CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Model-based Security Testing in Automotive Industry

Master's thesis in Computer Systems and Networks

MARTIN KASTEBO, VICTOR NORDH

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017

Model-based Security Testing in Automotive Industry

MARTIN KASTEBO
VICTOR NORDH

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Model-based Security Testing
in Automotive Industry
MARTIN KASTEBO
VICTOR NORDH

Industrial Supervisors: Jörgen Borg, Henrik Broberg, Volvo Cars Corporation
Supervisor: Riccardo Scandariato, Dept. Computer Science and Engineering
Examiner: Francisco Gomes, Dept. Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Computer Systems and Networks
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2017

Model-based Security Testing
in Automotive Industry
MARTIN KASTEBO, VICTOR NORDH
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The automotive industry is entering a new era as the cars becomes more complex and connected to the Internet. Today a modern car consist of over 100 ECUs and has an Internet connection, which makes the vehicle exposed for malicious attacks. Therefore, the importance of being confident that the system is behaving as intended increases.

This thesis survey the state-of-the-art in the model-based security testing (MBST) field and investigates the possibility to apply a MBST approach within the automotive industry, more specific at Volvo Cars Corporation (VCC). The focus is the gateway firewall in the infotainment subsystem which is the protection for incoming and outgoing traffic.

It is concluded that it is infeasible at this point to make use of an existing MBST approach. An evaluation of model-based testing tools is conducted which can be used for testing functionality of security mechanisms. However, no model-based testing tool is appropriate at Volvo Cars and a new tool needs to be implemented. The final conclusion is that it is possible to make use of a model-based security testing approach with the new AFT tool, which automatically verifies whether requirements are fulfilled or not. The result is that 10 out of 11 existing requirements at VCC can be covered by the MBST approach.

Keywords: MBST, firewall, testing, automotive

Acknowledgements

We would like to thank our supervisors; Jörgen Borg and Henrik Broberg at Volvo Cars and Riccardo Scandariato at Chalmers University of Technology for supporting us along this thesis. Lastly, we would also like to thanks Francisco Gomes for being engaged in our thesis and being our examinator.

Martin Kastebo, Victor Nordh, Gothenburg, May 2017

Contents

List of Figures

List of Tables

Terminology and Acronyms

VCC - Volvo Cars Corporation

ECU - Engine Control Unit

SUT - System under test

MBT - Model-based Testing

ST - Security Testing

MBST - Model-based Security Testing

CVE - Common Vulnerabilities and Exposures

CPE - Common Platform Enumeration

CVSS - Common Vulnerability Scoring System

AF - Awesome Framework

AFT - Awesome Firewall Tool

Security - "Security is a non-functional property, and defines how a system is supposed to be, in contrast to what a system is supposed to do." [3]

Security Requirements - Security requirements are the high level requirements that specify what should be allowed and not allowed, considering a security perspective.

Vulnerability - "Vulnerabilities are flaws in applications that allow attackers to do something malicious." [4]

Attack - "Attacks are successful exploitation of vulnerabilities." [4]

Test case - is a set of tests that together have a shared goal to verify.

Whitelisted - is stating allowed objects for a certain system.

1

Introduction

The first digital controller in form of a self-contained embedded system was integrated into a vehicle in the late 1970s. The controller was named Engine Control Unit (ECU) and its purpose was to adjust the mixture between fuel and oxygen before combustion, which would result in a reduced emission [5]. Since then, digital controllers have been integrated into the majority of the functions in a car, e.g. the throttle, brakes and lights, resulting in ECU being generalized into Electronic Control Unit. A vast majority of these ECUs are implemented to improve safety functionalities in the car e.g. Anti-lock braking [6].

Today, a modern car is far from being a mere mechanical product and consists of almost hundreds of ECUs. These create a highly advanced distributed system controlling the different parts of the car. The ECUs in the in-vehicle network communicate with each other over multiple sub-networks and gateways using different communication protocols e.g. CAN, LIN, MOST and FlexRay [7]. The in-vehicle network are prone to network attacks, mainly due to the lack of procedures of verifying the authenticity and integrity of the communications. Considering the safety aspect, a successful attack against the in-vehicle network can cause devastating consequences for not only the passengers but also to the surroundings of the vehicle. Miller and Valasek have presented a successful remote attack against a car, which showed the possibility to take control over brakes, steering and acceleration [8].

The development of vehicles is a highly distributed effort, involving individuals from many different organizations which use different methods. To introduce security as an integral part of the development, there is a need for a consistent way to interpret and verify the security requirements. Increased precision and traceability between analysis, requirements, tests and field data would decrease the risks of missing the security aspect. Furthermore, by automating this process, it also becomes possible to decrease lead times which is highly beneficial. Today, the verification of security requirements is performed late in the development phase since it requires the system under test (SUT) to be implemented, where both time and budget are very restricting factors. To fix a problem this late is prohibitively expensive and not always an option [9]. Furthermore, security testing (ST) is hard to automate since it lacks systematic approaches [10] and it is required of the test engineer to have an expertise within the security field and a deep understanding of the SUT [11]. To achieve a satisfying confidence, the test engineer's ability to execute relevant test

cases on the SUT is important. Therefore, by combining the security expertise and knowledge of the SUT, it is easier to decide what a relevant test case is.

Model-based security testing (MBST) is an approach that enables earlier verification of security requirements and automated generation of tests based on a model of the SUT. It is a new research field, mostly applied in academia [3], based on model-based testing (MBT) with focus on security requirements. These approaches are further described in Section 2.

1.1 Aim

Considering that a vehicle has a very complex and highly distributed in-vehicle network, the implementation of the chosen approach will be limited to a subsystem. This thesis investigates the possibility of applying a MBST approach on a firewall in the ECU responsible for the connectivity within the infotainment deployed in Volvos new models: XC90, V90, S90 and XC60. The goal of this thesis is to propose a suitable MBST approach which improves the verification accuracy of security requirements and the overall efficiency of the verification process. Additionally, we suggest how to adapt the existing process into new beneficial approaches for MBST. To achieve this, a literature study will be conducted, followed by an investigation whether there are any existing MBST tools that can be used. Furthermore, a review of model-based tools is performed to determine if they could be applicable for a MBST approach or if a new tool needs to be implemented.

Research Questions

- Are there a MBST approach suitable to apply at VCC?
  - What is required to apply a MBST approach at VCC?
- Are there any available tools for MBST?
  - What is required from VCC to use available tools for MBST?
- How much of the requirements of the firewall can be covered with a MBST approach?

1.2 Contribution

The contribution of this thesis is divided into two different categories, scientific and technical.

Scientific contribution

- The contribution of this thesis is to investigate the maturity of the security process at Volvo Cars Corporation to evaluate if MBST is an applicable approach in an industrial context
- Provide an evaluation of available MBT tools to be used for a MBST approach.

Technical contribution

- Implement a tool that takes text-based requirements of a firewall as input, generate relevant test cases and executes them to produce a status report illustrating whether the firewall configuration fulfills the requirements or not.
- Introduce a guideline for describing requirements of a firewall which is used as an input-model into our developed tool for MBST.

Outline

This report is structured as follows. Section 2 describes the different approaches of ST, MBT and MBST in more detail to give a better understanding of the concepts. Section 3 describes the methodology of the thesis. The MBST implementation will be presented in Section 4 together with an illustration of the implemented tool. Section 5 presents the results and an evaluation of the MBST approach. Furthermore, a discussion is presented in Section 6. Finally, Section 7 presents the conclusions of this thesis MBST approach in the automotive industry.

2

Background

In this section will the core characteristics of the thesis be presented. The different testing techniques of security testing, model-based testing and model-based security testing will be described in depth below.

2.1 Security Testing

The increase of connected devices in the society leads to larger, and more complex, systems. With more complex systems, it becomes easier by mistake to introduce flaws which possibly leads to an increased amount of vulnerabilities. A common way of testing the quality assurance of a system is to test that it fulfills the functional requirements. This is often done by performing dynamical tests to ensure that the features are correctly implemented [9]. However, security is not a feature but a tool to make sure that the system behaves correctly or as McGraw et.al. [11] states: "security should make the system behave correctly in the presence of a malicious attack". To ensure that a system behaves correctly under attack, it is important that security requirements are fulfilled. These requirements arise from security properties such as confidentiality, integrity, availability, authentication and authorization.

It is very challenging to determine if a system is fully secure. A system might behave correctly according to the functional requirements but can perform other unintended tasks in the process. Furthermore, testers easily miss these hidden bugs which could result in a vulnerability of the system. Therefore, security testing is important to ensure that the security requirements are fulfilled which verifies that the security features of the implementation are consistent with the design [12]. While traditional testing verifies that the tasks are performed correctly, security testing investigates the behaviour of the system when it is under attack. Therefore, a security tester needs to investigate the security risks by e.g. abuse cases in order to determine how the system behaves under attack.

2.1.1 Approaches

There are different approaches to verify security requirements and some of the most common are described below. A drawback for all these approaches are that they all require a physical system to perform the tests. Therefore, all the approaches can only be performed very late in the development phase where both time and cost are important factors. A found vulnerability late in the development phase could result in a very time consuming fix with high economical impact.

Penetration Testing

Penetration testing is the most popular principle for verifying security requirements [9]. However, penetration testing is not a systematic approach and is applied late in the development cycle to act as a final acceptance of the system. By using abuse cases, it is possible to detect anomalies in the system, e.g. by start a port scan on the system to detect services and potential weaknesses. From the results, further attacks are executed to try exploiting the found weaknesses.

However, an acceptance from a penetration test only proves that there are not faults for the performed test conditions. There is still a risk that a vulnerability is present in the system. Therefore, it is important that the penetration tester possesses an expertise within the security field and a good knowledge of the SUT to be confident that the system is secure and that the relevant parts are covered. Furthermore, it is important to determine relevant test cases in order to verify that the security requirements are fulfilled.

There are several approaches to verify security requirements, but is is common to involve two different diverse problems. Firstly, security mechanisms need to be verified to make sure that their implementation is correctly implemented. Lastly, different approaches for testing are used for simulating an attack of a potential adversary.

Risk-based testing

Risk-based testing is a widely used approach in the industry [13]. It uses the idea of prioritizing tests where the most critical parts of the systems are triggered. Risk-based testing could be broken down into three parts, Risk Driver, Risk Assessment and Risk-based test process. Risk drivers are the first part where it considers if functionality, safety or security is the major risk drivers of the system. This is followed up with risk assessment where the process identifies and analyzes the risks in order to determine the level of risk with e.g. impact ratings. Last is the risk-based test process which contains a regular test process (planning, design, implementation, execution, evaluation) with regards to the impacts of the potential risks [13]. A drawback with Risk-based testing is the human factor, since threats should be

evaluated and prioritized manually, the result is dependent on the tester's experience and knowledge within both security and the SUT.

Fault injection-based testing

The main idea of fault injection-based testing is to evaluate fault-tolerant mechanisms and determine fault-tolerant measures to improve coverage. This is done by injecting faults to investigate how the systems react with faults present and are often combined with stress-testing to evaluate the robustness of the system [14].

Mutation-based testing

A common denominator of testing is to the keep the time of testing as low as possible. Mutation-based testing is an approach to improving the quality of a test suite during the development phase and the goal for this approach is to improve the effectiveness of detecting faults while keeping the testing time shorter [14]. In practice, this is performed by introducing errors into the system and the quality of the test suites are measured by how many of these errors that get detected [15].

Vulnerability scan testing

Vulnerability scan testing is an approach where it scans the system to search for security risks in weak implementations. By performing a port-scan on a system, it is possible to determine if there exists open ports on the system that have services that are present of vulnerabilities such as cross-cite scripting or SQL-injection. A vulnerability scan is often performed as a first step to determine possible attack vectors. A drawback is that a vulnerability port scan is not broad enough and only touches the surface of the system.

Property-based testing

Property-based testing focuses on testing some specific security properties. If a tester wants to test and verify authentication, a property-based testing performs several tests cases with different input to test the authentication property [16]. This approach is useful when a specific property needs to be verified. However, it is not as powerful when testing a complete system on a high level since it is directed into a specific property.

Fuzz testing

Fuzz testing is an approach that works by injecting random data into the system to evaluate its behaviour. Fuzz testing is not logical and can therefore find flaws in the system which are difficult for logical tests. Furthermore, there exists a concept, smart fuzz testing, where the tester knows about the system and can perform a white-box testing which makes it possible to determine smart test cases with injections of random data [17]. A major drawback is that it is challenging to restrict the test cases and produce test cases that tests different aspects of the input without unnecessary redundancy.

2.1.2 Security Testing in Firewalls

The use of a firewall is usually the first step for keeping the network and computer secure. The aim of a firewall is to prevent undesirable data to pass through onto the network. To achieve this, firewalls are configured with rules that make them aware of what types of data to block and what to let through. At VCC, there exists general requirements for all firewalls. Requirements are important in a large company as VCC with many different people involved in the process. By having requirements, it becomes easier for people without a technical background to understand the functionality of the firewall instead of interpreting a configuration file of the firewall.

The most common approach for testing a firewall is by performing a penetration test. That makes it possible to detect flaws in the system by executing different types of attacks against the network, e.g. the Heartbleed attack against the OpenSSL-protocol [18]. An additional approach is to test the firewall rules to determine if the requirements hold from the set of rules [19].

However, firewalls are very important to keep the network infrastructure secure. As stated in Section 2.1.1, penetration testing is not a systematic approach and requires an expertise of the tester since it only can test for known flaws. There has been presented systematic approaches for conformance testing of a firewall [20] [21], but it is very complex since it makes use of formal models. The main difference between penetration testing and conformance testing of firewalls is that penetration testing tends to always use the same tests compared to conformance testing which generate tests based on the requirements [22].

2.2 Model-based Testing

MBT is an approach which is able to automatically generate test cases based on either one or a composition of several models of the SUT [1]. A model should be a more abstract representation of viewing the system compared to the SUT itself,

otherwise, it would be easier to only interpret the SUT. This more abstract model will lead to a simplified understanding of the SUT for all involved [23]. However, to generate meaningful tests based on the model, the model needs to be precise enough to be able to act as the SUT. It is also of importance to maintain the models updated to ensure that generated tests are relevant and consider the actual version of the SUT. The basic idea with MBT is to replace the manual test design into an automated process that generates test cases that could be executed on the SUT. However, there is a challenge to limit the amount of generated test cases to maintain an acceptable coverage [15].

There are three stages in the MBT process: Model Specification, Test Generation and Test Execution as shown in Figure 2.1. The definition of these three stages give rise to six dimensions of MBT approaches: Scope, Characteristics, Paradigm, Test Selection Criteria, Technology and On/Offline which will be further described below.

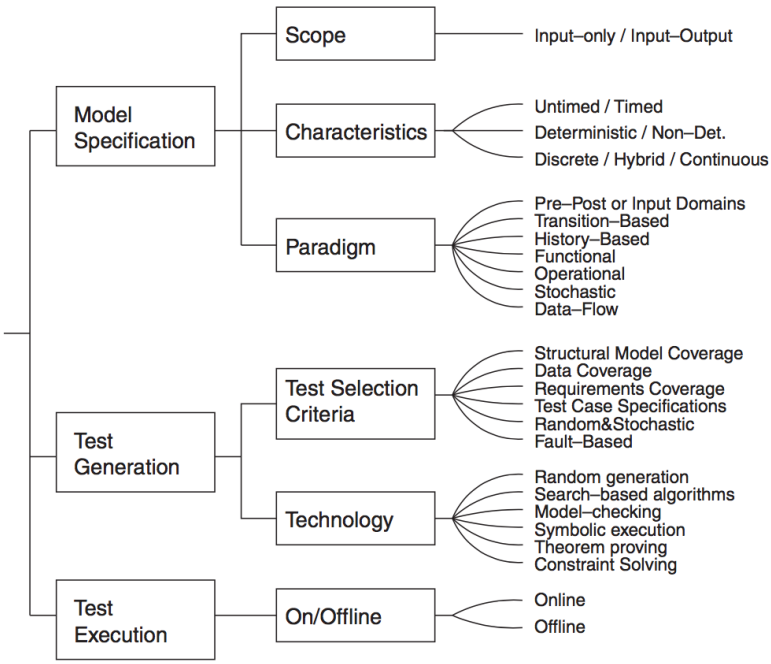

Figure 2.1: The overview of the MBT characteristics [1].

2.2.1 Model Specification

The model specification stage considers how the model should be built, the properties of the model and what data should be given as input. Furthermore, it also considers how the system is constructed and which model that could be suitable. The model specification has three dimensions; scope, characteristics and paradigm.

Scope - the scope of the model is a binary decision and specifies whether the model only considers inputs to the SUT or if it considers the expected output based on

the given input to the SUT. While the input-only model is easier to model, with the drawback that it is harder to verify the correctness of the result, the input-output model is more complex since it requires the correct output in advance for all given inputs to the system. This often requires knowledge about the environment and the behaviour of the system. However, it has the advantage to be able to act as an oracle and verify whether the output is correct or not.

Characteristics - specifies whether the SUT is continuous or event-discrete, if there are any timing issues and if the system is deterministic or non-deterministic. The research within the MBT field has mostly focused on event-discrete systems but continuous models are subject of on-going research [1].

Paradigm - describes the notation that is used to describe the model. One variant is transition-based notations where the focus is to describe transitions between different states, typically illustrated as a Final State Machine (FSM) i.e. a graph of nodes and arcs. Where the nodes represent the different states of the SUT and the arcs the actions of the SUT. Another notation is stochastic notations, where Markov chains could be used to describe a system with a probabilistic model of the events. It is usually used for describing the environment rather than the SUT.

2.2.2 Test Generation

Test generation creates test cases based on the given set of models specified in the model specification. Test generation has two dimensions; test selection criteria and technology.

Test selection - guides and control how the tests should be selected and how to measure the coverage. Selecting the best criteria in general is not possible [1]. It is the assignment for the test engineer to set up the configuration of the test generation to choose relevant test selection criteria and test the specifications of the requirements considering e.g. performance or security.

Technology - one of the most attractive characteristics of MBT is its ability to automate the test design process [1]. Given an input model and test case specifications, test cases can be obtained by applying a test generation technology. There exist different algorithms which are able to automatically derive test cases, e.g. search-based algorithms, model-checking, constraint solving, random generation, theorem proving or symbolic proving. Search-based algorithms take the model graph of the SUT as input and fulfill the coverage criteria by e.g. using the Chinese Postman algorithm. This approach makes it possible to catch corner cases that easily could be missed in manual tests. Constraint solving is another technique that can be used to set a coverage criteria. This approach is useful in combinatorial n-wise testing where data selection needs to be done in a complex domain. Tools used for test generation in MBT is usually considering multiple techniques to achieve automated test generation based on a model since the techniques have different focuses.

2.2.3 Test Execution

Test execution is the stage that is concerned with the test execution on the SUT. It verifies the result and the relation between the test case generation and test execution. Test execution can be performed either online or offline from the test generation.

Online/Offline - While offline testing is the easiest approach and means that the tests are generated strictly before any test execution is performed, the online test execution makes it possible for test generation algorithms to react on the output of the SUT. By using online testing, which executes each generated test before creating the next, a non-deterministic SUTs tests can be optimized since the test generation algorithm know which path a SUT has chosen and could therefore follow the same path again for further tests [1]. However, online execution is only applicable in an input-expected output model since the output needs to be able to determine whether the tests is successful or not.

2.2.4 Advantages

MBT enables an automatic generation of test cases and is able to guide the tests to be relevant and limited to a feasible amount. By having a model with an abstract view of the SUT it is easier to understand, modify and maintain it. After an update or modification, it is easy to generate new test cases for the updated system. Additionally, by guiding an algorithm to create test cases of the SUT instead of a human to review and design the test cases, the risk of missing corner cases and complex vulnerabilities will be decreased resulting in a higher test quality [23]. Especially in an advanced system where a computer can find more complex combinations that are difficult for a human brain to find. By using an online test execution, it is also possible to test non-deterministic systems which are depended on earlier actions and almost impossible for a human brain to find. However, MBT is mainly focusing on testing the functionality of the SUT rather than the security aspect.

2.3 Model-based Security Testing

Model-based security testing is based on models for automatic generation of test cases to verify security requirements [2] [3] [10]. Testing of security requirements have existed for a long time, however, MBST is a relatively new research field and almost only applied in academia [10]. Furthermore, Felderer et.al. mentions that MBST is of high relevance for industrial applications and can be seen as a combination of MBT and ST. It could also be described as MBT of security requirements. A graphical representation of the relation between MBST, MBT and ST can be seen in Figure 2.2.

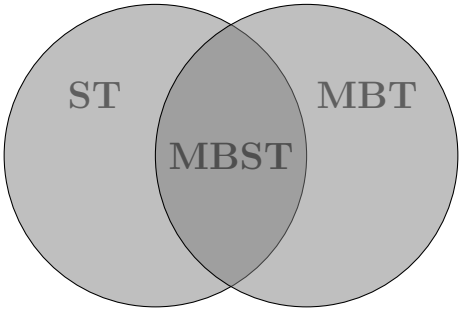

Figure 2.2: A venn diagram illustrating the relation between model-based security testing, model-based testing and security testing.

MBST makes use of models to generate test cases and includes security testing approaches e.g. penetration testing and fuzz testing. Felderer et.al. [2] presents new criteria that concern security, which complements the MBT classification scheme presented in Section 2.2. The two criteria are Filter criteria and Evidence criteria which contain several attributes where one or a combination of these are selected. Filter criteria includes the Model of System Security, Security Model of the Environment and Explicit Test selection criterion which is illustrated in Figure 2.3. With these combined, it is possible to, with one or several models, generate specific test cases of interest for security. Furthermore, it also includes the different security testing techniques, mentioned in Section 2.1 e.g. fuzz testing, vulnerability scan testing and risk-based testing. Evidence criteria include maturity of an evaluated system, evidence measures and evidence level. Furthermore, combining these are important to determine the applicability and utility of MBST in industry. Filter and Evidence criteria are more thoroughly described below in Section 2.3.1 and Section 2.3.2. An overview of the MBST classification criteria can be seen below in Figure 2.3.

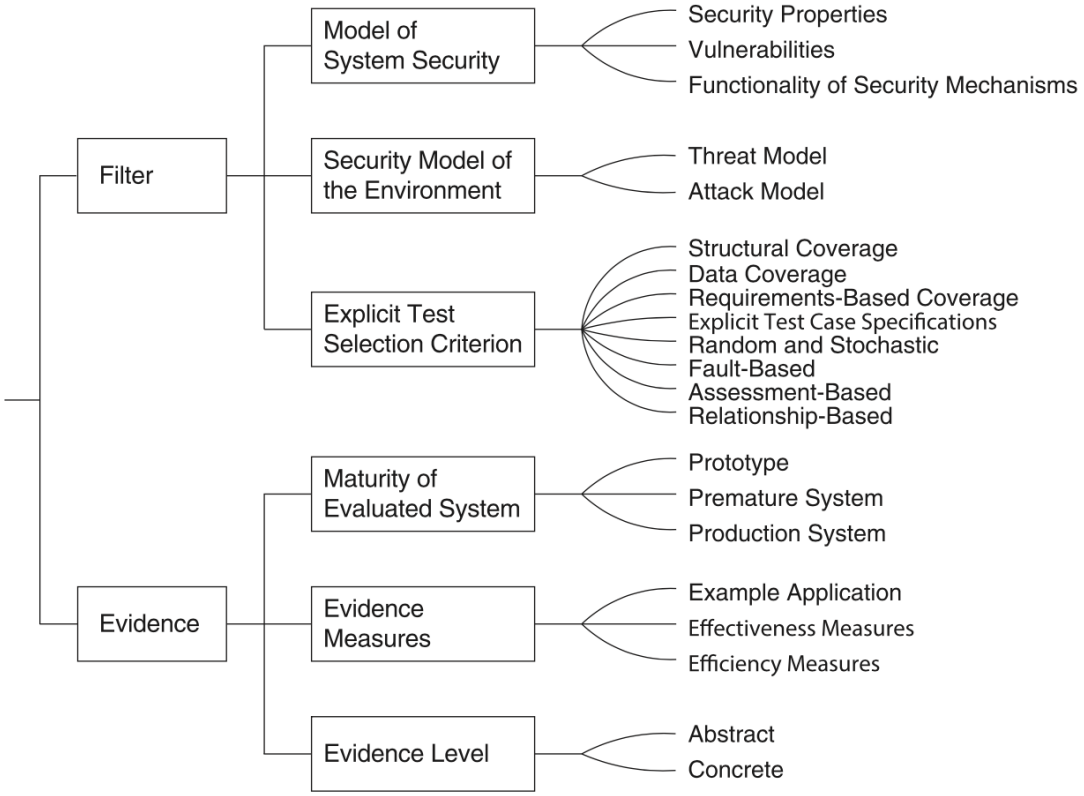

Figure 2.3: Overview of the MBST classification criteria. Picture taken from [2].

2.3.1 Filter Criteria

The filter criteria contains security-specific models, its environment and the test selection criteria. Therefore, it describes both the security test objectives and what to be modeled which makes it possible to generate relevant test cases and the model specification. Below, each of the different properties of the Filter criteria is described more thoroughly.

Model of System Security

Model of system security are one or several models, covering a part of a SUT and focuses on its security requirements. Security properties, vulnerabilities and functionality of security requirements are the different properties of a model of system security which describes the model or models of the SUT. It is possible to cover several of these properties by using a composition between models. The properties are briefly described below.

Security Properties - a model of security requirements e.g. confidentiality, integrity, availability, authorization and authentication.

Vulnerabilities - a model containing the vulnerabilities of the SUT.

Functionality of Security Mechanisms - a model of functional security which are defined by security specifications of the SUT.

Security Model of the Environment

The security model of the environment are models describing the security issues of an SUT. The difference between modelling a vulnerability and an exploit is that a vulnerability covers a part of the SUT compared to an exploit that covers the environment of the SUT [2]. Although, an exploit is a description how a vulnerability should target with specific attacks or a description of the vulnerability, making the exploit a threat. Therefore, there exists two different types of security models of the environment: threat model and attack model which can be composed. While a threat model describes the threats of an SUT, an attack model describes types of attacks to exploit different vulnerabilities of the SUT.

Explicit Test Selection Criterion

Test selection criteria make it possible to choose relevant test cases. It defines which paths to take depending on the different properties e.g. vulnerabilities and attacks. Test selection criteria are used to control the generation of test cases, i.e. it acts as a guidance for the test generation. Different types criteria are listed below.

Structural Model Coverage - a criteria that exploits the structure of a model. It verifies that every part of the model has been extensively tested e.g. every state has been visited in the model.

Data Coverage - filter only a subset of test cases that still represent the whole data set. The basic idea is to divide the large data set into different classes and choose candidates from each class, with the assumption that each candidate will represent the group in terms of failure detection.

Requirement-based Coverage - ensure that all requirements of the SUT have been fulfilled from the set of generated tests if the requirements can be determined from the model.

Ad-hoc test case specifications - makes it possible to focus on testing the important parts in the model i.e. limit the paths to be tested. A test case specification needs to be described by the tester in order to determine which test cases to be generated.

Fault-based Criteria - focus on finding faults in the SUT. A common approach to finding faults is to consider mutation coverage. This approach mutates the model, then generating tests that can be compared to the mutated model and the original model.

Random and Stochastic - criteria that consider probabilities of actions or randomly generated input.

2.3.2 Evidence Criteria

Evidence criteria is a valuable measure for evaluating if MBST is an applicable or useful approach. It depends on the maturity of the evaluated system, evidence measures and the evidence level which each contains several properties. It is an optional criteria since it evaluates the applicability or usefulness of the approach and can be seen as an extension. The evaluation is performed by manual analysis of the SUT.

Maturity of Evaluated System

The maturity of an evaluated system is the criteria which evaluates if the SUT is appropriate for an MBST approach. It depends on one of the following values Prototype, Premature System and Production System which are described further below.

Prototype - a first early version of a system to test the concept. It means that it is under development and limited. It is not a final product.

Premature System - a running system. It is more mature than a prototype but not a final product.

Production System - the final product of a running system.

Evidence Measures

Felderer et.al. describes the evidence measures as a determination of the qualitative and quantitative assessment criteria to evaluate a MBST approach on the SUT [2]. It contains of the following properties: Example Application, Effectiveness Measures and Efficiency Measures which are described below [24].

Example Application - an approach that is used in a text-based context. It describes the feasibility of the approach in a qualitative way.

Effectiveness Measures - measures the input-output and compare it to the expected input-output. It could be measured by number of faults divided by number of test cases to get a ratio of faults per tests.

Efficiency Measures - measures the efficiency by time and cost. It could measure by how many faults that are found depending on the cost to produce test cases.

Evidence Level

The evidence level evaluates whether an application can perform non-executable abstract test cases or executable test cases.

Abstract - the approach is evaluated for non-executable test cases. The evidence measures are limited because it is not possible to measure the concrete effect of the approach on the SUT.

Executable - the approach is evaluated for the executable test-cases on the SUT. The evidence measures are not limited to this level compared to the abstract level.

2.4 Related Work

MBST is a relatively new research field which extends the MBT approach with a security aspect. There is ongoing research and the topic has high relevance for an industrial contexts. Today, practical implementations mostly exist in academia and are highly theoretical [3]. This thesis will investigate the possibility to apply a MBST approach within the automotive industry.

There exists a case study on smart cards applying MBST based on UMLsec models [25]. The aim of the research was to investigate if it was possible to, in a systematic way, gain confidence that the implemented security-critical system holds. Jürjens presents a systematic approach of security testing using a formal specification language to generate test sequences to find vulnerabilities. More particularly, they focus on testing the load transaction, a central part of Common Electronic Purse Specifications (CEPS). The specification of the load transaction is slightly simplified by ignoring security-irrelevant details but including exception processing. It is possible to verify the resistance of an implemented load transaction function for CEPS by using UMLsec models with a specification-based testing approach. Where the specifications could be, for example, that a component should reach an accepted state. The paper also concludes that it is much harder to test a system for an absence of undesired than for the presence of desired behaviour.

2012, Ina Schieferdecker et al. published a paper providing a survey of MBST techniques and related models [10]. According to their paper, MBST includes different techniques e.g. security functional testing, model-based fuzzing, risk- and threat-oriented testing and security test patterns. In software security testing, there are two different approaches that can be used to verify and validate that the SUT fulfill its security requirements, functional security testing and security vulnerability testing. However, they state that security testing lacks systematic approaches. They provide three different types of models that could be used for test generation, but to securing the system it is needed to base the tests on several models since they consider different perspectives of covering security. The three suggested models are Architectural and functional models, Threat, faults and risk models and Weakness

and vulnerabilities models. Architectural and functional models consider the system requirements regarding the behaviour of the SUT, these models exist on a different level of abstraction and granularity. They are typically considering the structure and properties of the SUT. While threat, faults and risk models focus on causes and consequences an eventual system failure or vulnerability might implicate, weakness and vulnerabilities models focus on the fault or vulnerability itself. Finally, they also mention an European ITEA2 project called Development and Industrial Application of Multi-Domain Security Testing Technique (DIAMONDS) that are developing two main approaches for MBST: Risk-based security testing and Model-based fuzzing. Model-based fuzzing is one kind of white box testing since it considers the knowledge about the structure to systematically generate relevant tests, this approach is mainly used to detect flaws and vulnerabilities in the SUT that usually not are revealed by using incorrect input data. On the other hand, risk-based security testing can be used to optimize the overall test process. Since the result of a risk analysis could be used as a guidance for what to prioritize and how the requirements are fulfilled, it is possible to make the test process more efficient. For this thesis, the system is considered as a black box where only the requirements for the SUT is known. Therefore, neither risk-based security testing nor Model-based fuzzing seems to be useful approaches.

Research regarding **firewall testing** is a widespread area with different approaches. A common way of testing firewalls is to test for vulnerabilities in the implementation, however, there is research that focuses on testing the firewalls policy. Testing a firewalls policy could also be described as conformance testing of a firewall which is similar to our approach of testing. Several papers of specification-based firewall testing have been proposed. Jürjens and Wimmel [26] propose a new way of testing firewalls, specification-based firewall testing. It works by first introducing a formal model of the network to automatically derive test cases. El-Atawy et al. [27] [28] propose a new way of handling policies with a policy segmentation technique. As a result, it is possible to measure a policy which makes it possible to perform more thoroughly tests on the important segments. Additionally, they present a new framework that provides a better coverage of the different states of the firewall. Brucker et al. [20] present a case study with the model-based tool HOL-TESTGEN. It is based on a formal model of firewalls and the requirements are described with higher-order logic (HOL). The paper presents a method for minimization of requirements and discusses different test plans for test specification.

3

Methodology

This thesis is driven by an interest and need from VCC to improve the security verification and detect anomalies as early as possible in the development process. There is a need for improving the confidence of satisfying the security requirements as the cars become more complex and connected to the Internet for providing different services. One major reason that VCC want to achieve an early verification in the development phase is to avoid expensive and intractable vulnerabilities and bugs late in the development as it mostly is performed today. Additionally, a more automated process will improve the efficiency in the development and thereby decrease the develop time in total. Hopefully, MBST could be one approach that fulfills this desirable improvement.

An investigation has been conducted at VCC to analyze the current process and the handling of security requirements to evaluate the maturity for implementing an MBST. Since MBST is based on models, the modeling maturity for the different departments is relevant. VCC involves modelling a lot into the daily development process e.g. Simulink models, but only in a functional perspective. The maturity of managing the security requirements is still in a text-based representation, therefore, it is a large gap between the models presented in the active research today and the text-based models at VCC. Furthermore, the maturity of the specification and verification of security requirements are also of importance. Since the models that exist today are usually only considering the functionality aspect, the maturity of following security requirements is of highest relevance for this thesis since the model should be based on the given requirements. This was mainly why the infotainment department were chosen to be further investigated for this thesis. The infotainment department is responsible for the interaction between the car and the driver. This thesis has focused on the firewall in the connectivity component since it is the most exposed part to potential remote attacks over the network. The firewall considered is located late in the development phase and has been penetration tested. VCC has several functionalities that requires Internet connection e.g. Volvo on Call [29], therefore, the security aspect is highly relevant to consider for these functionalities and this thesis.

One reason that the infotainment was chosen except the maturity for security requirements, is that suppliers to different ECUs only takes responsibility for their own ECU and not the overall system. This put VCC in a position where they have

the main responsibility to ensure that the system work as it should with all ECUs connected and prevent security threats. Is there any problem with an ECU, the responsible supplier need to be contacted and solve the problem, which usually are very time consuming since it always is a question whether it is stated as a requirement in the specification given at the initialization of the project or not. This might be improved by being able to send test cases to the supplier that must be fulfilled before delivery to VCC. This thesis will investigate the possibility to achieve this by apply a MBST approach.

Furthermore, an evaluation of different MBT tools has been conducted to decide if they could be suitable for a MBST approach since, from our knowledge, it only exists one MBST tool which is infeasible to use for this thesis [30]. It is a tool presented in a paper that needed a license to be allowed to use and thereby not feasible for this thesis. The MBT tool evaluation is a scientific contribution since there has not been any comparison between MBT tool and if they could be applicable for a MBST approach. The conclusion of the evaluation is in regard to the scope of this thesis and its prerequisites.

3.1 Evaluation of Tools

There exist many different tools available for applying MBT. They all have advantages and disadvantages dependent on the SUT. The tools will be systematic evaluated of how suitable they are for applying MBST at VCC by considering availability, the input model, the effort to implement the approach and finally if they can perform tests online and/or offline. The availability of the tool is either commercial or open-source, since there is no budget for this thesis, open-source tools are the only tools considered. The important part considered for the input model is that all the requirements can be covered by the model. It is also important to consider that the effort to adapt the VCC's requirements into the input model are feasible since it would cost both time and money to deploy new techniques. If the tool supports online and/or offline testing is also considered but is not main priority since tests could be integrated into VCC's existing testing framework. The result of the evaluation for the different tools is presented in Table 3.1.

SpecExplorer

SpecExplorer [31] is a tool created by Microsoft. It is an extension for Microsoft Visual Studio and provides a graphical visualization analysis and performs a validity check of the behaviour models of the SUT and finally generate test cases based on the models. The input is a program model, which is implemented in C# .NET, and provide opportunity to express both interaction- and state-oriented modeling styles. The tool take usage of the program model stating the set of rules together with a behavioral description to make testable models of the SUT. Finally, SpecExplorer

is compatible with both online and offline test execution.

License Type: Spec Explorer is a commercial tool from Microsoft for model-based testing in Microsoft Visual Studio. **Input:** The input for test generation is a program model specified in C#. **Generated output:** The generated output from Spec Explorer is a sequence of actions which are executed online or offline. **Security Attributes:** From our knowledge, no security attributes are described in the program models. Therefore, only functional security can be considered. **Effort:** To use Spec Explorer, it benefits with knowledge in C# due to the input models. Spec Explorer is an extension to Visual Studio for creating models of software behavior and for graphical representation. Therefore, it would benefit to have knowledge with the Microsoft Visual Studio software which facilitate the effort to learn the tool.

GraphWalker

GraphWalker [32] is an open-source MBT framework in Java which takes models in the shape of directed graphs as input. Given the input model and guidance for the generator, Graphwalker uses mathematical algorithms to generate test cases in form of different paths through the model until the given stop conditions are fulfilled. The path is a series of pairs of edges and vertices, where actions are specified in the edges and verification of the action in the vertex. GraphWalker is compatible with both online and offline test execution.

License Type: GraphWalker is an open-source tool for model-based testing in Java. **Input:** The input for test generation in GraphWalker is a FSM. **Generated output:** The generated output is a sequence of actions which are executed online or offline. **Security Attributes:** From our knowledge, no security attributes are described in the FSM models. Therefore, only functional security can be considered. **Effort:** It benefits to have knowledge in modelling FSM's since it is used as input model to Graphwalker. Java is also relevant since the models are modelled in Java. GraphWalker is well documented with several examples which facilitates the effort to learning the tool.

PyModel

PyModel [33] is an open-source MBT framework in Python. It has three main program that makes it possible to create and analyze the model, generates test cases and executes them on the SUT. The input model could be represented by either a model program, a FSM and test suites. PyModel can use a composition of models to combine several models into a new model called product. The test generation could be based on either random generation or be guided by a strategy provided by the user. It is also possible to visualize the model's behaviour by using an analyzer. PyModel is compatible with both online and offline test execution.

License Type: PyModel is an open-source project for model-based testing and a framework in Python. **Input:** The input for PyModel is a program model specified in Python. **Generated Output:** The generated output are a sequence of actions which are executed online or offline. **Security Attributes:** From our knowledge, no security attributes are modelled in the program models. Therefore, only functional security can be considered. **Effort:** To make use of PyModel, it would benefit to be familiar with Python since the models are created in Python. It also benefits to have knowledge in modelling FSM since program models are similar. PyModel is well documented with several examples available which facilitates the effort of learning the tool.

T-VEC Tester

T-VEC Tester [34] is a tool to automatically verify and test Simulink models by generating test vectors passing all paths in a system. It takes usage of an advanced test generation algorithm that analyzing the constraints in the models and select tests to expose every condition in the SUT and extreme values. Therefore, the generated test cases are very effective to detect flaws and errors. It is based on various structural coverage criteria. T-VEC Tester is a commercial tool and from our knowledge, it makes use of offline testing.

License Type: T-VEC Tester is commercial tool from T-VEC Technologies for model-based testing. **Input:** The input for test generation is Simulink models. **Generated output:** The generated output for T-VEC Tester are test vectors to cover all different paths in the model. T-VEC Tester makes use of offline testing. **Security Attributes:** From our knowledge, no security attributes are specified in the Simulink models. Therefore, only functional security can be considered. **Effort:** To use T-VEC Tester, it is beneficial to have knowledge in modelling Simulink models. T-VEC Tester are well documented which facilitates the effort to learning the tool.

4Test

4Test [35] is a commercial tool that makes use of text-based models as input to generate test cases, i.e. no flows or diagrams are needed. The advantage of the text-based models is that it is human-readable and improve the understandable of the tests. The text-based models are based on the Gherkin language which is a domain specific language [36]. 4Test combines Constraint Driven Testing and Keyword Driven API Testing and aims to generate executable tests for test automation tools. From our knowledge, 4Test makes use of offline testing.

License Type: 4Test is a commercial tool from 4D Soft Kth where a test design method and a test automation method are combined. **Input:** The input for test generation is a test-based model described with the Gherkin language. **Generated**

output: The generated output from 4Test is executable test cases. 4Test makes use of offline testing. **Security Attributes:** From our knowledge, no security attributes are specified in the input models. Therefore, only functional security can be considered. **Effort:** Since it makes use of the Gherkin language to specify their input models, it is very easy to understand the models and constraints. However, it is briefly documented with no complete example which makes the effort to learn the tool harder.

HOL-TESTGEN

HOL-TESTGEN is an open-source tool that takes formal models as input and generates test-cases [37]. The formal models are written in high-order logic (HOL) and generates test scripts. As a result of using HOL, it is possible to use minimization to reduce a number of test-cases produced to fulfill the requirements. From our knowledge, it makes use of offline testing.

License Type: HOL-TESTGEN is an open-source tool for model-based testing developed at the University of Sheffield. **Input:** The input for test generation is a model described with HOL. **Generated output:** The generated output from HOL-TESTGEN is abstract test cases which can be connected to a test-harness which enables both online and offline testing. **Security Attributes:** From our knowledge, no security attributes are specified in the HOL models. Therefore, only functional security can be considered. **Effort:** The effort to learn HOL-TESTGEN is more challenging compared to the other tools in this evaluation since it is based on mathematics and formal models. Thereby knowledge within HOL and mathematics are beneficial. It is well documented with several publicized papers with extensive examples which facilitate to effort to learn the tool.

Table 3.1: Summarized tools for MBT.

MBT Tools				
Name	License	Input model	Execution	Sec. Attr.
PyModel	Open-source	PM[1] Python	Online & offline	No
Graphwalker	Open-source	FSM	Online & offline	No
Spec Explorer	Commercial	PM[1] C#	Online & offline	No
T-VEC Tester	Commercial	Simulink model	Offline	No
4Test	Commercial	Text model	Offline	No
HOL-TESTGEN	Open-source	HOL	Offline	No

The conclusion for this thesis regarding the evaluation of tools considers mainly the license type, input model and effort. Therefore, it is of importance that the tool is open-source and compatible with how the security requirements are modelled at

[1]Program models. Describes the behaviour of infinite systems.

VCC today. The effort is considered to determine if it is suitable to model our own input model, based on the requirements, to be applicable to the corresponding tool. Although, since only open-source tools are considered, three different tools are extracted from the evaluation; PyModel, GraphWalker and HOL-TESTGEN.

Since there is a gap between the input models for each tool and VCC's approach to model requirements, the applicability of these tools at VCC are limited. VCC's maturity in state-models are low since state-models becomes very complex in larger systems. Furthermore, there is no maturity for making models in HOL at VCC which is as expected since it has only been applied in academia. Another consideration from the evaluation is that there is no tool that specifies security attributes, they are only able to handle functional security. Therefore, we could conclude that none of the tools would be of any help to introduce a MBST approach at VCC. There is too little experience for making state-models and HOL-models, and there is no time for education of these. The final conclusion regarding the MBT tools are that this thesis need to develop a tool that handles text-based models, which would make it feasible to apply at VCC.

From this evaluation of the tools, the following **research questions** can be answered.

- Are there any available tools for MBST?
  - What is required from VCC to use available tools for MBST?

    From our knowledge, there is only one MBST tool that has been applied in practice. It is a commercial tool and requires UMLsec models as input. Therefore, it is unfeasible for this thesis to apply this approach considering the modeling maturity at VCC today. Therefore, an investigation of existing MBT tools were performed to determine whether they could be used for a MBST approach. Unfortunately, it was concluded that neither of the MBT tools at their current stage could to be used at VCC. There is a lack of maturity at VCC for modelling state-models and formail-models. Today, text-based requirements are the standard for specifying requirements which are very far from the MBT tools input models. Finally, to make it possible for VCC to use existing tools, we suggest the following:

    MBST tool

    * Adapt existing models into UMLsec models.
    * Include security attributes in their UML models.

    MBT tools

    * Adapt existing functional requirements into models
    * Increase maturity in making state-models or formal models (HOL).

3.2 Approach

The approach for solving the demand of an earlier verification of security requirements is by introducing a new systematic approach for verifying security requirements and implementing our own tool, Awesome Firewall Tool (AFT). Our MBST approach is based on text-based models since the requirements used at VCC today are illustrated in text. The requirements at VCC are presented at an abstract level to make it easier for people without a technical background to understand. Therefore, the text-based input model to AFT is derived from the requirements and specified in more detail, e.g. whitelisted ports are listed. Furthermore, the input model is non-complicated to adapt from the requirements at VCC today. AFT will generate and execute different commands in order to verify different requirements specified in the input model and if there exist any vulnerabilities. The result of the program will generate a report which describes the vulnerabilities and if the requirements are fulfilled or not. A picture showing a brief overview of how the structure of the approach is presented in Figure 3.1.

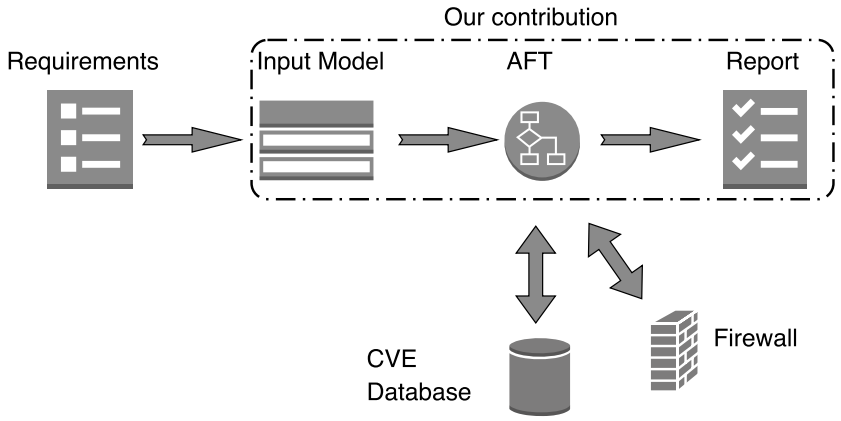

Figure 3.1: An overview of the high-level structure of the approach.

3.2.1 Classification of our Approach

This approach, in relation to the classification of MBST described in Section 2.3, is as follows. The filter criteria consist of three different parts: model of system security, security model of the environment and explicit test selection criterion. The considered property for the model of system security is *functionality of security mechanisms* since the approach aims to verify security requirements for a firewall i.e. testing a security mechanism. The model received from VCC is a text-based model, describing the security specification of a firewall. Therefore, it is convenient to translate the VCC requirements to a more formal model which the AFT tool use as input. The input model is a more detailed form of stating the requirements which ensure coverage when translating them. This approach focus on the *requirement-based coverage* in explicit test selection criterion since the model of system security

only considers the different security requirements of the firewall. For this thesis, the security model of the environment is not considered because it is no need to determine if the requirements are fulfilled or not.

3.2.2 Architecture

The purpose of the different steps in the approach presented in Figure 3.1 is more thoroughly described in this section. Firstly, a description of the VCC requirements are presented with examples of common requirements for firewalls. Based on those requirements, the input model is specified. AFT's aim is presented with its high-level structure. Lastly, the report that is generated from the AFT is described.

VCC requirements

The requirements are in a text-based document and are described in a very high-level perspective. The major reason to why it is stated in a high-level is because this is general requirements for all firewalls within the vehicle with different purposes and different locations in the vehicle. Therefore, a more detailed requirement could cause an unwanted limitation of the application. Instead, VCC provide the general policy that every firewall should consider but the technical details are different for different firewalls. The requirements are given below.

Firewall Requirements

Firewalls shall be used to block unintended IP traffic on internal (IP traffic between ECUs) as well as external network interfaces.

The firewall shall perform stateful packet inspection, keep track of connection states and have the following rule set:

Outgoing traffic on external interfaces:

- Accept packets to whitelisted destination ports and/or IP addresses only. Only source ports above 1023 allowed.
- Drop packets to non-whitelisted destination ports, i.e. use a default drop policy

Incoming traffic on external interfaces:

- Accept solicited incoming packets for active connections initiated in the vehicle, only.
- Drop unsolicited incoming packets, i.e. use a default drop policy

Incoming traffic on internal interfaces:

- Accept incoming packets on whitelisted ports only
- Drop incoming packets on non-whitelisted ports, i.e. use a default drop policy

Firewall and routing rules when Vehicle ECU is WiFi HotSpot:

- For devices on the in-vehicle hot-spot network, the firewall shall only allow access to Internet and not allow access to resources on the ECUs.
- The firewall shall perform stateful packet inspection, keep track of the state of all connections and have the following rule set in addition to the above:
  - Accept outgoing packets from a device on the in-vehicle hot-spot network. All destination ports and IP addresses allowed.
  - Accept incoming packets for active connections initiated from a device on the in-vehicle hot-spot network.

Input Model

The requirements are adapted into the text-based input model, which is presented in a text file that AFT can interpret. For AFT to be able to perform tests on the firewall, more detailed data than the one stated in VCCs existing requirements are needed, e.g. the IP address to the target i.e. the device behind the firewall. As shown in Figure 3.2, the input model is divided into four different sections: general information, incoming traffic on external interfaces, incoming traffic on internal interfaces and outgoing traffic. The input model is divided into these categories since the different interfaces of the firewall have different requirements e.g. there are different services provided at the internal network compared to the external network and thereby other requirements are needed. The general information is covering requirements that could be set for multiple interfaces to avoid repetition. In general information, requirements that are independent on whether the traffic is incoming or outgoing are specified i.e. if the firewall should have stateful packet inspection and if it should have a default drop or default accept policy. For the incoming and outgoing categories, more details from the high-level requirements are needed to be specified e.g. which ports that are whitelisted or blacklisted.

```
Inputfile Template
-- General --
Stateful: True
Policy: Default drop
-- Incoming external --
Target IP address: scanme.nmap.org
Whitelisted ports: [22, 53, 80]
-- Incoming internal --
Target IP address: 192.168.1/30
Whitelisted ports: [25, 53, 80, 44505]
-- Outgoing --
## type: ports or ip or combination
Type:
Target IP address:
Whitelisted ports: []
Whitelisted ip addresses: [scanme.nmap.org]
Combination: []
```

Figure 3.2: An overview of the input model.

The aim for the input model is to be able to cover all existing requirements but still be easy to adapt from the existing approach for handling requirements at VCC. As illustrating in the bullet list below, whitelisted ports that should be allowed to access could be stated as a list and the IP address to the SUT that should be evaluated.

- Whitelisted ports for incoming traffic: [22, 53, 80]
- Target IP Address: 192.168.1.2

AFT

AFT is the core of the approach. It is the tool where the actual test generation and test execution is performed. Based on the text-based input model, the tool performs relevant test cases to verify the requirements from the input model. The goal is to be able to verify if the requirements from the input model is fulfilled and to present the requirement-based coverage. The AFT considers traffic from three different interfaces; outgoing traffic on external interface, incoming traffic on internal interface and incoming traffic on external interface. Furthermore, for each of these network interfaces, different requirements can be considered and the goal is to cover all of them with both positive and negative testing.

AFT will make use of port scanning since it is a powerful way for determine if traffic are accepted or dropped. Then it will be possible to detect if a port is open or closed and to determine if a firewall performs a stateful packet inspection. Additional, AFT should also be able to determine if there exist vulnerabilities on the services at the whitelisted ports and present that information. This approach is presented in Figure 3.3, where the incoming traffic represent both incoming traffic on internal

and external interface since they will perform the same test but with different input data. The main goal with AFT is that it should in an automated way determine if the requirements are fulfilled or not.

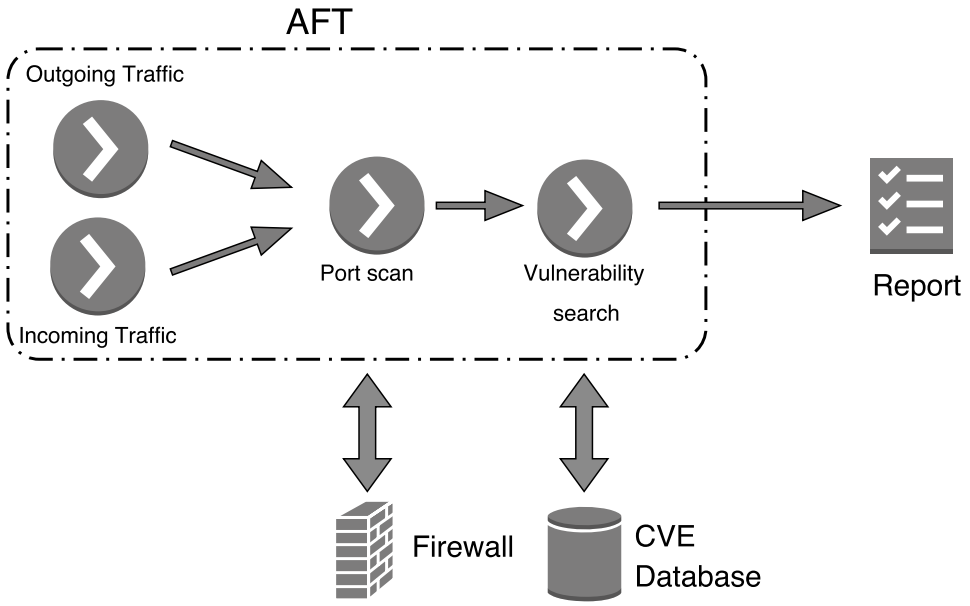

Figure 3.3: A simplified overview of the technical architecture of AFT.

Report

The output should visualize for the stakeholder whether the firewall fulfill the given requirements or not and if there are any vulnerabilities detected. The report is generated part wise during the execution of AFT, as soon as a requirement is verified it is appended to the report. The purpose of the report is to present a simplified view for the stakeholder whether the requirements are fulfilled or not. Being able to present an overview of how the SUT is following its requirements is beneficial and of high importance since the software developed for the automotive industry differs from software developed for personal computers in a way that patching a bug is a balance between the potential risk and the cost for the patch. To recall cars on the road to a service update entails a huge cost. Therefore, a found vulnerability needs to be proven to be critical to be worth the cost for patching it. To achieve this, it would be desirable to present how a certain vulnerability could be exploited and get a perception how complicated the exploit is to convince the responsible people to prioritize the measures needed. The level of importance for each vulnerability will be stated in the report to simplify the process of which vulnerabilities to prioritize.

4

Implementation

This section describes the implementation process and decisions made along the thesis. It is divided into the three parts of our contribution: Input model, AFT and the generated report. Finally, an illustration of the MBST approach with the AFT tool is presented.

4.1 Input Model

To create the input model, AFT can generate a template that the stakeholder fill with the relevant data for the SUT. The template can be generated by executing the command seen in Listing 4.1.

Listing 4.1: Initialization of the template for the input model.

```
./nmap.py --template
```

The template created after the command is presented in Figure 4.1. In the template, the different sections can be filled with the information from the requirements. If the firewall should perform a stateful packet inspection, stateful is set to true in the input model, otherwise false. This thesis has only considered default drop as policy since it is the most common policy and is therefore the only option to fill in. The IP address is the target address for each interface the scan is performed on. The whitelisted ports in each interface is the allowed ports. Under the outgoing interface in the template, the type can be filled in with three different types of options: ports, IP or combination. The *port* option should be chosen if the requirement regarding outgoing traffic only mentions whitelisted ports, *IP* if the requirement only mentions whitelisted IP addresses and *combination* if it mentions a combination between IP addresses and ports. The whitelisted ports is filled if the *ports* is chosen, whitelisted ip addresses is filled if the *IP* is chosen and combination is filled if the *combination* is chosen.

```
Inputfile Template
-- General --
Stateful: True or False
Policy: Default drop or Default accept
-- Incoming external --
Target IP address: IP to the target
Whitelisted ports: [Port,Port]
-- Incoming internal --
Target IP address: IP to the target
Whitelisted ports: [Port,Port]
-- Outgoing --
Type: Specify the type of outgoing traffic to test
Target IP address: IP address to the external target
Whitelisted ports: [port, port]
Whitelisted IP addresses: [IP, IP]
Combination: [(IP: port, port), (IP: port)]
```

Figure 4.1: An input model template.

4.2 AFT

VCC has an increasing focus on continuous integration testing on different level of abstraction. This is performed by developing Awesome Framework, a test framework implemented in Python. AF is compatible to execute tests suits on different system in the vehicle or even for a complete vehicle. Thereby, AFT is implemented in Python as well to simplify the possibility for integration with AF. The tool considers the functional security aspect i.e. verifying that security mechanisms are behaving as intended. Security attributes like confidentiality and integrity will be excluded in the tool. AFT consider three aspects, messages sent internally, messages sent from the internal network to an external network and messages sent from an external network to the local network. To execute the different parts of the tool, the user specifies the desired part by an extra argument to the program. The architecture of AFT is presented in Figure 4.2 where the handling of the different interfaces are illustrated. Incoming traffic on internal interfaces and external interfaces are represented as one flow in the figure, even if they consider different requirements and data because they perform the same type of tests.

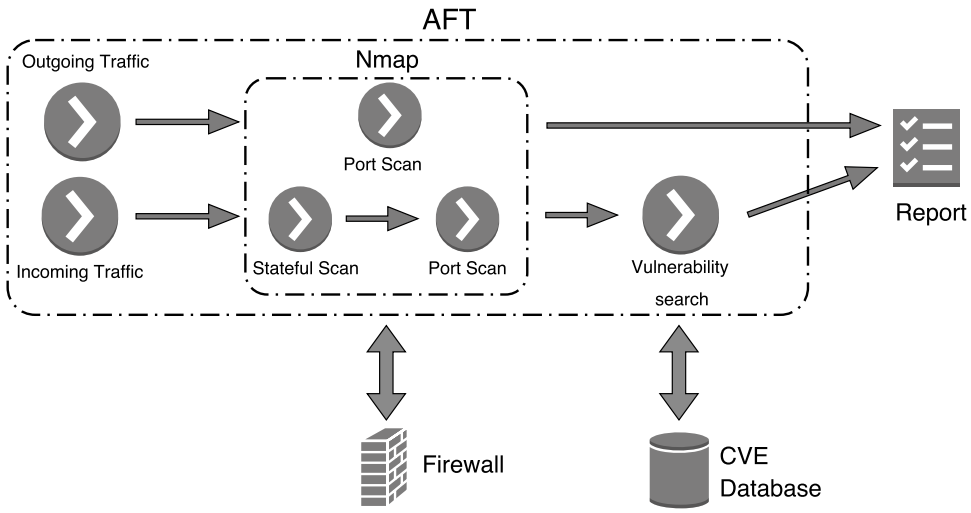

Figure 4.2: An overview of the technical architecture of AFT.

The first section of the tool is to receive and interpret the input model. Since the input model has predefined keywords, a function in AFT open the input model and search for the different words to be able to consider the requirements during the execution. In Listing 4.2, it is showed how AFT achieve the information from the input model whether the firewall should be stateful or not.

Listing 4.2: Initialization of Nmap object.

```
with open("input.in") as f:
    for line in f.readlines():
        # Check if the firewall should be stateful
        if 'Stateful' in line:
            stateful = True if
                (line.split(':')[1].rstrip().strip().lower() == 'true')
                else False
```

To simplify the implementation of AFT, two frameworks have been used. One to simplify the implementation of performing port scans with AFT, a framework for Python providing that functionality had to be found. One that provide Nmap functionality for Python programs were python-nmap[38]. It was chosen since it provided scan results in a suitable format for AFT, had thorough documentation and useful examples to follow. Another useful framework considered in AFT is cve-search [39], it provides functionality for detection of vulnerabilities for a specific service given from a scan. This is achieved by searching for CVEs stored in a database containing all known vulnerabilities with service as input. It was chosen for this thesis since it was an open-source software compatible with Python and store all CVEs in a database locally i.e. avoiding sending queries for each service to a public CVE database online which slow down the look up speed. The functionality of Nmap and the CVE database will be further described below.

Nmap

Nmap is a free and open-source tool used for network discovery and security auditing. It is widely used with thousands of downloads each day and have received numerous awards e.g. "Information Security Product of the Year" by Linux Journal, Info World and Codetalker Digest [40]. By sending raw IP packets, it can by different techniques determine which hosts that are available on the network, services (name and version) available, operating systems used, type of firewall used and much more. Nmap can be utilized on all operating systems and can be used to scan both large networks and single hosts. The different techniques that is relevant for this thesis are further described below mentioning both usage and functionality.

TCP connect() scan - makes use of the UNIX socket programming system call connect(). It is used to initiate a TCP connection on a remote host. If the connect() command succeeds with establishing a connection to the host, the responding port is open. Otherwise, if the connect() fails, it could be due to that the host is offline, port is closed or some other error that occurred. An illustration of the packet exchange for both an open and closed port is presented below in Figure 4.3 and Figure 4.4. An advantage of using the TCP connect() scan is that it easily can determine if a port is open since it will establish a connection to it. A disadvantage is that it easily can be detected on systems being scanned. However, that does not apply to this thesis since the stakeholders want the system to be scanned.

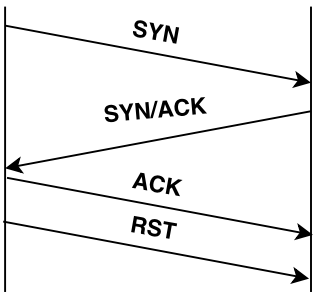

Figure 4.3: TCP Connect() scan with open port.

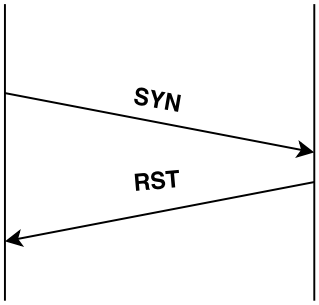

Figure 4.4: TCP Connect() scan with closed port.

Ping scan - is used to determine available hosts on the network. It is achieved by sending an ICMP ECHO REQUEST to the specified range/host and by interpreting the corresponding reply. If the received packed is an ICMP ECHO REPLY, it can be concluded that the host is online. A illustrating picture of the packet exchange is presented below in Figure 4.5. If there is no response, it could depend on ICMP packets being blocked and a TCP Ping is initiated to determine if ICMP packets are being blocked or if the host is offline. The TCP Ping scan can use two different techniques, initiating with an ACK or SYN/ACK packet to any port on the remote system (default 80). If the response is RST or SYN/ACK, the host is online. If there is no response, the remote system is either closed or the targeted port marked as "filtered" and not responding to anything.

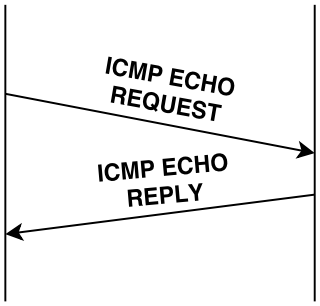

Figure 4.5: Ping scan with an open host.

UDP scan - is used to find open ports that uses UDP. The scan starts with sending a 0-byte UDP packet to each specified port on the target host. If the response is an ICMP Port Unreachable, it is considered as closed. Otherwise, it is assumed to be open. An illustration of the packet exchange is presented below in Figure 4.6 and Figure 4.7. If outgoing ICMP Port Unreachable messages are blocked, the ports will appear as open and it will be very hard to determine if the port is actually open or not. It also happens that services sometimes reply with a UDP packet. However, if there is no answer from the re-transmission, the port is classified as open|filtered.

A common practice is to limit the ICMP Port Unreachable packets that can be generated during a time period. Therefore, to avoid flooding a network with useless packets, Nmap adjusts its speed for generating packets. Therefore, a UDP scan of all 65536 ports can take more than 18 hours.

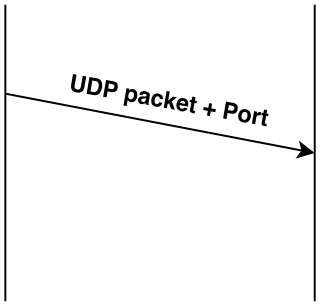

Figure 4.6: UDP scan with open port.

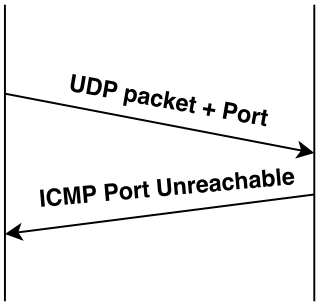

Figure 4.7: UDP scan with closed port.

Version Detection - is performed to collect information for a service running on an open port and includes the name of the service and the corresponding version. Version detection is based on a series of complex probes [40] and can use both OS fingerprinting and version detection.

ACK scan - makes it is possible to test if a firewall is stateful or not. The scan starts by sending an ACK packet to the remote host and either it responds with a RST or by dropping the packet. If the response is RST, it will mark the port as "unfiltered" and the firewall is not considered to be stateful. If there is no response, the port is

marked as "filtered" and the firewall is considered to be stateful. An illustration of the packet exchange for a stateful and stateless firewall is presented below in Figure 4.8 and Figure 4.9. However, with a ACK scan, it is not possible to determine if a port is open or not since it only marks a port as "unfiltered" or "filtered".

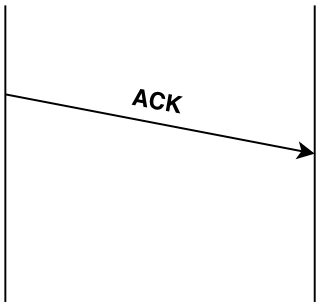

Figure 4.8: ACK scan with stateful firewall.

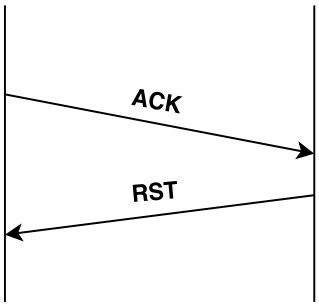

Figure 4.9: ACK scan with stateless firewall.

By taking usage of the python-nmap framework it is very easy to initiate a Nmap object which is able to take advantage of all functionality Nmap provides directly in AFT [38], as shown in Listing 4.3. By taking advantage of the string operator in Python, the scan command becomes very flexible and can perform several scans for several ports.

Listing 4.3: Initialization of a Nmap object.

```
import nmap
try:
    nm = nmap.PortScanner() # instantiate nmap.PortScanner object
    nm.scan(ip_addr, arguments='%s -p %s' % (flag, p))
except nmap.PortScannerError:
    print('Nmap not found', sys.exc_info()[0])
    sys.exit(1)
```

CVE database

CVE is a database of known vulnerabilities and exposures. Today, it is the industrial standard for vulnerabilities [41], it was initialized 1999 aiming to provide a common database to avoid having several databases referring to the same vulnerability differently. Furthermore, it also resulted in an approach to evaluate the vulnerabilities fairly since they could be compared based on the same criteria. A CVE object describes a specific vulnerability for a certain service and a certain version of that service, it is stored in a database and each object is assigned a unique id to provide efficient look up. This makes it possible to search for vulnerabilities connected to a given service and version. A CVE object consists of the unique identifier, a brief description of the problem, references and Common Vulnerability Scoring System

(CVSS). The CVSS is a scoring system grading each vulnerability within a scale between 0 and 10, where 10 is the most serious threat.

To improve the performance of queries in AFT, the CVE database is populated locally to avoid public look ups in a public CVE database. By using the commands shown in Listing 4.4, the scripts will fetch all known vulnerabilities and exposures from 2002 until 2017 and link them together in MongoDB, a noSQL database. The last of the three commands, is the updater script that also could be used whenever one want to update the database with new CVE objects.

Listing 4.4: Initialization of the CVE database.

```
./sbin/db_mgmt.py -p
./sbin/db_mgmt_cpe_dictionary.py
./sbin/db_updater.py -c
```

The CVEs could be found by searching for specific services and/or versions, e.g. as shown in Listing 4.5, a query searching for vulnerabilities for cisco's iOS products with version 12.4 is performed. One of the detected CVE objects of the response is shown in Figure 4.10.

Listing 4.5: Searching for a specific service and version in the CVE database.

```
./bin/search.py -p cisco:ios:12.4
```

```
CVE     : CVE-2009-0635
DATE    : 2009-03-27 12:30:02.077000
CVSS    : 7.1
Memory leak in the Cisco Tunneling Control Protocol (cTCP) encapsulation feature in Cisco IOS 12.4, when
 an Easy VPN (aka EZVPN) server is enabled, allows remote attackers to cause a denial of service (memory
 consumption and device crash) via a sequence of TCP packets.

References:
-----------
http://www.cisco.com/en/US/products/products_security_advisory09186a0080a90459.shtml
http://www.cisco.com/en/US/products/products_security_advisory09186a0080a90469.shtml
http://www.securityfocus.com/bid/34246
http://www.securitytracker.com/id?1021895
http://www.vupen.com/english/advisories/2009/0851
http://xforce.iss.net/xforce/xfdb/49417

Vulnerable Configs:
-------------------
cpe:2.3:o:cisco:ios:12.4t
cpe:2.3:o:cisco:ios:12.4xz
cpe:2.3:o:cisco:ios:12.4ya
```

Figure 4.10: The response from a CVE query showing one of the found CVE object.

4.2.1 Incoming Traffic

The incoming traffic is tested by two approaches. One consider incoming traffic from an external network and the other consider incoming traffic within the local network. This is achieved by connecting the computer with AFT to the car's gateway

IP address from either an external IP address to verify external requirements or an internal IP address to verify internal requirements. Then will AFT perform different scans to verify whether the requirements for the incoming traffic are fulfilled or not. This part is executed in two separately commands since the computer with AFT needs to have an internal IP address in one case, and an external IP address in the other case, and that needs to be switched manually between the two executions. The execution consists of three steps: stateful scan, port scan and vulnerability search. They will be further described below.

Stateful scan

The first scan to perform on incoming traffic is a stateful scan. This is performed with an ACK scan, described in Section 4.2, on all ports (65 536 ports) and if a port is classified as "filtered", this thesis considers the firewall as stateful. To make AFT more efficient, it will not continue the ACK scan after finding a filtered port since the firewall is either completely stateful or completely stateless. This scan is performed to verify the stateful requirement stated in the input model.

Port scan

To verify whether the whitelisted ports are open or not, and if there exist any other open port that not should be, a port scan is performed. All ports are considered and it is taking usage of the TCP and UDP scan, described in Section 4.2, to test if it is possible to get a connection to each port on the chosen protocols. AFT consider the whitelisted ports in the input model and note if anything is breaking it. This are two very time consuming scans, especially the UDP scan, additionally, by performing it on all ports makes it even more time consuming. However, it is still necessary to perform this scan for each port to be confident that all services that should be accepted success, but more important, that services that not should be allowed not could be found or accessed. This is a valuable scan to perform to detect eventual mistakes during the development or forgot to close ports support with debugging services e.g. SSH.

Vulnerability search

The functionality of detecting vulnerabilities is not necessary for AFT to determine whether the requirements are fulfilled or not but provides a feature for the stakeholder to also detect potential threats to the system. The vulnerability search take usage of the CVE database, by considering the result from the port scan, it can search for vulnerabilities for all open whitelisted ports. The reason why the vulnerability scan only search for vulnerabilities on open whitelisted ports is because all other ports should just be closed since the requirements states to only allow the

whitelisted ports. All vulnerabilities above the specified CVSS for each port are stated to the stakeholder to consider and prioritize.

4.2.2 Outgoing Traffic

For outgoing traffic, there are three different test scenarios. The firewall is either configured to allow outgoing traffic to whitelisted destination IP addressed, whitelisted destination ports or a combination of those. Dependent on which case the firewall consider, AFT will test all whitelisted IP addresses or whitelisted ports to verify that they can be accessed. If there are specific ports for specific IP addresses, will all those be tested together. The three different scenarios: whitelisted ports, whitelisted IP addresses and whitelisted ports and IP addresses will be further described below.

Whitelisted ports

If the requirements state that only packets addressed to specific ports should be allowed, AFT perform a TCP scan, described in Section 4.2, to determine if it is possible to initiate a connection for each port in the whole port range. AFT will then make sure that all whitelisted ports could be accessed and no non-whitelisted ports could get an established connection, otherwise AFT will notify the stakeholder.

Whitelisted IP addresses

In case that the requirements restrict the outgoing traffic to only allow access to specific IP addresses, AFT performs a ping scan. First a positive test is performed, by sending ping probes to all whitelisted addresses to verify that they are allowed to pass the firewall and could achieve an established connection. Secondly, a negative test is performed to verify that non-whitelisted IP addresses are blocked. This is performed in the same way as the positive test, but with a predefined set of non-whitelisted IP addresses to consider. This set of non-whitelisted IP addresses could be modified by the user to consider relevant non-whitelisted IP addresses that should be blocked.

Whitelisted ports & IP addresses

Probably the most common approach is to allow a combination of ports and IP addresses, certain ports should be allowed for specific IP addresses. AFT will start by testing all whitelisted ports for the different IP addresses to verify that they could be accessed by doing a TCP scan and an UDP scan. Lastly, non-whitelisted ports will be tested for each IP address to check that they not could be accessed. For this thesis, it is assumed to provide enough confidence by randomly choosing 50

ports from the non-whitelisted port space. However, this number could be modified to increase the confidence but will also affect the execution time.

4.3 Report generation

The report states whether the firewall fulfill the stated requirements. As soon as the AFT can determine whether a requirement are fulfilled or not after performing the different tests, it is appended to the report. After AFT has been executed, the report contains information about how the firewall have handled the test. If any vulnerabilities are found, all connected CVEs are stated as well. An example is illustrated in Listing 4.6, where the logic for how AFT interpret the results from the scan compared to the requirements in the input model. The variable stateful_scan is set to True if the response from any ACK scan returns filtered, then it is compared to the stateful_requirement to determine whether the requirement is fulfilled or not.

Listing 4.6: Initialization of Nmap object.

```
if (stateful_scan is stateful_requirement):
    writeReport("The stateful requirement is fulfilled\n")
else:
    writeReport("The stateful requirement is not fulfilled\n")
```

Since the report is a text file, it is easy in Python to just append a text string to an existing file. Dependent on the outcome from the different tests, different messages are appended to the report to state how the different requirements are fulfilled. Since AFT is dependent on a user to execute the different parts of the tool separately, the report is created with the date in the filename, meaning that the different tests are written to the same file if they are performed after each other in the same day. Thereby, all information regarding the requirements are stated in the same report.

4.4 An Illustration of AFT in Action

This section presents an illustration of how the MBST approach is applied with the target scanme.nmap.org. The website is provided by the Nmap organization for testing that the Nmap configuration works as intended. It should be scanned with care and only with a limited number of scans per day.

Requirements

The following made up requirements applies to the firewall of scanme.nmap.org.

Incoming traffic from external interfaces

- The firewall should perform a stateful packet inspection
- All whitelisted port should be allowed
- Default drop policy

Input Model

From the requirements stated in Section 4.4, the following input model is created which is presented in Figure 4.11.

```
Inputfile Template
-- General --
Stateful: True
Policy: Default drop
-- Incoming external --
Target IP address: scanme.nmap.org
Whitelisted ports: [53, 80]
-- Incoming internal --
Target IP address:
Whitelisted ports: []
-- Outgoing --
## type: ports or ip or combination
Type:
Target IP address:
Whitelisted ports: []
Whitelisted ip addresses: []
Combination: []
```

Figure 4.11: The input model based on the requirements.

As can be seen in the input model, the stateful option is true, the policy is default drop and the whitelisted ports are numbered under the incoming external interface. All other options are empty since the requirements do not cover incoming traffic on internal interfaces and outgoing traffic.

AFT

The AFT tool takes the input model as input and will produce a report which describes if the requirements are fulfilled or not. Since the requirements only cover incoming traffic from external interfaces, the command presented in Listing xx is executed.

Listing 4.7: Command to execute the AFT tool for incoming traffic on external interfaces.

```
sudo ./nmap.py --inExt
```

During the execution of the AFT, the stakeholder has interaction from the terminal to know the progress and what action AFT performs. An illustration of the interaction is presented in Figure 4.12.

```
Incoming traffic
------------------
Starts stateful scan
------------------
Starts port scan
------------------
Starts searching for vulnerabilities
------------------
Program done, total execution time: 0:03:57.283408
-----------------------------------------
```

Figure 4.12: Output from terminal while AFT is running.

Report

Based on the requirements stated in Section 4.4, the following report is produced from the AFT.

```
Incoming traffic on external interfaces
-------------------
The stateful requirement is fulfilled

The following non-whitelisted ports are open: [22]

The following whitelisted ports are not open: [53]

The default drop policy requirement is not fulfilled

CVE scan for open whitelisted ports:
Port: 80
Service: Apache httpd
Version: 2.4.7
Number of vulnerabilities found: 11
ID: CVE-2014-0226 and CVSS: 6.8


Execution done, total execution time: 0:03:55.070392
-----------------------------------------
```

Figure 4.13: The produced report from the MBST approach.

As seen in the report, the firewall performs a stateful packet inspection and fulfills the stateful requirement. The non-whitelisted port 22 is open which should be closed and therefore is the default drop requirement not fulfilled. Furthermore, the whitelisted port 53 is not open and the requirement of allowing all whitelisted ports is not fulfilled. 11 vulnerabilities are found for the open whitelisted port 80 with the service Apache httpd with version 2.4.7 and only one CVE is presented since

the lower limit of the CVSS score is set to 6. As described earlier, it is possible to search on a specific CVE-ID to get a more thoroughly description of the CVE.

5

Evaluation

This chapter presents the result from the MBST approach applied on a test rig at VCC. Furthermore, the results from the tests are evaluated.

5.1 Results

This section presents the results from the MBST approach. Firstly, the input model, based on the requirements of the firewall from VCC, is presented. Lastly, a snippet from the report for each test case is presented with its corresponding result.

5.1.1 Input Model

Based on the requirements stated in Section 3, the input model presented in Figure 5.1 is created.

```
Inputfile Template
-- General --
Stateful: True
Policy: Default drop
-- Incoming external --
Target IP address: XX.XX.XX.XX
Whitelisted ports: [XX, XX, XX, XX, XX, XX, XX, XX]
-- Incoming internal --
Target IP address: XX.XX.XX.XX
Whitelisted ports: [XX, XX, XX, XX, XX, XX, XX, XX]
-- Outgoing --
## type: ports or ip or combination
Type: ip
Target IP address: scanme.nmap.org
Whitelisted ports: []
Whitelisted ip addresses: [google.se, scanme.nmap.org
yahoo.com, bing.com, twitter.com]
Combination: []
```

Figure 5.1: The input model based on the firewall requirements.

As seen in Figure 5.1, the stateful packet inspection requirement is set to true and the default drop requirement is set under policy. The whitelisted ports and the target addresses for incoming traffic on both interfaces are not presented due to being sensitive data for VCC. Furthermore, for the test case regarding outgoing traffic, there was no information provided by VCC in order to confirm the requirements. Therefore, random data was used to demonstrate that the functionality of the AFT regarding outgoing traffic behaves as intended.

5.1.2 Generated Report

The report is generated after running the AFT tool for each test case; incoming traffic on external interface, incoming traffic on internal interface and outgoing traffic. Below is a snippet from the report for each test case and are presented in Figure 5.2, Figure 5.3 and Figure 5.4.

Incoming Traffic on External Interface

```
Incoming traffic on external interfaces
-------------------
The stateful requirement is fulfilled

No whitelisted ports are open
    The following ports are filtered and need further investigation: [XX, XX, XX]

No non-whitelisted ports are open

The default drop policy requirement is fulfilled

-----------------------------------------
```

Figure 5.2: The produced report for incoming traffic on external interface.

As seen in Figure 5.2, the stateful packet inspection requirement is fulfilled. No whitelisted ports are open, however, some of them are classified as filtered and need further inspection to determine if they are open or closed. No non-whitelisted ports are open and therefore the default drop requirement is fulfilled.

Incoming Traffic on Internal Interface

```
Incoming traffic on internal interfaces
-------------------
The stateful requirement is fulfilled

The following whitelisted ports are not open: [XX, XX, XX, XX, XX, XX]
    The following ports are filtered and need further investigation: [XX, XX, XX, XX]

No non-whitelisted ports are open

The default drop policy requirement is fulfilled

CVE scan for open whitelisted ports:
No cpe was found for port: XX; Version and/or Service could not be identified

-----------------------------------------
```

Figure 5.3: The produced report for incoming traffic on internal interface.

As seen in Figure 5.3, the stateful packet inspection requirement is fulfilled. The report presents the whitelisted ports that not are open. Some of the whitelisted ports are marked as filtered and need further investigation to determine if they are open or closed. No non-whitelisted ports are open and therefore the default drop requirement is fulfilled. One of the whitelisted ports were open, however, no CPE was found from the port scan which makes it impossible to search for vulnerabilities in the CVE database.

Outgoing Traffic

```
Outgoing traffic
-------------------
All IP addresses were successfully connected

-----------------------------------------
```

Figure 5.4: The produced report for outgoing traffic.

As seen in Figure 5.4, all addresses shown in the input model were successfully accessed and the requirement is fulfilled.

5.2 Analysis

This section considers the analysis of the results from AFT and the analysis of the maturity at VCC to introduce a MBST approach.

5.2.1 Results

As mentioned in Section 3, the SUT is late in its development phase and a penetration test has been performed on it i.e. it is well validated. The result is as expected, no information is revealed and the SUT works as intended. The list of whitelisted ports is the same for all incoming traffic, both on internal and external interfaces, and consists of totally 8 ports. For incoming traffic on external interface, 0 out of 8 whitelisted ports are classified as open, 3 ports as filtered and 5 as closed. The 3 filtered ports could potentially be open and needs therefore to be further investigated manually. The same result applies regarding the incoming traffic on external interface except that 1 port is classified as open, 4 ports is classified as filtered and 3 ports classified as closed. Furthermore, it was not possible to find a CPE connected to the open port for incoming traffic on external interface. Therefore, it is not possible to search in the CVE database for vulnerabilities. Regarding the outgoing traffic, the requirement holds. However, since no valid requirements were provided by VCC for the outgoing traffic data were made up and cannot be considered as relevant result more than that the functionality holds.

Based on the requirements of the SUT, our MBST approach could cover 10 out of 11 requirements. However, AFT could not verify the two requirements of outgoing traffic since we were only able to connect to the Wi-Fi hotspot which is a different interface from the one considered for the requirement regarding outgoing traffic. Secondly, since AFT is automated, it is infeasible for this thesis to create and capture traffic that are initiated from the in-vehicle. Therefore, AFT only performs negative testing on these requirements to verify if they are fulfilled or not. To determine the default drop policy requirement, negative testing are used as well, to verify that no non-whitelisted ports are accessible.

From the result we can conclude that all requirements could not be covered. This is more thoroughly described in the **research question** below.

- How much of the requirements of the firewall can be covered with a MBST approach?

  With the MBST approach, 10 out of the 11 requirements can be covered. All the requirements for incoming traffic can be covered by different port scans to determine if whitelisted ports are accepted and that the default drop policy is fulfilled. The stateful packet inspection requirement is covered by an ACK-scan to investigate if a port gets classified as "filtered". We could also successfully connect to the WiFi-hotspot to evaluate if all outgoing traffic and incoming traffic on internal interface succeeds. However, there are one requirement that not can be covered by this approach: verifying source ports for outgoing traffic. Since AFT is based on different scans, it is not possible to affect or control the source ports and can therefore not determine whether the requirement regarding source port is fulfilled.

However, only 6 out of the 10 covered requirements were fulfilled. Based on an

ACK-scan, it is determined that the firewall performs a stateful packet inspection by classifying a port as "filtered". For incoming traffic on both interfaces, the default drop policy is verified since all traffic is dropped. We also consider requirements regarding "initiated from in-vehicle" to be fulfilled by negative testing which is confirmed with the different port scans.

5.2.2 Evidence Criteria

Evidence Criteria is the optional criteria for the MBST classification which is used to validate if MBST is an applicable or useful approach at VCC, described in Section 2.3.2. It is evaluated by manual analysis of the SUT and consist of *Maturity of Evaluated System*, *Evidence Measures* and *Evidence Level* presented in Figure 2.3.

Maturity of Evaluated System

The system considered for this thesis is a *Premature System* since it is still under development. Although, it has been performed a penetration test on the system, indicating it is located in the end of the development phase and is therefore mature enough to be considered as a *Production System*.

Evidence Measures

The evidence measure for this thesis is *Effectiveness Measures*. The approach considers predefined tests that is modified based on the input model to cover common firewall requirements. The AFT then executes tests to compare the output with the expected output based on the input from the input model.

Evidence Level

The evidence level of this thesis is *Executable* which not restrict the Evidence measure. The approach considered in this thesis makes use of executable test cases to validate the generated output from AFT. The executable test cases are predefined and matched against common firewall rules, e.g. allow or block certain type of traffic.

Conclusion

The conclusion is that the system is mature enough to make it applicable to apply a MBST approach. The system is late in its development phase and a penetration test has been conducted on it. Therefore, it can be seen as a final system that is

ready for release. The evidence measures of the approach is an effectiveness measure with an executable evidence level. The MBST approach of this thesis makes use of executable test cases to evaluate the output considering the expected output and determine if a requirement is fulfilled or not.

This thesis aim to verify security requirements of a firewall with a MBST approach. However, in order to determine if a MBST approach is applicable at VCC, the maturity of handling and modelling security requirements needs to be considered as well. The SUT is mature enough to make it applicable to introduce a MBST approach, but only for functional security testing. VCC does not model security notations in their UML models which makes it impossible to make use of the only existing MBST tool, the UMLsec tool. Although, it is possible to make use of MBT tools to test functional security, but unfortunately there is a lack of experience in modelling state machines and formal models at VCC. As a result, VCC is not mature enough to make it applicable to introduce a MBST approach today. However, this is not surprising since MBST is a relative new research field and almost only used in academia.

From the evaluation of the results, the following **research questions** can be answered.

- Are there a MBST approach suitable to apply at VCC?
  - What is required to apply a MBST approach at VCC?

    To apply a MBST approach at VCC, it depends on whether functional security is enough or not for fulfilling the requirements. To use a MBST approach to test functional security, existing MBT tools or the AFT tools is applicable to use. However, in order to use existing MBT tools, a deeper knowledge in modelling of state machines and formal models is necessary. The only existing MBST tool, UMLsec tool, makes us of UMLsec models which considers security notations. The UML models at VCC does not consider security notations and needs therefore to introduce UMLsec into their UML models.

    Although, it is possible to make use of a MBST approach at VCC as demonstrated in this thesis. The drawback is that it is only available for functional security testing of firewalls. This thesis approach is very limited and only general for firewalls. By introducing UMLsec models, it would possible to get a wider scope for MBST.

6

Discussion

In this section, interesting findings and decisions that appeared during the thesis are further analyzed and discussed.

6.1 MBST vs Penetration Testing

AFT is considered to be a MBST tool that takes text-based requirements and generate tests automatically to verify each requirement for a firewall. However, this is not a replacement for Penetration Testing (PT), where the focus is to find flaws in the SUT by using a mindset of a hacker. The focus of the approach used in this thesis is to verify specified security requirements and that the SUT behave as intended. Therefore, both these approaches are necessary to test a SUT. The purpose of MBST approach is to support the stakeholder during the development and to simplify testing the implementation for mistakes meanwhile PT is usually performed late in the development phase when the SUT is almost complete. The aim of AFT is to reduce the faults found during the PT by making it feasible for testing more frequent and have more interaction between test results and implementation.

6.2 MBT tools for a MBST approach

As presented in Section 3.1, a MBT tool could be used for applying a MBST approach considering functional security i.e. verify security mechanisms. However, as mentioned about the characteristics of MBST, there are two other perspective of *Model of System Security* except Functionality of security mechanisms to consider: security properties and vulnerabilities. These two could not be considered with any of the MBT tools investigated in this thesis. By not being able to consider security attributes or vulnerabilities in the modeling, the process of developing security models becomes otiose. However, as mentioned in Section 3.1, dependent on the models of the SUT and the purpose of testing it, a MBT tool could be relevant e.g. testing the functionality of the security mechanisms for a SUT modelled in a state models or formal models. Considering that MBST is a relatively new research field with only one MBST tool available, it is reasonable to expect that more tools will

be available in the future as security becomes more prioritized.

6.3 AFT

AFT is a useful tool during the development of a firewall. It is a fully automated tool that are able to detect common mistakes during the configuration and presents potential vulnerabilities to stakeholders. This thesis considers an in-vehicle firewall, however, AFT can be used on an arbitrary firewall to evaluate if the requirements holds. In Section 4.4, the approach with AFT is used on a different firewall as a proof-of-concept on a web server. As seen in Section 5.2.1, a firewall that has reached a late state in the development and already has been penetration tested with the flaws fixed do not reveal that much information for AFT to present. For example, most of the whitelisted ports cannot be determined whether they are open or not and then needs to be manually verified by the stakeholder.

Balance between factors

This implementation is very time consuming to execute since it requires several network scans to be performed on all ports available. Therefore, it is a balance between execution time and confidence of the result to set a realistic part of all ports to be tested to decrease the time but still achieve enough confidence to trust the outcome of AFT. This will be set differently dependent on the SUT and the importance of a fully secure system. One approach to avoid testing all ports is to choose a fixed amount of ports to test and then randomly select them, either from the whole port range or from different sections of it to achieve a desired distribution of ports in the port scans.

It is also important to highlight the potential problem with UDP scans. As mentioned in Section 4.2, UDP scans can be very time consuming with only one scan potentially taking more than 18 hours. Therefore, it is important to determine if the time of performing a UDP scan on all 65536 ports is necessary in order to have enough of credibility that a requirement is fulfilled.

Limitation of AFT

A problematic requirement to fully verify is when a connection only should be allowed to be established from the source, in this case in the vehicle. Since AFT is the client that connect to the SUT, it is infeasible for this thesis to test positive testing because all attempts to connect will be refused and dropped by the firewall. However, the negative testing is still able to determine whether not allowed traffic will be denied and is decided to be enough for this thesis since from a security perspective, it is more valuable that a port scan detect that a non-whitelisted port is

open than detecting that a whitelisted port is closed. Since the whitelisted ports are supposed to be open, they are probably more protected than services behind ports that not should be open e.g. debug services used during the development. Furthermore, requirements restricting source ports for outgoing traffic is also infeasible for AFT since the scan tool only specify the destination ports and not affect the source port.

6.4 Requirements

It is a new era within the automotive industry with new threats that needs to be carefully measured, as the new vehicles gets connected. By dividing the requirements, it could be seen as an additional layer of protection. The first potential threat would be if an adversary could get access of the system from incoming traffic on an external interface. By having requirements for incoming traffic on internal interfaces and for outgoing traffic, an infected computer would then not be able to be used for further intrusions or malicious intentions, e.g. DDoS-attacks. It could be seen as adding additional layers of security.

It is a very strict requirement to restrict the outgoing traffic to specified whitelisted IP addresses and/or ports, but it is a huge advantage in a security perspective. In case of a successful attack, where the attacker get access to one device in the system, it makes it much harder to communicate outside the system if the outgoing traffic is restricted. Even worse, a scenario considering the fact that it is a vehicle, would be if the attacker could establish a connection to communicate with the internal network with ECUs and take control of critical functions e.g. braking and steering.“

6.5 CVE database

For this thesis, all available CVEs from 2002 were selected to be included in the database. However, it is possible to modify how old CVEs to consider when populating the CVE database. Since CVE from 2002 most likely is outdated, it would be more efficient to populate the database with only more recent CVE's. The framework used stores the CVEs in a NoSQL database, which is not as efficient as a SQL database considering the performance and flexibility. Since the lack of functionality to query objects with several criteria for the noSQL database, like CVE-ID, relevant years and CVSS, a SQL database would yield better performance and more flexibility.

Since it is not possible in the framework to search for a CVE by a specific year or range of years, it would be more efficient to make use of a relation database. It has big impact on the performance to make use of the existing database due to that each CVE is connected to the product which makes it hard to determine if is relevant for the specific version of the product.

7

Conclusion

This thesis survey the state-of-the-art in the MBST field and propose a new approach to apply MBST to verify security requirements for a firewall within a vehicle. After a thorough literature study, we could find one available MBST tool. The tool makes use of UMLsec models which is UML models including security notations. VCC today have UML models on a more abstract layer which do not consider security notations. Therefore, it is concluded that VCC is not mature in their modelling to make use of the MBST tool. As a result, an evaluation of MBT tools were conducted to decide if these could be used in a MBST approach. However, since the input models of the MBT do not consider security notations, only functional security can be verified. From the evaluation, we concluded that VCC is not mature enough to make us of the existing MBT tools due to lack of experience of modelling state machines and formal models. As a result, AFT was implemented to test functional security of a firewall and covers 10 out of 11 requirements. The final conclusion is that it is possible to make use of a model-based security testing approach with the new AFT tool, which automatically verifies whether firewall requirements are fulfilled or not.

Future Work

The future work of this thesis is to further develop the AFT tool. By extending the tool with more functionality, more requirements would be possible to be considered and also achieve more confidence regarding the result. Furthermore, an investigation to validate the process of adapting the text-based requirements into the input model would be beneficial since this thesis did not verified it. Additionally, evaluating different approaches that enables adaption of requirements into an input model in regards to usability. Since this thesis only consider functional security of firewall requirements, a further investigation of possibilities to use security notations in models used as input would be of interest to make use of the full potential of MBST.

Bibliography

[1] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22(5):297–312, 2012.

[2] Michael Felderer, Philipp Zech, Ruth Breu, Matthias Büchler, and Alexander Pretschner. Model-based security testing: a taxonomy and systematic classification. *Software Testing, Verification and Reliability*, 26(2):119–148, 2016.

[3] Michael Felderer, Berthold Agreiter, Philipp Zech, and Ruth Breu. A classification for model-based security testing. *Advances in System Testing and Validation Lifecycle (VALID 2011)*, pages 109–114, 2011.

[4] Hossain Shahriar and Mohammad Zulkernine. Automatic testing of program security vulnerabilities. In *Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International*, volume 2, pages 550–555. IEEE, 2009.

[5] M Melosi. The automobile and the environment in american history. *Online: http://www. autolife. umd. umich. edu/Environment/E_Overview/E_Overview. htm*, 2004.

[6] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447–462. IEEE, 2010.

[7] Nicolas Navet and Françoise Simonot-Lion. In-vehicle communication networks-a historical perspective and review. Technical report, University of Luxembourg, 2013.

[8] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015, 2015.

[9] Brad Arkin, Scott Stender, and Gary McGraw. Software penetration testing. *IEEE Security & Privacy*, 3(1):84–87, 2005.

[10] Ina Schieferdecker, Juergen Grossmann, and Martin Schneider. Model-based

security testing. *arXiv preprint arXiv:1202.6118*, 2012.

[11] Bruce Potter and Gary McGraw. Software security testing. *IEEE Security & Privacy*, 2(5):81–85, 2004.

[12] Gu Tian-yang, Shi Yin-Sheng, and Fang You-yuan. Research on software security testing. *World Academy of science, engineering and Technology*, 70:647–651, 2010.

[13] Michael Felderer and Ina Schieferdecker. A taxonomy of risk-based testing, 2014.

[14] Roberto Natella, Domenico Cotroneo, Joao A Duraes, and Henrique S Madeira. On fault representativeness of software fault injection. *IEEE Transactions on Software Engineering*, 39(1):80–96, 2013.

[15] Guido Wimmel and Jan Jürjens. Specification-based test generation for security-critical systems using mutations. In *International Conference on Formal Engineering Methods*, pages 471–482. Springer, 2002.

[16] George Fink and Matt Bishop. Property-based testing: a new approach to testing for assurance. *ACM SIGSOFT Software Engineering Notes*, 22(4):74–80, 1997.

[17] Sofia Bekrar, Chaouki Bekrar, Roland Groz, and Laurent Mounier. Finding software vulnerabilities by smart fuzzing. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, pages 427–430. IEEE, 2011.

[18] Zakir Durumeric, James Kasten, David Adrian, J Alex Halderman, Michael Bailey, Frank Li, Nicolas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, et al. The matter of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 475–488. ACM, 2014.

[19] Gerry Zaugg. Firewall testing. *ETH Zurich*, 2005.

[20] Achim D Brucker, Lukas Brügger, and Burkhart Wolff. Model-based firewall conformance testing. In *Testing of Software and Communicating Systems*, pages 103–118. Springer, 2008.

[21] Achim D Brucker, Lukas Brügger, and Burkhart Wolff. Formal firewall conformance testing: an application of test and proof techniques. *Software Testing, Verification and Reliability*, 25(1):34–71, 2015.

[22] Diana Senn, David Basin, and Germano Caronni. Firewall conformance testing. In *IFIP International Conference on Testing of Communicating Systems*, pages 226–241. Springer, 2005.

[23] Eckard Bringmann and Andreas Krämer. Model-based testing of automotive systems. In *Software Testing, Verification, and Validation, 2008 1st Interna-*

tional Conference on, pages 485–493. IEEE, 2008.

[24] Michael Felderer and Elizabeta Fourneret. A systematic classification of security regression testing approaches. *International Journal on Software Tools for Technology Transfer*, 17(3):305–319, 2015.

[25] Jan Jürjens. Model-based security testing using umlsec: A case study. *Electronic Notes in Theoretical Computer Science*, 220(1):93–104, 2008.

[26] Jan Jürjens and Guido Wimmel. Specification-based testing of firewalls. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pages 308–316. Springer, 2001.

[27] Adel El-Atawy, Khaled Ibrahim, Hazem Hamed, and Ehab Al-Shaer. Policy segmentation for intelligent firewall testing. In *Secure Network Protocols, 2005.(NPSec). 1st IEEE ICNP Workshop on*, pages 67–72. IEEE, 2005.

[28] Adel El-Atawy, Taghrid Samak, Zein Wali, Ehab Al-Shaer, Frank Lin, Christopher Pham, and Sheng Li. An automated framework for validating firewall policy enforcement. In *Policies for Distributed Systems and Networks, 2007. POLICY'07. Eighth IEEE International Workshop on*, pages 151–160. IEEE, 2007.

[29] Volvo on call. `http://www.volvocars.com/se/kop/uppkopplad/volvo-on-call`, 2017. [Online; accessed 2-May-2017].

[30] umlsec tool. `https://rgse.uni-koblenz.de/jj/umlsectool/index.html`, 2017. [Online; accessed 2-May-2017].

[31] Specexplorer. `https://msdn.microsoft.com/en-us/library/ee620512.aspx`. [Online; accessed 9-May-2017].

[32] Graphwalker. `https://rgse.uni-koblenz.de/jj/umlsectool/index.html`. [Online; accessed 9-May-2017].

[33] Pymodel. `http://staff.washington.edu/jon/pymodel/www/`. [Online; accessed 9-May-2017].

[34] T-vec tester. `https://www.t-vec.com/solutions/simulink.php`. [Online; accessed 9-May-2017].

[35] 4test. `http://www.testautomationday.com/4test-model-based-testing-agile/`. [Online; accessed 9-May-2017].

[36] Matt Wynne and Aslak Hellesoy. *The cucumber book: behaviour-driven development for testers and developers*. Pragmatic Bookshelf, 2012.

[37] Achim Brucker and Burkhart Wolff. hol-testgen. *Fundamental Approaches to Software Engineering*, pages 417–420, 2009.

[38] python-nmap : Python framwork for nmap. `http://xael.org/pages/python-nmap-en.html`, 2017. [Online; accessed 2-May-2017].

[39] cve-search - a tool to perform local searches for known vulnerabilities. `https://github.com/cve-search/cve-search`, 2017. [Online; accessed 2-May-2017].

[40] Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.

[41] Common vulnerabilities and exposures (cve) - the standard for information security vulnerability names. `https://cve.mitre.org/index.html`, 2017. [Online; accessed 2-May-2017].