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

Stefan Marksteiner
stefan.marksteiner@avl.com
AVL List Gmbh
Graz, Austria
Mälardalen University
Västerås, Sweden

Christoph Schmittner
christoph.schmittner@ait.ac.at
AIT - Austrian Institute of Technology GmbH
Vienna, Austria

Korbinian Christl
korbinian.christl@ait.ac.at
AIT - Austrian Institute of Technology GmbH
Vienna, Austria

Dejan Ničković
dejan.nickovic@ait.ac.at
AIT - Austrian Institute of Technology GmbH
Vienna, Austria

Mikael Sjödin
mikael.sjodin@mdu.se
Mälardalen University
Västerås, Sweden

Marjan Sirjani
marjan.sirjani@mdu.se
Mälardalen University
Västerås, Sweden

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.

CSCS '23, December 05, 2023, Darmstadt, Germany
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0454-3/23/12…$15.00
https://doi.org/10.1145/3631204.3631864

CCS CONCEPTS

• **Security and privacy** → **Systems security**; **Penetration testing**; • **Software and its engineering** → **Software verification and validation**.

KEYWORDS

Automotive, Cybersecurity, Testing, Life Cycle, CAL, TAF

ACM Reference Format:
Stefan Marksteiner, Christoph Schmittner, Korbinian Christl, Dejan Ničković, Mikael Sjödin, and Marjan Sirjani. 2023. From TARA to Test: Automated Automotive Cybersecurity Test Generation Out of Threat Modeling. In *Computer Science in Cars Symposium (CSCS '23), December 05, 2023, Darmstadt, Germany.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3631204.3631864

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 [36]. 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 [35]. 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 2 describes the latter concepts and their integration in a security testing process. Section 3 elaborates automating the process of generating suitable threat models and attack trees. Section 4 describes the transfer mechanism from attack trees to agnostic test cases and their application to an actual implementation. Section 5 describes the application of the process in a small case study. Section 6 gives an overview of different work in this direction and Section 7, eventually, concludes the paper.

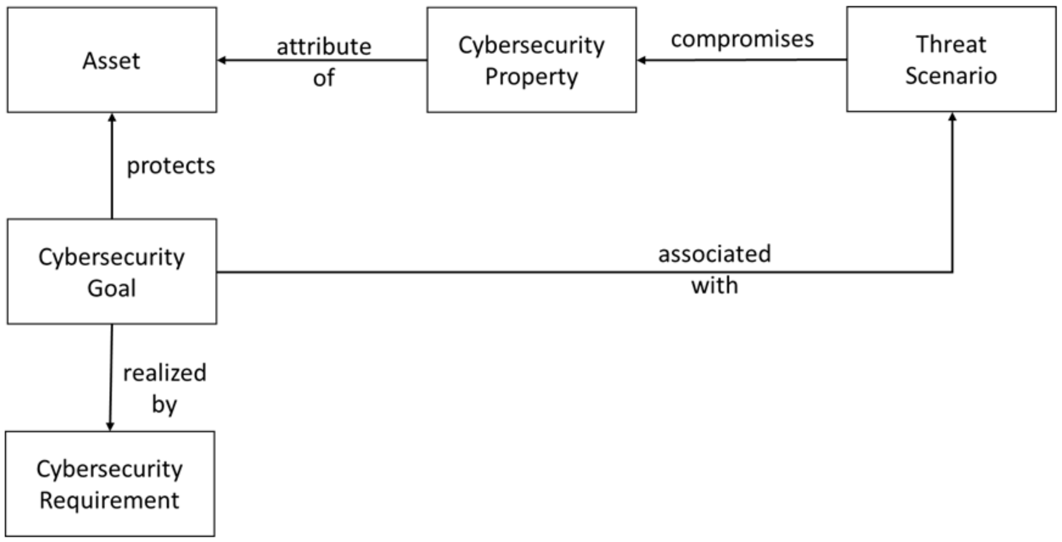


Figure 1: Relationship for risk mitigation.

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] [14] that copes with Cybersecurity Assurance Levels (CALs) and Targeted Attack Feasibility (TAF) and ISO/SAE PAS 8477 (WIP) [15] 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.

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 2.1). Item 2 discusses the merit of the CAL/TAF usage in security testing (Section 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 Analysis and Risk Assessment (TARA) (Section 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 4.2).

[1]Work-in-Progress

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 [22] 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 1). This process can be applied during all phases of the development, at item (see Section 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.

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 [35] 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) [12], 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.

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.

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 [23] and adapted to include CAL and TAF. The process is aligned with ISO/SAE 21434 [13]. The activities are basically sequential, although some activities provide input for more than one subsequent activity. Figure 2 provides an overview.

I Item Definition
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

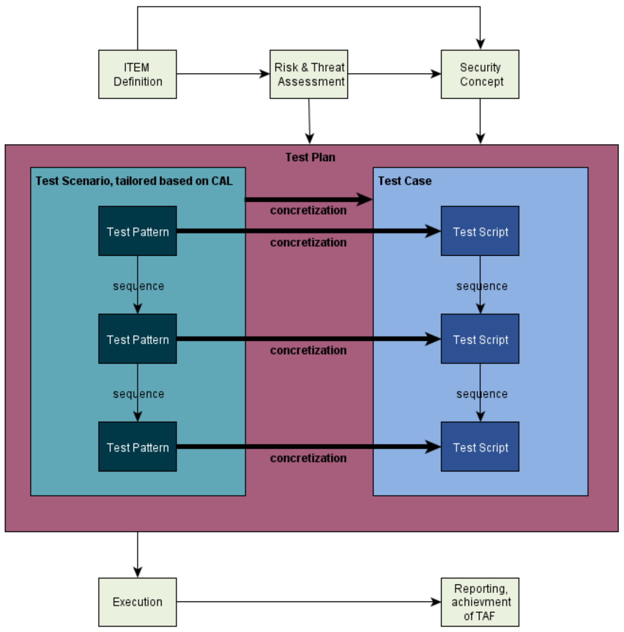


Figure 2: Layout of the security testing process from [23].

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 [30, 37]) 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 [25] and match test patterns which represent abstract (symbolic) actions in a distinct sequence. The scenarios are categorized into four classes [13]: 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 [7, 26, 39, 40]. 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 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 [20]. 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.

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 [33]. 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 [21]. 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 3 shows an example of a threat model based on [32]. 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 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 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.

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 [32, 37]. It involves systematically investigating threats, assessing their likelihood and impact, and developing strategies to address the identified risks [8]. 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 [11, 32]. 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 [10]. 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 [21]. In an attack tree [28, 31], 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

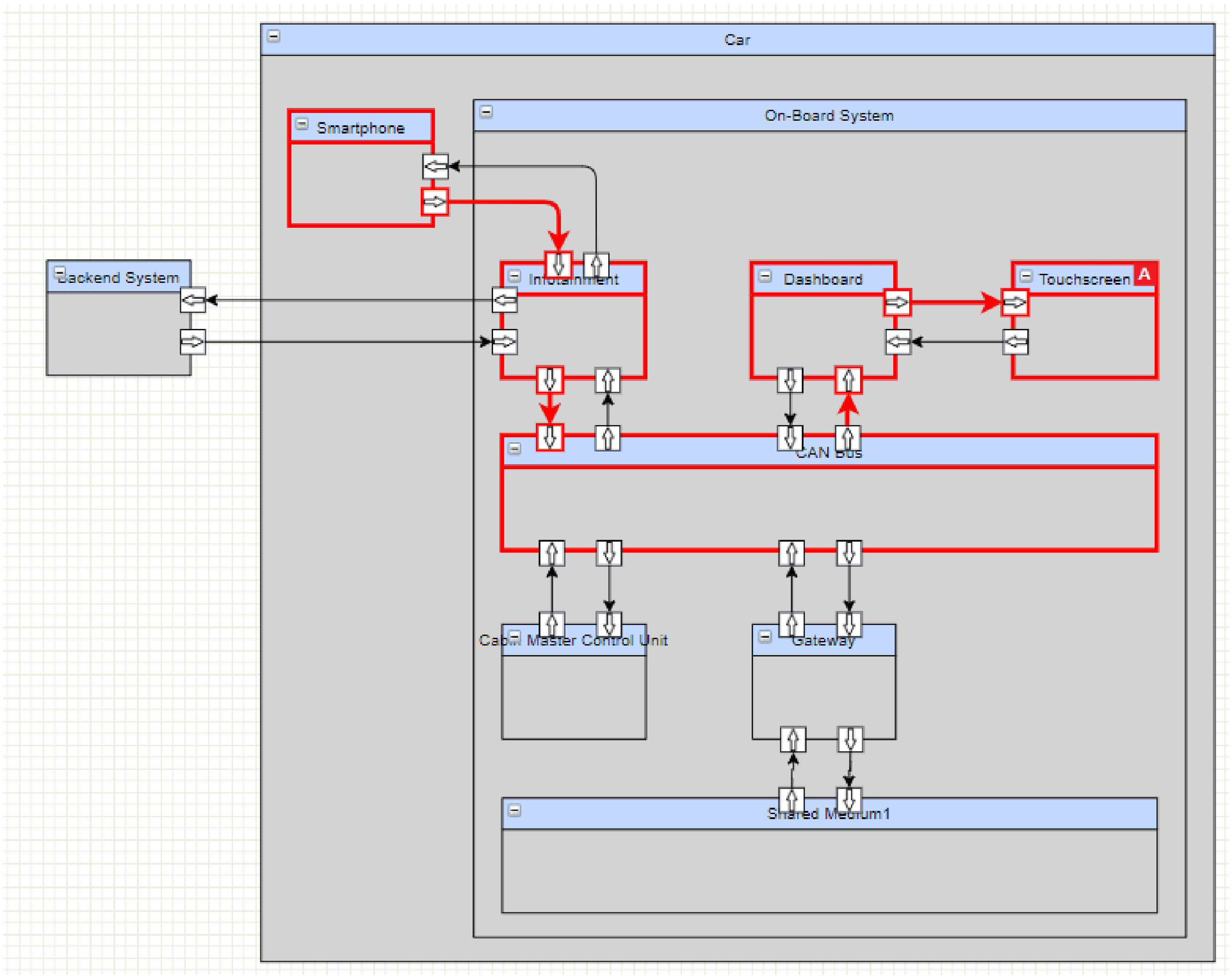


Figure 3: Example Threat Model.

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 inter-dependencies of threats within the attack tree not only simplifies, but also improves, the assessment of target attack feasibility [2]. 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 [11]. 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.

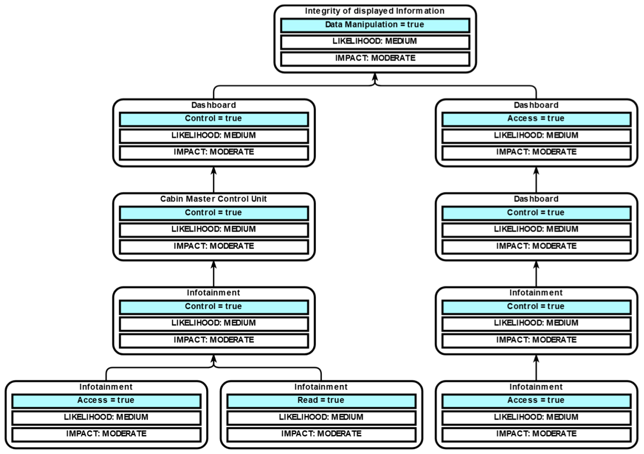


Figure 4: Example attack tree.

4 AUTOMATED TESTING

This section is concerned with the automated generation of security test cases stemming from a TARA using attack trees (see Section 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.

4.1 Security Tests and their relationship with the Security Analysis

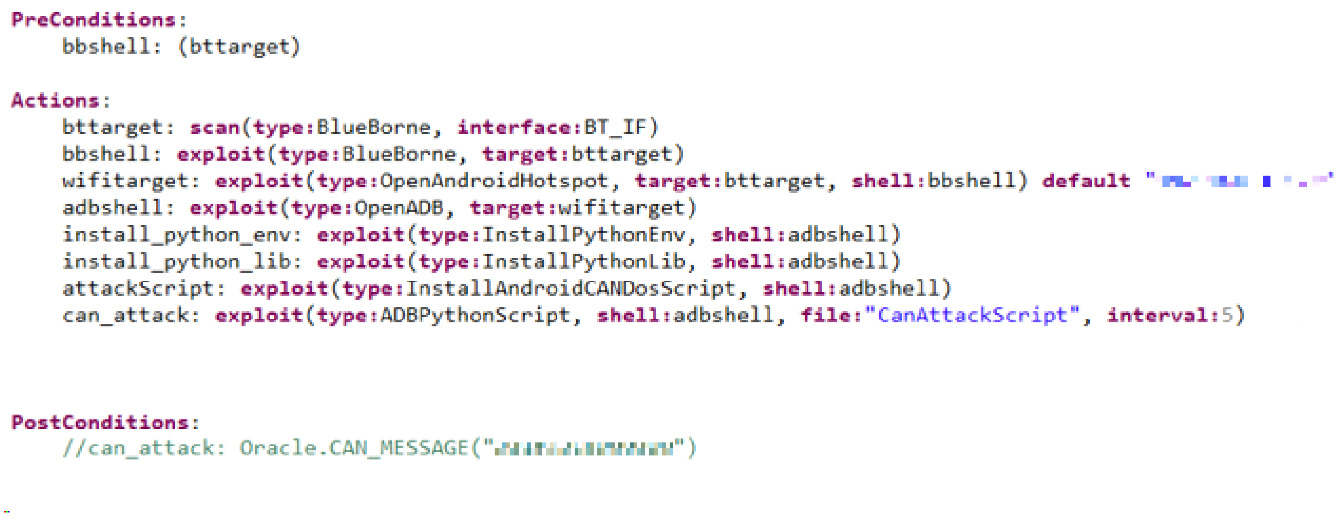
Following the method in Section 2.3, we store blueprints for test cases in a system-agnostic manner in the ALIA DSL [39] as test scenarios (see Figure 5 for an example). These test scenarios are an abstract representation of actions to be taken to execute a test case. The actions are accompanied by preconditions 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.

4.2 Security Test Generation

Using an attack graph (such as a tree, but also other structures like petri nets [27] 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 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 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 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 ($q, q\prime \in Q, \sigma \in \Sigma; q \xrightarrow{\sigma} q\prime$) [17]. 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 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



```
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 5: Example for a test scenario in the used attack description language.

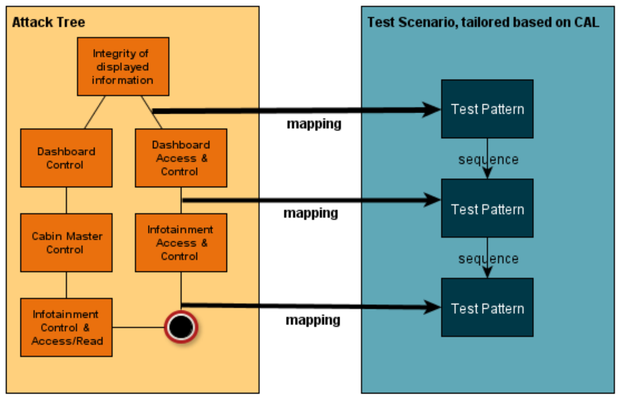

Attack Tree
Integrity of displayed information
Dashboard Control
Dashboard Access & Control
Cabin Master Control
Infotainment Access & Control
Infotainment Control & Access/Read
mapping
mapping
mapping
Test Scenario, tailored based on CAL
Test Pattern
sequence
Test Pattern
sequence
Test Pattern

Figure 6: Attack tree to Test Scenario transformation example.

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.

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 3. The threat analysis using the tool yielded a list of 103 threats (using the STRIDE methodology [18]). Using the methodology in Section 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 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 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 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 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 [39] 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.

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 [34]. 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 [6, 9, 29, 38]. 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 [6, 9, 29, 38].

Attack trees [24] 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 several tools [1, 16]. Attack trees can be extended with additional attributes such as possibility, cost, resources [24] or time [3]. 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 [19]. Attack trees are complementary to the more static threat model and the relation between the two has been only seldomly investigated. Isograph AttackTree [16] 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 [5].

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 [4]. 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.

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.

ACKNOWLEDGMENTS

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.

REFERENCES

[1] Amenaza Technologies Limited. 2023. SecurITree. Online. https://www.amenaza.com Accessed: 2023-10-03.

Table 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
Touchscreen	n/a	n/a	Tamper through external ports	TAMPERING		MED	MOD	3
Touchscreen	n/a	n/a	Physical Tampering	TAMPERING		MED	MOD	3
Touchscreen	Information Availability	Operational impact	Physical Tampering	TAMPERING	Operational	MED	SEV	1
Touchscreen	Information Integrity	Operational impact,	Physical Tampering	TAMPERING	Operational	MED	SEV	1
Touchscreen	Information Integrity	Safety impact	Physical Tampering	TAMPERING	Safety	MED	SEV	1

[2] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. 2002. Scalable, Graph-Based Network Vulnerability Analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM, New York, NY, USA, 217–224.

[3] Jeremy Bryans, Hoang Nga Nguyen, and Siraj Ahmed Shaikh. 2019-01. Attack Defense Trees with Sequential Conjunction. In *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*. IEEE, Hangzhou, China, 247–252. https://doi.org/10.1109/HASE.2019.00045

[4] Madeline Cheah, Hoang Nga Nguyen, Jeremy Bryans, and Siraj A. Shaikh. 2018. Formalising Systematic Security Evaluations Using Attack Trees for Automotive Applications. In *Information Security Theory and Practice*, Gerhard P. Hancke and Ernesto Damiani (Eds.). Vol. 10741. Springer International Publishing, Cham, 113–129. https://doi.org/10.1007/978-3-319-93524-9_7 Series Title: Lecture Notes in Computer Science.

[5] Sebastian Chlup, Korbinian Christl, Christoph Schmittner, Abdelkader Magdy Shaaban, Stefan Schauer, and Martin Latzenhofer. 2023. THREATGET: Towards Automated Attack Tree Analysis for Automotive Cybersecurity. *Inf.* 14, 1 (2023), 14. https://doi.org/10.3390/info14010014

[6] Korbinian Christl and Thorsten Tarrach. 2021. The analysis approach of ThreatGet. *CoRR* abs/2107.09986 (2021), 57 pages. arXiv:2107.09986 https://arxiv.org/abs/2107.09986

[7] Frédéric Cuppens and Rodolphe Ortalo. 2000. Lambda: A Language to Model a Database for Detection of Attacks. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, Berlin, Heidelberg, 197–216.

[8] Dag Eng. 2017. *Integrated Threat Modelling*. Master's thesis. University of Olso.

[9] Foreseeti AB. 2020. Foreseeti. Online. https://foreseeti.com/ Accessed: 2020-11-29.

[10] Md. Shariful Haque and Travis Atkison. 2017. An Evolutionary Approach of Attack Graph to Attack Tree Conversion. *International Journal of Computer Network and Information Security* 9, 11 (Nov. 2017), 1–16. https://doi.org/10.5815/ijcnis.2017.11.01

[11] Terrance R Ingoldsby. 2021. *Attack Tree-Based Threat Risk Analysis*. Technical Report. Amenaza Technologies Limited.

[12] International Organization for Standardization. 2022. *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.

[13] International Organization for Standardization and Society of Automotive Engineers. 2021. *Road Vehicles – Cybersecurity Engineering*. ISO/SAE Standard "21434". International Organization for Standardization.

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

[15] International Organization for Standardization and Society of Automotive Engineers. 2023. ISO/SAE PAS8477 (WIP) Road Vehicles - Cybersecurity Verification and Validation - SAE International. https://www.sae.org/standards/content/iso/sae%20pas8477/.

[16] Isograph. 2023. Isograph AttackTree. Online. https://www.isograph.com/software/attacktree/ Accessed: 2023-10-03.

[17] Robert M. Keller. 1976. Formal Verification of Parallel Programs. *Commun. ACM* 19, 7 (July 1976), 371–384. https://doi.org/10.1145/360248.360251

[18] Rafiullah Khan, Kieran McLaughlin, David Laverty, and Sakir Sezer. 2017. STRIDE-based threat modeling for cyber-physical systems. In *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*. IEEE, New York, NY, 1–6.

[19] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. 2011. Foundations of Attack–Defense Trees. In *Formal Aspects of Security and Trust*, Pierpaolo Degano, Sandro Etalle, and Joshua Guttman (Eds.). Vol. 6561. Springer Berlin Heidelberg, Berlin, Heidelberg, 80–95. https://doi.org/10.1007/978-3-642-19751-2_6 Series Title: Lecture Notes in Computer Science.

[20] D Richard Kuhn, Raghu N Kacker, and Yu Lei. 2010. *Practical Combinatorial Testing*. SP 800-142. National Institute of Standards and Technology.

[21] Harjinder Singh Lallie, Kurt Debattista, and Jay Bal. 2020. A Review of Attack Graph and Attack Tree Visual Syntax in Cyber Security. *Computer Science Review* 35 (Feb. 2020), 100219. https://doi.org/10.1016/j.cosrev.2019.100219

[22] Georg Macher, Harald Sporer, Reinhard Berlach, Eric Armengaud, and Christian Kreiner. 2015. SAHARA: A Security-Aware Hazard and Risk Analysis Method. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Grenoble, France, 621–624. https://doi.org/10.7873/DATE.2015.0622

[23] Stefan Marksteiner, Nadja Marko, Andre Smulders, Stelios Karagiannis, Florian Stahl, Hayk Hamazaryan, Rupert Schlick, Stefan Kraxberger, and Alexandr Vasenev. 2021. A Process to Facilitate Automated Automotive Cybersecurity Testing. In *2021 IEEE 93rd Vehicular Technology Conference (VTC Spring)*. IEEE, New York, NY, USA, 1–7.

[24] Sjouke Mauw and Martijn Oostdijk. 2005. Foundations of Attack Trees. In *Information Security and Cryptology - ICISC 2005*, Dong Ho Won and Seungjoo Kim (Eds.). Vol. 3935. Springer Berlin Heidelberg, Berlin, Heidelberg, 186–198. https://doi.org/10.1007/11734727_17

[25] C. C. Michael, Ken van Wyk, and Will Radosevich. 2005. *Risk-Based and Functional Security Testing*. Technical Report. U.S. Deparmtent of Homeland Security.

[26] Cédric Michel and Ludovic Mé. 2001. ADeLe: An Attack Description Language for Knowledge-Based Intrusion Detection. In *Trusted Information (IFIP International Federation for Information Processing)*, Michel Dupuy and Pierre Paradinas (Eds.). Springer US, Boston, MA, 353–368. https://doi.org/10.1007/0-306-46998-7_25

[27] Carl Adam Petri. 1962. *Kommunikation mit Automaten*. Ph. D. Dissertation. Technische Universität Darmstadt.

[28] Cynthia Phillips and Laura Painton Swiler. 1998. A Graph-Based System for Network-Vulnerability Analysis. In *Proceedings of the 1998 Workshop on New Security Paradigms*. ACM, New York, NY, USA, 71–79.

[29] Magdy El Sadany, Christoph Schmittner, and Wolfgang Kastner. 2019. Assuring Compliance with Protection Profiles with ThreatGet. In *SAFECOMP 2019 Workshops (Lecture Notes in Computer Science)*. Springer, Berlin, 62–73.

[30] Christoph Schmittner, Bernhard Schrammel, and Sandra König. 2021. Asset Driven ISO/SAE 21434 Compliant Automotive Cybersecurity Analysis with ThreatGet. In *Systems, Software and Services Process Improvement (Communications in Computer and Information Science)*, Murat Yilmaz, Paul Clarke, Richard Messnarz, and Michael Reiner (Eds.). Springer International Publishing, Cham, 548–563. https://doi.org/10.1007/978-3-030-85521-5_36

[31] Bruce Schneier. 1999. Attack Trees. *Dr. Dobb's journal* 24, 12 (1999), 21–29.

[32] Raivo Sell, Mairo Leier, Anton Rassõlkin, and Juhan-Peep Ernits. 2020. Autonomous Last Mile Shuttle ISEAUTO for Education and Research. *International Journal of Artificial Intelligence and Machine Learning* 10, 1 (Jan. 2020), 18–30. https://doi.org/10.4018/IJAIML.2020010102

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

[34] Tutamantic Ltd. 2020. Tutamen Threat Model Automator. Online. https://www.tutamantic.com/ Accessed: 2020-11-29.

[35] United Nations Economic and Social Council - Economic Commission for Europe. 2020. *UN Regulation on Uniform Provisions Concerning the Approval of Vehicles with Regard to Cyber Security and of Their Cybersecurity Management Systems*. Technical Report 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.

[36] Upstream Security. 2020. *Upstream Security Global Automotive Cybersecurity Report*. Technical Report. Upstream Security.

[37] David Ward, Ireri Ibarra, and Alastair Ruddle. 2013. Threat Analysis and Risk Assessment in Automotive Cyber Security. *SAE International Journal of Passenger Cars-Electronic and Electrical Systems* 6, 2013-01-1415 (2013), 507–513.

[38] Jan Was, Pooja Avhad, Matthew Coles, Nick Ozmore, Rohit Shambhuni, and Izar Tarandach. 2020. OWASP pytm. Online. https://owasp.org/www-project-pytm/ Accessed: 2020-11-29.

[39] Christian Wolschke, Stefan Marksteiner, Tobias Braun, and Markus Wolf. 2021. 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)*. Association for Computing Machinery, New York, NY, USA,

1–9. https://doi.org/10.1145/3465481.3470070
[40] Mark Yampolskiy, Péter Horváth, Xenofon D. Koutsoukos, Yuan Xue, and Janos Sztipanovits. 2015. A Language for Describing Attacks on Cyber-Physical Systems. *International Journal of Critical Infrastructure Protection* 8 (Jan. 2015), 40–52. https://doi.org/10.1016/j.ijcip.2014.09.003