Approaches For Automating Cybersecurity Testing Of Connected Vehicles

Stefan Marksteiner, Peter Priller, and Markus Wolf

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

1 Introduction

Mobility is a high priority in our society. Statistics report global annual car sales between 60 and 75 million[1] during recent years. According to the European Automobile Manufacturers' Association (ACEA), just in Europe approximately 350 million cars are currently in use [8], and the number grows to beyond 1 billion for a worldwide estimation. Cars are ubiquitous, for many families and businesses around the world, since decades. What has changed, however, is the fact that today's vehicles

Stefan Marksteiner
AVL List GmbH, Graz, Austria and Mälardalen University, Västerås, Sweden e-mail: stefan.marksteiner@avl.com

Peter Priller
AVL List GmbH, Graz, Austria e-mail: peter.priller@avl.com

Markus Wolf
AVL List GmbH, Graz, Austria e-mail: markus.wolf@avl.com

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

have become complex IT systems, often also called ”computers on wheels”. Modern cars run 100+ million lines of source code (MLOSC), and host complex computer networks both internally (in-vehicle networks) and externally. Many modern cars are now connected via the Internet to (maybe even multiple) cloud services, as well as to cellular networks (3G, LTE, 5G), and to specific vehicular networks (also known as car-to-car (C2C) or vehicle-to-everything (V2X), like ITS-G5). And that's not all: most vehicles also provide local networking capabilities (also called personal area networking, PAN). Typically based on WIFI and Bluetooth, it is used to connect to users' personal devices like smart phones and tablets, or to their home WLAN. To complement that already impressive array of wireless communication interfaces, some car manufacturers (or Original Equipment Manufacturers - OEMs) might add ultra-wide band (UWB) radios to communicate with car access systems like owner's keys or keycards. In addition, advanced driver assistance systems (ADAS) and future fully automated driving (AD) capabilities add GNSS receivers (Global Navigation Satellite System), TMC receivers (Traffic Message Channel), and active radar systems. Modern vehicles combine deeply complex software with exposure to a wide range of wireless networking technologies to both public and closed networks). In cybersecurity, this is called opening a large attack surface. This is worsened by the fact that vehicles are exposed for a much longer time than, e.g., personal computers (PC) or mobile devices like smart phones. Cars are in operation for some 15 years and more, which increases the threat that a vulnerability is found, shared, and at some point in time exploited by an attack. With such significant high exposure, let's consider potential threats which could evolve from malign attacks. Vehicles are highly dynamic (by nature), provide high levels of energy (storing 100kWh and more), are valuable (sometimes beyond 100k€) and exist as worldwide accessible objects in public, thus unrestricted places. When exploited by an attack taking over remote control, vehicles could become dangerous weapons, for both passengers inside, and for other road participants. Worse, if groups of vehicles would come under attacker's control, they could be used to stage threats on city or even national level. State-sponsored attackers could stage war or terror attacks of not-yet-seen scale. Other scenarios might be less about harming humans, but could include denial-of service on single vehicles (e.g., owners cannot use their vehicles unless a ransom is paid) or on fleet level (e.g., blocking important road infrastructure, threatening whole communities, and some serious damage of a brand's reputation). And of course, there is simple car theft. Data privacy is also an important aspect. Modern cars might ”know” quite a lot about their users, including their past and present locations, driving habits, additional passengers, anything spoken in the vehicle, attention level while driving, contact information like phone numbers from connected personal devices, etc. An attack could therefore retrieve quite a lot of personal information and thus become considerable value to attackers. While not all of these threats have been discussed widely in public, the automotive industry is very much aware of it, and has stepped up efforts in designing more secure systems in cars, and establishing secure life cycle processes to provide necessary updates to fix vulnerabilities. An important part of securing these vehicular systems is the verification and validation of the effectiveness of taken security measures through testing. This testing needs to be

done continuously through the life cycle (as new exploits might come up over time), and also as updating a system (or just a part of it) might alter its behavior an a way relevant to its security. In essence, (cyber)security testing must assure a system to display a small attack surface, be resilient and (possibly) to fix vulnerabilities before they are exploited in the wild.

The remainder of the chapter is structured the following way: Section 2 contains the current state of the art and related work. Section 3 contains measures for securing automotive systems. Section 4 contains specific approaches for automated cybersecurity testing of vehicular systems, including model-based testing and the description of an automated platform for executing tests. Section 5, eventually concludes this chapter.

2 State of the Art and Related Work

The automotive industry can draw from experience in other domains regarding security testing. General IT (managing e.g., corporate networks and IT systems) has established a history of penetration testing (abbreviated: pen testing), as simulated, authorized cyber-attacks. Typically executed by cybersecurity experts (acting as "white-hat hackers"), the goal is to identify weaknesses by letting these experts try to hack into the system under test (SUT) under pre-defined constraints (e.g., no physical access, no permanent harm), typically within a defined time window. If successful, these tests can thereby discover and document weaknesses. Translated to automotive industry, several companies offer similar pen-testing as a service on different levels (component, system, vehicle). While pen-testing might provide highly valuable insights into what level of security has been achieved for the vehicle, and might even uncover previously unknown vulnerabilities, it suffers from limited scalability and repeatability, as it is driven by and dependent on human experts. Security experts have toolboxes with highly effective tools (like the open source Metasploit framework[2]), but often need to supervise and configure these tools, and to adapt existing or write new scripts for complete attack chains to match a specific SUT. This requires skills and labor, and often involves considerable costs, which clearly limits scalability. Due to the sheer complexity of automotive software code (100+ MLOSC), it is also quite challenging for the experts to correctly hypothesize vulnerabilities, and to select (and execute) the most effective attacks, given the limited time available. This might heavily depend on expertise of the human testers, further limiting repeatability and comparability between pen tests campaigns. The threat of cyber attacks by adversaries has, however, also been recognized by standards and regulatory bodies. The United Nations Economic Council for Europe (UNECE) has issued a regulation (R 155 [35]) that prescribes the installation of a cybersecurity management system (CSMS). A CSMS is a process framework that accompanies the automotive development process over the complete life cycle and assures cybersecu-

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

rity in every phase. Consequently, the International Organization for Standardization (ISO) and the Society of Automotive Engineers (SAE) have issued a joint standard (ISO/SAE 21434 [16]) that defines such a CSMS. As testing guidelines in these standards are somewhat underrepresented in contrast to security engineering, a structured approach is needed, e.g., as defined in [23, 24]. It further became clear that in order to establish dependable security covering all variants of vehicle lines in their full life cycle, supporting the upcoming accelerated software development cycles (automotive DevOps), an advanced process based on smart automation was required, as suggested in [6].

3 Automotive Cybersecurtiy Lifecycle Management

In order to maintain secure (and through, security-related impacts, also safe) vehicular systems, the respective system needs a security concept. The cybersecurity testing (see Section 4) will eventually validate and verify the effectivness of that concept. To establish a security concept for the complete life cycle of a vehicle for testing, we mainly rely on five pillars:

1. Threat Modeling (see Section 3.1)
2. Variant Management
3. Vulnerability Assessment
4. Automated Test Generation (see Section 4.2)
5. Process Governance

Threat modeling (see Section 3.1) is a widely proliferated technique in the automotive industry, mainly as part of a threat analysis and risk assessment (TARA) process [38].

As an OEM's fleet contains various vehicle model configurations, all of which contain tens of ECUs all of which again may display different hardware and software versions, keeping track of this potentially vast number of variants is crucial to determine the security posture of each member of the fleet. Our approach to tackle this problem is to use calibration data management that links technical attributes with software calibrations, to keep track of all ECU variations over the system's life cycle [5, 30]. This system, CRETA, contains exhaustive information about the variants, including their ECU firmware binaries.

This allows for the stored firmwares to be subsequently analyzed, generating a digital model of the software. To do so, firstly the firmware is extracted by iterating through the file tree, using an extraction algorithm and validating the extraction's correctness. The extracted software undergoes a composition analysis that pre-processes executables and normalizes the software in order to compare to a large database of mapped components, identified e.g. by file paths, file names, and characteristic strings in the software or configuration data, yielding a Software Bill-of-Materials (SBOM). Subsequently, the model is analyzed for security properties using pattern recognition. Patterns of known attacks from Common Vulnerabilities and Exposures

(CVEs) are compared with each identified software library in the SBOM. Furthermore, the model undergoes a binary code analysis to find vulnerabilities not found in public databases: the binary is mapped in data and code sections, the code is then disassembled and later mapped into an intermediate language (for normalizing purposes) that allows for reconstructing the functions, analyzing the parameters and stack behavior and building control and data flows [11]. This matching, for instance, is able to identify common flaws like buffer overflows and, hence, is able to uncover zero-day vulnerabilities in software in a black box setting. Thirdly, patterns for proliferated code guidelines and relevant security standards are implemented, allowing for compliance checking against a given set of standards. This analysis, paired with full life cycle-coverage of the variants, allows for dealing with the parts lists and vulnerability management requirements mentioned above, as well as for verifying security requirements.

Vulnerabilities found in the code through pattern matching, however, are not necessarily exploitable for a variety of reasons. For instance, the location in the code could not be reachable, the impact of the vulnerability could be nullified through write protection of the memory or file system, or the interface might be protected by access controls. Therefore, the generated model also allows for model-based cybersecurity test case generation by using either the generated behavior model for model checking or by directly using the found patterns as basis for vulnerability exploitation [22]. We also aim for deriving test cases from threat modeling with a certain degree of automation (see Section 4.2).

To govern the process we developed our tool, FUSE, that guides activities of a given standard and provides standards-compliant documentation given the necessary input. We implemented ISO/SAE 21434 [16] and UNECE R155 [35] (as well as ISO 26262 [15], ISO 25119 [14]). The modeled objectives from the standards allow for providing all necessary artifacts for performing a review or audit, as well as keeping track of the conformance to relevant standards inside the development project.

3.1 Threat Modeling

One key element of cybersecurity analysis in all life cycle phases is threat modeling. This technique for security analysis is around for many years and well proliferated. It basically consists of modeling the information flows in an SUT and consequently examining them in a comprehensive way, e.g., via STRIDE or a similarly structured method [33].

Numerous software capable of performing a thread modeling process exists, but prior to ThreatGet none was specifically developed for embedded or IoT systems. ThreatGet is a software tool developed by Austrian Institute of Technology (AIT) and based on Microsoft Enterprise Architect, a commonly used platform for systems model engineering [7].

It is used to examine models, objects, connections and charts in a system to enable iterative threat and risk analysis, covering the following categories:

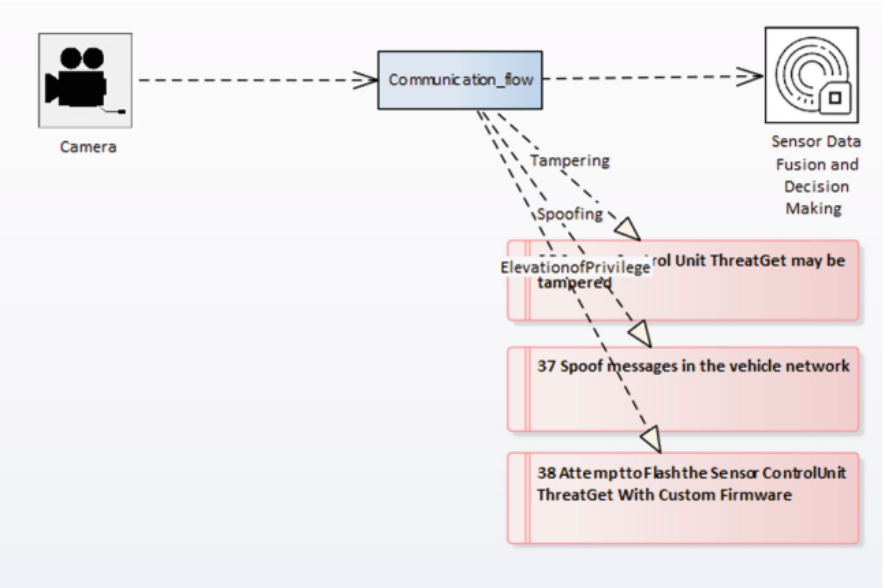

Fig. 1 A list of found threats between the camera and the sensor data fusion and decision making [7].

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

Objects and connections in ThreatGet have so called tagged values at creation time. These describe analysis or security relevant properties of elements. It is recommended for users to extend the properties in addition to already proposed tagged values. Additionally, a database is used in the background that contains objects, which can also be extended by a user [7].

As an application example, Figure 1 shows the threat diagram of a communication flow inside ThreatGet. In this case, the environment data from the camera is directed to the ”Sensor Data Fusion and Decision Making” unit. After all diagrams are completed, a threat-overview is derived. An automatic risk evaluation consists of suggested values and can be adapted in a manual risk evaluation. In this step it is possible to rate the impact and occurrence of a threat at different levels and afterwards results can be exported in a report [7].

4 Cybersecurity Testing

In order to assure the cybersecurity of automotive systems and provide evidence for the appropriateness and effectiveness of security measures (according to a cybersecurity management system) , rigorous, structured and comprehensible testing is necessary [35]. Therefore a structured process, aligned with ISO/SAE 21434 [16] is recommendable. Such a process for testing could contain the following activities [24]:

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

While items 1-3 correspond to a threat modeling process (see Section 3.1), the rest of them are the core testing process. To increase testing efficiency, these steps could be partially automated using model and learning-based approaches that can execute test planning and execution steps [6]. Here, the steps can be summarized into *concept design*. Item 4 *forms V &V planning*, while items 5 and 6 can be subsumed under *V & V Methods*. Finally, items 7 and 8 forms *V & V execution*. In between the planning and the methods, steps for automation can take effect: models from the concept design can be validated in an automated way and single components can be modeled using automated learning techniques and verified using methods from the V & V methods. An example of this used in the InSecTT project is described in Section 4.1. The full approach as described above consists of the following steps [6]:

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

4.1 Learning-based Testing

Following the approach described above, we use learning, more concretely active automata learning to derive a model of a system [36]. The methodology uses a learner-teacher system where an all-knowing teacher answers the learning system queries about the SUT, in the context of cyber-physical systems ordinarily by providing the output to a series of inputs. The learner tries to infer a state machine from the given information. Once it has a hypothesis of a state machine that describes the observed behavior, it presents it to the teacher who then acknowledges the hypothesis as correct or gives a counterexample. This again, in real-world situations of black-box learning will mostly be simulated by conformance testing algorithms: if conformance is shown, the hypothesis is assumed as correct, otherwise a failing test sequence serves as a counterexample. The counterexample is taken as new input to refine the hypothesis and the learning continues until no more counterexamples are found. The this algorithm has been first formulated by Angluin [3] and has experienced significant improvements since (e.g., [17, 31]).

In accordance with the process outlined in Section 4, we use this technique to infer a model of a component. As a proof-of-concept we test a car access system based on Near-Field Communications (NFC). The testing setup consists on a learner (as described above) based on the *Learnlib* Java library [18] and a Proxmark NFC device [12] with an respective API that enables us to learn a model of the ISO 14443-3 NFC handshake protocol [13]. Figure 2 shows an overview of this setup. The used learning setup allows for inferring a state machine of the protocol and compare it to the specification in the standard to check its conformance. Figure 3 shows the learned model of the actual SUT (and NXP test card of a car access system prototype). Further use of the model is to do actual model checking or to use the model as an input for guided fuzz testing.

4.2 Model-based Test Case Generation

On a macroscopic level, a model of a complete vehicle as defined in the threat model (see Section 3.1) has to be explored in order to identify single components and generate test cases based on an attack tree [29, 32], a petri net [28, 37], or similar. If the SUT is modeled manually and, therefore, the components are known, this is trivial. If the setting is a black or grey box situation, we follow the approach to assume a generic model as starting point and test various components of the model by, e.g., send certain CAN messages for enumeration or try out an exploit that is known to affect a very broad variety of systems. Based on a comparison of the expected and actual output of the test, one can narrow down the set of likely components and system architectures (as described in [25]), e.g., based on SAT solving [27].

In order to generate test cases on a component level, a model must be transformed into a form that can be examined using a model checker (e.g. the Rebeca model checker [34] or SLAM [4]). Violations of the specification found by a model checker

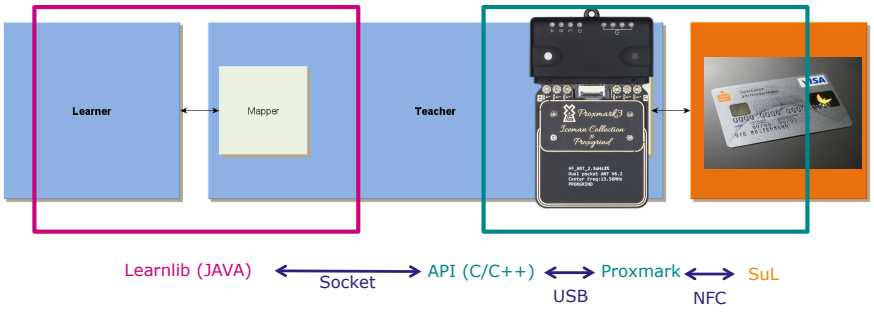

Fig. 2 NFC Automata Learning Setup

point towards an interesting position for a test case that could be extrapolated out of the traces leading to the respective states. There is also work regarding a toolchain using the UPPAAL framework [1].

Subsequently properties defining the security of a system shall be defined and used in the model checking. For c) where the model checking fails, a security problem might be present. The trace of the counter example can help in building a test case. Moreover, the input sequences used for the automata learning of the model shall be used to make test cases for the actual system-under-test. Using the traces as test vectors eliminate false positives from the model checking, as the exploitability of specification violations is test on the actual system. To concrete the abstract input, fuzzing techniques may be used [2].

4.3 Testing Platform

To realize the testing in the faction outlined in Section 4, a testing framework was developed and implemented. The high-level architecture was derived from the approach outlined in [23]. It has been adapted to suit the need of performing test in any phase of the product life cycle by adding co-simulation techniques into the testing framework architecture (see Figure 4 for an overview). The core component is a Security Testing Framework (see Section 4.4). It gains test cases from a generation engine that is fed by two sources: security functional tests from security requirements and penetration test attack vectors that have been tried out before (see description in Section 4.4) from a library. The core framework executes the attacks directly onto the SUT or into a co-simulation platform (indicated as *framework interfaces* in the

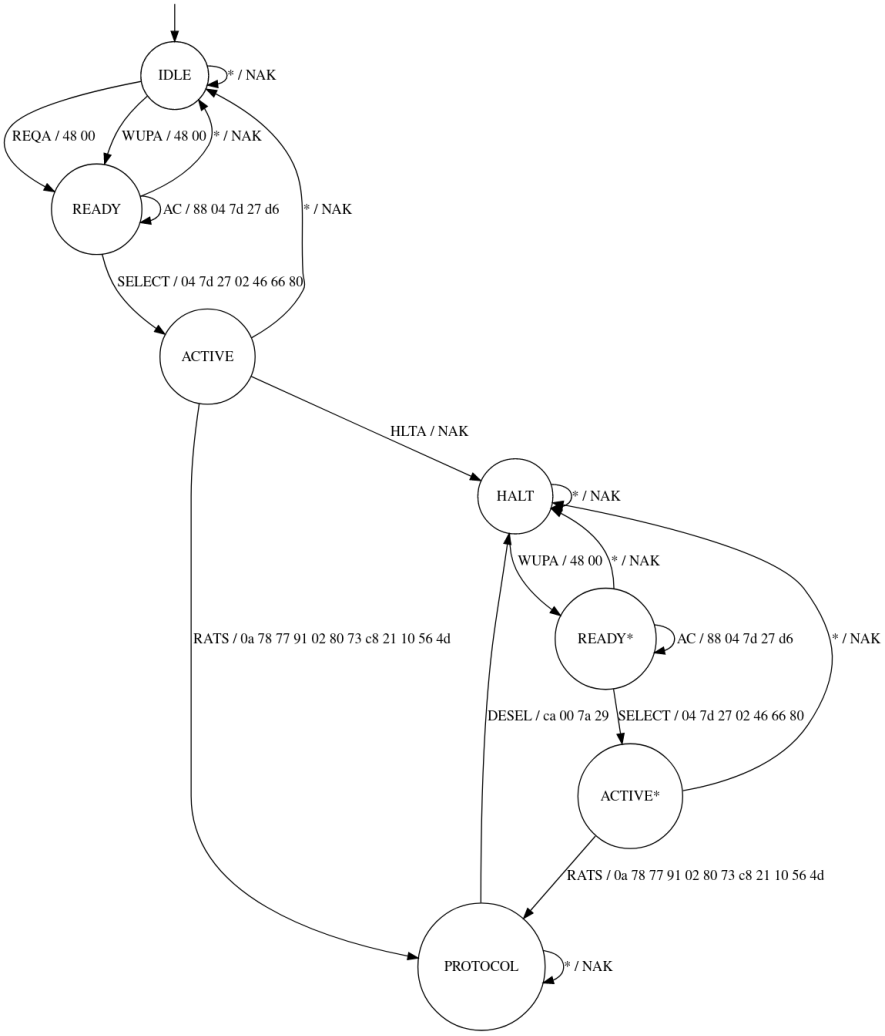

Fig. 3 Learned Model of an NXP NFC Test Card

figure) that interconnects various simulation parts: environment (i.e. other vehicles' and infrastructure's interference), network (generating mainly ITS-G5 traffic), channel (capable of simulating various physical layer signals as well as emitting them physically) and application (Section 4.4 contains an example with a platooning application). This way, each component can be stimulated the same way regardless if it is a physical or simulated component.

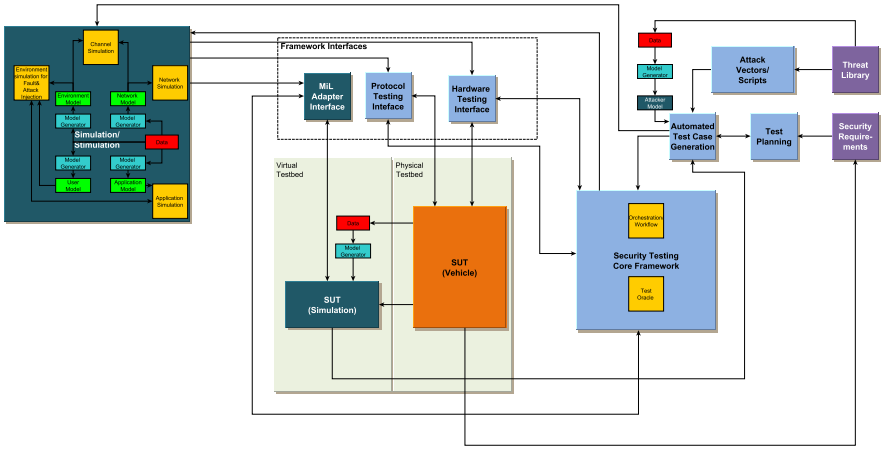

Fig. 4 Overview of the Automotive Cybersecurity Testing Framework's high-level architecture

4.4 Automated Test Execution

For test execution, the test cases that were derived as described in previous chapters are fed into the automated test execution environment. Test cases are either manually written or generated in the ALIA DSL [39] format, which aims to provide an abstract and system agnostic representation of logical steps in a test case. Out of the main test-script and its included sub-scripts (containing frequently occurring blocks that handle a specific task such as opening a listener) a JSON Object is generated. These test case descriptions in DSL and JSON format are stored into a Database and can be accessed through the Orchestration Application; a platform independent web application that allows a user to manage information about the current SUT, schedule test execution and review results. This Orchestration Application then sends the test cases that the user wants to execute to the Execution Engine (AXE) and afterwards generates a report out of the received output from the AXE and the Test Oracle. The AXE is a Python based software that runs on an instance of Kali Linux and utilizes a variety of different interfaces, libraries and other software tools to perform a test case execution. It takes either a single test case or a structured collection of tests as input in JSON format and starts to subsequently execute contained steps. This modular approach allows not only to target a specific SUT but also to control and parameterize whole (semi-) virtual SUT environments to manage SUT-behavior during a test scenario. Furthermore, it is possible to define and address different processes for tool execution which enables for example to host a malicious server, start a netcat listener and execute exploit code sequentially in a single test and afterwards perform code execution in an obtained reverse shell in the listener process.

One proof-of-concept use case implemented in the framework was security testing of the Ensemble platooning protocol [19] in a simulated environment. The concrete setup consisted of two truck simulations running on low-cost hardware connected

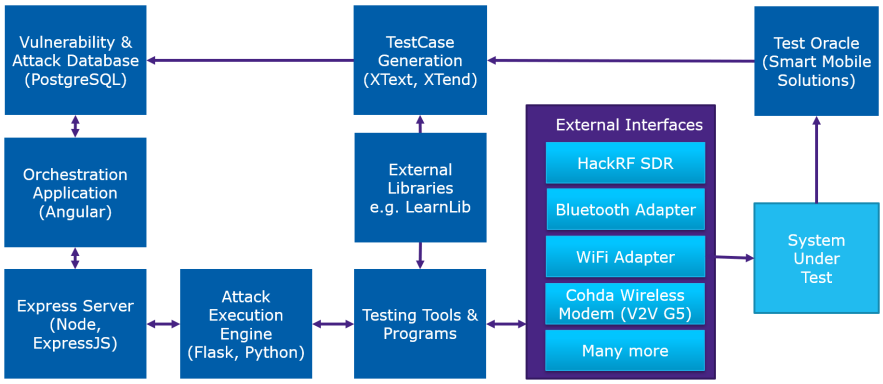

Fig. 5 AACT Test Execution Framework

via physical ITS-G5 [9] connection via Cohda modems. Another modem is used as an adversary to eavesdrop and interfere with the connection. The testing framework is able to start the simulation, so that the simulated trucks form a platoon. The actual test consists of a) listening to the communications b) distilling the session key out of a package c) cracking the key (for testing purposes, the key was reduced to eight bits) d) injecting a malicious message to disband the platoon. Figure 6 shows an overview of this setup. The result was that the injection failed for timing reasons, because the platoon keep-alive messages were sent in such a high frequency that they interfered with the break-up sequence. Even with reduced key (from AES-256 down to 8 bits) the protocol was secure against the tested attack. Furthermore, ITS-G5 built-in signatures, that were disabled for the test, would have prevented a successful injection. The test could therefore show the security of the protocol in an automated way as described above.

4.5 Fuzzing

The goal of fuzzing is to reach a non-intended state of a SUT by using completely or partially random input. The latter technique may use a structured frame structure that is compliant with communication standards used by the SUT and randomized payload data. [26] In case of a CAN-Bus, a fuzzing tool can create packets that consist of the standard ID Range (0 to 2047) and a previously learned or sniffed payload [20]. A fuzzer should include the following components [21]:

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

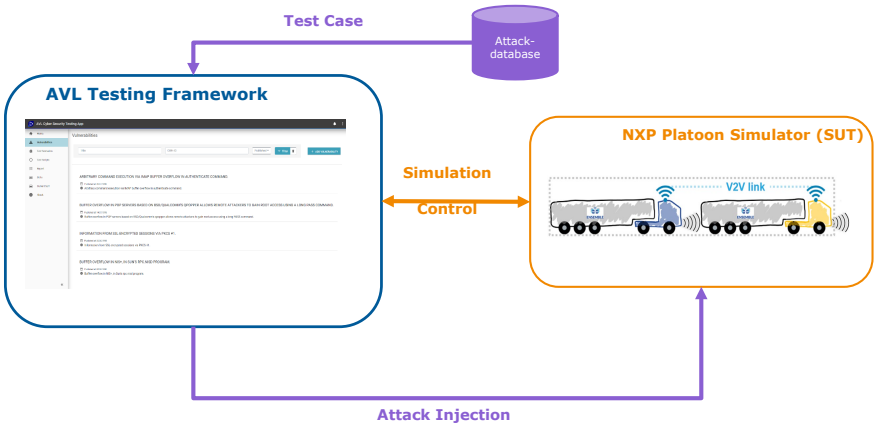

Fig. 6 Platooning Use Case Overview

signals or other physical responses and performs decisions based on it e.g. if a test passes or fails.

By using this approach, no in-depth knowledge about the SUT is needed and every component that provides external interfaces can be targeted for testing, including ECU software, ECU hardware, protocols and busses (e.g. CAN). Fuzzing may be utilized in the automotive environment to [10]:

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

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

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

5 Conclusion

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

References

1. Aarts, F., Heidarian, F., Kuppens, H., Olsen, P., Vaandrager, F.W.: Automata learning through counterexample guided abstraction refinement. In: FM 2012. pp. 10–27. Springer, Berlin (2012)
2. Aichernig, B.K., Muškardin, E., Pferscher, A.: Learning-Based Fuzzing of IoT Message Brokers. In: 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST). pp. 47–58 (Apr 2021). https://doi.org/10.1109/ICST49551.2021.00017
3. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2), 87–106 (Nov 1987). https://doi.org/10.1016/0890-5401(87)90052-6
4. Ball, T., Cook, B., Levin, V., Rajamani, S.K.: Slam and static driver verifier: Technology transfer of formal methods inside microsoft. In: International Conference on Integrated Formal Methods. pp. 1–20. Springer, Berlin, Heidelberg (2004)
5. Dobes, T., Kaserer, T., Schuch, N., Storfer, G.: Smart Variant Calibration with Data Analytics. ATZ - Automobiltechnische Zeitschrift (Extra August 2018) (2018)
6. Ebrahimi, M., Marksteiner, S., Ničković, D., Bloem, R., Schögler, D., Eisner, P., Sprung, S., Schober, T., Chlup, S., Schmittner, C., König, S.: A systematic approach to automotive security. In: Chechik, M., Katoen, J.P., Leucker, M. (eds.) Formal Methods. pp. 598–609. Springer International Publishing, Cham (2023)
7. El Sadany, M., Schmittner, C., Kastner, W.: Assuring compliance with protection profiles with ThreatGet. In: Romanovsky, A., Troubitsyna, E., Gashi, I., Schoitsch, E., Bitsch, F. (eds.) Computer Safety, Reliability, and Security. pp. 62–73. Springer International Publishing, Cham (2019)
8. European Automobile Manufacturers' Association (ACEA): Vehicles in use europe 2022. Tech. rep., European Automobile Manufacturers' Association (ACEA) (2021)
9. European Telecommunications Standards Institute: Intelligent transport systems (its); vehicular communications; basic set of applications; definitions. ETSI "TS 102 638", European Telecommunications Standards Institute (2009)
10. Fowler, D.S., Bryans, J., Shaikh, S.A., Wooderson, P.: Fuzz testing for automotive cybersecurity. In: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). pp. 239–246 (2018). https://doi.org/10.1109/DSN-W.2018.00070
11. Franco da Silva, A.C., Wagner, S., Lazebnik, E., Traitel, E.: Using a Cyber Digital Twin for Continuous Automotive Security Requirements Verification. IEEE Software pp. 0–0 (2022). https://doi.org/10.1109/MS.2022.3171305
12. Garcia, F.D., de Koning Gans, G., Verdult, R.: Tutorial: Proxmark, the swiss army knife for rfid security research: Tutorial at 8th workshop on rfid security and privacy (rfidsec 2012) (2012)
13. International Organization for Standardization: Cards and security devices for personal identification – Contactless proximity objects – Part 3: Initialization and anticollision. ISO/IEC Standard "14443-3", International Organization for Standardization (2018)

14. International Organization for Standardization: Tractors and machinery for agriculture and forestry – Safety-related parts of control systems. ISOStandard 25119, International Organization for Standardization (2018)
15. International Organization for Standardization, Society of Automotive Engineers: Road vehicles – Functional safety. ISOStandard 26262, International Organization for Standardization (2018)
16. International Organization for Standardization, Society of Automotive Engineers: Road Vehicles – Cybersecurity Engineering. ISO/SAE Standard "21434", International Organization for Standardization (2022)
17. Isberner, M., Howar, F., Steffen, B.: The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning. In: Bonakdarpour, B., Smolka, S.A. (eds.) Runtime Verification. pp. 307–322. Lecture Notes in Computer Science, Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_26
18. Isberner, M., Howar, F., Steffen, B.: The Open-Source LearnLib. In: Kroening, D., Păsăreanu, C.S. (eds.) Computer Aided Verification. pp. 487–495. Lecture Notes in Computer Science, Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_32
19. Ladino, A., Xiao, L., Adjenugwhure, K., Deschle, N., Klunder, G.: Cross-platform simulation architecture with application to truck platooning impact assessment. In: ITS World Congress (2021)
20. Lapczynski, P., Heinemann, H., Schöneberger, T., Metzker, E.: Automatically generating fuzz tests from automotive communication databases. 5th escar USA, Detroit, isits AG (Jun 2017)
21. Lee, H., Choi, K., Chung, K., Kim, J., Yim, K.: Fuzzing can packets into automobiles. In: 2015 IEEE 29th International Conference on Advanced Information Networking and Applications. pp. 817–821 (2015). https://doi.org/10.1109/AINA.2015.274
22. Marksteiner, S., Bronfman, S., Wolf, M., Lazebnik, E.: Using Cyber Digital Twins for Automated Automotive Cybersecurity Testing. In: 2021 IEEE European Symposium on Security and Privacy Workshops (EuroS PW). pp. 123–128 (Sep 2021). https://doi.org/10.1109/EuroSPW54576.2021.00020
23. Marksteiner, S., Ma, Z.: Approaching the Automation of Cyber Security Testing of Connected Vehicles. In: Proceedings of the Central European Cybersecurity Conference 2019. CECC 2019, ACM, New York, NY, USA (2019). https://doi.org/10.1145/3360664.3360729
24. Marksteiner, S., Marko, N., Smulders, A., Karagiannis, S., Stahl, F., Hamazaryan, H., Schlick, R., Kraxberger, S., Vasenev, A.: A Process to Facilitate Automated Automotive Cybersecurity Testing. In: 2021 IEEE 93rd Vehicular Technology Conference (VTC Spring). IEEE, New York, NY, USA (2021)
25. Marksteiner, S., Priller, P.: A Model-Driven Methodology for Automotive Cybersecurity Test Case Generation. In: 2021 IEEE European Symposium on Security and Privacy Workshops (EuroS PW). pp. 129–135 (Sep 2021). https://doi.org/10.1109/EuroSPW54576.2021.00021
26. McNally, R., Yiu, K.K.H., Grove, D.A., Gerhardy, D.: Fuzzing: The state of the art (2012)
27. Otten, S., Glock, T., Hohl, C.P., Sax, E.: Model-based Variant Management in Automotive Systems Engineering. In: 2019 International Symposium on Systems Engineering (ISSE). pp. 1–7 (2019)
28. Petri, C.A.: Kommunikation mit Automaten. Ph.D. thesis, Technische Universität Darmstadt (1962)
29. Phillips, C., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: Proceedings of the 1998 Workshop on New Security Paradigms. pp. 71–79. ACM (1998)
30. Rathfelder, M., Hsu, H., Brandau, T., Storfer, G.: Calibration Data Management for Porsche Chassis Systems. ATZ worldwide **118**(6), 16–21 (Jun 2016). https://doi.org/10.1007/s38311-016-0047-z
31. Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. In: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing. pp. 411–420. STOC '89, Association for Computing Machinery, New York, NY, USA (Feb 1989). https://doi.org/10.1145/73007.73047

32. Schneier, B.: Attack trees. Dr. Dobb's journal **24**(12), 21–29 (1999)
33. Shostack, A.: Threat Modeling: Designing for Security. John Wiley & Sons (2014)
34. Sirjani, M.: Rebeca: Theory, applications, and tools. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.P. (eds.) Formal Methods for Components and Objects, 5th International Symposium, FMCO 2006, Amsterdam, The Netherlands, November 7-10, 2006, Revised Lectures. Lecture Notes in Computer Science, vol. 4709, pp. 102–126. Springer (2006). https://doi.org/10.1007/978-3-540-74792-5_5
35. United Nations Economic and Social Council - Economic Commission for Europe: Uniform provisions concerning the approval of vehicles with regards to cyber security and cyber security management system. Regulation "155", United Nations Economic and Social Council - Economic Commission for Europe, Brussels (2021)
36. Vaandrager, F.: Model learning. Communications of the ACM **60**(2), 86–95 (Jan 2017). https://doi.org/10.1145/2967606
37. Varadharajan, V.: Petri net based modelling of information flow security requirements. In: [1990] Proceedings. The Computer Security Foundations Workshop III. pp. 51–61 (1990)
38. Ward, D., Ibarra, I., Ruddle, A.: Threat Analysis and Risk Assessment in Automotive Cyber Security. SAE International Journal of Passenger Cars-Electronic and Electrical Systems **6**(2013-01-1415), 507–513 (2013)
39. Wolschke, C., Marksteiner, S., Braun, T., Wolf, M.: An Agnostic Domain Specific Language for Implementing Attacks in an Automotive Use Case. In: The 16th International Conference on Availability, Reliability and Security. pp. 1–9. ARES 2021, Association for Computing Machinery, New York, NY, USA (Aug 2021). https://doi.org/10.1145/3465481.3470070