Mälardalen University Press Licentiate Theses
No. 355

MODEL-DRIVEN SECURITY TEST CASE GENERATION USING THREAT MODELING AND AUTOMATA LEARNING

Stefan Marksteiner

2024

School of Innovation, Design and Engineering

Copyright © Stefan Marksteiner, 2024
ISBN 978-91-7485-638-5
ISSN 1651-9256
Printed by E-Print AB, Stockholm, Sweden

Abstract

Automotive systems are not only becoming more open through developments like advanced driving assistance functions, autonomous driving, vehicle-to-everything communication and software-defined vehicle functionality, but also more complex. At the same time, technology from standard IT systems become frequently adopted in these systems. This development has two negative effects on correctness and security: the rising complexity adds potential flaws and vulnerabilities while the increased openness expands attack surfaces and entry points for adversaries.

To provide more secure systems, the amount of verification through testing has to be significantly increased, which is also a requirement by international regulation and standards. Due to long supply chains and non-disclosure policies, verification often have to use black-box methods. This thesis strives therefore towards finding more efficient methods of automating test case generation in both white- and black-box scenarios. The thesis focuses on communication protocols used in vehicular systems and we base our research on model-based methods. Our contributions include:

- We provide a practical method to automatically obtain behavioral models in the form of state machines of communication protocol implementations in real-world settings using automata learning.
- We also provide a means to automatically check these implementation models for their compliance with a specification (e.g., from a standard).
- We furthermore present a technique to automatically derive test-cases to demonstrate found deviations on the actual system.
- We also present a method to create abstract cybersecurity test case specifications from semi-formal threat models using attack trees.

Sammanfattning

Fordonssystem blir idag mer och mer öppna och komplexa genom introduktion av t.ex. avancerade körhjälpsfunktioner, autonom körning, kommunikation mellan fordon och allt mer mjukvarudefinierad fordonsfunktionalitet. Samtidigt används teknik från traditionella IT-system allt oftare i dessa system. Denna utveckling har två negativa effekter på korrekthet och säkerhet: den ökande komplexiteten lägger till potentiella brister och sårbarheter medan den ökade öppenheten utökar attackytor och ingångspunkter för digitala inkräktare.

För att tillhandahålla säkrare system måste mängden verifiering av systemsäkerhet genom testning ökas avsevärt, vilket också är ett krav enligt internationella regler och standarder. På grund av långa värdekedjor och sekretessprinciper måste verifieringsmetoder ofta använda s.k. black-box verifiering där man verifierar funktionen av en komponent utan att kunna studera hur den fungerar internt. Det är dock fördelaktigt om man kan använda s.k. while-box metoder där man kan verifiera en komponent medan man samtidigt studerar hur den fungerar internt.

Denna avhandling strävar därför mot att hitta mer effektiva metoder för att automatisera testfallsgenerering i både white- och black box-scenarier och vårt fokus ligger på kommunikationsprotokoll som används i fordonssystem. Den huvudsakliga ansatsen är att använda modellbaserade metoder. Vi presenterar en praktisk metod för att automatiskt erhålla beteendemodeller i form av tillståndsmaskiner för implementeringar av kommunikationsprotokoll med hjälp av automatinlärning. Vi presenterar också ett sätt att automatiskt kontrollera dessa implementeringsmodeller för att se om de överensstämmer med en specifikation (t.ex. från en standard). Vi presenterar vidare en teknik för att automatiskt härleda testfall för att demonstrera upptäckta avvikelser på det faktiska systemet. Vi presenterar också en metod för att skapa abstrakta testfallsspecifikationer för cybersäkerhet från semiformella hotmodeller med hjälp av attackträd.

To my family

Acknowledgments

I like to use this space to thank this thesis' advisors Marjan Sirjani and Mikael Sjödin for spending their precious time and giving me guidance in research directions and writing, which brought me forward in both professional and scientific manners. Furthermore, I like to thank my employer at the AVL List Gmbh, foremost my former and current supervisors Horst Pflügl and Kieran McAleer for giving me the freedom to do the necessary research, without losing the vision for the application and the practical use of it. I also want to thank my son, Benjamin Marksteiner, just for being there and brining joy (and also action) to my life. Most of all, I like to thank my wife Astrid Marksteiner for going this journey with me, even if it is not easy alongside manyfold job duties and a small kid. No acknowledgement could make it up with her support.

Stefan Marksteiner, February 2024, Graz, Austria

This research received funding from the program "ICT of the Future" of the Austrian Research Promotion Agency (FFG) and the Austrian Ministry for Transport, Innovation and Technology under grant agreements No. 867558 (project TRUSTED) and No. 880852 (project LEARNTWINS) and within the ECSEL Joint Undertaking (JU) under grant agreements No. 876038 (project InSecTT) and No. 101007350 (project AIDOaRt). The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Sweden, Spain, Italy, France, Portugal, Ireland, Finland, Slovenia, Poland, Netherlands, Turkey. The document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

List of Publications

Papers Included in the Licentiate Thesis[1]

Paper I: *A Systematic Approach to Automotive Security*, Masoud Ebrahimi, Stefan Marksteiner, Dejan Ničković, Roderick Bloem, David Schögler, Philipp Eisner, Samuel Sprung, Thomas Schober, Sebastian Chlup, Christoph Schmittner, and Sandra König. In Lecture Notes in Computer Science, vol. 14000. Cham: Springer International Publishing, 2023. [1]

Paper II: *Approaches For Automating Cybersecurity Testing Of Connected Vehicles*, Stefan Marksteiner, Peter Priller and Markus Wolf. In Intelligent Secure Trustable Things, M. Karner, J. Peltola, M. Jerne, L. Kulas, and P. Priller, Eds., in Studies in Computational Intelligence. Springer Nature, 2023. [2]

Paper III: *Using Automata Learning for Compliance Evaluation of Communication Protocols on an NFC Handshake Example*, Stefan Marksteiner, Marjan Sirjani and Mikael Sjödin. In Engineering of Computer-Based Systems Conference 2023, J. Kofroň, T. Margaria, and C. Seceleanu, Eds., in Lecture Notes in Computer Science, vol. 14390. Cham: Springer Nature Switzerland, 2023. [3]

Paper IV: *From TARA to Test: Automated Automotive Cybersecurity Test Generation Out of Threat Modeling*, Stefan Marksteiner, Christoph Schmittner, Korbinian Christl, Dejan Ničković, Mikael Sjödin, and Marjan Sirjani. In Proceedings of the 7th ACM Computer Science in Cars Symposium, in CSCS '23. New York, NY, USA: Association for Computing Machinery, 2023. [4]

[1]The included papers have been reformatted to comply with the thesis layout.

Additional Peer-reviewed Publications Related to the Thesis[2]

Paper 1: *Wireless Security in Vehicular Ad Hoc Networks: A Survey*, Thomas Blazek, Fjolla Ademaj, Stefan Marksteiner, Peter Priller, Hans-Peter Bernhard. In SAE International Journal of Connected and Autonomous Vehicles, vol. 6, no. 2, 2022. [5]

Paper 2: *A Global Survey of Standardisation and Industry Practices of Automotive Cybersecurity Validation & Verification Testing Processes and Tools*, Andrew Roberts, Stefan Marksteiner, Mujdat Soyturk, Berkay Yaman, Yi Yang. In SAE International Journal of Connected and Autonomous Vehicles, vol. 7, no. 2, 2023. [6]

[2]These papers are not included in this thesis.

Contents

I Thesis **1**

1 Introduction **3**

1.1 Practical Motivation: Industrial Problems Leading to Scientific Problems . . . 5

1.2 Thesis Outline . . . 7

2 Background and Preliminaries **9**

2.1 Threat Modeling . . . 9

2.2 Attack Trees . . . 10

2.3 Labelled Transition Systems . . . 10

2.4 Formalized Attack Languages . . . 10

2.5 Mealy Machines . . . 11

2.6 Automata Learning . . . 11

2.7 Behavioral Equivalences . . . 12

3 Research Overview **13**

3.1 Research Goals . . . 14

3.1.1 Threat Model-based Test Generation . . . 16

3.1.2 Automated State Machine Derivation . . . 16

3.1.3 Compliance Checking . . . 17

3.2 Research Method . . . 18

4 Research Contributions **21**

4.1 Threat Model-based Test Generation . . . 23

4.2 Automated State Machine Derivation . . . 23

4.3 Compliance Checking . . . 24

4.4 Publications . . . 25

4.4.1 Paper I 25
4.4.2 Paper II 26
4.4.3 Paper III 26
4.4.4 Paper IV 27
4.4.5 Related Papers Not Included in the Thesis 28

5 Related Work 31
5.1 Model-based Test Case Generation 31
5.2 Attack Trees-based Security Testing 32
5.3 Formalized Test Descriptions 32
5.4 Automated State Machine Derivation and Protocol Learning . 33
5.5 Conformance Checking using Equivalence of State Machines . 34

6 Conclusion and Future Work 35
6.1 Conclusions 35
6.2 Future Directions 36

Bibliography 39

II Included Papers 47

7 Paper I:
A Systematic Approach to Automotive Security 49
7.1 Introduction 51
7.2 TRUSTED Methodology 52
7.3 Automotive Security by Design 54
7.3.1 System Architecture Model 55
7.3.2 Threat Analysis 55
7.3.3 V&V Planning 57
7.4 Automotive Security Testing 58
7.4.1 Automata Learning for Correctness 58
7.4.2 Use-Case Scenarios 59
7.5 Conclusion 62
Bibliography 63

8 Paper II:
Approaches For Automating Cybersecurity Testing Of Connected Vehicles 67
8.1 Introduction 69

8.2 State of the Art and Related Work 70
8.3 Automotive Cybersecurtiy Lifecycle Management 72
8.3.1 Threat Modeling 73
8.4 Cybersecurity Testing 74
8.4.1 Learning-based Testing 76
8.4.2 Model-based Test Case Generation 78
8.4.3 Testing Platform 79
8.4.4 Automated Test Execution 80
8.4.5 Fuzzing 82
8.5 Conclusion 83
Bibliography 85

9 Paper III:
Using Automata Learning for Compliance Evaluation of Communication Protocols on an NFC Handshake Example 91
9.1 Introduction 93
9.1.1 Motivation 93
9.1.2 Contribution 94
9.2 Preliminaries 94
9.2.1 State Machines 95
9.2.2 Transitions and Equivalence 95
9.2.3 Automata Learning 98
9.2.4 LearnLib 99
9.2.5 Near Field Communication 100
9.2.6 The NFC Handshake Automaton 100
9.3 NFC Interface 103
9.3.1 Learner Interface Device 103
9.3.2 Adapter Class 103
9.4 Learning Setup 104
9.4.1 Comparing Learning Algorithms and Calibrations . . . 105
9.4.2 Abstraction 107
9.4.3 Labeling and Simplification 109
9.4.4 Compliance Evaluation 109
9.5 Evaluation 111
9.5.1 Test Cards and Credit Cards 111
9.5.2 Passports 111
9.5.3 Tesla Key Fob 111
9.6 Related Work 113
9.7 Conclusion 113

9.7.1 Discussion 113
9.7.2 Outlook 114
Bibliography 115

10 Paper IV:
From TARA to Test: Automated Automotive Cybersecurity Test Generation Out of Threat Modeling 119
10.1 Introduction 121
10.1.1 Motivation 122
10.1.2 Contribution 122
10.2 Automotive Security Communication 123
10.2.1 Cybersecurity Assurance Level (CAL) 123
10.2.2 Target Attack Feasibility (TAF) 124
10.2.3 Integrating CAL and TAF in security testing 124
10.3 Threat Modeling 128
10.3.1 Threat-Interdependencies and Attack Trees 130
10.4 Automated Testing 131
10.4.1 Security Tests and their relationship with the Security Analysis 132
10.4.2 Security Test Generation 132
10.5 Case Study 135
10.6 Related Work 136
10.7 Conclusion 138
Bibliography 139

I

Thesis

Chapter 1

Introduction

This thesis concentrates on using formal methods for assuring correctness and security of systems with a focus on the automotive domain. To utilize these methodologies in practice, some practical problems have to be solved, that in turn lead to solving some scientific problems that lay the fundament for the provided solutions. A special emphasis is given to verifying the correctness of communication protocol implementations in system components and to verification from an architectural threat modeling perspective. The upcoming UN-ECE regulation R.155 [7] mandates not only the introduction of a cybersecurity management system (CSMS) and according security measures for automotive systems, but also evidence of their appropriateness and effectiveness, which is to be demonstrated by testing. This requires an amount of testing of the cybersecurity of vehicle systems that is not to be covered with manual testing alone. Therefore, automated methodologies for automotive cybersecurity testing are needed. Furthermore, the testing should work in both white box and black box settings. Methods for the former do not work on the latter (for the lack of information), however black box methods are not efficient in white box settings (for not using present information). The former to cover the security of all aspects of a system in the most thorough way. The latter to on the one hand provide an attacker's view but also because of the lack of access to source code or other system component internals due to long supply chains, as well as the unwillingness of manufacturers to disclose system details.

Formal methods, particularly formal modeling has been used in engineering complex systems, for example in the automotive and aerospace domains. Despite the effort they normally come with, their founding on strong mathematical principles has three main advantages: a very structured approach that provides a good level of comprehensiveness, well-reasoned verdicts, and a high degree of automation capability (once a model has been derived) due to computer systems basing on the same logic. There are different approaches to formal modeling practically in use in the industry: models based on state machines have a long tradition of analyzing systems' correctness , on the other hand architectural threat models [8] in the form of graphs with clear semantics and first order logic-based threat rule sets are used in the software industry for quite some time and have become very widely adopted as part of a Threat Analysis and Risk Assessment (TARA) in the automotive industry [9]. Threat models analyze a system design based on data flows between its architecture's components [8].

Regardless of these beneficial traits, these methods are not only ordinarily very labor-intense, but also hard to apply on black-box components, as without access to internal information it is not trivial to obtain a state machine model (also called automaton) in an automated way. Even if such an automaton is present, reliable methods and rules to check it for correctness in terms of security have to be in place – none such is so far formulated in general, therefore sets of these rules and methods have to be established for each use case individually. Lastly, it is an open question how white box and black box modeling methods could support each other. An architectural threat model from a TARA is usually (manually) generated at design time and is based on assumptions that are often tacitly made when using modeling components. We therefore strive to investigate, if it is possible to check whether made assumptions actually hold in the implementation. Concentrating on external interfaces of system components, automatically derived state machine models of communication protocol are investigated. One further question is also which algorithm and parameter configuration is most suitable for inferring models of implementations of a specific protocol.

This thesis therefore provides an approach to generate security test cases from an architectural threat model in white box scenarios, concentrating on the question how to provide a formal translation from threat modeling results to actions in test cases utilizing attack trees [10, 11] and labelled transition systems (LTS) [12]. For black box scenarios, this thesis investigates inferring a behavioral model in form of a state machine. To derive such a model, the L algorithm by Angluin [13] and variations thereof [14], as well as more modern algorithms (e.g. [15]) were examined. Automatic model derivation is beneficial because not only manual modeling in general is very resource-intense, but also hard to perform at all in black box settings. On an example case of the

Near-field Communication's (NFC) handshake protocol (ISO 144443-3 [16]), for which a learning interface working in real-world environments is provided, the different algorithm and parameter sets were investigated, giving details (efficacy and performance) for automata learning in special cases. The thesis also works out the necessary level of abstraction in order to investigate the possibilities of implementing this concept in a practical (i.e., real-world) setup. Using these learned models, a behavior-based black box compliance checking method using bisimulation or trace equivalence was provided. The comparison object for these equivalences is an automaton modeled after the respective specification or standard, the learned implementation should comply to. The reason for choosing bisimulation at this point, despite trace equivalence being the actual property for black box compliance, is its superior efficiency over the latter. The threat model-based test case generation is a novel approach, while the method of combining automata learning and bisimulation checking is rarely used in general, and not at all for behavior-based black box implementation compliance checking. Nevertheless, it provides a more rigorous way of compliance checking due to its more exhaustive way of model building. So far, little comprehensive work for automata learning in different settings has been performed, for which this thesis also provides a contribution. Lastly, the thesis sketches some notions of using feedback from the behavioral state machine checking to the architectural threat models. The latter are ordinarily generated manually during design phase and based on assumptions about the modeled components (e.g., a modeled component complying to a certain standard or featuring a distinct property). These assumptions can be checked in later phases by checking the state machines of the implemented systems if these assumptions hold (e.g., if the implemented component actually complies to the standard). This allows for statements about the accuracy of the threat model from checking the behavioral state machine models.

1.1 Practical Motivation: Industrial Problems Leading to Scientific Problems

This section contains an outline of what practical problems motivate the research goals (Section 3.1) and solutions to scientific problems (Section 4) in this thesis. It also serves the purpose of giving a context how the problem fields for the research goals are related, targeting to give a better understanding of the overall picture of the research. The practical overall motivation of this thesis is facilitating automated cybersecurity testing of vehicular systems.

Analyzing contemporary state-of-the-art automotive cybersecurity engineering processes (most prominently ISO/SAE 21434:2021 [17]), several gaps that hinder the efficiency of testing were identified. A standard cybersecurity engineering process is aligned with the general automotive engineering processes. One proliferated example is Automotive Spice, which is roughly defined along four major activities [18]:

- Threat Analysis and Risk Assessment (TARA – accompanying the system design as part of a cybersecurity security requirements elicitation process) to analyze potential weaknesses and threats in the design and assess their severity during the design phase.
- Implement the system using security goals and claims drawn from the TARA mitigating the found risks and therefore implementing a secure design (in cybersecurity implementation process).
- Validate and verify the system security measurements' effectiveness, providing evidence and arguments for the implemented system's cybersecurity (as risk treatment verification and validation processes).
- Repeat the actions above during the rest of the system's life cycle after the start of production (SOP) with both updates of the system (functional and non-functional) and of the threat landscape (e.g., discovery of new vulnerabilities).

To provide both more efficient and more rigorous testing of system correctness from a cybersecurity perspective, structuring and automation is desirable to be applied. One of potential fields identified is to combine threat modeling done in the TARA with testing by automatically deriving test cases from the former. This enhances efficiency by doing two necessary things at once (TARA and test case definition) and enhances testing rigor by directing the testing to the very security measures derived from the security goals drawn from the TARA. To gain this ability, formalized test cases descriptions (done in previous work) has to be connected to the results of a TARA, constituting a research problem resulting in research goal 1 (RG1 – Section 3.1.1). Although the TARA is conducted during design time (without an implementation available), it is still possible to create the necessary test case descriptions, as those can be made technology-agnostic and formed into practically executable test cases once an implementation is available (see Section 5.3). Also, it is quite usual in the automotive industry that an Original Equipment Manufacturer (OEM – in the automotive context mostly a car maker) integrates components from suppliers without getting access to its internals (i.e., not getting source code, internal specifications, etc.), engineered after the OEM's specifications. This creates the needs to test the correctness and security of these systems in a black box setting. One scientific problem this requirement opens up is a means to automatically obtain a formalized description of the behavior of a (sub-)system's implementation to have an object to automatically analyze, which eventually lead to research goal 2 (RG2 – Section 3.1.2) how to create a state machine model from a black box system. The other side of the medal is to have a means to actually check that model for its correctness and security. This relays to the research problem of how to check the behavior of a model against a given specification, yielding research goal 3 (RG3 – Section 3.1.3). Both the solution for RG1 and for RG3 have been created after industrial needs which is underpinned by patents that have been filed (Austrian patent applications No. A50667/2023 and A50660/2023, respectively). Future work after the licentiate completion includes methods to use model checking for dedicated security properties, use learnt models for fuzz testing and derive features to check from threat modeling in order to check modeled assumptions more rigorously. As this kind of test automation usually iterate over the complete life cycle the correctness of the implementation also feeds back to the threat modeling, by check if the assumptions made in the design phase hold in the implementation, which influences future iterations in the life cycle. Figure 1.1 gives an overview of the automotive cybersecurity process group [18] and the practical implications of its automation leading to the research goals of this thesis; the boxes represent the processes of the cybersecurity engineering process group, while the arrows represent practical improvements as presented in this thesis (see above), leading to the research goals.

1.2 Thesis Outline

This thesis consists of two parts: Part I provides a coat for the research, namely the necessary preliminaries, the aim of the research, its contribution and comparison with existing work. Part II consists of the research papers constituting this thesis. the remainder of Part I is organized as follows: Chapter 2 contains the background, Chapter 3 gives and overview of the research goals and methods, Chapter 4 outlines the research contributions including a short description of the included publications, Chapter 5 outlines related work and Chapter 6 concludes the thesis and gives an outlook to future work.

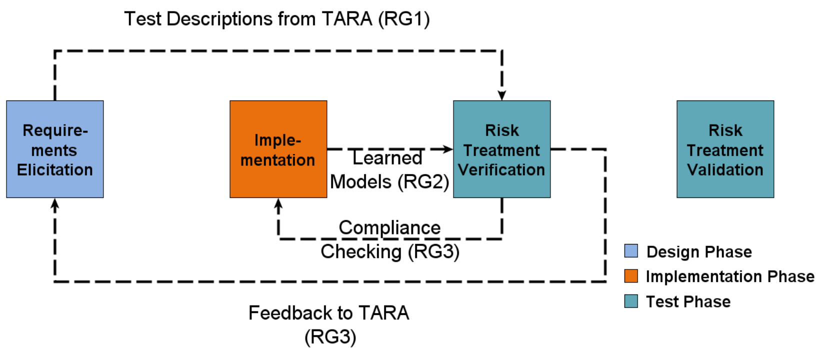

Test Descriptions from TARA (RG1)
Require-ments Elicitation
Imple-mentation
Learned Models (RG2)
Risk Treatment Verification
Risk Treatment Validation
Compliance Checking (RG3)
Feedback to TARA (RG3)
Design Phase
Implementation Phase
Test Phase

Figure 1.1: Research contributions in relation to the Security Engineering Process after Automotive Spice [18]. The boxes represent the processes of the cybersecurity engineering process group, while the arrows represent practical improvements as presented in this thesis (see above), leading to the research questions in Section 3.1

Chapter 2

Background and Preliminaries

This section very briefly explains some basic concepts that are used in this thesis. Other related work and alternative approaches towards reaching the research goal are outlined in Section 5. The usage of this background research this thesis builds on is as follows: Threat Modeling, Attack Trees, Labelled Transition Systems (LTS'), and Formalized Attack Languages in RG1 (Section 3.1.1), Mealy Machines and Automata Learning in RG2 (Section 3.1.2), Mealy Machines and Behavioral Equivalences in RG3 (Section 3.1.3).

2.1 Threat Modeling

Threat modeling is a systematic, semi-formal approach to scrutinize systems for potential threats and pitfalls. Threat modeling ordinarily requires two components [8]. One is a structured representation of the considered system (i.e., a system model), containing all information necessary to determine potential threats, as well as assessing their impact and likelihood of occurrence. A commonly used form of representation in sophisticated tools for threat modeling are data flow diagrams. The second component is the actual threat model. This model consists of a set of rules that determine which potential threat would occur if two components in the system model are connected in a certain way. These rule sets can, dependent on the application domain, become very complex, with the goal being to scrutinize the considered system very comprehensively and structured. Sophisticated threat models contain a considerable amount of domain knowledge and are usually created by groups of security experts in the respective domain. This thesis uses threat models as a basis to create test cases in a structured way (see Section 4.1).

2.2 Attack Trees

Attack trees display relations, interdependencies and hierarchies of threats and vulnerabilities [10, 11]. The advantage of this form of representation is the ability to display different paths towards a certain objective i.e., to show different opportunities to concatenate attacks in order to exploit a certain vulnerability from a distinct starting point (mostly an interface accessible from the outside). This way, attack trees are a capable tool for assessing how combined attacks that exploit a complete set of threats impact the overall attack surface and success likelihood [19]. In this thesis, attack trees stemming from threat models are the origin for a method to automatically derive technology-agnostic security test scenarios to provide evidence for the correct functioning of implemented security measures (see Section 4.1).

2.3 Labelled Transition Systems

A principal notation for formal representations of systems used in this thesis are *Transition Systems (TS)* and *Labelled Transition System (LTS)*. A TS is defined as a set of states (Q) and a transition relation ($\rightarrow \in Q \times Q$, with $q, q\prime \in Q; q \rightarrow q\prime$). A *Labelled Transition System (LTS)* additionally possesses a set of labels (Σ), such that each transition is named with a label σ in Σ ($q, q\prime \in Q, \sigma \in \Sigma; q \xrightarrow{\sigma} q\prime$) [12]. LTS can describe the behavior of systems and mechanisms at different levels. This thesis uses LTS for a translation mechanism from attack trees to attack descriptions in a specifically designed attack description language (see Section 4.1).

2.4 Formalized Attack Languages

Domain-specific languages (DSLs) are computer programming language of limited expressiveness focused on a particular domain [20]. That means that they should be just expressive enough to model any necessary features and conditions of the respective domain and lean enough for domain experts to be easily read and communicate about. Besides they ordinarily have formal syntax and semantics in the sense that a state machine (particularly a deterministic finite acceptor – DFA) built for one particular language should be able to decide if a word or a statement is a well-formed statement in that language. A DSL (see Section 5.3 is used as a means for describing attacks in a technology-agnostic manner as part of test case generation (see Section 4.1).

2.5 Mealy Machines

Mealy machines are a specific form of state machines (or automata), which are a fundamental concept in computer science. Similar to LTS, Mealy machines provide a formal notation for systems' behaviors. The main difference is, that a Mealy machine provides an output for any input, which makes them an adequate representation for real-world cyber-physical systems. The definition of Mealy machines reads $M = (Q, \Sigma, \Omega, \delta, \lambda, q_0)$, with Q being the set of states, Σ the input alphabet, Ω the output alphabet (that may or may not identical to the input alphabet), δ the transition function ($\delta : Q \times \Sigma \rightarrow Q$), λ the output function ($\lambda : Q \times \Sigma \rightarrow \Omega$), and q_0 the initial state [21]. The transition and output functions might be merged ($Q \times \Sigma \rightarrow Q \times \Omega$). This thesis uses Mealy machines to represent learnt system behavior through observation of inputs and outputs in automata learning (see Section 4.2).

2.6 Automata Learning

Active automata learning is a method of actively querying systems and noting the output to a given respective input. This allows for inferring behavioral models of black-box systems. The classic method of automata learning, called the L* algorithm, uses the concept of the *minimally adequate Teacher* [22]. This teacher has (theoretically) perfect knowledge of system to learn and can answer two kinds of questions:

- *Membership queries* and
- *Equivalence queries*.

The membership queries' answers are denoted in an observation table, that eventually allows for trying to infer an automaton. The equivalence queries determine if the inferred automaton is correct. Lacking a teacher with perfect system knowledge in a black-box situation, the equivalence queries are ordinarily replaced by traditional conformance testing. More modern algorithms (like TTT [23]) rely on discrimination trees instead of observation tables to be more efficient [24]. This thesis uses both traditional and modern types of automata learning to infer behavioral component models (see Section 4.2).

2.7 Behavioral Equivalences

LTS and automata (particularly of the Mealy type used in this thesis) can be compared for their equivalence. In particular for the purpose of this thesis, an equivalent behavior is important. That means that two automata do not necessarily have to be identical (i.e. all states and transitions being identical), but merely the same input has to yield the same output. This equivalence can be evaluated by trace equivalence (i.e., assessing the same output from the same input) or various degrees of bisimulation [2]. For Mealy machines, bisimulation can be defined (with Q_1 and Q_2 being two compared Mealy machines as defined in Section 2.5) as [2]:

A) $q_{0_1} \in Q_1, q_{0_2} \in Q_2 \cdot (q_{0_1}, q_{0_2}) \in R$.

B) for all $q_1 \in Q_1, q_2 \in Q_2 \cdot (q_1, q_2) \in R$ must hold

   1) $\sigma \in \Sigma \cdot \lambda_1(q_1, \sigma) = \lambda_2(q_2, \sigma)$
   2) if $q_1\prime \in Post(q_1)$ then there exists $q_2\prime \in Post(q_2)$ with $(q_1\prime, q_2\prime) \in R$
   3) if $q_2\prime \in Post(q_2)$ then there exists $q_1\prime \in Post(q_1)$ with $(q_1\prime, q_2\prime) \in R$

This thesis uses behavioral equivalence for compliance checking (see Section 4.3).

Chapter 3

Research Overview

To increase the comprehensiveness and efficiency of both black box and white box verifying the correctness and security of systems, formalized methods are needed to improve structure and reproducibility. The overall objective is to research more comprehensive and efficient methods for verification through testing. This thesis therefore proposes a structured and automated way to model-based testing in order to leverage this objective at a architectural and at a component level. Although principally applicable to general correctness verification, the thesis ultimately proposes methods for assuring the cybersecurity of and enhancing the trust in systems with a special emphasis on communication protocols used in vehicular systems.

Generally, in the automotive domain, security engineering starts with defining security goals and requirements using a threat modeling process at an architectural level during the design phase. Once the design is implemented, the fulfillment of the derived security requirements has to be verified by testing [17]. As these components are often delivered as black boxes, their verification should not rely on internal systems knowledge. Because of this, the threat model provides context and prioritization of component tests. These tests should check a correct behavior of the system with regard to the security requirements. In other words, the components should comply to a certain specification – this specification is often based on a standard. On the other hand, a threat model is based on certain assumptions about the modeled components during design. If a component implementation later does not comply with the given specification, these assumptions may not hold, which makes the threat model and the resulting test prioritization inaccurate. The overall objective of this thesis is to provide a set of formal methods facilitates the automation of test generation from threat models, as well as automatically checking implementations for specification compliance (which again requires an automatic method to derive behavioral component models). As the component behavior has to work black box, because of the reasons stated above, the model generation concentrates on outside interfaces of that component, which is generally a specific implementation of a (standardized or proprietary) communication protocol. Based on our literature survey of previous approaches in these areas, we decided to concentrate on Automata Learning [24] to infer Mealy-type state machines of the behavior of implementations of communication protocols. Then we used trace and bisimulation equivalence checking [25] for compliance checks.

Overall Objective: Automatically generating formal attack descriptions from architectural models and use automata learning to verify whether the implementation satisfies a standard specification.

3.1 Research Goals

The issues stated above lead to three research goals, namely create working methods to perform:

RG1 Transforming a system's threat model into formal descriptions of cyber-attacks.

RG2 Automatically obtaining state machines of communication protocols from black box scenarios that can be used for correctness and security analysis.

RG3 Facilitating behavioral equivalence as a method for compliance checking of a learned implementation to a given specification (e.g., a standard).

The attack descriptions (RG1) provide a test scenario in the form of an abstract attack description for the overall system. This scenario is derived from threat modeling the architectural design using a rule set, that scrutinizes potential threats based on the architecture. The state models are inferred from implemented components (RG2) and the compliance checking of these state models (RG3) provide a verdict that verifies compliance to a specification (e.g., a standard). This compliance (RG3) is assumed beforehand in the design phase

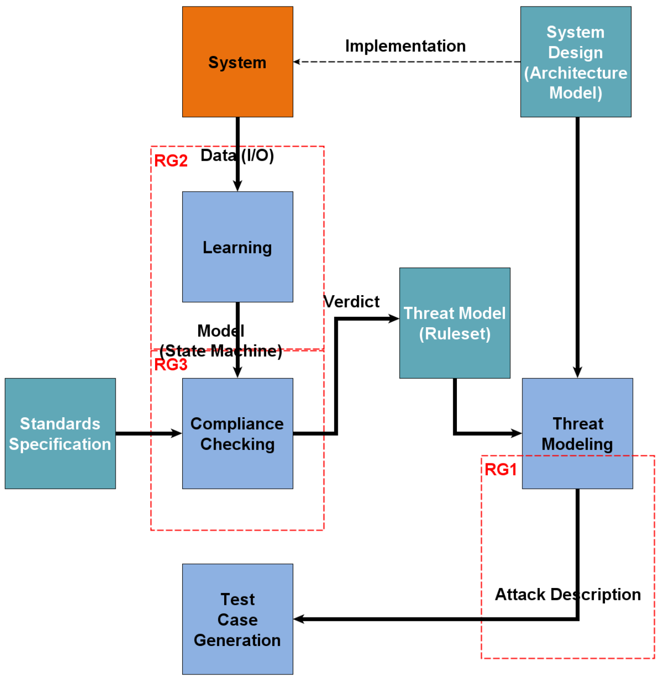


Figure 3.1: Positioning of the research goals in a structured testing process. Amber denotes artifacts, blue denotes activities, and cyan denotes specification inputs. The arrows denote inputs and outputs, with the dashed input denotes a process including output. The research goals are marked with the dashed red boxes.

during the architectural threat modeling (RG1 - that lead to the attack descriptions). A counterexample regarding the compliance provides (currently manually) input to the rule set for the threat modeling and therefore potentially leads to a different outcome of the threat modeling process. RG3 therefore provides an iterative feedback loop from the implementation back to the design phase and the threat model-based test generation in RG1. This also indirectly brings the standards specification into the threat modeling rule set. Figure 3.1 provides an overview of the research goals in the context of an exemplary automotive cybersecurity testing process. In this figure, amber denotes artifacts, blue denotes activities, and cyan denotes specification inputs. The arrows denote inputs and outputs, with the dashed input denotes a process including output. The research goals are marked with the dashed red boxes.

3.1.1 Threat Model-based Test Generation

A prominent example of model-based security analysis is the threat analysis and risk assessment (TARA) process widely used in the automotive industry [9]. It uses a threat modeling, based on an architectural design model, to identify threats and prioritize them in order to derive security goals and requirements, which ultimately results in security measures to be implemented in the architecture and components. Some kind of assessment in the fashion of a TARA (although not necessarily the exact same) is even prescribed by the automotive admission process in the UNECE region and the only recognized international standard for implementing a cybersecurity management system [7, 17]. Both admission and standard also mandate to verify cybersecurity measures by testing. In order to create these tests in an efficient manner, another goal of this thesis is to automatically derive test cases from the models made in the design phase to use it later after implementation to verify the efficacy of the planned security measures. The TARA process also determines the verification and validation planning and methods. In this process we included learning-based component testing as presented in RG2 and RG3 [1]. On the other hand, during threat modeling certain assumptions about the model elements are made (e.g., it is assumed that a component's communication complies with a certain standard) [26, 8]. If these assumptions do not hold, the model becomes inaccurate. It is therefore beneficial for the model's accuracy that the component's behavior is checked against the assumptions. When these assumptions can be formulated into a specification, the respective component's behavior can be automatically checked to comply with that specification. This behavioral compliance checking is formulated in RG3.
Contributing papers: Paper I, Paper II, Paper IV.

RG1: Create a method to derive formal descriptions of cyberattacks from a system's threat model.

3.1.2 Automated State Machine Derivation

Formal models have been used very broadly in both research and industrial applications. It is, however, very tedious and costly to create suitable models for correctness and security analysis manually. Furthermore, in some industries

like the automotive, the necessary information to manually creating models might not be present due to very long supply chains and/or non-disclosure. It is therefore beneficial to possess a method to automatically infer formal behavioral models (i.e., state machines) of systems under test in order to foster more efficient analysis and verification processes. As these state machines have to be derived from black box systems (due to the reasons stated above), the interfaces to interact with these systems are their respective implementations of communications protocols. These implementations are the first entry point for adversaries through faults and vulnerabilities. Inferring state machines for correctness and security analysis (as well as test generation) is therefore a significant building block for security improvement. In the course of this, it should also be examined how effective Automata Learning is to infer state machines of industrial real-world communication systems. In an automated process, attack descriptions derived from threat models (RG1) provide the V&V planning for components that should be examined, while the actual checking refers to RG3.
Contributing papers: Paper I, Paper II, Paper III.

RG2: Create a method to automatically obtain state machines of communication protocols from industrial black box scenarios to use these state machines for correctness and security analysis.

3.1.3 Compliance Checking

In many contemporary industries one of the main means for collaborations along the supply chain is written specifications and standards. These include (semi-)formal specifications like development interface agreements (DIAs), specifications in calls for tenders, as well as international and (de facto-) industry standards. To deliver a correct system it is crucial to comply with the respective specification. Furthermore, deviations from standards are most often faults that might lead to security vulnerabilities. Therefore, one goal of this thesis is to create a means for compliance-checking real-world systems in an automated manner. The compliance checking is targeted to be based on state machine models (as derived in RG2), as checking an accurate state machine uncovers consistent and inconsistent behavior both more comprehensively and efficiently, and, thus, more solid than using traditional conformance checking. The specification is modeled into a state machine and its behavior compared to that of learned state machine. Thereby, the behavioral aspect of the equiv-

alence is crucial: it is not necessary that a system's state machine is *identical* to a specification state machine, only that both state machines *behave* exactly the same. These checks also provide confirmation or rebuttal of assumptions made in threat modeling (based on specifications tailored to these assumptions) creating a link to RG1.
Contributing papers: Paper II, Paper III.

RG3: Demonstrate the applicability of behavioral equivalence as a method for compliance checking of a learned implementation against a given specification (e.g., based on a communication protocol standard)

3.2 Research Method

This thesis follows the Design Science Research Methodology (DSRM) [27]. First thoughts on Design Science were made by Simon in 1969 [28], where he asked how to scientifically scrutinize artificial artifacts of a certain complexity. Artificial in that sense means everything not being deducted strictly by (apodictic) natural laws, i.e., everything human-made, including engineering. Based on this fundament, Nunamaker et al. provided a framework for DSRM for Information Systems (IS) [29]. The framework provides a multi-methodological approach to IS research considering theory building, systems development, experimentation, observation, and their relations to each other. Their work also provides a process to research IS consisting of the following activities and underlying research issues. [29]:

1. Construct a conceptual framework
   - State a meaningful research question
   - Investigate the system functionalities and requirements
   - Understand the system building processes/procedures
   - Study relevant disciplines for new approaches and ideas
2. Develop a system architecture
   - Develop a unique architecture design for extensibility, modularity, etc.
   - Define functionalities or system components and interrelationships among them

3. Analyze and design the system
    - Design the database/knowledge base schema and processes to carry out system functions
    - Develop alternative solutions and choose one solution
4. Build the system
    - Learn about the concepts, framework, and design through the system building process
    - Gain insight about the problems and the complexity or the system
5. Experiment, observe, and evaluate the system
    - Observe the use or the system by case studies and field studies
    - Evaluate the system by laboratory experiments or field experiments
    - Develop new theories/models based on the observation and experimentation of the system's usage
    - Consolidate experiences learned

The conceptual framework (1) was done prior to the actual research by defining the overall objective and the research goals (Section 3.1) out of the motivational identified practical problems (Section 1.1), along with studying related work (Section 5). The system architecture (2) was defined in four ways: a) as a structured overall architecture for implementing a testing process (in paper I), b) a system architecture for model learning and model-based compliance testing (papers II and III), c) a conceptual architecture for test case generation from threat models and d) an meta-architecture concept for integrating the components a-c (in this thesis). As system for RG1 was designed (paper IV). Various approaches were considered and examined for RGs 2 and3 (mainly outlined in papers I, II and III). Based on this evaluation, a system implementation was built for learning and compliance checking, which was also used for extensive system evaluation (paper II).

Chapter 4

Research Contributions

This chapter contains the research contributions that have been made towards reaching the research goals stated in Section 3.1 and outlines the solutions, their novelty and their distribution among the publications. Table 4.1 gives an overview of the contributions of individual papers (outlined in Chapter 4.4) towards reaching the research goals. For each research a different solution is presented, namely:

Table 4.1: Publication contributions to the research goals.

Paper/Goal	1	2	3
I	X	X	
II	X	X	X
III		X	X
IV	X		

1 Threat Model-based Test Generation (achieving RG1)

2 Automated Model Derivation and Algorithm Evaluation (achieving RG2)

3 Compliance Checking (achieving RG3)

The threat model-based test generation (1) goes a little bit beyond RG1, as it also partially provides a V&V method selection, although the latter also is meant to contain model checking for test generation, which is reserved for future work (see Section 6.2). The automated behavior model (state machine)

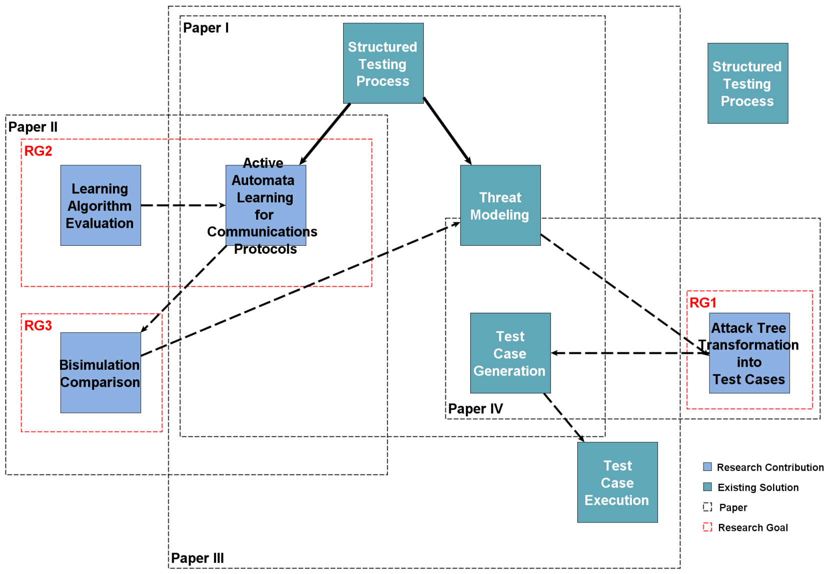

Paper I
Structured Testing Process
Structured Testing Process
Paper II
RG2
Learning Algorithm Evaluation
Active Automata Learning for Communications Protocols
Threat Modeling
RG3
Bisimulation Comparison
Test Case Generation
RG1
Attack Tree Transformation into Test Cases
Paper IV
Test Case Execution
Research Contribution
Existing Solution
Paper
Research Goal
Paper III

Figure 4.1: Relations of the main research contributions. Blue boxes mark the contributions, surrounded by the research goals in red dashed lines. The cyan boxes mark previous work the contributions build on, while the dashed black boxes denote the respective papers including the contributions and previous work. The arrows indicate dependencies; solid ones indicate sub-parts and dashed ones indicate inputs.

derivation (2) and the compliance checking (3) achieve to RG2 and RG3, respectively. The compliance checking yields a verdict that not only highlights deviations from a specification but also a counterexample that (manually) feeds back into threat model-based test generation by providing input for altering the threat modeling rule set (see Section 3.1). Figure 4.1 provides an overview of the contributions in relation to the research goals and to the thesis papers. Blue boxes mark the contributions, surrounded by the research goals in red dashed lines. The cyan boxes mark previous work the contributions build on, while the dashed black boxes denote the respective papers including the contributions and previous work. The arrows indicate dependencies; solid ones indicate sub-parts and dashed ones indicate inputs.

4.1 Threat Model-based Test Generation

To fulfill RG1, Paper IV presents a method for transferring threat models, via attack trees and labelled transition systems (LTS), into attack descriptions conceived in a domain-specific language (DSL). The method bases on an existing threat modeling tool [30] and an existing DSL called Agnostic domain-specific Language for the Implementation of Attacks (ALIA) [31] and concentrates on the transition between those two. ALIA is a procedural text-based language consisting of sequences of single actions (called test patterns) as a pseudo code that stand for specific steps of a composed attack in a technology-agnostic manner (not bound to a specific system-under-test). These steps will be translated into concrete executable instructions for specific systems-under-test based on Xtext and Xtend [32]. Alia also supports pre and post conditions and flow controls like conditionals and loops. With the labelling function of an LTS, the alphabet of ALIA will be attributed to paths within attack trees generated out of the threat model. Subsequently traversing the resulting LTS along a tree's path will automatically sequence that input and generate an attack description in ALIA. From the ALIA, it is possible to generate concrete test cases, once an implementation of the architecture is ready. Paper I describes a structured approach to derive a testing strategy from threat modeling RG1 and its connection to learning-based component testing (RG2), and Paper II describes the general embedding of threat modeling into a security testing process.

The approach of transferring a threat model into test descriptions using attack trees and LTS is novel.

4.2 Automated State Machine Derivation

A solution to achieve RG2 was developed by using active automata learning to infer behavioral communication protocol models (concretely state machines, also called automata) on the example of the handshake protocol (ISO 14443-3 [16]) of NFC systems. A similar system for a platooning protocol is in development (see Section 6.2). The developed solution consists of an NFC adapter interface library for the LearnLib library [33], containing the necessary adjustments (including compiling a new firmware) to an NFC hardware adapter along with an abstraction layer that transforms symbols from the learning algorithm to NFC hardware signals and vice versa. The solution also delivers insights on learning the ISO 14443-3 protocol, as it has some very characteristic features in the handshake protocol (particularly, two intertwined combination lock structures with almost identical states and the property that it does not send a response in case of a non-expected signal). In this setting also, different propagated learning algorithms were evaluated for their suitability and (surprisingly) an older algorithm was found best performing to learn correct implementations (namely the L* algorithm with the closing strategy by Rivest/Schapire), while (less surprisingly) the modern TTT algorithm was best performing when it comes to detecting flaws (see next chapter). The theory for the solution was worked out in Paper I, while Paper II describes the solution concept. Paper III contains the full solution implementation with a description of the complete details for deriving a state machine of NFC handshake protocol implementations.

Specifically, there is quite some previous work on automata learning-based approaches for learning communications systems (even one for NFC banking cards - see Section 5), but for the ISO 14443-3 protocol, so far no automated black box-learning solution has been presented.

4.3 Compliance Checking

RG3 was reached by comparing two automata using bisimulation and trace equivalence: one inferred using automata learning (see 4.2) and a second one based on a specification. While the basic concept was mentioned in Paper II, this was implemented for the NFC handshake protocol in Paper III. This way of checking the conformance is more comprehensive than traditional conformance checking trough testing, because it compares the complete behavior of a system with the complete behavior of a specification instead of merely testing a small subset in form of specific traces. The reason for using bisimulation and trace equivalence instead of a much simpler full (graph) identity is that standards compliance does not require the state machines to be identical, but merely to behave the same way. A system with a deviating automaton could still behave equivalent to and therefore be compliant to a specification (or standard).

There is (though few) previous work using the concept of automata learning paired with bisimulation for behavior comparison (see Section 5), however, no solution for practically working protocol performance checking.

4.4 Publications

This section contains an outline of each publication in this thesis consisting of an abstract, the work in this thesis' context, its contribution to the research goals and the author's contribution to the respective paper.

4.4.1 Paper I

Title: A Systematic Approach to Automotive Security
Authors: Masoud Ebrahimi, Stefan Marksteiner, Dejan Ničković, Roderick Bloem, David Schögler, Philipp Eisner, Samuel Sprung, Thomas Schober, Sebastian Chlup, Christoph Schmittner, and Sandra König
Abstract: We propose a holistic methodology for designing automotive systems that consider security a central concern at every design stage. During the concept design, we model the system architecture and define the security attributes of its components. We perform threat analysis on the system model to identify structural security issues. From that analysis, we derive attack trees that define recipes describing steps to successfully attack the system's assets and propose threat prevention measures. The attack tree allows us to derive a verification and validation (V&V) plan, which prioritizes the testing effort. In particular, we advocate using learning for testing approaches for the black-box components. It consists of inferring a finite state model of the black-box component from its execution traces. This model can then be used to generate new relevant tests, model check it against requirements, and compare two different implementations of the same protocol. We illustrate the methodology with an automotive infotainment system example. Using the advocated approach, we could also document unexpected and potentially critical behavior in our example systems.
Work in the thesis context: The paper outlines a structured process to verification by testing, containing threat modeling an black box-inferring behavioral models of systems using automata learning.
Contributes to research goals: RG1, RG2.
Thesis author's contribution: One equally contributing main author. Main responsible for section 4 (security testing), contributed parts of sections 1 and 2, complete section 4.1 and parts of 4.2. This corresponds to co-developing the overall testing concept based on learning methods, describing the automata learning theory and general parts of the use cases.

4.4.2 Paper II

Title: Approaches for Automating Cybersecurity Testing of Connected Vehicles
Authors: Stefan Marksteiner, Peter Priller, and Markus Wolf
Abstract: Vehicles are on the verge building highly networked and interconnected systems with each other. This requires open architectures with standardized interfaces. These interfaces provide huge surfaces for potential threats from cyber attacks. Regulators therefore demand to mitigate these risks using structured security engineering processes. Testing the effectiveness of this measures, on the other hand, is less standardized. To fill this gap, this book chapter contains an approach for structured and comprehensive cybersecurity testing of contemporary vehicular systems. It gives an overview of how to define secure systems and contains specific approaches for (semi-)automated cybersecurity testing of vehicular systems, including model-based testing and the description of an automated platform for executing tests.
Work in the thesis context: The paper outlines a concept to automate automotive cybersecurity testing using automata learning, incorporating the results of a threat model, an a platform for automated execution based on a domain-specific language.
Contributes to research goals: RG1, RG2, RG3.
Thesis author's contribution: Main driver and main author of this paper. Contributed all content except the introductory sections 1 and 2, 3.1, 4.4 and 4.5 (delivered review for these sections). This corresponds with the main concept, an automotive life cycle testing description, a testing process and a model-based testing concept based on automata learning.

4.4.3 Paper III

Title: Using Automata Learning for Compliance Evaluation of Communication Protocols on an NFC Handshake Example
Authors: Stefan Marksteiner, Marjan Sirjani, and Mikael Sjödin
Abstract: Near-Field Communication (NFC) is a widely proliferated standard for embedded low-power devices in very close proximity. In order to ensure a correct system, it has to comply to the ISO/IEC 14443 standard. This paper concentrates on the low-level part of the protocol (ISO/IEC 14443-3) and presents a method and a practical implementation that complements traditional conformance testing. We infer a Mealy state machine of the system-under-test

using active automata learning. This automaton is checked for bisimulation with a specification automaton modelled after the standard, which provides a strong verdict of conformance or non-conformance. As a by-product, we share some observations of the performance of different learning algorithms and calibrations in the specific setting of ISO/IEC 14443- 3, which is the difficulty to learn automata of system that a) consist of two very similar structures and b) very frequently give no answer (i.e. a timeout as an output).
Work in the thesis context: This paper contains an examination how to efficiently infer automata of black box NFC systems using automata learning and automatically comparing the behavior (using bisimulation) to a specification automaton, therefore comprehensively assessing the standards compliance of the system-under-test.
Contributes to research goals: RG2, RG3.
Thesis author's contribution: Main driver and main author of this paper. Contributed all of the content.

4.4.4 Paper IV

Title: From TARA to Test: Automated Automotive Cybersecurity Test Generation Out of Threat Modeling
Authors: Stefan Marksteiner, Christoph Schmittner, Korbinian Christl, Dejan Ničković, Mikael Sjödin, and Marjan Sirjani
Abstract: The UNECE demands the management of cyber security risks in vehicle design and that the effectiveness of these measures is verified by testing. This mandates the introduction of industrial-grade cybersecurity testing in automotive development processes. The regulation demands also to keep the risk management current, which again creates the need of stretching the testing over the full life cycle of an automotive system. Currently, the automotive cybersecurity testing procedures are not specified or automated enough to be able to deliver tests in the amount and thoroughness needed to keep up with that regulation, let alone doing so in a cost-efficient manner. This paper introduces an automotive security life cycle governance approach, that takes the currently being developed concepts of Cybersecurity Assurance Levels and Targeted Attack Feasibility into account and provides a means to automatically generate test cases at early development stages. These tests can also be used in later phases to verify and validate the implementations of developed systems. These formalized concepts increase the both the completeness and efficiency of auto-

motive cybersecurity testing over vehicles' complete life cycles.
Work in the thesis context: The paper contains an approach to derive an attack tree from a threat model and a concept to transform into an agnostic attack script in a domain-specific language, with a labelled transition system (LTS) as an intermediate step, allowing for automatically creating a test case once the modeled system is implemented.
Contributes to research goals: RG1.
Thesis author's contribution: Main driver and main author of this paper. Contributed sections 1,2, 3.2, and 5. This corresponds to the paper's motivation and a concept for transforming a threat model-based attack tree into attack descriptions written in an (formal) domain-specific language.

4.4.5 Related Papers Not Included in the Thesis

This section outlines papers not directly attributed to the thesis, but demonstrating professional and scientific skills expected for obtaining a licentiate degree. It contains two journal publications [5, 6].

Paper 1

Title: Wireless Security in Vehicular Ad Hoc Networks: A Survey
Authors: Thomas Blazek, Fjolla Ademaj, Stefan Marksteiner, Peter Priller, Hans-Peter Bernhard
Abstract: Vehicular communications face unique security issues in wireless communications. While new vehicles are equipped with a large set of communication technologies, product life cycles are long and software updates are not widespread. The result is a host of outdated and unpatched technologies being used on the street. This has especially severe security impacts because autonomous vehicles are pushing into the market, which will rely, at least partly, on the integrity of the provided information. We provide an overview of the currently deployed communication systems and their security weaknesses and features to collect and compare widely used security mechanisms. In this survey, we focus on technologies that work in an ad hoc manner. This includes Long-Term Evolution mode 4 (LTE-PC5), Wireless Access in Vehicular Environments (WAVE), Intelligent Transportation Systems at 5 Gigahertz (ITS-G5), and Bluetooth. First, we detail the underlying protocols and their architectural components. Then, we list security designs and concepts, as well as the currently known security flaws and exploits. Our overview shows the in-

dividual strengths and weaknesses of each protocol. This provides a path to interfacing separate protocols while being mindful of their respective limitations.
Work contribution in relation the licentiate study: Demonstrating the ability perform a systematization of knowledge survey of a specialized topic (the security posture of wireless ad-hoc networks relevant to automotive systems).
Thesis author's contribution: Main responsible for the security-related sections of the paper. Selecting relevant protocols in consensus with the co-authors, working out security measures implemented, and security flaws found in the respective protocols.

Paper 2

Title: A Global Survey of Standardisation and Industry Practices of Automotive Cybersecurity Validation & Verification Testing Processes and Tools
Authors: Andrew Roberts, Stefan Marksteiner, Mujdat Soyturk, Berkay Yaman, Yi Yang
Abstract: The United Nation Economic Commission for Europe (UNECE) Regulation 155 - Cybersecurity and Cybersecurity Management System (UN R155), mandates the development of cybersecurity management systems (CSMS) as part of a vehicle's life cycle. An inherent component of the CSMS is cybersecurity risk management and assessment. Validation and verification testing is a key activity for measuring the effectiveness of risk management and it is mandated by UN R155 for type approval. Due to the focus of R155 and its suggested implementation guideline, ISO/SAE 21434:2021 - Road Vehicle Cybersecurity Engineering, mainly centering on the alignment of cybersecurity risk management to the vehicle development life cycle, there is a gap in knowledge of proscribed activities for validation and verification testing. This research provides guidance on automotive cybersecurity testing and verification by providing an overview of the state-of-the-art in relevant automotive standards, outlining their transposition into national regulation and the currently used processes and tools in the automotive industry. Through engagement with state-of-the-art literature and workshops and surveys with industry groups, our study found that national regulatory authorities are moving to enshrine UN R155 as part of their vehicle regulations, with differences of implementation based on regulatory culture and pre-existing approaches to vehicle regulation. Validation and verification testing is developing aligned to UN R155 and ISO21434:2021, however, the testing approaches currently used

within industry, utilise elements of traditional enterprise information technology methods for penetration testing and tool sets. Electrical/electronic (E/E) components such as embedded control units (ECUs) are considered the primary testing target, however, connected and autonomous vehicle technologies are increasingly attracting more focus for testing.
Work contribution in relation the licentiate study: Demonstrating the ability to systematically conducting a comprehensive literature research (the stance of automotive standards towards cybersecurity testing) and conducting a survey-based research (survey on automotive cybersecurity testing tools and processes used in the industrial practice).
Thesis author's contribution: Main driver and corresponding author of this paper. Contributed sections on global standards and European regional (Germany, France, partly UK) standards, as well as mainly conducting the included industry survey on tools and processes.

Chapter 5

Related Work

This section considers other work in this field and adjacent fields. Given the research goals and contributions it is divided into contributions made in different fields namely: model-based test generation, automated state machine derivation and protocol learning, conformance checking, and applications of automata learning to cybersecurity.

5.1 Model-based Test Case Generation

Model-based testing (MBT) uses a model representation (normally behavioral, but also structural or other kinds) of a system-under-test. Model-based test case generation is an automated test case generation based on model-based testing. It is used in very diverse application domains like Information and Communications Technology (ICT), Automotive, Consumer electronic, Railway, Aerospace, Avionic, Tourism, Agriculture, Finance, Management, Construction, Sport, Automation. The used models include broad variety of different types (like state machines, activity and sequence diagrams, Simulink models, pre/post models, Simulink models, etc.) and the approaches to generate tests include structural coverage (based on control-flow, data-flow, transitions or UML), data coverage (boundary values, statistical or pairwise testing), fault and requirements-based criteria, and explicit and statistical test generation, state, search; model checking, requirements, event, random-based and others [34, 35].

5.2 Attack Trees-based Security Testing

Attack trees are a formal (graphical) representation of the set of possibilities to attack a certain system and have been described in the late 1990ies [10, 11]. They connect specific small attacks (i.e., exploiting threats) to a system in order to attack a complete system or a specific target inside a system with a combined or concatenated attack. The single attacks can be underlaid with different information like necessary skills or features or a success probability, allowing for also calculating this information (i.e., a complete set of skills or features or the combined attack success probability) for the complete attack. It further allows for selecting different paths through that tree that are most efficient regarding defined criteria (e.g., maximized success probability). There is also an approach that combines security-related attack trees with safety-related fault trees and also provides a translation mechanism to transfer them into stochastic timed automata. These can be analyzed using model checking [36]. This can be used to generate test cases. More directly, attack trees have been used to build fault injection-based attacks that can be used directly onto a system-under-test [37]. There is also work to adopt attack trees for automotive systems [38]. There is also an automotive-related method to create attack trees from threat models [26]. The thesis builds upon the later work by providing a translation mechanism from attack trees into a formal attack description language that provides blueprints for cyberattacks in RG1 (Section 3.1.1).

5.3 Formalized Test Descriptions

There is quite extensive work on languages for describing attacks to computer systems [39, 40, 41, 42, 43, 44]. However, this thesis builds upon a domain-specific language (DSL) tailored for automating attacks on automotive systems called *Agnostic domain-specific Language for the Implementation of Attacks (ALIA)* co-authored by the thesis author [31]. The language concept stems from the principle to abstract attacks on specific automotive systems from their (proprietary) technology-specific traits, leaving a blueprint structure for an attack that needs to be concretized again to be executed against a different system. The ALIA DSL is therefore designed for describing attacks on automotive systems in a technology-agnostic way. Apart from the original intention of porting attacks from one (proprietary) system to another, this allows for specifying attacks at design time and concretizing them once an implementation is available. This thesis integrates this language as a formal description for attacks to achieve RG1 (Section 3.1.1.

5.4 Automated State Machine Derivation and Protocol Learning

One of the key elements for fully automating model-based test case generation is automatically obtaining a suitable model to analyze. Finite state machines have been frequently used for correctness analyses [45, 46, 47, 48] possibilities to analyze them for their correctness and security properties. There are various approaches to automatically inferring (i.e., learning) state machines. Recurrent networks have been used to learn state machines already in the early 1990ies [49]. Some algorithms work on steering learning from traces by using a two-stage approach. They first analyze traces and mine a rule set and secondly using the rule set for learning automata from traces [50]. Others impose constraints on learning using linear temporal logic [51]. Many of the trace-based inferring methods base on the *KTail* algorithm [50]. This algorithm has been defined already 1972 by Biermann and Feldman [52]. Trace-based mechanisms are also used to generate other models like sequence diagrams [53]. Since the aim of this thesis is to black box-learning behavioral models of real-world systems, it concentrates on approaches actively querying a system. A method for this that has made many advances in the recent years is automata learning (for the basics see Section 2.6). There is quite some work of using automata learning for security analysis and testing, specifically for learning communication protocols [54, 55, 56, 57, 58, 59, 60]. This also includes NFC but concentrates on the upper layer (ISO/IEC 14443-4) protocol, dodging the specific challenges of the handshake protocol [61]. Aichernig et al. provided a benchmark for different automata learning setups using existing benchmark data [62]. This thesis also provides and automata learning performance evaluation, which is however very specially tailored for the ISO/IEC 14443-3 protocol, with accordingly different results. Recent works also concentrated on making use of these techniques for practical use, e.g., for security analyses [63, 60] or model-based fuzz testing [58]. The thesis differentiates from these works by combining learning with compliance checking and also using this to checking assumptions in threat modeling.

5.5 Conformance Checking using Equivalence of State Machines

There are, partly theoretic, approaches of learning a state machine and comparing it with other ones, targeting target DFAs [64] or probabilistic transition systems (PTS) [65]. For Mealy type machines, which (through their input and output behavior modeling) are better suited for describing reactive systems, Neider et al. provided some fundamental work, using automata learning and bisimulation [66]. Similar things were put into practice by viewing different machines as Labelled Transition Systems (LTS) for model comparison [67, 68] and to verify inferred embedded control software models [69]. However, there is no known comprehensive approach for using bisimulation for protocol compliance checking, which is the differentiation mark of this thesis compared to the described approaches (RG3 - Section 3.1.3)

Chapter 6

Conclusion and Future Work

This chapter summarizes the work included in the Licentiate thesis and outlines further directions to go from the current status of the research done in the thesis and generally in the research field.

6.1 Conclusions

The research described in this thesis aims for facilitating the usage of formal methods for generating tests to assure correctness and security with a focus on the automotive domain and on communication protocols. Following the need of the domain, we concentrated on generating test cases from the security analysis during the design phase, namely Threat Analysis and Risk Assessment (TARA) process and from the implementation, namely by checking the implementation's compliance with a specification using automata learning. The latter part provides feedback for the design phase: since TARA models systems components with given properties that are based on assumptions about a later implementation (e.g., conforming to international communication protocol standards), the actual compliance to a specification can prove these assumptions to be correct or incorrect. The first part leads to RG1, which is creating attack descriptions out of threat models, which can be used to generate concrete test cases for automotive systems once systems are implemented (Section 3.1.1). The second part is twofold, first we aim for mining a suitable model for security and correctness analysis (RG2 - Section 3.1.2) and second, we aim for a suitable methodology to use a behavioral equivalence with a specification as means for compliance checking (RG3 - 3.1.3).

Each of these research goals is met with a respective contribution namely, a method for test generation based on threat models (Section 4.1). This occurs by generating technology-agnostic test specifications written in the Agnostic domain-specific Language for the Implementation of Attacks (ALIA) out of attack trees derived from an existing tool for TARA (ThreatGet) using Labelled Transition Systems (LTS) as a means for the transformation. This approach is, to the best of our knowledge, novel. The second goal is met by automated state machine derivation based on active automata learning (Section 4.2). We showed the practical use of this technique by deriving state machines of Near-Field Communication (NFC) system for correctness and security analyses. We also provide insights on setups, abstraction and performance evaluations of different algorithms in special settings. The third goal was matched by a compliance checking method (Section 4.3). This method compares the behavior two state machines; one learnt from an implementation and one modeled after a specification (e.g., the ISO 14443-3 standard). We therefore use bisimulation and trace equivalence, which in combination with automata learning is novel for protocol conformance checking.

6.2 Future Directions

Despite the efforts taken in the licentiate thesis so far, quite some closely related problems have been left open to fulfill the overall objective in its entirety. Some of these directions are:

- Further implementations of adapters for specific protocols using the same, general learning framework. E.g., dealing with the specifics of V2X protocols or the reader parts of NFC systems in order to create a comprehensive multi-protocol learning framework. The research goal is to create a generally applicable method for protocol model inference.

- Create a method for automated model checking of the learned models for cybersecurity properties. The practical motivation is to automate cybersecurity analysis based on derived models, while the research goal is to derive general rules to check for the security of diverse communications protocols.

- Create a method to derive properties to check from threat models to rigorously check the underlying assumptions.

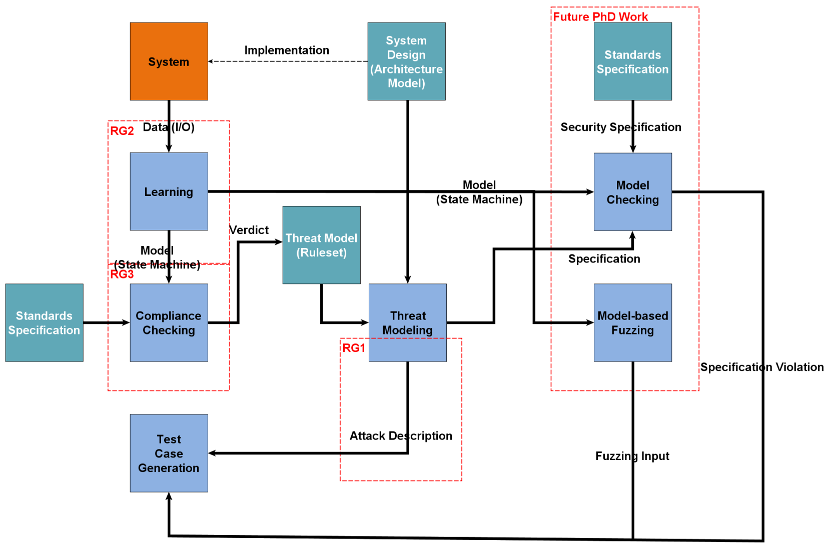


Figure 6.1: Positioning of the research goals and future directions in a a structured testing process. Amber denotes artifacts, blue denotes activities, and cyan denotes specification inputs. The arrows denote inputs and outputs, with the dashed input denotes a process including output. The research goals are marked with the dashed red boxes.

- Utilize the learned models for fuzz test generation using different strategies based on node and transition properties of the learned models. The research goal is to create highly efficient approaches for fuzz testing in order to create effective zero-input testing methods.

While the first item is basically applying the same principles this thesis uses to new domains, the latter two provide new methods for test case generation. Figure 6.1 shows an overview of the Licentiate research goals in relation with the latter three items above (shapes and colors are the same as in Figure 3.1).

Bibliography

[1] M. Ebrahimi, S. Marksteiner, D. Ničković, R. Bloem, D. Schögler, P. Eisner, S. Sprung, T. Schober, S. Chlup, C. Schmittner, and S. König, “A Systematic Approach to Automotive Security,” in *Formal Methods* (M. Chechik, J.-P. Katoen, and M. Leucker, eds.), vol. 14000 of *Lecture Notes in Computer Science*, (Cham), pp. 598–609, Springer International Publishing, 2023. Reproduced with permission from Springer Nature.

[2] S. Marksteiner, P. Priller, and M. Wolf, “Approaches For Automating Cybersecurity Testing Of Connected Vehicles,” in *Intelligent Secure Trustable Things* (M. Karner, J. Peltola, M. Jerne, L. Kulas, and P. Priller, eds.), Studies in Computational Intelligence, Springer Nature, 2023. Reproduced with permission from Springer Nature.

[3] S. Marksteiner, M. Sirjani, and M. Sjödin, “Using Automata Learning for Compliance Evaluation of Communication Protocols on an NFC Handshake Example,” in *Engineering of Computer-Based Systems* (J. Kofroň, T. Margaria, and C. Seceleanu, eds.), vol. 14390 of *Lecture Notes in Computer Science*, (Cham), pp. 170–190, Springer Nature Switzerland, 2024. Reproduced with permission from Springer Nature.

[4] S. Marksteiner, C. Schmittner, K. Christl, D. Nickovic, M. Sjödin, and M. Sirjani, “From TARA to Test: Automated Automotive Cybersecurity Test Generation Out of Threat Modeling,” in *Proceedings of the 7th ACM Computer Science in Cars Symposium*, CSCS ’23, (New York, NY, USA), pp. 1–10, Association for Computing Machinery, Dec. 2023.

[5] T. Blazek, F. Ademaj, S. Marksteiner, P. Priller, and H.-P. Bernhard, “Wireless Security in Vehicular Ad Hoc Networks: A Survey,” *SAE International Journal of Connected and Automated Vehicles*, vol. 6, Aug. 2022.

[6] A. Roberts, S. Marksteiner, M. Soyturk, B. Yaman, and Y. Yang, “A Global Survey of Standardization and Industry Practices of Automotive Cybersecurity Validation and Verification Testing Processes and Tools,” *SAE International Journal of Connected and Automated Vehicles*, vol. 7, Nov. 2023.

[7] United Nations Economic and Social Council - Economic Commission for Europe, “Uniform provisions concerning the approval of vehicles with regards to cyber security and cyber security management system,” Regulation ”155”, United Nations Economic and Social Council - Economic Commission for Europe, Brussels, 2021.

[8] A. Shostack, *Threat Modeling: Designing for Security*. Indianaplois, IN: John Wiley & Sons, 2014.

[9] D. Ward, I. Ibarra, and A. Ruddle, “Threat Analysis and Risk Assessment in Automotive Cyber Security,” *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, vol. 6, no. 2013-01-1415, pp. 507–513, 2013.

[10] C. Phillips and L. P. Swiler, “A graph-based system for network-vulnerability analysis,” in *Proceedings of the 1998 Workshop on New Security Paradigms*, (New York, NY, USA), pp. 71–79, ACM, 1998.

[11] B. Schneier, “Attack trees,” *Dr. Dobb’s journal*, vol. 24, no. 12, pp. 21–29, 1999.

[12] R. M. Keller, “Formal verification of parallel programs,” *Communications of the ACM*, vol. 19, pp. 371–384, July 1976.

[13] D. Angluin, “Learning regular sets from queries and counterexamples,” *Information and Computation*, vol. 75, pp. 87–106, Nov. 1987.

[14] R. L. Rivest and R. E. Schapire, “Inference of finite automata using homing sequences,” in *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC ’89, (New York, NY, USA), pp. 411–420, Association for Computing Machinery, Feb. 1989.

[15] M. Isberner, F. Howar, and B. Steffen, “The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning,” in *Runtime Verification* (B. Bonakdarpour and S. A. Smolka, eds.), Lecture Notes in Computer Science, (Cham), pp. 307–322, Springer International Publishing, 2014.

[16] International Organization for Standardization, “Cards and security devices for personal identification – Contactless proximity objects – Part 3: Initialization and anticollision,” ISO/IEC Standard ”14443-3”, International Organization for Standardization, 2018.

[17] International Organization for Standardization and Society of Automotive Engineers, “Road Vehicles – Cybersecurity Engineering,” ISO/SAE Standard ”21434”, International Organization for Standardization, 2021.

[18] V. Q. P. G. 13, “Automotive SPICE - Process Reference and Assessment Model for Cybersecurity Engineering,” Core Specification 1.0, Quality Management Center of the German Association of the Automotive Industry, 2021.

[19] T. R. Ingoldsby, “Attack tree-based threat risk analysis,” tech. rep., Amenaza Technologies Limited, 2021.

[20] M. Fowler and R. Parsons, *Domain-Specific Languages*. Pearson Education, Sept. 2010.

[21] G. H. Mealy, “A method for synthesizing sequential circuits,” *The Bell System Technical Journal*, vol. 34, pp. 1045–1079, Sept. 1955.

[22] D. Angluin, “Learning regular sets from queries and counterexamples,” *Information and Computation*, vol. 75, pp. 87–106, Nov. 1987.

[23] M. Isberner, F. Howar, and B. Steffen, “The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning,” in *Runtime Verification* (B. Bonakdarpour and S. A. Smolka, eds.), Lecture Notes in Computer Science, (Cham), pp. 307–322, Springer International Publishing, 2014.

[24] F. Vaandrager, “Model learning,” *Communications of the ACM*, vol. 60, pp. 86–95, Jan. 2017.

[25] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, Apr. 2008.

[26] S. Chlup, K. Christl, C. Schmittner, A. M. Shaaban, S. Schauer, and M. Latzenhofer, “THREATGET: towards automated attack tree analysis for automotive cybersecurity,” *Inf.*, vol. 14, no. 1, p. 14, 2023.

[27] A. Hevner and S. Chatterjee, “Design Science Research in Information Systems,” in *Design Research in Information Systems: Theory and Practice* (A. Hevner and S. Chatterjee, eds.), Integrated Series in Information Systems, pp. 9–22, Boston, MA: Springer US, 2010.

[28] H. A. Simon, *The Sciences of the Artificial*. Cambridge, MA, US: MIT press, 1969.

[29] J. F. Nunamaker, M. Chen, and T. D. Purdin, “Systems Development in Information Systems Research,” *Journal of Management Information Systems*, vol. 7, pp. 89–106, Dec. 1990.

[30] C. Schmittner, B. Schrammel, and S. König, “Asset Driven ISO/SAE 21434 Compliant Automotive Cybersecurity Analysis with ThreatGet,” in *Systems, Software and Services Process Improvement* (M. Yilmaz, P. Clarke, R. Messnarz, and M. Reiner, eds.), Communications in Computer and Information Science, (Cham), pp. 548–563, Springer International Publishing, 2021.

[31] C. Wolschke, S. Marksteiner, T. Braun, and M. Wolf, “An Agnostic Domain Specific Language for Implementing Attacks in an Automotive Use Case,” in *The 16th International Conference on Availability, Reliability and Security*, ARES 2021, (New York, NY, USA), pp. 1–9, Association for Computing Machinery, Aug. 2021.

[32] L. Bettini, *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing Ltd, Aug. 2016.

[33] M. Isberner, F. Howar, and B. Steffen, “The Open-Source LearnLib,” in *Computer Aided Verification* (D. Kroening and C. S. Păsăreanu, eds.), Lecture Notes in Computer Science, (Cham), pp. 487–495, Springer International Publishing, 2015.

[34] M. L. Mohd-Shafie, W. M. N. W. Kadir, H. Lichter, M. Khatibsyarbini, and M. A. Isa, “Model-based test case generation and prioritization: A systematic literature review,” *Software and Systems Modeling*, vol. 21, pp. 717–753, Apr. 2022.

[35] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. Elsevier, July 2010.

[36] R. Kumar and M. Stoelinga, "Quantitative security and safety analysis with attack-fault trees," in *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*, (Singapore), pp. 25–32, IEEE, 2017.

[37] A. Morais, E. Martins, A. Cavalli, and W. Jimenez, "Security protocol testing using attack trees," in *2009 International Conference on Computational Science and Engineering*, vol. 2, pp. 690–697, 2009.

[38] K. Karray, J.-L. Danger, S. Guilley, and M. Abdelaziz Elaabid, "Attack tree construction and its application to the connected vehicle," in *Cyber-Physical Systems Security*, pp. 175–190, Cham: Springer International Publishing, 2018.

[39] C. Michel and L. Mé, "Adele: An attack description language for knowledge-based intrusion detection," in *Trusted Information* (M. Dupuy and P. Paradinas, eds.), (Boston, MA), pp. 353–368, Springer US, 2001.

[40] M. Yampolskiy, P. Horváth, X. D. Koutsoukos, Y. Xue, and J. Sztipanovits, "A language for describing attacks on cyber-physical systems," *International Journal of Critical Infrastructure Protection*, vol. 8, pp. 40 – 52, 2015.

[41] M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, "Chapter one - security testing: A survey," in *Advances in Computers* (A. Memon, ed.), vol. 101, pp. 1 – 51, Elsevier, 2016.

[42] P. X. Mai, F. Pastore, A. Goknil, and L. C. Briand, "A natural language programming approach for requirements-based security testing," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 58–69, 2018.

[43] P. Johnson, R. Lagerström, and M. Ekstedt, "A meta language for threat modeling and attack simulations," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES 2018, (New York, NY, USA), Association for Computing Machinery, 2018.

[44] S. Katsikeas., P. Johnson., S. Hacks., and R. Lagerström., "Probabilistic modeling and simulation of vehicular cyber attacks: An application of the meta attack language," in *Proceedings of the 5th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, pp. 175–182, INSTICC, SciTePress, 2019.

[45] R. Alur and M. Yannakakis, “Model checking of hierarchical state machines,” *ACM SIGSOFT Software Engineering Notes*, vol. 23, pp. 175–188, Nov. 1998.

[46] K. Winter, *Model checking abstract state machines*. PhD thesis, Technische Universität Berlin, 2001.

[47] C. K. F. Tang and E. Ternovska, “Model Checking Abstract State Machines with Answer Set Programming,” in *Logic for Programming, Artificial Intelligence, and Reasoning* (G. Sutcliffe and A. Voronkov, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 443–458, Springer, 2005.

[48] F. Tsarev and K. Egorov, “Finite state machine induction using genetic algorithm based on testing and model checking,” in *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO ’11, (New York, NY, USA), pp. 759–762, Association for Computing Machinery, July 2011.

[49] Z. Zeng, R. M. Goodman, and P. Smyth, “Learning Finite State Machines With Self-Clustering Recurrent Networks,” *Neural Computation*, vol. 5, pp. 976–990, Nov. 1993.

[50] D. Lo, L. Mariani, and M. Pezzè, “Automatic steering of behavioral model inference,” in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE ’09, (New York, NY, USA), pp. 345–354, Association for Computing Machinery, Aug. 2009.

[51] N. Walkinshaw and K. Bogdanov, “Inferring Finite-State Models with Temporal Constraints,” in *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pp. 248–257, Sept. 2008.

[52] A. W. Biermann and J. A. Feldman, “On the Synthesis of Finite-State Machines from Samples of Their Behavior,” *IEEE Transactions on Computers*, vol. C-21, pp. 592–597, June 1972.

[53] M. McGavin, T. Wright, and S. Marshall, “Visualisations of execution traces (vet) an interactive plugin-based visualisation tool,” in *Proceedings of the 7th Australasian User interface conference-Volume 50*, pp. 153–160, 2006.

[54] C. Y. Cho, D. Babi ć, E. C. R. Shin, and D. Song, "Inference and analysis of formal models of botnet command and control protocols," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, (New York, NY, USA), pp. 426–439, Association for Computing Machinery, Oct. 2010.

[55] G. Argyros, I. Stais, S. Jana, A. D. Keromytis, and A. Kiayias, "SFADiff: Automated Evasion Attacks and Fingerprinting Using Black-box Differential Automata Learning," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, (New York, NY, USA), pp. 1690–1701, Association for Computing Machinery, Oct. 2016.

[56] P. Fiterău-Broştean, T. Lenaerts, E. Poll, J. de Ruiter, F. Vaandrager, and P. Verleg, "Model learning and model checking of SSH implementations," in *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software*, SPIN 2017, (New York, NY, USA), pp. 142–151, Association for Computing Machinery, July 2017.

[57] P. Fiterău-Broştean and F. Howar, "Learning-Based Testing the Sliding Window Behavior of TCP Implementations," in *Critical Systems: Formal Methods and Automated Verification* (L. Petrucci, C. Seceleanu, and A. Cavalcanti, eds.), Lecture Notes in Computer Science, (Cham), pp. 185–200, Springer International Publishing, 2017.

[58] B. K. Aichernig, E. Muškardin, and A. Pferscher, "Learning-Based Fuzzing of IoT Message Brokers," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pp. 47–58, Apr. 2021.

[59] A. Pferscher, B. Wunderling, B. K. Aichernig, and E. Muškardin, "Mining Digital Twins of a VPN Server," in *Preproceedings of the Workshop on Applications of Formal Methods and Digital Twins*, Sept. 2022.

[60] I. Karim, A. A. Ishtiaq, S. R. Hussain, and E. Bertino, "BLEDiff: Scalable and Property-Agnostic Noncompliance Checking for BLE Implementations," in *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 3209–3227, IEEE Computer Society, May 2023.

[61] F. Aarts, J. De Ruiter, and E. Poll, "Formal Models of Bank Cards for Free," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pp. 461–468, Mar. 2013.

[62] B. K. Aichernig, M. Tappler, and F. Wallner, "Benchmarking Combinations of Learning and Testing Algorithms for Automata Learning," *Formal Aspects of Computing*, June 2023.

[63] A. Pferscher and B. K. Aichernig, "Fingerprinting and analysis of Bluetooth devices with automata learning," *Formal Methods in System Design*, May 2023.

[64] Y.-F. Chen, C.-D. Hong, A. W. Lin, and P. Rümmer, "Learning to prove safety over parameterised concurrent systems," in *2017 Formal Methods in Computer Aided Design (FMCAD)*, pp. 76–83, Oct. 2017.

[65] C.-D. Hong, A. W. Lin, R. Majumdar, and P. Rümmer, "Probabilistic Bisimulation for Parameterized Systems," in *Computer Aided Verification* (I. Dillig and S. Tasiran, eds.), Lecture Notes in Computer Science, (Cham), pp. 455–474, Springer International Publishing, 2019.

[66] D. Neider, R. Smetsers, F. Vaandrager, and H. Kuppens, "Benchmarks for Automata Learning and Conformance Testing," in *Models, Mindsets, Meta: The What, the How, and the Why Not? Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday* (T. Margaria, S. Graf, and K. G. Larsen, eds.), Lecture Notes in Computer Science, pp. 390–416, Cham: Springer International Publishing, 2019.

[67] F. Aarts, H. Kuppens, J. Tretmans, F. Vaandrager, and S. Verwer, "Improving active Mealy machine learning for protocol conformance testing," *Machine Learning*, vol. 96, pp. 189–224, July 2014.

[68] M. Tappler, B. K. Aichernig, and R. Bloem, "Model-Based Testing IoT Communication via Active Automata Learning," in *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pp. 276–287, Mar. 2017.

[69] W. Smeenk, J. Moerman, F. Vaandrager, and D. N. Jansen, "Applying automata learning to embedded control software," in *Formal Methods and Software Engineering* (M. Butler, S. Conchon, and F. Zaïdi, eds.), (Cham), pp. 67–83, Springer International Publishing, 2015.

II

Included Papers

Chapter 7

Paper I: A Systematic Approach to Automotive Security

Masoud Ebrahimi, Stefan Marksteiner, Dejan Ničković, Roderick Bloem, David Schögler, Philipp Eisner, Samuel Sprung, Thomas Schober, Sebastian Chlup, Christoph Schmittner, Sandra König.

Proceedings of the Formal Methods 2023 conference, In Lecture Notes in Computer Science, vol. 14000. Cham: Springer International Publishing, 2023. DOI: 10.1007/978-3-031-27481-7_34. Reproduced with permission from Springer Nature.

Abstract

We propose a holistic methodology for designing automotive systems that consider security a central concern at every design stage. During the concept design, we model the system architecture and define the security attributes of its components. We perform threat analysis on the system model to identify structural security issues. From that analysis, we derive attack trees that define recipes describing steps to successfully attack the system's assets and propose threat prevention measures. The attack tree allows us to derive a verification and validation (V&V) plan, which prioritizes the testing effort. In particular, we advocate using learning for testing approaches for the black-box components. It consists of inferring a finite state model of the black-box component from its execution traces. This model can then be used to generate new relevant tests, model check it against requirements, and compare two different implementations of the same protocol. We illustrate the methodology with an automotive infotainment system example. Using the advocated approach, we could also document unexpected and potentially critical behavior in our example systems.

7.1 Introduction

The advent of *connected, cooperative automated mobility* provides a huge opportunity to increase mobility efficiency and road safety. However, the resulting connectivity creates new attack surfaces that affect the vehicle's safety, security, and integrity. With an estimated 100 million lines of embedded code, modern vehicles are highly complex systems that need to provide consistent cyber-security assurances. Indeed, there are an alarming spike in cyber-attacks targeting connected cars, their electronic control units (ECUs), and the original equipment manufacturer (OEM) back-end servers.

Therefore, making the right security decisions from the early design stages is crucial. The ad-hoc security measures done by domain experts are insufficient to meet the requirements in the automotive domain. The standard ISO/SAE 21434 and the mandatory regulation UN R155 advocate for more systematic reasoning about system security. The United Nations Economic Commission for Europe (UNECE) has adopted new security regulations, such as UNECE R155 and R156, for the homologation of future vehicles that address the identified cyber-attack risks, for example, during software updates. Similarly, the cyber security standard ISO/SAE 21434, introduced in 2021, defines precise security requirements for vehicles during the entire product life cycle, from its development to its operation and maintenance. Hence, there is an urgent need for methods and tools that address multiple security-related aspects, from early vehicle design to deployment and operation phases.

This paper proposes a top-down methodology for systematically assessing automotive security at different stages of vehicle development. The proposed methodology follows the product cycle in several steps. During the early design phase, we use threat modeling, analysis, and repair to provide more systematic support for the concept design of secure (automotive) systems. These methods allow us to identify the system's weaknesses in security threats and develop structural measures to prevent and mitigate them. We then use the threat analysis results to capture the system's critical components concerning security properties and derive a verification and validation (V&V) plan. We apply established processes (fuzz testing, penetration testing, etc.) for testing the implemented system components. However, the source code of the component implementation is often unavailable to the V&V team, and they cannot efficiently use the classical testing methods and tools. In that case, we advocate using automata learning for testing that builds an explainable model of a black-box implementation of a component from a set of executed test cases that facilitates testing and other V&V activities. This methodology is a re-

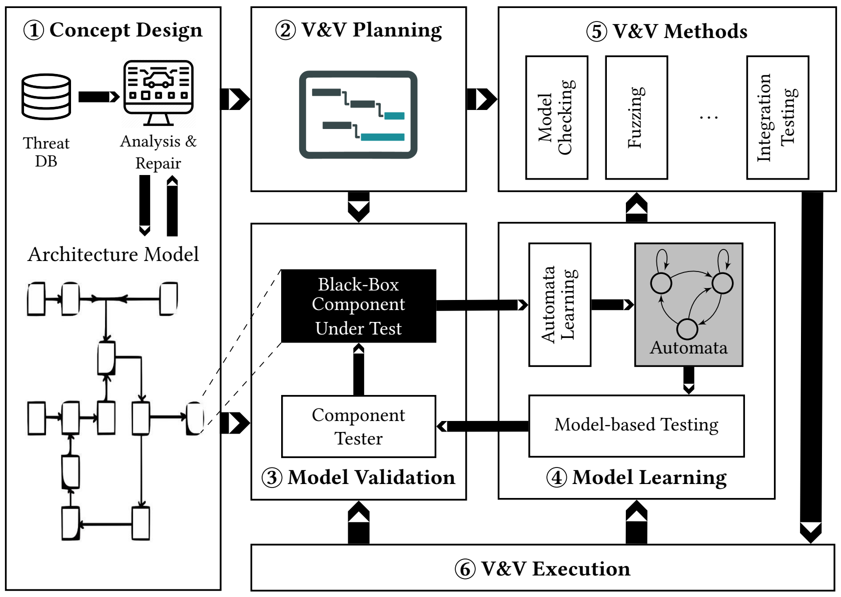


Figure 7.1: Overview of the TRUSTED methodology

sult of a joint research effort amongst the industrial and academic partners in TRUSTED[1], a project focusing on trust and security in autonomous vehicles. In implementing our proposed methodology, we were also supported by partners from the related LearnTwins[2] project, which focuses on learning-based testing methods for digital twins.

7.2 TRUSTED Methodology

The TRUSTED methodology starts with the concept design with a *threat model* of the vehicle; see Stage ① in Figure 7.1. The threat model consists of two components: (i) a system model architecture and (ii) a threat database. The system model architecture provides a structural view of the vehicle. This view includes vehicle components and subsystems (e.g., sensors, actuators, ECUs) and describes their (wireless or wired) interconnections. We can assign security

[1]https://TRUSTED.iaik.tugraz.at/
[2]https://learntwins.ist.tugraz.at/

attributes (e.g., authentication, encryption) to system components and communication links. A system model can define security boundaries that enclose trusted subsystems and assets we need to protect from potential attacks. The threat database contains a set of known threats—these threats from public domain sources, relevant standards, and previous experience. The threat model is an input to a threat analysis method allowing the detection of structural weaknesses in the system's architecture. We then combine the threat analysis with the repair activities to identify prevention and mitigation actions required to protect the system from identified threats.

The high-level threat analysis performed in the early stages of the design provides essential insights into the security-related weaknesses in the system architecture. We can take structural defense actions to improve the system's security based on threat repair outcomes (e.g., implementing authentication in a specific component). Yet, there is no guarantee that an attacker cannot break the resulting measures. Hence, it is imperative to have a solid verification and validation (V&V) plan. In the TRUSTED methodology, we use the insights gained by threat analysis and repair to identify risks and prepare an effective V&V plan corresponding to ② in Figure 7.1.

We use the system architecture model developed during the concept design phase to implement and integrate the components of the system. The implementation step is outside the scope of the TRUSTED methodology, but we assume the components are available as black boxes (see ③ in Figure 7.1). That is, we assume that we can execute components, but we cannot access their implementations.

During the development and integration of different components from the system architecture, verifying and testing safety and security functionalities becomes another critical aspect that we must address. Model validation (③ in Figure 7.1) tests the model for conformance against the component under test. This step provides either affirmation for the correctness (or completeness, respectively) of the model or counterexamples to refine the latter in a loop until the model is considered good enough to be used for test case generation.

We propose a learning-for-testing approach using automata learning (④ in Figure 7.1) as the core method for generating tests during V&V. In automata learning (see Section 7.4.1), we construct a Finite State Machine (FSM) of the System Under Test (SUT). We use the inferred FSM to: (1) obtain potential attack data, and (2) identify critical inputs that might show differences between the FSM and the SUT. We must automatically perform the necessary tests during the development and especially the maintenance phase to guarantee a quick response in the event of a threat.

We chose the learning-based testing approach due to its versatility and numerous V&V activities that we can undertake with the inferred FSM (⑤ in Figure 7.1). We can use the inferred FSM to: (1) visualize and understand the implementation, (2) model check it against its formalized requirements (possibly generating test cases on specification violations), (3) generate additional test cases by fuzz testing, and (4) Test for equivalence between implementation and a reference model or another implementation.

In the last phase (⑥ in Figure 7.1), we use various V&V strategies to verify the specified properties against the actual component under test. The test results are final verification outcomes; meanwhile, we can use them as counterexamples for the learning algorithms in ④ in Figure 7.1. This policy provides a feedback loop for refining the model in the learning-based testing approach. We execute and store tests using an automated test execution platform that augments generic test cases with additional information. This additional information comes from a test database or is provided in a grey box testing [1].

The threat model and the tests created during various design phases must be continuously maintained and updated throughout the vehicle lifecycle. We must incorporate new unknown threats and vulnerabilities into the model and re-evaluate the model to find new security issues. We must also integrate the changes to functions resulting from software updates into the system model and their impact on the vehicle's security analyzed and re-tested. This closely corresponds with the notions on testing in ISO 21434 and UNECE R155.

7.3 Automotive Security by Design

In this section, we demonstrate the use of THREATGET [2], a tool for threat modeling and analysis to improve the security of automotive applications during their early stages of design (step ① in Figure 7.1) and generate an appropriate V&V plan (step ② in Figure 7.1). We illustrate the approach with an automotive infotainment system developed by the industrial partner.

We first model the system using THREATGET (Section 7.3.1) and apply analysis to identify potential structural weaknesses in the system architecture (Section 7.3). We then use this analysis to derive a V&V plan (Section 7.3.3). Finally, we can augment it with threat repair to propose additional security measures [3].

7.3.1 System Architecture Model

We first create an accurate model of the automotive infotainment system (IS), shown in Figure 7.2. The IS is part of a larger ADAS reference model. It has several external interfaces that expose an attack surface of the vehicle. The external interfaces in Figure 7.2 are Bluetooth, WiFi, Interior Camera, and On-Board Diagnostics (OBD). The Multimedia Interface Hub (MIH) is an essential component of the infotainment system that (co-)implements core functionalities, including navigation, phone calls, and music playback. MIH also bridges external and internal interfaces. The Telematics Communication Unit (TCU) is the primary interface to the Internet. Many components in a modern vehicle depend on the TCU. For example, navigation systems use TCUs to access and update maps, and ECUs use them for over-the-air updates. Finally, all components except for TCU and Head Unit communicate through a CAN interface. We add two assets to the model – the confidentiality asset associated with the Head Unit and the availability asset associated with the TCU. The assets need to be protected, and their associated components are potential targets for attackers.

The IS is a weak security link in modern vehicles because it is more prone to successful cheap attacks than other components (e.g., *Body Control Unit* or the *Engine Control Unit*). This is due to versatile attack scenarios provided by the use of mainstream Unix-like operating systems, e.g., *Uconnect* and *Automotive Grade Linux*, the user requirements demanding functionalities like a built-in internet browser and installing third-party apps enabling remote code execution attacks, and the use of CAN bus that cannot guarantee communication integrity between the vehicle's external and internal interfaces.

7.3.2 Threat Analysis

We analyze the system model with THREATGET against its *threat database*, defining a set of possible threats formulated as *rules*. The threat descriptions are collected from multiple sources: automotive security standards and regulations (e.g., ISO/SAE 21434, ETSI, UNECE WP29 R155, and UNECE R156), publicly documented threats identified in past incidents, and expert knowledge.

We illustrate threat rules with two examples used during the analysis of the infotainment system model: the rule named "Gain Control of Wireless Interface (e.g., WiFi, Bluetooth, or BLE)" and the rule named "Flood CAN Communication with Messages". Both threat rules originate from automotive security analyses performed by domain experts. The first threat's formaliza-

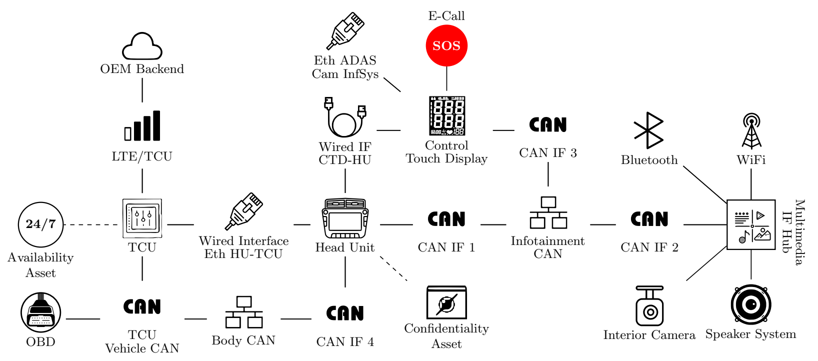


Figure 7.2: Automotive infotainment system model.

tion is

```
ELEMENT  : "Wireless Interface"{
    "Authorization" NOT IN ["Yes", "Strong"] &
    "Input Sanitization" != "Yes" &
    "Authentication" NOT IN ["Yes", "Strong"] &
    "Input Validation" != "Yes" &
    PROVIDES CAPABILITY "Control" := "true". }
```

This rule specifies that a wireless interface (e.g., WiFi or Bluetooth) that neither implements authorization and authentication nor sanitizes or validates its inputs is susceptible to threats. The last line in the rule explicitly states that if this threat is exploited, the malicious user can control the wireless interface. The “Threat Flood CAN Communication with Messages” threat is formalized as

```
FLOW {
    SOURCE ELEMENT  : "ECU"
        { REQUIRES CAPABILITY "Control" >= "true" } &
    TARGET ELEMENT  : "ECU" {
        HOLDS ASSET {
            "Cybersecurity Attribute" = "Confidentiality" &
            PROVIDES CAPABILITY "Read" := "true" } } &
    INCLUDES ELEMENT  : "BUS Communication" &
    INCLUDES NO ELEMENT  : "ECU" {"Anomaly Detection" = "Yes".} }
```

This rule states that the threat is present if there is a path starting from an ECU that is under the control of a malicious user to another ECU that holds the confidentiality asset and that there is a bus between them and no ECU on the path has implemented anomaly detection.

When applied to the infotainment system model, THREATGET identifies multiple threats. One threat is “Spoof messages in the vehicle network because of the missing components”. It describes a pattern that starts at an Interface with no Authentication and ends at an ECU with no Input Validation and holds an asset. It includes a wired Shared Medium representing a vehicle’s CAN BUS. Moreover, no element (of type Firewall, Server, ECU, or Gateway) on the flow from the Interface to the ECU takes care of Anomaly Detection.

We can address the identified threats with appropriate security measures. Threat repair [3] consists of preventing concrete threats by proposing security measures that can be implemented during the system’s design. THREATGET implements *attribute repair*, a method that proposes changes in the components’ security attributes as locally deployed measures with a simple cost model.

In the case of the automotive infotainment system model, e.g., the proposed threat repair measures include enabling authorization and implementing authentication in the WiFi and Bluetooth components. We note that threat repair does not remove the need for the planned V&V activities. The fact that authentication is integrated into the WiFi device, following the outcomes of threat repair, does not guarantee that the authentication algorithm’s implementation is weakness free. On the contrary, systematic testing of the WiFi’s authentication protocol is even more necessary to gain confidence that the WiFi device is not a possible entry point for malicious users.

7.3.3 V&V Planning

In addition to threat analysis, there is support for identifying and modeling more sophisticated threats using attack trees; c.f. [4]. This results in more knowledge about potential attackers’ steps when intruding into a system. Simple rules can be assigned attributes called capabilities that are either required for an intrusion or can be gained through the intrusion of a system component. Moreover, we can define the different access levels to a component (e.g., $Access < Read < Modify < Control$). Depending on previously acquired capabilities, different attack tree rules trigger, yielding distinct attack trees. An example of such a generated attack tree is illustrated in Figure 7.3.

The attack tree depicted in Figure 7.3 shows how a malicious user can access the *confidentiality* asset associated with the Head Unit via external interfaces such as WiFi and Bluetooth. For instance, control of the Bluetooth interface can be gained if its security attributes (input validation and sanitization, authorization and authentication) are not implemented or have weaknesses. From there, the user can gain control of the Multimedia Interface Hub, which is not sufficiently secure, and then get control of the Head Unit and hence the access to the asset. The attack tree exposes the most critical components that need to be protected. We note that the attack tree from Figure 7.3 is not maximal nor unique – while THREATGET generates multiple trees for each asset in the model, including the maximal attack trees, we use a simpler tree for illustration purposes.

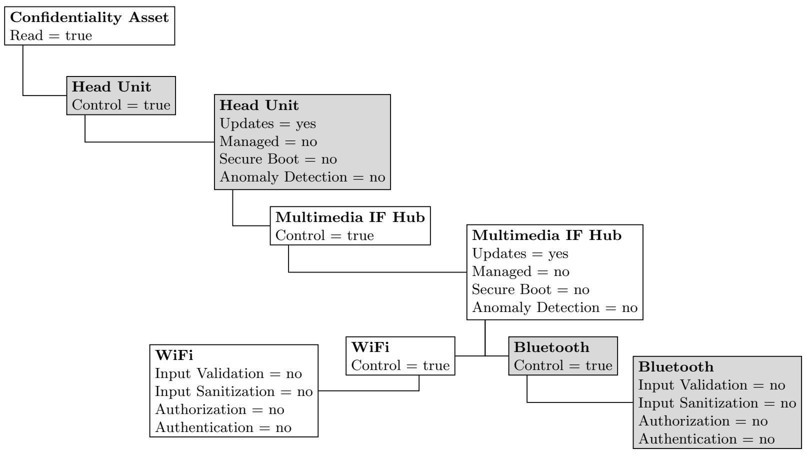


Figure 7.3: Attack tree derived from THREATGET. Multiple children from the same node are implicitly interpreted with an OR operation.

7.4 Automotive Security Testing

In this section, we advocate an approach based on learning to test critical components identified by the threat analysis methods during concept design, when these components are assumed to be black-box to the tester.

7.4.1 Automata Learning for Correctness

Many cyber-physical components in the automotive domain implement one or multiple finite state machines (FSMs).

Implementing larger automotive FSMs becomes cumbersome mainly because: (1) ensuring FSM's correctness w.r.t. its specification is expensive, (2) correctly coding the structure of a large FSM is difficult, and (3) correct integration of FSMs in complex software is hard.

Unfortunately, many software-driven components in the automotive industry are black boxes from different manufacturers, hence are hard to verify and thus do not provide functional or non-functional guarantees.

Given an FSM of a black-box automotive component, we can test and verify it to increase our confidence in its correctness. Automata learning has proven to be a successful method for learning-based testing of communication protocols that are also used in the automotive domain, e.g., MQTT [5] or Bluetooth Low Energy [6]. We use automata learning [7] to infer an FSM model (concretely a Mealy machine) of the the SUT. In the learning context we refer to the SUT by system-under-learning (SUL). In automata learning, a *learner* asks an *oracle* two types of queries. First, *membership queries* to determine the SUL's output for a given input word. Second, *equivalence queries* check whether a learned model conforms to the SUL, to which the oracle returns positive answer or a counterexample. A counterexample is an input-output word distinguishing SUL from hypothesis. In practice, oracles for black box systems work with conformance testing.

Ordinarily, real-world systems' alphabets are not manageable for learning algorithms. Abstraction helps to both cope with this fact and to make inferred models more human-readable. Too much abstraction, however, might induce non-deterministic behavior and hide problems we intend to find. There are also automatic abstraction refinement approaches for an optimum of abstraction in a mapper [8, 9]. An abstraction mapper consists of a mapping function that converts a concrete input into an abstract symbol. It also observes the SUL's concrete outputs and sends an abstraction to the learner. To send a concrete input to the SUL, the mapper inverses the abstraction. There are multiple methods to assess the behavioral correctness of the learned FSMs, including (1) black-box checking [10], adaptive model checking [11], a combination of learning-based testing and machine learning [12] and symbolic execution [13].

7.4.2 Use-Case Scenarios

The attack tree (see Figure 7.3) poses the critical components that need to be tested for security. In this section, we illustrate our learning-based testing approach on the two components highlighted in gray color in Figure 7.3 - the Bluetooth interface (as an entry vector) and the Head Unit ECU.

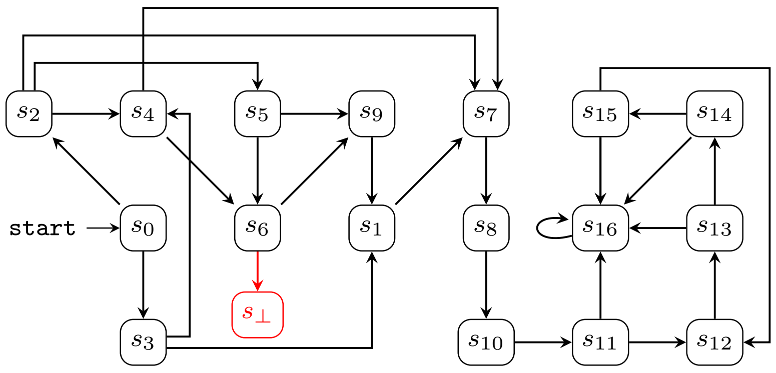


Figure 7.4: Inferred FSM structure for Bluetooth pairing.

Bluetooth and Bluetooth Low Energy

Bluetooth is a well-established standard for wireless audio used in most infotainment systems. Bluetooth Low Energy (BLE) grows in popularity for car access and sensor data transmission. The protocols have a variety of known vulnerabilities [14, 15, 16, 17, 18], some also specifically for automotive systems[3].

Learning Setup we use Intel Wireless Controllers (AC 8265 and AX200) implementing Bluetooth and BLE. The learning setups are similar, the difference is in the radio hardware and the physical layer, requiring three entities: (1) Radio Device, (2) Learner, and (3) Interface between the two with a mapper. The learner was implemented using the LearnLib framework [33].

Learned Model and Findings We inferred the pairing process models, which are used for encryption and therefore security-critical in the SULs. As a tangible result, we discovered a BLE deadlock state (red state in Figure 7.4) in the Linux BLE host software. With repeated out-of-order transmission of pairing requests of different types, we force the respective BLE stack into a state that limits the device to respond to basic link-layer control packets. After the state is reached, each following connection will start in this state until the controller is reset.

[3]https://research.nccgroup.com/2022/05/15/technical-advisory-tesla-ble-phone-as-a-key-passive-entry-vulnerable-to-relay-attacks/

Unified Diagnostic Services

Each ECU has a secure access mode reachable through its UDS implementation, available via vehicle's OBD connector. An attacker able to exploit UDS security features would be also able to manipulate data or even flash the ECU with a malicious firmware.

Learning Setup To communicate with the ECU we used a CAN interface. To learn a different ECU we only need to adapt the interface. We started by implementing a reduced UDS interface, consisting of instructions to put an ECU into secure access mode. Communications occures via a CAN bus interface. The learner was implemented using the AALpy framework [19].

Learned Model and Findings The learning experiment resulted in a reduced FSM of the UDS shown in Figure 7.5. An analysis of the results shows that once being successfully authenticated (state s_4), an incorrect authentication key will still result in the same state. This is unexpected and allows for prolonging a session without authentication. When requesting a new seed for re-authentication (s_5) this behavior persists. Moreover, on re-entering a secure session afterwards (from s_6), the ECU accepts an old key as well; an unexpected behavior after re-initiating the key authentication. Figure 7.5 marks all unexpected behaviors in red.

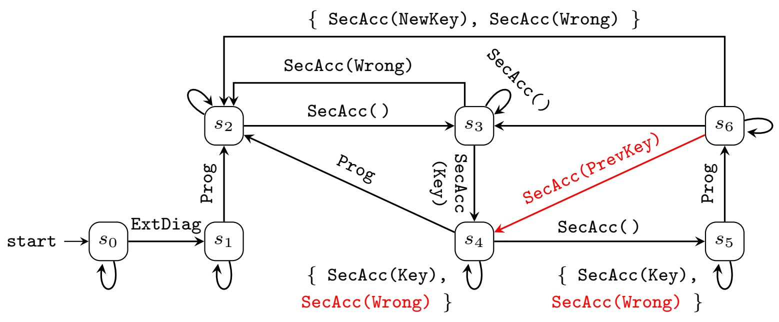


Figure 7.5: Inferred UDS FSM.

7.5 Conclusion

We introduced the TRUSTED methodology for designing and assessing trusted and secure automotive systems. The main novelty of the proposed methodology is its holistic and systematic approach to security, which starts at concept design and is carried down to the implementation and assessment of individual components. We instantiated the different parts of the methodology using the state-of-the-art methods and tools for threat modelling and analysis, automata learning and testing. We illustrated the use of the methodology by applying it step-by-step an automotive infotainment system. Using the learning-based testing approach we could document previously unpublished denial-of-service conditions in the examined BLE setups, as well as unexpected behavior allowing for extending secure UDS programming sessions on the scrutinized ECU.

Future Work We plan to further automate the transition from the concept design and V&V planning on one side, to the actual testing activities done on the level of components by devising a domain-specific test description language that can define abstract V&V plans derived from the attack trees, and be refined in a way so that eventually it can be executed on a platform (e.g., as in [31]). Second, the TRUSTED methodology mainly focuses on the transition from concept design to testing the implementation. We plan to also study the opposite direction – how to use the component testing results to update the system model and have a more refined threat analysis and a more realistic threat assessment.

Acknowledgements

This research received funding from the program “ICT of the Future” of the Austrian Research Promotion Agency (FFG) and the Austrian Ministry for Transport, Innovation and Technology under grant agreement No. 867558 (project TRUSTED) and within the ECSEL Joint Undertaking (JU) under grant agreement No. 876038 (project InSecTT). The JU receives support from the European Union’s Horizon 2020 research and innovation programme and Austria, Sweden, Spain, Italy, France, Portugal, Ireland, Finland, Slovenia, Poland, Netherlands, Turkey. The document reflects only the author’s view and the Commission is not responsible for any use that may be made of the information it contains.

Bibliography

[1] S. Marksteiner, N. Marko, A. Smulders, S. Karagiannis, F. Stahl, H. Hamazaryan, R. Schlick, S. Kraxberger, and A. Vasenev, "A Process to Facilitate Automated Automotive Cybersecurity Testing," in *2021 IEEE 93rd Vehicular Technology Conference (VTC Spring)*, (New York, NY, USA), pp. 1–7, IEEE, 2021.

[2] C. Schmittner, S. Chlup, A. Fellner, G. Macher, and E. Brenner, "Threatget: Threat modeling based approach for automated and connected vehicle systems," in *AmE 2020 - Automotive meets Electronics; 11th GMM-Symposium*, (Berlin), pp. 1–3, VDE Verlag, 2020.

[3] T. Tarrach, M. Ebrahimi, S. König, C. Schmittner, R. Bloem, and D. Nickovic, "Threat repair with optimization modulo theories," *CoRR*, 2022.

[4] M. Ebrahimi, C. Striessnig, J. C. Triginer, and C. Schmittner, "Identification and verification of attack-tree threat models in connected vehicles," in *SAE Intelligent and Connected Vehicles Symposium*, (Shanghai, China), pp. 1–12, SAE International, 2022.

[5] M. Tappler, B. K. Aichernig, and R. Bloem, "Model-Based Testing IoT Communication via Active Automata Learning," in *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pp. 276–287.

[6] A. Pferscher and B. K. Aichernig, "Fingerprinting Bluetooth Low Energy Devices via Active Automata Learning," in *Formal Methods* (M. Huisman, C. P˘ asăreanu, and N. Zhan, eds.), Lecture Notes in Computer Science, pp. 524–542, Springer International Publishing.

[7] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, pp. 87–106, Nov. 1987.

[8] F. Aarts, F. Heidarian, H. Kuppens, P. Olsen, and F. W. Vaandrager, "Automata learning through counterexample guided abstraction refinement," in *FM 2012*, (Berlin), pp. 10–27, Springer, 2012.

[9] F. Howar, B. Steffen, and M. Merten, "Automata Learning with Automated Alphabet Abstraction Refinement," in *Verification, Model Checking, and Abstract Interpretation* (R. Jhala and D. Schmidt, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 263–277, Springer, 2011.

[10] D. Peled, M. Y. Vardi, and M. Yannakakis, "Black Box Checking," in *Formal Methods for Protocol Engineering and Distributed Systems: FORTE XII / PSTV XIX'99 IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX) October 5–8, 1999, Beijing, China* (J. Wu, S. T. Chanson, and Q. Gao, eds.), IFIP Advances in Information and Communication Technology, pp. 225–240, Boston, MA: Springer US, 1999.

[11] A. Groce, D. Peled, and M. Yannakakis, "Adaptive Model Checking," in *Tools and Algorithms for the Construction and Analysis of Systems* (J.-P. Katoen and P. Stevens, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 357–370, Springer, 2002.

[12] K. Meinke, "Learning-Based Testing of Cyber-Physical Systems-of-Systems: A Platooning Study," in *Computer Performance Engineering* (P. Reinecke and A. Di Marco, eds.), Lecture Notes in Computer Science, (Cham), pp. 135–151, Springer International Publishing, 2017.

[13] B. K. Aichernig, R. Bloem, M. Ebrahimi, M. Tappler, and J. Winter, "Automata Learning for Symbolic Execution," in *2018 Formal Methods in Computer Aided Design (FMCAD)*, (Austin, Texas, USA), pp. 1–9, IEEE, 2018.

[14] D. Antonioli, N. O. Tippenhauer, and K. B. Rasmussen, "The KNOB is broken: Exploiting low entropy in the encryption key negotiation of bluetooth BR/EDR," in *28th USENIX Security Symposium, USENIX Security 2019* (N. Heninger and P. Traynor, eds.), (Santa Clara, CA, USA), pp. 1047–1061, USENIX Association, 2019.

[15] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, “BIAS: Bluetooth Impersonation AttackS,” in *2020 IEEE Symposium on Security and Privacy (SP)*, (San Francisco, CA, USA), pp. 549–562, IEEE, May 2020.

[16] D. Antonioli, N. O. Tippenhauer, K. Rasmussen, and M. Payer, “BLURtooth: Exploiting Cross-Transport Key Derivation in Bluetooth Classic and Bluetooth Low Energy,” in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS ’22, (New York, NY, USA), pp. 196–207, Association for Computing Machinery, May 2022.

[17] B. Seri and G. Vishnepolsky, “The dangers of Bluetooth implementations: Unveiling zero day vulnerabilities and security flaws in modern Bluetooth stacks.,” tech. rep., Armis Inc., 2017.

[18] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, “Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy,” *ACM Trans. Priv. Secur.*, vol. 23, pp. 14:1–14:28, June 2020.

[19] E. Muˇ skardin, B. K. Aichernig, I. Pill, A. Pferscher, and M. Tappler, “AALpy: An active automata learning library,” vol. 18, no. 3, pp. 417–426.

Chapter 8

Paper II: Approaches For Automating Cybersecurity Testing Of Connected Vehicles

Stefan Marksteiner, Peter Priller and Markus Wolf.

In Intelligent Secure Trustable Things, M. Karner, J. Peltola, M. Jerne, L. Kulas, and P. Priller, Eds., in Studies in Computational Intelligence. Springer Nature, 2023. DOI: 10.1007/978-3-031-54049-3_13. Reproduced with permission from Springer Nature.

Abstract

Vehicles are on the verge building highly networked and interconnected systems with each other. This requires open architectures with standardized interfaces. These interfaces provide huge surfaces for potential threats from cyber attacks. Regulators therefore demand to mitigate these risks using structured security engineering processes. Testing the effectiveness of this measures, on the other hand, is less standardized. To fill this gap, this book chapter contains an approach for structured and comprehensive cybersecurity testing of contemporary vehicular systems. It gives an overview of how to define secure systems and contains specific approaches for (semi-)automated cybersecurity testing of vehicular systems, including model-based testing and the description of an automated platform for executing tests.

8.1 Introduction

Mobility is a high priority in our society. Statistics report global annual car sales between 60 and 75 million[1] during recent years. According to the European Automobile Manufacturers' Association (ACEA), just in Europe approximately 350 million cars are currently in use [1], and the number grows to beyond 1 billion for a worldwide estimation. Cars are ubiquitous, for many families and businesses around the world, since decades. What has changed, however, is the fact that today's vehicles have become complex IT systems, often also called "computers on wheels". Modern cars run 100+ million lines of source code (MLOSC), and host complex computer networks both internally (in-vehicle networks) and externally. Many modern cars are now connected via the Internet to (maybe even multiple) cloud services, as well as to cellular networks (3G, LTE, 5G), and to specific vehicular networks (also known as car-to-car (C2C) or vehicle-to-everything (V2X), like ITS-G5). And that's not all: most vehicles also provide local networking capabilities (also called personal area networking, PAN). Typically based on WIFI and Bluetooth, it is used to connect to users' personal devices like smart phones and tablets, or to their home WLAN. To complement that already impressive array of wireless communication interfaces, some car manufacturers (or Original Equipment Manufacturers - OEMs) might add ultra-wide band (UWB) radios to communicate with car access systems like owner's keys or keycards. In addition, advanced driver assistance systems (ADAS) and future fully automated driving (AD) capabilities add GNSS receivers (Global Navigation Satellite System), TMC receivers (Traffic Message Channel), and active radar systems. Modern vehicles combine deeply complex software with exposure to a wide range of wireless networking technologies to both public and closed networks). In cybersecurity, this is called opening a large attack surface. This is worsened by the fact that vehicles are exposed for a much longer time than, e.g., personal computers (PC) or mobile devices like smart phones. Cars are in operation for some 15 years and more, which increases the threat that a vulnerability is found, shared, and at some point in time exploited by an attack. With such significant high exposure, let's consider potential threats which could evolve from malign attacks. Vehicles are highly dynamic (by nature), provide high levels of energy (storing 100kWh and more), are valuable (sometimes beyond 100k€) and exist as worldwide accessible objects in public, thus unrestricted places. When exploited by an attack taking over remote control, vehicles could become dangerous weapons, for both passengers inside, and for other road participants.

[1] https://www.statista.com/statistics/200002/international-car-sales-since-1990/

Worse, if groups of vehicles would come under attacker's control, they could be used to stage threats on city or even national level. State-sponsored attackers could stage war or terror attacks of not-yet-seen scale. Other scenarios might be less about harming humans, but could include denial-of service on single vehicles (e.g., owners cannot use their vehicles unless a ransom is paid) or on fleet level (e.g., blocking important road infrastructure, threatening whole communities, and some serious damage of a brand's reputation). And of course, there is simple car theft. Data privacy is also an important aspect. Modern cars might ”know” quite a lot about their users, including their past and present locations, driving habits, additional passengers, anything spoken in the vehicle, attention level while driving, contact information like phone numbers from connected personal devices, etc. An attack could therefore retrieve quite a lot of personal information and thus become considerable value to attackers. While not all of these threats have been discussed widely in public, the automotive industry is very much aware of it, and has stepped up efforts in designing more secure systems in cars, and establishing secure life cycle processes to provide necessary updates to fix vulnerabilities. An important part of securing these vehicular systems is the verification and validation of the effectiveness of taken security measures through testing. This testing needs to be done continuously through the life cycle (as new exploits might come up over time), and also as updating a system (or just a part of it) might alter its behavior an a way relevant to its security. In essence, (cyber)security testing must assure a system to display a small attack surface, be resilient and (possibly) to fix vulnerabilities before they are exploited in the wild.

The remainder of the chapter is structured the following way: Section 8.2 contains the current state of the art and related work. Section 8.3 contains measures for securing automotive systems. Section 8.4 contains specific approaches for automated cybersecurity testing of vehicular systems, including model-based testing and the description of an automated platform for executing tests. Section 8.5, eventually concludes this chapter.

8.2 State of the Art and Related Work

The automotive industry can draw from experience in other domains regarding security testing. General IT (managing e.g., corporate networks and IT systems) has established a history of penetration testing (abbreviated: pen testing), as simulated, authorized cyber-attacks. Typically executed by cybersecurity experts (acting as “white-hat hackers”), the goal is to identify weaknesses by letting these experts try to hack into the system under test (SUT) under predefined constraints (e.g., no physical access, no permanent harm), typically within a defined time window. If successful, these tests can thereby discover and document weaknesses. Translated to automotive industry, several companies offer similar pen-testing as a service on different levels (component, system, vehicle). While pen-testing might provide highly valuable insights into what level of security has been achieved for the vehicle, and might even uncover previously unknown vulnerabilities, it suffers from limited scalability and repeatability, as it is driven by and dependent on human experts. Security experts have toolboxes with highly effective tools (like the open source Metasploit framework[2]), but often need to supervise and configure these tools, and to adapt existing or write new scripts for complete attack chains to match a specific SUT. This requires skills and labor, and often involves considerable costs, which clearly limits scalability. Due to the sheer complexity of automotive software code (100+ MLOSC), it is also quite challenging for the experts to correctly hypothesize vulnerabilities, and to select (and execute) the most effective attacks, given the limited time available. This might heavily depend on expertise of the human testers, further limiting repeatability and comparability between pen tests campaigns. The threat of cyber attacks by adversaries has, however, also been recognized by standards and regulatory bodies. The United Nations Economic Council for Europe (UNECE) has issued a regulation (R 155 [2]) that prescribes the installation of a cybersecurity management system (CSMS). A CSMS is a process framework that accompanies the automotive development process over the complete life cycle and assures cybersecurity in every phase. Consequently, the International Organization for Standardization (ISO) and the Society of Automotive Engineers (SAE) have issued a joint standard (ISO/SAE 21434 [3]) that defines such a CSMS. As testing guidelines in these standards are somewhat underrepresented in contrast to security engineering, a structured approach is needed, e.g., as defined in [4, 5]. It further became clear that in order to establish dependable security covering all variants of vehicle lines in their full life cycle, supporting the upcoming accelerated software development cycles (automotive DevOps), an advanced process based on smart automation was required, as suggested in [6].

[2]https://github.com/rapid7/metasploit-framework

8.3 Automotive Cybersecurtiy Lifecycle Management

In order to maintain secure (and through, security-related impacts, also safe) vehicular systems, the respective system needs a security concept. The cybersecurity testing (see Section 8.4) will eventually validate and verify the effectivness of that concept. To establish a security concept for the complete life cycle of a vehicle for testing, we mainly rely on five pillars:

1. Threat Modeling (see Section 8.3.1)
2. Variant Management
3. Vulnerability Assessment
4. Automated Test Generation (see Section 8.4.2)
5. Process Governance

Threat modeling (see Section 8.3.1) is a widely proliferated technique in the automotive industry, mainly as part of a threat analysis and risk assessment (TARA) process [7].

As an OEM's fleet contains various vehicle model configurations, all of which contain tens of ECUs all of which again may display different hardware and software versions, keeping track of this potentially vast number of variants is crucial to determine the security posture of each member of the fleet. Our approach to tackle this problem is to use calibration data management that links technical attributes with software calibrations, to keep track of all ECU variations over the system's life cycle [8, 9]. This system, CRETA, contains exhaustive information about the variants, including their ECU firmware binaries.

This allows for the stored firmwares to be subsequently analyzed, generating a digital model of the software. To do so, firstly the firmware is extracted by iterating through the file tree, using an extraction algorithm and validating the extraction's correctness. The extracted software undergoes a composition analysis that pre-processes executables and normalizes the software in order to compare to a large database of mapped components, identified e.g. by file paths, file names, and characteristic strings in the software or configuration data, yielding a Software Bill-of-Materials (SBOM). Subsequently, the model is analyzed for security properties using pattern recognition. Patterns of known attacks from Common Vulnerabilities and Exposures (CVEs) are compared with each identified software library in the SBOM. Furthermore, the model undergoes a binary code analysis to find vulnerabilities not found in public databases: the binary is mapped in data and code sections, the code is then disassembled and later mapped into an intermediate language (for normalizing purposes) that allows for reconstructing the functions, analyzing the parameters and stack behavior and building control and data flows [10]. This matching, for instance, is able to identify common flaws like buffer overflows and, hence, is able to uncover zero-day vulnerabilities in software in a black box setting. Thirdly, patterns for proliferated code guidelines and relevant security standards are implemented, allowing for compliance checking against a given set of standards. This analysis, paired with full life cycle-coverage of the variants, allows for dealing with the parts lists and vulnerability management requirements mentioned above, as well as for verifying security requirements.

Vulnerabilities found in the code through pattern matching, however, are not necessarily exploitable for a variety of reasons. For instance, the location in the code could not be reachable, the impact of the vulnerability could be nullified through write protection of the memory or file system, or the interface might be protected by access controls. Therefore, the generated model also allows for model-based cybersecurity test case generation by using either the generated behavior model for model checking or by directly using the found patterns as basis for vulnerability exploitation [11]. We also aim for deriving test cases from threat modeling with a certain degree of automation (see Section 8.4.2).

To govern the process we developed our tool, FUSE, that guides activities of a given standard and provides standards-compliant documentation given the necessary input. We implemented ISO/SAE 21434 [3] and UNECE R155 [2] (as well as ISO 26262 [12], ISO 25119 [13]). The modeled objectives from the standards allow for providing all necessary artifacts for performing a review or audit, as well as keeping track of the conformance to relevant standards inside the development project.

8.3.1 Threat Modeling

One key element of cybersecurity analysis in all life cycle phases is threat modeling. This technique for security analysis is around for many years and well proliferated. It basically consists of modeling the information flows in an SUT and consequently examining them in a comprehensive way, e.g., via STRIDE or a similarly structured method [14].

Numerous software capable of performing a thread modeling process exists,

but prior to ThreatGet none was specifically developed for embedded or IoT systems. ThreatGet is a software tool developed by Austrian Institute of Technology (AIT) and based on Microsoft Enterprise Architect, a commonly used platform for systems model engineering [15].
It is used to examine models, objects, connections and charts in a system to enable iterative threat and risk analysis, covering the following categories:

- Actor,
- Sensor,
- Vehicle Unit,
- Data Store,
- Communication Interface,
- Communication Flow

Objects and connections in ThreatGet have so called tagged values at creation time. These describe analysis or security relevant properties of elements. It is recommended for users to extend the properties in addition to already proposed tagged values. Additionally, a database is used in the background that contains objects, which can also be extended by a user [15].

As an application example, Figure 8.1 shows the threat diagram of a communication flow inside ThreatGet. In this case, the environment data from the camera is directed to the ”Sensor Data Fusion and Decision Making” unit. After all diagrams are completed, a threat-overview is derived. An automatic risk evaluation consists of suggested values and can be adapted in a manual risk evaluation. In this step it is possible to rate the impact and occurrence of a threat at different levels and afterwards results can be exported in a report [15].

8.4 Cybersecurity Testing

In order to assure the cybersecurity of automotive systems and provide evidence for the appropriateness and effectiveness of security measures (according to a cybersecurity management system) , rigorous, structured and comprehensible testing is necessary [2]. Therefore a structured process, aligned with ISO/SAE 21434 [3] is recommendable. Such a process for testing could contain the following activities [5]:

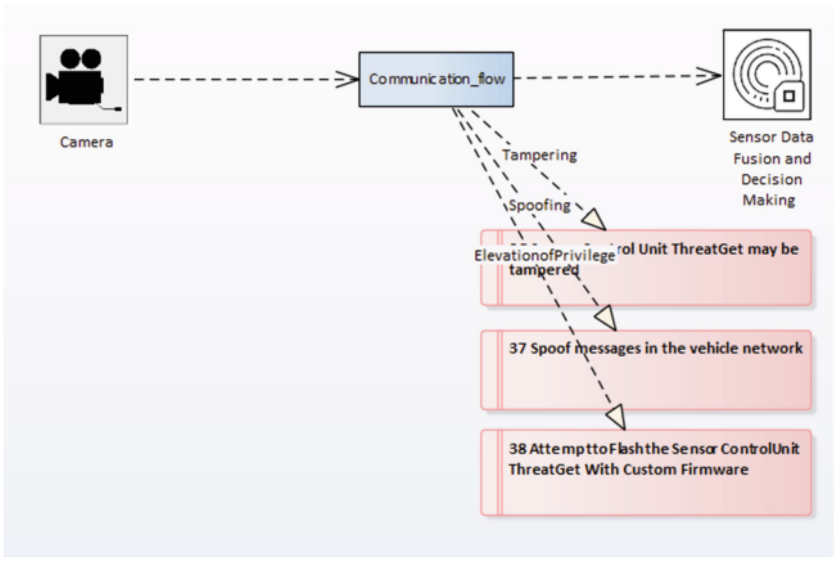

Camera
Communication_flow
Sensor Data Fusion and Decision Making
Tampering
Spoofing
ElevationofPrivilege
rol Unit ThreatGet may be tampered
37 Spoof messages in the vehicle network
38 AttempttoFlashthe Sensor ControlUnit ThreatGet With Custom Firmware

Figure 8.1: A list of found threats between the camera and the sensor data fusion and decision making [15].

1. Item Definition
2. Threat Analysis and Risk Assessment
3. Security Concept Definition (mainly including the test targets)
4. Test Planning and Scenario Development
    (a) Penetration Test Scenario Development
    (b) Functional and Interface Test Development
    (c) Fuzz Testing Scenario Development
    (d) Vulnerability Scanning Scenario Development
5. Test Script Development
    (a) Test Script Validation
6. Test Case Generation

   (a) Test Environment Preparation

7. Test Case Execution

8. Test Reporting

While items 1-3 correspond to a threat modeling process (see Section 8.3.1), the rest of them are the core testing process. To increase testing efficiency, these steps could be partially automated using model and learning-based approaches that can execute test planning and execution steps [6]. Here, the steps can be summarized into *concept design*. Item 4 *forms V &V planning*, while items 5 and 6 can be subsumed under *V & V Methods*. Finally, items 7 and 8 forms *V & V execution*. In between the planning and the methods, steps for automation can take effect: models from the concept design can be validated in an automated way and single components can be modeled using automated learning techniques and verified using methods from the V & V methods. An example of this used in the InSecTT project is described in Section 8.4.1. The full approach as described above consists of the following steps [6]:

1. Concept Design
2. V&V Planning
3. Model Validation
4. Model Learning
5. V&V Methods
6. V&V Execution

8.4.1 Learning-based Testing

Following the approach described above, we use learning, more concretely active automata learning to derive a model of a system [16]. The methodology uses a learner-teacher system where an all-knowing teacher answers the learning system queries about the SUT, in the context of cyber-physical systems ordinarily by providing the output to a series of inputs. The learner tries to infer a state machine from the given information. Once it has a hypothesis of a state machine that describes the observed behavior, it presents it to the teacher who then acknowledges the hypothesis as correct or gives a counterexample. This again, in real-world situations of black-box learning will mostly be simulated

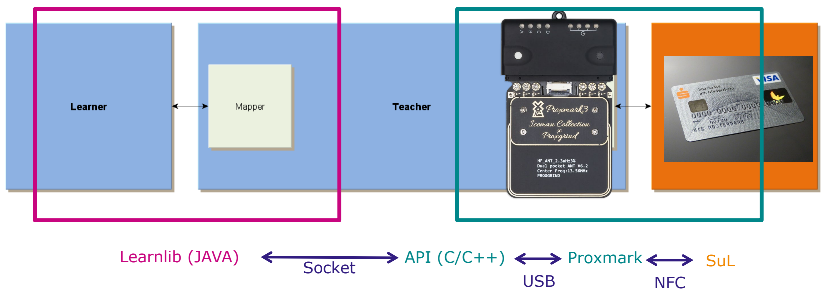


Figure 8.2: NFC Automata Learning Setup [23]

by conformance testing algorithms: if conformance is shown, the hypothesis is assumed as correct, otherwise a failing test sequence serves as a counterexample. The counterexample is taken as new input to refine the hypothesis and the learning continues until no more counterexamples are found. The this algorithm has been first formulated by Angluin [17] and has experienced significant improvements since (e.g., [18, 19]).

In accordance with the process outlined in Section 8.4, we use this technique to infer a model of a component. As a proof-of-concept we test a car access system based on Near-Field Communications (NFC). The testing setup consists on a learner (as described above) based on the *Learnlib* Java library [20] and a Proxmark NFC device [21] with an respective API that enables us to learn a model of the ISO 14443-3 NFC handshake protocol [22]. Figure 8.2 shows an overview of this setup. The used learning setup allows for inferring a state machine of the protocol and compare it to the specification in the standard to check its conformance. Figure 8.3 shows the learned model of the actual SUT (and NXP test card of a car access system prototype). Further use of the model is to do actual model checking or to use the model as an input for guided fuzz testing.

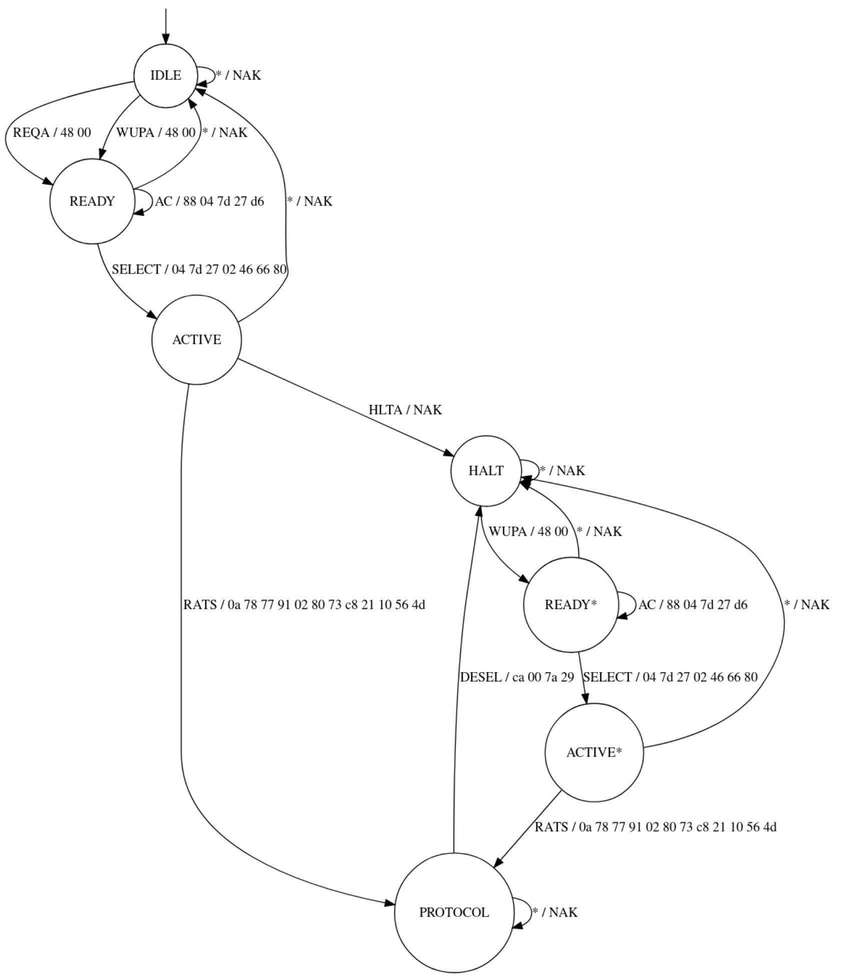


Figure 8.3: Learned Model of an NXP NFC Test Card

8.4.2 Model-based Test Case Generation

On a macroscopic level, a model of a complete vehicle as defined in the threat model (see Section 8.3.1) has to be explored in order to identify single components and generate test cases based on an attack tree [24, 25], a petri net [26, 27], or similar. If the SUT is modeled manually and, therefore, the components are known, this is trivial. If the setting is a black or grey box situation, we follow the approach to assume a generic model as starting point and test various components of the model by, e.g., send certain CAN messages for enumeration or try out an exploit that is known to affect a very broad variety of systems. Based on a comparison of the expected and actual output of the test, one can narrow down the set of likely components and system architectures (as described in [28]), e.g., based on SAT solving [29].

In order to generate test cases on a component level, a model must be transformed into a form that can be examined using a model checker (e.g. the Rebeca model checker [30] or SLAM [31]). Violations of the specification found by a model checker point towards an interesting position for a test case that could be extrapolated out of the traces leading to the respective states. There is also work regarding a toolchain using the UPPAAL framework [32].

Subsequently properties defining the security of a system shall be defined and used in the model checking. For c) where the model checking fails, a security problem might be present. The trace of the counter example can help in building a test case. Moreover, the input sequences used for the automata learning of the model shall be used to make test cases for the actual system-under-test. Using the traces as test vectors eliminate false positives from the model checking, as the exploitability of specification violations is test on the actual system. To concrete the abstract input, fuzzing techniques may be used [33].

8.4.3 Testing Platform

To realize the testing in the faction outlined in Section 8.4, a testing framework was developed and implemented. The high-level architecture was derived from the approach outlined in [4]. It has been adapted to suit the need of performing test in any phase of the product life cycle by adding co-simulation techniques into the testing framework architecture (see Figure 8.4 for an overview). The core component is a Security Testing Framework (see Section 8.4.4). It gains test cases from a generation engine that is fed by two sources: security functional tests from security requirements and penetration test attack vectors that have been tried out before (see description in Section 8.4.4) from a library. The core framework executes the attacks directly onto the SUT or into a co-simulation platform (indicated as *framework interfaces* in the figure) that interconnects various simulation parts: environment (i.e. other vehicles' and infras-

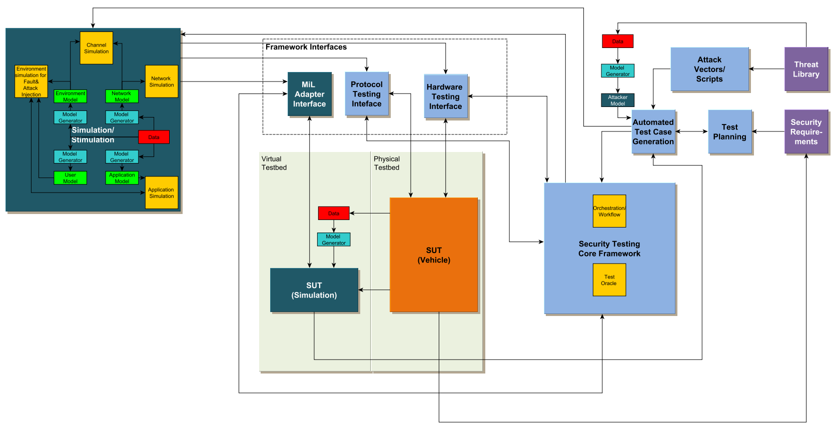


Figure 8.4: Overview of the Automotive Cybersecurity Testing Framework's high-level architecture

tructure's interference), network (generating mainly ITS-G5 traffic), channel (capable of simulating various physical layer signals as well as emitting them physically) and application (Section 8.4.4 contains an example with a platooning application). This way, each component can be stimulated the same way regardless if it is a physical or simulated component.

8.4.4 Automated Test Execution

For test execution, the test cases that were derived as described in previous chapters are fed into the automated test execution environment. Test cases are either manually written or generated in the ALIA DSL [34] format, which aims to provide an abstract and system agnostic representation of logical steps in a test case. Out of the main test-script and its included sub-scripts (containing frequently occurring blocks that handle a specific task such as opening a listener) a JSON Object is generated. These test case descriptions in DSL and JSON format are stored into a Database and can be accessed through the Orchestration Application; a platform independent web application that allows a user to manage information about the current SUT, schedule test execution and review results. This Orchestration Application then sends the test cases that the user wants to execute to the Execution Engine (AXE) and afterwards generates a report out of the received output from the AXE and the Test Oracle.

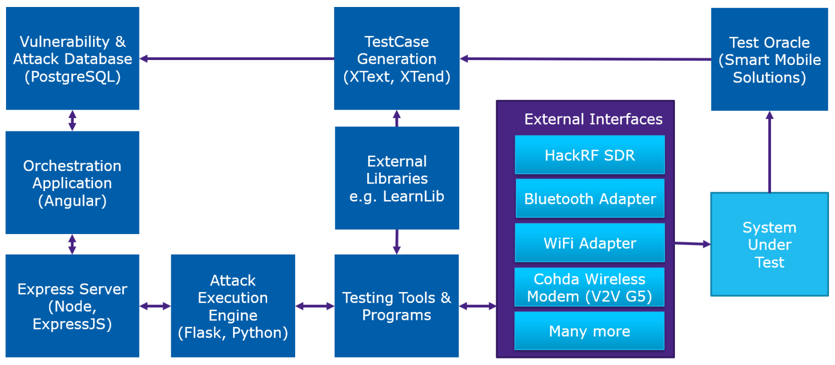

Vulnerability & Attack Database (PostgreSQL)
TestCase Generation (XText, XTend)
Test Oracle (Smart Mobile Solutions)
Orchestration Application (Angular)
External Libraries e.g. LearnLib
External Interfaces
HackRF SDR
Bluetooth Adapter
WiFi Adapter
Cohda Wireless Modem (V2V G5)
Many more
System Under Test
Express Server (Node, ExpressJS)
Attack Execution Engine (Flask, Python)
Testing Tools & Programs

Figure 8.5: AACT Test Execution Framework

The AXE is a Python based software that runs on an instance of Kali Linux and utilizes a variety of different interfaces, libraries and other software tools to perform a test case execution. It takes either a single test case or a structured collection of tests as input in JSON format and starts to subsequently execute contained steps. Figure 8.5 shows an overview of this architecture. This modular approach allows not only to target a specific SUT but also to control and parameterize whole (semi-) virtual SUT environments to manage SUT-behavior during a test scenario. Furthermore, it is possible to define and address different processes for tool execution which enables for example to host a malicious server, start a netcat listener and execute exploit code sequentially in a single test and afterwards perform code execution in an obtained reverse shell in the listener process.

One proof-of-concept use case implemented in the framework was security testing of the Ensemble platooning protocol [35] in a simulated environment. The concrete setup consisted of two truck simulations running on low-cost hardware connected via physical ITS-G5 [36] connection via Cohda modems. Another modem is used as an adversary to eavesdrop and interfere with the connection. The testing framework is able to start the simulation, so that the simulated trucks form a platoon. The actual test consists of a) listening to the communications b) distilling the session key out of a package c) cracking the key (for testing purposes, the key was reduced to eight bits) d) injecting a malicious message to disband the platoon. Figure 8.6 shows an overview of this setup. The result was that the injection failed for timing reasons, because the platoon keep-alive messages were sent in such a high frequency that they in-

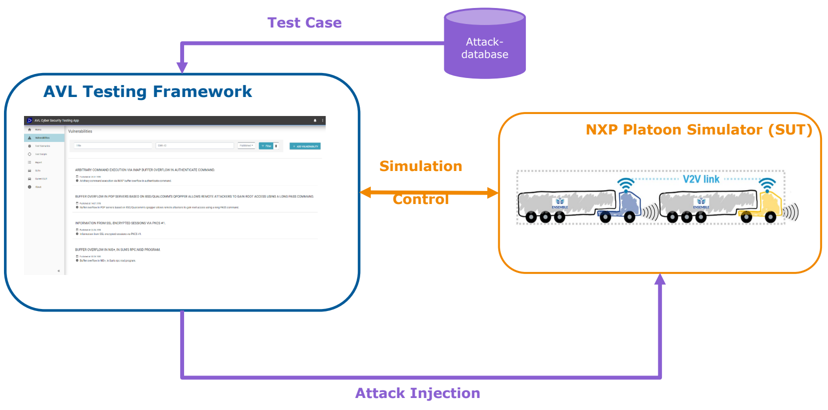


Figure 8.6: Platooning Use Case Overview

terfered with the break-up sequence. Even with reduced key (from AES-256 down to 8 bits) the protocol was secure against the tested attack. Furthermore, ITS-G5 built-in signatures, that were disabled for the test, would have prevented a successful injection. The test could therefore show the security of the protocol in an automated way as described above.

8.4.5 Fuzzing

The goal of fuzzing is to reach a non-intended state of a SUT by using completely or partially random input. The latter technique may use a structured frame structure that is compliant with communication standards used by the SUT and randomized payload data. [37] In case of a CAN-Bus, a fuzzing tool can create packets that consist of the standard ID Range (0 to 2047) and a previously learned or sniffed payload [38]. A fuzzer should include the following components [39]:

- A fuzz generator that assembles input from non-random components and random components with a sufficient amount of randomness
- A deliver mechanism that sends the generated inputs to the SUT
- A monitoring system (test oracle), which interprets the results such as

SUT responses, monitored network communication, debug interface output, system signals or other physical responses and performs decisions based on it e.g. if a test passes or fails.

By using this approach, no in-depth knowledge about the SUT is needed and every component that provides external interfaces can be targeted for testing, including ECU software, ECU hardware, protocols and busses (e.g. CAN). Fuzzing may be utilized in the automotive environment to [40]:

- Reverse engineer messages on busses
- Disrupt an in-vehicle communication network
- perform a cyber-attack
- lead to vehicle component damage.

Depending on the used interface and protocol it may not be possible to fuzz-test every possible combination of input in its entirety in a feasible time frame. Therefore, it makes sense to pre-select meaningful value and position ranges for randomized content. Because of this potentially large test case space, fuzzing may be applied in parallel to other test methods as long as the complete run-time is still in a defined range and produces positive results.

In case of the AVL AXE, fuzzing CAN bus signals is a very common use case. A fuzzing software e.g. booFuzz, American Fuzzy Lop or caring caribou is armed with valid CAN Messages or a template with a specification which parts of messages should be randomized and then handles the tasks of subsequently sending the (generated) data to the SUT as well as receiving and interpreting the feedback (such as Vector Tools CANoe).

8.5 Conclusion

This chapter showed a holistic approach of cybersecurity testing of modern vehicles over the complete life cycle. It showed how, proceeding from threat modeling and variant management, test cases can (semi-)automatically be derived using structured processes and learning techniques. The generated tests are subsequently executed on an automated platform that is capable of controlling the test and/or simulation setup and applying the respective attack vector. The described methodology provides an end-to-end means to test vehicular systems over the complete life cycle.

Acknowledgements

This research received funding within the ECSEL Joint Undertaking (JU) under grant agreement No. 876038 (project InSecTT). The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Sweden, Spain, Italy, France, Portugal, Ireland, Finland, Slovenia, Poland, Netherlands, Turkey. The document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

Bibliography

[1] European Automobile Manufacturers' Association (ACEA), "Vehicles in use europe 2022," tech. rep., European Automobile Manufacturers' Association (ACEA), 2021.

[2] United Nations Economic and Social Council - Economic Commission for Europe, "Uniform provisions concerning the approval of vehicles with regards to cyber security and cyber security management system," Regulation "155", United Nations Economic and Social Council - Economic Commission for Europe, Brussels, 2021.

[3] International Organization for Standardization and Society of Automotive Engineers, "Road Vehicles – Cybersecurity Engineering," ISO/SAE Standard "21434", International Organization for Standardization, 2022.

[4] S. Marksteiner and Z. Ma, "Approaching the Automation of Cyber Security Testing of Connected Vehicles," in *Proceedings of the Central European Cybersecurity Conference 2019*, CECC 2019, (New York, NY, USA), ACM, 2019.

[5] S. Marksteiner, N. Marko, A. Smulders, S. Karagiannis, F. Stahl, H. Hamazaryan, R. Schlick, S. Kraxberger, and A. Vasenev, "A Process to Facilitate Automated Automotive Cybersecurity Testing," in *2021 IEEE 93rd Vehicular Technology Conference (VTC Spring)*, (New York, NY, USA), IEEE, 2021.

[6] M. Ebrahimi, S. Marksteiner, D. Ničković, R. Bloem, D. Schögler, P. Eisner, S. Sprung, T. Schober, S. Chlup, C. Schmittner, and S. König, "A systematic approach to automotive security," in *Formal Methods* (M. Chechik, J.-P. Katoen, and M. Leucker, eds.), (Cham), pp. 598–609, Springer International Publishing, 2023.

[7] D. Ward, I. Ibarra, and A. Ruddle, "Threat Analysis and Risk Assessment in Automotive Cyber Security," *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, vol. 6, no. 2013-01-1415, pp. 507–513, 2013.

[8] T. Dobes, T. Kaserer, N. Schuch, and G. Storfer, "Smart Variant Calibration with Data Analytics," *ATZ - Automobiltechnische Zeitschrift*, no. Extra August 2018, 2018.

[9] M. Rathfelder, H. Hsu, T. Brandau, and G. Storfer, "Calibration Data Management for Porsche Chassis Systems," *ATZ worldwide*, vol. 118, pp. 16–21, June 2016.

[10] A. C. Franco da Silva, S. Wagner, E. Lazebnik, and E. Traitel, "Using a Cyber Digital Twin for Continuous Automotive Security Requirements Verification," *IEEE Software*, pp. 0–0, 2022.

[11] S. Marksteiner, S. Bronfman, M. Wolf, and E. Lazebnik, "Using Cyber Digital Twins for Automated Automotive Cybersecurity Testing," in *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, pp. 123–128, Sept. 2021.

[12] International Organization for Standardization and Society of Automotive Engineers, "Road vehicles – Functional safety," ISOStandard 26262, International Organization for Standardization, 2018.

[13] International Organization for Standardization, "Tractors and machinery for agriculture and forestry – Safety-related parts of control systems," ISOStandard 25119, International Organization for Standardization, 2018.

[14] A. Shostack, *Threat Modeling: Designing for Security*. John Wiley & Sons, 2014.

[15] M. El Sadany, C. Schmittner, and W. Kastner, "Assuring compliance with protection profiles with ThreatGet," in *Computer Safety, Reliability, and Security* (A. Romanovsky, E. Troubitsyna, I. Gashi, E. Schoitsch, and F. Bitsch, eds.), (Cham), pp. 62–73, Springer International Publishing, 2019.

[16] F. Vaandrager, "Model learning," *Communications of the ACM*, vol. 60, pp. 86–95, Jan. 2017.

[17] D. Angluin, “Learning regular sets from queries and counterexamples,” *Information and Computation*, vol. 75, pp. 87–106, Nov. 1987.

[18] M. Isberner, F. Howar, and B. Steffen, “The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning,” in *Runtime Verification* (B. Bonakdarpour and S. A. Smolka, eds.), Lecture Notes in Computer Science, (Cham), pp. 307–322, Springer International Publishing, 2014.

[19] R. L. Rivest and R. E. Schapire, “Inference of finite automata using homing sequences,” in *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC ’89, (New York, NY, USA), pp. 411–420, Association for Computing Machinery, Feb. 1989.

[20] M. Isberner, F. Howar, and B. Steffen, “The Open-Source LearnLib,” in *Computer Aided Verification* (D. Kroening and C. S. Păsăreanu, eds.), Lecture Notes in Computer Science, (Cham), pp. 487–495, Springer International Publishing, 2015.

[21] F. D. Garcia, G. de Koning Gans, and R. Verdult, “Tutorial: Proxmark, the swiss army knife for rfid security research: Tutorial at 8th workshop on rfid security and privacy (rfidsec 2012),” 2012.

[22] International Organization for Standardization, “Cards and security devices for personal identification – Contactless proximity objects – Part 3: Initialization and anticollision,” ISO/IEC Standard ”14443-3”, International Organization for Standardization, 2018.

[23] S. Marksteiner, M. Sirjani, and M. Sjödin, “Using Automata Learning for Compliance Evaluation of Communication Protocols on an NFC Handshake Example,” in *Engineering of Computer-Based Systems* (J. Kofroň, T. Margaria, and C. Seceleanu, eds.), vol. 14390 of *Lecture Notes in Computer Science*, (Cham), pp. 170–190, Springer Nature Switzerland, 2024.

[24] C. Phillips and L. P. Swiler, “A graph-based system for network-vulnerability analysis,” in *Proceedings of the 1998 Workshop on New Security Paradigms*, pp. 71–79, ACM, 1998.

[25] B. Schneier, “Attack trees,” *Dr. Dobb’s journal*, vol. 24, no. 12, pp. 21–29, 1999.

[26] C. A. Petri, *Kommunikation mit Automaten*. PhD thesis, Technische Universität Darmstadt, 1962.

[27] V. Varadharajan, "Petri net based modelling of information flow security requirements," in *[1990] Proceedings. The Computer Security Foundations Workshop III*, pp. 51–61, 1990.

[28] S. Marksteiner and P. Priller, "A Model-Driven Methodology for Automotive Cybersecurity Test Case Generation," in *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, pp. 129–135, Sept. 2021.

[29] S. Otten, T. Glock, C. P. Hohl, and E. Sax, "Model-based Variant Management in Automotive Systems Engineering," in *2019 International Symposium on Systems Engineering (ISSE)*, pp. 1–7, 2019.

[30] M. Sirjani, "Rebeca: Theory, applications, and tools," in *Formal Methods for Components and Objects, 5th International Symposium, FMCO 2006, Amsterdam, The Netherlands, November 7-10, 2006, Revised Lectures* (F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, eds.), vol. 4709 of *Lecture Notes in Computer Science*, pp. 102–126, Springer, 2006.

[31] T. Ball, B. Cook, V. Levin, and S. K. Rajamani, "Slam and static driver verifier: Technology transfer of formal methods inside microsoft," in *International Conference on Integrated Formal Methods*, (Berlin, Heidelberg), pp. 1–20, Springer, 2004.

[32] F. Aarts, F. Heidarian, H. Kuppens, P. Olsen, and F. W. Vaandrager, "Automata learning through counterexample guided abstraction refinement," in *FM 2012*, (Berlin), pp. 10–27, Springer, 2012.

[33] B. K. Aichernig, E. Muškardin, and A. Pferscher, "Learning-Based Fuzzing of IoT Message Brokers," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pp. 47–58, Apr. 2021.

[34] C. Wolschke, S. Marksteiner, T. Braun, and M. Wolf, "An Agnostic Domain Specific Language for Implementing Attacks in an Automotive Use Case," in *The 16th International Conference on Availability, Reliability and Security*, ARES 2021, (New York, NY, USA), pp. 1–9, Association for Computing Machinery, Aug. 2021.

[35] A. Ladino, L. Xiao, K. Adjenugwhure, N. Deschle, and G. Klunder, "Cross-platform simulation architecture with application to truck platooning impact assessment," in *ITS World Congress*, 2021.

[36] European Telecommunications Standards Institute, "Intelligent transport systems (its); vehicular communications; basic set of applications; definitions," ETSI "TS 102 638", European Telecommunications Standards Institute, 2009.

[37] R. McNally, K. K.-H. Yiu, D. A. Grove, and D. Gerhardy, "Fuzzing: The state of the art," 2012.

[38] P. Lapczynski, H. Heinemann, T. Schöneberger, and E. Metzker, "Automatically generating fuzz tests from automotive communication databases," 5th escar USA, Detroit, isits AG, June 2017.

[39] H. Lee, K. Choi, K. Chung, J. Kim, and K. Yim, "Fuzzing can packets into automobiles," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pp. 817–821, 2015.

[40] D. S. Fowler, J. Bryans, S. A. Shaikh, and P. Wooderson, "Fuzz testing for automotive cyber-security," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 239–246, 2018.

Chapter 9

Paper III: Using Automata Learning for Compliance Evaluation of Communication Protocols on an NFC Handshake Example

Stefan Marksteiner, Marjan Sirjani, and Mikael Sjödin.

In Proceedings of the Engineering of Computer-Based Systems Conference (ECBS) 2023, J. Kofroň, T. Margaria, and C. Seceleanu, Eds., in Lecture Notes in Computer Science, vol. 14390. Cham: Springer Nature Switzerland, 2023. DOI: 10.1007/978-3-031-49252-5_13. Reproduced with permission from Springer Nature.

Abstract

Near-Field Communication (NFC) is a widely adopted standard for embedded low-power devices in very close proximity. In order to ensure a correct system, it has to comply to the ISO/IEC 14443 standard. This paper concentrates on the low-level part of the protocol (ISO/IEC 14443-3) and presents a method and a practical implementation that complements traditional conformance testing. We infer a Mealy state machine of the system-under-test using active automata learning. This automaton is checked for bisimulation with a specification automaton modelled after the standard, which provides a strong verdict of conformance or non-conformance. As a by-product, we share some observations of the performance of different learning algorithms and calibrations in the specific setting of ISO/IEC 14443-3, which is the difficulty to learn models of system that a) consist of two very similar structures and b) very frequently give no answer (i.e. a timeout as an output).

9.1 Introduction

In this paper we describe an approach of very thoroughly evaluating the compliance of Near-Field Communications (NFC)-based chip systems with the ISO/IEC 14443-3 NFC handshake protocol [1] using formal methods, concretely automata learning and equivalence checking. We present a tool chain that is easy to use - both the learning and the equivalence checking can run fully automatic. A complete automaton of the system-under-test (SUT) compared with a specification automaton modeled after the standard, provides a strong complement to conformance testing. The remainder of this paper structures as follows. First we provide its motivation and contribution. Section 9.2 gives an overview of basic concepts in this paper, including a formal definition of bisimulation for Mealy Machines as used in this paper. Section 9.3 describes the developed interface for automata learning of NFC systems, while Section 9.4 describes the learning setup including a comparison of different algorithms and calibrations to be most suitable for the specifics of the NFC handshake protocol. Section 9.5 shows real-world results, while Section 9.6 compare them to the works of others. Section 9.7, eventually, concludes the paper and gives and outlook on future work.

9.1.1 Motivation

As the NFC protocol is widely adopted in a broad variety of different, often security-critical, chip systems like banking cards, passports, access systems, etc., that use relatively weak hardware, a correct implementation is utterly important. While there are many works about security weaknesses in NFC (e.g., [2, 3]), also specifically regarding the ISO/IEC 14443-3 handshake (e.g., [4, 5]), there is few works on comprehensive testing (see Section 9.6). Assuring the correctness of the system is a principal step in the quest to trustworthy systems. As there is, to the best of our knowledge, no comprehensive works regarding assessment of the handshake protocols, as the fundament of secure protocols build atop, we aim for a strong verdict of ISO compliance for NFC systems. To make this verdict more scalable than manual modeling, yet strongly verified, we choose automata learning to automatically infer a formal model of the implementations under scrutiny. For the actual compliance checking, we use bisimulation and trace equivalence checks against a specification automaton from the ISO/IEC 14443-3 standard (a rationale is given in Section 9.2.2).

9.1.2 Contribution

Overall, this paper is on the interface between communications protocols, embedded systems and formal methods. This work provides the following contributions for people with scholarly or applied interest in this approach of strong compliance checking:

- Insights regarding the specifics of learning NFC using active automata learning
- An evaluation on the performance of different learning algorithms in systems with very similar structures
- Developing an NFC interface for a learning system
- An approach for automated compliance checking using bisimulation and trace equivalence

We saw the NFC handshake to be specific in two aspects: a) it consists of two parts that are very similar and hard to distinguish for Learners and b) the vast majority of outputs from a system-under-learning are timeouts. This has severe impact on the learning where we examined different algorithms and configurations. The maximum word length has an impact on correctly inferring an automaton: too short yields incomplete automata, too long seemed to have a negative performance impact. Surprisingly the L* algorithm [6] with Rivest/Schapire (LSR) closure [7] surpassed more modern ones in learning performance. For discovering deviations from the standard, the minimum word length was found to have an impact. Here, the TTT algorithm [8] performed best, also followed by LSR. We further created a concrete hardware/-software interface using a Proxmark device and an abstraction layer for NFC systems. Lastly, we integrated bisimulation and trace equivalence checking into the learning tool chain, which enables completely automated compliance checking with counterexamples in the case of deviations from the standard.

9.2 Preliminaries

This section outlines the theoretical fundamentals of state machines and automata learning including a definition of equivalence and bisimilarity in the context of this paper. It further briefly describes the used framework and the basics and characteristics of the scrutinized protocol.

9.2.1 State Machines

A state machine (or automaton) is a fundamental concept in computer science. One of the most widely used flavors of state machines are Mealy machines, which describe a system as a set of states and functions of resulting state changes (transitions) and outputs for a given input in a certain state [9]. More formally, a Mealy machine can be defined as $M = (Q, \Sigma, \Omega, \delta, \lambda, q_0)$, with Q being the set of states, Σ the input alphabet, Ω the output alphabet (that may or may not identical to the input alphabet), δ the transition function ($\delta : Q \times \Sigma \to Q$), λ the output function ($\lambda : Q \times \Sigma \to \Omega$), and q_0 the initial state. The transition and output functions might be merged ($Q \times \Sigma \to Q \times \Omega$). An even simpler type of automaton is a deterministic finite acceptor (DFA) [10]. It lacks of an output (i.e. no Ω and no λ), but instead it has a set of accepted finishing states F, which are deemed as valid final states for an input word (i.e. sequence of input symbols), resulting in a definition of $D = (Q, \Sigma, \delta, q_0, F)$. The purpose is to define an automaton that is capable of deciding if an input word is a valid part of a language. A special subset of DFAs are combination lock automata (with the same properties) but the additional constraint that an invalid symbol in an input sequence would set the state machine immediately back into the initial state [11].

9.2.2 Transitions and Equivalence

An element of the combined transition/output function can be defined as 4-tuple ($\langle p, q, \sigma, \omega \rangle$) with $p \in Q$ as origin state of the transition, $q \in Q$ as destination state, $\sigma \in \Sigma$ as input symbol and $\omega \in \Omega$ as output symbol. Generally, to conform to a standard, a system must display the behavior defined in that standard. The ISO 14443-3 standard [1] describe the states of the NFC handshake with their respective expected input and result. . That means one can derive an automaton from this specification. The problem of determining NFC standard compliance can therefore be seen as comparing two (finite) automata. There is a spectrum of equivalences between Labeled Transition Systems (LTS) including automata. For being compliant with a standard, not necessarily every state and transition must be identical as long as the behavior of the system is the same. There might be learned automata that deviate from the standard automaton and still be compliant, e.g., if they are not minimal (the smallest automaton to implement a desired behavior). Figure 9.1 shows a very simple example of a three-state automaton and its behavior-equivalent (minimal) two-state counterpart.

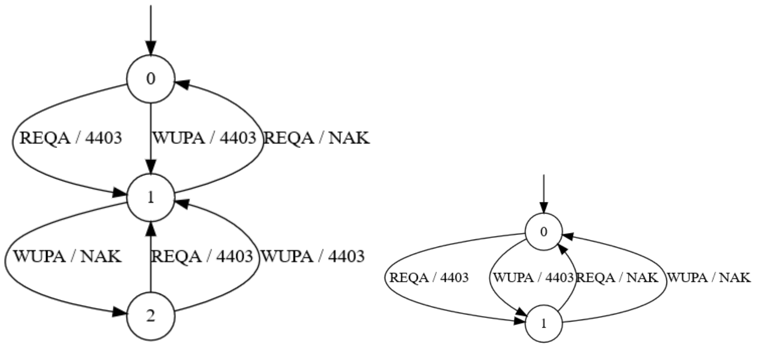


Figure 9.1: Example for a partial automaton and its minimal counterpart.

To compare this type of equivalence between two LTS LTS_1 and LTS_2, commonly used are (various degrees of) simulation, bisimulation (noted as $LTS_1 \sim LTS_2$) and trace equivalence. Simulation means that one automaton can completely reproduce the behavior of the other, for the bisimulation, this relation becomes bidirectional (i.e. functional). Trace equivalence compares the respective output of automata. Just (uni-directional) simulation alone is not sufficient as this would only the presence *or* absence of a certain behavior with respect to the specification, while the standard compliance mandates both. Bisimilarity of two transition systems is originally defined for labeled transition systems (LTS), defined as $LTS = (S, Act, \to, I, AP, L)$, with S being the set of states, Act a set of actions, $\to$ a transition function, I the set of initial states, AP a set of atomic propositions and L a labelling function.

Definition 1 (Bisimilarity)**.** *Bisimlarity of two LTS LTS_1 LTS_2 is defined as exhibiting a binary relation $R \subseteq QxQ$, such that [12]:*

A) $\forall s_1 \in I_1 \exists s_2 \in I_2 \cdot (s_1, s_2) \in R$ *and* $\forall s_2 \in I_2 (\exists s_1 \in I_1 \cdot (s_1, s_2) \in R$*.*

B) for all $(s_1, s_2) \in R$ *must hold*

1) $L_1(s_1) = L_2(s_2)$

2) if $s_1\prime \in Post(s_1)$ *then there exists* $s_2\prime \in Post(s_2)$ *with* $(s_1\prime, s_2\prime) \in R$

3) if $s_2\prime \in Post(s_2)$ *then there exists* $s_1\prime \in Post(s_1)$ *with* $(s_1\prime, s_2\prime) \in R$

Condition A of Definition 1 means that all initial states must be related, while Condition B means that for all related states the labels must be equal (1) and their successor states must be related (2-3). Formally the succession ($Post$) is defined as $Post(s, \alpha) = \{s\prime \in S | s \xrightarrow{\alpha} s\prime\}$ and $Post(s) = \bigcup_{\alpha \in Act} Post(s, \alpha)$, meaning the union of all action successions, which again are again the result the transition function with a defined action and state as input. As this is recursive, a relation of the initial states implies that all successor states are related. Since all reachable states are (direct or indirect) successor states of the initial states, this definition encompasses the complete LTS. We interpret Mealy machines as LTS using the output functions as labeling functions for transitions and the input symbols as actions, similar to [13]. Based on this, we define Mealy bisimilarity (M_1 M_2) for our purpose follows:

Definition 2 (Mealy Bisimilarity). *A) $q_{0_1} \in Q_1, q_{0_2} \in Q_2 \cdot (q_{0_1}, q_{0_2}) \in R$.*

B) for all $q_1 \in Q_1, q_2 \in Q_2 \cdot (q_1, q_2) \in R$ must hold

1) $\sigma \in \Sigma \cdot \lambda_1(q_1, \sigma) = \lambda_2(q_2, \sigma)$

2) if $q_1\prime \in Post(q_1)$ then there exists $q_2\prime \in Post(q_2)$ with $(q_1\prime, q_2\prime) \in R$

3) if $q_2\prime \in Post(q_2)$ then there exists $q_1\prime \in Post(q_1)$ with $(q_1\prime, q_2\prime) \in R$

As the transition function is dependent on the input, we define $Post(q, \sigma) = \delta(q, \sigma)$ and $Post(\sigma) = \bigcup_{\sigma \in \Sigma} Post(q, \sigma)$, which is essentially the same as for LTS brought into the notation of Section 9.2.1. There are a couple of different bisimulation types that differentiate by the handling of non-observable (internal) transitions (ordinarily labeled as τ transitions), e.g. strong and weak bisimulation, and branching bisimulation to give a few examples. This distinction is, however, theoretical in the context of this paper. The reason is that we intend to compare a specification, which consists of an automaton that does not contain any τ transitions, with an implementation that is externally (black box) learned, rendering τs unobservable. Therefore, two automata without any τs are compared directly, which makes this distinction not applicable. More precisely, from a device perspective, the type of bisimulation equivalence cannot be determined, as the SUTs are black boxes. This means that internal state changes (commonly denoted as τ) are not visible, which determines the kind of bisimulation. From a model perspective, the chosen comparison implies strong bisimulation (i.e the initial state is related (formally, $q_{0_{M_l}} = q_{0_{M_s}}$) and all subsequent states are related as well (formally $Q = Q_{M_l} = Q_{M_s}; n = |Q|; \forall n \in Q | q_{nM_l} = q_{n_{M_s}}$).

Trace equivalence, on the other hand, means that two transitions systems produce the same traces for each same input.

Definition 3 (Trace equivalence)**.** $Traces(LTS_1) = Traces(LTS_2)$

Although both bisimulation and trace equivalence might be principally capable of comparing a specification with an implementation automaton for determining the standard compliance, determining bisimulation is a problem to be solved in efficiently, whereas trace equivalence is PSPACE complete [14]. However, this might be negligible with a relatively low number of states and transitions. In any case, bisimulation implies trace equivalence ($LTS_1 \sim LTS_2$ implies $Traces(LTS_1) = Traces(LTS_2)$, but is finer than the latter [12]. For the purpose of this paper, we consider two automata equivalent if they are trace or bisimulation equivalent. In practice, we have obtained positive results with both bisimulation and trace equivalence (see Section 9.4.4). Therefore, trace equivalence is preferred as it is sufficient for standard compliance, but bisimilarity might be used in cases where more efficient checking algorithms are necessary.

9.2.3 Automata Learning

The classical method of actively learning automata of systems, was outlined in Angluin's pivotal work known as the L* algorithm [6]. This work uses a *minimally adequate Teacher* that has (theoretically) perfect knowledge of the SUT (in this case called System-under-learning – SUL) behind a *Teacher* and is allowed to answer to kinds of questions:

- *Membership queries* and
- *Equivalence queries*.

The membership queries are used to determine if a certain word is part of the accepted language of the automaton, or, in the case of Mealy machines, which output word will result of a specific input word. These words are noted in an observation table that will be made *closed* and *consistent*. The observation table consists of suffix-closed columns (E) and prefix-closed rows. The rows are intersected in short prefixes (S) and long prefixes ($S.\Sigma$). The short prefixes initially only contain the empty prefix (λ), while the long ones and the columns contain the members of the input alphabet. The table is filled with the respective outputs of prefixes concatenated with suffixes ($S.E$ or $S.\Sigma.E$). The table closed if for every long prefix row, there is a short prefix row with the same content ($\forall s.\sigma \in S.\Sigma \exists s \in S : s.\sigma = s$). The table is consistent if for any two equal short prefix rows, the long prefix rows beginning with these short prefixes are also equal ($\forall s, s\prime \in S \forall a \in \Sigma : s = s\prime \rightarrow s.a = s\prime.a$. A complete, closed and consistent table can be used to infer a state machine (set of states Q consists of all *distinct* short prefixes, the transition function is derived by following the suffixes). Even though this algorithm was initially defined for DFAs, it has been adapted to other types of state machines (e.g., Mealy or Moore machines) [15]. Alternatively, some algorithms use a discrimination tree that uses inputs as intermediate nodes, states as leaf nodes, and outputs as branch labels, with a similar method of inferring an automaton. One of these algorithms, TTT[8], is deemed currently the most efficient [16]. Other widely used algorithms include a modified version of the original L* with a counterexample handling strategy by Rivest and Schapire [7], or the tree-based Direct Hypothesis Construction (DHC) [17] and Kearns-Vazirani (KV) [18] algorithms.

Once this is performed, the resulting automaton is presented to the Teacher, which is called equivalence query. The Teacher either acknowledges the correctness of the automaton or provides a counterexample. The latter is incorporated into the observation table or discrimination tree and the learning steps described above are repeated until the model is correct. To allow for learning black box systems, the equivalence queries in practice often consist of a sufficient set of conformance tests instead of a Teacher with perfect knowledge [19]. Originally for Deterministic Finite Automata, this learning method could be used to learn Mealy Machines [20]. This preferred for learning black box reactive systems (e.g. cyber-physical systems), as modeling these as Mealy is comparatively simple.

9.2.4 LearnLib

To utilize automata learning we use a widely adopted Java library called LearnLib [21]. This library provides a variety of learning algorithms (L* and variants thereof, KV, DHC and TTT), as well as various strategies for membership and equivalence testing (e.g., conformance testing like random words, random walk, etc.). The library provides Java classes for instantiating these algorithms and interfaces systems under test. The interface classes further allow for defining the input alphabets that the algorithm routines uses to factor queries used to fill an observation table or tree. Depending on the used algorithms, the library is capable of inferring DFAs, NFAs (Non-deterministic finite acceptors), Mealy machines or VPDAs (Visibly Pushdown Automata).

9.2.5 Near Field Communication

Near Field Communication (NFC) is a standard for simple wireless communication between close coupled devices with relatively low data rates (106, 212, and 424 kbit/s). One distinctive characteristic of this standard (operating at 13.56 Mhz center frequency) is that it, based on Radio-Frequency Identification (RFID), uses passive devices (proximity cards - PICCs) that receive power from an induction field from an active device (reader or proximity coupling device PCD) that also serves as field for data transmission. There are a couple of defined procedures that allow for operating proximity cards in presence of other wireless objects in order to exchange data [22]. One standard particularly defines two handshake procedures based on cascade-based anti-collision and card selection (called type A and type B), one of which NFC proximity cards must be compliant with [1]. This handshake is the particular target system-under-learning (SUL) of this paper, with the purpose of providing very strong evidence for compliance. Due to the proliferation and the nature of the given system-under-learning, this paper concentrates on type A devices. Therefore, all statements on NFC and its handshake apply for type A only.

9.2.6 The NFC Handshake Automaton

ISO 14443-3 contains a state diagram that outlines the Type A handshake procedure for an NFC connection (see Figure 9.2). This diagram is not a state machine of the types described in Section 9.2.1, for it lacks both output and final states. As we learn Mealy machines, we augmented it with abstract outputs (see Sections 9.4.2 and 9.4.4) to get a machine of the same type. The goal of the handshake is to reach a defined state in which a higher layer protocol (e.g. as defined in ISO 14443-4 [22]) can be executed (the *PROTOCOL* state). The intended way described in the standard to reach this state is: when coming into an induction field and powering up, the passive NFC device enters the *IDLE* state. After receiving a wake-up (*WUPA*) or request (*REQA*) message it enters the *READY* state. In this state, anti-collision (*AC*, remaining in that state) or card selection (*SELECT* going to the *ACTIVE* state) occur. In the latter state, the card waits for a request to answer-to-select (*RATS*), which brings it into

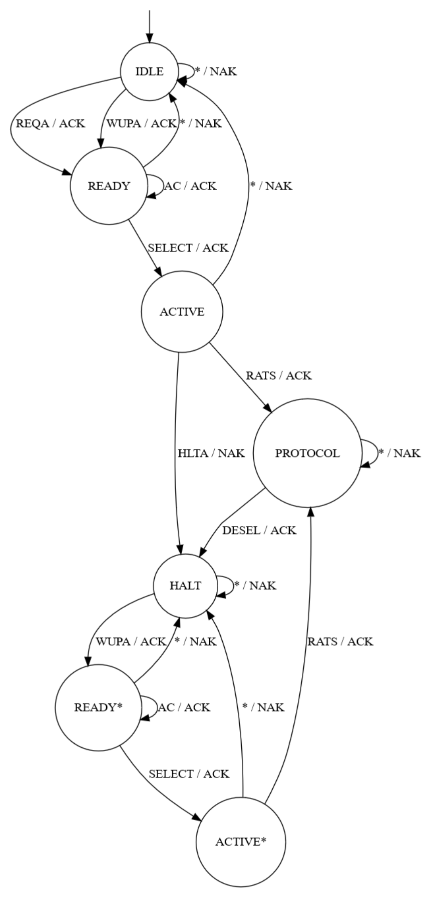


Figure 9.2: NFC handshake automaton after ISO 14443-3 [1] augmented with abstract outputs. Note: star (*) as input means any symbol that is not explicitly stated in another outbound transition of the respective state.

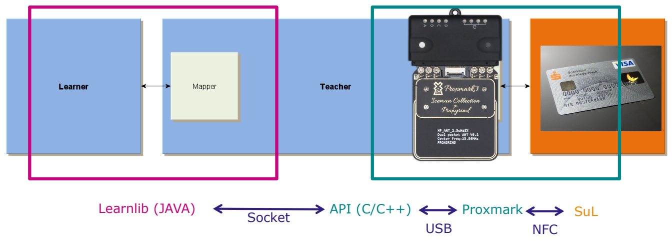


Figure 9.3: NFC interface setup.

said *PROTOCOL* state. In all of these states, an unexpected input would return the system to the *IDLE* state, no giving an answers (denoted as *NAK*). Based solely on ISO 14443-3 commands, the card should only leave this state after a *DESELECT* command, after which it enters the *HALT* state. Apart from a complete reset, it only leaves the *HALT* state after a wake-up (*WUPA*) signal (in contrast to the initial *IDLE* state, which also allows a *REQA* message). This brings it into the *READY** state, which again gets via a *SELECT* into the *ACTIVE** state that can be used to get to the *PROTOCOL* state again. The only difference between *READY* and *READY**, as well as *ACTIVE* and *ACTIVE** state is that it comes from the *HALT* instead of *IDLE* state. Similar to the first part of the automaton, an unexpected answer brings the state back to *HALT* without an answer (*NAK*).

Apart from the commands stated above that are expected by a card in the respective state, every other (i.e. unexpected) command would reset the handshake if its not complete (i.e. wrong commands from *IDLE, READY,* and *ACTIVE* states would lead back to the *IDLE* state, while *HALT, READY*,* and *ACTIVE** lead back to the *HALT* state and unexpected commands in the *PROTOCOL* state let it remain in that state. Even though this behavior of falling back into a base state resembles a combination-lock automaton or generally an accepting automaton, we model the handshake as a Mealy Machine for the following reasons:

a) As we observe a black box, input/output relations are easier to observe than not intrinsically defined accepting states

b) The states are easier distinguishable: a variety of input symbols with the corresponding output may represent a broader signature than just if a state is accepting (apart from the transition to other states)

c) The output may processed at different level of abstraction (see Section 9.4.2)

There is also one specific feature to the NFC handshake protocol: unlike most communication protocols, an unexpected or wrong input yield to no output. This has an implication to learning, as a timeout will be interpreted as a general error message.

9.3 NFC Interface

As Learner, we use the algorithm implementations in the Learnlib Java library (see Section 9.2.4), configured as outlined in Section 9.4. To interact with the NFC SUL, a Proxmark RFID/NFC device (see Section 9.3.1) is used that works with an adapter written in C++ (see Section 9.3.2). Figure 9.3 provides an overview of the setup.

9.3.1 Learner Interface Device

The interface with an NFC SUL is established via Proxmark3. Proxmark3 is a pocket-size NFC device capable of acting as an NFC reader (PCD) or tag (PICC), as well as sniffing device [23]. Proxmark3 can be controlled from a PC, as well as, allowing firmware updates. Thus it allows us to construct the NFC frames needed for learning and establishing a connection to the learning library via a software adapter (see Section 9.3.2).

9.3.2 Adapter Class

The actual access to the NFC interface runs over a C++ program, running on a PC, based on a provided application that comes with the Proxmark device. As this application is open source, it was possible to modify it in order to adapt it for learning. The main interface to the Java-based Learner is a Socket connection that take symbols from the Learner (see Section 9.4.2) and concretizes them by translating the symbols into valid NFC frames utilizing functions from the *SendCommand* and *WaitForResponse* families. These functions send and receive, respectively, command data (i.e. concrete inputs, symbol for symbol) to the Proxmark device where the firmware translates it into frames and sends them to the SUL and proceeds vice versa for the response. This, however, turned out to create an error prone bottleneck at the connection between the PC application and the Proxmark device running over USB. Due to round-trip times and timeouts, the learning was slowed down and occasional non-deterministic behavior was introduced, which jeopardized the learning process and made it necessary to repeat the latter (depending on the scrutinized system, multiple times, which hindered the overall learning greatly). Therefore, the Learner was re-implemented to send bulk inputs (i.e. send complete input words instead of single symbols), which improved the throughput significantly and solved non-determinism.

Firmware Modifications

In order to be able to transfer traces word-wise instead of symbol-wise, significant modifications of the device's firmware were necessary. The standard interface of the device is designed for sending a single packet at one time (via a provided application on a PC) and delivering the answer back to the application via a USB interface. This introduces latency, which through the sheer amount of symbols sent in the learning process, has a significant performance impact. To reach the device's firmware with multiple symbols at once, we modulate the desired inputs into one sent message in Type-Length-Value (TLV) format (implemented types are with or without CRC and a specialized type for SELECT sequences) and modify the main routine of the running firmware to execute a custom function if a certain flag is set. This custom function deserializes the sent commands and sends them to the NFC SUT. Answers are modulated into an answer packet in length-value format, followed by subsequent answer messages containing precise logging and timestamps, if used. As NFC is a protocol that works with relatively low round-trip times and time outs these modifications, eliminating a great portion of the latency times of frequently used USB connections, boost the performance of the learning using different learning algorithms significantly (for a performance evaluation see Section 9.4.1).

9.4 Learning Setup

One distinctive attribute of ISO14443-3 with respect to learning is that it specifies to not give an answer on unexpected (i.e. not according to the standards specification) input. Ordinarily, the result of such a undefined input is to drop back to a defined (specifically the IDLE or HALT) state. In this sense, the NFC handshake resembles a combination lock. A positive output on the other hand, ordinarily consists of a standardized status code or information that is needed

Max. Word	Algorithm						
Length	L*-C	L*-RS	DHC	KV-L	KV-B	TTT-L	TTT-B
10	5.92	5.05	6.00	4.38	4.38	5.45	5.37
20	20.08	9.34	10.93	12.24	11.65	7.66	7.40
30	41.90	12.92	9.82	12.19	11.47	10.67	10.04
40	68.17	8.54	11.16	15.56	12.89	10.87	9.49
50	34.75	7.87	11.02	15.60	12.53	11.29	9.91
60	77.33	17.15	12.98	17.16	13.37	13.04	10.85
70	134.65	11.34	14.46	17.68	14.81	13.06	11.32

Table 9.1: Runtime (minutes) per algorithm and maximum word length.

for the next phase of the handshake, e.g., parts of a card's unique identifier (UID). The non-answer to undefined is a characteristic feature of the NFC standard. This directly affects the learning because it yields many identical answers and efficient time-out handling is essential. It is therefore necessary to evaluate different state-of-the-art learning algorithms for their specific fitness (see Section 9.4.1) well as determining the optimal parameter set (Section 9.4.1). We scrutinize the main algorithms supported by Learnlib: classical L*, L* with Rivest/Schapire counterexample handling, DHC, KV and TTT - the latter two with linear search (L) and binary search (B) counterexample analysis.

9.4.1 Comparing Learning Algorithms and Calibrations

All of the algorithms can be parameterized regarding the membership and equivalence queries. The former are mainly defined via the minimum and maximum word length, while the equivalence queries (lack of a *perfect Teacher*), is determined by the method and number of conformance tests. Generally speaking, a too short (maximum) word length results in an incompletely learned (which, if the implementation is correct, should contain seven states). The maximum length, however, has a different impact on the performance for observation and tree-based algorithms: table-based are quicker with a short maximum word length, whereas for tree-based ones there seems to be a break-even point between many sent words and many sent symbols in our specific setting. Table 9.1 shows a comparison of the runtime of different algorithms with different maximum word lengths (in red the respective algorithm's shortest runtime that learned the correct 7-state model). Some of the non-steadiness in the results can be explained by the fact that some calibrations with shorter word lengths required more equivalence queries and, thus, refinement proce-

Algorithm	L*-C (20)	L*-RS (10)	DHC (30)	KV-L (30)	KV-B (30)	TTT-L (30)	TTT-B (40)
States	7	7	7	7	7	7	7
Runtime (min)	20.08	5.05	9.82	12.19	11.47	10.67	9.49
Words	1137	282	539	496	451	468	382
Symbols	10192	2588	5124	7932	7607	6628	6213
EQs	2	3	2	5	5	4	4

Table 9.2: Performance evaluation of different algorithms for a compliant system with their respective fastest calibration in the given setting.

dures. Table 9.2 shows the results with the best performing (correct) run of the respective algorithm. This, however, only covers the performance of learning a correct implementation. The opposite side, discovering a bug, shows a different picture. We therefore used a SUT with a slightly deviating behavior (see Section 9.5.3). This system is much more error-prone, needing significantly higher timeout values, resulting in higher overall runtimes. One key property in this case seems to be the minimum word length. Some of the algorithms by their require a lower minimum word length to discover than others. This has a significant impact with the special setting of getting relatively many timeouts, which is greatly aggravated by the necessary long timeout periods. With a minimum word length of 10 symbols, again the original L* with the Rivest/Schapire closing strategy was performing quickest, but discovered only 7 out of 10 states of the deviating implementation. DHC yielded a similar result. Both needed a word length of 20 to discover the actual non-compliant model, which was significantly less efficient in terms of runtime. The TTT and KV algorithms needed a minimum length of 10, however with quite some deviation in efficiency. While TTT was the best performing algorithm to learn the SUT's actual behavior model, KV was performing worst. The runtimes roughly correspond with the amount of sent symbols, in this case the a very long timeout has to be set to avoid non-determinism. The classical L* is not in the list, as the algorithm crashed after more than 24 hours of runtime. Table 9.3 provides an overview of minimum word lengths, run time, words, symbols and equivalence queries. Lower minimum word lengths yielded false negatives (i.e. the result showed a correct model with the deviation not uncovered).

9.4.2 Abstraction

Ordinarily, when applying automata learning to real-world systems, the input and output spaces are very large. To reduce the alphabets' cardinalities to a manageable amount, an abstraction function (∇), that transforms the concrete inputs (I) and outputs (O) to symbolic alphabets (Σ and Ω) using equivalence classes. Of all possible combinations of data to be send, we therefore concentrate on relevant input for the purpose of compliance verification. In the following we present some rationales for the chosen degree of abstraction through the input and output alphabets. These alphabets' symbols are abstracted and concretized via an according adapter class that translates symbols to data to be send (see section 9.3.2).

Input Alphabet

For the input alphabet we use the one needed for successfully establishing a handshake (cf. Figure 9.2), according to the state diagram for Type-A cards in the ISO 14443-3 standard [1]:

- Wake-UP command Type A (WUPA)
- Request command, Type A (REQA)
- Anticollision (AC)
- Select command, Type A (SELECT)
- Halt command, Type A (HLTA)
- Request for answer to select (RATS)
- Deselect (DESEL)

The last two commands are actually defined in the ISO 14443-4 standard [22]. However, as the handshake's purpose is to enter and leave the protocol state, they are included in the 14443-3 state diagram and, consequentially, in our compliance verification.

Output Alphabets

In general, the output alphabet does not need to be defined beforehand. It simply consists of all output symbols observed by the Learner in a learning run. The Learner can derive the output alphabet implicitly. This means that

Algorithm	L*-RS	DHC	KV-L	KV-B	TTT-L	TTT-B
Min Length	20	20	10	10	10	10
Runtime (min)	309.81	328.83	520.34	423.27	277.67	131.43
Words	575	855	952	679	688	616
Symbols	14637	15262	23867	19241	13353	11769
Eqs	5	3	6	6	5	5

Table 9.3: Performance evaluation of different algorithms for a non-compliant system with their respective fastest calibration in the given setting.

if a system behaved non-deterministicly, the output alphabet could vary – although when learning Mealy machines, which are deterministic by definition, nondeterminism would jeopardize the Learner. The output alphabet has obviously to be defined (in the abstraction layer) when abstracting the output. Therefore, using raw output has the benefit of not having to define the alphabet beforehand. The raw method has one drawback: there are cards that use a random UID (specifically, this behavior was observed in passports). Every anti-collision (*AC*) and *SELECT* yields a different output, which introduces non-deterministic behavior. This is not a problem with abstract output, as the concrete answer is abstracted away. We therefore tried a heavily abstracted output consisting of only two symbols, namely *ACK* for a (positive) answer and *NAK* for a timeout, which in this case means a negative answer (see Section 9.2.5). This solves the problem, but degrades the performance of the Learner, since states are harder to distinguish if the possible outputs are limited to two (aggravated by the similar behavior of certain states - see Section 9.2.6). This idea was therefore forfeit in favor of raw output for the learning. We still maintained this higher abstraction for the equivalence checking (see Section 9.4.4 for the reasoning). Raw output, however, retains this problematic nondeterminism. We therefore introduce a caching strategy to cope with this issue. Whenever a valid (partial) UID is received as an answer to an anti-collision or select input symbol, we put it on one of two caches (one for partial UIDs from *AC* and one for full ones from *SELECT* sequences). The Learner will subsequently only be confronted with the respective top entries of these caches. We therefore abstract away the randomness of the UID by replacing it with an actual but fixed one. This keeps the learning deterministic while saving the other learned UIDs for analysis, if needed.

9.4.3 Labeling and Simplification

An implementation that conforms to the standard will automatically labeled correctly, as the labelling function follows a standards-conform handshake trace:

a) label the initial state with *IDLE*,

b) from that point, find the state, where the transition with *REQA* as an input and a positive acknowledgement as an output ends and label it as *READY*,

c) from that point, find the endpoint of a positively acknowledged *SELECT* transition and label it as *ACTIVE*,

d) from that point, find the endpoint of a positively acknowledged *RATS* transition and label it as *PROTOCOL*,

e) from that point, find the endpoint of a positively acknowledged *DESELECT* transition and label it as *HALT*

f) from that point, find the endpoint of a positively acknowledged *WUPA* transition and label it as *READY**

g) from that point, find the endpoint of a positively acknowledged *SELECT* transition and label it as *ACTIVE**

If the labeling algorithm fails or there are additional states (which are out of the labeling algorithm's scope), this is an indicator for the learned implementation's non-compliance with the ISO 14443-3 standard (given that only the messages defined in that standard are used as an input alphabet - see Section 9.4.2).

To simplify the state diagram for better readability and analysis, we cluster the transitions of each states for output/target tuples and label the input for that mostly traveled tuple with a star ($*$). Normally that is the group of transitions that mark an unexpected input and transitions back to the IDLE or HALT state. This reduces the diagram significantly. Therefore, in those simplified diagrams, all inputs not marked explicitly in a state can be subsumed under the respective star ($*$) transition.

9.4.4 Compliance Evaluation

Proving or disproving compliance needs a verdict if a potential deviation from the standard violates the (weak) bisimulation relation. We use mCRL2 with the Aldebaran (.aut) format for bisimilarity and trace equivalence checking (as described in Section 9.2.2) [24]. As the Learnlib toolset provides to possibility to store the learned automata in a couple of formats, including Aldebaran, setting up the tool chain is easy, even though some re-engineering was necessary. Learnlib's standard function for exporting in the Aldebaran format does not include outputs. This accepts transitions as equal that are in fact not (as they distinguish only through the output). We therefore rewrote this function to use the transition's in the label of an LTS as well. mCRL2 comes with a model comparison tool that uses, amongst others, the algorithm of Jansen et al. [25] for bisimilarity checking. We therefore simply model the specification in form of the handshake diagram (see Figure 9.2) as an LTS with the corresponding Mealy's input and output as a label in the Aldebaran format and use the mCRL2 tool to compare it to automata of learnt implementations. The models of SUTs, although, could differ greatly event if the behavior is similar . Due to different UIDs the outputs to legit AC and SELECT commands would ordinarily differ between any two NFC cards. Also most other outputs might differ slightly. E.g., we observed some cards to respond to select with *4800*, others with *4400*. We therefore use the higher abstraction level as described above and use only NAK and ACK as output, circumventing this problem. This way, inequalities as detected by the tool indicate non-compliance to the ISO 14443-3 standard of the scrutinized implementation. If a non-compliance (i.e. a missing or additional state or transition actually countering the bisimulation relation) is found, all we need is to do a simple conformance test. A trace of the non-compliant state/transition is trivial to extract from the automaton (see the example in Section 9.5.3). If that trace is executed on the system-under-test and actually behaves like predicted in the model, we have found the actual specification violation in the real system, disproving the compliance.

Alternatively, an actual positive verdict of compliance of a learned model is simple. A full compliance proof can be made when doing identity equivalence, that is comparing the learned model state by state and transition by transition with the model manually derived from the ISO 14443-3 standard. If every state and transition is equal, we consider the system as compliant. More formally, the learned machine M_l must be fully equal the specification machine M_s, i.e. $M_l = M_s \wedge (M_l = M_s \models Q_{M_l} = Q_{M_s} \wedge \Sigma_{M_l} = \Sigma_{M_s} \wedge \Omega_{M_l} = \Omega_{M_s} \wedge \delta_{M_l} = \delta_{M_s} \wedge \lambda_{M_l} = \lambda_{M_s} \wedge q_{0_{M_l}} = q_{0_{M_s}})$. This, obviously, is a simpler but stronger relation that is not coersive for ISO protocol compliance. The probability of learning (with a sufficient amount of conformance testing) an incorrect model that is still compliant with the standard is negligible.

9.5 Evaluation

In this section we briefly outline the achieved results with the described tool chain. We used serveral different NFC card systems for testing, which are described below. All of these systems have shown to be conform to the ISO14443-3 standard, except for the Tesla key fob.

9.5.1 Test Cards and Credit Cards

We used five different NFC test test cards by NXP (part of an experimental car access system) to develop and configure the Learner. Furthermore, we used two different banking cards, a Visa and a Mastercard debit. All of these cards are conform to the standard, with only minor differences. One of these deffenrences is replying with diffent ATQA to REQA/WUPA messages with 4400 and 4800 respectively. Overall, the results with these cards are very similar. Figure 9.4 shows an example of a learnt automaton (left side).

9.5.2 Passports

We also examined two different passports from European Union countries: one German and one Austrian. The main noticeable difference (at ISO 14443-3 levlel) between the other systems is that these systems answer to AC and SELECT inputs with randomly generated (parts of) UIDs. This implements a privacy feature to make passports less traceable. Without accessing the personal data stored on the device the passport should not be attributable. This, however, requires authentication.

9.5.3 Tesla Key Fob

Apart from significantly slower answers than the other devices, which required to adapt the timeouts to avoid nondeterministic behavior, the learned automaton slightly differs when learnt with the TTT algorithm. Figure 9.4 (right side) shows a model of a Tesla car key fob learnt with TTT. The (unnamed) states 3,4 and 6 are very similiar to the HALT, READY* and ACTIVE* states, respectively. Apart from the entry points (HALTA from the ACTIVE state for the first and DESEL from the PROTOCOL state, respectively) these two structures are identical and in the reference model, those two transitions lead to the same state. However, the ACTIVE* transition allows for issuing a DESELECT

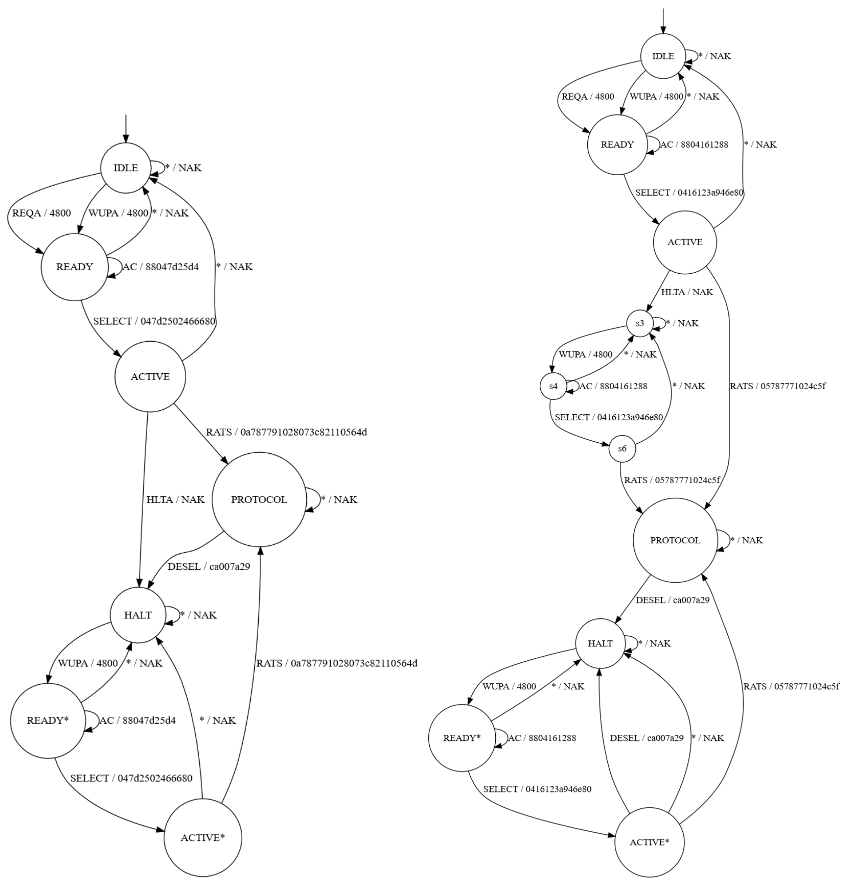


Figure 9.4: Automaton of an NXP test card (left) and a Tesla car key fob (right) learnt with TTT.

command that actually returns a value (i.e. an ACK in the higher abstraction), which does not correspond to the standard.

The mCRL2 comparison tool rightfully identifies this model not to be bisimilar and trace equivalent with the specification. Using the according option, the tool also provided a counterexample in the form of the trace ($\langle$*REQA/ACK*$\rangle$, $\langle$*SELECT/ACK*$\rangle$, $\langle$*RATS/ACK*$\rangle$, $\langle$*DESEL/ACK*$\rangle$, $\langle$*WUPA/ACK*$\rangle$, $\langle$*SELECT/ ACK*$\rangle$, $\langle$*DESEL/ACK*$\rangle$). According to the specification, the last label should be $\langle$DESEL/NAK$\rangle$.

9.6 Related Work

There are other, partly theoretic, approaches of inferring a model using automata learning and comparing it with other automata using bisimulation algorithms. However, they target DFAs [26] or probabilistic transition systems (PTS) [27]. Neider et al. [28] contains some significant theoretic fundamentals of using automata learning and bisimulation for different types of state machines, including Mealys. It also contains the important observation that (generalized) Mealy Machines are bisimilar if their underlying LTS are bisimilar. Tappler et al. [13] used a similar approach of viewing Mealy Machines as LTS to compare automata regarding their bisimilarity. Similarly, bisimulation checking was also used to verify a model inferred from an embedded control software [29]. There is also previous work on using automata learning for inferring models of NFC cards [30], which concentrates on the upper layer (ISO/IEC 14443-4) protocol, dodging the specific challenges of the handshake protocol. Also there is no mentioning of automatic compliance checking in this approach. To the best of our knowledge, there is no comprehensive approach for compliance verification of the ISO/IEC 14443-3 protocol.

9.7 Conclusion

In this paper, we demonstrated the usage of automata learning to infer models of systems under test and evaluate their compliance with the ISO 14443-3 protocol by checking their bisimilarity with a specification. We described a learning interface setup, showed practical results and made interesting observations on the impact of the protocol specifics on learning algorithms' performances.

9.7.1 Discussion

Using our learning setup on real-world devices, we found little differences between the SUTs – all examined systems were compliant to ISO/IEC 14443-3. Observed differences were mainly in the privacy-related random UIDs sent by passports and the slow answers and a slightly different automaton of the Tesla key fob. However, the scrutinized NFC handshake protocol has two characteristics that are distinct from other communications protocols: a) it does not send an answer on unexpected input and b) the automaton has two almost identical parts (IDLE/READY/ACTIVE and HALT/READY*/ACTIVE*) that pose challenges in learning. Supposedly these characteristics are responsible for the somewhat surprising finding that the L* algorithm with the Rivest/Schapire improvement surpasses more modern tree-based algorithms for correct systems. However, TTT performed best in finding a non-compliant system, which is the actual purpose of the testing and that the minimum word length has an impact on the ability to find incompliances. This might give some hints for optimization of learning strategies for similar structures.

9.7.2 Outlook

The compliance checking is but a first step towards assuring correctness and, subsequently, cybersecurity for NFC systems. Concretely, further research directions include test case generation using model checking and using the model to guide an intelligent fuzzer to leverage cybersecurity validation and verification (V&V). The target of these V&V activities are on the one hand upper layer protocols and on the other hand NFC reader devices to search for faults that might lead to exploitable security vulnerabilities. To talk to readers, because of the low latency of NFC communications, it is crucial to already know what to send before a conversation, which is satisfied by the predefined input words in the automata learning process.

Acknowledgements

This research received funding within the ECSEL Joint Undertaking (JU) under grant agreement No. 876038 (project InSecTT) and from the program “ICT of the Future” of the Austrian Research Promotion Agency (FFG) and the Austrian Ministry for Transport, Innovation and Technology under grant agreement No. 880852 (project LEARNTWINS). The JU receives support from the European Union’s Horizon 2020 research and innovation programme and Austria, Sweden, Spain, Italy, France, Portugal, Ireland, Finland, Slovenia, Poland, Netherlands, Turkey. The document reflects only the author’s view and the Commission is not responsible for any use that may be made of the information it contains.

Bibliography

[1] International Organization for Standardization, “Cards and security devices for personal identification – Contactless proximity objects – Part 3: Initialization and anticollision,” ISO/IEC Standard ”14443-3”, International Organization for Standardization, 2018.

[2] W. Issovits and M. Hutter, “Weaknesses of the ISO/IEC 14443 protocol regarding relay attacks,” in *2011 IEEE International Conference on RFID-Technologies and Applications*, pp. 335–342, Sept. 2011.

[3] J. Vila and R. J. Rodríguez, “Practical Experiences on NFC Relay Attacks with Android,” in *Radio Frequency Identification* (S. Mangard and P. Schaumont, eds.), Lecture Notes in Computer Science, (Cham), pp. 87–103, Springer International Publishing, 2015.

[4] G. Hancke, “Practical attacks on proximity identification systems,” in *2006 IEEE Symposium on Security and Privacy (S&P’06)*, pp. 6 pp.–333, May 2006.

[5] M. Maass, U. Müller, T. Schons, D. Wegemer, and M. Schulz, “NFCGate: An NFC relay application for Android,” in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec ’15, (New York, NY, USA), pp. 1–2, Association for Computing Machinery, June 2015.

[6] D. Angluin, “Learning regular sets from queries and counterexamples,” *Information and Computation*, vol. 75, pp. 87–106, Nov. 1987.

[7] R. L. Rivest and R. E. Schapire, “Inference of Finite Automata Using Homing Sequences,” *Information and Computation*, vol. 103, pp. 299–347, Apr. 1993.

[8] M. Isberner, F. Howar, and B. Steffen, "The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning," in *Runtime Verification* (B. Bonakdarpour and S. A. Smolka, eds.), Lecture Notes in Computer Science, (Cham), pp. 307–322, Springer International Publishing, 2014.

[9] G. H. Mealy, "A method for synthesizing sequential circuits," *The Bell System Technical Journal*, vol. 34, pp. 1045–1079, Sept. 1955.

[10] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, Dec. 1943.

[11] E. F. Moore, "Gedanken-Experiments on Sequential Machines," in *Automata Studies*, vol. 34 of *AM-34*, pp. 129–154, Princeton University Press, 1956.

[12] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, Apr. 2008.

[13] M. Tappler, B. K. Aichernig, and R. Bloem, "Model-Based Testing IoT Communication via Active Automata Learning," in *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pp. 276–287, Mar. 2017.

[14] L. Aceto, A. Ingolfsdottir, and J. Srba, "The algorithmics of bisimilarity," in *Advanced Topics in Bisimulation and Coinduction*, pp. 100–172, Cambridge University Press, 2011.

[15] B. Jacobs and A. Silva, "Automata Learning: A Categorical Perspective," in *Horizons of the Mind. A Tribute to Prakash Panangaden: Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday* (F. van Breugel, E. Kashefi, C. Palamidessi, and J. Rutten, eds.), Lecture Notes in Computer Science, pp. 384–406, Cham: Springer International Publishing, 2014.

[16] F. Vaandrager, "Model learning," *Communications of the ACM*, vol. 60, pp. 86–95, Jan. 2017.

[17] M. Merten, F. Howar, B. Steffen, and T. Margaria, "Automata Learning with On-the-Fly Direct Hypothesis Construction," in *Leveraging Applications of Formal Methods, Verification, and Validation* (R. Hähnle,

J. Knoop, T. Margaria, D. Schreiner, and B. Steffen, eds.), vol. 336, pp. 248–260, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[18] M. J. Kearns and U. Vazirani, *An Introduction to Computational Learning Theory*. MIT Press, Aug. 1994.

[19] D. Peled, M. Y. Vardi, and M. Yannakakis, "Black Box Checking," in *Formal Methods for Protocol Engineering and Distributed Systems: FORTE XII / PSTV XIX'99 IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX) October 5–8, 1999, Beijing, China* (J. Wu, S. T. Chanson, and Q. Gao, eds.), IFIP Advances in Information and Communication Technology, pp. 225–240, Boston, MA: Springer US, 1999.

[20] M. Shahbaz and R. Groz, "Inferring Mealy Machines," in *FM 2009: Formal Methods* (A. Cavalcanti and D. R. Dams, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 207–222, Springer, 2009.

[21] M. Isberner, F. Howar, and B. Steffen, "The Open-Source LearnLib," in *Computer Aided Verification* (D. Kroening and C. S. Păsăreanu, eds.), Lecture Notes in Computer Science, (Cham), pp. 487–495, Springer International Publishing, 2015.

[22] International Organization for Standardization, "Cards and security devices for personal identification – Contactless proximity objects – Part 4: Transmission protocol," ISO/IEC Standard "14443-4", International Organization for Standardization, 2018.

[23] F. D. Garcia, G. de Koning Gans, and R. Verdult, "Tutorial: Proxmark, the swiss army knife for rfid security research: Tutorial at 8th workshop on rfid security and privacy (rfidsec 2012)," tech. rep., Radboud University Nijmegen, ICIS, Nijmegen, 2012.

[24] O. Bunte, J. F. Groote, J. J. A. Keiren, M. Laveaux, T. Neele, E. P. de Vink, W. Wesselink, A. Wijs, and T. A. C. Willemse, "The mCRL2 Toolset for Analysing Concurrent Systems," in *Tools and Algorithms for the Construction and Analysis of Systems* (T. Vojnar and L. Zhang, eds.), Lecture Notes in Computer Science, (Cham), pp. 21–39, Springer International Publishing, 2019.

[25] D. N. Jansen, J. F. Groote, J. J. A. Keiren, and A. Wijs, "An O(m log n) algorithm for branching bisimilarity on labelled transition systems," in *Tools and Algorithms for the Construction and Analysis of Systems* (A. Biere and D. Parker, eds.), Lecture Notes in Computer Science, (Cham), pp. 3–20, Springer International Publishing, 2020.

[26] Y.-F. Chen, C.-D. Hong, A. W. Lin, and P. Rümmer, "Learning to prove safety over parameterised concurrent systems," in *2017 Formal Methods in Computer Aided Design (FMCAD)*, pp. 76–83, Oct. 2017.

[27] C.-D. Hong, A. W. Lin, R. Majumdar, and P. Rümmer, "Probabilistic Bisimulation for Parameterized Systems," in *Computer Aided Verification* (I. Dillig and S. Tasiran, eds.), Lecture Notes in Computer Science, (Cham), pp. 455–474, Springer International Publishing, 2019.

[28] D. Neider, R. Smetsers, F. Vaandrager, and H. Kuppens, "Benchmarks for Automata Learning and Conformance Testing," in *Models, Mindsets, Meta: The What, the How, and the Why Not? Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday* (T. Margaria, S. Graf, and K. G. Larsen, eds.), Lecture Notes in Computer Science, pp. 390–416, Cham: Springer International Publishing, 2019.

[29] W. Smeenk, J. Moerman, F. Vaandrager, and D. N. Jansen, "Applying automata learning to embedded control software," in *Formal Methods and Software Engineering* (M. Butler, S. Conchon, and F. Zaïdi, eds.), (Cham), pp. 67–83, Springer International Publishing, 2015.

[30] F. Aarts, J. De Ruiter, and E. Poll, "Formal Models of Bank Cards for Free," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pp. 461–468, Mar. 2013.

Chapter 10

Paper IV: From TARA to Test: Automated Automotive Cybersecurity Test Generation Out of Threat Modeling

Stefan Marksteiner, Christoph Schmittner, Korbinian Christl, Dejan Ničković, Mikael Sjödin, and Marjan Sirjani. in Proceedings of the 7th ACM Computer Science in Cars Symposium, CSCS'23. New York: Association for Computing Machinery, 2023. DOI: 10.1145/3631204.3631864.

Abstract

The United Nations Economic Commission for Europe (UNECE) demands the management of cyber security risks in vehicle design and that the effectiveness of these measures is verified by testing. Generally, with rising complexity and openness of systems via software-defined vehicles, verification through testing becomes a very important for security assurance. This mandates the introduction of industrial-grade cybersecurity testing in automotive development processes. Currently, the automotive cybersecurity testing procedures are not specified or automated enough to be able to deliver tests in the amount and thoroughness needed to keep up with that regulation, let alone doing so in a cost-efficient manner. This paper presents a methodology to automatically generate technology-agnostic test scenarios from the results of threat analysis and risk assessment (TARA) process. Our approach is to transfer the resulting threat models into attack trees and label their edges using actions from a domain-specific language (DSL) for attack descriptions. This results in a labelled transitions system (LTS), in which every labelled path intrinsically forms a test scenario. In addition, we include the concept of Cybersecurity Assurance Levels (CALs) and Targeted Attack Feasibility (TAF) into testing by assigning them as costs to the attack path. This abstract test scenario can be compiled into a concrete test case by augmenting it with implementation details. Therefore, the efficacy of the measures taken because of the TARA can be verified and documented. As TARA is a de-facto mandatory step in the UNECE regulation and the relevant ISO standard, automatic test generation (also mandatory) out of it could mean a significant improvement in efficiency, as two steps could be done at once.

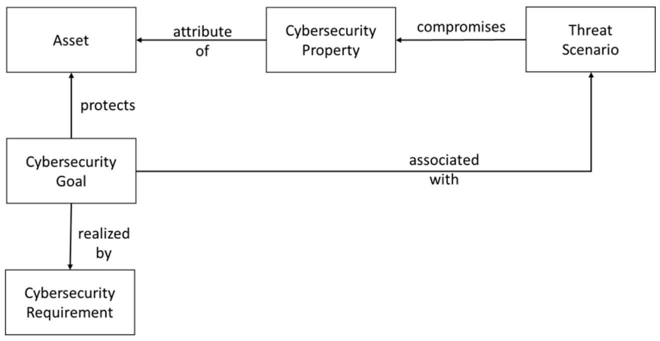


Figure 10.1: Relationship for risk mitigation.

10.1 Introduction

The market introduction of vehicle-to-x (V2X) functions and advanced driving assistance systems (ADAS) to automotive systems make them increasingly complex. At the same time, cybersecurity incidents (increasingly induced by criminals) display an exponential growth [1]. This is being recognized by standards and regulation bodies. For example, the United Nations Economic Commission for Europe (UNECE) issued a regulation (R155) that demands cybersecurity concerns to be addressed over the complete life cycle and verify the measures through testing [2]. Therefore, a holistic approach for cybersecurity engineering and testing over the complete life cycle is needed. This paper presents the confluence of a life cycle governance and a structured semi-automated testing approach to provide fast, comprehensive and cost-efficient cybersecurity testing over the complete automotive life cycle in conjunction with the concepts of Cybersecurity Assurance Levels (CALs) and Targeted Attack Feasibility (TAF). Section 10.2 describes the latter concepts and their integration in a security testing process. Section 10.3 elaborates automating the process of generating suitable threat models and attack trees. Section 10.4 describes the transfer mechanism from attack trees to agnostic test cases and their application to an actual implementation. Section 10.5 describes the application of the process in a small case study. Section 10.6 gives an overview of different work in this direction and Section 10.7, eventually, concludes the paper.

10.1.1 Motivation

As current standards (most prominently ISO/SAE 21434 and UNECE R155) lack the details of how to test, there are two initiatives ongoing in ISO's standardization: ISO/SAE PAS 8475 (WIP)[1] [3] that copes with Cybersecurity Assurance Levels (CALs) and Targeted Attack Feasibility (TAF) and ISO/SAE PAS 8477 (WIP) [4] that deals with verification and validation (V&V) methods. In order to include these concepts-in-development into security processes, giving clarity to Original Equipment Manufacturers (OEMs) and suppliers, this paper aims for giving suggestions how to align security testing on CALs and TAFs originating from the earliest stages of the (security) engineering process. Furthermore, the aim is to turn the overhead necessary for formalizing the combined engineering and testing process into an advantage by automatizing them. More specifically, these formalized processes can be used to automate test case generation from threat models. As a result, test case blueprints can be generated during the modeling process, that can be later on (semi-)automatically compiled into executable test cases. This allows for structured and efficient testing of the fulfillment of the requirements stemming from the threat analysis.

10.1.2 Contribution

This paper contributes mainly four things to the body of knowledge:

1. A structural concept how to incorporate CALs and TAFs into the cybersecurity engineering process.
2. A process to align testing on CALs and TAFs.
3. A method to generate attack trees from TARA.
4. A concept to transform attack trees into technology-agnostic test scenarios automatically as a blueprint to verify and validate security claims and requirements.

Item 1 explains the upcoming developments of ISO/SAE 8475 and describes the usage of CALs and TAFs (Section 10.2.1). Item 2 discusses the merit of the CAL/TAF usage in security testing (Section 10.2.1). Item 3 shows an approach how the formalization necessary to include the first two items can be used to increase the efficiency of testing by generating attack trees from a Threat

[1] Work-in-Progress

Analysis and Risk Assessment (TARA) (Section 10.3.1). Item 4 provides a method to transform attack trees into abstract test scenarios by labelling the edges with actions from the alphabet of a domain-specific language (DSL) for attack descriptions (see Section 10.4.2).

10.2 Automotive Security Communication

Effective communication plays a pivotal role in the automotive industry, particularly within the complex network of Original Equipment Manufacturers (OEMs) and Tier 1 and 2 suppliers. Especially in the cybersecurity domain, with interlocking layers of defense [5] the criticality of clearly communicating expected requirements, is required for achieving optimal outcomes. By fostering a shared understanding of risk mitigation strategies, OEMs and suppliers can collaboratively address cybersecurity challenges, enhance product security, and streamline operations. ISO/SAE 21434 defines here a framework in which during the Threat Assessment and Risk Analysis cybersecurity goals are defined. A cybersecurity goal is aimed at reducing the risk of threat scenarios and realized by cybersecurity requirements (see Figure 10.1). This process can be applied during all phases of the development, at item (see Section 10.2.3), system, or component level. Cybersecurity goals can be defined by the OEM and by the supplier. Cybersecurity requirements are assigned to components and implemented.

10.2.1 Cybersecurity Assurance Level (CAL)

An important aspect is here on the interplay between customer requirements and regulatory needs. As mentioned in the introduction, UNECE requires in the new UN R155 [2] that cybersecurity in a vehicle has to be tested and demonstrated during the type approval. With the complexity of modern vehicles, this testing effort needs to be distributed through the supply chain. ISO/SAE 21434 already establishes as an informative part the concept of the Cybersecurity Assurance Level (CAL). Inspired from assurance level schemes like the Common Critiera Evaluation Assurance Levels (EALs) [6], the goal of CAL is to describe the expected level of assurance and rigor for a defined cybersecurity goal. ISO/SAE 21434 defines an informative framework regarding the mapping of CAL to the impact and the attack vector. In addition, for concept and product development potential aspects that can be adjusted by CAL like testing effort or independence are given. CAL is assigned per cybersecurity goal and derived requirements inherit the CAL. If a requirement addresses multiple cybersecurity goals, the highest CAL is inherited.

10.2.2 Target Attack Feasibility (TAF)

In practical applications of CAL and ISO/SAE 21434, there has been a noticeable lack of clarity regarding the expected strength of security controls. This ambiguity becomes particularly evident when suppliers attempt to translate high-level security goals and requirements into technical specifications and implementations. While CAL provides insights into the engineering rigor, it falls short in communicating their actual strength. To address this gap, the concept of Target Attack Feasibility (TAF) has been introduced. TAF is designed to be associated with specific security controls, offering a measure of their expected strength. For instance, a security goal such as ”protect the integrity of the message” could be interpreted through various security controls based on their TAF levels:

- TAF1: cryptographic hash
- TAF2: symmetric encryption
- TAF3: asymmetric encryption

However, the temporal relevance of TAF is still a topic of debate. As more TAF levels are designated to specific security control technologies, there's an increasing risk that these assignments might become obsolete over time. One potential solution is to map TAF levels to Attack Feasibility, where, for example, TAF1 would necessitate a specific level of expertise, equipment, and time to breach. This approach, in contrast to a fixed technological assignment, offers a more flexible interpretation, though it also introduces a degree of subjectivity.

10.2.3 Integrating CAL and TAF in security testing

Due to the impact of CAL and TAF on the overall process and especially on the cybersecurity testing, a well-structured process is necessary. We adapt here a testing process, presented in [7] and adapted to include CAL and TAF. The process is aligned with ISO/SAE 21434 [8]. The activities are basically sequential, although some activities provide input for more than one subsequent activity. Figure 10.2 provides an overview.

I Item Definition

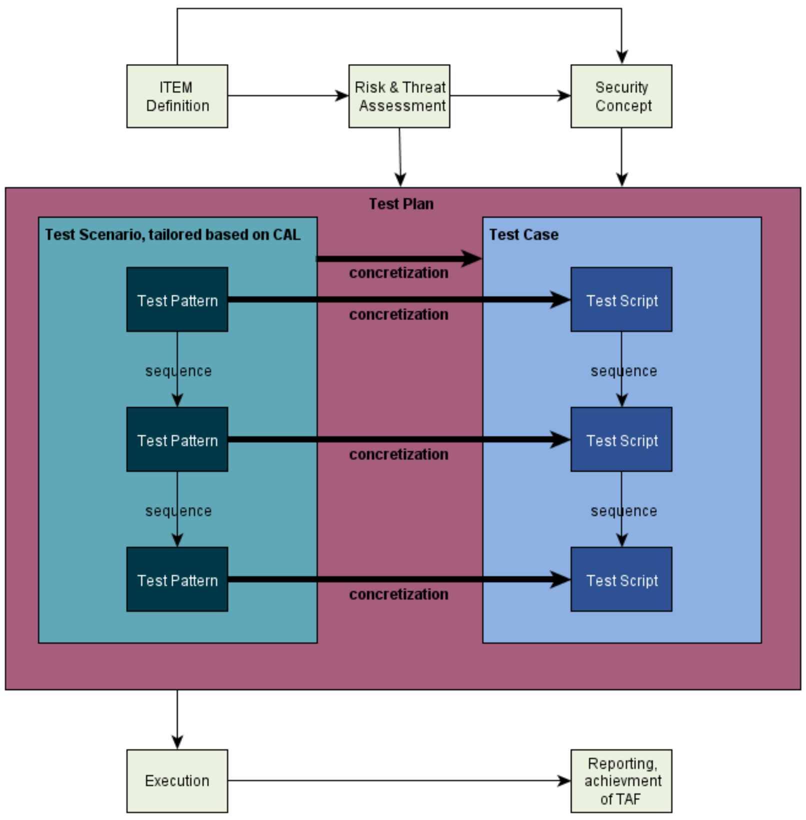


Figure 10.2: Layout of the security testing process from [7].

II Risk and Threat Assessment

III Security Concept Definition (including the test targets)

IV Test Planning and Scenario Development

(a) Penetration Test Scenario Development

(b) Functional and Interface Test Development

(c) Fuzz Testing Scenario Development

    (d) Vulnerability Scanning Scenario Development

V Test Script Development

VI Test Script Validation

VII Test Case Generation

    (a) Test Environment Preparation

VIII Test Case Execution

IX Test Reporting

In the item definition (i), the scope of the development is defined. This can range from a complete car model to specific systems or combination of systems. Risk and threat assessment (ii) (e.g., TARA [9, 10]) identifies potential vulnerabilities to be addressed and prioritizes them, focusing on certain threats that are deemed graver, while neglecting others. Here CAL and TAF are assigned for each Cybersecurity goal. The security concept definition (iii) mainly aims at anticipating measures to counter the threats from the previous activity. Measures that should be present and effective to counter specific threats that should be validated in the course of this process. TAF plays a major role in the selection of suitable security measures, that achieve a sufficient level of risk reduction. The test planning and scenario development (iv) derives an abstract test plan, consisting of scenarios, based on the security targets from the previous activity. The test plan should contain an overall test strategy. Tests are based on threats and focus on risky areas, denoted by an increased CAL. Test data inputs are selected based on threats from the risk analysis [11] and match test patterns which represent abstract (symbolic) actions in a distinct sequence. The scenarios are categorized into four classes [8]: penetration testing, functional and interface test, fuzz testing and vulnerability scanning. Although derived from the analysis of a test item, the scenario description is used to be generic: no specific information of an item on a lower technical level should be incorporated for portability reasons. Sensibly, descriptions could be composed in a domain specific language (DSL) for attack descriptions [12, 13, 14, 15]. Selection of scenarios and also independence of persons who test the SUT are based on CAL. The test script development (v) turns the test patterns from the scenarios into executable scripts. It should develop a script to match a test pattern by either using an existing exploit from an available database or develop an own attack on the system. This means that the pattern must be equipped

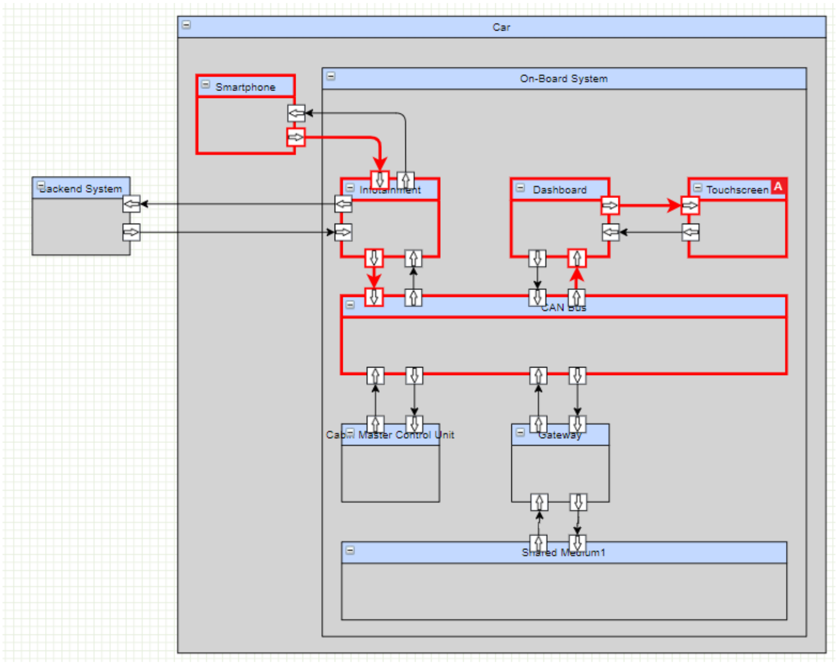


Figure 10.3: Example Threat Model.

with specific information and brought in a form that it is executable on a testing system, e.g., on a Linux shell. The test case generation (vi) assembles the test scripts to a consistent test case (a full attack on an SUT) by processing a DSL-based description (the generic test scenario) and using additional information from an SUT database, as well as using combinatorial methods to economically increase the test coverage [16]. Lastly, the tests have to be executed (vii) and their result reported (viii). These activities also include proper feedback from the test. If the process is to be automated, proper information for an autonomous test oracle has to be provided in the form of pre and post conditions that have to be fulfilled in order to assess a positive or negative (or even inconclusive) test result. Here the achievement of the intended TAF has to be included.

10.3 Threat Modeling

In this Section, we present an approach that generates test scenarios in a technology-agnostic manner out of a threat model. In the context of this paper, we conceptualize threat modeling as an iterative process used to identify and analyze potential threats in information technology (IT) systems. This iterative process basically requires two major components as inputs [17]. The first component is a threat model that summarizes the accumulated knowledge of known and documented threats, vulnerabilities, and weaknesses for the domain under study, such as automotive and IoT . It serves as a comprehensive repository of potential threats that could compromise the system. The second component is a systematic and abstract representation of the system under consideration. This representation contains all the key information required for a thorough threat analysis. Our approach uses an adapted version of the internal SysML block diagram that facilitates the representation of the relationships and properties of the system components, providing the basis for a comprehensive analysis.

The modeling process itself is the comparative analysis between the threat model and the system model. This critical comparison helps derive a list of existing threats, which is the completion of one cycle of the process. This list is expanded by recognizing the intrinsic interdependencies of the identified threats, which overcomes the limitation of looking at threats in isolation [18]. By leveraging the data revealed by the identified threats, we can explore the intricacies of their interdependencies. It is worth noting that threats rarely occur in a vacuum; they primarily build on previous steps and can trigger subsequent events.

To map these interdependencies, we use the concept of pre- and post-conditions. With this strategy, we can not only detect these dependencies, but also visually represent this additional information using attack graphs and attack trees to improve the understanding and analysis of potential threat interactions. In Section 3, we elaborate on the intricacies of this enhanced process, detailing the concept of threat interdependencies and the resulting strengthened approach to threat modeling.

Figure 10.3 shows an example of a threat model based on [19]. In this example, the electrical/electronic (E/E) architecture of an autonomous low-speed shuttle is presented. This architecture was modelled in ThreatGet, a tool for threat modelling and analysis, to facilitate automated security analysis and demonstrate the process from TARA to CAL and TAF.

We denote here one of many potential assets, with is the integrity of the Master Controller. If an attacker would be able to modify the firmware, he

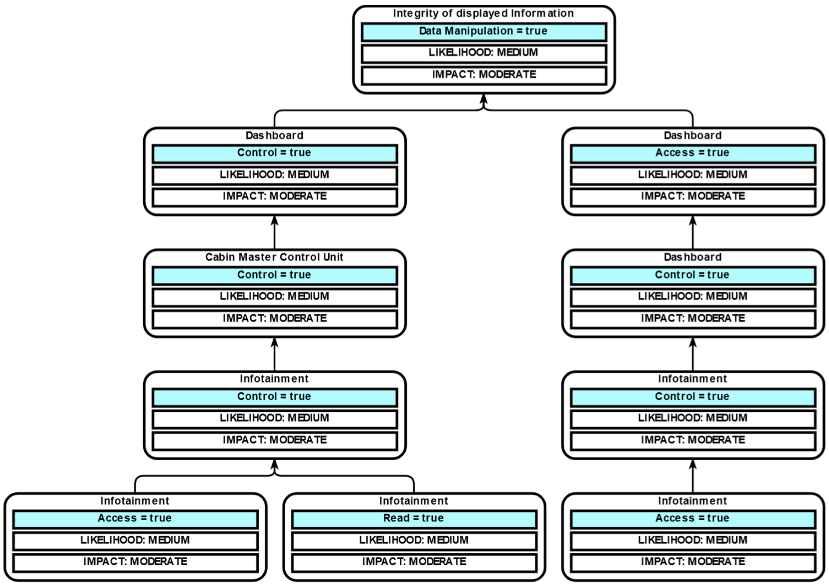


Figure 10.4: Example attack tree.

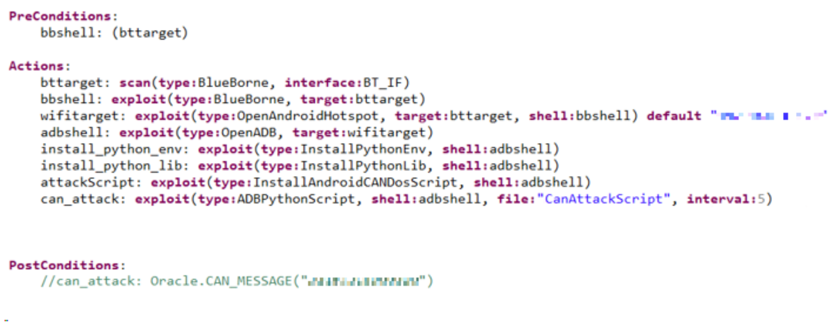
could send any command and cause potential safety and operational issues (due to the low speed of the vehicle)

- Asset: Firmware of the Master Controller (Integrity)

- Damage Scenario: Unintended steering causing collision with an obstacle ASIL C

An analysis shows a potential attack starting from an unencrypted wireless connection between external services and the AI & Drive Algorithm (see Figure 10.4). This allows an attacker to reach the dashboard and manipulate data on this element (=¿ violating the integrity of the displayed information).

In order to address this a security goal is defined, which states that the master controller has to be protected and this security goal gets a CAL assigned, based on the potential impact (CAL 3). This security goal is then mapped to a security requirement, that encryption with at least TAF 3 is added to external connections. TAF 3 could be mapped to asymmetric encryption.



```
PreConditions:
    bbshell: (bttarget)

Actions:
    bttarget: scan(type:BlueBorne, interface:BT_IF)
    bbshell: exploit(type:BlueBorne, target:bttarget)
    wifitarget: exploit(type:OpenAndroidHotspot, target:bttarget, shell:bbshell) default "[redacted]'
    adbshell: exploit(type:OpenADB, target:wifitarget)
    install_python_env: exploit(type:InstallPythonEnv, shell:adbshell)
    install_python_lib: exploit(type:InstallPythonLib, shell:adbshell)
    attackScript: exploit(type:InstallAndroidCANDosScript, shell:adbshell)
    can_attack: exploit(type:ADBPythonScript, shell:adbshell, file:"CanAttackScript", interval:5)


PostConditions:
    //can_attack: Oracle.CAN_MESSAGE("[redacted]")
```

Figure 10.5: Example for a test scenario in the used attack description language.

10.3.1 Threat-Interdependencies and Attack Trees

The Threat Analysis and Risk Assessment (TARA) process aims to identify potential threats and assess the associated risks to ensure effective risk mitigation [9, 19]. It involves systematically investigating threats, assessing their likelihood and impact, and developing strategies to address the identified risks [20]. The first step of the TARA process is to analyse for potential threats. This step is essential because only what has been identified can be assessed later. It involves identifying vulnerabilities, weaknesses or potential attack vectors [19, 21]. It is not advisable to look at threats solely in isolation as part of the TARA process, as this approach ignores the interactions between different threats. Threats often interact with or reinforce each other, resulting in attack chains or paths. Failure to consider these interactions can result in missing relevant risks and inadequate prioritization of resources for effective risk mitigation [22]. The concept of pre- and post-conditions for threats can be used to represent the interdependencies of threats within the TARA process. Preconditions represent the necessary circumstances or events that must be met for a threat to occur, while postconditions represent the possible consequences or outcomes that result from the occurrence of a particular threat. It should also be emphasized that the postconditions of some threats may be the preconditions of others. By identifying and analysing these preconditions and postconditions, we can better understand how threats are connected and how they propagate or influence each other [18]. In an attack tree [23, 24], the hierarchical structure illustrates the connections between threats, their relationships, and the different attack scenarios. The root of an attack tree is usually connected to the attack target, which is the overall goal of an attacker. From this attack target, a security objective can be derived, which represents the desired outcome of the attack defence. By visualizing threats in an attack tree, we can analyse the preconditions and postconditions associated with each threat. Considering the interdependencies of threats within the attack tree not only simplifies, but also improves, the assessment of target attack feasibility [25]. By visualizing the connections and dependencies between different threats, it becomes easier to analyse the feasibility of attacking a particular target. Understanding how multiple threats contribute to a given postcondition provides a more comprehensive view of the potential attack surface and the likelihood of a successful attack [21]. Considering the inter-dependencies within the attack tree improves understanding of the overall risk landscape and facilitates more informed decision-making regarding resource allocation, security control implementation, and mitigation prioritization. This approach improves the accuracy and effectiveness of target attack feasibility assessments and results in more robust and proactive security measures. In addition, consideration of dependencies enables organizations to effectively prioritize remediation efforts. By identifying critical paths and dependencies within the attack tree, resources can be strategically allocated to protect the most vulnerable areas. While the CAL can be easily derived based on the impact, the TAF can focus on elements in the tree which have the highest contribution to the Attack Feasibility. In summary, considering interdependencies in the TARA process and attack tree not only simplifies but also improves threat assessment and increases overall cybersecurity. By understanding the interrelationships and dependencies, organizations can effectively identify, prioritize, and mitigate risks, resulting in higher CAL and greater confidence in the security of their systems.

10.4 Automated Testing

This section is concerned with the automated generation of security test cases stemming from a TARA using attack trees (see Section 10.3.1). The principal idea is to use the resulting attack tree and create blueprints for testing in the form of implementation-agnostic test scenarios, through mapping rule sets. These agnostic test scenarios can later be concretized and executed on a specific system implementation.

10.4.1 Security Tests and their relationship with the Security Analysis

Following the method in Section 10.2.3, we store blueprints for test cases in a system-agnostic manner in the ALIA DSL [12] as test scenarios (see Figure 10.5 for an example). These test scenarios are an abstract representation of actions to be taken to execute a test case. The actions are accompanied by preconditions that determine if an action is to be carried out (i.e., is the step sensible in the current situation). Postconditions determine the expected result and contain therefore information for a test oracle. The respective steps in the scenarios (test patterns) use symbolic instructions. Concrete test cases are compiled by augmenting the scenarios with concrete information about the system-under-test (e.g., exploit code, or specific messages on the CAN bus that would yield an expected result). This scenario can be seen as a recipe for an attack with the concrete information as ingredients. The result is a concretely executable set of instructions (in JSON format) to be ran on a Linux-based attack system. To generate tests that would subsequently provide evidence for the successful satisfaction of the requirements derived from the TARA, taking CAL and TAF into account, we propose a flow that uses the attack tree analysis' results and transforms it into attack scenarios that can be augmented with concrete implementation details in later phases of the development.

10.4.2 Security Test Generation

Using an attack graph (such as a tree, but also other structures like petri nets [26] are thinkable) allows for closing the loop from TARA to testing through an automated process. The missing link to achieve this pervasive chain is a transform mechanism from paths in the generated attack graph structure to test scenarios in the DSL. We therefore propose a mechanism that transforms a specific path in an attack tree (see Section 10.3.1) into a test scenario. This is achieved by mapping the edges of that path with actions in a DSL-based test scenario. The basic idea is that an action is required to realize a threat. Therefore, traversing trough a path in an attack tree requires a set of actions, each action responsible getting from one node in the tree to another. As the test patterns in the DSL principally consist of such (abstract) actions (accompanied by optional sets of pre- and postconditions), the resolution is a rule-based translation function to simply map the tree edges to test patterns. Figure 10.6 gives an overview of this process. Formally, the attack tree can be seen as a directed graph with rules (sequencing and parallelization). This resembles a *Transition*

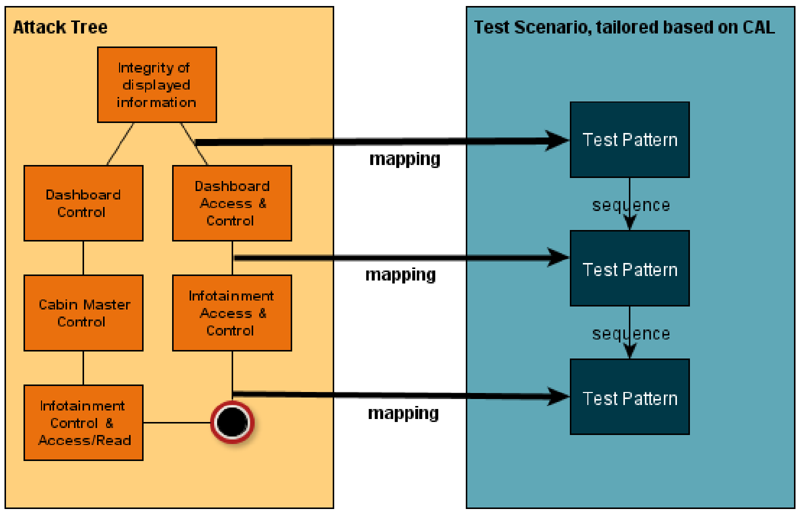

Attack Tree
Integrity of displayed information
Dashboard Control
Dashboard Access & Control
Cabin Master Control
Infotainment Access & Control
Infotainment Control & Access/Read
mapping
mapping
mapping
Test Scenario, tailored based on CAL
Test Pattern
sequence
Test Pattern
sequence
Test Pattern

Figure 10.6: Attack tree to Test Scenario transformation example.

System (TS), defined as a set of states (Q) and a transition relation ($\rightarrow \in Q \times Q$, with $q, q\prime \in Q; q \rightarrow q\prime$). In this case, Q is the set of nodes in the attack graph, while $\rightarrow$ is determined by the edges and rules in the tree. A *Labelled Transition System (LTS)* additionally possesses a set of labels (Σ), such that each transition is named with a label σ in Σ ($q, q\prime \in Q, \sigma \in \Sigma; q \xrightarrow{\sigma} q\prime$) [27]. The set of labels is taken from the set of test patterns (i.e., possible actions) in the DSL. A labeling function attributes a label σ to a transition using an associative array. Once this LTS has been established, generating the abstract test case is trivially conducted by traversing along the respective path in the LTS an collecting the labels. The sequential set of collected labels (i.e., test patterns) automatically constitutes a test scenario. In simple words, we use an attack tree to select actions needed for an test scenario out of the set of all available test patterns and brings them into sequence. The way the DSL is currently structured, an action can be identified by the tuple keyword (currently one of *scan*, *exploit*, and *execute* - the first two are to detect and attack devices, while the latter is a generic keyword for auxiliary tasks) and type (which defines the action closer). There are other attributes like *interface*, *target*, and *shell*, that depend on the action type. More than one action can be necessary to change the state in an attack tree (i.e. to traverse from one node to another). In this case the label attributed to the transition contains both actions. As an abstract example, the transition from *access to a system* to *control of a system* could require *execute, escalate privilege* as an action from the DSL. Therefore the resulting transition in the LTS would be $A_s \xrightarrow{xc_{ep}} C_s$ with A_s is the system access, C_s system control and xc_pe the execute (xc) privilege escalation (pe). A more concrete example follows in the case study in Section 10.5.

The course of action to use the TARA results for test cases also allows for prioritizing test cases, as attack paths can have calculated path costs (based on CALs and TAFs). As perfect security is infeasible, a *sufficiently* secure system can be defined as a system that does not exhibit an attack path with a cost below a certain threshold. Through the test case generation, it can be verified that relevant attack paths discovered through the threat modeling are mitigated through the measures in the security concept and effectively blocked in the implementation later.

The reason for using an LTS as a transition model is that it can be regarded as a more powerful structure than a tree (a tree can be viewed as a subset of an LTS in this regard) and can be easily converted into other structures like Directed Acyclic Graphs or even a general directed graph (in case of allowed loops needed), which makes it suitable as an internal structure. It can also be practically used in a three-layered process in this application. First, the attribution between tree edges and DSL actions (i.e., the labelling function) must be established only once initially and if the base set of node types in the TARA process or the possible actions in the DSL change (this happens rarely). Second, the LTS generation (low effort if the labelling function is present) must be done once, when an attack tree is generated or updated. Thirdly, the test cases have to be generated based on selection of paths, defining an origin (i.e. an entry point into the system) and a target is trivial, just collecting labels.

10.5 Case Study

To practically demonstrate the approach, we give an example of a realistic use case scenario. This use case has been practically tested using our test system. It consists of a standard car model that possesses a single can bus with an after-market infotainment system, running on Android, built in. The conducted test was to manipulate the speed gauge using a wireless access an entry point. We first created an architecture model using ThreatGet. The critical components for the attack are the infotainment system, the CAN bus the attacked dash board (including a screen) and a smart phone that is under the attacker's control (i.e., it is the attacker's smartphone) – see Figure 10.3. The threat analysis using the tool yielded a list of 103 threats (using the STRIDE methodology [28]). Using the methodology in Section 10.4.2, we generate attack trees and respective paths using different origin and destination points in the architecture diagrams and the threat attributions along the way. One specific result of this process is the attack tree in Figure 10.4. In this sequence, access to infotainment is followed by control of the infotainment, which is succeed by control (implying access) to the dashboard. This enables to corrupt the integrity of displayed information. This in practice means e.g., fake readings on the speed and RPM gauges or similar things - including potential safety implications – Table 10.1 provides an overview of threats applicable to the display. Please note that those apply directly to the display, while the attack tree allows for applying threats indirectly not requiring direct access to the system. The key element is the CAN bus, any device (also the cabin master control unit) connected to the (right) CAN bus (cf. Figure 10.3) that is taken control of could be used to gain access to the dashboard and manipulate the display under certain circumstances modeled in the threat model and attack tree. The transitions between these items have been matched with fitting action items from the DSL. To reach access to the Infotainment from an initial state in the LTS, a wireless scan and already an exploit (labels $s_{BlueBorne}$ and $xp_{BlueBorne}$ for scanning and executing a BlueBorne attack, with s for a scan and xp for an exploit) has to be take as actions. Please note that this is one of more possibilities to gain access, there could be others. To gain control, we use the actions of opening a connection to a remote hotspot using the access ($xp_{OpenAndroidHotspot}$) and opening an Android Debug (ADB) shell ($xp_{OpenADB}$). The rest of the tree is a special case, as the access to the Dashboard, its control and the data manipulation can occur in one step by sending fake CAN messages. These messages are represented by the different step *can_attack* in the DSL ($xp_{CanAttack}$). Figure 10.5 shows the resulting attack description in the DSL. The steps immediately preceding the CAN attack (*install_python_env*, *install_python_lib*, and *attackScript*) are intrinsic, as these are just necessary steps to fulfill the last one. In that sense, they can also be seen as part of gaining control over the infotainment, as it is only after these three steps capable of carrying out the rest of the attack. The concretization for a specific system eventually works by generating a JSON code that contains executed environments and exploit code, as well as information as CAN packet structures from a database or directly given information from the tester as form of a grey-box test. This is out-of-scope of this paper and already published elsewhere [12] in detail, but for the sake of the functioning of the approach it should be briefly mentioned that the DSL items (i.e. *Test Patterns*) are augmented with information from a systems database containing information about the systems-under-test (partially pre-filled and completed by a client in a grey box setting or penetration testers in a black box setting) with the necessary information (e.g., pieces of code to exploit a certain software or version, specific data of CAN messages to send, etc.). This is translated into a JSON format containing an environment (e.g., BASH, Python, a framework like Metasploit, etc.) and sent to an execution engine that is instrumented with the SUT and calls the respective software tools tools to execute the concrete attack.

10.6 Related Work

Threat modeling is an approach that responds to the increasing need to address security concerns from the early phases of product development. The popularity of threat modelling is reflected by a variety of available methods and tools, ranging from open-source academic prototypes to full-fledged commercial solutions. There are roughly speaking three categories of threat modeling approaches. The first class of tools only allow manual modeling based on Excel sheets and questionnaires [29]. Threat identification and mitigation is identified without and automated reasoning support. The second class of tools improves the modelling experience by providing a graphical modelling environment but without a rigorous formal model [30, 31, 32, 33]. Finally, the third class of tools are model-based system engineering solutions with an underlying formal threat model and provide full support for automated threat analysis [30, 31, 32, 33].

Attack trees [34] describe sophisticated attack patterns that capture sequences of basic attack steps and describe how these can be combined to reach a target. Graphical modelling and analysis of attack trees is supported by sev-

Table 10.1: Threats related to the display in the case study example (MED=medium; MOD=moderate; SEV=severe)

Target	Affected Asset	Damage Scenario	Threat Title	Category	Impact Cat.	Likel.	Impact	Risk
Touch-screen	n/a	n/a	Tamper through external ports	TAMPERING		MED	MOD	3
Touch-screen	n/a	n/a	Physical Tampering	TAMPERING		MED	MOD	3
Touch-screen	Information Availability	Operational impact	Physical Tampering	TAMPERING	Operational	MED	SEV	1
Touch-screen	Information Integrity	Operational impact,	Physical Tampering	TAMPERING	Operational	MED	SEV	1
Touch-screen	Information Integrity	Safety impact	Physical Tampering	TAMPERING	Safety	MED	SEV	1

eral tools [35, 36]. Attack trees can be extended with additional attributes such as possibility, cost, resources [34] or time [37]. Attack trees can be combined with fault trees for a more integrated safety and security analysis or with defender's mitigation measures resulting in the attack-defence tree model [38]. Attack trees are complementary to the more static threat model and the relation between the two has been only seldomly investigated. Isograph Attack-Tree [35] supports threat analysis and risk assessment from the attack tree, following the relevant ISO standards. On the other hand, THREATGET allows automatic generation of attack trees from threat analysis results [39].

The integration of the threat and attack tree modeling and analysis and testing has not been sufficiently investigated in the literature. The only work that we are aware of on this topic is about test generation from attack trees has been studied in the context of the vehicle security in the automotive domain [40]. In this paper, we propose a methodology that goes from threat modelling to the generation of test cases, where attack trees are used as an intermediate step in this process.

10.7 Conclusion

We described a method to automatically generate abstract test scenarios out of a TARA using attack trees and LTSs. The main improvement of this method is that these test scenarios can be derived from a process that is mandated by a CSMS in a simple, automated, and resource-efficient way, which surpasses manual test case generation while still maintaining targeted tests as a result. The resulting scenarios can be further compiled into executable test cases with very low effort once the details of the implemenation are known. We also showed incorporation of CALs and TAF into a security analysis and testing pipeline. These concepts define the level of thoroughness of testing as well as providing a metric for the effectiveness of included safeguards. The required formalization in this manifestation of a testing process is used to increase completeness and efficiency in security testing by using the products of formalized steps in an automated process. Overall, this paper demonstrates a workflow originating from CALs and a TARA, which results are used to generate test cases in an automated manner (via attack tree generation). These tests can be used at various stages of the life cycle and also determine TAFs in the practical implementation stages. Future work includes to utilize machine learning to attribute the test patterns to attack tree edges (instead of a fixed function). This allows for more flexible and experience-based test case generation.

Acknowledgements

This research has received funding within the ECSEL Joint Undertaking (JU) under grant agreements No.101007326 (project AI4CSM) and No. 101007350 (project AIDOaRt). The JU receives support from the European Union's Horizon 2020 research and innovation program and Austria, Sweden, Spain, Italy, France, Portugal, Ireland, Finland, Slovenia, Poland, Netherlands, Turkey. The document reflects only the author's view, and the Commission is not responsible for any use that may be made of the information it contains.

Bibliography

[1] Upstream Security, “Upstream Security Global Automotive Cybersecurity Report,” tech. rep., Upstream Security, 2020.

[2] United Nations Economic and Social Council - Economic Commission for Europe, “UN Regulation on Uniform Provisions Concerning the Approval of Vehicles with Regard to Cyber Security and of Their Cybersecurity Management Systems,” Tech. Rep. ECE/TRANS/WP.29/2020/79, United Nations Economic and Social Council - Economic Commission for Europe / United Nations Economic and Social Council - Economic Commission for Europe, Brussels, 2020.

[3] International Organization for Standardization and Society of Automotive Engineers, “ISO/SAE PAS8475 (WIP) Road Vehicles – Cybersecurity Assurance Levels and Targeted Attack Feasibility - SAE International.” https://www.sae.org/standards/content/iso/sae%20pas8475/, 2022.

[4] International Organization for Standardization and Society of Automotive Engineers, “ISO/SAE PAS8477 (WIP) Road vehicles - cybersecurity verification and validation - SAE International.” https://www.sae.org/standards/content/iso/sae%20pas8477/, 2023.

[5] G. Macher, H. Sporer, R. Berlach, E. Armengaud, and C. Kreiner, “SAHARA: A security-aware hazard and risk analysis method,” in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, (Grenoble, France), pp. 621–624, IEEE, Mar. 2015.

[6] International Organization for Standardization, “Information security, cybersecurity and privacy protection – Evaluation criteria for IT security – Part 2: Security functional components,” ISO/IEC Standard 15408-2:2022, International Organization for Standardization, 2022.

[7] S. Marksteiner, N. Marko, A. Smulders, S. Karagiannis, F. Stahl, H. Hamazaryan, R. Schlick, S. Kraxberger, and A. Vasenev, “A Process to Facilitate Automated Automotive Cybersecurity Testing,” in *2021 IEEE 93rd Vehicular Technology Conference (VTC Spring)*, (New York, NY, USA), pp. 1–7, IEEE, 2021.

[8] International Organization for Standardization and Society of Automotive Engineers, “Road Vehicles – Cybersecurity Engineering,” ISO/SAE Standard ”21434”, International Organization for Standardization, 2021.

[9] D. Ward, I. Ibarra, and A. Ruddle, “Threat Analysis and Risk Assessment in Automotive Cyber Security,” *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, vol. 6, no. 2013-01-1415, pp. 507–513, 2013.

[10] C. Schmittner, B. Schrammel, and S. König, “Asset Driven ISO/SAE 21434 Compliant Automotive Cybersecurity Analysis with ThreatGet,” in *Systems, Software and Services Process Improvement* (M. Yilmaz, P. Clarke, R. Messnarz, and M. Reiner, eds.), Communications in Computer and Information Science, (Cham), pp. 548–563, Springer International Publishing, 2021.

[11] C. C. Michael, K. van Wyk, and W. Radosevich, “Risk-Based and Functional Security Testing,” tech. rep., U.S. Deparmtent of Homeland Security, 2005.

[12] C. Wolschke, S. Marksteiner, T. Braun, and M. Wolf, “An Agnostic Domain Specific Language for Implementing Attacks in an Automotive Use Case,” in *The 16th International Conference on Availability, Reliability and Security*, ARES 2021, (New York, NY, USA), pp. 1–9, Association for Computing Machinery, Aug. 2021.

[13] F. Cuppens and R. Ortalo, “Lambda: A language to model a database for detection of attacks,” in *International Workshop on Recent Advances in Intrusion Detection*, (Berlin, Heidelberg), pp. 197–216, Springer, 2000.

[14] C. Michel and L. Mé, “ADeLe: An Attack Description Language for Knowledge-Based Intrusion Detection,” in *Trusted Information* (M. Dupuy and P. Paradinas, eds.), IFIP International Federation for Information Processing, (Boston, MA), pp. 353–368, Springer US, 2001.

[15] M. Yampolskiy, P. Horváth, X. D. Koutsoukos, Y. Xue, and J. Sztipanovits, “A language for describing attacks on cyber-physical systems,” *International Journal of Critical Infrastructure Protection*, vol. 8, pp. 40–52, Jan. 2015.

[16] D. R. Kuhn, R. N. Kacker, and Y. Lei, “Practical combinatorial testing,” SP 800-142, National Institute of Standards and Technology, 2010.

[17] A. Shostack, *Threat Modeling: Designing for Security*. Indianaplois, IN: John Wiley & Sons, 2014.

[18] H. S. Lallie, K. Debattista, and J. Bal, “A review of attack graph and attack tree visual syntax in cyber security,” *Computer Science Review*, vol. 35, p. 100219, Feb. 2020.

[19] R. Sell, M. Leier, A. Rassõlkin, and J.-P. Ernits, “Autonomous Last Mile Shuttle ISEAUTO for Education and Research,” *International Journal of Artificial Intelligence and Machine Learning*, vol. 10, pp. 18–30, Jan. 2020.

[20] D. Eng, “Integrated Threat Modelling,” Master’s thesis, University of Olso, 2017.

[21] T. R. Ingoldsby, “Attack tree-based threat risk analysis,” tech. rep., Amenaza Technologies Limited, 2021.

[22] M. S. Haque and T. Atkison, “An Evolutionary Approach of Attack Graph to Attack Tree Conversion,” *International Journal of Computer Network and Information Security*, vol. 9, pp. 1–16, Nov. 2017.

[23] C. Phillips and L. P. Swiler, “A graph-based system for network-vulnerability analysis,” in *Proceedings of the 1998 Workshop on New Security Paradigms*, (New York, NY, USA), pp. 71–79, ACM, 1998.

[24] B. Schneier, “Attack trees,” *Dr. Dobb’s journal*, vol. 24, no. 12, pp. 21–29, 1999.

[25] P. Ammann, D. Wijesekera, and S. Kaushik, “Scalable, graph-based network vulnerability analysis,” in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, (New York, NY, USA), pp. 217–224, ACM, 2002.

[26] C. A. Petri, *Kommunikation mit Automaten*. PhD thesis, Technische Universität Darmstadt, 1962.

[27] R. M. Keller, "Formal verification of parallel programs," *Communications of the ACM*, vol. 19, pp. 371–384, July 1976.

[28] R. Khan, K. McLaughlin, D. Laverty, and S. Sezer, "Stride-based threat modeling for cyber-physical systems," in *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, (New York, NY), pp. 1–6, IEEE, 2017.

[29] Tutamantic Ltd., "Tutamen threat model automator." Online, 2020. Accessed: 2020-11-29.

[30] Foreseeti AB, "Foreseeti." Online, 2020. Accessed: 2020-11-29.

[31] J. Was, P. Avhad, M. Coles, N. Ozmore, R. Shambhuni, and I. Tarandach, "Owasp pytm." Online, 2020. Accessed: 2020-11-29.

[32] M. E. Sadany, C. Schmittner, and W. Kastner, "Assuring compliance with protection profiles with threatget," in *SAFECOMP 2019 Workshops*, Lecture Notes in Computer Science, (Berlin), pp. 62–73, Springer, 2019.

[33] K. Christl and T. Tarrach, "The analysis approach of threatget," *CoRR*, vol. abs/2107.09986, 2021.

[34] S. Mauw and M. Oostdijk, "Foundations of attack trees," in *Information Security and Cryptology - ICISC 2005* (D. H. Won and S. Kim, eds.), vol. 3935, pp. 186–198, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[35] Isograph, "Isograph attacktree." Online, 2023. Accessed: 2023-10-03.

[36] Amenaza Technologies Limited, "Securitree." Online, 2023. Accessed: 2023-10-03.

[37] J. Bryans, H. N. Nguyen, and S. A. Shaikh, "Attack defense trees with sequential conjunction," in *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*, (Hangzhou, China), pp. 247–252, IEEE, 2019-01.

[38] B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer, "Foundations of attack–defense trees," in *Formal Aspects of Security and Trust* (P. Degano,

S. Etalle, and J. Guttman, eds.), vol. 6561, pp. 80–95, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. Series Title: Lecture Notes in Computer Science.

[39] S. Chlup, K. Christl, C. Schmittner, A. M. Shaaban, S. Schauer, and M. Latzenhofer, "THREATGET: towards automated attack tree analysis for automotive cybersecurity," *Inf.*, vol. 14, no. 1, p. 14, 2023.

[40] M. Cheah, H. N. Nguyen, J. Bryans, and S. A. Shaikh, "Formalising systematic security evaluations using attack trees for automotive applications," in *Information Security Theory and Practice* (G. P. Hancke and E. Damiani, eds.), vol. 10741, pp. 113–129, Cham: Springer International Publishing, 2018. Series Title: Lecture Notes in Computer Science.