicle manufacturers can provide these hardware security devices to workshops, and also have full control to manage and revoke them. Depending on both security and performance requirements *CSA* can also be placed in a workshop HSM or even located in the cloud. Because of the proposed entity separation (implemented as modules), other approaches are also possible, such as integrating *CDA* and *CIA* in an update tool with *CSA* integrated in hardware or separated. It is also possible to use *CDA* separately (outside the vehicle) and securely push the update package to the in-vehicle *CIA* (e.g., via mutually authenticated communication). *CIA* then validates and executes the installation instructions and uses the ECU-internal *CSA* to perform secure transfer and execution of cryptographic operations. The different entities can be securely containerized out in the cloud or kept within vehicle manufacture premises. This solution fulfills the constraints and conditions stated in Section 5.2 and is therefore highly adaptable to accommodate various scenarios within the automotive industry.

5.6 Related Work

Samuel et al. suggest using a layered approach with the use of different roles and keys called *The Update Framework (TUF)* to ensure the integrity of the downloaded data, however, this approach does not consider the installation of these updates and is not adapted for vehicles [153]. In [147] T. Kuppusamy, propose an implementation and adaption of the TUF framework named *Uptane* for vehicle over-the-air updates, where the authors add more metadata to improve its resilience. Another approach proposed by Idress et al. [154] suggests deploying a new architecture where all in-vehicle ECUs use HSMs for over-the-air updates. In [155] Mahmud et al. propose an architecture that relies on sending multiple copies to secure the software update, an approach which we believe is not realistic due to infrastructure constraints. M. Steger et al. propose a framework named *SecUp* which uses handheld devices to wirelessly connect and update vehicles over an IEEE 802.11s mesh network for local environments (i.e. factory and workshop) [156]. In [157] Nilsson et al. present an approach for securing firmware updates over-the-air by combining encryption, hashing, and signing of firmware by chaining fragments. In [158] Nilsson et al. continue their work on hash-chain verification and suggest an over-the-air update framework that validates firmware after it is flashed to the ECUs, however, this requires all

in-vehicle ECUs to be adapted to this approach and that integrity verification of the download is solved by other means.

However, the aforementioned solutions lack necessary details for a *unified* and *versatile* approach that supports updates over-the-air; from a workshop computer; at the factory production; use of diagnostic update tools; and third-party vehicle platform users e.g., using the vehicle as a base controlled by other autonomous systems. As a case in point, *Uber* is using the *Volvo Cars* platform in their fleet of cars [150]; a scenario which most likely will become more prevalent in the future due to increased sustainability requirements. Thus, solutions such as ride-sharing will probably be more common where collective fleets of cars require integrating 3rd party hardware and software which are dependent on a unified software update framework. UniSUF supports 3rd party components by appending its related data to the VUUP file i.e., adding additional instructions and at the same time keeping the VUUP file intact. Moreover, other details are missing in current solutions such as required installation instructions including handling of necessary pre-, peri- and post-state phases and secure transport and secure execution of ECU-specific cryptographic keys. UniSUF considers all these three states, and ensures a secure transport to a trusted execution environment, following a secure execution for all sensitive data. Many existing solutions also consider changes to all ECUs which usually is not possible; something which is not required by UniSUF. The mentioned versatility of UniSUF can keep required adaptions of the vehicle as well as the required cost to a minimum.

5.7 Future Work and Conclusion

We have contributed with a comprehensive and novel unified software update framework named UniSUF, well aligned with industry stakeholders. As part of future work, we have already begun defining an attacker model and started a security analysis of our proposed solution. We aim to perform a more detailed evaluation of UniSUF, including a discussion on the fulfilled requirements as well as a comparison to other approaches, in a future paper.

UniSUF is made to accommodate various scenarios for the automotive domain by encapsulating needed data into one single file, a Vehicle Unique Update Package (VUUP). This vehicle unique file can be processed within a vehicle ECU, using a workshop computer, at factory production, with a diagnostic update tool, or in other compositions. Moreover, the complete update process can be performed without any external communication dependencies, since all files are inherently secured. A continuous secure software update process is a prerequisite for facilitating vehicle resilience towards cyber attacks in a rapidly changing environment. We believe our contributions in this paper can facilitate further research in this area, towards securing the connected car.

Bibliography

[1] Volvo Cars, "Volvo cars - global newsroom," https://www.media.volvocars.com/, 2021, accessed: 2021-12-17.

[2] K. Strandberg, T. Rosenstatter, R. Jolak, N. Nowdehi, and T. Olovsson, "Resilient shield: Reinforcing the resilience of vehicles against security threats," in *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, 2021, pp. 1–7.

[3] United Nations Economic Commission for Europe (UNECE), "UN Regulation No. 156 - Software update and software update management system," https://unece.org/transport/documents/2021/03/standards/un-regulation-no-156-software-update-and-software-update, 2021, accessed: 2021-06-02.

[4] UNECE, "Draft recommendation on cyber security of the task force on cyber security and over-the-air issues of UNECE wp.29 GRVA," UNECE, Tech. Rep., 2018.

[5] "ISO/SAE 21434 Road Vehicles – Cybersecurity Engineering," International Organization for Standardization (ISO), Standard, 2020.

[6] I. O. for Standardization, "Road vehicles Software update engineering," https://www.iso.org/standard/77796.html, 2021, accessed: 2021-06-02.

[7] T. Rosenstatter, K. Strandberg, R. Jolak, R. Scandariato, and T. Olovsson, "REMIND: A framework for the resilient design of automotive systems," *IEEE Secure Development*, 2020, in press.

[8] National Institute of Standards and Technology, "Approaches for federal agencies to use the cybersecurity framework."

[9] K. Strandberg, T. Olovsson, and E. Jonsson, "Securing the connected car: A security-enhancement methodology," *IEEE Vehicular Technology Magazine*, vol. 13, no. 1, pp. 56–65, 2018.

[10] K. Strandberg, D. Kengo Oka, and T. Olovsson, "Unisuf: a unified software update framework for vehicles utilizing isolation techniques and trusted execution environments," in *19th escar Europe : The World's Leading Automotive Cyber Security Conference*, pp. 86–100.

[11] T. Llansó and M. McNeil, "Estimating software vulnerability counts in the context of cyber risk assessments," in *HICSS*, 2018.

[12] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Comprehensive experimental analyses of automotive attack surfaces," in *USENIX Security Symposium*. San Francisco, 2011.

[13] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno *et al.*, "Experimental security analysis of a modern automobile," in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 447–462.

[14] The Industrial Control Systems Cyber Emergency Response Team (ISC-CERT), "Alert (ICS-ALERT-15-203-01)," https://us-cert.cisa.gov/ics/alerts/ICS-ALERT-15-203-01, 2015, accessed: 2021-11-22.

[15] R. Baldwin, "OwnStar car hacker can remotely unlock BMWs, Benz and Chrysler, engadget)," http://www.engadget.com/2015/08/13/ownstar-hack/, 2015, accessed: 2021-11-22.

[16] Microsoft Corporation, "The STRIDE Threat Model," https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20), 2009, accessed: 2021-11-22.

[17] A. Ruddle, D. Ward, B. Weyl, S. Idrees, Y. Roudier, M. Friedewald, T. Leimbach *et al.*, "Deliverable D2.3: Security requirements for automotive on-board networks based on dark-side scenarios," E-safety vehicle intrusion protected applications (EVITA), Deliverable, 2009.

[18] M. Islam, A. Lautenbach, C. Sandberg, and T. Olovsson, "A risk assessment framework for automotive embedded systems," in *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security - CPSS 16*. Association for Computing Machinery (ACM), 2016.

[19] M. Rosenquist and T. Casey, "Prioritizing information security risks with threat agent risk assessment (tara)," 12 2009.

[20] K. Strandberg, "Avoiding Vulnerabilities in Connected Cars," https://publications.lib.chalmers.se/records/fulltext/238172/238172.pdf, 2016, accessed: 2021-11-22.

[21] S. Harris, *CISSP All-in-One Exam Guide, Seventh Edition*. McGraw-Hill Education, 2016.

[22] OffSec Services Limited, "Kali Tools," https://www.kali.org/tools/, 2021, accessed: 2021-11-22.

[23] Hak5, "Kali Tools," https://www.wifipineapple.com, 2021, accessed: 2021-11-22.

[24] K.-T. Cho and K. G. Shin, "Error handling of in-vehicle networks makes them vulnerable," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 10441055. [Online]. Available: https://doi.org/10.1145/2976749.2978302

[25] Vector, "Testing ECUs and Networks with CANoe," https://www.vector.com/se/en-se/products/products-a-z/software/canoe/, 2021, accessed: 2021-11-22.

[26] "NISTIR 7628 Rev 1 – Guidelines for smart grid cybersecurity," National Institute of Standards and Technology, Tech. Rep., Sep. 2014. [Online]. Available: https://doi.org/10.6028/NIST.IR.7628r1

[27] "The Guidelines on Cyber Security Onboard Ships," BIMCO, CLIA, ICS, INTERCARGO, INTERMANAGER, INTERTANKO, IUMI, OCIMF and WORLD SHIPPING COUNCIL, Tech. Rep., Dec. 2017. [Online]. Available: https://www.ics-shipping.org/docs/default-source/resources/safety-security-and-operations/guidelines-on-cyber-security-onboard-ships.pdf

[28] "Cyber Security and Resilience of smart cars," The European Union Agency for Network and Information Security (ENISA), Tech. Rep., 2016.

[29] "SAE J3061: Cybersecurity Guidebook for Cyber-Physical Vehicle Systems," SAE International, Standard, 2016.

[30] UNECE, "TFCS-09-14 Draft Recommendation on Cyber Security of the Task Force on CyberSecurity and Over-the-air issues of UNECE WP.29 IWG ITS/AD," 2017.

[31] J.-C. Laprie, "From dependability to resilience," in *38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks*, 2008, pp. G8–G9.

[32] J. P. Sterbenz, D. Hutchison, E. K. etinkaya, A. Jabbar, J. P. Rohrer, M. Schller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer Networks*, vol. 54, no. 8, pp. 1245 – 1265, 2010, resilient and Survivable networks. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128610000824

[33] O. Andersson, "The Car - A Computer on Wheels," URL: https://www.icse2018.org/getImage/orig/The+Car+%E2%80%93+computer+on+wheels.pdf, May 2018, visited on 2020-05-21.

[34] V. Chang, M. Ramachandran, Y. Yao, Y.-H. Kuo, and C.-S. Li, "A resiliency framework for an enterprise cloud," *International Journal of Information Management*, vol. 36, no. 1, pp. 155 – 166, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S026840121500095X

[35] S. Hukerikar and C. Engelmann, "Resilience design patterns: A structured approach to resilience at extreme scale," *arXiv preprint arXiv:1708.07422*, 2017.

[36] R. Ross, V. Pillitteri, R. Graubart, D. Bodeau, and R. McQuaid, "Developing cyber resilient systems:: a systems security engineering approach," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST SP 800-160v2, Nov. 2019. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v2.pdf

[37] D. Ratasich, F. Khalid, F. Geissler, R. Grosu, M. Shafique, and E. Bartocci, "A Roadmap Toward the Resilient Internet of Things for Cyber-Physical Systems," *IEEE Access*, vol. 7, pp. 13 260–13 283, 2019.

[38] J. P. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Redundancy, diversity, and connectivity to achieve multilevel network resilience, survivability, and disruption tolerance invited paper," *Telecommunication Systems*, vol. 56, no. 1, pp. 17–31, 2014.

[39] M. Segovia, A. R. Cavalli, N. Cuppens, and J. Garcia-Alfaro, "A study on mitigation techniques for scada-driven cyber-physical systems (position paper)," in *Foundations and Practice of Security*, N. Zincir-Heywood, G. Bonfante, M. Debbabi, and J. Garcia-Alfaro, Eds. Cham: Springer International Publishing, 2019, pp. 257–264.

[40] Z. Bakhshi, G. Rodriguez-Navas, and H. Hansson, "Dependable Fog Computing: A Systematic Literature Review," in *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2019, pp. 395–403.

[41] I. Egwutuoha, D. Levy, B. Selic, and S. Chen, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," *The Journal of Supercomputing*, vol. 65, no. 3, pp. 1302–1326, 2013.

[42] P. Kumari and P. Kaur, "A survey of fault tolerance in cloud computing," *Journal of King Saud University - Computer and Information Sciences*, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1319157818306438

[43] M. A. Mukwevho and T. Celik, "Toward a smart cloud: A review of fault-tolerance methods in cloud systems," *IEEE Transactions on Services Computing*, pp. 1–1, 2018.

[44] Vidar Slåtten, Peter Herrmann, and Frank Alexander Kraemer, "Chapter 4 - model-driven engineering of reliable fault-tolerant systemsa state-of-the-art survey," in *Advances in Computers*, ser. Advances in Computers, A. Memon, Ed. Elsevier, 2013, vol. 91, pp. 119 – 205. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780124080898000045

[45] D. Wanner, A. Trigell, L. Drugge, and J. Jerrelind, "Survey on fault-tolerant vehicle design," *World Electric Vehicle Journal*, vol. 5, no. 2, p. 598609, Jun 2012. [Online]. Available: http://dx.doi.org/10.3390/wevj5020598

[46] E. Bartocci and Y. Falcone, *Lectures on Runtime Verification: Introductory and Advanced Topics*. Springer, Cham, 2018, vol. 10457.

[47] D. Heffernan, C. Macnamee, and P. Fogarty, "Runtime verification monitoring for automotive embedded systems using the ISO 26262 functional safety standard as a guide for the definition of the monitored properties," *IET Software*, vol. 8, no. 5, pp. 193–203, 2014.

[48] M. Müter, A. Groll, and F. C. Freiling, "A structured approach to anomaly detection for in-vehicle networks," in *2010 Sixth International Conference on Information Assurance and Security*, 2010, pp. 92–98.

[49] N. Nowdehi, W. Aoudi, M. Almgren, and T. Olovsson, "CASAD: CAN-Aware Stealthy-Attack Detection for In-Vehicle Networks," *arXiv preprint arXiv:1909.08407*, 2019.

[50] M. Hanselmann, T. Strauss, K. Dormann, and H. Ulmer, "Canet: An unsupervised intrusion detection system for high dimensional can bus data," *IEEE Access*, vol. 8, pp. 58 194–58 205, 2020.

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

[52] K.-T. Cho and K. G. Shin, "Viden: Attacker identification on in-vehicle networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS 17. New York, NY, USA: Association for Computing Machinery, 2017, p. 11091123. [Online]. Available: https://doi.org/10.1145/3133956.3134001

[53] M. Husk, J. Komrkov, E. Bou-Harb, and P. eleda, "Survey of attack projection, prediction, and forecasting in cyber security," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 640–660, 2019.

[54] H. Kopetz, *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.

[55] F. Gustafsson, "Particle filter theory and practice with positioning applications," *IEEE Aerospace and Electronic Systems Magazine*, vol. 25, no. 7, pp. 53–82, 2010.

[56] L. Chen and A. Avizienis, "N-version programming: A fault-tolerance approach to reliability of software operation," in *Proc. 8th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-8)*, vol. 1, 1978, pp. 3–9.

[57] J.-C. Laprie, J. Arlat, C. Beounes, and K. Kanoun, "Definition and analysis of hardware-and software-fault-tolerant architectures," *Computer*, vol. 23, no. 7, pp. 39–51, 1990.

[58] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser, "N-Variant Systems: A Secretless Framework for Security through Diversity," in *15th USENIX Security Symposium*, 2006, pp. 105–120.

[59] A. Höller, T. Rauter, J. Iber, and C. Kreiner, "Towards dynamic software diversity for resilient redundant embedded systems," in *Software Engineering for Resilient Systems*, A. Fantechi and P. Pelliccione, Eds. Cham: Springer International Publishing, 2015, pp. 16–30.

[60] T. Dagan, Y. Montvelisky, M. Marchetti, D. Stabili, M. Colajanni, and A. Wool, "Vehicle safe-mode, concept to practice limp-mode in the service of cybersecurity," *SAE Int. J. Transp. Cyber. & Privacy 2 (2)*, feb 2020.

[61] T. Ishigooka, S. Otsuka, K. Serizawa, R. Tsuchiya, and F. Narisawa, "Graceful degradation design process for autonomous driving system," in

Computer Safety, Reliability, and Security, A. Romanovsky, E. Troubitsyna, and F. Bitsch, Eds. Cham: Springer International Publishing, 2019, pp. 19–34.

[62] A. Reschka, G. Bagschik, S. Ulbrich, M. Nolte, and M. Maurer, "Ability and skill graphs for system modeling, online monitoring, and decision support for vehicle guidance systems," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, 2015, pp. 933–939.

[63] J. Rubio-Hernan, R. Sahay, L. De Cicco, and J. Garcia-Alfaro, "Cyber-physical architecture assisted by programmable networking," *Internet Technology Letters*, vol. 1, no. 4, p. e44, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/itl2.44

[64] Z. Jiang, N. C. Audsley, and P. Dong, "BlueVisor: A Scalable Real-Time Hardware Hypervisor for Many-Core Embedded Systems," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018, pp. 75–84.

[65] P. Alho and J. Mattila, "Service-oriented approach to fault tolerance in cpss," *Journal of Systems and Software*, vol. 105, pp. 1 – 17, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121215000643

[66] M. Wu, H. Zeng, C. Wang, and H. Yu, "INVITED: Safety guard: Runtime enforcement for safety-critical cyber-physical systems," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[67] L. F. Cmbita, J. Giraldo, A. A. Crdenas, and N. Quijano, "Response and reconfiguration of cyber-physical control systems: A survey," in *2015 IEEE 2nd Colombian Conference on Automatic Control (CCAC)*, 2015, pp. 1–6.

[68] Y. Zhang and J. Jiang, "Bibliographical review on reconfigurable fault-tolerant control systems," *Annual Reviews in Control*, vol. 32, no. 2, pp. 229 – 252, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1367578808000345

[69] M. Möstl, J. Schlatow, R. Ernst, N. Dutt, A. Nassar, A. Rahmani, F. J. Kurdahi, T. Wild, A. Sadighi, and A. Herkersdorf, "Platform-Centric Self-Awareness as a Key Enabler for Controlling Changes in CPS," *Proceedings of the IEEE*, vol. 106, no. 9, pp. 1543–1567, 2018.

[70] T. D. Nya, S. C. Stilkerich, and C. Siemers, "Self-aware and self-expressive driven fault tolerance for embedded systems," in *2014 IEEE Symposium on Intelligent Embedded Systems (IES)*, Dec 2014, pp. 27–33.

[71] S. Zeadally, T. Sanislav, and G. D. Mois, "Self-Adaptation Techniques in Cyber-Physical Systems (CPSs)," *IEEE Access*, vol. 7, pp. 171 126–171 139, 2019.

[72] D. Weyns, *Software Engineering of Self-adaptive Systems*. Cham: Springer International Publishing, 2019, pp. 399–443. [Online]. Available: https://doi.org/10.1007/978-3-030-00262-6_11

[73] H. Zhang, B. Huang, P. Zhang, and H. Ju, "A New SoS Engineering Philosophy - Vitality Theory," in *2019 14th Annual Conference System of Systems Engineering (SoSE)*, May 2019, pp. 19–24.

[74] G. Vachtsevanos, B. Lee, S. Oh, and M. Balchanos, "Resilient design and operation of cyber physical systems with emphasis on unmanned autonomous systems," *Journal of Intelligent & Robotic Systems*, vol. 91, no. 1, pp. 59–83, 2018.

[75] R. de Lemos, H. Giese, H. A. Müller, and M. Shaw et al., *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–32. [Online]. Available: https://doi.org/10.1007/978-3-642-35813-5_1

[76] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive CAN networksPractical examples and selected short-term countermeasures," *Reliability Engineering & System Safety*, vol. 96, no. 1, pp. 11 – 25, 2011, special Issue on Safecomp 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832010001602

[77] S. Lee, W. Choi, J. H. J., and D. H. Lee, "T-Box: A Forensics-Enabled Trusted Automotive Data Recording Method," *IEEE Access*, vol. 7, pp. 49 738–49 755, 2019.

[78] H. Mansor, K. Markantonakis, R. N. Akram, K. Mayes, and I. Gurulian, "Log your car: The non-invasive vehicle forensics," in *2016 IEEE Trustcom/BigDataSE/ISPA*, 2016, pp. 974–982.

[79] D. K. Nilsson and U. E. Larson, "Conducting forensic investigations of cyber attacks on automobile in-vehicle networks," in *Proceedings of the 1st International Conference on Forensic Applications and Techniques in Telecommunications, Information, and Multimedia and Workshop*, ser. e-Forensics 08. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[80] W. Bortles, S. McDonough, C. Smith, and M. Stogsdill, "An introduction to the forensic acquisition of passenger vehicle infotainment and telematics systems data," SAE Technical Paper, Tech. Rep., 2017.

[81] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial Examples: Attacks and Defenses for Deep Learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[82] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.

[83] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks*, vol. 31, no. 8, pp. 805

– 822, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128698000176

[84] G. Welch and G. Bishop, "An Introduction to the Kalman filter," 1995.

[85] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[86] Burdi Motorworks, "Mercedes Limp Home Mode," https://burdimotors.com/2017/11/30/mercedes-limp-home-mode, Accessed 2020-03-16, 2018. [Online]. Available: https://burdimotors.com/2017/11/30/mercedes-limp-home-mode

[87] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*. San Francisco, 2011.

[88] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, vol. 2015, 2015.

[89] G. Macher, E. Armengaud, E. Brenner, and C. Kreiner, "Threat and risk assessment methodologies in the automotive domain," *Procedia Computer Science*, vol. 83, pp. 1288–1294, 2016.

[90] O. Henniger, L. Apvrille, A. Fuchs, Y. Roudier, A. Ruddle, and B. Weylr, "Security requirements for automotive on-board networks," in *2009 9th International Conference on Intelligent Transport Systems Telecommunications, (ITST)*. Institute of Electrical and Electronics Engineers (IEEE), oct 2009.

[91] T. Rosenstatter and T. Olovsson, "Towards a Standardized Mapping from Automotive Security Levels to Security Mechanisms," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2018, pp. 1501–1507.

[92] V. H. Le, J. den Hartog, and N. Zannone, "Security and privacy for innovative automotive applications: A survey," *Computer Communications*, vol. 132, pp. 17 – 41, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S014036641731174X

[93] D. Bodeau and R. Graubart, "Cyber Resiliency Engineering Framework (MITRE Technical Report MTR1-10237)," *Bedford, MA: MITRE Corporation*, 2011.

[94] B. Baudry and M. Monperrus, "The multiple facets of software diversity: Recent developments in year 2000 and beyond," *ACM Comput. Surv.*, vol. 48, no. 1, Sep. 2015. [Online]. Available: https://doi.org/10.1145/2807593

[95] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, *Microservices: Yesterday, Today, and Tomorrow*. Cham: Springer International Publishing, 2017, pp. 195–216. [Online]. Available: https://doi.org/10.1007/978-3-319-67425-4_12

[96] C. Engelmann, G. R. Vallee, T. Naughton, and S. L. Scott, "Proactive Fault Tolerance Using Preemptive Migration," in *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, 2009, pp. 252–257.

[97] R. Romagnoli, B. H. Krogh, and B. Sinopoli, "Design of Software Rejuvenation for CPS Security Using Invariant Sets," in *2019 American Control Conference (ACC)*, 2019, pp. 3740–3745.

[98] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, 2015.

[99] Tencent Keen Security Lab, "Experimental Security Assessment of BMW Cars: A Summary Report," https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Assessment_of_BMW_Cars_by_KeenLab.pdf, 2018, accessed: 2020-09-11.

[100] S. Kamkar, "Drive it like you hacked it: New attacks and tools to wirelessly steal cars," *Presentation at DEFCON*, vol. 23, 2015.

[101] CVE Details, "Security vunerabilities bluelink," https://www.cvedetails.com/vulnerability-list/vendor_id-16402/product_id-37376/Hyundaiusa-Blue-Link.html, accessed: 2020-09-11.

[102] CVE List, "CVE-2019-9493," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9493, accessed: 2020-09-11.

[103] "ISO/SAE 21434 Road Vehicles – Cybersecurity Engineering," International Organization for Standardization (ISO), Standard, 2020.

[104] "Good practices for security of smart cars," ENISA, Tech. Rep., 2019.

[105] "Cyber security and resilience of smart cars," ENISA, Tech. Rep., 2016.

[106] "SAE J3061: Cybersecurity guidebook for cyber-physical vehicle systems," SAE International, Standard, 2016.

[107] EVITA, "EVITA deliverables," https://www.evita-project.org/deliverables.html, accessed: 2020-09-11.

[108] M. Islam, A. Lautenbach, C. Sandberg, and T. Olovsson, "A risk assessment framework for automotive embedded systems," *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, 2016.

[109] Microsoft, "The STRIDE threat model," https://msdn.microsoft.com/en-us/library/ee823878.aspx, 2005, accessed: 2020-09-11.

[110] T. Rosenstatter and T. Olovsson, "Towards a standardized mapping from automotive security levels to security mechanisms," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 1501–1507.

[111] A. Karahasanovic, P. Kleberger, and M. Almgren, "Adapting threat modeling methods for the automotive industry," *15th ESCAR, Berlin*, 2017.

[112] "ISO 26262:2011 Road Vehicles – Functional Safety," International Organization for Standardization (ISO), Standard, 2011.

[113] F. D. Garcia, D. Oswald, T. Kasper, and P. Pavlidès, "Lock it and still lose iton the (in) security of automotive remote keyless entry systems," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016.

[114] A. Greenberg, "Just a pair of these $11 radio gadgets can steal a car," https://www.wired.com/2017/04/just-pair-11-radio-gadgets-can-steal-car/, accessed: 2020-09-11.

[115] A. Francillon, B. Danev, and S. Capkun, "Relay attacks on passive keyless entry and start systems in modern cars," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. ETH Zürich, Department of Computer Science, 2011.

[116] M. L. Psiaki and T. E. Humphreys, "GNSS spoofing and detection," *Proceedings of the IEEE*, vol. 104, no. 6, pp. 1258–1270, 2016.

[117] K. Iehira, H. Inoue, and K. Ishida, "Spoofing attack using bus-off attacks against a specific ECU of the CAN bus," in *15th IEEE Consumer Communications & Networking Conference (CCNC)*, 2018, pp. 1–4.

[118] Q. Meng, L. Hsu, B. Xu, X. Luo, and A. El-Mowafy, "A GPS spoofing generator using an open sourced vector tracking-based receiver," *Sensors*, vol. 19, no. 18, p. 3993, 2019.

[119] CVE List, "CVE-2019-12797," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-12797, accessed: 2020-09-11.

[120] Y. Cao, C. Xiao, B. Cyr, Y. Zhou, W. Park *et al.*, "Adversarial sensor attack on LiDAR-based perception in autonomous driving," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 22672281.

[121] Cyware Hacker News, "Seven car manufacturers hit by GPS spoofing attacks," https://cyware.com/news/seven-car-manufacturers-hit-by-gps-spoofing-attacks-146701c4, accessed: 2020-09-11.

[122] Help Net Security, "Research shows Tesla Model 3 and Model S are vulnerable to GPS spoofing attacks," https://www.helpnetsecurity.com/2019/06/19/tesla-gps-spoofing-attacks/, accessed: 2020-09-11.

[123] CVE, "CVE-2018-11478," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-11478, accessed: 2020-09-11.

[124] D. Schmidt, K. Radke, S. Camtepe, E. Foo, and M. Ren, "A survey and analysis of the GNSS spoofing threat and countermeasures," *ACM Comput. Surv.*, vol. 48, no. 4, May 2016. [Online]. Available: https://doi.org/10.1145/2897166

[125] Pen Test Partners, "Hacking the Mitsubishi Outlander PHEV hybrid," https://www.pentestpartners.com/security-blog/hacking-the-mitsubishi-outlander-phev-hybrid-suv/, accessed: 2020-09-11.

[126] J. Petit, B. Stottelaar, M. Feiri, and F. Kargl, "Remote attacks on automated vehicles sensors: Experiments on camera and lidar," *Black Hat Europe*, vol. 11, p. 2015, 2015.

[127] C. Yan, W. Xu, and J. Liu, "Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicle," *DEF CON*, vol. 24, no. 8, p. 109, 2016.

[128] Tencent Keen Security Lab, "Experimental security assessment on lexus cars," https://keenlab.tencent.com/en/2020/03/30/Tencent-Keen-Security-Lab-Experimental-Security-Assessment-on-Lexus-Cars/, 2020, accessed: 2020-09-15.

[129] P. Murvay and B. Groza, "Practical security exploits of the FlexRay in-vehicle communication protocol," *International Conference on Risks and Security of Internet and Systems*, pp. 172–187, 2019.

[130] Argus Cyber Security, "A remote attack on the Bosch Drivelog connector dongle," https://argus-sec.com/remote-attack-bosch-drivelog-connector-dongle/, accessed: 2020-09-11.

[131] CVE List, "CVE-2016-9337," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-9337, accessed: 2020-09-11.

[132] S. Woo, H. J. Jo, and D. H. Lee, "A practical wireless attack on the connected car and security protocol for in-vehicle can," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 993–1006, 2015.

[133] M. Yan, J. Li, and G. Harpak, "Security Research on Mercedes-Benz: From Hardware to Car Control," https://i.blackhat.com/USA-20/Thursday/us-20-Yan-Security-Research-On-Mercedes-Benz-From-Hardware-To-Car-Control.pdf, 2020, accessed: 2020-09-15.

[134] G. H. Ruffo, "Tesla Data Leak: Old Components With Personal Info Find Their Way On eBay," https://insideevs.com/news/419525/tesla-data-leak-personal-info-ebay/, accessed: 2020-09-11.

[135] CVE List, "CVE-2018-11477," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-11477, accessed: 2020-09-11.

[136] J. Liu, S. Zhang, W. Sun, and Y. Shi, "In-vehicle network attacks and countermeasures challenges and future directions," *IEEE Network*, 2017.

[137] J. C. Norte, "Hacking industrial vehicles from the internet," http://jcarlosnorte.com/security/2016/03/06/hacking-tachographs-from-the-internets.html, accessed: 2020-09-11.

[138] A. Palanca1, E. Evenchick, F. Maggi, and S. Zanero, "A stealth, selective, link-layer denial-of-service attack against automotive networks," *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2017.

[139] P. Murvay and B. Groza, "DoS attacks on controller area networks by fault injections from the software layer," *Proceedings of the 12th International Conference on Availability, Reliability and Security*, 2017.

[140] CVE List, "CVE-2016-2354," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-2354, accessed: 2020-09-11.

[141] K. Cho and K. Shin, "Error handling of in-vehicle networks makes them vulnerable," *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[142] J. Dürrwang, J. Braun, M. Rumez, and R. Kriesten, "Security evaluation of an airbag-ECU by reusing threat modeling artefacts," in *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2017, pp. 37–43.

[143] T. Brewster, "BMW updates kills bug in 2.2 million cars that left doors wide open to hackers," https://www.forbes.com/sites/thomasbrewster/2015/02/02/bmw-door-hacking/, 2015, accessed: 2020-09-11.

[144] T. Hunt, "Controlling vehicle features of Nissan LEAFs across the globe via vulnerable APIs," https://www.troyhunt.com/controlling-vehicle-features-of-nissan/, 2016, accessed: 2020-09-11.

[145] S. Sanwald, L. Kaneti, M. Stttinger, and M. Bhner, "Secure boot revisited," *17th escar Europe*, 2019.

[146] *MISRA C: Guidelines for the Use of the C Language in Critical Systems 2012*. Motor Industry Research Association, 2013.

[147] T. Karthik, A. Brown, S. Awwad, D. McCoy, R. Bielawski *et al.*, "Uptane: Securing software updates for automobiles," *14th ESCAR Europe*, 2016.

[148] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks*, vol. 31, no. 8, pp. 805 – 822, 1999.

[149] C. G. Rieger, D. I. Gertman, and M. A. McQueen, "Resilient control systems: Next generation design research," in *2009 2nd Conference on Human System Interactions*, 2009, pp. 632–636.

[150] Volvo Car Corporation, "Volvo Cars and Uber present first autonomous drive-ready production car," https://group.volvocars.com/news/future-mobility/2019/volvo-and-uber-present-autonomous-drive-ready-xc90, 2019, accessed: 2021-06-02.

[151] Volvo Cars, "Volvo Cars teams up with worlds leading mobility technology platform DiDi for self-driving test fleet," https://www.media.volvocars.com/global/en-gb/media/pressreleases/280668/volvo-cars-teams-up-with-worlds-leading-mobility-technology-platform-didi-for-self-driving-test-flee, 2021, accessed: 2021-06-07.

[152] yubico, "Protect your digital world with YubiKey," https://www.yubico.com/, 2021, accessed: 2021-06-02.

[153] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, "Survivable key compromise in software update systems," 12 2010, pp. 61–72.

[154] 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 *Communication Technologies for Vehicles*, T. Strang, A. Festag, A. Vinel, R. Mehmood, C. Rico Garcia, and M. Röckl, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 224–238.

[155] S. Mahmud, S. Shanker, and I. Hossain, "Secure software upload in an intelligent vehicle via wireless communication links," in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, 2005, pp. 588–593.

[156] M. Steger, C. A. Boano, T. Niedermayr, M. Karner, J. Hillebrand, K. Roemer, and W. Rom, "An efficient and secure automotive wireless software update framework," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2181–2193, 2018.

[157] D. K. Nilsson and U. E. Larson, "Secure firmware updates over the air in intelligent vehicles," in *ICC Workshops - 2008 IEEE International Conference on Communications Workshops*, 2008, pp. 380–384.

[158] D. K. Nilsson, L. Sun, and T. Nakajima, "A framework for self-verification of firmware updates over the air in vehicle ecus," in *2008 IEEE Globecom Workshops*, 2008, pp. 1–5.