Automotive Network Protocol Detection for Supporting Penetration Testing

Florian Sommer*, Jürgen Dürrwang†, Marius Wolf‡, Hendrik Juraschek§, Richard Ranert¶ and Reiner Kriesten‖
Institute of Energy Efficient Mobility (IEEM)
Karlsruhe University of Applied Sciences
Karlsruhe, Germany
email:{*florian.sommer, †juergen.duerrwang, ‡woma1029, §juhe1012, ¶rari1012, ‖reiner.kriesten}@hs-karlsruhe.de

***Abstract*—Currently, the automotive industry aims to integrate security into the vehicle development process. In this process, a vehicle is analyzed for possible security threats in order to develop security concepts or security measures. Another important aspect in vehicle security development is security testing. Penetration testing is often used for this purpose. In penetration testing, a tester acts from the perspective of an attacker and tries to violate security properties of a vehicle through attacks (tests) in order to uncover possible vulnerabilities. Since this task is usually performed as a black box test with little knowledge about the system, penetration testing is a highly experience-based activity. Due to this, an automation of this process is hard to achieve. In this paper, we want to support the penetration testing process and its automation by introducing an extension of our automotive portscanner tool. This scanner was developed to scan vehicle networks, which are different from typical Information Technology (IT) networks, in order to extract information about the vehicle. Our tool is able to gather Electronic Control Units (ECUs) installed in a vehicle, as well as diagnostic services and subfunctions they provide. This functionality is extended by an automatic detection of transport and diagnostic protocols used in vehicles. With this knowledge, new use cases and functionalities like fuzzing or an automated generation of penetration test cases can be realized.**

***Keywords*–Automotive Security; Penetration Testing; Automation; Network Protocols.**

I. Introduction

A trend towards autonomous driving is currently pursued in the automotive industry [1]. This increases the number of sensors and actuators installed in vehicles, as well as the complexity of internal and external communication of vehicle components. The required communication with the outside world for autonomous driving results in an increased risk of security attacks. This has already been demonstrated by various research groups [2]–[8]. Attacks were carried out on vehicles in which it was possible to manipulate actuators, which had an influence on driving physics, such as steering and braking systems. Since only a few methods have been established in the automotive sector to protect against such attacks, a high effort is currently being invested in research and development of security measures, standards and processes. The development partnership AUTomotive Open System ARchitecture (AUTOSAR) presented a measure to secure internal vehicle networks with Secure Onboard Communication (SecOC) [9], which enables authenticated communication of the vehicle's internal bus systems. In January 2016, Society of Automotive Engineers (SAE) International published SAE J3061 (Cybersecurity Guidebook for Cyber-Physical Vehicle Systems) [10], a guideline in which security was integrated into the vehicle development process. In this process, a vehicle is analyzed for possible security threats in order to develop security concepts or security measures. Another important aspect in vehicle security development is security testing. In addition to the verification of implemented security measures, this also includes testing the vehicle for vulnerabilities. Penetration testing [11] is often used for this purpose. In penetration testing, a tester acts from the perspective of an attacker. The tester tries to violate security properties of a vehicle through attacks (tests) in order to uncover possible vulnerabilities. Penetration tests can be carried out as black box tests, without any information about the internal function of a system, or as white box tests in case of knowledge about the internal function. Especially in case of black box tests, the success of a penetration test depends on the experience of a tester, since there is limited knowledge about the system.

Problem: Penetration testing can be time consuming and potential vulnerabilities could be missed, depending on available system information. It is an explorative test method which highly depends on a tester's experience. As a result, an automation of this process is hard to achieve.

Approach: We present a way to support the process of penetration testing through a tool-based solution. Our automotive portscanner, which was introduced in the past [12], serves as a basis. This scanner was developed in order to scan vehicle networks, since they differ from IT networks by used communication technologies, protocols and operating systems. Our tool is used to support the information gathering process and it is able to gather ECUs installed in a vehicle, as well as their diagnostic services and subfunctions.

Contribution: We extend the functionality of our portscanner by an automatic detection of transport and diagnostic protocols which delivers additional information about a vehicle and its internal structure. This leads to a greater coverage when extracting vehicle data. We show how this can contribute to an automation of penetration testing subprocesses by presenting use cases which are possible with the knowledge about a vehicle's transport and diagnostic protocols. As a result, new functionalities like automated testing of attacks or fuzzing [13] based on these transport and diagnostic protocols can be realized.

This work is structured as follows: In Section II, we discuss existing vehicle network communication systems and relevant transport and diagnostic protocols. Furthermore, we present basic penetration testing processes and automotive related adaptions. In Section III, we present our automotive portscanner, as well as its extension for automatic protocol detection and resulting use cases. We also illustrate, how ECUs, transport protocols and diagnostic protocols are automatically detected. To point out how this can contribute to support automated penetration tests, we present different functionalities in Section IV, which can be realized by our automatic protocol detection. Finally, we summarize our results in Section V and present planned future work.

II. BACKGROUND

In this section, we want to give a short overview of a vehicle's network and its communication systems, as well as the process of penetration testing and how it is performed in the automotive domain.

A. *Vehicle Network Protocols*

The Open Systems Interconnection (OSI) layer model [14] is used for a structured description of communication systems. It describes seven different layers which include specific tasks of message transmission. Since we focus on transport and diagnostic protocols for automotive communication systems, the relevant layers for our purposes are: physical layer (1), data link layer (2), transport layer (4) and application layer (7). The layers 1 and 2 are represented by the used communication systems. There are different communication systems in a vehicle, like FlexRay [15], Controller Area Network (CAN) [16], Local Interconnect Network (LIN) [17], Ethernet [18] and Media Oriented System Transport (MOST) [19]. These systems are used for various applications and have different properties. FlexRay is a cyclic network communication system which is used for applications requiring high data rates (10 Mbit/s). MOST and Ethernet are mainly applied for multimedia purposes with even higher data rates. LIN is a serial network protocol which connects components like sensors and ECUs. For the automotive sector, CAN [16] is one of the most important and commonly used bus systems. It is a bitstream-oriented bus system, using twisted pair wires as physical medium. CAN is a broadcast system in which each message is uniquely characterized by an identifier. Since it is currently the most used bus system in the automotive domain, we first focus on CAN for the automated protocol detection. To explain the functionality of that mechanism, this paper is focused on CAN-based transport and diagnostic protocols. Transport protocols represent the fourth layer of the OSI layer model. They are required to transfer data larger than the maximum message length. This is particularly necessary for diagnostic applications and flash programming of ECUs. A further task of the transport protocols is to control time intervals between individual data packages. Transport protocols are also used to forward messages via gateways to networks with a different address space. The widespread protocols are International Organization for Standardization Transport Protocol (ISOTP) [20] and SAE J1939 [21], which are standardized, and Transport Protocol (TP) 2.0 [22], which is a proprietary protocol of vehicle manufacturer Volkswagen. ISOTP and TP 2.0 come into operation for passenger cars, whereas the SAE J1939 protocol is used for commercial vehicles. The application layer is represented by diagnostic protocols. Diagnostic protocols use a so-called Service Identifier (SID) [23] to select different diagnostic services an ECU offers. To understand their functionality, the communication principle is shown in Figure 1. An external diagnostic testing device runs as a client, whereas the ECU runs as a server. To start a diagnostic communication, the diagnostic testing device has to send a diagnostic request message to an ECU. This request contains a SID and a subfunction which is necessary to address diagnostic services like reading the error memory. The addressed ECU can answer with a positive or negative response. A positive response is characterized by the addition of value 0x40 to the SID. A negative response is characterized by an Error-ID (0x7F), the original SID and a Response Code that contains the reason for the negative reponse.

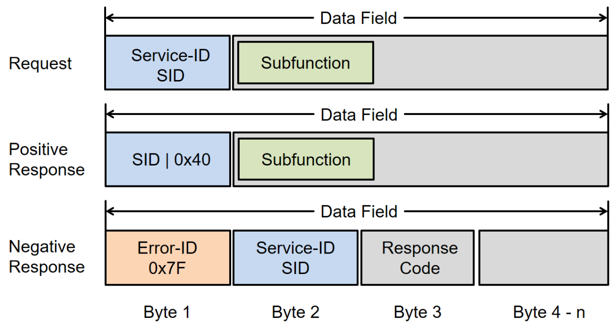

Figure 1. Request and response scheme of automotive diagnostic protocols.

The following three diagnostic protocols are relevant: Keyword Protocol (KWP) 2000 [24], Unified Diagnostic Services (UDS) [23] and On-Board Diagnostics (OBD) [25]. These protocols are standardized and similar to each other, since all of them follow the communication principle in Figure 1. To sum up, there are three relevant components in vehicle networks: the communication systems, tranport protocols and diagnostic protocols. The transport protocol is embedded into the data field of the communication system message and the diagnostic protocol is embedded into the transport protocol.

B. *Penetration Testing*

Penetration tests are carried out on running systems and take place in the late phases of the development cycle. Usually, these tests are black box tests, since the tester has no knowledge of the internal functionality of the system. Thus, the tester acts from the attacker's point of view. Several standards and guidelines have been published for conducting penetration tests. Pure penetration testing standards can be seen as a part of security assessment methods, whereas security assessment methods describe a comprehensive assessment of the security of a system or company. Examples for security assessment guides are National Institute of Standards and Technology (NIST) SP 800-115 (Technical Guide to Information Security Testing and Assessment) [26], Open Source Security Testing Methodology Manual (OSSTMM) 3 [27], Information Systems Security Assessment Framework (ISSAF) [28] and Open Web Application Security Project (OWASP) Testing Guide [29]. An example of a pure penetration testing standard is the Penetration Testing Execution Standard (PTES) [30], which is intended to support companies and security service providers in conducting penetration tests. A methodology for security testing in the automotive sector is presented in the dissertation [31] with Automotive Security Testing Methodology (ASTM), which is divided into five areas: planning phase, detection phase, safe state analysis, moving vehicle analysis, documentation and review. The methodology covers the typical phases of the aforementioned methods and transfers them to the automotive sector, especially the vehicle networking of the control units. Our portscanner is used as an examplary tool for the information gathering phase (detection phase). Vehicle penetration testing has also been addressed in other works. Bayer et al. [32] address penetration testing as a part of practical automotive security testing. In [33], they classify penetration testing as a parallel test method to functional testing, fuzz testing and vulnerability testing, distinguishing between hardware, software, backend and network penetration testing and also considering organizational aspects. In [34], Bayer et al. present an approach for the realization of a penetration testing framework for CAN networks which is based on the work mentioned above and enables a systematic approach for penetration testers. This approach is demonstrated by two examples in which a reverse engineering of CAN identifiers and an exploitation of UDS diagnostic commands is carried out. Another approach to penetration testing was presented by Smith [35]. An overview of possible CAN tools was presented by Pozzobon et al. [36] and Sintsov [37]. Additionally, Dürrwang et al. [38] emphasise the benefits of penetration testing in the automotive sector by exploiting a vulnerability they found in an airbag ECU with a systematic penetration testing process.

III. Approach

In this section, our automotive portscanner and its extension for an automated detection of a vehicle's network protocols is introduced.

A. Automotive Portscanner

The portscanner's purpose is to detect ECUs in a vehicle and to search for offered diagnostic services and subservices, as well as specific data from an ECU. The tool operates without the knowledge of any manufacturer specific information by the user and can be connected to the OBD connector, as shown in Figure 2, and also directly to a bus system.

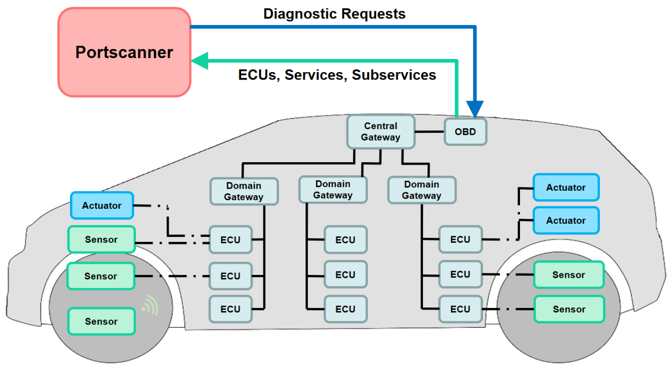

Figure 2. The portscanner sends diagnostic requests to the car to gather ECUs, their diagnostic services and subservices.

In the first step, the portscanner uses an exhaustive search method to detect all ECUs inside a vehicle network. Therefore, standard diagnostic requests, as defined in International Organization for Standardisation (ISO) 14229, are sequencially sent to every possible CAN Identifier (ID). If a response to a request is received (positive or negative), an ECU is identified. In the next step, supported diagnostic services are identified. Similar to the ECU identification process before, every possible SID is checked by sending diagnostic requests to every detected ECU. A service is supported if there is a positive or negative response to a request. As a last step, subservices of all found diagnostic services are identified by sending diagnostic requests to every possible subservice for all supported services of each ECU. A challenge with the portscanner is the variation of transport protocols that can be used in vehicle networks. ISOTP in combination with OBD is required by law for diagnostic purposes. However, specific areas are reserved for vehicle manufacturers in diagnostic standards. For example, transport protocols, such as TP 2.0 are used for these diagnostic requests in vehicles of Volkswagen AG. An example of current capabilities of the portscanner is given in [31], where the tool was applied on two vehicles. On the first vehicle, our portscanner could find 47 ECUs, 380 diagnostic services and 1,924 subservices within 48 minutes and 27 seconds. On the second vehicle, it could find 43 ECUs, 282 diagnostic services and 2,538 subservices within 36 minutes and 5 seconds. A further evaluation across several vehicle types and manufacturers will have to be carried out in the future. The portscanner functionality is extended by an automatic protocol detection in order to achieve a greater coverage during the extraction of vehicle data.

B. Automatic Protocol Detection

To operate the portscanner in an automated way, it is necessary to know the used transport protocol and type of CAN identifiers. Unfortunately, this information is unknown by default. Because of that, we decided to develop an automatic protocol detection to extend the functionality of our portscanner. At first, a differentiation between 11-bit and 29-bit CAN bus systems is necessary. The 11-bit CAN system is refered to as CAN 2.0A, while a 29-bit system is refered to as CAN 2.0B. Both formats are specified in [16]. A differentiation between these formats can be made by the Identifier Extension (IDE) bit, which is a part of the control field of a CAN message. On this account, the tool monitors the CAN bus and checks if the IDE bit is dominant (value 0) or recessive (value 1). A dominant bit signals the usage of the standard 11-bit format. After the ID format is known, transport and diagnostic protocols can be identified. As mentioned in Section II, relevant diagnostic protocols (UDS, OBD, KWP 2000) follow the same scheme, which is illustrated in Figure 1. The main difference between these protocols are the services they address. This results in different SID areas that can be called. In order to identify supported diagnostic protocols, requests have to be sent to vehicle ECUs in which potential SIDs are addressed. In case of a response to a SID, the service and its related diagnostic protocol is supported. Since diagnostic protocols are embedded in a transport protocol format, the identification of these protocols can be executed at the same time. In order to identify a transport protocol, the exhaustive search attempt of the portscanner is extended. The method starts by sending diagnostic requests embedded in every possible transport protocol (ISOTP, TP 2.0, SAE J1939). If there is a response (positive or negative) to one of the combinations, the used transport protocol is supported. In Figure 3, this process is illustrated for 11-bit CAN IDs. If the CAN ID is in range 0x7E0 to 0x7EF, it concerns OBD, since that range is reserved for this diagnostic protocol combined with ISOTP. If the CAN ID is not in that range, we have to check for the transport protocol by checking the CAN frame for ISOTP and TP 2.0 formats. After the transport protocol is recognized, the diganostic protocol support for UDS and KWP 2000 is checked. After the diagnostic protocol is recognized, the next CAN frame with the next CAN ID can be evaluated until all IDs are checked. It should be mentioned that more than one transport protocol can be used within a communication system. For example, even if CAN uses ISOTP, it can additionally use

TP 2.0. The same applies for diagnostic protocols.

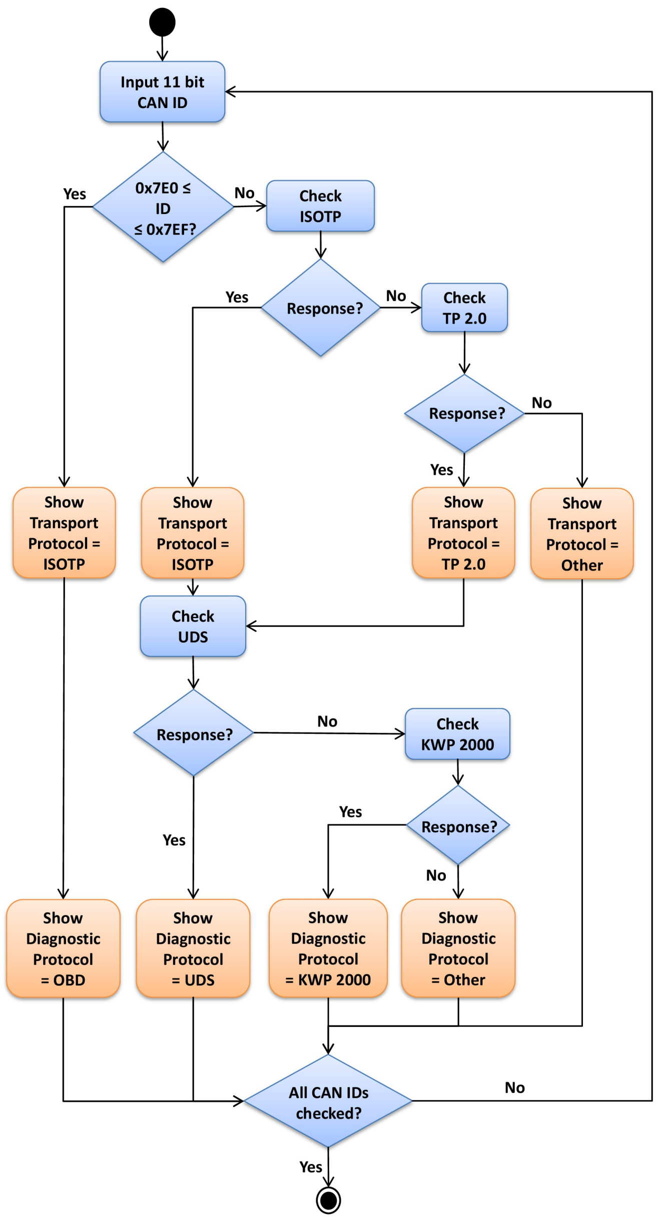
Input 11 bit CAN ID
0x7E0 ≤ ID ≤ 0x7EF?
Yes
No
Check ISOTP
Response?
Yes
No
Check TP 2.0
Response?
No
Yes
Show Transport Protocol = ISOTP
Show Transport Protocol = ISOTP
Show Transport Protocol = TP 2.0
Show Transport Protocol = Other
Check UDS
Response?
No
Check KWP 2000
Yes
Response?
Yes
No
Show Diagnostic Protocol = OBD
Show Diagnostic Protocol = UDS
Show Diagnostic Protocol = KWP 2000
Show Diagnostic Protocol = Other
All CAN IDs checked?
No
Yes

Figure 3. Protocol detection procedure for 11-bit CAN IDs.

This possibility is not shown in Figure 3 due to clearness of the process illustration. Based on the CAN ID, it is possible to reduce the number of potential protocol combinations. For example, since SAE J1939 is only defined for 29-bit systems, it has not to be considered for 11-bit CAN systems. Another reduction can be made for 29-bit CAN IDs. In theory, there are 2^{29} possible IDs for these systems, so an exhaustive search on a 29-bit identifier is not feasible in practice. In order to bypass this problem, the specifications of diagnostic protocols are used. For diagnostic purposes, 29-bit CAN IDs have a specified structure. For ISO 15765, the ID structure is illustrated in Figure 4. As can be seen, there is a differentiation between Source and Destination Address. Source Address is the testers (portscanners) address, which is usually set to 0xF1. Destination Address is the address of an ECU. Since a Destination Address only consists of 11 bit, the number of possible identifiers is reduced to 2^{11}, which is equal to CAN 2.0A 11-bit IDs. The SAE J1939 protocol has a similar structure in which the Destination Address only consists of 8 bit, so there are just 2^8 possible identifiers. These two specifications lead to a significant reduction of the original 2^{29} possible CAN IDs.

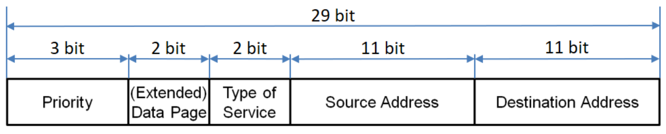

Figure 4. 29-bit CAN identifier in ISO 15765.

C. Example

To illustrate how the protocol detection works, its functionality is described with an example of a 11-bit CAN system, which is shown in Table I. This example follows the control flow shown in Figure 3 and distinguishes three iterations. In the first iteration (line 1 in Table I) of the example, the protocol detection for ISOTP in combination with OBD is explained. The second iteration (line 2) explains the protocol detection for ISOTP in combination with UDS. The last iteration (line 3) shows the detection for ISOTP in combination with KWP 2000. For conciseness reasons, we decided not to show an example of the protocol detection for TP 2.0 or ISOTP in case of a message segmentation (more than 8 data bytes), since the structure of these transport protocols is far more complex and would go beyond the scope of this publication.

TABLE I. EXAMPLE OF THE AUTOMATIC PROTOCOL DETECTION BY SENDING DIFFERENT REQUESTS AS DESCRIBED IN FIGURE 3 (NUMBERS ARE IN HEXADECIMAL FORMAT).

Message	ID	Data	Identified
ISOTP OBD Request ISOTP OBD Response	7E0 7E8	02 01 05 00 00 00 00 00 06 41 05 22 AA 00 D5 00	ISOTP, OBD
ISOTP UDS Request ISOTP UDS Response	602 630	02 10 03 00 00 00 00 00 04 40 03 00 CD 00 00 00	ISOTP, UDS
ISOTP KWP 2000 Request ISOTP KWP 2000 Response	604 6A0	02 1A 01 00 00 00 00 00 06 5A 01 00 89 23 41 00	ISOTP, KWP 2000

In the first iteration (line 1), a diagnostic request based on ISOTP and OBD is sent on CAN. Since there is a response to this request and the CAN ID is in range 0x7E0 to 0x7EF, the transport protocol ISOTP and the diagnostic protocol OBD is supported. The second iteration (line 2) is similar to the first one. The difference is the diagnostic protocol used for the request, which is UDS now. The CAN ID of the reponse is not in the OBD ID range, so according to Figure 3 the response is checked for ISOTP, which is supported, since the reponse has the format of this transport protocol. After that, the diagnostic protocol has to be recognized. The requested service (SID = 0x10) and its subservice (0x03) is a UDS specific service, so the diagnostic protocol is UDS. The last iteration (line 3) contains a diagnostic request based on ISOTP and KWP 2000. The detection works similar to the former two iterations. Since the requested service (SID = 0x1A) is a KWP 2000 specific service, the diagnostic protocol is KWP 2000. It should be mentioned that presented transport and diagnostic protocols are relatively complex, so the example in Table I and detection process in Figure 3 can differ for some protocol combinations or detection functions. UDS, for example, is a replacement for KWP 2000 and many services between those protocols have the same SID, so in this case a differentiation has to be made at the level of subservices. Another difference could be more physical when considering the baudrate of the bus system. For example, UDS does not prescribe a baudrate, in contrast to OBD. We do not want to go into too much detail of the special protocol properties. Instead, we want to focus on the use cases our automatic protocol detection can enable for penetration testing, which is described in next Section.

IV. Use Cases

In this section, use cases of the portscanner and its automatic protocol detection for penetration testing purposes are presented.

A. Gathering ECUs, Services, Subservices and Vehicle Data

The possibility to extract information about existing ECUs, its diagnostic services and subservices has been described before and is part of the portscanner functionality. Another feature facilitates the extraction of vehicle data like fault memory, chassis number, or ECU firmware versions. This can be done by requesting diagnostic services. Further, the extension to support an automatic protocol detection enables our tool to achieve a greater coverage by gathering even more vehicle information including prescribed and manufacturer-specific diagnostic functions. This is an advantage for penetration testing, as it enables the tester to obtain far more information about the vehicle which is particularly relevant for black box penetration tests.

B. Reverse Engineering of the Routing Table

Normally, a diagnostic tester is connected to the OBD connector of a vehicle. For vehicles, which usually have several bus systems, these bus systems are separated via gateways, whereas each gateway is responsible for mapping the message format of one bus system to the message format of another bus system (for example CAN to LIN). If a diagnostic request is sent to a control unit that is connected to the OBD connector via several gateways and bus systems, diagnostic messages have to be routed via these connections to the target ECU. This routing is determined during the development of the vehicle in form of a routing table, which is not known to testers. From a penetration testing point of view, reverse engineering of that table can be accomplished with the automatic protocol detection by sending a diagnostic request to observe the message routing on the bus systems between OBD connector, gateways and target ECUs. However, it should be mentioned that this requires a physical connection to these bus systems.

C. Automatic Test Case Generation

Based on recognized vehicle network protocols, it is possible to automatically derive test cases. This can be done on the basis of known vulnerabilities which can be used for an exploitation of diagnostic services that were attacked in the past. Many of the attacks mentioned in Section I are based on exploited diagnostic services [2]–[4]. Therefore, this information can be used as data input for the automatic generation of test cases. Another possible data input could be our own collection [39] of vulnerabilities and attacks on vehicles, which was classified in form of a taxonomy [40], to support penetration testing.

D. Fuzzing

In fuzzing, an attempt is made to test a system for its susceptibility to errors or robustness by entering random or modified data [13]. This method can uncover new vulnerabilities in a system. The usage of fuzzing techniques in the automotive sector has been shown in [41] in which fuzzing is performed using the data bytes of CAN messages, or in [42] in which the fuzzing tool beSTORM was extended by the Controller Area Network Flexible Datarate (CAN FD) [43] protocol. Another fuzzing tool in the automotive industry is CaringCaribou, which was developed as part of the HEAling Vulnerabilities to ENhance Software Security and Safety (HEAVENS) research project [44]. By knowing the transport and diagnostic protocols, the input sequence for fuzzing can be specified based on the given information by simply changing the data within these protocols.

E. Vulnerability Scanning

Another use case incorporates vulnerability scanning in which a system is scanned for known security vulnerabilities. Through an automatic detection of supported vehicular network protocols, the scanning process can be automated. Vulnerability databases can serve as data input for our tool. The Karlsruhe University of Applied Sciences [45] currently aims to develop such a database for the automotive sector as part of the Security For Connected, Autonomous caRs (SecForCARs) [46] project.

F. Exploitation Tool

Considering the aforementioned features, our tool could be extended to an exploitation tool, with which it is possible to exploit found vulnerabilities, in order to carry out an actual attack on the vehicle. Therefore, the tool could actively support the process of penetration testing and its partial automation.

V. Conclusion and Future Work

In this paper, we presented an extension of our automotive portscanner [12] by introducing an automatic protocol detection for vehicle networks. The automatic protocol detection results in new use cases, which allow an extension of penetration test activities by additional functions. This especially supports black box penetration tests and enables a partial automation of the process. Since only the use cases *Gathering ECUs, Services, Subservices and Vehicle Data* and *Reverse Engineering of the Routing Table* are realized currently, future work could include the extension of our tool by further use cases. To put the aforementioned use cases into practice, an incorporation of further bus systems into our tool is required. Examples are LIN, Ethernet, CAN FD and FlexRay. These bus systems partially use different protocols like User Datagram Protocol (UDP) [47] and Transmission Control Protocol (TCP) [48] for the transport layer or Diagnostics over IP (DoIP) [49] for the application layer of Ethernet. An extension of our tool by these protocols could be conceivable. Since there have been several remote attacks on vehicles, another extension could include an implementation of wireless technologies to our tool, so the risk for this type of attack can be assessed. In this way, the portscanner can be extended to a useful penetration testing tool for vehicle networks.

Acknowledgements

This work has been developed in the project SAFE ME ASAP (reference number: 03FH011IX5) that is partly funded by the German ministry of education and research (BMBF).

References

[1] M. Maurer, J. C. Gerdes, B. Lenz, and H. Winner, ”Autonomes Fahren: technische, rechtliche und gesellschaftliche Aspekte [Autonomous Driving: Technical, Legal and Social Aspects]”. Springer-Verlag, 2015.

[2] C. Miller and C. Valasek, “Adventures in automotive networks and control units,” Def Con, vol. 21, 2013, pp. 260–264.

[3] ——, “A survey of remote automotive attack surfaces,” Black Hat USA, vol. 2014, 2014.

[4] ——, “Remote exploitation of an unaltered passenger vehicle,” Black Hat USA, vol. 2015, 2015.

[5] Keen Lab, “Experimental Security Assessment of BMW Cars: A Summary Report,” 2017.

[6] K. Mahaffey, “Hacking a Tesla Model S: What we found and what we learned,” 2015, https://blog.lookout.com/hacking-a-tesla. [accessed: 2019-09-03].

[7] K. Koscher et al., “Experimental Security Analysis of a Modern Automobile,” in 2010 IEEE Symposium on Security and Privacy. IEEE, 5/16/2010 - 5/19/2010, pp. 447–462.

[8] S. Checkoway et al., “Comprehensive Experimental Analyses of Automotive Attack Surfaces,” in USENIX Security Symposium, 2011.

[9] AUTOSAR, “Specification of Secure Onboard Communication,” 2018, https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_SecureOnboardCommunication.pdf. [accessed: 2019-09-03].

[10] SAE Vehicle Electrical System Security Committee, “SAE J3061-Cybersecurity Guidebook for Cyber-Physical Vehicle Systems,” SAE-Society of Automotive Engineers, 2016.

[11] B. Arkin, S. Stender, and G. McGraw, “Software penetration testing,” IEEE Security & Privacy, vol. 3, no. 1, 2005, pp. 84–87.

[12] M. Ring, J. Dürrwang, F. Sommer, and R. Kriesten, “Survey on Vehicular Attacks - Building a Vulnerability Database,” in 2015 IEEE International Conference on Vehicular Electronics and Safety (ICVES). IEEE, 11/5/2015 - 11/7/2015, pp. 208–212.

[13] B. P. Miller, L. Fredriksen, and B. So, “An empirical study of the reliability of UNIX utilities,” Communications of the ACM, vol. 33, no. 12, 1990, pp. 32–44.

[14] ISO/IEC 7498-1:1994, “Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model,” 1994.

[15] ISO 17458-1:2013, “Road vehicles–FlexRay communications system–Part 1: General information and use case definition,” International Organization for Standardization, 2013.

[16] ISO 11898-1:2015, “Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling,” 1993.

[17] L. Consortium, “LIN: Specification Package. Revision 2.2 A, 2010.”

[18] IEEE 802.3bw-2015, “IEEE Standard for Ethernet Amendment 1: Physical Layer Specifications and Management Parameters for 100 Mb/s Operation over a Single Balanced Twisted Pair Cable (100BASE-T1),” 2015.

[19] M. Cooperation, “MOST Specification Rev 3.0,” 2008.

[20] ISO 15765-2:2016, “Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) – Part 2: Transport protocol and network layer services,” 2016.

[21] SAE J1939_201206, “Serial Control and Communications Heavy Duty Vehicle Network - Top Level Document,” 2012.

[22] W. Zimmermann and R. Schmidgall, ”Bussysteme in der Fahrzeugtechnik: Protokolle, Standards und Softwarearchitektur [Bus Systems in Automotive Engineering: Protocols, Standards and Software Architecture]”, 5th ed. Wiesbaden: Springer Vieweg, 2014.

[23] ISO 14229:2006, “Road vehicles — Unified diagnostic services (UDS) — Specification and requirements,” 2006.

[24] ISO 14230-3:1999, “Road vehicles – Diagnostic systems – Keyword Protocol 2000 – Part 3: Application layer,” 1999.

[25] ISO 15031-3:2016, “Road vehicles – Communication between vehicle and external equipment for emissions-related diagnostics – Part 3: Diagnostic connector and related electrical circuits: Specification and use,” 2016.

[26] K. A. Scarfone, M. P. Souppaya, A. Cody, and A. D. Orebaugh, “SP 800-115. Technical Guide to Information Security Testing and Assessment,” 2008.

[27] P. Herzog, “Open-Source Security Testing Methodology Manual,” Institute for Security and Open Methodologies (ISECOM), 2003.

[28] B. Rathore et al., “ISSAF-Information System Security Assesment Framework-30.04,” 2006.

[29] M. Meucci, E. Keary, and D. Cuthbert, “OWASP Testing Guide, v3,” OWASP Foundation, vol. 16, 2008.

[30] C. Nickerson et al., “The Penetration Testing Execution Standard,” 2017.

[31] M. Ring, “Systematische Security-Tests von Kraftfahrzeugen [Systematic Security Tests of Motor Vehicles],” Dissertation, Universität Ulm, Ulm, 2019.

[32] S. Bayer, T. Enderle, D. Oka, and M. Wolf, “Automotive Security Testing—The Digital Crash Test,” in Energy Consumption and Autonomous Driving. Springer, 2016, pp. 13–22.

[33] S. Bayer, “Practical Security Evaluations of Automotive Onboard IT Components,” 2015.

[34] S. Bayer, K. Hirata, and D. Oka, “Towards a Systematic Pentesting Framework for In-Vehicular CAN,” 2016.

[35] C. Smith, The Car Hacker’s Handbook: A Guide for the Penetration Tester. No Starch Press, 2016.

[36] E. Pozzobon, N. Weiss, S. Renner, and R. Hackenberg, “A Survey on Media Access Solutions for CAN Penetration Testing,” 2018.

[37] A. Sintsov, “Pentesting Vehicles with CANToolz: YACHT - Yet Another Car Hacking Tool,” 2016.

[38] J. Dürrwang, J. Braun, M. Rumez, R. Kriesten, and A. Pretschner, “Enhancement of Automotive Penetration Testing with Threat Analyses Results,” SAE International Journal of Transportation Cybersecurity and Privacy, vol. 1, no. 11-01-02-0005, 2018, pp. 91–112.

[39] F. Sommer and J. Dürrwang, “Automotive Attack Database (AAD),” 2019, https://github.com/IEEM-HsKA/AAD [accessed: 2019-09-03].

[40] F. Sommer, J. Dürrwang, and R. Kriesten, “Survey and Classification of Automotive Security Attacks,” Information, vol. 10, no. 4, 2019, p. 148.

[41] H. Lee, K. Choi, K. Chung, J. Kim, and K. Yim, “Fuzzing CAN Packets into Automobiles,” in 2015 IEEE 29th International Conference on Advanced Information Networking and Applications, 2015, pp. 817–821.

[42] R. Nishimura et al., “Implementation of the CAN-FD protocol in the fuzzing tool beSTORM,” in 2016 IEEE International Conference on Vehicular Electronics and Safety (ICVES), 2016, pp. 1–6.

[43] F. Hartwich, “CAN with flexible data-rate,” in Proc. iCC. Citeseer, 2012, pp. 1–9.

[44] The HEAVENS project, “CaringCaribou: A friendly car security exploration tool,” https://github.com/CaringCaribou/caringcaribou, 2019, https://github.com/CaringCaribou/caringcaribou [accessed: 2019-09-03].

[45] Karlsruhe University of Applied Sciences, “Sichere Datenverarbeitung beim autonomen Fahren: Starke IT-Sicherheit für das Auto der Zukunft – Forschungsverbund entwickelt neue Ansätze [Secure Data Processing during Autonomous Driving: Strong IT Security for the Car of the Future – Research Consortium develops new Approaches],” 2019, https://www.hs-karlsruhe.de/presse/secforcars/. [accessed: 2019-09-03].

[46] Federal Ministry of Education and Research, “SecForCARs: Security For Connected, Autonomous Cars,” 2019, https://www.forschung-it-sicherheit-kommunikationssysteme.de/projekte/sicherheit-fuer-vernetzte-autonome-fahrzeuge. [accessed: 2019-09-03].

[47] J. Postel, “RFC 768: User datagram protocol,” Tech. Rep., 1980.

[48] ——, “RFC 793: Transmission control protocol,” 1981.

[49] ISO 13400-2:20126, “Road vehicles - Diagnostic communication over Internet Protocol (DoIP) – Part 2: Transport protocol and network layer services,” 2012.