CHALMERS

Testing and Evaluation to Improve Data Security of Automotive Embedded Systems

Master's thesis in Computer Systems & Networks

JOHANNES WESCHKE
FILIP HESSLUND

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2015
Master's thesis 2011:01

Testing and Evaluation to Improve Data Security of Automotive Embedded Systems
JOHANNES WESCHKE
FILIP HESSLUND

Master's thesis 2011:01
ISSN 1652-8557
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Chalmers Reproservice
Göteborg, Sweden 2015

Abstract

In the last two decades, the number of electronic control units (ECUs) in vehicles has increased dramatically. This has resulted in an increased complexity of the vehicles electrical and electronic systems. Electrical and electronic systems have gone from just controlling the engine to controlling every part of the vehicle, from the infotainment system to safety-critical systems.

To allow for better collaboration between players in the automotive industry, a development partnership called AUTOSAR has emerged. Included in AUTOSAR is a module handling diagnostics (DCM). The module can be used to read data and change parameters in the ECUs and in the ECU software, since the DCM can access confidential information about the vehicle and modify running software of the ECU, for example the software controlling the engine, it is an attractive target for adversaries. There has been no published research about the security of the DCM module of the AUTOSAR software architecture (that we know of) and how the safety of the passengers can be affected in the case of a security breach. This thesis tries to fill this research gap by conducting a threat analysis and risk assessment for the DCM module inside AUTOSAR. This thesis evaluates the security of an ECU assumed to control the engine of a vehicle and how possible consequences of an intrusion can affect the overall safety. It also presents a number of tests used to evaluate the threats and risks found. The tests done targets threats regarding denial of service, tampering, and information disclosure. The thesis is concluded with proposing countermeasures for the threats and risks.

Keywords: security, automotive, testing, evaluation, AUTOSAR, ISO 14229

Acknowledgements

We would like to thank Tomas Olovsson, at Chalmers Univerity of Technology, and Christian Sandberg and Mafijul Islam, from Volvo AB, for supervising this thesis, and for their invaluable help and input during this work. We would also like to thank Erland Jonsson, at Chalmers University of Technology, for being our examiner.

Contents

1 Introduction

The automotive industry has in the last two decades gone through a large development regarding the computerization of the vehicle's embedded systems. The systems have gone from only controlling parameters in the engine to controlling almost all parts of the vehicle. The on-board computers, *electronic control units* (ECUs), have come to include everything from the infotainment system to controlling safety-critical systems.

The increasing number of ECUs and software components forming these new services has led to a rising complexity of the vehicles electrical and electronic systems. To decrease this problem and allow for better collaboration between players in the automotive industry, a development partnership called AUTOSAR has emerged.

AUTOSAR is a software architecture that is based on modular components to form a complete system. This allows developers to focus on and develop one component at a time.

At the bottom of the architecture is the *Basic Software Layer* (BSW) which acts like a base to build service specific software on, see Figure 1.1. The BSW includes the standardized software such as the Operating System, the Memory Manager and the Diagnostic module that are necessary for the system to work.

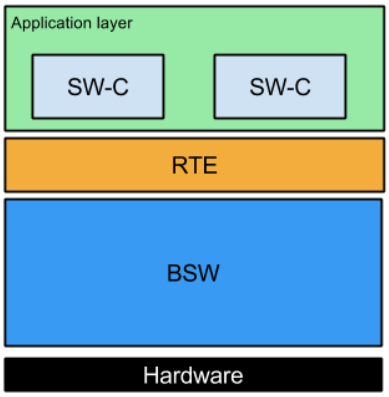

Figure 1.1: *The layered AUTOSAR architecture*

The Diagnostic module in the BSW layer is responsible for the communication between the vehicle software and the manufacturer. The Diagnostic module is used to manage, update and reprogram the components that are running inside the ECU, which makes the module an attractive target for adversaries. If unauthorized access is granted, an adversary could be able to reprogram the software and possibly endanger the safety of the vehicle's passengers.

This thesis aims to evaluate the security of the standardized Diagnostic module of AUTOSAR, leading to the following research questions.

- **What vulnerabilities can be found in the AUTOSAR diagnostics Basic Software module?**
- **What are the possible threats and risks of these vulnerabilities?**

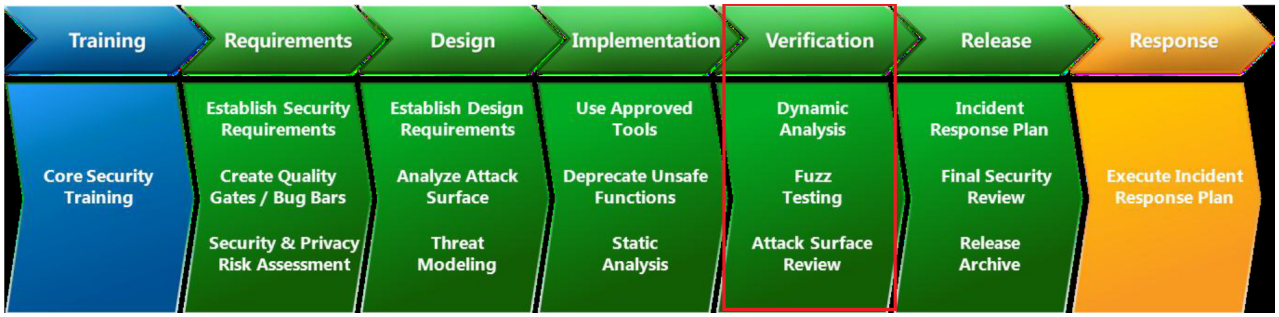

Figure 1.2: *Simplified Microsoft Secure Development Lifecycle. [23]*

- **Can the vulnerabilities be mitigated and the security be improved?**

This thesis is a part of the HEAVENS project that aims to create a common strategy to identify vulnerabilities and define methodologies to evaluate security inside vehicles electrical and electronics system [31].

Previous parts of the project have included creating a threat model and declaring a common view of how to address security (see Section 3.1), and this thesis follows the HEAVENS terminology [14].

This thesis investigates the security of the AUTOSAR architecture Diagnostic Module, it evaluates how security vulnerabilities are related to threats and safety risks. Proof of concept security tests of the found vulnerabilities are shown, and as such this thesis fits into the Verification step of Microsofts Secure Development Lifecycle (SDL) (see Figure 1.2).

1.1 Related Work

Vehicular security has gained some focus in recent years, and much academic research has been done on the subject. Many security protocol implementations have been suggested to secure the onboard network of a vehicle. Both software and hardware modules have been shown to mitigate the security vulnerabilities of the advanced electronic devices present in today's vehicles [11]. Most of the research done has been to reduce the threat of an external adversary, but less research has been done on vulnerabilities exploited by an internal adversary. A typical example is where a vehicle owner who exploits vulnerabilities to gain access to restricted features of the vehicle, such as more engine power, without paying for the feature.

In *Securing Vehicle Diagnostics in Repair Shops*, Kleberger and Olovsson [19] describe suggestions on how to secure both wired and wireless remote diagnostic inside a repair shop. They suggest that well known security protocols for secure data transmission over LAN:s and internet (such as IPsec, SLL, Certificates and PKI) can be used to secure remote vehicle diagnostics. They also conclude that IPsec is the most desirable security architecture to achieve secure diagnostics and that it additionally includes the possibility to perform secure diagnostics over the Internet. Their focus is on the communication with an ECU, whereas our thesis focuses on vulnerabilities inside the ECUs diagnostics module.

Koscher et al. [20] wrote a widely cited paper in 2010 that included multiple frightening scenarios of the lack of security in vehicles at that time. Their work included defining a threat model, and an experimental analysis on both stationary and moving targets.

Bayer et al. [4] wrote a paper providing insight on some motivations and possible attacks on automotive systems, as well as a theoretical evaluation of penetration testing techniques. We feel that their work misses a more practical approach, however, some of their ideas will be the foundation for our testing.

Vyleta [42] presented a thesis in which they did some security fuzz-testing on a simulated ECU. The tests were conducted on only two services available in the DCM: `WriteMemory` and `RequestDownload`. Our thesis presents tests made on hardware using a real implementation, and includes more services.

This shows that researchers have been aware of the problem with security in the automotive industry for quite some time, but new standards still doesn't include security in a complete way. As pointed out by Kleberger and Olovsson [19] some existing standards have even removed previously present security "best-practices".

1.2 Research Methodology

The methodology for this thesis is described in Figure 1.3. The process starts with defining the problem and the objectives to follow throughout the project. At this stage, we decided that the area of investigation would be to evaluate the security of the diagnostic module in the AUTOSAR architecture.

In the second step, a threat model of the area was created which resulted in a list of threats that the module could be vulnerable to. These results were then used as input to the third step, where a risk assessment was performed. A number of attack trees were created based on the risk assessment, and these attack trees inspired the tests made in the next step.

In the fifth step a number of security tests were performed to evaluate the threats that were found in the threat modeling, to see if the system was vulnerable to these kinds of threats. These security tests were performed in two environments, a white-box and a grey-box environment. In the white-box environment the tests were performed and evaluated on a self configured system where all configurations were known. The tests were then applied on a grey-box production system environment. After tests were performed in both environments, the result of each test was evaluated, discussed and suggestions of countermeasures were done.

At the final step of the process all the results were documented and further work was discussed.

This process follows the HEAVENS security model workflow (see Figure 3.2) in parts, but instead of the last step of identifying security requirements we swerve off onto a path of testing the systems if the threats identified can be targeted. Since the threat analysis only identifies threats conceptually this is a good approach.

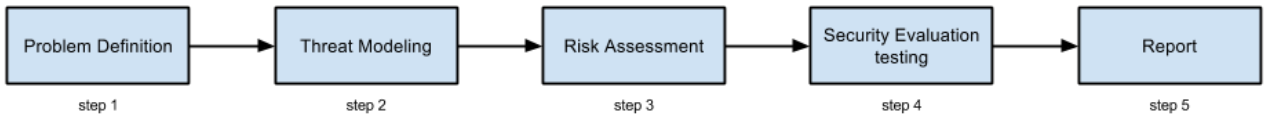

Figure 1.3: *Methodology*

1.3 Thesis outline

The rest of this thesis is structured as follows: chapter 2 contains a list of terms and definitions used, chapter 3 introduces security concepts, chapter 4 treats basic concepts used later on in the thesis. Chapter 5 contains the threat model of the system on which the tests were made, chapter 6 presents the tests made. Chapter 7 presents results and an evaluation of them. Chapters 8 and 9 contains a discussion and our conclusions.

2 Terms and Definitions

Here follows a list of terms and definitions used in the HEAVENS project [14].

Access	Establish logical or physical communication or otherwise interact with **assets** and/or **TOEs**.
Asset	Anything of value to **stakeholders**, potential targets of **attacks**.
Attack	Series of actions performed by **attackers** to achieve unauthorized results in relation to **assets** and/or **TOEs**.
Attacker	Any type of individual, group, or entity aiming to mount **attacks**.
Attack surface	Set of resources associated with an **asset** and/or **TOE** that **attackers** can exploit to mount **attacks**.
Attack tree	An attack tree is a graphical representation to show possible **attack vectors** that can be used to mount an **attack**.
Attack vector	Set of actions to establish a path or means to mount an **attack**.
Client	The ECU or appliance requesting services during a diagnostic session.
Harm	Negative impact on **stakeholders** as a result of **attacks**.
Impact level	Estimate of the magnitude of **harm** to **stakeholders** originating from a **threat** and/or an **attack**.
Server	The diagnostic module inside the connected ECU under a diagnostic session.
Stakeholders	Individuals and/or organizations that can affect, be affected by, or perceive themselves to be affected by **attacks**.
Target of Evaluation (ToE)	A set of **assets**.
Threat	Potential cause of an unwanted incident, which may result in harm to a system or organization.
Vulnerability	Weakness in an **asset** and/or **ToE** that can potentially be exploited by **attackers** to mount **attacks**.

Table 2.1: Terms and Definitions.

3 Security model

This chapter introduces the risk assessment and threat analysis processes used in the HEAVENS security model. Sections 3.2 and 3.3 introduces different kinds of penetration testing and fuzz testing techniques used to verify the conceptual findings from the threat analysis.

3.1 The HEAVENS Security Model

The workflow for the HEAVENS security model can be seen in Figure 3.2 and consists of four modules: defining the target of evaluation, threat analysis, risk assessment, and identifying security requirements.

As can be seen in the figure, the first step is to identify the TOE and to find an appropriate level of abstraction to enable the use of a threat analysis tool. The TOE description is then used in the Threat Analysis module. Here, potential threats are mapped to assets and security attributes, according to STRIDE (Section 3.1.1). The next step is the Risk Assessment, that uses the threat & asset mappings, and the threat level- (TL) and impact level- (IL) parameters as input. These can be seen in Tables 3.2 and 3.3. The Risk Assessment step outputs security levels for each of the identified threats. HEAVENS defines security level as: "A measure to estimate the level of risk used to specify countermeasures for assets and/or TOEs to avoid unreasonable risk." The last step is the Security Requirement which take the Security Level output from the Risk Assessment module and the Threats & Security attributes from the Threat Analysis module as input. The Security Requirement module then produces a resulting Security Level value for each asset and threat pair that is based on the threat and impact level. The Security Level result is defined as a value between the lowest level, Quality Management (QM) to the highest level Critical, where QM states that no extra attention is needed from a security point of view. The security levels can be seen in Figure 3.1, and Table 3.1.

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 3.1: *HEAVENS security levels based on the Threat and Impact levels [21].*

3.1.1 STRIDE

The threat analysis method STRIDE [37], developed by Microsoft, threats are categorized depending on the goals and purposes of the attacks. STRIDE is used in the HEAVENS security model threat analysis. The STRIDE model is considered to be threat-centric or attacker-centric, since every threat is linked with a certain asset from the attackers perspective. STRIDE extends the CIA (Confidentiality, Integrity, Availability) model by linking threats to a larger set of security attributes than the ones present in CIA, i.e. *authenticity*, *freshness*, *non-repudiation*, *privacy*, and *authorization*. STRIDE is

named after the first letter in every threat it includes.

- **Spoofing identity** - attackers pretend to be someone or something else - *(Authenticity, Freshness)*
- **Tampering with data** - attackers change data in transit or in a data store - *(Integrity)*
- **Repudiation** - attackers performs actions that cannot be traced back to them - *(Non-repudiation, Freshness)*
- **Information disclosure** - attackers get access to data in transit or in a data store - *(Confidentiality, Privacy)*
- **Denial of service** - attackers interrupt a systems legitimate operation - *(Availability)*
- **Elevation of privilege** - attackers perform actions they are not authorized to perform - *(Authorization)*

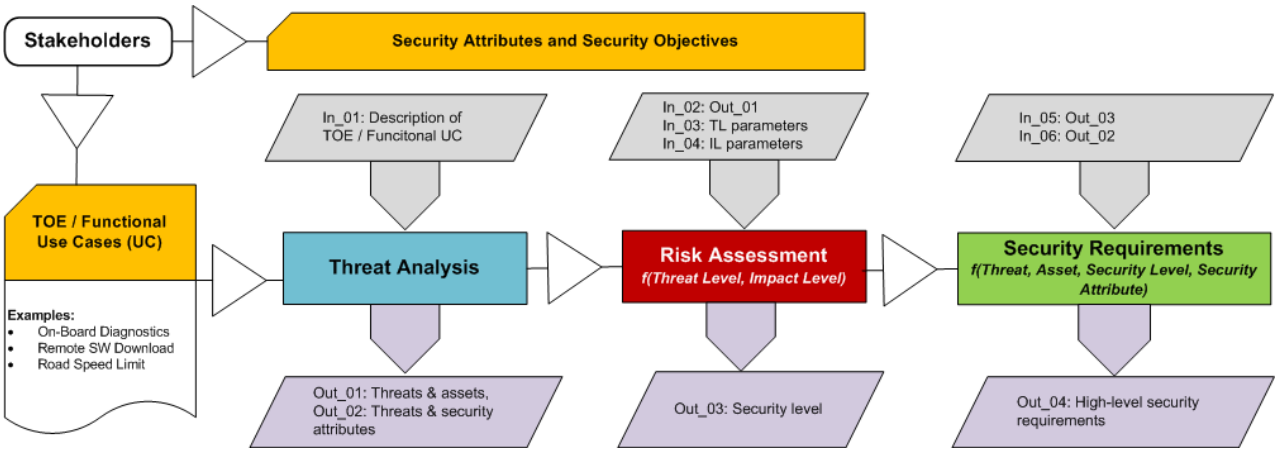

Figure 3.2: *HEAVENS security model workflow [21].*

Security threat severity class	Aspects of security threats			
	Safety	**Privacy**	**Financial**	**Operational**
0	No injuries.	No authorized access to data.	No financial loss.	No impact on operational performance.
1	Light or moderate injuries.	Anonymous data only.	Low-level loss.	Impact not discernible to driver.
2	Severe injuries (survival probable). Light/moderate injuries for multiple vehicles.	Identification of vehicle or driver. Anonymous data for multiple vehicles.	Moderate loss. Low losses for multiple vehicles.	Driver aware of performance degradation. Indiscernible impacts for multiple vehicles.
3	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. Moderate losses for multiple vehicles.	Significant impact on performance. Noticeable impact for multiple vehicles.
4	Life threatening or fatal injuries for multiple vehicles.	Driver or vehicle tracking for multiple vehicles.	Heavy losses for multiple vehicles.	Significant impact for multiple vehicles.

Table 3.1: Threat severity classification scheme proposed by EVITA [33], used in the HEAVENS project.

Impact levels	
Safety	No injury.
	Light and moderate injuries.
	Severe and life-threatening injuries (survival probable).
	Life-threatening injuries (survival uncertain), fatal injuries.
Financial	Financial losses, either direct or indirect.
Operational	Damages caused by unexpected incident, like loss of secondary or comfort/entertainment functionality.
Privacy and legislation	Damages to stakeholders caused by loss of privacy or violation of regulations.

Table 3.2: HEAVENS security model Impact Levels

Threat levels		
Expertise	Layman	- No particular expertise.
	Proficient	- General knowledge about the security field.
	Expert	- Familiar with underlying hardware, security behaviour, and attack methods.
	Multiple Experts	-
Knowledge about TOE	Public	- Available on-line without non-disclosure agreements.
	Restricted	- Controlled within organization with non-disclosure agreements.
	Sensitive	- Constrained to only members of certain teams.
	Critical	- Knowledge known by only a few individuals.
Equipment	Standard	- Available for unregulated purchase, or for free on-line.
	Specialized	- More expensive equipment that can be obtained with little effort.
	Bespoke	- Not readily available to public. Needs to be specially produced.
	Multiple Bespoke	-
Window of opportunity	Low	- Physical access required.
	Medium	- Limited physical, or logical access required.
	High	- Logical access without physical presence.
	Critical	- Logical access without physical presence or time limitation.

Table 3.3: HEAVENS security model Threat Levels

3.2 Penetration testing

Penetration testing is a non-malicious process in which the goal is to find vulnerabilities in a system or a software. This is usually done by analyzing the system using some tool, for example OpenVAS used to analyze network vulnerabilities [27]. Tools like OpenVAS will produce a report specifying what services (and versions of them) are available on the TOE and what vulnerabilities are present for the versions of the services available. When the system has been identified the penetration tester can use known attacks on the systems assets and access points.

The first step of a penetration test is reconnaissance, to determine what interfaces and specifications would be open to a potential attacker. The second step could be to mount small scale attacks on some external interfaces. In a web server one such interface could be an open port, and in an automotive setting it could be the CAN bus [4].

Penetration testing is divided into three main categories, *Black-box*, *White-box* and *Grey-box* depending on the amount of information that the attacker has about the target [24].

White-box testing technique is a technique where the tester has full knowledge of the internal architecture of the system and has full access to the software source code. This technique give a detailed test of the internal logic and the code structure of the system. By testing the system with this approach implementation errors such as poor key and cryptographic algorithms can be revealed.

Black-box is a testing technique where the tester instead has no knowledge about the source code and how the internal system is working. Here the tester only has access to the external interfaces that are available from the software.

Grey-box is a technique where the black- and the white-box techniques are combined. Here the tests are performed without access to the source code but with fundamental knowledge of the system software structure.

The different techniques have different advantages and disadvantages. The White-box technique has the advantage of giving a full coverage of the tested areas of the software and can also find unused code that is still present. This techniques is however expensive and need the tester to be highly skilled of the tested software.

The advantage of the Black-box is that the tester knows less about the system and that the development of tests are quicker. The disadvantages of this approach is however that the testing becomes inefficient and only a limited number of scenarios are covered.

In the Grey-box technique the benefits of both the white- and the black-box approaches are combined. Here the tester can rely on the interface definitions and the system functions rather than focus on gaining a deeper understanding of the source code. In this technique the tester can create well specified test scenarios that are tested from the user, attacker, point of view instead of the developers view.

The Grey-box has the same disadvantage as the black-box, where the coverage of the test are limited as the source code is unavailable when the designing the tests are done. Many data flow path may also be untested due to the same reason.

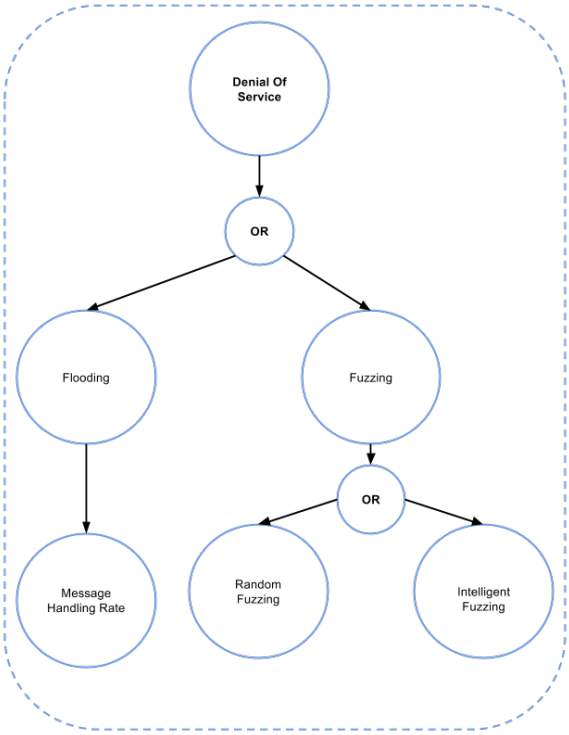

Figure 3.3: *Example attack tree.*

A way to help design proper penetration tests is the use of attack trees. Attack trees were proposed by Bruce Schneier in 1999 as a way of modeling security threats [34]. The attack tree is built with the attacker's goal as the root node, and every branch from the root contains ways of accomplishing that goal (an example attack tree can be seen in Figure 3.3). In this thesis, attack trees were used to come up with sound test cases.

3.3 Fuzzing

Fuzzing is a technique to ensure the correctness of a function or a piece of software. The basic approach is to input random data and see if it is validated in a satisfying way, and does not cause a crash or end up in any unwanted state. The technique is used by both security and quality assurance experts. Fuzzing gives developers the tools to find vulnerabilities triggered by malformed or malicious input [38].
Fuzzing can be considered a subset of penetration testing since the technique might reveal vulnerabilities that may be exploited.

4 AUTOSAR

This chapter introduces concepts that are important to this thesis. It starts by describing the AUTOSAR architecture and then quickly narrows down towards the Diagnostic module and how the communication for the diagnostic module is standardized.

4.1 Automotive Open System Architecture 4.1

The Automotive Open System Architecture (AUTOSAR) is an alliance of vehicle manufacturers and automotive suppliers with the purpose to develop an open industry standard software architecture for the automotive E/E systems. The standardization of automotive electronic development by AUTOSAR aids to remove the previously used OEM specific standards and reduce the need for every supplier to develop their own software architecture. The standardization is intended to reduce the large development cost and remove the need of a large number of tools and communication protocols that was needed by the suppliers earlier. The aim of AUTOSAR is to change the development of new software application from coding based to configuration based development. This allows the developers to focus more on the innovation process of new functions instead of the OEM specific standard configuration coding for each new software.

AUTOSAR provides a layered architecture model to meet these goals and motivations. This architecture includes three abstract layers, the *Application layer*, the *Run-time environment layer* and the *Basic Software layer*. Where the Basic Software forms the infrastructural base and includes the main functionalities to run AUTOSAR 4.1.

The *Software layer* is placed at the top of the architecture and includes the OEM specific Software components (SW-Cs). An SW-C can implement a complete system or just a part of a system function, e.g. a SW-C can include the code controlling the vehicles windscreen wipers or the parameters for tuning the engine. The SW-C can be compared to an application running on a personal computer.

The *Run-time environment* (RTE) is placed underneath the Software layer and controls all the communication between the different Software components. The communication between the SW-Cs is always done through the RTE no matter if the components are located on the same or on separated ECUs inside the vehicle. This gives the system the flexibility to move Software components between different ECUs in the vehicle, and also reusing SW-Cs from other vehicles, without the need of reprogramming the communication paths and the interface of the component.

The *Basic Software layer* is a standardized software layer that provides services to connect the Software components to the functionality of the hardware module. The basic software layer is divided into three main sub layers, the *Service layer*, the *ECU Abstraction layer* and the *Microcontroller Abstraction layer*.

The Service layer provides the basic services that builds the AUTOSAR architecture and includes the vital functions, such as the Operating System, the Communication services, the Memory Management module and the Diagnostic module.

The ECU abstraction layer forms a higher level of interfacing with the drivers in the Microcontroller abstraction layer and include Onboard Device Abstraction, Memory Hardware Abstraction, Communication Hardware Abstraction and a Input/Output Hardware abstraction.

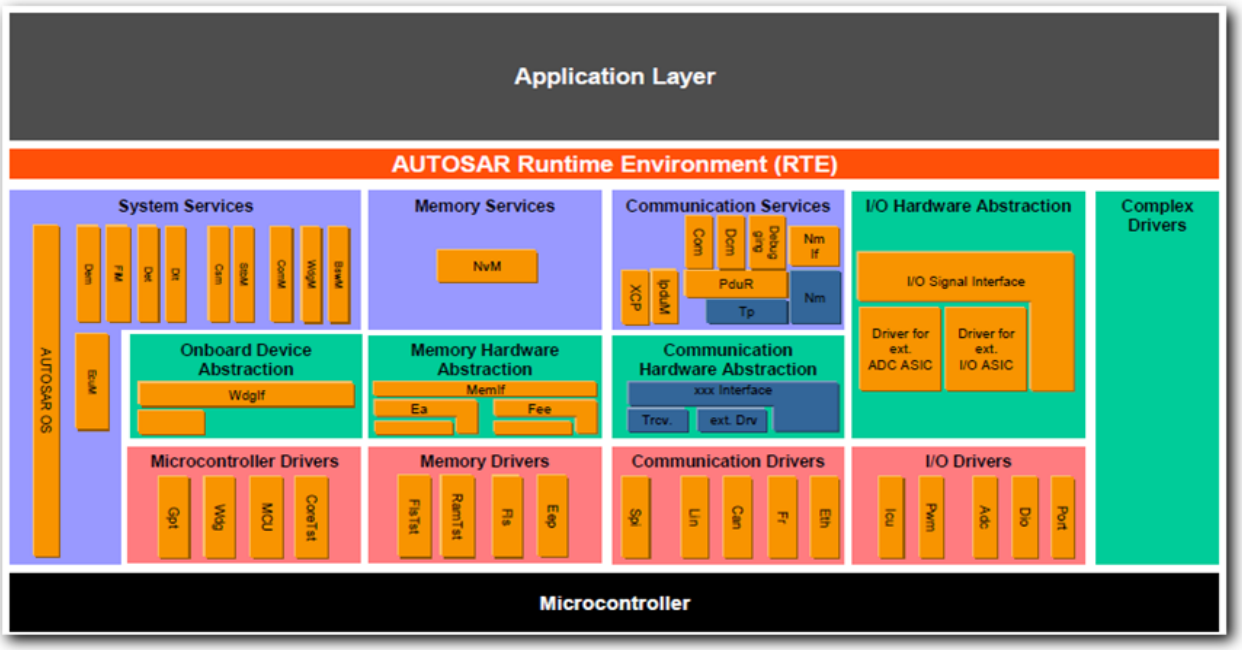

Figure 4.1: *Detailed layers of the AUTOSAR architecture.*

The Microcontroller Abstraction layer is the lowest level of the Basic Software and includes the ECU specific drivers and has direct access to the ECU hardware functionality, see Figure 4.1

Due to these layers the internal implementation of the components are independent of the others as long as they follow the AUTOSAR standardized interfaces. This gives the possibility to develop components separately, that can work together without the exact knowledge of the others internal implementations.

Diagnostic Module

The Diagnostic module is placed in the BSW layer, see Figure 4.2, and allows technicians to interact with the controllers and the software components on-board the vehicle. It allows system information gathering, system configuration and reprogramming of the ECU software and are implemented according to the ISO 14229-1 standard, see Section 4.2.
The AUTOSAR diagnostic stack consists of two main modules, the Diagnostic Communication Manager (DCM) and the Diagnostic Event Manager (DEM), where the DCM manages all diagnostic communication requests and the configured services of the diagnostic module. The DEM module records error events from the SW-Cs and is responsible for storing these in the persistent memory.

Diagnostic Communication Manager - The task of the DCM is to handle the communication between the external diagnostic tool and the diagnostic module inside the ECU. It also implements the necessary management to control that the communication requests are done in the correct sessions states of the module, see Figure 4.3.

The DCM communicates using a client-server model where the external tool acts as a client that sends requests to the ECU that acts as a server, see Figure 4.4. The client is often referred to as the ”tester”.

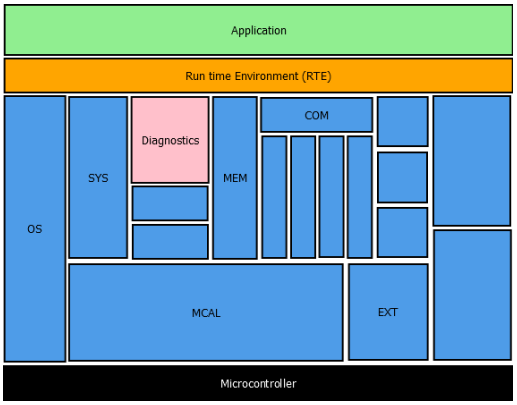

Figure 4.2: *The Diagnostic module*

The DCM module has three main sessions states [8], the *Default Diagnostic Session* state, the *Extended Diagnostic Session* state and the *ECU Programming Session* state while communicating with a tester.

When the vehicle is powered on, the ECU will enter the Default Diagnostic Session state where it is idle and listens for requests. When the DCM receives a connection from a technician and the diagnostic tool, the DCM enter the Extended Diagnostic session state where it is ready to respond to client requests. When an ECU Programming Session is requested, the DCM transits into the third state, where the software can be reprogrammed and updated.

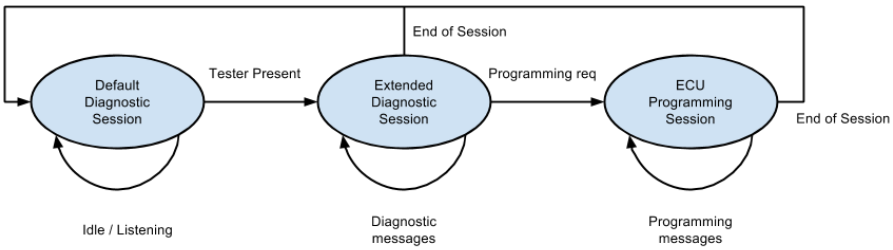

Figure 4.3: *DCM session states*

To communicate the DCM uses the ISO 14229-1 diagnostic protocol standard (see Section 4.2), where the use of Service Id:s (SID), Sub-Service Id:s and Data Identifiers (DID) are implemented to send the incoming diagnostic calls to the correct software components in the Application layer. The communication uses a request-response approach where the tester asks the DCM for information by sending a request and then get a response including the requested information.

The diagnostic tool uses predefined SIDs that are stated by the standard [16] to start the desired service, see Table 4.1. A diagnostic request of reading information from a DID could look like this:
1. The client sends a Service Id followed by the number of the wanted DID.
2. The ECU responds with a message containing the specified information.
See Figure 4.5.

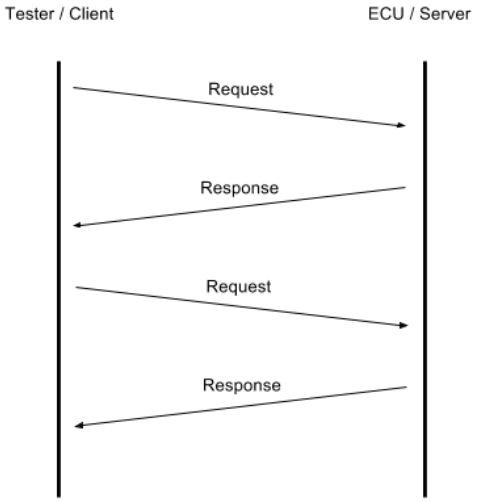

Figure 4.4: *Client Server Communication*

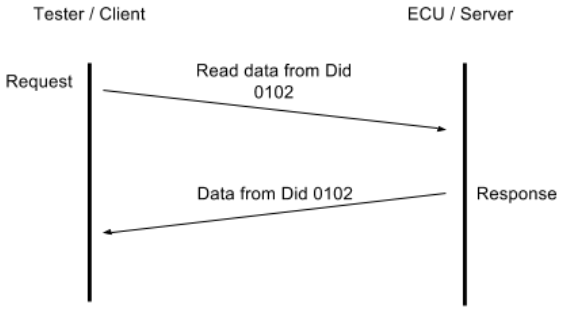

Figure 4.5: *Communication of a* `ReadDataByIdentifier` *request and the response*

4.2 ISO standards for vehicle diagnostic

The International Standard Organisation (ISO) is a non-governmental organization that develops and forms world wide used standards [13]. ISO has developed and is still developing standards such as 14229-1, 15765 and 15764 that aim to standardize the technical development inside the vehicle industry.

ISO 14229-1 Road vehicles - Unified diagnostic services (UDS)

ISO 14229-1 [16] is a standard that defines the automotive diagnostic services in road vehicles. The standard defines a protocol stating how to communicate with the DCM inside an ECU in a vehicle. The standard provides detailed information about the protocol when modifying, programming or

Service Identifier	Service Name	Description
0x10	DiagnosticSessionControl	Used for switching between Default-, Extended and ECU Programming Session.
0x11	EcuReset	Used to reset the ECU.
0x14	ClearDiagnosticInformation	Clear the diagnostic information stored in the ECU memory.
0x19	ReadDTCInformation	Used to read the diagnostic trouble codes.
0x22	ReadDataByIdentifier	Read information stored in the ECU using a Data identifier.
0x24	ReadScalingDataByIdentifier	Used to read the scaling information of a Data identifier.
0x27	SecurityAccess	Used to access security levels.
0x2A	ReadDataByPeriodicIdentifier	Used when Data identifier data want to be read peridocally.
0x2C	DynamicallyDefineDataIdentifier	Is used together with 0x22 to read multiple Did data in one time.
0x2E	WriteDataByIdentifier	To write data to a specific Did.
0x2F	InputOutputControlByIdentifier	Control the input output.
0x31	RoutineControl	Used to start, stop or return the value of a routine.
0x3E	TesterPresent	Is used to keep a none-default session active
0x85	ControlDtcSetting	Opens or close DTC record

Table 4.1: List of known service IDs.

reprogramming the ECUs firmware.

ISO 15765 Diagnostic communication over Controller Area Network (DoCAN)

ISO 15765 specifies the transport protocol and the network layer services, when communicating over CAN, and has been defined in accordance with the diagnostic standard defined in ISO 14229[17].

5 Threat modeling

A threat model of the system (see Section 6.1) was set up in Microsoft Threat Modeling Tool 2014 [36]. The test environment was abstracted to be viewed as a generic process and data store modeling the ECU and RAM. A laptop running diagnostic tests acted as the Tester entity in Figure 5.1. From the generated report a Microsoft Excel file was created in which the different threats were ranked based on their severity, according to the HEAVENS security model (see Section 3.1). The result can be seen in Table 5.1, which includes the threats identified by the modeling tool. Since the tool is not specialized for embedded systems a number of non-applicable threats have been removed from the table, and some threats that were not identified by the tool have been added.

Risk assessment is a subjective process in which the threat and impact levels must be decided upon. The above threat model and risk assessment is done assuming that the ECU under assessment is an engine control ECU, but threat analysis and risk assessment needs to be done for every ECU in the entire system. This leads to that the threats concerning information disclosure will not be as severe as the threats concerning tampering or elevation of privilege. It also results in that the impact level regarding safety can reach the higher levels. This would not be the case if the risk assessment was done on some other ECU e.g. the climate control, or the ECU in control of the electrical window elevators.

The abstraction chosen in this thesis allows for viewing the engine control unit with corresponding diagnostic communication as three parts, the ECU, the tester, and the diagnostic communication between the two. A common thing for every threat listed in Table 5.1 is that an adversary would need access to some restricted information to mount an attack, especially contents in ISO 14229 [16], but the adversary would not require any specialized equipment.

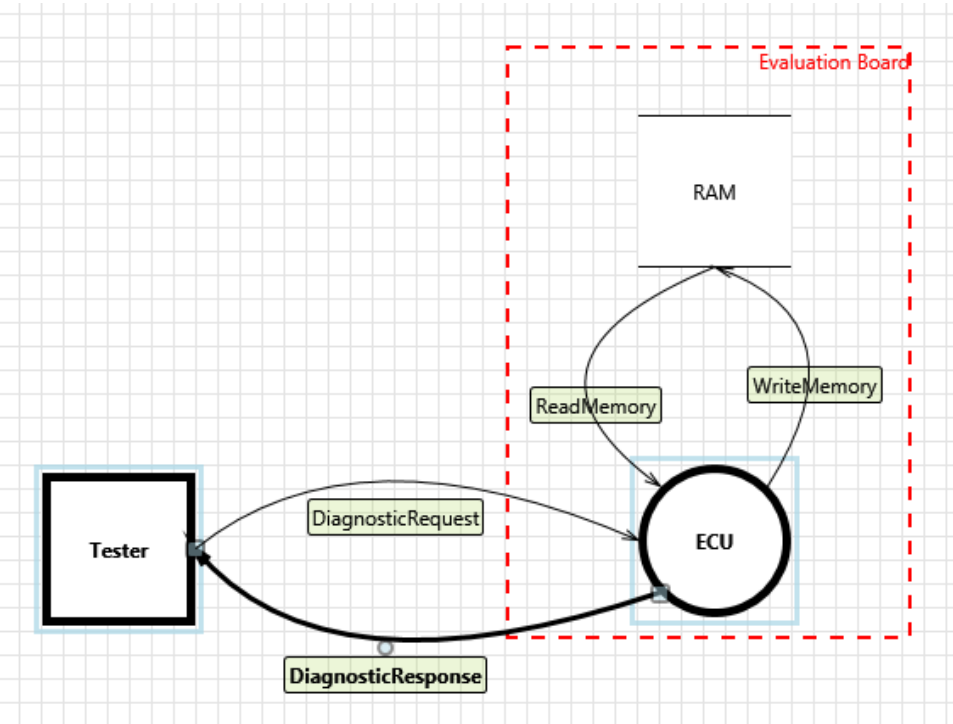

Figure 5.1: *Model created in Microsoft Threat Modeling Tool 2014*

5.1 Diagnostic communication

There are three threats linked to the diagnostic communication: information disclosure, denial of service, and tampering.

Information disclosure From the risk assessment done, it can be seen that information disclosure from the diagnostic communication does not pose a serious risk, even though an adversary does not need any particular skills or non-standard equipment. It receives such a low security level because of the low impact levels, there are no impacts to either safety, privacy, financial, nor operational categories.

Tampering The tampering threat receives a higher security level. The tampering of the diagnostics communication might result in changed engine control parameters that can lead to involuntarily acceleration or other consequences of that sort. Involuntary acceleration can of course lead to safety issues, it might lead to legislative issues for the stakeholders, and it of course can lead to a high operational impact since the driver might lose control of the vehicle.

Denial of service Denial of service of the diagnostic communication could be done by cutting cables, which means that no expertise is necessary for an adversary realize the threat. It would most likely not result in any impact on safety, but it could result in damage to the vehicle. Assume for example that no error messages are shown when oil level was low for a long period of time without any stakeholders knowledge.

5.2 ECU

Five threats are connected to the ECU asset.

Denial of service Denial of service of the ECU results in a high security level. Having the engine control ECU unreachable might lead to the vehicle stopping in unsuitable places and could potentially lead to minor injuries. The operational impact level however is set to high since the engine control will no longer work. The security level reaches a high level since an adversary does not require any specific expertise to realize this threat.

Elevation of privilege A threat concerning elevation of privilege has been identified and ranked high on the security level scale. If an adversary were to gain elevated privileges on the engine control ECU we would see impacts similar to that of the tampering with the diagnostics communication were the adversary could potentially modify engine control parameters resulting in unwanted behaviour of the vehicle and potential harm to persons in and around it.

Information disclosure The information disclosure threat on the ECU asset does not pose any particular risk, it receives a low security level rating. It would most likely not result in any injuries or have any operational impact.

Tampering Tampering with the engine control ECU would require expert knowledge, and includes reprogramming it. If successful, the results would be similar to the tampering of diagnostic communication, with severe injuries and high operational impact. It could also potentially lead to legislative repercussions for the stakeholders and therefore also have financial impact.

Spoofing Spoofing the ECU into thinking the adversary is a legitimate tester could potentially allow the adversary to change engine control parameters. This would have similar impact to tampering with the ECU, or gaining elevated privileges, resulting in severe injuries and operational impact.

5.3 Tester

Only one threat was identified against the tester asset. Spoofing the tester into thinking the adversary is the engine control ECU would most likely not result in any injuries or have an operational impact, but could potentially have legislative and financial impact since measurements from the engine could be faked.

5.4 Security requirements

Assets with special security requirements can be identified from Table 5.1. In this specific case each asset in the system has need of special security requirements, since each asset has at least one threat resulting in a higher than Quality Management (QM) security level [21].
Since this is just a conceptual evaluation, it should be tested in practice as well to see if the threats and risks have been assessed correctly. To do this, a number of tests have been designed, and executed (see Chapter 6).

			Threat level				Impact level			
Asset	**Threat**	**Security Level**	**Expertise**	**Window of opportunity**	**Knowledge about TOE**	**Equipment**	**Safety**	**Privacy & Legislation**	**Financial**	**Operational**
Diagnostic Communication	Information Disclosure	QM	Layman	Medium	Restricted	Standard	No injuries	No impact	No impact	No impact
Diagnostic Communication	Tampering	Medium	Layman	Low	Restricted	Standard	Severe and life-threatening injuries (survival probable)	Medium	Low	High
Diagnostic Communication	Denial of Service	High	Layman	Medium	Restricted	Standard	No injuries	Medium	Low	High
ECU	Denial of Service	High	Layman	Medium	Restricted	Standard	Light and moderate injuries	Low	Low	High
ECU	Elevation of Privilege	High	Proficient	Medium	Restricted	Standard	Severe and life-threatening injuries (survival probable)	Medium	Medium	High
ECU	Information Disclosure	Low	Layman	Medium	Restricted	Standard	No injuries	Low	No impact	No impact
ECU	Tampering	Medium	Expert	Medium	Restricted	Standard	Severe and life-threatening injuries (survival probable)	Medium	Medium	High
ECU	Spoofing	N/A	Proficient	Medium	Restricted	Standard	Severe and life-threatening injuries (survival probable)	Medium	Medium	High
Tester	Spoofing	High	Proficient	Medium	Restricted	Standard	No injuries	Medium	Medium	No impact

Table 5.1: Threats identified by the threat modeling tool and risks assessed.

6 Security Tests

This section describes the tests that were done in this thesis work to evaluate if a system is vulnerable to different security threats and to see what information an attacker performing similar attacks could gain.

The test are describe here and the test results are described in Section 7. The outcome of the test are further described in Section 8.

The security evaluation tests were conducted on two systems. The first system was an evaluation board running AUTOSAR 4.1.1 where we configured and implemented the running services that constitute the complete DCM system. This gave us the advantage of possessing all information about the system. The second test system was a part of an existing production system. This system gave us the ability to evaluate the security of a real system that is used in vehicles today.

The tests to evaluate the security vulnerabilities of the DCM module in the different systems are performed using two approaches. On the evaluation board the tests were conducted in a white-box environment (see Section 3.2). The production system could be seen as a grey-box system (Section 3.2) due to that it complied to the ISO 14229-1 standard [16], which means that some information about how it would respond was already known.

The security tests are divided according to three root goals of an adversary extracted from the threat modeling, see Section 5. The *Information Disclosure / System Reconniassance*, the *Tampering of ECU software* and *Denial of Service* goal. How the tests are related to the adversary goals are described and graphically shown in the following section, see Section 6.2 regarding attack trees.

6.1 Experimental Environment

The tests were created and written as *Python* scripts, that were run on a standard PC connected to the test systems CAN bus interfaces. The CAN bus traffic was monitored by a second PC using BUSMASTER v2.6.0 [5]. The PCs were connected to the CAN bus using a CAN-Case [6] and a PEAK-CAN [29] "CAN bus to USB" converter (see Figure 6.1).
The two test environments are described below.

Evaluation board environment

This environment consisted of a VK-EVB-M3 evaluation board developed by ArcCore [3] running ArcCores open source implementation of AUTOSAR 4.1 (see Section 4.1), ArcticCore v7.0.0 [2]. The DCM implementation was done according to ISO 14229-1 [16]. The boards AUTOSAR implementation was configured using *ArcticStudio* [1], and *winIDEA Open* [44] was used for the programing and the debugging of the hardware.

Production system environment

The production system consisted of a fully running system that was mounted in a lab environment for live testing of new software. The system consisted of an ECU that was connected to a diagnostic client that could run diagnostic test sessions towards the DCM module. To generate an environment that represented the environment of an adversary as realistically as possible, access to the diagnostic client was not allowed during the security testing.

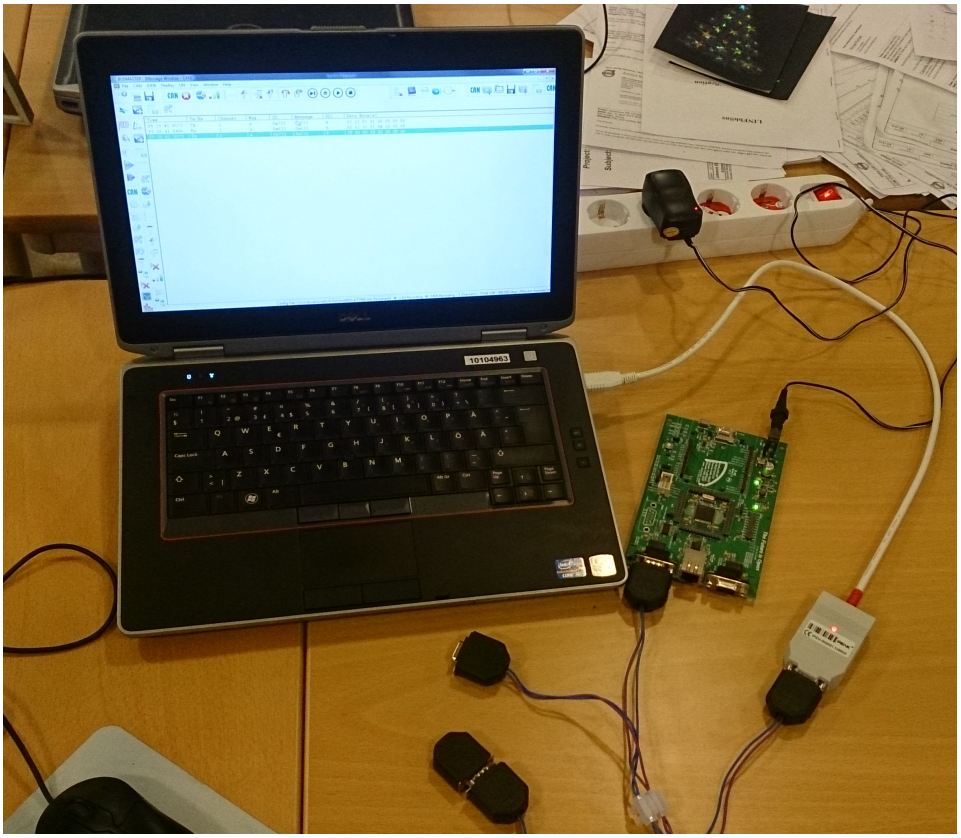

Figure 6.1: *Laptop connected to the evaluation board from ArcCore, using a PEAK CAN to USB adapter.*

6.2 Attack Trees

In this section, attack trees based on the results from the threat modeling of the system are introduced, see Chapter 5. The attack trees gives a graphical representation of possible sequences of events that can be implemented by an adversary to achieve the root goal of an attack [35]. At the root of the trees the main goals of the adversary are placed, then the trees branches shows possible attack vectors to achieve it. At the bottom of the tree the leafs represent possible footholds for the adversary. These footholds are presented as security tests and has been performed throughout this thesis work. The tests are described in the following sections, see Sections 6.3, 6.4, 6.5.

6.2.1 Information Disclosure / System Reconnaissance

This attack tree represent the goal of an adversary trying to retrieve initial information of the system. Getting a view of the configured system is a top priority for an adversary to make qualified attacks and discover vulnerabilities present in the system. This attack tree contains three adverse actions that an adversary can attempt to gain information about the DCM module, see Figure 6.4. Each of the actions can be used separately to disclose partial information, or can be combined to get an almost complete view of the system. The leaves contains four security tests to address the information disclosure threat. The tests are further described in Sections 6.3.1, 6.3.2, 6.3.3.

6.2.2 Tampering of ECU Software

The attack tree of this threat includes two main attack vectors that the adversary can use to tamper with the ECU software, see Figure 6.3. The first vector is the left branch where the software upload of new content is performed, and the second vector is the right branch where an adversary is able to modify the memory of the ECU. At the bottom of the tree the adverse action to be able to compromise the software are placed. The adversary's action leafs are implemented as security tests, and are described in Sections 6.4.1, 6.4.2, 6.4.3, 6.4.4, 6.4.5.

6.2.3 Denial of Service

The Denial of Service threat resulted in a attack tree including two branches. The first branch includes the possibility for an adversary to flood the DCM module, and the other branch covers the possibility to send invalidly formed messages that could potentially result in that the module end up in an unresponsive state. In the first branch an adversary starts by getting information about the request handling rate of the DCM module, see Section 6.3.5. The information is then used to perform a flooding attack, see Section 6.5.4. At the bottom of the second branch the adverse actions is performed through two fuzzing tests, see Section 6.5.2, 6.5.3.

6.3 Information Disclosure / System Reconnaissance Test

This section describes the test derived from the bottom of the Information Disclosure / System Reconnaissance attack tree adversary actions. It describe how the test was performed in this thesis to achieve the root goal at the top of the tree.

6.3.1 System Scan

The scan test is built in three steps, the service identifier-, the sub-service/data identifier- and the input size step. The test takes advantage of the response codes that are specified in ISO 14229-1.

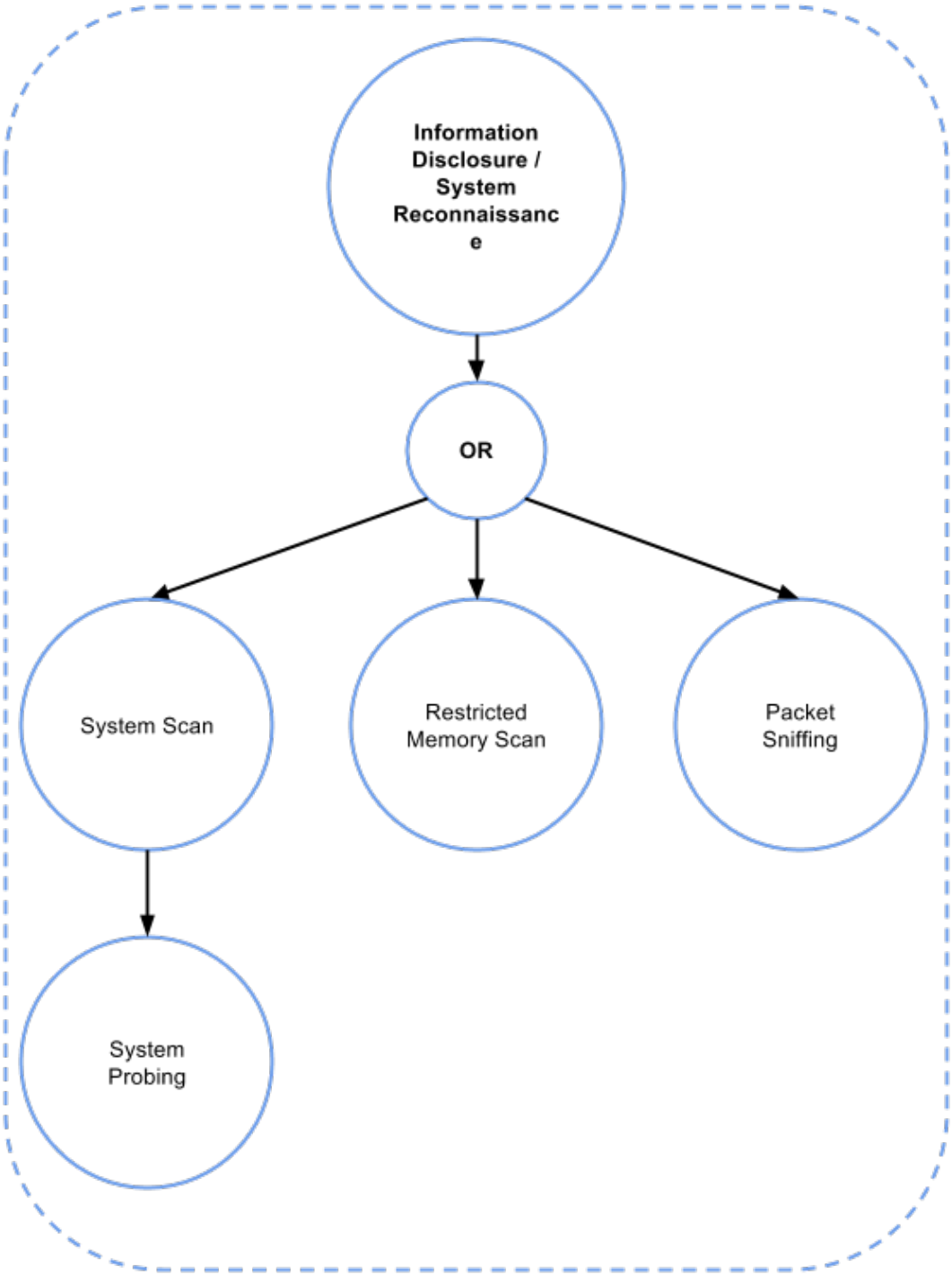

Figure 6.2: *Attack tree of Information Disclosure and System Reconnaissance threat*

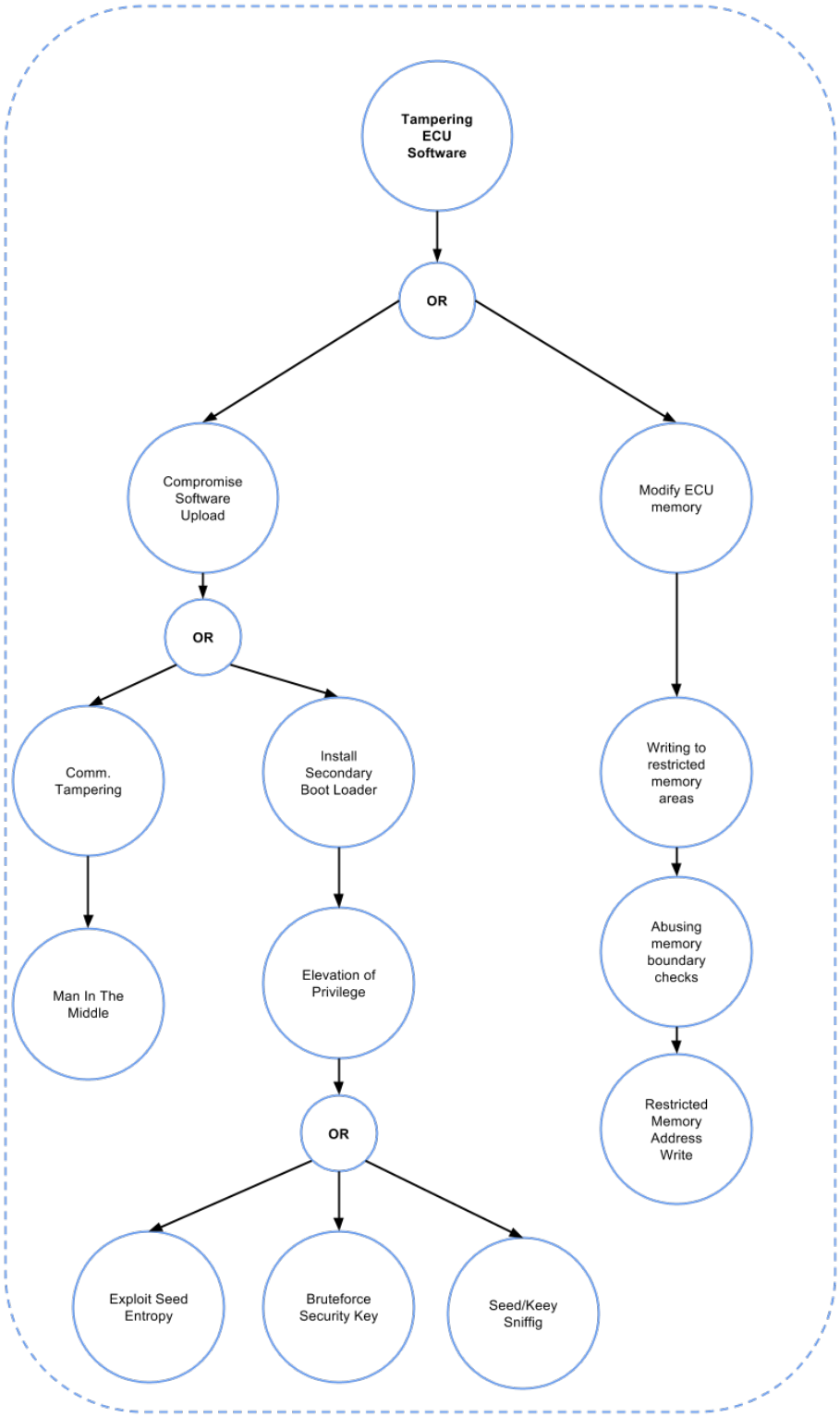

Figure 6.3: *Attack tree for Compromising the ECU Software*

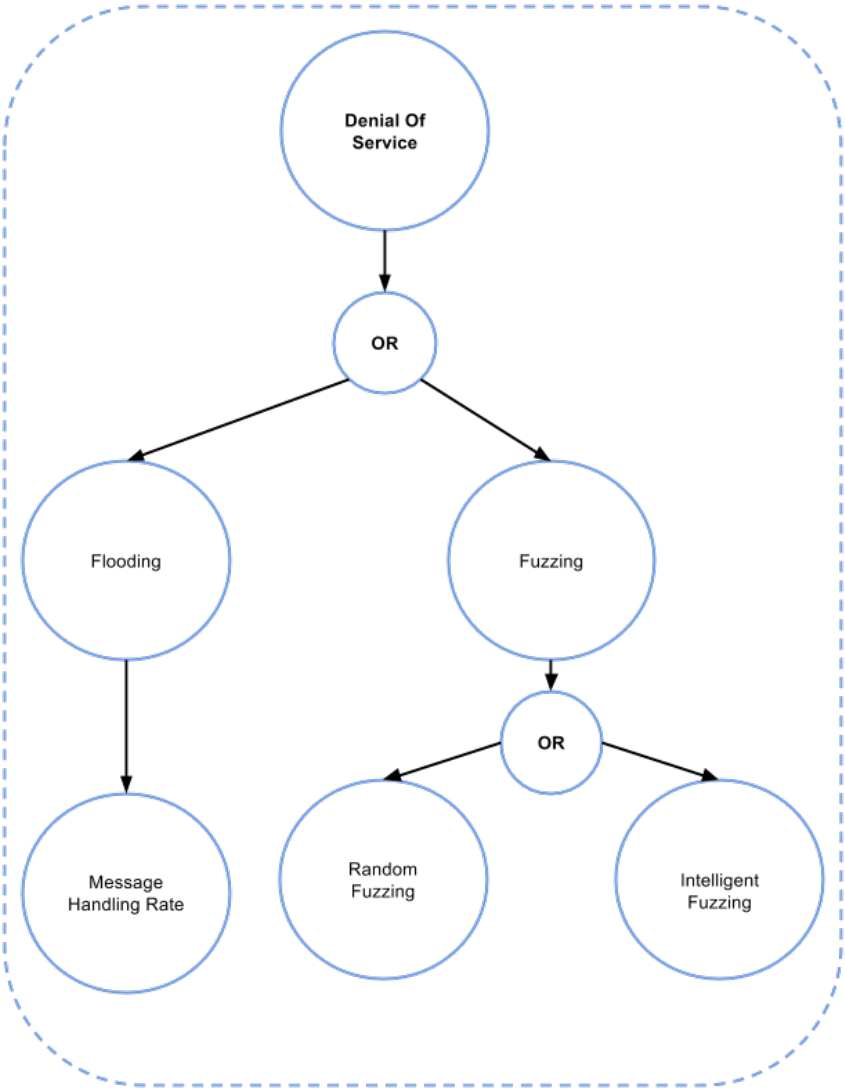

Figure 6.4: *Attack tree of the Denial of Service threat*

In the first step the test sends messages with all possible service ID numbers, the test is then able to verify if the service is present in the system or not depending on the response.
In the second step the test uses information from step one to scan the running services for possible data identifiers. The scan sends messages with all possible data identifiers to the services and categorizes the data identifiers in three categories:

- Present
- Present but not available in current security/session level
- Not present

In the final step of the test, different lengths of input is sent to each found data identifier and the request depending on the service to find the expected length. The scan outputs a list of present services, their associated DIDs and the input length. The list can be seen as a mapping of the DCM structure and can be used as input to the *Intelligent Fuzzing* test.

6.3.2 System Probing

The DCM module in each of the ECU:s has an identification number that acts as an address for communicating. The ISO 15765-1 states that the address of a module is a number between 0 and 2047. This test aims to find the identification number of the DCM module, that is used to communicate, by probing the system [26]. The test is conducted by sending a DCM requested message towards all the possible identification numbers in the system. If the test detects a proper DCM response message the test logs the identification number and alerts the tester that a possible DCM module has been detected.

6.3.3 Packet sniffing

This test is used to find information about the system architecture and how the system is communicating. It aims to find the identification numbers of the running modules and components of the system. This test is a compliment to the previous test 6.3.2 to find the identification number that the DCM module listens to during communication and how other non-DCM modules are addressed.

The test is conducted by connecting a node to the network that listens to the communication traffic for DCM requests/responses and record the ID number used.
Since the communication in vehicles today is unencrypted the test can also be used to find sensitive information such as seed/key pairs that are used to unlock security levels.

6.3.4 Restricted Memory Address scan

This test aims to find out information about memory sections that are restricted for reading without security access. If an area is restricted it can potentially contain the security access key, and could maybe be read if the binary file of the ECU could be extracted.

6.3.5 Message handling rate

This test aims to gather information about the DCMs ability to handle messages that are received at an unusual rate. The test sends 100 valid requests to the DCM at different rates. The rate is changed using different delays between sending each request. By counting the responses received the test calculates the number of requests the DCM has dropped or has been unable to handle. The test changes the delays from a zero millisecond delay up until it reaches a delay where all request receives

a response. The test runs the test cycle three times to verify the result.

6.4 Tampering of ECU Software Test

This section describes the test performed to gain the access to tamper with the software inside the ECU. The test are derived from the entry point leafs at the bottom of each attack vector in the tree, see Figure 6.3.

6.4.1 Man In The Middle

The Man in the Middle test [22] is a well known spoofing attack in the computer security field. In the attack an adversary places itself between two or more nodes and pretends to be the other part of the communication. This test is based on the same approach, where an undetectable malicious node is connected to the communication network that can control the traffic between the DCM and the legitimate client. The test aims to evaluate the possibility of client spoofing and alteration of request messages on the way between the test client and the server.

The test was conducted by placing a PC between the TOE node and the test client. The computer forwards all traffic to and from the node until certain conditions are met. When the condition is met the computer can block all traffic from the client and start communicating with the node acting as the legitimate client and gain elevated privileges.

6.4.2 Brute force security access key

This test aims to test the size of the key use to unlock the security access. The security access protocol allows 5 failed key tries before invoking a time penalty of 10 seconds. To reduce the brute force time the test abuses the ECU hard reset service to bypass the penalty time. A hard reset takes about 1 second, therefore it reduces the time it takes by a factor of 10. The test can only be used if it can be determined that the key is fixed and will not change during the time it takes to complete the test.

6.4.3 Seed/Key Sniffing

This test aims to listen to legitimate traffic in the CAN network. Specifically looking for diagnostic messages requesting security access using seeds and keys. The test builds a database of seeds and the corresponding keys that can later be used to be granted elevated privileges. After building the database it starts sending frames to the DCM requesting seeds until it receives one it has seen before, and can then send the corresponding key and be granted elevated privileges.

6.4.4 Seed Entropy

This test is aimed to test the entropy of the seed algorithm in the `SecurityAccess` service of the DCM. The test is sending a request seed message to the `SecurityAccess` that is responding with a legitimate seed response. The seed is stored and the tests send a new request seed message. The test can run until a seed collision is detected, and can then analyze the seeds to find a pattern in their generation.

6.4.5 Restricted Memory Address Write

This test is implemented to check the possibility of writing data to restricted parts of the ECUs memory. The test is performed by first finding an unrestricted memory area that allows writing, and then by using the `WriteMemoryByAddress` service it tries to write over the boundaries to restricted areas. Due to that the restricted memory areas still allow reading, the test can read the memory area it tried to write to and validate if the write was successful or not.

The test is performed in two parts. First the write request has a validly formed message where the `NoB` field in the header corresponds to the size of the payload of the request. In the second part, the test sets the value of the `NoB` to be inside the memory area that allows writing, but includes a longer payload.

6.5 Denial of Service Test

In this section the tests performed to achieve the root goal of a Denial of Service attack are described.

6.5.1 Fuzzing based Security Tests

The tests in this section are based on fuzzing [38], where packets with random or partially random data are sent to the DCM. This approach can be applied to all modules inside the ECU and is a good way to retrieve information about implementation flaws and system behaviour. Fuzzing can also be of use to get a view of the system and the services implemented inside the diagnostic module.

The tests are constructed in a way that makes it possible to retrieve information from an unknown system and to identify possible vulnerabilities and attack surfaces.

6.5.2 Random Fuzzing

This test is constructed to check the request input validation of the DCM module, where incorrectly formed request should be disregarded and dropped. The test retrieves information if it is possible to set the system in a non responsive state by sending special formed requests.

The test is performed in two parts. In the first part the test is performed using a fixed payload, where all bytes has the value of `0xFF` (the value is insignificant and could be any value between 0x00 and 0xFF). In the second part, the bytes in the payload have a random value.

The test is done in three steps, where each step tests different variant of incorrectly formed requests.

In the first step the test checks the validation of DCM request of unexpected length, such as very long or very short sized requests. The standard request message has the length of 4-8 bytes, the test uses different lengths varying between 0 and 255 bytes long. The first step sends a valid sized DCM message where the `numberOfBytes` (`NoB`) field correctly corresponds to the length of the payload.

Step 1 message example:

`[NoB, Payload]`

(`NoB` == Payload size)

In the second step the test check the validation of request messages that has longer or shorter payload then stated in the `NoB` field. In this step the test sets that the message length is between 0 and 255 bytes long and create a payload length different from that.

Step 2 message example:

`[NoB, Payload]`
(`NoB !=` Payload size)

The third step of the test sets the `NoB` to a fixed size value and change the payload length between 0 and 1000 bytes long.

Step 3 message example:

`[0x07, 0xFF,...,0xFF]`
(`NoB` = 7, Payload size = range(0,1000))

The steps are repeated in the second part but using random values for the payload.

All the steps follow the flowchart, see Figure 6.5, where the test starts by creating a reference answer by sending a valid message and storing the response. Next, a message is created according to the test parameters of each test setup and sent to the DCM, the test then waits for a response for 5 seconds before timing out. If a timeout occurs the test logs the crash and alerts the penetration tester. If a response is received the test sends the reference message and compares the corresponding response to the reference answer. If the responses differ the test logs the result and restart. Each test consists of 500 000 different messages.

6.5.3 Intelligent Fuzzing

This test sends arbitrarily long messages to known services running in the DCM. The test checks how the running services handle invalid diagnostic messages. The test is applied when the running services are known or as an additional step to the *System Scan* test (see Section 6.3.1) where the services and input sizes are revealed.

The test works in the same way as the *Random Fuzzing* test but with a more specified target for the messages that are being sent.

Example:
If the DCM is running the services in Table 6.1 the test creates fuzzing messages according to the same steps in the *Random Fuzzing* test but with the focus on the Service Id numbers of the specific services.

The test is a more intelligent fuzzing test where the scope is focused to known services and input sizes. The test controls the input validation of the services running on the system.

6.5.4 Flooding

This test aims to test the DCM modules ability to handle abnormal amounts of messages and see if it is vulnerable to flooding attacks [18]. The test is conducted by sending a large amount of messages

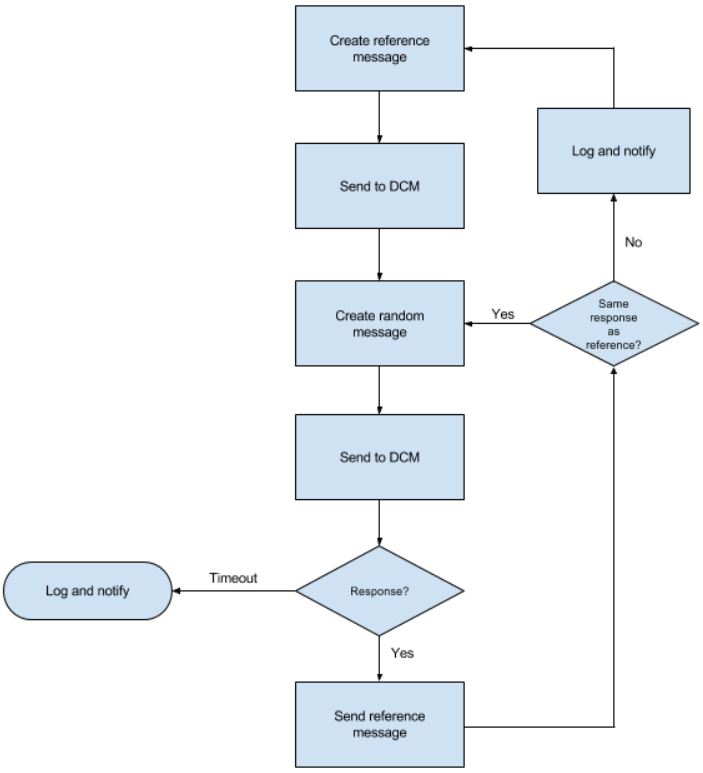

Figure 6.5: *Flowchart for steps one through six.*

Service	**Input size**
DiagnosticSessionControl	1 byte
EcuReset	1 byte
ReadDataByIdentifier	2 bytes
ReadMemoryByAddress	>3 bytes
SecurityAccess	4 byte
WriteDataByIdentifier	2 bytes
WriteMemoryByAddress	>3 bytes

Table 6.1: Example services and their input size.

from one test client while sending valid service requests from a second client. The test then analyzes if the second client receives any responses on the sent requests.

7 Results and test evaluations

This section includes a presentation of the results from the tests described in Section 6. What the test results can be used for as an adversary and possible countermeasures will be discussed in Section 8. The results are evaluated for the two different test environments described in Section 6.

7.1 Information Disclosure / System Reconnaissance

7.1.1 System Scan

The scanning experiments were successful and found all enabled services and available DIDs as well as their respective data lengths. On the evaluation board the results could be confirmed since it was a white-box system. This gives confidence that the results from the production system also included every service present.

In the production system environment the scan resulted in an complete view of the DCM module configuration running. The scan was able to report all running services including the implemented subfunctions and corresponding data identifiers. The scan discovered the specified input length of all reachable services. The scan reported that some subfuctions and data identifiers were placed behind a higher security level and were therefore not reachable at the default security level.

7.1.2 System Probing

The probing of the system was successful and was able to find the correct identification number of the DCM module in both of the test systems. The test also recorded the identification number of the response messages successfully in the systems.

7.1.3 Packet sniffing

Sniffing the CAN traffic allowed for the identification of the DCMs arbitration ID. Sniffing was done on both a production system and an evaluation board.

In the production system the test resulted in a clear view of what nodes were communicating on the CAN bus. Identifying the arbitration ID of the DCM was only a matter of recognizing a frame as a diagnostics message. In this case a `TesterPresent` message was found, and the DCMs arbitration ID could be identified by manually looking at the CAN traffic using the *BusMaster* logging software.

The evaluation board did not communicate externally, and therefore there were no packets to sniff.

7.1.4 Restricted Memory Address scan

This test could not be performed on the evaluation board due to that the `ReadMemoryByAddress` service was not configured on the evaluation board.

In the production environment the test showed that it was possible to scan the memory that is reachable of the DCM module. It showed that it was possible to read all data in RAM but not possible to write without having the right security access level.

7.1.5 Message handling rate

This test gave information about the rate at which the DCM module was able to handle requests. On the evaluation board the DCM module was able to respond and act on requests every 20 milliseconds and on the production system the DCM was able to handle requests every 10 milliseconds. The test also discovered that if messages were sent to the evaluation board with no delay in between, the system crashed after a small amount of time and did not recover without a complete reset. The production system was not negatively affected by the huge number of messages.

7.2 Tampering of ECU Software

7.2.1 Man In The Middle

The Man In The Middle test showed that it was possible to connect a malicious node inside the system and record the message received from the legitimate client and manipulate the payload before redirecting it to the DCM module without detection. The results was equal in both the evaluation environment and the production environment.

7.2.2 Brute force security access key

The test showed that the security access key could not be discovered by a brute force test within a reasonable time if the key length is 4 bytes or longer. The test could only try two keys every second due to communication delays. And the brute force technique is only applicable if the access key stays the same during the entire test. In the production system the test could not be applied due to that the security key algorithm was implemented in such a way that a new key was used after each reset.

7.2.3 Seed/Key sniffing

The sniffing test was done on an ECU ready for production. The test was able to pick up seven seed/key pairs while sniffing the traffic of the system. It was also able to retrieve information about the seed sized used in the system, in this system a seed size of 4 bytes was used. They were stored in a database before the test started to request seeds. Within just 5 minutes a match was found, allowing other scripts to write to any address in RAM.

There was no security access implementation ready on the evaluation board which made it impossible to run the test on that specific system.

7.2.4 Seed Entropy

This test was only available to be performed in the production system environment. The `SecurityAccess` service in the evaluation board was not configured to use a seed/key algorithm and we did not have the knowledge to implement the service.

The result in the production system showed that it was possible to find a seed collision and to retrieve all possible seeds in 15 minutes. The test also resulted that it was possible to manually analyse the list of seeds and find the seed generation algorithm.

7.2.5 Restricted Memory Address Write

This test showed that both evaluation environment and the production environment were able to validate the messages and therefore nullify the possibility to write to the restricted parts of the memory using the `WriteMemoryByAddress` service.

7.3 Denial of Service

7.3.1 Random Fuzzing

The Random fuzzing test is a tool to allow a security evaluator to test the robustness of the TOE. In the tests made the results differed depending on what system being tested.

Testing the evaluation board resulted in two major discoveries. The first was that when the system received more than 30 000 messages the entire system crashed and stopped responding to requests, not only the DCM. The second discovery was that when the system received a valid service message requesting a reading of a DID longer than 16 bytes, the DCM stopped working. In neither of the cases the system recovered to a running state.

The test was conducted on the production system by sending over 3 million requests. The results showed that the system was able to validate and discard the invalid requests out of all of the requests sent. The test showed that the system is not vulnerable to random messages that exceed boundaries and that it performs input validation.

The Random Fuzzing test showed that the configurations of both test environments give a high protection against invalid request that is sent to the systems. The test also showed how important message validation is, as could be seen in the first test when the evaluation system completely stopped working when a specific request was sent.

7.3.2 Intelligent Fuzzing

This test was preformed on both of the systems after running the *System Scan test*, where the services running on each of the systems was revealed. This information was then used as input for this test. In the evaluation board environment we also had knowledge about what services the DCM module was running, and could therefore verify the result.

When running the test on the evaluation board a limit had to be put in place so that the system would not crash from receiving more than 30 000 messages, the limit found in the *Random Fuzzing* test. The test showed that the system could single out the invalid messages and exclude the processing of these requests. The test showed the same results as we could see in *Random Fuzzing test*, that the DCM module stopped working when valid message was sent to service that requested a response longer than 16 bytes.

In the production system the intelligent fuzzing gave the same result as in *Random Fuzzing test*, the system was able to discard invalid or unexpected requests.

7.3.3 Flooding

The DCM was successfully flooded when sending messages at higher rates than it could handle (identified in the *Message handling rate* test). Legitimate traffic to the DCM was unable to get responses from the DCM since it was busy handling the bogus traffic sent to flood it. Additionally,

the evaluation board crashed when receiving messages at a high rate and was unable to recover to a working state.

8 Discussion and countermeasures

This section includes discussions and suggestions of countermeasures for each of the performed tests. The countermeasures discussed are meant to help securing a system running in a production system environment. At the end of the chapter a general discussion is presented that covers the overall security of the DCM module. It also includes a general way to counter the vulnerabilities of the system.

8.1 Information Disclosure / System Reconnaissance

8.1.1 System Scan

This test collects information about the target of evaluation and gets a view of the systems configuration. The test can be compared to a server scan in the web domain where the goal is to find information of the running services and the server configurations. When a clear picture of the DCM module has been collected an adversary can start to look for vulnerabilities in a structured way. Many of the tests performed in this report build upon knowledge collected through this test. The test can also be a tool for developers to verify that the target system is configured according to the specifications. It can also help security evaluation of the system by revealing if any old services that was used under development has been left in.

The System Scan test uses the ISO 14229-1 standard and that the DCM responds with specific error responses depending on how the services, sub-services and input lengths are configured.

The scan can be hard to counter, as explained in the previous section the communication can be encrypted and require authentication, but this can be computationally heavy and can be hard to implement on all traffic and still cope with the real time demands of the system tasks. To cope with this information leakage the developer can use this test and learn what the adversary might find out and prevent possible threats through that. This information leakage can be mitigated by having the DCM not responding unless the tester/client has been authenticated in some way. The system would then ignore to reply with the negative return codes found in ISO 14229-1 unless the client has been validated. The system could comply with the standard when the client has authenticated itself.

The test can easily be modified to scan other parts of the AUTOSAR system and not only the DCM module that was scanned in this thesis. In this way an even larger picture of the internal system can be revealed.

8.1.2 System Probing

System probing is the first test used to gain information of a system when the adversary has minimal knowledge about the software architecture. By probing the system the adversary can not only find information about the ID of the DCM module, but could also be able to find other interesting ID numbers. This test could also be used to find components that should not be present when the system is deployed in production.

The difference between this test and the System Scan is that this test finds arbitration ID numbers (belonging to for example SW-Cs and specific ECUs), and the System Scan finds services available in the DCM.

This test does however require some working knowledge about automotive systems, otherwise it would be more like a fuzzing test only sending random messages.

8.1.3 Packet sniffing

The test gives a clear view of what an adversary can see and what information can be discovered from only listening to the communication. To an adversary with knowledge of the automotive systems the test can supply information about what nodes are connected and how they are communicating on the network. The test can very quickly discover the node IDs used.

Packet sniffing can also be very useful for an adversary due to the fact that the system communication is in plain text and that the test can log secret seed/key communication. If the seed and the key can be reused, the adversary can use the information to mount an impersonation attack.

To mitigate the possibility of packet sniffing a solution would be to encrypt all communication sent on the network.

8.1.4 Restricted Memory Address scan

The memory address space of the DCM is laid out according to the locality principle, meaning that the memory used by the DCM is continuous in the RAM. With this test it is possible to find information of accessible, writable, memory areas. If the test reveals that some memory addresses can be accessed without extra security privileges, the adversary can try to exploit this in different ways. An adversary can use an accessible memory address to start a writing session and attempt to write over the allowed memory space and continue writing into restricted areas. The adversary could also exploit the possibility that the open memory area could contain sensitive data that is used by services in the DCM.

This test can also be a tool for developers to test that the memory accessibility works as it is expected before deploying the system.

8.1.5 Message handling rate

The message handling rate test gets information about at what rate requests can be managed. The test reveals information that can be used in further security tests, such as the flooding test or a system stability test. The results from the two test environments show that the evaluation board was configured to handle messages at a rate of one request every 20 milliseconds. The tested production system was configured to handle a request every 10 milliseconds. The test also showed that the evaluation board crashed and did not recover when messages were sent without any delay and that the production system was able to handle it without crashing. The production system might have an upper limit unreachable by the test setup used in this project.

8.2 Tampering of ECU Software

8.2.1 Man In The Middle

This test reveals one of the most severe threats based on the result. Due to the problem of unauthenticated plain text communication inside the vehicle an adversary could place itself on the bus between a legitimate client and the targeted DCM module. The adversary can in this way decide what information should be sent to and from the DCM. The adversary can thereby change or block requests and responses that are being sent to and from the DCM. Due to the nature of the CAN bus, the

`SecurityAccess` service unlocks privileges for the entire bus and not only for the client that performs a valid authentication session, the adversary can take over the bus when the privileges are unlocked. This makes it possible for an adversary to connect to the CAN bus network, not only as a man in the middle, and wait for the security authentication is performed before taking over the communication.

After the authentication is complete an adversary can record the data transferred during a reprogramming session and replay the programming session in another vehicle and yield the same result. For example to unlock higher performance of the engine and only paying for one vehicle.

For an adversary with previous knowledge about ECU programming and the expertise to program their own software component, the system can be severely vulnerable to both security and safety threats. An adversary able to add new software components could for example add a trojan that triggers under specific conditions, such as a speed over 100 km/h and create a total block of the brake systems.

This threat can be countered by always using an authentication scheme of the client that ensures that the client is authorized to perform the changes of the software. It can also be countered by using encrypted communication while performing a reprogramming session.

8.2.2 Brute force security access key test

This test can be used to check that the system behaves in an expected way when an adversary tries to guess security access passwords. The test should not be applicable in real production systems where the security key should depend on a seed key response and not be hardcoded as in the evaluation board setup used in this project.

A possible scenario could be:
The standard states that a seed/key approach should be used to gain elevated privileges, it also states that a penalty time should be applied if a number of failed keys has been attempted [16]. The standard does not state how this should be implemented. This could potentially result in a insecure seed/key implementation that still is valid according to the standard. If the seed/key algorithm e.g. uses a hardcoded seed and key pair or a small sized key the implementation could potentially be insecure but valid.

Examples of insecure configurations, but valid according to the ISO standard, could be:

- **Key length of less than 4 bytes**. The ISO 14229-1 does not state a required length to be used for the security key. This open the possibility for a valid configuration using a very short key length. The test concluded that a key length over 4 bytes is desirable, resulting in a brute force time of over 50 years.
- **Key hardcoded and less than 4 bytes**. The standard states the Seed/key handshake as optional. This open the configuration to be implemented using a fixed key that is used to unlock the security access every time needed, resulting in a valid but insecure implementation.
- **Number of failed key attempts before penalty back off time is too high, over 10 tries**. The standard does only state that there need to be a back off time penalty after a number of failed attempts, but not the number of attempts. This open the configuration to be implemented using a insecure large number of attempts before a penalty is applied, resulting in a very reduced time to brute force the key.

8.2.3 Seed/Key sniffing

This test is a sub-test to the packet sniffing test, but this test only targets the vital part of the security access session data needed to unlock the security levels. The test looks for and stores the information as soon as a login session is started by any legitimate client. This information can then be shared in an adversary community and might lead to a large database of seed/key pairs allowing adversaries to access the security levels or to reverse engineer the algorithm for computing the key.

8.2.4 Seed Entropy

Since the security access is built as a seed/key challenge response protocol, seed entropy is important. The security access key responses are limited to 5 tries before applying a penalty time of 10 seconds making a bruteforce attack on the key hard for an adversary. The seed request does not have a penalty rule similar to that of the key sending, but instead allows the adversary to request seeds at a speed only limited by the message handling rate of the DCM. The advantage for an adversary is that it can request seeds until one matches another previously recorded seed, and the key can be reused. This makes the the entropy of the seed very important. This test can be a good way to test the seed entropy of the system before deploying it.

As we could see from the result in Section 7.2.4 a seed collision was found within a few minutes and by recording the seeds it could quickly be determined that there was no randomness in the seed generation, making the entropy of the seed to small to be secure. In tested system we found that a seed size of 4 bytes was used. Implemented using a complete random seed algorithm this could be a secure entropy size but as it was configured the entropy became just a fraction of the possible entropy space.

A recommendation for mitigating this vulnerability would be to require that the seed is generated in a cryptographically secure way, and not through a counter or something of that sort. The stated security implementations and restrictions for the key should also apply on the seed since they are dependent on each other.

8.2.5 Restricted Memory Address Write

This test concluded that in both of the test environments the systems were able to detect and protect against writing to the restricted areas.

This is a required security test to perform before any release of new DCM software. If the test were to succeed, an adversary would be able to overwrite crucial memory areas that could lead to system failure. An adversary could possibly be able to upload a secondary boot loader including flash routines to reprogram the full ECU software.

8.3 Denial of Service

8.3.1 Fuzzing based Security Tests

The fuzzing based tests only control the handling of malformed and invalid request messages and not the rate at which the DCM handles requests.

Random Fuzzing

This test is constructed to reveal weaknesses in the systems input validation. The test is useful for validating the system before applying it in production. For an adversary the test can reveal untested input validation and invalidly formed requests that can result in safety and security risks of the vehicle. The test can be used to complement the intelligent fuzzing test due to that it creates malformed requests that neither the developer nor the adversary would have thought of to test. Due to the randomization and the lack of structure when creating requests the test can lead to very long execution times before giving any results, if there are any. The test does not ensure that even known vulnerabilities will be found and should be complemented with a more intelligent fuzzing method.

Due to that the test uses the valid communication path with the DCM the test can only be mitigated by a well tested message validation, as the one implemented in both of the tested systems. To ensure that the system has a correct boundary control and that the input validation works as expected a test like this should be applied in all steps of production, when each node DCM is configured and also when a fully deployed system running multiple nodes is configured.

Intelligent Fuzzing

This test has the same capability as the dumb fuzzing test but with the advantage of being able to aim the fuzzing towards known services in the system. Since there often are multiple developers involved when implementing a system this can be a good way for each of the developers to test their service implementations individually before deploying it to be a part of the full system. For an adversary trying to perform this test the knowledge of how the system is configured and what services are running is important. This information can be collected by the *System Scan* test.

To protect the system against an adversary able to run this test the same advice as in the *Random Fuzzing* test should be applied here.

8.3.2 Flooding

This test shows that it can be possible to deny communication with the DCM by sending a large number of requests. The test showed how vulnerable the systems are against blocking legitimate clients or other nodes from communicating with DCM modules. It also showed that it was possible to block requests under selected times, such as software update sessions. This can potentially lead to both safety and security problems.

If a reprogramming session were to take place and a flooding was being performed at the same time, the reprogramming frames would be blocked out. The scenario of a lost reprogramming session is not addressed in ISO 14229-1, and since the DCM is configured to suppress positive responses a technician can believe that a software component has been updated even if the reprogramming was not actually performed. If the reprogramming was a security patch, the system would still be vulnerable and could be exploited by an adversary. If flooding can be performed while the vehicle is in motion and the flooding can result in that a crucial `ECUReset` request to the DCM was dropped, the safety of passengers or people close to the vehicle can be affected. According to our contacts in the industry, there are cases where such `ECUReset` messages are sent while the vehicle is in motion to force an ECU into a safe state. If this message was denied the ECU in question could be stuck in a non-safe state.

The flooding problem is hard to mitigate due to the CAN communication structure. The message structure does not include source information that can be used to exclude a node that sends an unusual amount of requests. In the case where a communication protocol has source information included, as

in the 29 bit extended mode of CAN [32], the attack can be mitigated by using an excluding policy. If the adversary could be able to change the source identification in the messages the attack would still be possible and an authentication policy would be preferred instead. To protect against flooding when communication without identification is used, that is the most common communication protocol in vehicles and also in our setup, the separation of critical nodes can be used, where the critical nodes are placed in separate sub-networks. This would make sure that the more easily accessible ECUs for an attacker, i.e. infotainment and the like, would not be able to deny communication in the more critical network where for example engine control would reside. The use of a low speed communication bus where nodes are not able to send messages faster than the DCM module can handle can be one way of mitigating the problem. A token based communication protocol can be used where all the involved communicating nodes have their allotted time slot where they are able to send message on the bus.

8.4 ISO 14229

ISO 14229-1 defines how the information flow of the DCM services should be done. The flow to perform a service execution is divided into three parts, the standardized steps, optional/recommended steps and the vehicle manufacturer steps. The standardized steps are the only mandatory steps needed to comply with the standard. Following the standardized steps, the programming of an ECU will be performed through an insecure session without any authentication or integrity.

If the Optional/recommended step were instead included in the standardized flow, the services would be implemented more securely.

8.5 Overall Discussion

The tests presented in this report shows that it was possible to find security weaknesses of an AUTOSAR DCM module. It also shows that even if the security rules of ISO 14229-1 are implemented in the system it can still be vulnerable to attacks.

Our tests can be expanded to include more tests that explore more and other attack surfaces of the DCM module. The number of security tests in this report was limited due to the time limit of the project and the ability to configure the evaluation system to include more services. We decided to include these tests due to that they all together are in the scope of the possible threats presented in the threat model.

The tests done in this report was limited to the scope of just embracing the DCM module but could be adapted to test a more complete set and other modules of the AUTOSAR implementation.

When doing a threat model of a system it is important to have both security experts and experts in the area in which the system operates present. Otherwise the threat model and risk assessment may be skewed to be either to lenient or to strict. Having developers with an education in security should also be preferred since they have knowledge of the most frequent pitfalls.

Today most developers in the automotive industry are encouraged to follow the MISRA-C standard [7], which has a goal to "promote best practice in developing safety related electronic systems in road vehicles and other embedded systems". Another set of coding guidelines targeting security instead of safety is the CERT secure coding guidelines. It's a community driven effort to provide rules and recommendations to "eliminate insecure coding practices and undefined behaviours that can lead to

exploitable vulnerabilities".

As presented by Vembar and Holle [40] 43% of the security recommendations in CERT are not covered in MISRA, 37% of the guidelines are covered, and 1% of the guidelines in CERT are contradicted by MISRA. The rest of the CERT guidelines are only partially covered by MISRA.

Developers of automotive embedded systems must have guidelines not only for safety, but for security as well, since vulnerabilities in security will affect safety.

Quantitative security such as threat modeling and risk assessment are very subjective processes. Two teams assessing the risks of one system may very well come up with different assessments. A research paper by Verendel [41] concludes that there's a lack of validation of quantitative methods regarding security. It does not, however, rule out quantitative processes as inherently lacking, but states that more empirical validation of such processes would be beneficial.

9 Conclusions

This report has shown that it is possible to gain unauthorized access to vital parts of the vehicle's embedded system by using common attacks and vulnerabilities found in the computer security domain. We have shown that some implementations of the AUTOSAR diagnostic basic software module may be susceptible to denial of service attacks even by adversaries that are new to the automotive setting, even when the implementations were following the ISO standard. We also discuss the potential risk that the security vulnerabilities can lead to and how they can endanger safety of the vehicle's passengers.

A number of the vulnerabilities found in the diagnostic communication manager in this thesis can also be found in the OWASP Top 10 web security risks from 2013 [28]. OWASP (Open Web Application Security Project) is an international organization aiming to enable other organizations develop and maintain web applications that can be trusted, through rigorous security.

The threats from OWASPs Top 10 list that has been identified in our tests are:

- Broken Authentication and Session Management
- Sensitive Data Exposure

The broken authentication management vulnerability was identified in the Seed/Key sniffing test (see Section 7.2.3). OWASP describes this vulnerability as: "Credentials can be guessed or overwritten through weak account management functions". In this case the Seed/Key algorithm could be guessed quite easily, which is a direct mapping from the OWASP vulnerability description.

A sensitive data exposure vulnerability was found through the packet sniffing test (see Section 7.1.3). This vulnerability is described as: "Is any of this data transmitted in clear text, internally or externally?". In the setting for this thesis all data is transmitted in clear text, this vulnerability is clearly present in the system.

We present different possible attack scenarios and show how the tested systems responded to the different scenarios. We also give examples on how the security vulnerabilities can be mitigated.

By creating attack trees for the three threats identified (see Section 6.2) we have designed a number of tests that should be done during the development phase of the diagnostic system. The threats could all be mitigated through the use of encrypted communication and an authentication scheme for the communication channels. As presented by Kleberger and Olovsson, an IPSec implementation could counter these security threats if ethernet is used together with the IP stack to communicate between the nodes inside the future vehicles.

Van Herrewege et al. presents an authentication scheme specifically for CAN they call CANAuth [39]. The authentication scheme is backwards compatible and can be run on any CAN bus without modifying existing nodes. However, it does not enable encryption of the communication, and is not protected against denial of service attacks.

9.1 Future work

This report shows that it is possible to endanger the safety of a vehicle if it is possible to gain access to its internal communication network. With this knowledge, future work can be done on investigate the possibility of employing the same security tests from a remote system gaining access via any

wireless interface, such as Bluetooth, RFID or WLAN.
We have focused on tests on the internal communcation, but one could imagine tests done more on hardware, for example analysing a binary file extracted from JTAG.
It could also be an interesting project to implement some security mechanisms that counter the threats and vulnerabilities found in this thesis. For example a project could be to implement some cryptographic authentication scheme similar to IPSec for the CAN bus.

References

[1] ArcCore. *Arctic Studio*. Feb. 2015. URL: http://www.arccore.com/products/arctic-studio/.

[2] ArcCore. *ArcticCore v.7.0.0*. May 2015. URL: http://www.arccore.com/products/arctic-core/arctic-core-for-autosar-4-x/.

[3] ArcCore. *VK-EVB-M3 Evaluation board*. May 2015. URL: http://www.arccore.com/products/vk-evb/vk-evb-m3/.

[4] S. Bayer et al. *Automotive Security Testing - The Digital Crash Test*.

[5] *BUSMASTER homepage*. Feb. 2015. URL: http://rbei-etas.github.io/busmaster/.

[6] *CAN-Case homepage*. Feb. 2015. URL: http://vector.com/vi_cancase_xl_log_en.html.

[7] M. Consortium. *MISRA homepage*. May 2015. URL: http://www.misra-c.com/.

[8] C. K. Deepika, G. Bjyu, and V. S. Vishnu. Implementation of DCM module for AUTOSAR Version 4.0 (2013).

[9] *Hackers Reveal Nasty New Car Attacks–With Me Behind The Wheel*. July 2013. URL: http://www.forbes.com/sites/andygreenberg/2013/07/24/hackers-reveal-nasty-new-car-attacks-with-me-behind-the-wheel-video/.

[10] F. Holik et al. "Effective penetration testing with Metasploit framework and methodologies". Nov. 2014.

[11] M. S. Idrees et al. "Secure automotive on-board protocols: a case of over-the-air firmware updates". *Communication Technologies for Vehicles*. Springer, 2011, pp. 224–238.

[12] IHS Global Insight, Inc. Resistance is Futile - Electronics are on the rise: Electronic Control units and ommunication protocols (2009).

[13] *International Organization for Standardization homepage*. Feb. 2015. URL: http://www.iso.org.

[14] M. Islam. *HEAVENS Project Terminologies*. Jan. 2015.

[15] ISO 11898-1. 2003.

[16] ISO 14229-1. 2013.

[17] ISO 15765-2. 2011.

[18] M. Jensen, N. Gruschka, and N. Luttenberger. The Impact of Flooding Attacks on Network-based Services (2008).

[19] P. Kleberger and T. Olovsson. Securing Vehicle Diagnostics in Repair Shops (2014).

[20] K. Koscher et al. "Experimental security analysis of a modern automobile". *In Proceedings of IEEE Symposium on Security and Privacy in*. 2010.

[21] A. Lautenbach and M. Islam. *HEAVENS Deliverable D2 - Security models, ver. 1.0*. Dec. 2014.

[22] B. M. Luettmann and A. C. Bender. Man-in-the-middle attacks on auto-updating software (2007).

[23] Microsoft. "Simplified implementation of the Microsoft SDL". Nov. 2010.

[24] E. K. Mohd and F. Khan. A Comparative Study of White Box, Black Box and Grey Box Testing Techniques (2012).

[25] S. Myagmar, A. J. Lee, and W. Yurcik. "Threat modeling as a basis for security requirements". *Symposium on requirements engineering for information security (SREIS)*. Vol. 2005. 2005, pp. 1–8.

[26] *Nmap*. URL: http://nmap.org.

[27] *OpenVAS homepage*. Feb. 2015. URL: http://openvas.com/about.html.

[28] OWASP. *OWASP Top Ten Project 2013*. May 2015. URL: https://www.owasp.org/index.php/Top10#OWASP_Top_10_for_2013.

[29] *PEAK-CAN homepage*. Feb. 2015. URL: http://www.peak-system.com/.

[30] E. Project. *E-safety Vehicle Intrusion Protected Applications (EVITA)*. URL: http://www.evita-project.org/.

[31] H. Project. *HEAling Vulnerabilities to ENhance Software Security and Safety*. URL: http://www.sp.se/en/index/research/dependable_systems/heavens/Sidor/default.aspx.
[32] H.-C. Reuss. *Extended Frame Format - A New Option of the CAN protocol*. May 1993.
[33] A. Ruddle et al. *EVITA Deliverable D2.3: Security requirements for automotive on-board networks based on dark-side scenarios*. Dec. 2009.
[34] B. Schneier. Attack trees. *Dr. Dobb's Journal* **24** (12 Dec. 2009), 21–29.
[35] B. Schneier and J. Wiley. Secrets and Lies: Digital Security in a Network World (2000).
[36] A. Shostack. *Threat Modeling: Designing for Security*. John Wiley & Sons, Inc., 2014, pp. 209–213. ISBN: 978-1-118-80999-0.
[37] F. Swiderski and W. Snyder. *Threat modeling*. Microsoft Press, 2004.
[38] A. Takanen. Fuzzing for the Masses. *Network Security* **2008**.8 (Aug. 2008), 4–6.
[39] A. Van Herrewege, D. Singelee, and I. Verbauwhede. "CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus". *ECRYPT Workshop on Lightweight Cryptography 2011*. 2011.
[40] P. Vembar and J. Holle. *Secure Coding and MISRA C in ECU Development*. ESCAR, Hamburg. Nov. 2014.
[41] V. Verendel. "Quantified security is a weak hypothesis: a critical survey of results and assumptions". *Proceedings of the 2009 workshop on New security paradigms workshop*. ACM. 2009, pp. 37–50.
[42] P. Vyleta. "Automated penetration testing in automotive industry". MA thesis. Czech Technical University in Prague, 2014.
[43] G. Weidman. *Penetration testing: a hands-on introduction to hacking*. English. San Francisco, CA: No Starch Press, 2014. ISBN: 9781593275648; 1593275641. URL: www.summon.com.
[44] *winIDEA open homepage*. Feb. 2015. URL: http://www.isystem.com/download/winideaopen.
[45] M. Wolf, A. Weimerskirch, and C. Paar. "Security in automotive bus systems". *in: Proceedings of the Workshop on Embedded Security in Cars (escar)'04*. 2004.