Reaching Grey Havens
Industrial Automotive Security Modeling with SAM

Markus Zoppelt

Department of Computer Science
Nuremberg Institute of Technology
Nuremberg, Bavaria 90489
Email: markus.zoppelt@th-nuernberg.de

Ramin Tavakoli Kolagari

Department of Computer Science
Nuremberg Institute of Technology
Nuremberg, Bavaria 90489
Email: ramin.tavakolikolagari@th-nuernberg.de

***Abstract*—Autonomous vehicles have a greater attack potential than any previous individual mobility vehicle. This is primarily due to the considerable communication demands of the vehicles, which on the one hand emerge for reasons of functionality and safety, and on the other hand for reasons of comfort. Driverless vehicles require communication interfaces to the environment, direct connections (e.g., Vehicle-to-X) and connections to an original equipment manufacturer backend service or a cloud. These communication connections could all be used as backdoors for attacks. Most existing countermeasures against cyber attacks, e.g., the use of message cryptography, concentrate on concrete attacks and do not consider the complexity of the various access options offered by modern vehicles. This is mainly due to a solution-oriented approach to security problems. The model-based technique SAM (Security Abstraction Model) adds to the early phases of (automotive) software architecture development by explicitly documenting attacks and managing them with the appropriate security countermeasures. It additionally establishes the basis for comprehensive security analysis techniques, e.g., already available attack assessment methods. SAM thus contributes to an early, problem-oriented and solution-ignorant understanding combining key stakeholder knowledge. This paper provides a detailed overview of SAM and evaluates this security technology using interviews with industry experts and a grounded theory analysis. The resulting analyses of this evaluation show that SAM puts the security-by-design principle into practice by enabling collaboration between automotive system engineers, system architects and security experts. The application of SAM aims to reduce costs, improve overall quality and gain competitive advantages. Based on our evaluation results, SAM is highly suitable, comprehensible and complete to be used in the industry.**

Keywords–Automotive Security; Automotive Software Engineering; Security Modeling; Model-based Security; Autonomous Driving.

I. INTRODUCTION

Modern vehicles are interconnected computer networks in which many electronic control units (ECUs) communicate with one another and with the environment (Vehicle-to-X communication). In recent years, car manufacturers have been producing vehicles that have an online connection and offer cloud services, such as the mobile app from Tesla, BMW iDrive or Audi Connect. In most cases, the user can actually monitor or control parts of the vehicle via a mobile application or cloud service. These convenience features are intended to attract new customers, but can also be access points for new attacks [1]. Considering the fact that autonomous vehicles will continue rather than reverse the trend towards more communication interfaces for reasons of functionality, safety and comfort, making collective research efforts in the field of vehicle security understandable; after all, human lives are at stake every time these "driving computers" are the target of attacks.

As far as security experts are concerned, it should be noted that car attackers do not target cars the same way as they attack desktop computer systems, because cars use different networks, protocols and architectures [2], [3]. In addition, vehicles often contain obsolete legacy mechanisms with unsecure and unencrypted protocols (e.g., Controller Area Network (CAN)) in their system design, because they were originally not designed in accordance with today's security principles [4], [5]. Secure automotive network architectures were not prioritized in the past due to the general prejudice that cars are secure due to their technical complexity (security by obscurity). Sluggish development processes, lack of standard guidelines and low societal pressure, due to little attack experience in practice, lead to a rather slow transformation of automotive development processes, which systematically implement security by design.

Most existing countermeasures against cyber attacks, e.g., the use of message cryptography for encrypting, authenticating or randomizing vehicle-level network messages, focus on concrete attacks and do not consider the complexity of the access options offered by modern vehicles, as shown by Zoppelt et al. [6]. This is mainly due to a solution-oriented approach to security problems.

The model-based technique SAM (Security Abstraction Model) [7] adds to the early, solution-ignorant phases of (automotive) software architecture development by explicitly documenting attacks and managing them with appropriate security countermeasures. The documentation of the attacks together with their motivation, vulnerability, attackable property and other relevant properties is put in relation to the entire system description by SAM being an annex to the domain-specific architecture description language EAST-ADL [8]. Thus, all available information about the system is linked with potential attack scenarios at an early stage of the automotive system development and cooperation between the key stakeholders is made possible. The problem-oriented documentation allows automotive system developers and security experts to gain a comprehensive picture of the overall attack situation before

developing a solution that otherwise may be too short-sighted.

Zoppelt et al. [7] presented a Security Abstraction Model (SAM) for automotive software systems. In this publication, for the first time, we present a comprehensive description of SAM, put it in context with key security challenges for autonomous driving and evaluate it using grounded-theory evaluation based on interviews.

In this paper, we show:

- A systematic discussion of the current state of the art for security techniques along the V-Model as a common software engineering practice.
- A detailed description of SAM, including all of its metamodel entities.
- An evaluation of the security technique via grounded-theory interviews with industry experts.

The rest of this paper is structured as follows: Section II reviews related work on security architectures for automotive software systems. Section III reviews the state of the art on attacks on modern vehicles and automotive security modeling. Section IV discusses possible attack scenarios and the security challenges in the automotive domain. Section V describes the Security Abstraction Model in detail, including all of its metamodel entities. In Section VI we evaluate SAM and elaborate on the interviews and qualitatively analyse the results via grounded theory. Section VII concludes the paper and gives an outlook on future work.

II. Related Work

The ISO/SAE 21434 "Road Vehicles—Cybersecurity engineering" standard [9], which is currently under development at the time of writing this paper, proposes the introduction of security work packages, security concepts and architectures along the V-Model [10]. It suggests security support before after-sales. Specifically, during product validation and production ramp-up. The standard is being delegated between an consortium of 12 countries. The scope of the standard is to define a framework to include requirements for cybersecurity processes and a common language for communicating and managing cybersecurity risk among stakeholders. Our work considers the early efforts and design principles of the ISO/SAE 21434 and integrates them into the EAST-ADL.

The SAE J3061 "Cybersecurity Guidebook for Cyber-Physical Vehicle Systems" [11], also only available as a work in progress, wants to establish a set of high-level guiding principles for cybersecurity as it relates to cyber-physical vehicle systems, including lifecycle process frameworks and information on common existing tools and methods.

Although not much final information on those standards is currently available, we join and unite many of the proposed methods and principles in our contribution and show its practical applicability in a system model.

PRESERVE was an "EU-funded project running from 2011 to 2015 and contributed to the security and privacy of future vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2X) communication systems. It provides security requirements of vehicle security architectures" [12]. The EVITA project tries to "design, verify and prototype an architecture for automotive on-board networks where security-relevant components are protected against tampering and sensitive data are protected against compromise. It focuses on V2X communications and provides a base for secure deployment of electronic safety applications" [13]. Holm [14] features a Cyber Security Modeling Language (CySeMoL) for enterprise architectures. Juerjens [15] introduces UMLSec, which allows to express security-relevant information within the diagrams in a system specification. Other solutions include INCOSE work on integrating system engineering with system security engineering [16], NIST SP 800-160 [17] and other NIST work on cyber-physical systems [18].

All these solutions have one essential downside: Other than SAM (see Section V), they are stand-alone and are not integrated into an existing system model. SAM is fully integrated into EAST-ADL. In comparison with alternative solutions, a tightly coupled solution with the system model enables a seamless integration of a security model into a system model that is extensively used in the automotive industry. This helps overall acceptance and increases probability for adoption. The tight interplay of SAM with existing system models architectural considerations and practical security considerations together.

III. State of the Art

The automotive core development process is organized according to the traditional software engineering V-Model [10]. Each phase of the V-Model stands for a coherent set of process steps in which a set of artifacts are produced. The phases are logically organized, not temporally. In the system analysis phase, requirements are elicited and documented, in the system design phase, a logical, function-oriented architectural structure is developed that is the basis for both the hardware and software development phases, which results in the implementation of the automotive system. In the following, we discuss the current state of the art for security techniques along the V-Model as a common software engineering practice. For that, we will differentiate between the four major phases of software engineering, namely analysis phase, design phase, implementation phase and software test. Moreover, we will also touch on the topic of security techniques during maintenance, since this is part of the extended V-Model.

A. Security Techniques in Analysis Phase

In the system analysis phase, requirements are elicted and documented. The current state of the practice for security techniques in this phase is to capture requirements from a specification or textual annotations of the system model. Security experts identify threats and vulnerabilities of a system, while software engineers fix bugs and implement security functionality, e.g., cryptographic functions. System architects define the architecture of the system (i.e., the software and hardware topology), taking—among other things—security requirements into consideration.

A case study conducted by Zoppelt et al. [7] has shown that textual annotations cannot fully explain security scenarios in a detailed, yet compact manner. The technical details and the relevance of the threat get lost because the software engineers could not decide for what purpose or security goal textual notes were intended. Security has an inner complexity, especially considering the requirements entailed. Requirements alone are not sufficient enough. System architects and security experts need to be able to mutually annotate the same model. Only

then they can make the necessary adjustments to the system's architecture. A better practice is to define a set of security goals and systematically derive security requirements from a common architecture model and reference architectures.

Threat modeling and risk assessment is a basis for security requirements. Studies like Kadhirvelan's work [19] have shown that new processes, standards, methods and tools are necessary for evaluating the security and safety of software-intensive automotive systems.

Results of the analysis phase (mostly requirements) are then used in the design phase to deduce attack vectors and think of solutions according to derived requirements.

B. Security Techniques in Design Phase

In the system design phase, a logical, function-oriented architectural structure is developed utilizing the requirements (for both software and hardware) from the analysis phase. Many design decisions can be derived from knowing, analyzing or building different attack vectors a potential adversary has to target the system.

Attack vectors are a path or means by which an adversary can gain unauthorized access to a target system [20] or which hurts one or more security goals. Attack vectors can be identified and extracted via attack trees. Attack trees can be used to illustrate complex attack structures. The root of an attack tree describes the main goal or motivation of an attack, e.g., controlling certain functions of the target vehicle. Every sub-attack needs to be completed in order to fulfill the parent attack in the tree. The leafs of the attack tree are atomic actions or conditions. A complete path from one of the leaves to the root of the tree represents a concrete attack vector.

Moreover, security techniques in design phase utilize various systems for attack rating and threat analyses. Those techniques, e.g., the CVSS [21], allow for an early evaluation of essential security measures. The CVSS is an acclaimed industry standard for rating vulnerabilities in computer systems. The CVSS bundles the result of threat analyses via multiple different metrics, e.g., attack complexity, security goal impacts, etc. and produces a numerical score reflecting its severity. Forcing developers to think about attack vectors and vulnerability already in design phase and conducting a CVSS analysis, ensures that errors are detected early. This way, expensive corrections can be prevented from the beginning. Unfortunately, this is not the case. The current state of the practice shows that OEMs forego this analysis at an early stage. We are trying to contribute to this problem with the solution approach presented in this paper. Alongside CVSS—which is not per-se automotive related—more scoring systems exist, e.g., the SecL levels from the SAHARA method [22].

Another significant design choice is how modern vehicles communicate critical- and safety-relevant commands between different types of ECUs. The most popular broadcast network used for communication—even today—is the CAN bus [23]. CAN bus messages are unencrypted and unsigned by default, because back in the 80's—when CAN was designed—automotive security was not perceived a major issue. Remote exploitation of a single ECU item on the CAN bus causes a major security threat because it allows an attacker to send valid (and potentially harmful) messages over the bus to critical parts of the vehicle's ECU network. Modern vehicles have a tremendous amount of remote attack surfaces like wireless protocols, mobile application support and more. Examples of specific remote technologies are the passive anti-theft system (PATS), tire pressure monitoring systems (TPMS), remote keyless entry (RKE), Bluetooth, radio data systems (3G, 4G, LTE, 5G, etc.), Wi-Fi and telematics. Typically, infotainment systems tend to feature Internet access and support for third-party applications. Various attacks [24] have shown that adversaries are able to cause serious threats by compromising a vehicle's ECU (or adding an external device) and sending malicious CAN commands to the devices listening on the bus. Once the adversary has the ability to send arbitrary CAN messages, she is able to control the braking system, engine behaviour, the air vents, (un-)lock the doors, etc. Therefore, there is a strong need to secure the vehicle before the adversary can gain access to the CAN bus. If the adversary has access to the powertrain it is already too late.

Common countermeasure decisions in design phase consider network bus separation. By conceptually and physically separating safety-relevant ECUs from the remaining network system, many attack vectors can be mitigated. In reality, many ECUs are still connected physically, but are being separated through higher protocol abstractions, e.g., Unified Diagnostic Service (UDS) or virtualized applications. Those happen in implementation phase.

C. Security Techniques in Implementation Phase

According to the V-Model, the proposed countermeasures are deployed in the implementation phase.

On the hardware side, implementation may differ by different physical bus systems and wiring. If one or some of these applications or services become vulnerable to hacking attacks over the network, an adversary might be able to control a crucial participant in the physical network of the vehicle: the CAN bus. Another approach in the automotive domain is Automotive Ethernet, though, it is not expected to fully replace the CAN bus. CAN will continue to exist as a low-cost component, for example for connecting low-cost and computationally weak actuators and sensors with their corresponding ECUs or gateways, rather than be used as the main powertrain. As of today, the LIN-bus (Local Interconnect Network) is used for this type (low-cost, low-risk) of connection.

Cost is a limiting factor as well, when it comes to implementing expensive hardware into the vehicle. Automobile manufacturers prefer to spend more money on the salaries of programmers (fixed costs; used for entire fleet) rather than spending a cent more on a hardware part of a vehicle (variable costs; for each vehicle) because of the huge market scale. This means that hardware modules like TPMs (Trusted Platform Modules) are unattractive (cost, weight, space) as a key storing solution for each and every communicating part in the vehicle.

On software side, different protocol variations can be used to implement security measures. Some network protocols like ISO-TP, UDS and OBD2 are on a higher level of abstraction that remedy a few shortcomings of CAN. Figure 1 illustrates selected automotive protocols discussed in this paper in the ISO/OSI reference model.

Although the CAN specification describes CAN as unencrypted by default, a sound solution for encryption and

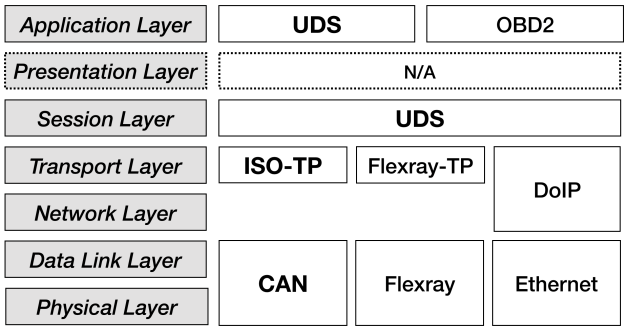

Figure 1. Selected automotive protocols classified in the ISO/OSI reference model

authentication is necessary to ensure a safe and secure distribution of critical new software over this public channel. In the automotive domain, there are not only software updates to consider, but hardware updates as well. If a workshop, for instance, replaces one of the brakes in a vehicle, they might also replace the corresponding ECU. In that scenario, how will the new cryptographic key (for message cryptography) be obtained? Common key distribution techniques like the Diffie-Hellman key exchange [25] are difficult to implement, since many of the smaller network participants are low-cost and computationally weak ECUs. These ECUs often do not feature enough memory or CPU power to perform those cryptographic algorithms and methods. Message cryptography on the CAN bus is not only hard to realize due to the strong network complexity, where key distribution is a difficult problem, but because an adversary in control of an ECU also gets access to the keys stored on that device. For some parts of the vehicle, where stronger threat models are required, e.g., keyless entry systems, emerging technologies like Password Hardened Encryption (PHE) services [26] are promising candidates for securing system components where classic challenge-and-response techniques are insufficient. Protocol implementations of UDS, for example, feature a security seed mechanism that hinders attackers from getting advanced security access. It is shown, however, that weak ciphers or a badly executed implementation of such protocols still allow for successful attacks, e.g., shown by Garcia et al. [27].

D. Security Techniques during Software Test

The implemented countermeasures for the attack vectors analyzed in design phase are tested during software test. Software test in the V-Model verifies that the software is built according to the specification given by the client. Additionally, security testing techniques are quite different to classic software test. Mostly—as is later shown in the evaluation section—security techniques are not even part of the software engineering process. Penetration testing and vulnerability assessments are familiar techniques to check a system for vulnerabilities and security measures. Currently, OEMs are starting to integrate those techniques into their existing processes. Known penetration testing techniques are to reverse engineer bus network traffic or disassembling ECU firmware images trying to get access to keys, secrets or specific messages.

E. Security Techniques during Maintenance

Vehicles have to be maintained and tested after delivery, thus over-the-air (OTA) updates are important, albeit challenging, because there is no secure OTA interface, yet. A clever solution, like PHE with an authentication scheme could resolve this issue. In addition, a fully secured and encrypted system excludes workshops and third-party service providers, which require open access, e.g., for resetting error codes, etc.

OTA updates are most often pulled and received via the infotainment unit, which has access to a 4G, LTE or 5G broadband connection. From there, each and every ECU that needs to receive an update has to get the new firmware or software patch from the infotainment unit via the CAN bus. Rolling out sensitive data, especially new firmware or security patches in case of OTA updates over the CAN bus is incredibly critical and a major liability. OEM updates must be checked and validated before they can be deployed to the range of ECUs connected to the CAN bus. Faulty network configurations and the lack of authentication checks for OTA updates and patches increase the risk of cloud and botnet attacks, e.g., Mirai [28]. Basically, cloud features and OTA updates have to be considered skeptically from the start. Even if the distribution source of the software is the OEM, attacks are still possible. A potential attacker might have found a way to distribute his malware over the OEM's infrastructure (e.g., their servers) and as a result a trust problem arises. It is fair to assume that any kind of roll-out (software updates, cloud data) is untrusted until the key distribution problem has been solved. Even if a solution for key distribution in heterogeneous CAN bus networks is developed, the number of remote attack vectors will rise harshly in comparison to the number of direct attack vectors.

IV. Automotive Attack Scenarios

This section describes the motivation of our approach. This motivation is necessary to highlight the threats and dangers of automotive attack scenarios and attack vectors. Section V will describe how to assess them in more detail with SAM.

Security goals like authenticity, integrity, confidentiality, etc, are especially important to make sure that the safety-critical software of the vehicle stays untampered. The following is a non-exhaustive list of attack vectors that cause major threats to automotive software systems:

- Injection of CAN frames from ECUs that were taken over after the remote attack (e.g., replay attacks, spamming attacks, etc.) [24], [29], [30]
- Reverse engineering of CAN frames by filtering by arbitration IDs and identifying frames via tools like cansniffer or other can-utils [31]
- Rolling out malicious (possibly unsigned) firmware to ECUs [24], [29], [30], [32], [33], [34], [35], [36]
- Gaining remote control access to the vehicle using the OEMs cloud and/or mobile application's infrastructure [33], [35], [37], [38]
- Getting SecurityAccess via Unified Diagnostic Services (UDS) [39]
- Controlling the car via Onboard Diagnostic (OBD) injection [40]
- Remotely breaking into the telematics unit [41]

- Exploiting remote keyless entry with software-defined radios [27]
- Denial of Service (DoS) attacks, e.g., as shown by Palamanca et al. [42]
- Infecting the system with ransomware. [43]

The easiest way to understand the SAM metamodel is to explain which piece of information the individual language components (on the M2 level) actually represent in a concrete M1 model. Therefore, in this section we present an already published attack, which we use to make the individual language building blocks accessible in an exemplary manner in addition to the conceptual explanation. SAM presented in this paper is a tangible solution for this kind of security analysis and security by design. All information needs to be documented in a system model that takes attack modeling for automotive software systems into account. The latest version of SAM introduces new attributes for rating these kinds of attacks.

V. Description of the Security Abstraction Model

In this section we describe our innovative contribution: a Security Abstraction Model (SAM) language specification for the automotive modeling environment as an extension for the EAST-ADL. We clarify the differences between security modeling and functional safety modeling and describe our metamodel entities of SAM to provide a comprehensive modeling environment for automotive security modeling. The entities can be used on the type-level (M1) to create functional architectures for safe and secure automotive systems. SAM is available as an open source project [44] and contains a concrete set of security modeling entities that are fully compliant to the EAST-ADL and AUTOSAR [45] specifications. As such, SAM is a proposition for an annex extending EAST-ADL with security modeling facilities, which are currently not covered by the existing language specification.

A. SAM Metamodel

For the sake of better understanding, we will use a published attack from the literature as a modeling example. In the following, brief descriptions of the attack details are given along with the SAM entity description used for representing the attack details. The full SAM metamodel is illustrated in Figure 2. Afterwards, the complete SAM model figure is shown on type-level (M1) in Figure 3.

We describe the attack in detail in the following. **The Tesla Remote Control Attack [33], [35], [37]**: This attack enables an adversary to break into the vehicle via the infotainment unit. The researchers of Tencent Keen Security Lab have demonstrated how to remotely control and steer the vehicle, how to disturb the autowipers and how to eliminate the lane detection of the vehicle.

Attack*: Represents a cyber-physical attack on the system described by an attack vector. An attack vector is a path or means by which an adversary can gain unauthorized access to a target system [20] or hurts one or more SecurityGoals. Attack vectors can be identified and extracted via attack trees. In an attack tree, child nodes are conditions which must be satisfied to make the direct parent node true. When the root is satisfied, the attack is complete. Typically, child nodes on the same level are linked with "OR" conditions. SAM uses a SubAttackGroup to also allow "AND" conditions and any other project-specific conditions between the grouped subattacks.*

The attack can be performed over a remote network connection. The vehicle user does not need to interact actively with the vehicle to allow for the attack to work. The privileges required for this attack are high. Also, impact on confidentiality, integrity and availability is high as well. As a result, an adversary can cause serious harm to passengers and other road users. As this attack is a research approach by security experts, the actual adversaries are not real-world attackers with bad intentions. They do have a knowledge-level, however, which they can provide for ill-minded attackers.

Adversary*: Attacks are performed by either an individual or the system's environment. Either way, adversaries are derivates of the system environment because they are not part of the main systems model and interact from the outside. An adversary can, however, come from within the system, e.g., from an unauthorized part or device.*

Environment*: This entity describes a collection of the environment functional descriptions. Many circumstances may be important for the attack description and a better understanding of the attack vector. Adversaries and security experts are conceptionally part of the environment. The Environment is not a newly introduced entity as it already exists in its own package, though it is extended due to the adversary's ability to use the environment for her attacks, e.g., external or real world attacks via adversarial examples [46], [47].*

In a real world scenario where adversaries with bad intentions perform the attack, the motivation behind such an attack would be to harm car occupants or other road users by crashing the vehicle. The attack therefore has a high safety relevance.

AttackMotivation*: An abstract representation of the adversary's motivations. This motivation is especially useful, when no other information, e.g., broken security goals, is available from the start. In that case, it offers an easy differentiation of the degree of severity and one can prioritise attacks according to their motivation. Moreover, it is fairly easy to find out if certain attack motivations are causing safety hazards, e.g., when tampering with safety-critical systems or modifying software-components related to the reliability of the system. The safety relevance can either be "High" (system failures), "Low" (fail-safe) or "None". It turns out that every single attack (or sub-attack) is part of a larger attack motivation. Through generalization methods we found out that there are only four higher motivations behind each attack: Harm, information retrieval, financial gain and product modification. There is also the motivation of prestige and other abstract ideals but those inherently cause consequences in at least one of the other motivations so they are not listed here. There is at least one AttackMotivation in an attack tree (its root). AttackMotivations collide with SecurityGoals. All attacks can be subsumed under one of those higher motivations:*

Harm*: A threat by an attack meant to actively or passively harm passengers and other road users, e.g., crashing the vehicle or causing a threat to other road users.*

InformationRetrieval*: A threat by an attack meant to, e.g., invade the privacy of passengers, other road users and other situational or political stakeholders, e.g., the OEM. Furthermore,*

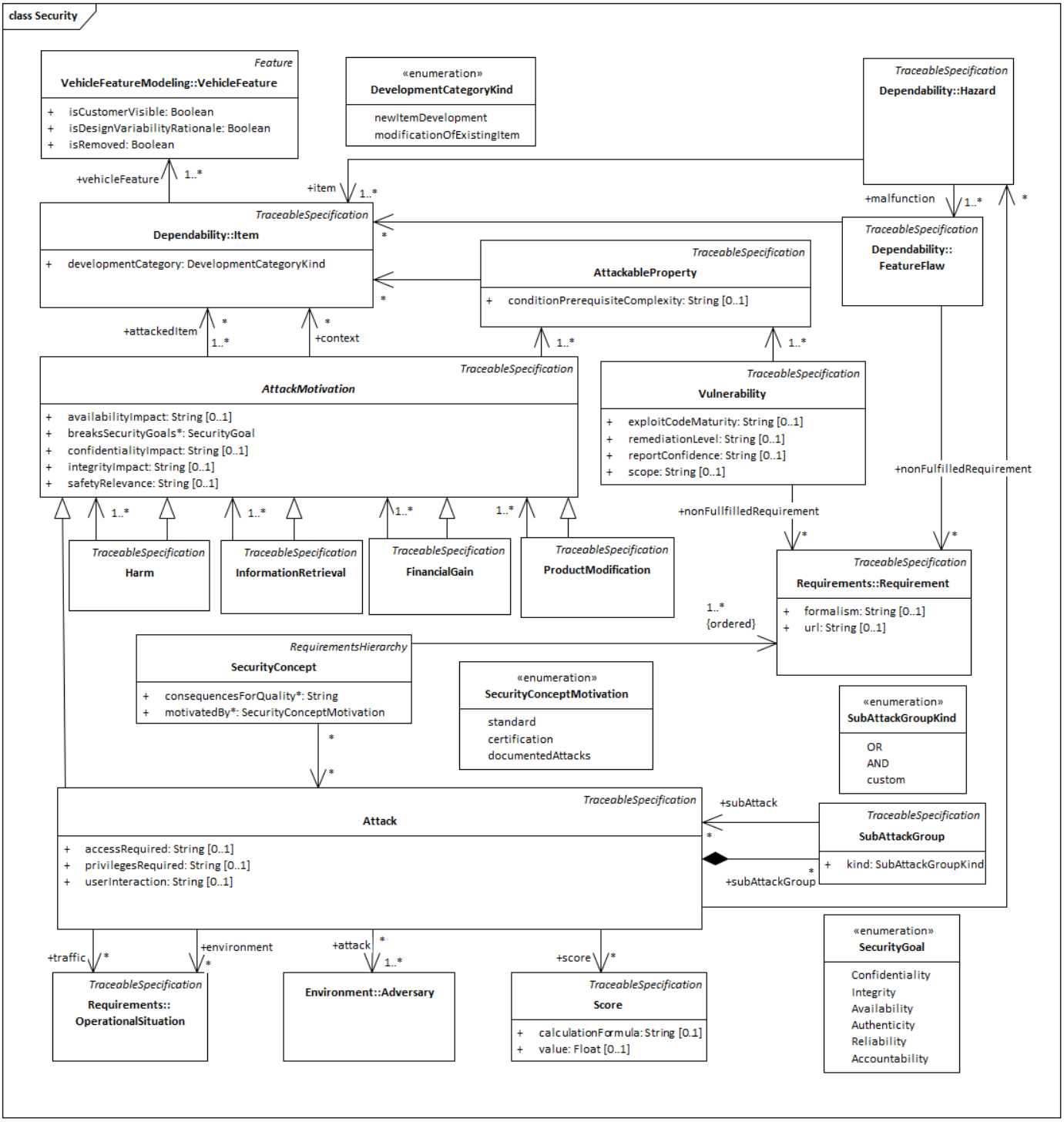

Figure 2. SAM Metamodel

getting access to other types of info, e.g., software/firmware by performing reverse engineering. Even research interest to break into the system for academic reasons is subsumed by this motivation.

***FinancialGain**: A threat by an attack meant to steal or cause financial or material gain for the adversary, service workshops or insurance companies. This usually leads to a financial loss for the owner or the OEM.*

***ProductModification**: A threat by tampering with the product's specification, e.g., getting more functionality out the car or tampering with the software in general, e.g., down-/upgrading or performance tuning.*

The affected item in the Tesla Remote Control Attack is the

AutoPilot ECU (APE).

Item: *The Item entity identifies the scope of safety/security information and the safety/security assessment. Safety/security analyses are carried out on the basis of an item definition and the safety/security concepts are derived from it.*

The exploited vulnerability in the Tesla attack is the Webkit browser framework of the infotainment unit, which offers the JSArray function and can be used for privilege escalation.

Vulnerability*: An abstract failure of a set of items, i.e., an inability to fulfill one or several of its requirements. In order to represent the weak spots in the system architecture, a vulnerability describes the weakness and affiliation to one or more Items. Vulnerabilites are concrete definitions of faulty software or configurations and requirements must be derived from them. Vulnerabilites have a scope. The scope of a vulnerability is changed, if a successful attack affects more security goals and vulnerabilities, i.e., enabling follow-up attacks.*

In case of a successful attack, adversaries can cause functional safety hazards by tampering with or disabling safety-critical functions of the Tesla vehicle.

Hazard*: The hazard metaclass represents a condition or state in the system that may contribute to accidents. The hazard is caused by malfunctioning behavior of E/E safety-related systems including interaction of these systems.*

The exploited vulnerability changes the scope of the attack, meaning that all six security goals are broken if the attack is performed successfully.

SecurityGoal*: This entity offers enumerations for common security goals [48] across any communication or data flow. These goals are: Confidentiality, Integrity, Availability, Authenticity, Reliability and Accountability.*

The exploited vehicle feature is Tesla's Autopilot, used for autonomous driving. In this case there exists a 1:1 relationship between item and vehicle feature.

VehicleFeature*: Provided by the Dependability package, a VehicleFeature represents a special kind of feature intended for use on Vehicle Level. Items consist of a set of VehicleFeatures.*

The JSArray function is the attackable property the adversary is looking for, i.e., his anchor of the attack. Attackable properties are concrete characteristics that describe the potential attack surface, e.g., if they have known security bugs and flaws like the JSArray function.

AttackableProperty*: Characteristics or certain properties of Items an adversary searches / needs for his attack to succeed, e.g., wireless communication capabilities, used ciphers, features, etc. If exploited, attackable properties allow the adversary to successfully perform an attack and define a vulnerability.*

After analysing the attack properties via the CVSS metrics, one can calculate the base score and temporal score of the attack and derive the requirement: code signing protection for over-the-air (OTA) updates.

Score*: Score is the entity for attack rating. SAM allows for any generic type of scoring system. Properties of other entities will provide all the relevant information that are needed for attack rating. The attribute calculationFormula describes which scoring system is used, e.g., CVSS, SecL, etc. Alternatively, an empirical value or expert opinion can be given if this attribute is left empty. Section V-C will explain scoring systems for SAM in broader detail.*

Requirement*: To define requirements to fix vulnerabilities, a so-called requirement is the packed result of lesson's learned and is derived from an attack. It represents a capability or condition that must (or should) be satisfied.*

The described attack is only possible when the vehicle is "Slow or Standing". Otherwise, the vehicle does not allow to use the Webkit browser. The Tesla remote control attack is possible in any operational situation of the vehicle. Once the adversary has gained full access over the system she can fully control the system over the CAN bus.

OperationalSituation*: In security modeling, it is often beneficial to know about this situation, e.g., whether the car is standing, driving or in parking mode. An operational situation is a state, condition or scenario in the environment that may influence the vehicle. It may be further detailed by a functional definition in the EnvironmentModel. Examples are: "Driving on highway", "Driving in city", "In reverse gear", "Parking", "Any", etc.*

SAM has no explicit specifications for a security concept. However, SAM proposes Common Criteria (CC) ISO/IEC 15408 protection profiles [49] as a possible solution. Common Criteria is an established standard in the security domain to provide guidance during the development of dependable systems.

SecurityConcept*: Represents the set of security requirements that together fulfill at least one SecurityGoal. An exemplary structure and classification of respective security requirements can be found in Common Criteria (CC) ISO/IEC 15408. Ideally, the security concept is motivated by analyses of the documented attacks connected to the respective item (in this case, the motivatedBy property is set to "documentedAttacks"). Otherwise, security concepts can just as well be motivated by standard or certification demands (then, the motivatedBy property is set to either "standard" or "certification".)*

SecurityConceptMotivation*: This entity offers enumerations for motivations of security requirements. These motivations are: "standard", "certification" and "documentedAttacks".*

B. Methodical Context for SAM

In order to protect and defend a system from attacks and threats it is necessary to identify and classify these threats first. The categorization of AttackMotivations already creates methodological benefits with regard to the identification of attacks. Systematic security analyses can be used to quantify the required effort for a potential attack. There is a constant battle between the attacker's efforts and the layers of security devised by system engineers. Because no system can be completely secured against all sorts of attack, system engineers compromise on varying levels of security abstractions to reach an acceptable

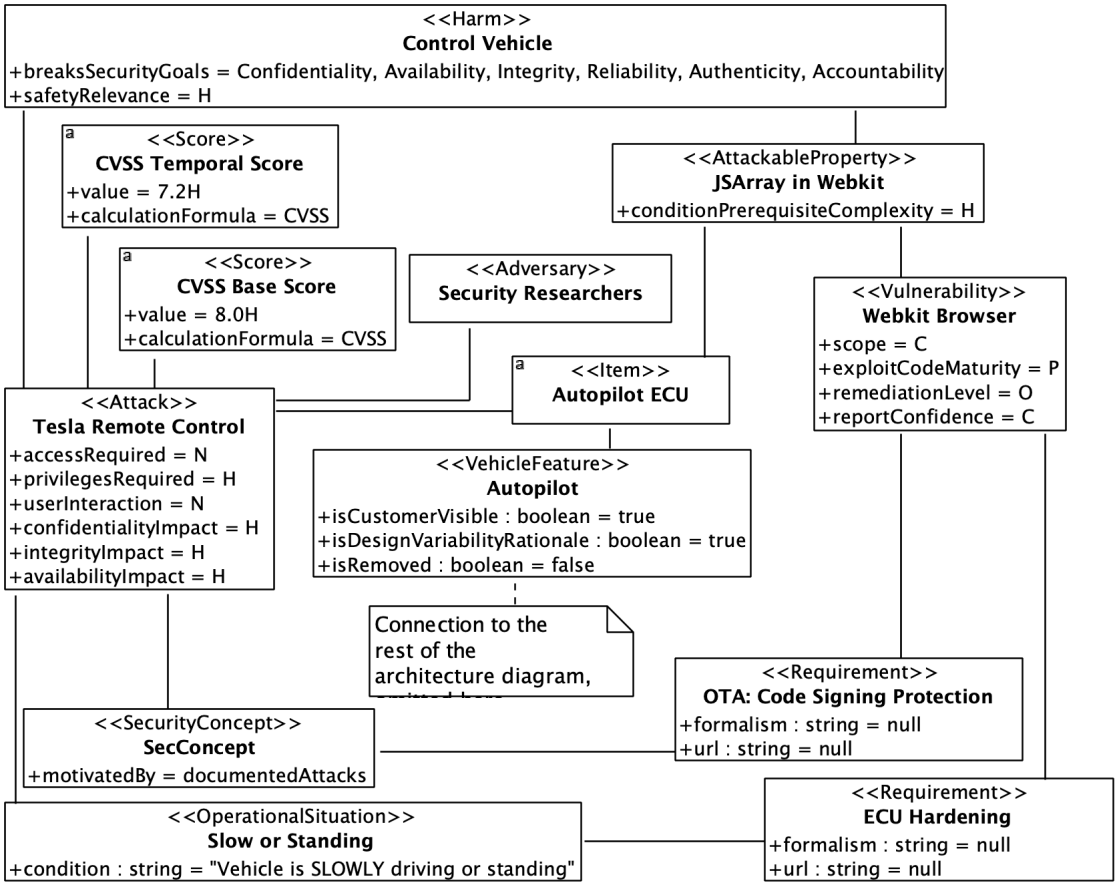

Figure 3. SAM model of Tesla Remote Control Attack—CVSS v3.0 Vector String: CVSS:3.0/AV:N/AC:H/PR:H/UI:N/S:C/C:H/I:H/A:H/E:P/RL:O/RC:C

degree of security. Hence, any security system ultimately results in a trade-off.

Although SAM does not instill security in the system design, it enforces reflection about attacks and their consequences for the system, ideally as a collaboration between system engineers and security experts. While SAM's metalevel is rather abstract, its application becomes concrete on metalevel M1. Notice that the multiplicity from AttackMotivation to Item is 1..* to 1..*, requiring the system engineer to describe at least one attack motivation for every item of the automotive system. This is an important methodical support for the discovery of threats. If a single item has no associated motivation for an attack, increased caution is required, e.g., because no attack against the item is known yet. In this case, system engineers might simply desist to scrutinize an item for possible attack motivations. With the 1..* multiplicity, however, they are forced to think about at least one attack motivation for every item. Therefore, the reason for this decision is to eagerly enforce the methodology of SAM's security approach.

The main difference between safety risks and security threats is that security threats do not happen at random (i.e., they are not bound by probability) but always occur in worst-case scenarios. For safety hazards, a statistical probability can be assumed. Cyber attacks are performed by an intelligent attacker at the most suitable time for the adversary and at the lowest defense barrier. Furthermore, it can be misleading to confuse safety goals with security goals. Security threats, however, can cause safety hazards and vice-versa. Though it is not recommended to treat them in the same way during the system design phase for reasons mentioned above. Additionally, text annotations are bad practice. Usually, the transfer from annotations in natural language is imprecise and the original intent of the security experts, which is needed to represent the system model and its security mechanisms accordingly, might be lost during the transfer. An extensive reuse of security solutions can be established by embedding SAM in the "Dependability" package of EAST-ADL and the subsequent integration into AUTOSAR. This makes it possible to keep the development effort at a minimum and to implement comprehensive safety and security solutions in a wide range of applications in the vehicle.

SAM offers the possibility to model socio-technical systems by providing the modeling entity `Adversary`. Security goals need to be fulfilled in a *socio-technical context* or a *socio-technical system*. The definition of a socio-technical system is an organized group of humans and connected technologies, which are constructed in a certain manner to produce a specific result [48]. Nevertheless, trying to improve security simply by adding cryptography to the system is a fallacy. At best, cryptography can ensure confidentiality but cannot cover security goals like availability, reliability or accountability. With our approach, we offer co-engineering processes of security and safety for automotive software engineering (security and safety by design).

C. Using Generic Scoring Systems for SAM

The latest major release of SAM [1] introduced many new attributes to the modeling entities, which allow for using well-known security scoring systems like CVSS [21]. In order to be able to keep SAM up-to-date and gain some flexibility by not making a strong commitment to one particular system we designed SAM to use any generic scoring system. When modeling attack scenarios, users of SAM can choose among their favorite. Here, we will use the CVSS. The latest version of SAM is available open source [44]. The architecture description has been completed to the extent that common scoring systems are now able to find the necessary information and thus perform their analyses. Inspired by the CVSS, which is an acclaimed industry standard for rating vulnerabilities in computer systems, we added new attributes to some of SAM's entities. The CVSS proposes three different metric groups for calculating the vulnerability scores. In the following, an explanation of the interplay between SAM and the metrics is given. The assignment of the attributes to the meta entities and partly their naming does not come from CVSS, but was developed by the athors.

The Base Metric Group reflects the intrinsic properties of Attack: from SAM's automotive-oriented perspective, this group therefore indicates the characteristics that result if the attack in question is aimed at the automotive domain in general. The entity AttackableProperty refers to the properties of the attacked item that are beyond the control of the attacker and must exist in order to exploit the vulnerability, for example, in the case of a side channel attack, the use of shared caches within a multicore system. The attribute `conditionPrerequisiteComplexity` ("Low" and "High") in the AttackableProperty refers to the complexity of encountering or creating such conditions. For example, in the case of the side channel attack mentioned above, the `conditionPrerequisiteComplexity` is "Low" because shared caches are to be expected nowadays. It would be "High" if the attack made it necessary for all tasks on all cores to use one single common cache. When evaluating this property, all user interaction requirements for exploiting the vulnerability must be excluded (these conditions are recorded in the property `privilegesRequired` of Attack instead). If the `conditionPrerequisiteComplexity` is "Low", the attack is more dangerous than if the `conditionPrerequisiteComplexity` is "High". The property `privilegesRequired` describes the level of privileges an attacker must possess before successfully exploiting the vulnerability. This metric is greatest if no privileges are required. Also, the Attack entity has been extended with the attributes `accessRequired` and `userInteraction`. The attribute `accessRequired` describes the context in by which vulnerability exploitation is possible. It must not be confused with general attack vector handling in SAM, which describes the path from attack motivation of an attack tree to one of its leafs. Whether the user or driver of the vehicle needs to interact with the system in a certain way, e.g., by pressing a button, is captured in `userInteraction`. Attacks that do not require any user interaction increase the score of the attack. **The Temporal Metric Group** allows for adjustment of the score after more information of the exploited vulnerability is available. If, for example, `exploit code` has been published or the `report confidence` of a vulnerability is confirmed, the temporal score rises. In SAM, temporal metrics are part of the vulnerability. **The Environmental Score Metrics** additionally enable the general CVSS Score (resulting from the Base Metric Group) to be adapted to the specific (automotive) company. The metrics are the modified equivalent of the base metrics weighting properties related to the concrete company's infrastructure and business risk. SAM offers a fully comprehensive basis to analyse the CVSS Base Metric Group, which means that SAM can also be used to evaluate the Environmental Metric Group. Environmental Metrics do not require any additional information beyond the Base Metrics, but merely a readjustment of the analysis perspective towards the concrete company. This means that the security scoring analysis can be carried out entirely by an analyst based on the available information provided by SAM.

This allows for more flexibility and SAM does not have to be adapted for any future CVSS updates. All attributes used for attack assessment are of the type String. This allows for SAM to be used with generic assessment techniques and is not tightly coupled with the CVSS attribute descriptions. In the model itself, or from the model itself, a CVSS score cannot be calculated automatically anyway. Doing so would happen in a behaviour model while SAM models are structure models. But if a security analyst is familiar with the CVSS, she will be able to calculate the CVSS score with all the information that is provided by the structure model. It is therefore still possible to find related information about the attribute types ("High" and "Low", etc.) in the notes of the meta model, but does not lead to problems in case of non-compliance.

The additional benefit of having SAM models compared to directly giving the properties and a vulnerability score is that not only the CVSS (or scoring systems in general) is used, but also the possibility to construct attack trees via sub-attacks and follow-up attacks. SAM is also a method for hierarchical processing of attack vectors. In terms of substance, this goes beyond the classic attack rating. SAM makes the scoring system available to the software architect or in other words: SAM's strength lies in its ability to integrate with existing automotive architectures. What is brought together are architectural considerations with pure security considerations as regards the attack itself (attack vectors that can be derived from it, motivations, target areas) and all scoring systems that are known, which can derive all necessary information from the properties.

VI. Evaluation

To prove that SAM is feasible, we have evaluated our solution approach through "Grounded Theory" [50] interviews with two experts (in the following we will refer to them as E1 and E2) of two different automotive software companies from the industry. Both interview partners have a professional background in automotive systems engineering and/or embedded security. The interviews were structured as follows: First, we asked some general questions about automotive security modeling and current processes in the analysis, design, implementation and test phase. Afterwards, the authors gave a brief introduction and explanation of SAM. During the presentation, a real-world attack was shown and illustrated to show how to use SAM. Together with the interview partners, we jointly created a reference SAM model for a real-world usecase. Figure 4 illustrates a Function Unlock Attack. The

attack describes an attack vector that allows an adversary (here: a third party workshop) to unlock specific vehicle functions for the driver of the vehicle via tampering with the TimeServer item. The answers for the remainder of our interview were related to this modeling example. Finally, the industry experts were asked questions about the five categories: suitability, scalability, comprehensibility, completeness and tool support.

A. General Findings

Our interviews have shown that there is no clear process for security engineering, yet. Companies are in the process of creating them. Hopes are that the process will be similar to the ISO 21434 and ISO 26262 standards for safety, which is currently seen as a general guideline for automotive engineers. The SAE J3061 standard would also be a good starting point if engineers want to get started with this matter. Our interviews show that a detailed development process for security requirements is highly desired. Many projects in the industry are done by service providers, where the instructed companies integrate security into the finished standard components. On the part of Tier 1 there are also no concrete specifications for the security development process

Both experts confirmed that they are indeed working according to the V-Model. They have quite clear process models. Ideally, they get a specification, write a requirement specification, create an architecture for the software, create module specifications, derive tests from them, test the modules, then test what is integrated. Depending on what needs to be developed in the end, they then develop a piece of software or an entire ECU. Afterwards, they go up (in the V-Model) until they have finally tested all the specification requirements. They do this for all requirements.

For some requirements, security is not in the focus. Even if security is mentioned at all, it is only given implicitly by implementation instructions or which software to use. Putting a security concept in place especially for all phases of the V-Model would be desirable, even though integration might be challenging.

B. Suitability

We asked the experts how existing processes could use SAM in their current workflow or if it could even be integrated into their processes. We also explained the intention of SAM to facilitate the exchange between security experts, software architects and software engineers and asked, if it serves this purpose. We tried to find out if SAM is a suitable solution for the industry and whether or not it solves current industry questions. Finally, for this category, we wanted to know if SAM is ready for the development of autonomous vehicle systems and what could be missing. The following is a summary of the experts' answers:

Both experts agreed that they could use SAM because it is a good way of making attacks comparable. A missing process is that items have to be seen in a way that potentially there has to be an attack for each item at the time of module specification. The main problem is that the process is actually driven only once during development. Attacks, however, only happen during maintenance. A visualization (like SAM models) would help, however. A possible solution of integration SAM would be using SAM models like bug reports for security. With SAM, they would already have the constraints available. The current process currently only entails development and delivery. Teams would need a kind of "response team for security". Someone who monitors the whole systems and knows what kind of hardware-software combinations are rolled out and what kind of errors there are, e.g., SPECTRE [51]. This could suddenly affect a huge portion of products that are out in the field. The response team could classify attacks like this with SAM.

Integrating SAM into workflows is rather difficult for service providers, but on OEM side it would very well possible. Analogue to considering ASIL levels, i.e. according to ISO 26262, SAM could be well integrated on the OEM side. SAM increases transparency and simplifies understanding attacks. After a training of the team members, SAM would definitely help and make communication easier, especially because there are no alternatives. It can be, however, that one could not understand a bad SAM model without additional textual description.

Regarding suitability for autonomous driving, the experts were unsure if this is even possible to answer at this point. SAM would first have to prove itself in practice and people would have to work with it so that it could be finally used. Even if systems would become simpler, software development would probably become more complex, because there would be many more lines of code. Moreover, security has to guarantee that the safety mechanisms still work and that there is a balance of availability, security and safety. If a car goes into fail-safe mode with the slightest security suspicion, autonomous driving is not possible in this scenario.

According to one expert, some modeling entities for machine learning components could be useful. This is rather challenging, though, as machine learning components are black boxes.

C. Comprehensibility

We asked the industry experts if something about SAM is difficult to understand. Although they understood everything, an introduction with examples would help beginners getting started, because SAM is not that formal, rather practical.

D. Scalability

For this category we evaluated for what complexity or size of a software project SAM is best suited and how developers would ideally use SAM in practice.

Here, the industry partners disagreed. Although one could easily create multiple SAM models for more complex scenarios, one expert believes that SAM might not scale well for bigger projects. There would be no way to see the overall security for, e.g., a whole vehicle. To scale it, one would have to link several SAM models together. Overall models could get confusing because everything always depends on one item and vehicles are very complex. However, he said, SAM would be the right approach for smaller projects.

According to one expert, automotive engineers definitely have to allocate time for security engineering. SAM would help with that, because in the end, it would save time if everyone speaks the same language. There would have to be support for security techniques. Central offices that have an overview of which products, which software and which hardware are currently in the field could use SAM to determine that.

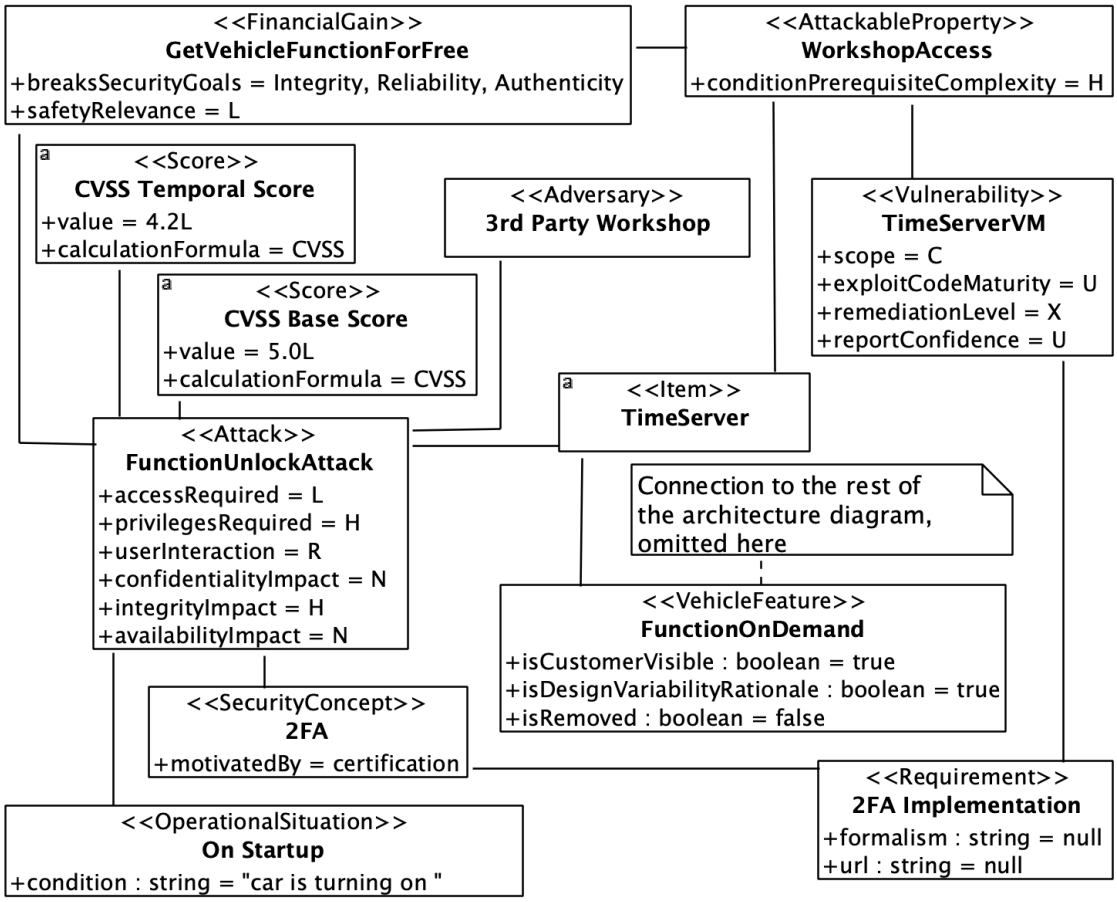

Figure 4. SAM model of Function Unlock Attack—CVSS v3.0 Vector String: CVSS:3.0/AV:L/AC:H/PR:H/UI:R/S:C/C:N/I:H/A:N/E:U/RC:U

In contrary, developers might not want to use SAM because it is on a relatively high level of abstraction. Fortunately, SAM has close familiarity to UML and developers could get used to it quickly. Traceability over requirements would be a good approach to make them adopt SAM, because they work very strongly according to requirements and the V-Model and they can validate their requirements.

E. Completeness

To make sure that SAM is indeed complete, we asked if the explanations and attributes are complete or what could be deleted from SAM. Moreover, we asked what a possible solution would look like for how to add tests to SAM.

During the discussion with the experts it became clear that some entities cover redundant information at that time. The current version presented in this paper, however, includes the latest changes, even the refined changes taken into account after the interviews. Furthermore, one expert had the opinion that the score entity might be redundant because all the relevant information is already in the entities' properties and could also be calculated that way. This is theoretically redundant information. If one would have a modeling tool now, it would display it permanently and calculate it in the background. It would not have to exist as an entity. As of right now, it is shown in its own separate entity.

Overall feedback showed that if users of SAM are already familiar with security, one could understand all the terms and explanations presented in SAM. SAM is doing a good job of illustrating the weak points. So one can deduce test cases from the model, make them more concrete and test countermeasures to achieve a certain CVSS score. One could use the score to adapt the tests and cover the fact that, for example, an attack is no longer possible and, if necessary, derive further test cases. However, SAM would actually only show the problem to engineers: the attack scenario. The resulting actions would be twofold: Fixing and testing the bug. After the bug is found, something in the architecture must be changed. Reusability is an important factor. Someone would have to test the change and then has to make sure that this bug will not happen again in future models or products. Someone could consider whether he can make the test abstract so that it will be fired in the future. However, one would need a lot more details to automatically turn it into a test. The tester should, however, be able to look at the model and write a test from it himself.

F. Tool Support

There is no seamless tool support from SAM/EAST-ADL to AUTOSAR as of right now. We asked whether or not it is needed. Furthermore, we asked if an automatic generation of SAM models (e.g., from tests or code) would be useful. Finally, we asked for remaining comments, hints and feedback.

One expert was not sure if SAM must fit directly to AUTOSAR. SAM would be useful for modeling attacks, AUTOSAR is used for modeling software. Tool support, where

one could create SAM models with, or where the score is calculated automatically would be helpful, though, and improve usability. If SAM is easy to use and a suitable tool would exist, developers would use it for sure.

The other expert disagreed and told us that an automatic generation of SAM models would not be necessary at the moment. He also said that the industry does not know whether it will stay with AUTOSAR or not. However, SAM can be used independently at the moment. His experience with AUTOSAR told him that he would not build too much onto it. One could automatically generate or update SAM models as post-processing after modeling, just as one can already generate code or header files, templates, etc. Leaving SAM the way it is right now, i.e., without automatic generation, someone has to think about security himself. It has to be thought of from the beginning.

General advice we received, suggested a "Getting Started" or "HowTo" document. This would be helpful for acceptance in practice. Finally, SAM should be used more on the OEM side, because vehicles are very complex products and only the product owner currently have a complete overview over the vehicle's components. With SAM, someone can sufficiently document security thoughts and research, including evaluation.

G. Overall Results

Our industrial evaluation has shown that SAM is indeed suitable as a solution approach and integration of the methods used into industry processes is feasible. Moreover, SAM is easy to understand according to our interviewed industry experts. Although SAM might not scale for bigger software projects, it establishes a process to get started with on a smaller scale. Also, our evaluation shows that SAM is indeed complete and offers enough tools, methods and descriptions for threat modeling and attack rating. Evaluation results regarding tool support were twofold. Further investigations on a practical tool support are necessary.

Table II shows a summary of the evaluation results. The symbol keys are explained in Table I.

TABLE I. Symbol key for the evaluation results table

Symbol	*Meaning*
++	High agreement or interest (E1 and E2 agreed)
+	Notable agreement or interest (E1 or E2 agreed)
o	Moderate agreement or interest (neither E1 or E1 showed interest)
-	Limited agreement or interest (E1 or E2 disagreed)
--	Low agreement or interest (E1 and E2 disagreed)
+/-	Mixed agreement or interest (E1 and E1 were of different opinion)

TABLE II. Summary of the evaluation results

Categories and codes	*Results*
General	
Process: Along the V-Model	++
Missing process for security	++
Suitability	
Processes can use SAM	++
Integration in process	++
Exchange between security experts, architects and engineers	++
Solves relevant industry challenges	o
Ready for autonomous driving (AD)	+
Is something missing for AD?	+/-
Comprehensibility	
SAM is easy to understand	++
Scalability	
Scales for all sizes of projects	+/-
Developers have time to use SAM	+/-
Completeness	
Nothing should be removed from SAM	o
Descriptions and entities are complete	++
Use SAM for testing	++
Tool Support	
AUTOSAR tool support	+/-
Automatic generation of SAM models	--

VII. Conclusion and Future Work

We have presented a detailed description of the Security Abstraction Model, including all of its metamodel entities. We have indications that the approch is feasible. The security technique has been evaluated with industry experts and a grounded theory analysis. The resulting analyses of the evaluation show that SAM puts the security-by-design principle into practice by enabling collaboration between automotive system engineers and security experts. Future work will concentrate on the bottom-up approach, i.e., improving embedded security and network security on the application layer and cryptographic protocol design, e.g., utilizing PHE. Next steps need to develop automotive software solutions to actually be included in the security concept. Our research focuses particularly on a PHE authentication scheme for secure authentication in autonomous car sharing scenarios and fingerprint entry systems. Our work aims to support security by design in the automotive industry and SAM offers the necessary insights and fundamentals to continue conducting relevant research in this domain.

Acknowledgment

This work is funded by the Bavarian State Ministry of Science and the Arts in the framework of the Centre Digitisation.Bavaria (ZD.B).

M.Z. was supported by the BayWISS Consortium Digitization.

References

[1] M. Zoppelt and R. Tavakoli Kolagari, "UnCle SAM : Modeling Cloud Attacks with the Automotive Security Abstraction Model," in CLOUD COMPUTING 2019, The Tenth International Conference on Cloud Computing, GRIDs, and Virtualization, Venice, Italy, 2019, pp. 67–72.

[2] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-Vehicle Networks: A Review," pp. 534–545, 2015.

[3] W. Zeng, M. A. Khalid, and S. Chowdhury, "In-vehicle networks outlook: Achievements and challenges," IEEE Communications Surveys and Tutorials, vol. 18, no. 3, 2016, pp. 1552–1571.

[4] ISO/IEC, "ISO/IEC 15408-1:2009 - Evaluation Criteria for IT Security," vol. 2009, 2009, p. 64.

[5] A. Happel and C. Ebert, "Security in vehicle networks of connected cars," 15. Internationales Stuttgarter Symposium: Automobil- und Motorentechnik, no. March, 2015, pp. 233–246.

[6] M. Zoppelt and R. Tavakoli Kolagari, "What Today's Serious Cyber Attacks on Cars Tell Us: Consequences for Automotive Security and Dependability," in International Symposium on Model-Based Safety and Assessment, M. Papadopoulos, Yiannis and Aslansefat, Koorosh and Katsaros, Panagiotis and Bozzano, Ed., Springer. Springer International Publishing, 2019, pp. 270–285. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-32872-6_18

[7] M. Zoppelt and R. Tavakoli Kolagari, "SAM: A Security Abstraction Model for Automotive Software Systems," in Security and Safety Interplay of Intelligent Software Systems. Springer, 2018, pp. 59–74.

[8] H. Blom, H. Lönn, F. Hagl, Y. Papadopoulos, M. Reiser, C. Sjöstedt, D. Chen, and R. Tavakoli Kolagari, "EAST-ADL–An Architecture Description Language for Automotive Software-Intensive Systems–White Paper Version 2.1. 12," Hyperlink: http://www.maenad.eu/public/conceptpresentations/EAST-ADL_WhitePaper_M2 [retrieved: December 2018], vol. 1.

[9] ISO/SAE, "ISO/SAE CD 21434 - ROAD VEHICLES – CYBERSECURITY ENGINEERING," https://www.iso.org/standard/70918.html.

[10] W. Dröschel, W. Heuser, and R. Midderhoff, Inkrementelle und objektorientierte Vorgehensweise mit dem V-Modell 97. München: Oldenbourg, 1998.

[11] SAE, "SAE J 3061 - Cybersecurity Guidebook for Cyber-Physical Vehicle Systems," https://www.sae.org/standards/content/j3061/.

[12] N. Bißmeyer, S. Mauthofer, J. Petit, M. Lange, M. Moser, D. Estor, M. Sall, M. Feiri, R. Moalla, M. Lagana, and F. Kargl, "PREparing SEcuRe VEhicle-to-X Communication Systems," 2014.

[13] O. Henniger, L. Apvrille, A. Fuchs, Y. Roudier, A. Ruddle, and B. Weyl, "Security requirements for automotive on-board networks," in 2009 9th International Conference on Intelligent Transport Systems Telecommunications, ITST 2009. IEEE, 2009, pp. 641–646.

[14] H. Holm, M. Ekstedt, T. Sommestad, and M. Korman, "A Manual for the Cyber Security Modeling Language," 2013, p. 110.

[15] J. Jürjens, "UMLsec: Extending UML for Secure Systems Development," in International Conference on The Unified Modeling Language. Springer, 2002, pp. 412–425.

[16] INCOSE, "Systems Engineering Handbook," in Systems Engineering, no. August, 2000.

[17] R. Ross, M. McEvilley, and J. Carrier Oren, "Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems," vol. 160, no. November 2016, 2016.

[18] J. Lee, B. Bagheri, and H.-a. Kao, "A Cyber-Physical Systems architecture for Industry 4 . 0-based manufacturing systems," Manufacturing Letters, vol. 3, 2015, pp. 18–23.

[19] S. P. Kadhirvelan and A. Söderberg-Rivkin, "Threat Modelling and Risk Assessment Within Vehicular Systems," Chalmers University of Technology, no. August, 2014, p. 52.

[20] V. L. Thing and J. Wu, "Autonomous Vehicle Security: A Taxonomy of Attacks and Defences," in Proceedings - 2016 IEEE International Conference on Internet of Things; IEEE Green Computing and Communications; IEEE Cyber, Physical, and Social Computing; IEEE Smart Data, iThings-GreenCom-CPSCom-Smart Data 2016, 2017, pp. 164–170.

[21] Common Vulnerability Scoring System [retrieved: April, 2019]. [Online]. Available: https://www.first.org/cvss/

[22] G. Macher, A. Höller, H. Sporer, E. Armengaud, and C. Kreiner, "A combined safety-hazards and security-threat analysis method for automotive systems," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9338, 2015, pp. 237–250.

[23] Bosch, "CAN Specification," Robert Bosch GmbH, 1991.

[24] C. Miller and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle," Defcon 23, vol. 2015, 2015, pp. 1–91.

[25] W. Diffie and M. Hellman, "New directions in cryptography," IEEE transactions on Information Theory, vol. 22, no. 6, 1976, pp. 644–654.

[26] R. W. F. Lai, C. Egger, M. Reinert, S. S. M. Chow, M. Maffei, and D. Schröder, "Simple Password-Hardened Encryption Services," in 27th {USENIX} Security Symposium ({USENIX} Security 18). Baltimore, MD: {USENIX} Association, 2018, pp. 1405–1421. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/lai

[27] F. D. Garcia, D. Oswald, T. Kasper, and P. Pavlidès, "Lock It and Still Lose It—On the (In)Security of Automotive Remote Keyless Entry Systems," Proceedings of the 25th USENIX Security Symposium, 2016, pp. 929—-944.

[28] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," Computer, vol. 50, no. 7, 2017, pp. 80–84.

[29] C. Valasek and C. Miller, "Adventures in Automotive Networks and Control Units," Technical White Paper, vol. 21, 2013, p. 99.

[30] C. Miller and C. Valasek, "CAN Message Injection," 2016, pp. 1–29. [Online]. Available: http://illmatics.com/can message injection.pdf

[31] can-utils repository on GitHub [retrieved: April, 2019]. [Online]. Available: https://github.com/linux-can/can-utils

[32] T. Bécsi, S. Aradi, and P. Gáspár, "Security issues and vulnerabilities in connected car systems," in 2015 International Conference on Models and Technologies for Intelligent Transportation Systems, MT-ITS 2015, 2015, pp. 477–482.

[33] S. Nie, L. Liu, Y. Du, and W. Zhang, "Over-the-Air : How We Remotely Compromised the Gateway , Bcm , and Autopilot Ecus of Tesla Cars," Defcon, vol. 1, 2018.

[34] T. Zhang, H. Antunes, and S. Aggarwal, "Defending connected vehicles against malware: Challenges and a solution framework," IEEE Internet of Things Journal, vol. 1, no. 1, 2014, pp. 10–21.

[35] Tencent Keen Security Lab, "Experimental Security Research of Tesla Autopilot," 2019, p. 38.

[36] C. Smith and S. Francisco, THE CAR HACKER ' S HANDBOOK A Guide for the Penetration Tester About the Contributing Author About the Technical Reviewer, 2016.

[37] S. Nie, L. Liu, and Y. Du, "Free-fall: hacking tesla from wireless to can bus," Defcon, 2017, pp. 1–16.

[38] Tencent Keen Security Lab, "Experimental Security Assessment of BMW Cars: A Summary Report," 2018.

[39] J. den Herrewegen and F. D. Garcia, "Beneath the Bonnet: A Breakdown of Diagnostic Security," in European Symposium on Research in Computer Security. Springer, 2018, pp. 305–324.

[40] Y. Zhang, B. Ge, X. Li, B. Shi, and B. Li, "Controlling a Car Through OBD Injection," in Proceedings - 3rd IEEE International Conference on Cyber Security and Cloud Computing, CSCloud 2016 and 2nd IEEE International Conference of Scalable and Smart Cloud, SSC 2016, 2016, pp. 26–29.

[41] I. Foster, A. Prudhomme, K. Koscher, and S. Savage, "Fast and Vulnerable: A Story of Telematic Failures," 9th USENIX Workshop on Offensive Technologies (WOOT 15), 2015.

[42] A. Palanca, E. Evenchick, F. Maggi, and S. Zanero, "A stealth, selective, link-layer denial-of-service attack against automotive networks," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), M. Polychronakis and M. Meier, Eds. Cham: Springer International Publishing, 2017, vol. 10327 LNCS, pp. 185–206.

[43] T. Ring, "Connected cars - The next target for hackers," Network Security, vol. 2015, no. 11, 2015, pp. 11–16.

[44] SAM repository on Bitbucket [retrieved: April, 2019]. [Online]. Available: https://bitbucket.org/east-adl/sam

[45] AUTOSAR Enabling continuous innovations https://www.autosar.org [retrieved: July, 2019]. [Online]. Available: https://www.autosar.org/

[46] J. Hayes and G. Danezis, "Machine Learning as an Adversarial Service: Learning Black-Box Adversarial Examples," vol. 2, 2017.

[47] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in Machine Learning: From Phenomena to Black-Box Attacks Using Adversarial Samples," 2016.

[48] F. Dalpiaz, E. Paja, and P. Giorgini, "Security requirements engineering via commitments," in 2011 1st Workshop on Socio-Technical Aspects in Security and Trust (STAST). IEEE, 2011, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6059249

[49] H. C. A. Van Tilborg and S. Jajodia, Encyclopedia of Cryptography and Security. Springer Science & Business Media, 2011. [Online]. Available: http://link.springer.com/10.1007/978-1-4419-5906-5

[50] B. G. Glaser, A. L. Strauss, and E. Strutzel, "The discovery of grounded theory; strategies for qualitative research." Nursing research, vol. 17, no. 4, 1968, p. 364.

[51] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in 40th IEEE Symposium on Security and Privacy (S&P'19), 2019.