Automated Threat Evaluation of Automotive Diagnostic Protocols

Nils Weiss[1,2], Sebastian Renner[1,3], Jürgen Mottok[1], and Václav Matoušek[2]

[1] University of Applied Sciences, Regensburg, Germany
{nils2.weiss, sebastian1.renner, juergen.mottok}@othr.de
[2] University of West Bohemia, Pilsen, Czech Republic
matousek@kiv.zcu.cz
[3] Technical University of Munich

Abstract. Diagnostic protocols in automotive systems can offer a huge attack surface with devastating impacts if vulnerabilities are present. This paper shows the application of active automata learning techniques for reverse engineering system state machines of automotive systems. The developed black-box testing strategy is based on diagnostic protocol communication. Through this approach, it is possible to automatically investigate a highly increased attack surface. Based on a new metric, introduced in this paper, we are able to rate the possible attack surface of an entire vehicle or a single Electronic Control Unit (ECU). A novel attack surface metric allows comparisons of different ECUs from different Original Equipment Manufacturers (OEMs), even between different diagnostic protocols. Additionally, we demonstrate the analysis capabilities of our graph-based model to evaluate an ECUs possible attack surface over a lifetime.

Keywords: Automotive Diagnostic Protocols · Security Metrics · Automated Network Scan.

1 Introduction

Modern cars can be seen as safety-critical systems that get connected to external networks or even in between each other to satisfy upcoming requirements of both owners and manufacturers. This development presents new attack surfaces that are subject to exploitation, especially in recent years [16,5,19]. Since unauthorized manipulation of safety-critical systems in a vehicle can lead to serious danger, mechanisms for securing a car's network against intruders need to be implemented during the development process already. This is also required by upcoming standards from recognized organizations like International Organization for Standardization (ISO) [15] and United Nations Economic Commission for Europe (UNECE) [27]. Directing the development of a product towards these regulations is time-consuming and expensive. Often traditional companies might not have enough experience in security engineering, which demands a high degree of automation in this process. This work targets automated attack surface exploration of the car's network structure, focusing especially on diagnostic protocols. In addition to that, automated scans allow continuous monitoring of future vulnerabilities over a vehicle's lifetime. In summary, the following contributions are made:

– We present a **open-source tool** for scanning a car's network on the application layer through leveraging the diagnostic protocol stack. Our tool can derive various controllable system states for an ECU **without specific knowledge** being necessary.

- Gained information from our scan can be used to **stimulate** the internal state of the **system under test**, which leads to more data on the system's behavior and it's attack surfaces.
- We demonstrate how the output of the scanning phase can be instrumented in order to **determine** which **type of firmware** the ECU is running in which detected state.
- We present an **attack surface model** together with an **attack surface metric** for rating the potential attack surface of ECUs or entire vehicles.
- We show the application of our attack surface model for **estimations over** a vehicles or an ECUs **lifetime**.

This paper is structured as follows: In the next section, we discuss related work. Section 3 introduces the protocol stack for diagnostic communication in vehicles, including an overview of the most relevant protocols and layers. Section 4 presents threat definitions and measurements for possible attack surface metrics. We introduce our scan algorithm and necessary definitions in section 5. The scan algorithm and threat definitions are combined into a graph-based attack surface model in section 6. Section 7 provides information on our test setup. Section 8 discusses results obtained from various scans on different targets and demonstrates the application of our attack surface model. Section 9 concludes our research and mentions ideas regarding future work in this domain.

2 Related Work

In the past years, research on automotive network security and stateful scanning has been published. Ruiter et al. applied state-based fuzzing techniques to reveal security vulnerabilities in popular implementations of the Transport Layer Security (TLS) protocol [24]. Moreover, they used state machine interference to conduct a black-box analysis of the OpenSSL state machine implementation in an active learning process [23]. Another work from Bayer et al., rather targeted at the automotive domain, discusses stateful fuzzing of vehicle systems. They present a message generator and publisher for fuzzing diagnostic protocols. In the following, an evaluation of their fuzzing design is carried out by performing a fuzz test of an ECU on the Unified Diagnostic Service (UDS) protocol level [2]. A manual strategy to detect vulnerabilities in diagnostic protocols is introduced by Van den Herrenwegen and Garcia [9]. With this technique, it is possible to derive similar information as we aim for in our work. However, we focus on highly automated testing in order to facilitate the information collection phase. In comparison to the work of Sommer et. al., we utilize active automata learning to feedback the detected endpoints and system states into the state machine of our scan algorithm [26]. This feedback helps us in performing a deeper and more extensive scan. Furthermore, our online state manipulation can trigger different behaviors of the ECU while testing. Lastly, we achieve a high degree of automation in our testing workflow, which speeds up scanning, helps in the collection and presentation of the results, and allows for simple reproduction and validation.

3 Diagnostic Protocol Architecture

Since this research focuses on scanning approaches of diagnostic protocols, Ethernet and Controller Area Network (CAN) based networks are targeted. A clear separation between the transport and the application layer allows creating one scanner for both network stacks. Figure 1 provides an overview of relevant protocols and the corresponding layers. UDS defines a clean separation between application and transport layer. On CAN based networks, Transport Layer (ISO-TP) [13] is used for this purpose. The CAN protocol can be treated as the network access protocol. This allows to replace

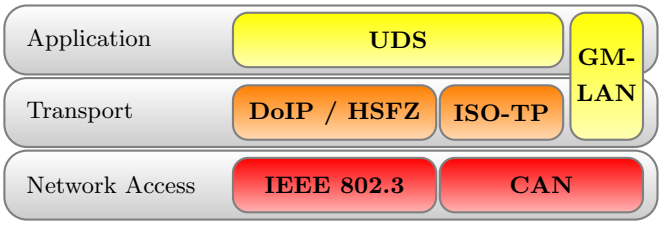

Fig. 1. Automotive Diagnostic Protocol Architecture

Table 1. List of measurands and units for CVE flaw types.

Type	Measurand	Unit
upload	`Availability of service`	`Boolean`
dos-flood	`Number of supported sub-functions`	`Integer`
rand	`Number of supported sub-functions`	`Integer`
pass	`Number of supported sub-functions`	`Integer`
crypt	`Number of supported sub-functions`	`Integer`
phys	`Number of supported sub-functions`	`Integer`
infoleak	`Number of data bytes readable`	`Integer`
buf	`Number of data bytes writable`	`Integer`
int-overflow	`Number of data bytes writable`	`Integer`

ISO-TP and CAN with Diagnostic over IP (DoIP) [14] or High-Speed Car Access (High-Speed-Fahrzeug-Zugang) (HSFZ) and Ethernet [10]. The General Motor Local Area Network (GMLAN) protocol combines transport and application layer specifications very similar to ISO-TP and UDS. Because of that similarity, identical application layer-specific scan techniques can be applied. To overcome the bandwidth limitations of CAN, the latest vehicle architectures use an Ethernet-based diagnostic protocol (DoIP, HSFZ) to communicate with a central gateway ECU. The central gateway ECU routes application layer packets from an Ethernet-based network to a CAN based vehicle internal network. In general, the diagnostic functions of all ECUs in a vehicle can be accessed from the On Board Diagnostic (OBD) connector over UDSonCAN or UDSonIP [12].

4 Threat Definitions

We've analyzed published attacks on automotive systems and combined these findings with a threat analysis of the specifications for GMLAN and UDS. This allows the creation of general threat definitions for automotive diagnostic protocols substantiated by proven security flaws. The proposed threat definitions are based on the Common Vulnerabilities and Exposures (CVE) flaw terminology [21]. Since CVE flaw types are initially designed for web applications, office, and personal computer systems, a new keyword is added. The keyword **phys** indicates that this service of a diagnostic protocol can cause a physical action on a vehicle, becoming important if analysis regarding the safety-criticality of supported services of an ECU is performed. In table 2, all functions and features (also called services) of GMLAN and UDS are annotated with possible or proven CVE flaw types. These threat definitions allow mapping services of an ECU to possible security flaws to perform an

Table 2. Threat definitions for the diagnostic protocols UDS and GMLAN. Each relevant service (referenced by the hexadecimal service identifier) is mapped to one or multiple possible CVE flaw types. The description gives an explanation why a certain flaw type is possible. A list of all service names of UDS and GMLAN can be found in appendix A.1 in table 10.

UDS	GMLAN	Type	Descriptions and references for the combination of flaw types with UDS/GMLAN services
10h	10h, A5h	*dos-flood*	These commands will change the session of an ECU. This command's actual impact can reach from no effect in the functionality to the execution of a different firmware or the ECUs bootloader. Miller & Valasek and Nie et al. used this service to disable (Denial of Service (DoS)) individual ECUs [20, p. 9][25, p. 12].
11h		*dos-flood*	During a reset, an ECU is unavailable. Researchers from Keen Labs were able to trigger this function at any speed of a vehicle. Unavailability of safety-critical ECUs in extreme driving conditions can cause serious dangers [4, p. 28].
19h, 22h, 23h, 24h, 2Ah, 2Ch, 86h	12h, 1Ah, 22h, 23h, 2Ch, 2Dh, A9h, AAh	*infoleak*	These commands can be used to gather internal information about an ECU. This can be used to obtain static information (Vehicle Identification Number (VIN), software versions, etc.), dynamic information to understand the internal behavior of an ECU, or even to extract the entire firmware [22].
27h	27h	*crypt* *pass* *rand*	Van den Herrewegen et al. and Dürrwang et al. demonstrated impacts of weak cryptographic implementations [9,6]. Miller & Valasek revealed many hard-coded cryptographic secrets inside an ECUs firmware [18, p. 46]. Nie et al. analyzed weak security access implementations and showed the lack of random seed creation [25, p. 11].
28h	28h	*dos-flood*	This service grants the total bandwidth of the CAN bus to only one ECU. Attackers can prevent ECUs from communicating, which causes a DoS of the attacked ECU [16, p. 7].
2Dh		*int-overflow* *buf*	his service specification describes two possible use-cases, clearing of non-volatile memory and changing of calibration values [11, p. 147]. Both use-cases can be used to cause program flow corruptions, e. g. integer- or buffer-overflows. See above. Identical to *int-overflow*.
2Eh	3Bh	*int-overflow* *phys* *buf*	Identifiers can be any payload. The protocol specifications are very generic for these commands. If a data-identifier is mapped to numeric values, it might be possible that these values can trigger execution errors, such as integer overflows. Cai et al. demonstrated the manipulation of the driver's seat position through this service [4, p. 8]. Payloads can contain complex data, e. g. certificates or ring buffer contents. Increasing data size and complexity leads to more likelihood of security flaws in interpreters and parsers. Additionally, writable memory areas allow attackers to place exploit code into known and defined memory sections.
2Fh		*phys*	Miller & Valasek demonstrated the control of a vehicle's pre-collision system seat belt functionality. This proves the possibility to trigger physical actions through this service [18, p. 15].
31h		*dos-flood* *buf* *phys* *infoleak*	Miller & Valasek identified sub-functions that allow the erase of an ECUs memory. Such an operation would brick an ECU and lead to the entire vehicle's unavailability [20, p. 12]. RoutineControl jobs accept individual payloads with various lengths. The more complex data leads to a higher likelihood of implementation flaws. Cai et al. demonstrated an insecure implementation, combined with a time-of-check to time-of-use (TOCTOU) attack, which led to code execution [4, p. 8]. RoutineControl jobs can be used to control actuators on a vehicle. Miller & Valasek were able to kill a vehicle's engine [18, p. 51]. Dürrwang et al. showed the deployment of airbags through insecure implementations of RoutineControl jobs [6]. The sub-function `requestRoutineResults` can potentially leak sensitive data.
34h	34h	*upload*	These commands are intended to initiate a software update. Miller & Valasek and Van den Herrewegen et al. demonstrated arbitrary code execution by abusing this command [20,9].
35h		*infoleak*	This command could be used to leak internal information of an ECU.
36h, 84h	36h	*buf*	These commands are part of the update process. An implementation flaw is unlikely; nevertheless, buffer overflow vulnerabilities are potentially possible.
87h		*dos-flood*	Allows the modification of communication parameters. Attackers can prevent an ECU from communicating by providing an invalid configuration.
	AEh	*phys* *dos-flood*	Koscher et al. demonstrated the possibility of triggering physical actions on ECUs [16, p. 8]. The GMLAN standard describes the possibility to trigger an ECU reset [7].

Table 3. Summary of state modifying services in UDS and GMLAN.

	UDS		GMLAN
`10h`	DiagnosticSessionControl	`10h`	InitiateDiagnosticOperation
`11h`	ECUReset		
`27h`	SecurityAccess	`27h`	SecurityAccess
`28h`	CommunicationControl	`28h`	DisableNormalCommunication
`31h`	RoutineControl		
		`34h`	RequestDownload
`3Eh`	TesterPresent	`3Eh`	TesterPresent
		`A5h`	ProgrammingMode

in-depth impact analysis of a diagnostic protocol implementation. Table 1 proposes a measurand and a measuring unit for every flaw type to achieve a comparable rating.

5 Automotive Diagnostic Protocol Scanner

This section describes the implementation of an automotive diagnostic protocol scanner. The Scanner uses an active automata learning technique to reverse engineer the ECU's system state machine automatically. This allows the creation of a black-box testing strategy solely based on the application layer communication. Through dynamic reverse engineering of the system state machine of an ECU, it is possible to scan an increased attack surface. Differences in an ECUs communication can be modeled with a system state graph that describes the ECUs behavior.

5.1 System States

UDS and GMLAN have a very similar protocol structure. Since various OEMs use UDS, the concrete implementations and, therefore, an ECU's behavior varies between ECUs from different OEMs. The GMLAN standard is much more descriptive in terms of an ECU's communication behavior. The current state of an ECU defines its communication behavior. Most modern ECUs have the possibility to execute at least two different types of software to fulfill safety requirements and update features. One software is called a bootloader; the other one is an ECU's application software for normal operations. Every software component of an ECU will show a different attack surface derived from a different set of supported protocol services. Furthermore, bootloader and application software is often developed by different suppliers. For diagnostic purposes, an ECU's application software supports a diagnostic mode, often with capabilities to trigger physical actions. The security access service is used for authentication to unlock protected services. Both security access and diagnostic mode can change the entire communication behavior and, therefore, the ECU's attack surface. On some ECUs, the communication behavior already changes as soon as a `TesterPresent` message is sent. Every variation of the communication behavior will be called a **state of an ECU** and represented by a node in the system state graph.

Since every state of an ECU can support a different set of services, it implies that certain services that modify the state (transitions in the system state machine) can become available only in a specific state or under special conditions. Therefore a full scan for supported services must be performed in every identified state of the system state machine.

5.2 Transitions In the System State Graph

During the implementation of the protocol scanner with active state learning, several conditions that alter an ECU's internal state were identified. The following services can trigger state transitions in GMLAN and UDS. The collection in table 3 is not necessarily complete and can vary for individual ECUs. Especially for UDS-based ECUs, OEMs often implement their custom protocol specifications

or additions. To anticipate this circumstance, the software architecture of the implemented Scanner is highly extendable.

Reset State The scan algorithm needs a reliable way to bring an ECU under test back into its reset state, even if the ECU entered an unknown state. Undocumented commands could cause an undesired state change during a scan. Through a reliable reset function, misbehavior can be identified, and interruptions of further scans can be prevented with frequent resets of the scan target. A power cycle of an ECU is used in later tests as a reliable reset function, which is easy to implement and sufficient since the proposed scan does not alter non-volatile memories.

Return Code Evaluation UDS and GMLAN follow a strict communication scheme. Every request to a scan target triggers a response if the request does not explicitly suppress the response. Two response types are possible, either a positive or a negative one.

If a negative response is received, the state of a scan target is not changed. The return code of negative responses can be used to identify the reason why a specific request failed. This return code can leak information about the possible attack surface.

The reception of a positive response indicates the successful execution of the request. If the scan algorithm, for example, has sent a `DiagnosticSessionControl` request, it can identify a state change of the scan target from a positive response. Furthermore, the scan algorithm now knows the previous state, the new state, and the corresponding transition function to trigger this state change. The transition function, in this case, is simply a `DiagnosticSessionControl` request with a specific parameter. The algorithm can append this new state onto the previous state in its internal system state graph. On the next scan iteration, the scan target can be set into the new state by concatenating all transition functions necessary to enter the desired system state.

Security Access Testing System states which granted security access are crucial for attack surface evaluations. `SecurityAccess` routines in diagnostic protocols are used to grant further privileges to repair shop testers during ECU development or vehicle production. These privileges may open new attack surfaces once they are gained. Through manual reverse engineering steps on the investigated ECUs, multiple security access algorithms could be obtained. The actual implementations of the analyzed security access functions are entirely different between the individual OEMs. A categorization of the reverse-engineered security access functions delivered the following groups:

- **Simple Arithmetic Operations**
  This group contains security access algorithms based on single arithmetic operations such as XOR, NOT, or ADD with a fixed value. Examples are given by the work of Dürrwang et al. and Nie et al. [6,17].
  $$key = \neg seed \tag{1}$$
- **Mathematical Operations**
  The security access mechanism of one analyzed OEM relies on complex mathematical operations. To obtain a key for this ECU, one needs to know five different numeric values which act

as a shared secret. With this secret, a random seed has to be multiplied in different ways to obtain a valid key. An example operation for this group can be the following:

$$key = (seed * secret1 + secret2) \bigoplus (seed * secret3 + secret4) \bigoplus secret5 \tag{2}$$

- **Proprietary XOR-Shift-Loop**
  Security access algorithms for this group were analyzed in-depth by Van den Herrewegen et al. [9]. Their publication provides examples as well as a cryptographic analysis.
- **Cryptographic Operations**
  One analyzed OEM relies on cryptographic authentication mechanisms for its security access algorithms. The following equation shows an example:

$$key = RSA_{sign}(MD5(seed \mid salt), private\_key) \tag{3}$$

Each group of security access functions has a different probability of being broken over an ECUs lifetime. The implemented scanner algorithm can automatically test known or trivial security access functions. If a positive response is received after sending a key to the ECU under test, this `SecurityAccess` routine is stored as a transition function to enter this new state with granted security access. Additionally, this new state gets inserted into the system state graph of the scanner algorithm.

Time-Dependent State Changes The `TesterPresent` command has a time-dependent return function implemented. After a `TesterPresent` request is acknowledged from a scan target, the scan target remains in this state for a fixed amount of time. After five seconds, the scan target automatically leaves the `TesterPresent` state, which also involves a return to the default diagnostic session.

Summary The explained properties of system state graphs in automotive diagnostic protocols can be defined as follows:

Definition 1. *A system state machine M is a directed graph (S, E, Δ), with the following properties:*

- $S = \{s_0, s_1, \ldots, s_n\}$ *is a finite set of nodes, each node represents a system state.*
- *The state s_0 is defined as the default system state after the power-up of the system.*
- $E = \{(v, w) \in S^2\}$ *is a set of ordered pairs of nodes, called directed edges.*
- $\Delta = \{\delta_0, \delta_1, \ldots, \delta_k\}, \delta_k : S^2 \mapsto S, \delta_k(v, w) = z$ *is a set of transitions functions for each $e \in E$.*
- *For each state $s_i \in S \setminus \{s_0\}$ a reset function $\delta_k \in \Delta, \delta_k(s_i, s_0) = s_0$ is given through the power cycle of the system.*

5.3 Exploration Algorithm for Reverse-Engineering of System States

The scan algorithm for application layer protocol scans consists of two different parts. For every diagnostic service, a unique module with service-specific knowledge exists. This module performs the enumeration of all sub-functions and can evaluate responses. This module will be called “Enumerator”. Enumerators store all scan results of a specific service and map the results to a state.

Furthermore, an Enumerator keeps track of which states it has been executed. This is necessary to know if an Enumerator has finished its scan of a service.

Algorithm 1: Enumerator

```
Data: current system state s_i
forall sub-function in service do
    test sub-function;
    store result;
    evaluate return code;
    if state changed then
        create s_j, δ_k(s_i, s_j) = s_j;
        add s_j, δ_k(s_i, s_j) to M;
        return;
    end
end
s_i finished;
return;
```

Algorithm 2: Scanner

```
forall en in enumerators do
    forall s_i in S do
        if en finished for s_i then
            continue;
        end
        reset target to s_0;
        call δ_k(s_0, s_i);
        if not entered s_i then
            continue;
        end
        execute en(s_i);
    end
end
```

The second part is called Scanner, which stores a scan target's system state machine as a directed graph with transition functions. Through a reset function, the Scanner can reliably reset the scan target on each scan iteration. After each reset, the Scanner computes all known states of the scan target from its system state graph. The shortest path algorithm delivers the minimal transitions necessary to set the scan target into the desired state. If a system state manipulation were successful, the next, not finished enumerator would be executed for the desired state.

If an Enumerator detects a state modification of the scan target through return code evaluation, a new state is inserted into the Scanner's system state graph. The last operation performed on the target is identified as a transition function to enter this new state. Now the Scanner knows a new system state of the scan target and the necessary transition function to set the scan target into this state. Every other enumerator will be executed in this newly identified system state on the next iteration of the Scanner.

The separation in Enumerator and Scanner objects opens the algorithm for additions and customization. If one focuses on ECUs from a specific OEM, he might have access to knowledge under Non-Disclosure Agreements (NDAs). `SecurityAccess` algorithms are one example of such knowledge. The object-oriented way in which the scanner software is written allows the implementation of custom Enumerators. This enables researchers to implement, e.g., custom `SecurityAccess` algorithms into a new Enumerator class. A proprietary `SecurityAccess` enumerator object can be provided to the Scanner, which increases the scan depth and will add further states to the system state graph.

6 The Attack Surface Model for Automotive Diagnostic Protocols

Combining the previously introduced threat definitions (table 2) with the proposed system state machine (definition 1) generates an extensive attack surface model for automotive diagnostic protocols. The Scanner can measure all possible threats for each system state while reverse-engineering

the system state machine. This allows performing threat estimations concerning the system state machine. Furthermore, threat evolution over a system's lifetime can be evaluated through Cumulative Distribution Functions (CDFs).

Definition 2. *An attack surface model* (M, R) *for an automotive diagnostic protocol implementation contains:*

- *A system state machine* M *of a scanned ECU.*
- *A set* R *of threat measurements, in which every threat measurement is defined as a tuple* $r_k = (f, S, ser, sub)$*, that contains a flaw type* f *according to table 2, a set of system states* $S = \{s_i, s_j, ... \mid s_i, s_j \in M\}$*, the service identifier* ser *and the sub-function identifier* sub *according to the protocol specification.*

This definition of a attack surface model allows evaluation of the exploitation risk over an ECU's lifetime and, therefore, for an entire vehicle. Just one addition to the definitions of the system state machine M is required. In the performed scans on real-world ECUs, multiple different types of transition functions in the reverse-engineered system state machines were found, for example, reset of an ECU, change of the diagnostic session, or security access authentication. Only security access transitions rely on authentication mechanisms that are relevant for analysis over the system's lifetime. Let X be a function that describes the time it takes until a security access algorithm is successfully attacked. $\mathbb{P}(X \leq t)$ denotes the probability that a successful attack occurs within time t. Call $F : \mathbb{R} \mapsto [0, 1]$ given by $F(t) = \mathbb{P}(x \leq t)$ the CDF of X. To evaluate systems with more than one security access function, the following operations are required in this model. Let $F_1(t)$ and $F_2(t)$ be two CDFs for independent random variables, the operations summation, maximum and minimum are defined as follows: $F_{sum}(t) = \int_0^t F_1(t-x)F_2(x)dx$, $F_{max}(t) = F_1(t)F_2(t)$, and $F_{min}(t) = 1 - (1 - F_1(t))(1 - F_2(t))$ [1]. Every transition function $\delta_k(v, w)$ in the system state machine can be extended with a CDF to describe this transition's behavior over time.

Defining a proper CDF for each security access algorithm is a challenging task on its own. Furthermore, the CDF is also dependant on the implemented mitigation on a system level. Some analyzed security access implementations, for example, are vulnerable to brute force attacks; others not. An in-depth analysis of every individual target is required to obtain a suitable CDF. Additionally, a comprehensive analysis to identify a CDF for a security access algorithm also includes studying an OEMs key management in repair shops, factories, and suppliers' production sites. The benefit of the proposed model lies in the simplicity and coverage of its analysis. One, interested in the system's security over a lifetime, only needs to identify one proper CDF for each security access algorithm in his system. All further analysis can then be performed automatically, based on the gathered attack surface model resulting from the automated scan. The necessary steps are the following:

1. Let B be a directed graph (V, E, Δ). B is built from M by the following steps:
   - V is a finite set of vertices obtained from the following operations:
     - From a given system state machine M, remove all edges e_k with a security access transition function $\delta_k \in \Delta$. This returns a set V of n disjoint sub-graphs. Every sub-graph $v_k \in V$ contains multiple system states $v_k = \{s_i, s_j, \dots \mid s_i, s_j \in M\}$.
     - $E = \{(w, z) \in V^2\}$ is defined as set of ordered pairs of vertices. In this case, all security access transitions.

    - Δ is a set of CDFs, one for each edge $e_k \in E$. The sub-graph $v_0 \in V$, which contains the system state $s_0 \in M$ obtains the $CDF : F(t) = 1$, since all system states s_k in this sub-graph v_0 are immediately reachable.
  - Let F_i be the CDF of the sub-graph v_i.
  - Vertices $v_k \in V \setminus \{v_0\}$ get a CDF defined by $F(t) = \sum F_i(t)$ for every F_i in a shortest path from v_0 to v_k.
  - Since every system state $s_k \in M$ is contained in only one sub-graph $v_k \in B$, the CDF of the sub-graph v_k also applies for all system state $s_k \in v_k$.
2. Each threat tuple r_k can contain multiple system states in its set S. The CDF of a threat tuple is defined by $CDF : F_{max}(t) = \prod F_i(t),\ \forall F_i$ assigned to each system state $s_k \in S$.
3. Define a suitable CDF for each security access algorithm.
4. The behavior over time of all measured threats, reachable over security access functions, can now be modeled by the corresponding CDF.

Additionally, for security investigations, it is rewarding to identify possible threats that are only present in system states, reachable through a security access transition. OEMs protect these services and sub-functions by security access algorithms which indicates privileged functionalities.

7 Hardware Architecture and Test Setup

As part of the conducted research, a cheap and scalable test setup to perform automated scans on different ECUs was built. Raspberry Pi 4B single board computers, equipped with two CAN interfaces for communication and a relay to control the ECUs power supply, were used as the hardware interface to ECUs under test. The Raspberry Pis are operated with the latest Raspbian OS. For ISO-TP support, the can-isotp Linux kernel module was used [8]. No modifications to the Operating System (OS) were made. All timing measurements were performed from user-land Python applications. For scans performed over DoIP or HSFZ, an Ethernet connection and standard User Datagram Protocol (UDP) and Transmission Datagram Protocol (TCP) sockets were sufficient. 13 different ECUs (shown in table 4) from five different OEMs were installed into hardware in the loop test setup to verify the OEM independence of the implemented scan algorithm. This setup contains ECUs from Daimler AG, Tesla Inc., Opel Automobile GmbH, Volkswagen AG, and BMW AG. On every investigated ECU, the following manual installation steps were required:

1. The **power supply** connector pins of an ECU had to be identified.
2. **CAN or DoIP interface** pins needed to be identified.
3. Some ECUs require periodic **keep-alive CAN messages** to be sent or Ethernet activation line signals applied.

8 Evaluations

This section discusses gathered results from the analyzed ECUs as well as the application of our proposed attack surface model over a systems lifetime.

Table 4. Overview of investigated ECUs. A label **E{x}** is assigned to each ECU for later reference.

Ref.	OEM	ECU Type	Part No.
E1	BMW AG	Gateway ECU	LR-01
E2	BMW AG	Body Domain Controller (BDC)	LR-01
E3	BMW AG	Gateway ECU	9243211
E4	BMW AG	Telematics Control Unit (TCU)	9342881
E5	VW AG	Body Control Module (BCM)	5WA93
E6	VW AG	Dashboard ECU	5G0920961A
E7	Opel GmbH	Airbag ECU	13575447
E8	Opel GmbH	BCM	13588153
E9	Tesla Inc.	Airbag ECU	1031642
E10	Daimler AG	Gateway ECU	1679012003
E11	VW AG	Antenna ECU	AU651
E12	VW AG	Gateway ECU	80B907468B
E13	VW AG	Airbag ECU	T15XX164919

Table 5. Overview of reverse-engineered system state machine complexities for all analyzed ECUs. Row *Edges* does not contain the reset edges through the power cycle. Row *Security Access (SA)* indicates the number of different security access algorithms that were reverse-engineered.

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13
Edges	15	7	9	7	9	23	23	22	13	19	11	32	5
Nodes	9	5	5	5	5	6	10	8	6	8	6	11	3
SA	3	1	2	1	0	0	1	1	1	2	0	2	0

8.1 Automated System State Reverse Engineering

The presented scan algorithm was able to identify multiple different system states for each tested ECU. All ECUs showed individual system state machines, even if the same manufacturer developed them. As an example, figure 2 shows two different system state machines with all transitions. This gives an impression of how different system state graphs can be. Table 5 provides a comprehensive overview of the complexity of all reverse-engineered system state graphs and indicates the number of security access algorithms known from the scanner utility.

8.2 Detection of Bootloaders

Five tested ECUs showed a significant behavior change in the measured communication timings on different injected system states. Through manual reverse engineering, it could be proven that these ECUs can enter the bootloader if the correct sequence of commands is sent. ECUs which implement the GMLAN protocol showed this change after a `RequestDownload` service request. ECUs with UDS support could be forced into the bootloader through a successful `DiagnosticSessionControl` command with `DiagnosticSessionType=ProgrammingSession` as the parameter. Other ECUs of the presented test setup required additional proprietary commands or security access authentication to unlock the bootloader mode.

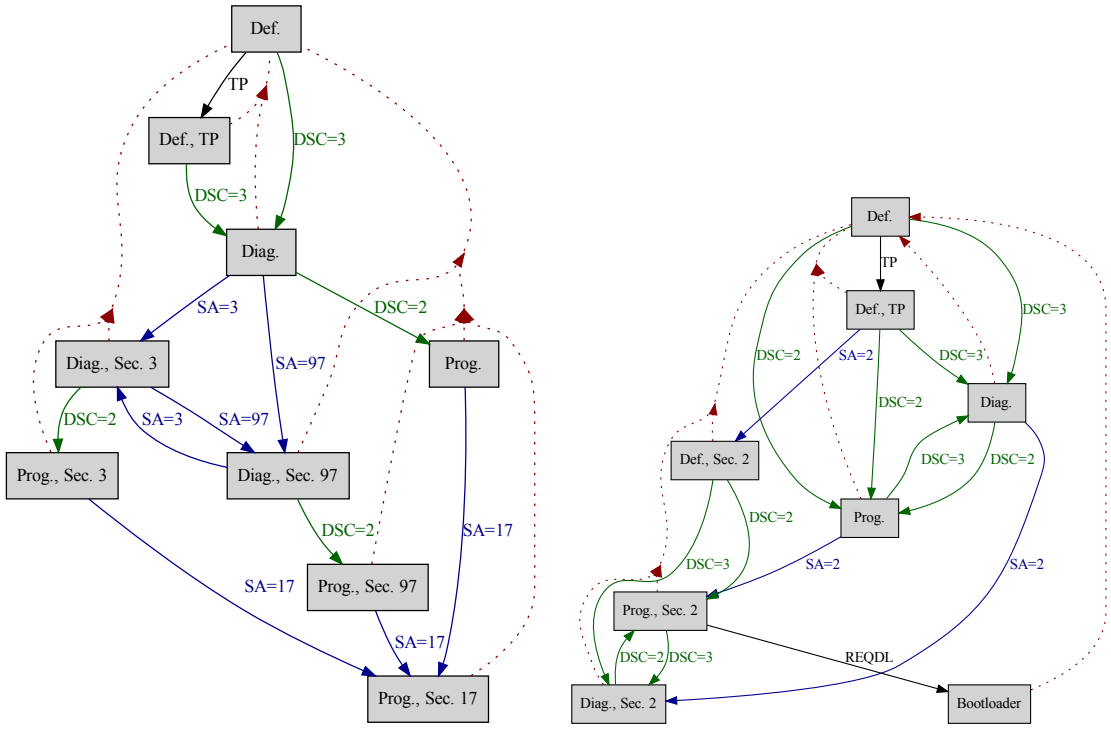

Fig. 2. ***Left:*** Automatically reverse-engineered system state graph of ECU E1. ***Right:*** Automatically reverse-engineered system state graph of ECU E7. ***Both:*** Reset through power cycle is represented by the red dotted lines. Blue lines indicate Security Access (SA) authentication. Diagnostic Session Control (DSC) transitions are shown by the green lines. TP stands for Tester Present and REQDL for Request Download.

Table 6. Average response time for negative responses and number of samples. Timings of five different ECUs in the default and the bootloader state.

	Default	Bootloader
E1	0.89 ms, 65k	0.78 ms, 65k
E7	20.5 ms, 1.9k	8.18 ms, 0.7k
E8	7.36 ms, 7.6k	0.60 ms, 2.4k
E10	6.00 ms, 65k	0.61 ms, 65k
E12	1.16 ms, 65k	0.73 ms, 65k

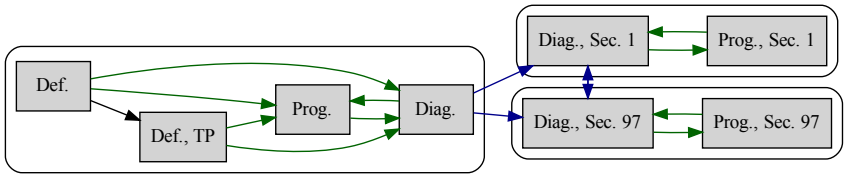

Fig. 3. Automatically reverse-engineered system state graph for ECU E10 without reset transitions. Colors and abbreviations are identical to figure 2. Clusters indicate sub-graphs with unique security access levels.

Negative response messages have a fixed size and fit in a single CAN frame. Therefore the timings of negative responses, shown in table 6, are more comparable to positive responses' timings. The ECUs E1, E8, E10, and E12 communicate with 500 kbit/s CAN speed, E7 with 33.3 kbit/s CAN speed. This explains the higher average response times of E7. The measurements in table 6 show a significant change in an ECUs communication behavior, caused by the execution of different firmware. Since the bootloader has a much smaller codebase than the application firmware, it is reasonable that the response times significantly decrease. A simple average computation is sufficient to detect the execution of different firmware automatically. The fact that a bootloader firmware is often developed by a different software supplier and sold as a product implies that different ECUs from different OEMs can have an identical bootloader firmware. This leads to the possibility of common vulnerabilities, respectively, an identical attack surface, between different ECUs from different OEMs.

8.3 Attack Surface Increase

The scan results of ECU E10 in table 7 are discussed, and a more general overview of the results of all tested ECUs is provided in table 8 to demonstrate that the proposed scan algorithm can automatically explore an increased attack surface on diagnostic protocols. Figure 3 shows all system states of ECU E10 with its possible transitions. For each system state, table 7 provides detailed measurements. These measurements are grouped by their possible flaw type from all executable services per individual system state. All flaw types, which count the number of available sub-functions as measurand, accumulate all positive responses and negative responses with the response code `Incorrect message length or invalid format (0x13)`. This negative response code indicates the availability of a sub-function, the previously sent request does not match the required format, which is caused by the proprietary implementations of sub-functions. Nevertheless, once this negative response code is received, some manual reverse engineering or automated request mutation is required to trigger this sub-function successfully. Table 7 clearly shows that each state supports a different set of services and sub-functions, which leads to different attack surfaces. To further underline this statement, table 8 provides an overview of the attack surface of all investigated ECUs. For readability, only two states per ECU are shown. **State** s_0 indicates the default state of an ECU. Row S stands for the set of all automatically explored system states through the scanner algorithm. In most cases, the measured values and, therefore, the attack surface increases.

The scanner algorithm was able to identify the necessary system state for software updates on seven different ECUs, indicated by the *upload* column. Another remarkable identification is the increased *infoleak* on E7 and E8. During these scans, it was possible to dump the ECU firmware by abusing the `ReadMemoryByAddress` service automatically. For all tested ECUs, except E4 and E9,

Table 7. Detailed threat model for ECU E10.

State	buf	dos-flood	infoleak	int-overflow	phys	rand/pass/crypt
Def. Session s_0	0	6	13958	0	2	0
s_0, TesterPresent	0	6	16133	0	2	0
Diag. Session	6392	81	16270	6392	169	2
Diag. Session, Sec. level 1	6405	81	16270	6405	171	2
Diag. Session, Sec. level 97	6405	81	16270	6405	171	2
Prog. Session	2	9	13961	2	6	1
Prog. Session, Sec. level 1	6407	83	16272	6407	174	2
Prog. Session, Sec. level 97	6407	83	16270	6407	174	3

the number of sub-functions that could trigger physical actions (column `phys`) increased system states' injection.

8.4 Threats over Time Evaluation

As a final evaluation, the proposed attack surface model's capabilities for lifetime security analysis will be discussed. ECU E1 supports three different security access levels. First, the protected attack surface per security access level is analyzed. All threat tuples $r \in R$ with a set of system states S only containing system states reachable through a security access transition are selected. Table 9 shows that security access level 17 is dedicated to the software update service. Security levels 3 and 97 protect sub-functions with the possibility to trigger physical actions and protect information. A CDF to model the time until a security access function is successfully attacked can be obtained either from an analysis of historical data or on the basis of expert opinion [1]. For this example evaluation, the exponential distribution $exp(1/t)$ is chosen to model the meantime t until a successful attack. Assuming that each security level has an individual resistance against attacks expressed by exponential distribution functions with different success rates. A unit less value for t is used, since the objective is only a demonstration of the models capabilities. A CDF for a real world system would contain a proper unit for t.

Figure 4 shows the individual CDFs for each security access level and their application to the measured attack surface. This estimates the attack surface over 20 time-units t, supposed that the meantime until a security access algorithm is broken, behaves like the proposed CDFs. For example, let $F_3(t)$ and $F_{97}(t)$ be the CDFs to model the meantime until security level 3, respectively security level 97, will be available for an attacker. The expected attack surface for a `dos-flood` at a certain time t is given from $F_{df}(t) = 4 \cdot F_3(t) + 1 \cdot F_{97}(t)$. Multipliers are obtained from the measurements in table 9. If a more realistic evaluation is required, one can construct a detailed attack tree for each security access algorithm and derive a CDF. Besides, it should be noted that a successful attack of the *upload* attack surface on the latest ECUs requires a further vulnerability to leverage the firmware signature mechanism, which results in an additional attack step. Nevertheless, previously referenced real-world attacks show that not every ECU implements firmware signatures.

9 Conclusion & Further Work

Our paper shows the capabilities of automated security scans in automotive diagnostic protocols, in combination with a graph-based model for attack surface exploration and threat estimation. We

Table 8. Overview of identified potential attack surface per ECU. Rows with state s_0 stands for default session, rows with state S describes the combination of all identified system states.

ECU	States	buf	dos-flood	infoleak	int-overflow	phys	rand/pass/crypt	upload
E1	s_0	443	52	1780	443	55	0	0
	S	±0	+23	+280	±0	+21	+3	+1
E2	s_0	17	7	255	17	8	0	0
	S	+17	+2	±0	+17	+1	+1	+1
E3	s_0	375	19	2984	375	24	0	0
	S	±0	+17	+5	±0	+15	+2	0
E4	s_0	0	6	666	0	4	0	0
	S	±0	+1	±0	±0	±0	+1	+1
E5	s_0	101	14	8157	101	20	0	0
	S	+68	+37	+26	+68	+50	±0	±0
E6	s_0	0	4	1221	0	0	0	0
	S	+1	+10	+859	+1	+20	+2	±0
E7	s_0	0	3	747	0	0	0	0
	S	+20	±0	+394k	+20	+4	+1	+1
E8	s_0	145	3	2.2M	145	43	0	0
	S	+26	±0	+12.6M	+26	+4	+1	+1
E9	s_0	0	2	0	0	0	0	0
	S	±0	+1	+2559	±0	±0	+1	+1
E10	s_0	0	6	13958	0	2	0	0
	S	+6407	+78	+2313	+6407	+172	+3	±0
E11	s_0	4	5	3637	4	2	0	0
	S	+5	+4	+1	+5	+7	+1	±0
E12	s_0	0	8	30001	0	6	0	0
	S	+35	+50	+6709	+35	+56	+2	+1
E13	s_0	24	5	706	24	5	0	0
	S	±0	+2	+420	±0	+2	+1	±0

Table 9. Attack surface metrics protected by individual security access levels for ECU E1.

Security level	dos-flood	infoleak	phys	upload
3	4	1	4	0
17	0	0	0	1
97	1	146	1	0

could show that ECUs have a different attack surface, depending on their internal system state. Active automata reverse engineering techniques enable the proposed scan algorithm to automatically discover system states during a black-box scan. Through active detection and stimulation of system states, the algorithm can perform a more comprehensive analysis of an ECUs or a vehicle's attack surface. The introduced metric assists security researchers to rate the possible attack surfaces of ECUs and allows them to analyze their evolution over a lifetime. Additionally, the attack surface model can point researchers or penetration testers to safety- and security-critical services. With published open-source tools, we want to help developers to speed up automated attack surface

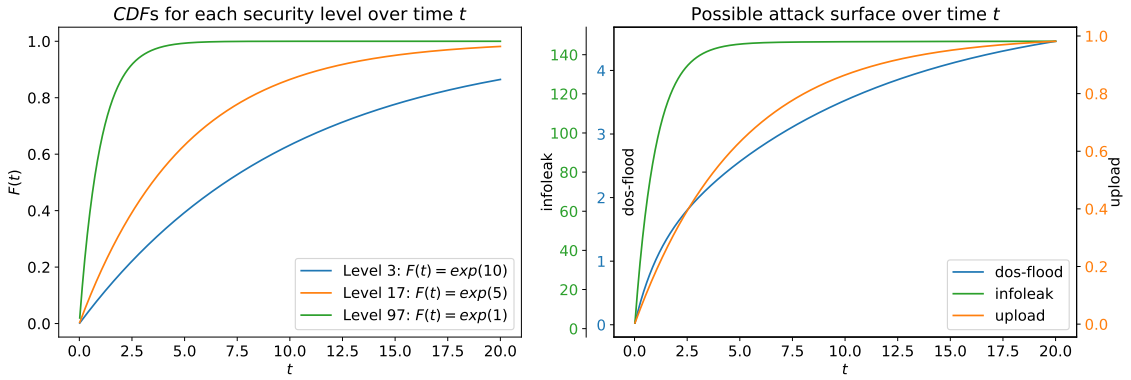

Fig. 4. ***Left:*** Three CDFs of different exponential distribution functions to model the meantime t until a successful attack of the corresponding security access algorithm for three different security access levels. ***Right:*** Evaluation of the attack surface over time t for measured attack surfaces protected by different security access algorithms. ***Both:*** The x-axis indicates the time t, the y-axis shows the expected value of the attack surface metric. For demonstrational purposes the x-axis is unitless since no specific time-unit was defined by the chosen $CDFs$.

discovery for automotive diagnostic protocols to lower the attack surface of future vehicles.

Our scan algorithm can be used to collect detailed information about the implementation of a diagnostic protocol of an ECU. This could be extended to enable device and firmware fingerprinting during a scan. Common vulnerabilities in automotive systems could automatically be tested through custom Enumerator objects. Our open-source tool provides a basis for further automated protocol scans in automotive networks. Furthermore, a reverse-engineered system state machine could potentially be used to increase the efficiency of protocol fuzzers.

Availability

We provide all our tools for application layer scans in automotive networks as part of the open-source software project Scapy [3]. Our work aims to support the automotive security community to solve future security challenges.

Acknowledgments

The present work as part of the PetS^3 project was funded by the Bavarian Ministry of Economic Affairs, Regional Development and Energy (Bayerisches Staatsministerium für Wirtschaft, Landesentwicklung und Energie) under grant number IUK-1711-0018. The authors are responsible for the content of this publication. We gratefully thank Gabriel Potter, Guillaume Valadon, and Pierre Lalet, the maintainers of the Scapy framework, for their support during the development of our tools, and finally Philippe Biondi, the author of Scapy.

References

1. Arnold, F., Hermanns, H., Pulungan, R., Stoelinga, M.: Time-Dependent Analysis of Attacks, Lecture Notes in Computer Science, vol. 8414, p. 285–305. Springer Berlin Heidelberg (2014). https://doi.org/10.1007/978-3-642-54792-8_16, http://link.springer.com/10.1007/978-3-642-54792-8_16
2. Bayer, S., Ptok, A.: Don't fuss about fuzzing: Fuzzing controllers in vehicular networks. In: escar 2015. p. 10 (2015)
3. Biondi, P., Valadon, G., Lalet, P., Potter, G.: Scapy (2018), http://www.secdev.org/projects/scapy/ (accessed 2021-04-14)
4. Cai, Z., Wang, A., Zhang, W.: 0-days & Mitigations: Roadways to Exploit and Secure Connected BMW Cars. In: BlackHat USA. pp. 1–37 (Aug 2019), https://i.blackhat.com/USA-19/Thursday/us-19-Cai-0-Days-And-Mitigations-Roadways-To-Exploit-And-Secure-Connected-BMW-Cars-wp.pdf (accessed 2021-04-14)
5. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T.: Comprehensive Experimental Analyses of Automotive Attack Surfaces. In: Proceedings of the 20th USENIX Conference on Security. pp. 1–6. SEC'11, USENIX Association, USA (2011)
6. Dürrwang, J., Braun, J., Rumez, M., Kriesten, R., Pretschner, A.: Enhancement of Automotive Penetration Testing with Threat Analyses Results. SAE International Journal of Transportation Cybersecurity and Privacy **1**(2), 91–112 (Nov 2018). https://doi.org/10.4271/11-01-02-0005
7. (GMW), G.M.W.: General Motors Local Area Network Enhanced Diagnostic Test Mode Specification. Standard GMW3110, General Motors Worldwide (GMW) (2018)
8. Hartkopp, O.: Linux Kernel Module for ISO 15765-2:2016 CAN transport protocol (2020), https://github.com/hartkopp/can-isotp (accessed 2021-04-14)
9. Van den Herrewegen, J., Garcia, F.D.: Beneath the Bonnet: A Breakdown of Diagnostic Security, Lecture Notes in Computer Science, vol. 11098, p. 305–324. Springer International Publishing (2018). https://doi.org/10.1007/978-3-319-99073-6_15, http://link.springer.com/10.1007/978-3-319-99073-6_15
10. IEEE: IEEE Standard 802.3-2018 - Standard for Ethernet (2018)
11. ISO Central Secretary: Road vehicles – Unified diagnostic services (UDS) – Part 3: Unified diagnostic services on CAN implementation (UDSonCAN). Standard ISO 14229-3:2012, International Organization for Standardization, Geneva, CH (2012), https://www.iso.org/standard/55284.html
12. ISO Central Secretary: Road vehicles – Unified diagnostic services (UDS) – Part 5: Unified diagnostic services on Internet Protocol implementation (UDSonIP). Standard ISO 14229-5:2013, International Organization for Standardization, Geneva, CH (2013), https://www.iso.org/standard/55287.html
13. ISO Central Secretary: Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) – Part 2: Transport protocol and network layer services. Standard ISO 15765-2:2016, International Organization for Standardization, Geneva, CH (2016), https://www.iso.org/standard/66574.html
14. ISO Central Secretary: Road vehicles – Diagnostic communication over Internet Protocol (DoIP) — Part 2: Transport protocol and network layer services. Standard ISO 13400-2:2019, International Organization for Standardization, Geneva, CH (2019), https://www.iso.org/standard/74785.html
15. ISO Central Secretary: Road vehicles – Cybersecurity engineering. Standard ISO/SAE DIS 21434, International Organization for Standardization, Geneva, CH (2020), https://www.iso.org/standard/70918.html
16. Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S.: Experimental Security Analysis of a Modern Automobile. In: 2010 IEEE Symposium on Security and Privacy. pp. 447–462 (May 2010). https://doi.org/10.1109/SP.2010.34
17. Lab, T.K.S.: Car Hacking Research: Remote Attack Tesla Motors (2020), https://keenlab.tencent.com/en/2016/09/19/Keen-Security-Lab-of-Tencent-Car-Hacking-Research-Remote-Attack-to-Tesla-Cars/ (accessed 2021-04-14)

18. Miller, D.C., Valasek, C.: Adventures in Automotive Networks and Control Units. DEF CON 21 Hacking Conference. Las Vegas, NV: DEF CON (August 2013), http://illmatics.com/car_hacking.pdf (accessed 2021-04-14)
19. Miller, D.C., Valasek, C.: Remote Exploitation of an Unaltered Passenger Vehicle. DEF CON 23 Hacking Conference. Las Vegas, NV: DEF CON (August 2015)
20. Miller, D.C., Valasek, C.: Advanced can injection techniques for vehicle networks. In: BlackHat USA (Aug 2016), http://illmatics.com/can%20message%20injection.pdf (accessed 2021-04-14)
21. (MITRE), T.M.C.: Vulnerability Type Distributions in CVE – Flaw Terminology, https://cve.mitre.org/docs/vuln-trends/index.html (accessed 2021-04-14)
22. Pareja, R., Cordoba, S.: Fault injection on automotive diagnostic protocols (2018), https://www.riscure.com/uploads/2018/06/Riscure_Whitepaper_Fault_injection_on_automotive_diagnostic_protocols.pdf (accessed 2021-04-14)
23. de Ruiter, J.: A tale of the openssl state machine: A large-scale black-box analysis. In: Brumley, B.B., Röning, J. (eds.) Secure IT Systems. pp. 169–184. Springer International Publishing, Cham (2016)
24. de Ruiter, J., Poll, E.: Protocol state fuzzing of TLS implementations. In: 24th USENIX Security Symposium (USENIX Security 15). pp. 193–206. USENIX Association, Washington, D.C. (Aug 2015), https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/de-ruiter
25. Sen Nie, Ling Liu, Y.D.: FREE-FALL: HACKING TESLA FROM WIRELESS TO CAN BUS (2020), https://www.blackhat.com/docs/us-17/thursday/us-17-Nie-Free-Fall-Hacking-Tesla-From-Wireless-To-CAN-Bus-wp.pdf (accessed 2021-04-14)
26. Sommer, F., Durrwang, J., Wolf, M., Juraschek, H., Ranert, R., Kriesten, R.: Automotive network protocol detection for supporting penetration testing. In: SECURWARE. p. 6. IARIA (2019)
27. United Nations Economic Commission for Europe: The World Forum for the harmonization of vehicle regulations (wp.29), https://www.unece.org/trans/main/wp29/presentation_wp29.html (accessed 2020-05-27)

Acronyms

BCM Body Control Module. 11
BDC Body Domain Controller. 11

CAN Controller Area Network. 2–4, 10, 13
CDF Cumulative Distribution Function. 9, 10, 14
CVE Common Vulnerabilities and Exposures. 3

DoIP Diagnostic over IP. 3, 10
DoS Denial of Service. 4

ECU Electronic Control Unit. 1–16

GMLAN General Motor Local Area Network. 3–6, 11

HSFZ High-Speed Car Access (High-Speed-Fahrzeug-Zugang). 3, 10

ISO International Organization for Standardization. 1
ISO-TP Transport Layer. 2, 3, 10

NDA Non-Disclosure Agreement. 8

OBD On Board Diagnostic. 3
OEM Original Equipment Manufacturer. 1, 5–10, 13
OS Operating System. 10

TCP Transmission Datagram Protocol. 10
TCU Telematics Control Unit. 11
TLS Transport Layer Security. 2
TOCTOU time-of-check to time-of-use. 4

UDP User Datagram Protocol. 10
UDS Unified Diagnostic Service. 2, 3, 5, 6, 11
UNECE United Nations Economic Commission for Europe. 1

VIN Vehicle Identification Number. 4

A Appendix

A.1 UDS and GMLAN Service Request Identifiers

Table 10. List of hexadecimal service identifiers and according service names of UDS and GMLAN.

UDS Id.	UDS Service Name	GMLAN Id.	GMLAN Service Name
		0x04	ClearDiagnosticInformation
0x10	DiagnosticSessionControl	0x10	InitiateDiagnosticOperation
0x11	ECUReset		
		0x12	ReadFailureRecordData
0x14	ClearDiagnosticInformation		
0x19	ReadDTCInformation		
		0x1A	ReadDataByIdentifier
0x22	ReadDataByIdentifier	0x22	ReadDataByParameterIdentifier
0x23	ReadMemoryByAddress	0x23	ReadMemoryByAddress
0x24	ReadScalingDataByIdentifier		
0x27	SecurityAccess	0x27	SecurityAccess
0x28	CommunicationControl	0x28	DisableNormalCommunication
0x2A	ReadDataPeriodicIdentifier		
0x2C	DynamicallyDefineDataIdentifier	0x2C	DynamicallyDefineMessage
		0x2D	DefinePIDByAddress
0x2E	WriteDataByIdentifier		
0x2F	InputOutputControlByIdentifier		
0x31	RoutineControl		
0x34	RequestDownload	0x34	RequestDownload
0x35	RequestUpload		
0x36	TransferData	0x36	TransferData
0x37	RequestTransferExit		
0x38	RequestFileTransfer		
		0x3B	WriteDataByIdentifier
0x3D	WriteMemoryByAddress		
0x3E	TesterPresent	0x3E	TesterPresent
0x83	AccessTimingParameter		
0x84	SecuredDataTransmission		
0x85	ControlDTCSetting		
0x87	LinkControl		
		0xA2	ReportProgrammingState
		0xA5	ProgrammingMode
		0xA9	ReadDiagnosticInformation
		0xAA	ReadDataByPacketIdentifier
		0xAE	DeviceControl