Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Testování zabezpečení v sítích pro bezpečnostně kritické aplikace

Disertační práce

Nils Weiß M. Sc.

Školitel: Prof. Ing. Václav Matoušek, CSc.
Školitel-specialista: Prof. Dr. rer. nat. Jürgen Mottok

Plzeň, květen 2021

University of West Bohemia in Pilsen
Faculty of Applied Sciences
Department of Computer Science and Engineering

Security Testing in Safety-Critical Networks

Doctoral Thesis

Nils Weiß M. Sc.

Supervisor: Prof. Ing. Václav Matoušek, CSc.
Supervisor-specialist: Prof. Dr. rer. nat. Jürgen Mottok

Plzeň, May 2021

Abstrakt

Současný vývoj vozidel směřující od čistě mechanických systémů k systémům řízeným specializovanými procesory vytváří nové požadavky na oblast bezpečnosti automobilů. V dnešní době je každé moderní vozidlo vybaveno bezpečnostně kritickou realtime komunikační sítí, která zajišťuje a kontroluje všechny funkce automobilu. Neustále rostoucí konektivita automobilových systémů je ale ve stále větší míře ohrožena možnými kybernetickými útoky. Ačkoli se bezpečnostní inženýrství v této oblasti rozvíjí již několik desetiletí, zajištění bezpečnosti těchto systémů stále více vyžaduje velmi intenzivní výzkum.

Tato práce seznamuje čtenáře s procesem "black-box" analýzy existujících elektronických automobilových systémů a jejich součástí a popisuje možnosti zranitelnosti bezpečnostních systémů na základě výsledků testování čtyř různých elektronických řídicích jednotek. Z hodnocení dosavadního výzkumu bezpečnosti vnitřní počítačové sítě automobilu je spolehlivost, resp. zranitelnost, celého elektronického systému uváděna jako největší hrozba pro bezpečnost vozidla, a proto mimořádné automotivní schopnosti interních sítí automobilu jsou rozebírány v druhé části práce.

Při koncepci automatových prostředků pro automotivní sítě automobilů je nezbytné vytvořit komplexní softwarové prostředky emulující logické systémy speciálního účelu. Proto jako další část této práce byl vyvinut, implementován a též již nabízen obsáhlý open-source programový paket pro testování bezpečnosti specializovaných automobilových sítí. Další výzkumné činnosti v tomto směru jsou pak zaměřeny na výzkum bezpečnosti vozidlových sítí založené na otevřeném, popř. volně dostupném programovém vybavení.

Moderní prostředky pro zjišťování a automatickou identifikaci zdrojů kybernetických útoků jsou vytvořeny a hodnoceny v rámci navržené a realizované implementace automotivního diagnostického protokolu. Tyto prostředky umožňují vytvoření speciální metriky pro hodnocení úrovně útoku prostřednictvím "black-box" scanování

libovolné elektronické řídicí jednotky. Učící se automaty a reverzní techniky vyhodnocování stavu systému dále značně rozšiřují možnosti vytvořených programových prostředků. Navržený algoritmus byl testován na třinácti různých řídicích jednotkách, získaná data byla rozsáhle testována v laboratorních i reálných podmín-kách, objektivně vyhodnocena a výsledky testování jsou uvedeny zčásti v poslední části práce, zčásti pak v přílohách.

V závěru práce jsou pak diskutovány dosud otevřené problémy návrhu takových systémů a zaměření dalšího výzkumu založeného na výše uvedených výsledcích.

Abstract

The evolution of cars from mechanical systems to rolling computers creates new requirements for safety and security engineering. Nowadays, every vehicle contains a safety-critical real-time communication network to fulfill its function. Especially the increasing connectivity of automotive systems enlarged the attack surface for cyber-attacks. Safety engineering in this area is well understood and studied for decades, though the security engineering of these systems needs further research.

This thesis introduces a black-box investigation process to analyze existing automotive systems and components and identifies security vulnerabilities in four different ECUs. Combined with a survey of published security research, vehicle-internal networks are identified as an extraordinary threat to the vehicle's safety and security. The outstanding automation capabilities of security tests for these networks are leveraged in the second part of this thesis.

In order to create automated tools for automotive networks, a software foundation is necessary. As part of this thesis, a comprehensive open-source software framework for security testing in vehicular networks was developed and published. This aims to support further security research based on open and free software.

Novel tools for the automated identification and exploration of attack surfaces in automotive diagnostic protocol implementations are created and evaluated. These tools allow the creation of comparable attack surface metrics through black-box scans of arbitrary ECUs. Automata learning and system state reverse-engineering techniques highly increase the exploration capabilities of the presented tools. The exploration algorithm is tested on thirteen different ECUs from independent OEMs. All gathered results are evaluated and discussed in the final part of this thesis.

Finally, open issues and further research based on this contribution are discussed.

Zusammenfassung

Die Entwicklung des Autos von einem mechanischen System zu einem rollenden Computer schafft neue Anforderungen an die Sicherheit der Fahrzeugsoftware. Heutzutage enthält jedes Fahrzeug ein sicherheitskritisches Echtzeit-Kommunikationsnetzwerk, um seine Funktionen zu erfüllen. Vor allem die zunehmende Konnektivität der automobilen Systeme vergrößert die Angriffsfläche für Cyber-Attacken. Die Softwarezuverlässigkeit in diesem Bereich ist seit Jahrzehnten gut verstanden und erforscht, die Softwaresicherheit gegen Angriffe auf diese Systeme bedarf jedoch weiterer Forschung.

Diese Arbeit führt einen Black-Box-Untersuchungsprozess zur Analyse bestehender Automobilsysteme und -komponenten ein und identifiziert Sicherheitsschwachstellen in vier verschiedenen Steuergeräten. Kombiniert mit einer Übersicht über die veröffentlichte Sicherheitsforschung werden fahrzeuginterne Netzwerke als außergewöhnliche Bedrohung für die Sicherheit des Fahrzeugs identifiziert. Die herausragenden Automatisierungsmöglichkeiten von Sicherheitstests für diese Netzwerke werden im zweiten Teil dieser Arbeit genutzt.

Um automatisierte Werkzeuge für automobile Netzwerke zu erstellen, ist ein Softwareframework notwendig. Im Rahmen dieser Arbeit wurde ein umfassendes Open-Source-Software-Framework für Sicherheitstests in Fahrzeugnetzwerken entwickelt und veröffentlicht. Damit soll Sicherheitsforschung auf Basis von offener und freier Software unterstützt werden.

Es werden neuartige Werkzeuge zur automatisierten Identifikation und Exploration von Angriffsflächen in automobilen Diagnoseprotokoll-Implementierungen erstellt und evaluiert. Diese Werkzeuge ermöglichen die Erstellung von vergleichbaren Angriffsflächenmetriken durch Black-Box-Scans beliebiger Steuergeräte. Automatisches Lernen und Systemzustands-Reverse-Engineering-Techniken erhöhen die Explorationsfähigkeiten der vorgestellten Werkzeuge erheblich. Der Explorationsalgorithmus

wird auf dreizehn verschiedenen Steuergeräten von unabhängigen OEMs getestet. Alle gesammelten Ergebnisse werden im letzten Teil dieser Arbeit ausgewertet und diskutiert.

Abschließend werden offene Fragen und weitere Forschung auf Basis dieser Arbeit diskutiert.

Prohlašuji, že jsem tuto disertační práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

I hereby declare that this thesis has been written only by the undersigned and without any assistance from third parties.

Furthermore, I confirm that no sources have been used in the preparation of this thesis other than those indicated in the thesis itself.

In Pilsen on May 2021

Author's signature

Contents

List of Figures

List of Tables

Acronyms

ACU Airbag Control Unit. 40

APN Access Point Name. 38, 43, 45

ASIC Application-Specific Integrated Circuit. 33

AUTOSAR Automotive Open System Architecture. 13, 15

BCM Body Control Module. 78

BDC Body Domain Controller. 27, 28, 32, 33, 35, 78

BTS Base Transceiver Station. 43, 110, 111, 117, 118

CAL Calibration. 18

CAN Controller Area Network. xi, 5–8, 10–13, 15, 17, 28–31, 33–35, 37–39, 41–45, 50, 54, 59–61, 64, 67, 77, 80, 103, 105, 106, 111, 112, 114–116, 119–122, 124

CAN FD Controller Area Network Flexible Data-Rate. 12, 17

CCP CAN Calibration Protocol. 17, 123

CDF Cumulative Distribution Function. 74–76, 85

CGW Central Gateway Controller. 27–31, 44, 116

CMD Command Packet. 18

CPS Cyber-Physical System. 24, 55, 116–118, 120

CRC Cyclic Redundancy Check. 13

CTO Command Transfer Object. 17, 18

CVE Common Vulnerabilities and Exposures. 65

CVSS Common Vulnerability Scoring System. 23

DAQ Data Acquisition. 17, 18

DBC Data Base CAN. 121

DC Domain Controller. 8, 9

DoIP Diagnostic over IP. 15, 16, 61, 77, 123

DoS Denial of Service. 11, 38, 66, 67

DTO Data Transfer Object. 18

ECU Electronic Control Unit. 3, 5–11, 13, 15–17, 20–23, 25–45, 49, 50, 53–55, 61, 64–72, 74, 75, 77–83, 85, 86, 89–91, 105, 106, 108, 109, 111, 114, 116–120, 122, 123

EDGE Enhanced Data Rates for GSM Evolution. 117, 118

EEPROM Electrically Erasable Programmable Read-Only Memory. 22, 29, 30, 33, 34, 102, 105

ERR Error. 18

EV Event Packet. 18

FoD Feature on Demand. 9

GMLAN General Motor Local Area Network. 15, 16, 34, 61, 64, 65, 68–70, 79, 105, 123

GPS Global Positioning System. 36–38

GSM Global System for Mobile Communications. 36–38, 117, 118

GW Gateway. 8–10, 15

HSFZ High-Speed-Fahrzeug-Zugang (High-Speed Car Access). 28, 29, 61, 77, 122

I2C Inter-Integrated Circuit. 29

IC Integrated Circuit. 22, 29, 41, 42

IoT Internet of Things. 2

IP Internet Protocol. 15, 28, 29, 43, 61, 111

ISO-TP Transport Layer. 13–16, 28, 30, 57, 60, 61, 77

JTAG Joint Test Action Group. 29, 30, 33, 34, 37, 39, 48, 53, 107

LIN Local Interconnect Network. 5, 6, 29, 30, 33, 34

LTE Long Term Evolution. 37, 38

MCU Microcontroller Unit. 45, 48, 104

MEMS Micro-Electro-Mechanical System. 41

MMU Multimedia Unit. 28, 29, 31, 44, 45, 111–115, 117, 119

MOST Media Oriented Systems Transport. 5

NAD Network Access Device. 37, 38

NDA Non-Disclosure Agreement. 74, 89

NGTP Next Generation Telematics Protocol. 43, 44, 48, 110, 111, 117, 118

OBD On-Board Diagnostic. 28, 31, 34, 35, 62, 105, 106, 109, 123

OEM Original Equipment Manufacturer. 5–10, 15, 16, 49, 50, 64, 69, 71, 74–77, 81, 88, 89

OS Operating System. 77

OSI Open Systems Interconnection. 6

PC Personal Computer. 17

PCB Printed Circuit Board. 22, 23, 28, 29, 31, 33, 37, 41, 50, 52, 53, 102, 104, 105, 107, 108, 110

PGM Programming. 18

RAM Random Access Memory. 33, 37, 106, 107

RCE Remote Code Execution. 48, 109

REC Receive Error Counter. 12

RES Command Response Packet. 18

RFID Radio-Frequency Identification. 34

SBC System Basis Chip. 33

SERV Service Request Packet. 18

SIM Subscriber Identity Module. 37, 38

SOME/IP Scalable service-Oriented MiddlewarE over IP. 17

SPI Serial Peripheral Interface. 17, 29, 33, 41, 42, 112

STIM Stimulation. 18

SWCAN Single Wire Controller Area Network (CAN). 33, 34, 37, 39, 41, 108, 109

TARA Threat Assessment & Remediation Analysis. 24

TCP Transmission Control Protocol. 28, 29, 77, 122, 123

TCU Telematics Control Unit. 36–39, 43, 44, 47, 78, 110, 117

TEC Transmit Error Counter. 12

TOCTOU time-of-check to time-of-use. 44, 48, 67, 116, 117

UART Universal Asynchronous Receiver Transmitter. 33, 39, 53, 104

UDP User Datagram Protocol. 77

UDS Unified Diagnostic Service. 15, 16, 30, 44, 48, 61, 64, 65, 69, 70, 80, 103, 114, 116–118, 123

USB Universal Serial Bus. xi, 17, 37, 39, 108

USB-OTG Universal Serial Bus (USB) On-The-Go. 37, 108

VIN Vehicle Identification Number. 66

WLAN Wireless Local Area Network. 2, 37, 38, 45, 54, 113, 115, 118, 120

XCP Universal Measurement and Calibration Protocol. 17, 18, 61, 124

Acknowledgment

I like to thank everyone who has supported me along the way. My special thanks go to my supervisors, Professor Mottok and Professor Matoušek, as well as to my colleague Enrico Pozzobon, and last but not least, to my wife, Julia.

Part I

Introduction and Background

Chapter 1

Introduction

The increasing connectivity of our modern world exponentially creates safety-critical networks. Every connected computer or microcontroller participates in a communication network to deliver intelligent services. The security of a network is not inheritable, leading to the fact that two individual secure networks will not form a secure network after their connection. Every new connection of a system can be used as an attack surface. If at least one of the participants in a network is a safety-critical component, the combined network forms a not secure safety-critical network. This process happens countless times, every day, and at an ever-increasing rate. A smartphone connects to a car, a car connects to a Wireless Local Area Network (WLAN), a WLAN connects to a pacemaker, and industrial control systems get remotely maintained over the Internet. Even if a connection is not persistent, it does not mean that this connection can not be used as an attack surface to a safety-critical system.

Nowadays, formal proof of a system's security is only possible on minimal and limited systems. If we consider the complexity of the latest Internet of Things (IoT)-microcontrollers, this already rules out any possibility for formal security verification, inheriting this problem into any network such a microcontroller may get connected.

To conquer this increasing problem of our modern world, automated security testing of safety-critical networks becomes necessary. Semi- and fully-automated security testing of safety-critical networks can identify bugs, vulnerabilities, and exploits of a system. The focus of this thesis lies on automotive networks as safety-critical systems. A modern vehicle has various remote attack surfaces and safety-critical communication networks for its internal functions. Security vulnerabilities in this context can lead to devastating results. Even the next step in automotive technol-

ogy, autonomous vehicles, is only achievable if a specific safety and security level can be guaranteed. This thesis summarized different approaches and methods for semi-automated and fully-automated security testing of safety-critical vehicular networks.

1.1 Goals of the Thesis

- to analyze current security flaws and vulnerabilities in automotive systems,
- to evaluate existing open-source software projects regarding their suitability for security testing in safety-critical networks,
- to analyze automation capabilities for security testing in automotive systems,
- to devise software tools for semi- and fully-automated security testing in safety-critical networks,
- to find comparable metrics for the evaluation of possible attack surfaces in automotive components.

1.2 Thesis Outline

This thesis is structured as follows. Chapter 2 serves as an introduction to vehicular networks, containing relevant physical protocols (Section 2.1), general network topologies (Section 2.2), and related automotive communication protocols (Section 2.3).

Chapter 3 defines an investigation-process (Section 3.1), a vulnerability scoring system (Section 3.2), and a standardized description of vulnerabilities (Section 3.3) for black-box security investigations of automotive systems.

Chapter 4 contains the application of the previously defined investigation-process, as well as the scoring system to four different Electronic Control Units (ECUs) (Section 4.1), and a survey of published security research targeting automotive systems (Section 4.2).

Chapter 5 analyzes 28 identified vulnerabilities and eight vulnerability chains from the previous chapter. This analysis reveals joint attack surfaces of automotive components. Furthermore, the automation capabilities for security testing of each

identified attack surface are discussed.

Chapter 6 compares existing open-source tools for security testing (Section 6.1) and summarizes necessary contributions to obtain tools with automation capabilities for security testing of vehicular networks (Section 6.2).

Chapter 7 describes an attack surface metric for automotive diagnostic protocols (Section 7.1) and a novel security scanner with system state reverse-engineering capabilities (Section 7.2). This scanner is built upon the developed tool for security testing.

Chapter 8 discusses gathered results from conducted security scans of the devised automotive diagnostic protocol scanner with automated system state reverse engineering.

Finally, Chapter 9 concludes the thesis and discusses the achievements of the thesis.

Chapter 2

Vehicular Networks

Modern vehicles are more similar to a computer than to a mechanical device. Up to 100 different computers, called ECUs, connected by hundreds of cables (more than two kilometers of cable on average), shape a modern vehicle's internal network. Additionally to these wired connections, multiple ECUs support wireless connectivity, mainly for convenience and emergency functionalities.

This chapter introduces relevant protocols and existing network topologies of recent vehicles. Used protocols in a vehicle's network are introduced, and the relevance for security-evaluations is discussed briefly.

2.1 Physical Protocols

Protocols on the the Media Access Layer of the OSI model

More than 20 different communication protocols exist for the vehicle's internal wired communication. Most vehicles make use of five to ten different protocols for their internal communication. The decision which communication protocol is used from an Original Equipment Manufacturer (OEM) is usually made by the trade-off between the costs for communication technology, the final car price, and the desired features. The four major communication technologies for inter-ECU communication are CAN, FlexRay, Local Interconnect Network (LIN), and Automotive Ethernet. For security considerations, these are the most relevant protocols for wired communication in vehicles. Since Media Oriented Systems Transport (MOST) networks are only used to communicate multimedia payloads, MOST networks are not discussed in this thesis. Usually, safety-critical data is not communicated through MOST networks.

2.1.1 LIN

Connects ECUs with actuators and sensors

LIN is a single wire communication protocol for low data rates. Actuators and sensors of a vehicle exchange information with an ECU, acting as a LIN master. Software updates over LIN are possible, but the LIN slaves usually do not need software updates because of their limited functionality. This thesis excludes LIN as a protocol for security-relevant considerations. Attacks on LIN are possible [54], but the attack range is very limited. An attack propagation through LIN into a topologically higher network is unlikely and was not shown yet.

2.1.2 CAN

Interconnects ECUs

CAN is by far the most used communication technology for inter-ECU communication in vehicles. In older or cheaper vehicles, CAN is still the primary protocol for a vehicle's backbone communication. Safety-critical communication during a vehicle's operation, diagnostic information, and software updates are transferred between ECUs over CAN. The lack of security features in the protocol itself, combined with the general use, makes CAN the primary protocol for security investigations.

2.1.3 FlexRay

Interconnects safety-critical ECUs

The FlexRay consortium designed FlexRay as a successor of CAN. Modern vehicles have higher demands on communication bandwidth. By design, FlexRay is a fast and reliable communication protocol for inter-ECU communication. FlexRay components are more expensive than CAN components, leading to a more selective use by OEMs. The lack of open-source hardware and software for FlexRay communication rules out the consideration of FlexRay in this thesis.

2.1.4 Automotive Ethernet

Acts as vehicle backbone

Recent upper-class vehicles implement Automotive Ethernet, the new backbone technology for internal vehicle communication. The rapidly grown bandwidth demands already replace FlexRay [48]. The primary reasons for these demands are driver-assistant and autonomous-driving features. Only the physical layer (layer 1) of the Open Systems Interconnection (OSI) model distinguishes Ethernet (IEEE 802.3) from Automotive Ethernet (BroadR-Reach). This design decision leads to multiple advantages. For example, communication stacks of high-level operating systems can

be used without modification and routing, filtering, and firewall systems. Automotive Ethernet components are already cheaper than FlexRay components, which will lead to vehicle topologies, where CAN and Automotive Ethernet are the most used communication protocols.

2.2 Topologies

The used topology of a vehicle network depends mainly on the OEM and the vehicle's price category. OEMs implement vehicle topologies through construction kits for different vehicle categories, which means vehicle models from the same OEM with comparable selling prices have very similar, sometimes even identical, ECUs and network topologies.

Depending on the vehicle topology, the overall vehicle's attack-ability can be completely different [O5]. All participating ECUs must be analyzed regarding their functionalities to distinguish if a network or a sub-network is a safety-critical network. A vehicle's attack-ability can be rated lower if a network separation between safety-critical ECUs and ECUs with remote attack surfaces exists [37]. The same applies vice versa.

2.2.1 Line-Bus

Low security level, used in cheaper or older cars

The first vehicles with CAN bus used a single network with a line-bus topology, as illustrated from figure 2.1. Some lower-priced vehicles still use one or two shared CAN bus networks for their internal communication nowadays. The downside of this topology is its vulnerability and the lack of network separation. All ECUs of a vehicle are connected on a shared bus. Since CAN does not support security features from its protocol definition, any participant on this bus can communicate directly with all other participants, which allows an attacker to affect all ECUs, even safety-critical ones, by compromising one single ECU. The overall security level of this network is given from the security level of the weakest participant. The famous attack from Miller and Valasek was possible because this topology was used in the car they attacked [38]. Attackers can escalate an attack of a vulnerable ECU over the network to interfere directly with safety-critical ECUs.

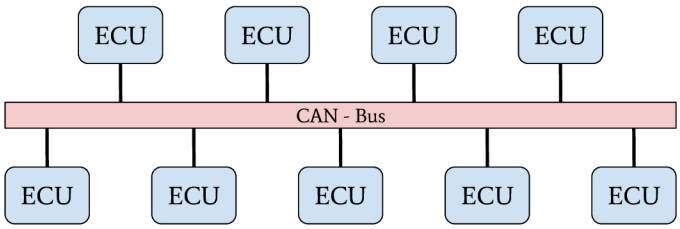

Figure 2.1: Line-Bus network topology

2.2.2 Central Gateway

Medium security level, standard in most current cars

The central Gateway (GW) topology can be found in higher-priced older cars and medium- to lower-priced recent cars. A centralized GW ECU separates domain-specific sub-networks, as shown in figure 2.2. This allows an OEM to encapsulate all ECUs with remote attack surfaces in one sub-network. ECUs with safety-critical functionalities are located in an individual CAN network. Next to CAN, FlexRay might also be used as a communication protocol inside a separate network domain. The security of a safety-critical network in this topology depends mainly on the central GW ECU's security. This architecture increases the overall security level of a vehicle through domain separation. After an attacker successfully exploited an ECU through an arbitrary attack surface, a second exploitable vulnerability or a logical bug is necessary to compromise a different domain, a safety-critical network, inside a vehicle. This second exploit or logical bug is necessary to overcome the network separation of the central GW ECU.

2.2.3 Central Gateway and Domain Controller

High security level, used in latest higher-priced cars

A new topology with central GW and Domain Controllers (DCs) can be found in the latest higher-priced vehicles. The general structure of this topology is shown in figure 2.3. The increasing demand for bandwidth in modern vehicles with autonomous driving and driver assistant features led to this topology. An Automotive Ethernet network is used as a communication backbone for the entire vehicle. Individual domains, connected through a DC with the central GW, form the vehicle's backbone. The individual DCs can control and regulate the data communication between a domain and the vehicle's backbone. This topology achieves a very-high

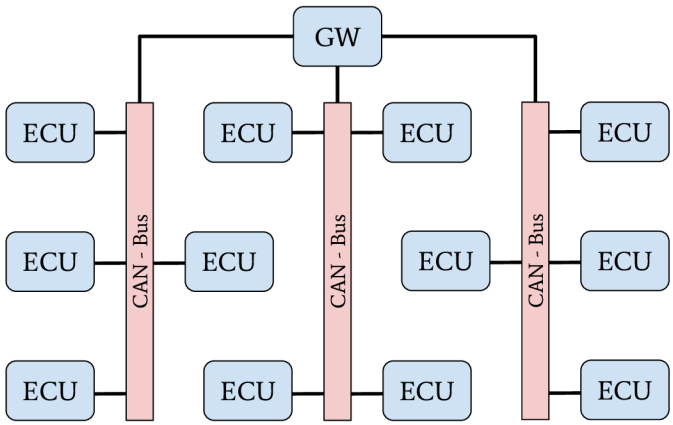

Figure 2.2: Network topology with central GW ECU

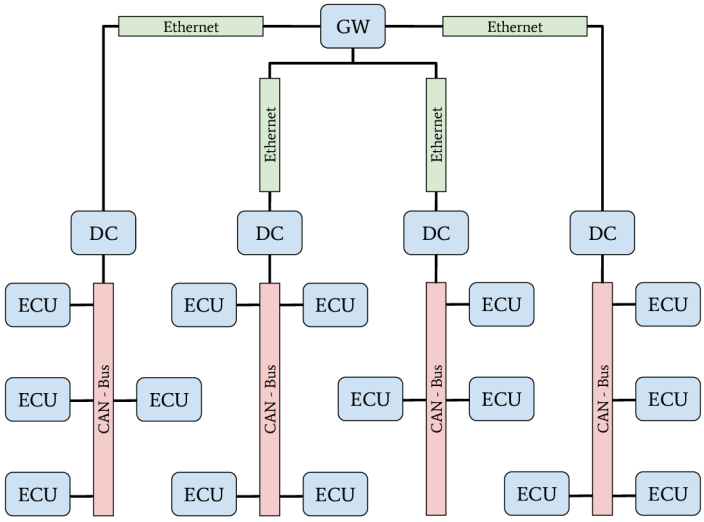

Figure 2.3: Network topology with Automotive-Ethernet backbone and DC

security level through a strong network separation with individual DCs, acting as gateway and firewall, to the vehicle's backbone network. OEMs have the advantage of dynamic information routing next to this security improvement, an enabler for Feature on Demand (FoD) services.

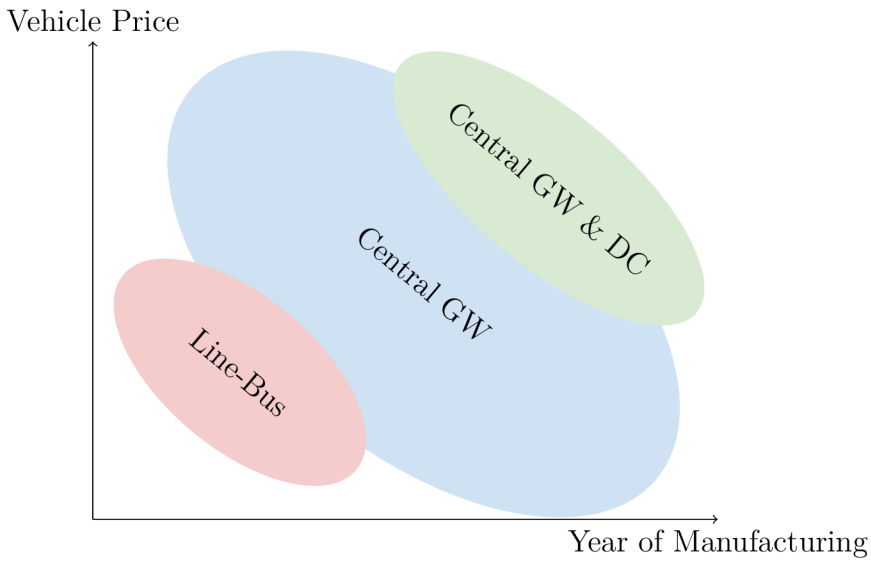

Figure 2.4: Overview of used topology, vehicle price, and year of manufacturing

2.2.4 Summary

The topology of a vehicle's network has a very high impact on a vehicle's security and safety properties. Figure 2.4 shows the dependencies between a vehicle's price, a vehicle's age, and the used network topology. Only a few vehicles from the lower price segment or higher age are still using one or two CANs for their internal communication. Most vehicles are equipped with a central GW ECU. After the publication of Miller and Valasek, even the cheapest cars received an upgrade of their internal network topology, and OEMs made use of central GW ECUs as mitigation for cyber-attacks [38]. Only a few vehicles from the higher price segment already implement a network architecture with Automotive Ethernet as a communication backbone. Nevertheless, this group will rapidly grow soon since the advantages will compensate for the higher prices for the communication equipment.

2.3 Automotive Communication Protocols

This section provides an overview of relevant communication protocols for security evaluations in automotive networks. In contrast to section 2.1, this section focuses on properties for data communication. Figure 2.5 provides an overview of the network layer a protocol serves its function.

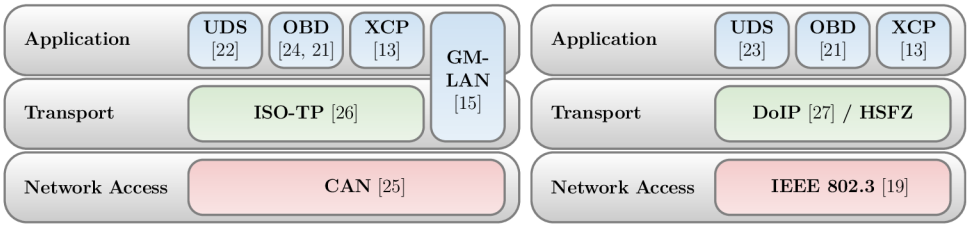

Figure 2.5: Automotive diagnostic protocol stack for CAN (left) and IEEE 802.3 (right) based networks. This figure provides an overview of relevant protocols and their location in the automotive diagnostic protocol stack.

2.3.1 CAN

Error handling can be abused for DoS attacks

The CAN communication technology was invented in 1983 as a message-based robust vehicle bus communication system. The Robert Bosch GmbH designed multiple communication features into the CAN standard to achieve a robust and computation efficient protocol for controller area networks. Remarkable for the communication behavior of CAN is the internal state machine for transmission errors. This state machine implements a fail silent behavior to protect a safety-critical network from babbling idiot nodes. If a specific limit of reception errors (REC) or transmission errors (TEC) occurred, the CAN driver changes its state from error-active to error-passive and finally to bus-off.

In recent years, this protocol specification was abused for Denial of Service (DoS) attacks and information gathering attacks on the CAN network of a vehicle. Cho et al. demonstrated a DoS attack against CAN networks by abusing the bus-off state of ECUs [8]. Injections of communication errors in CAN frames of one specific node caused a high transmission error count in the node under attack, forcing the attacked node to enter the bus-off state, seen in figure 2.6. In 2019 Kulandaivel et al. combined this attack with statistical analysis to achieve a fast and inexpensive network mapping in vehicular networks [30]. They combined statistical analysis of the CAN network traffic before and after the bus-off attack was applied to a node. All missing CAN frames in the network traffic after an ECU was attacked could now be mapped to the ECU under attack, helping researchers identify the origin ECU of a CAN frame. Ken Tindell published a comprehensive summary of low level attacks on CANs in 2019 [56].

Figure 2.7 shows a CAN frame and its fields as it is transferred over the network.

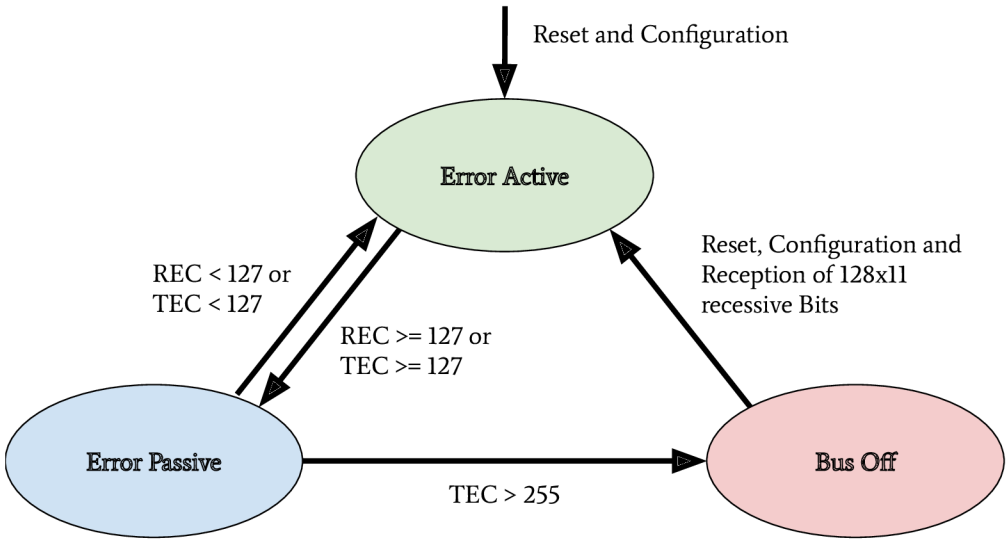

Figure 2.6: CAN bus states on transmission errors. Receive Error Counter (REC), Transmit Error Counter (TEC)

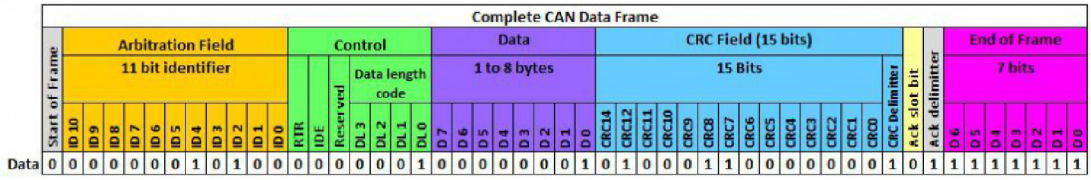

Figure 2.7: Complete CAN data frame structure [33]

For information exchange, only the fields arbitration, control, and data are relevant. These are the only fields to which a usual application software has access. All other fields are evaluated on a hardware-layer and, in most cases, are not forwarded to an application. The data field has a variable length and can hold up to 8 bytes. The length of the data field is specified by the data length code inside the control field. Important variations of this example are CAN-frames with extended arbitration fields and the Controller Area Network Flexible Data-Rate (CAN FD) protocol. On Linux, every received CAN frame is passed to SocketCAN. SocketCAN allows the CAN handling via network sockets of the operating system. SocketCAN was created by Oliver Hartkopp and added to the Linux Kernel version 2.6.25 [17]. Figure 2.8 shows the frame structure, how CAN frames are encoded if a user-land application receives data from a CAN socket.

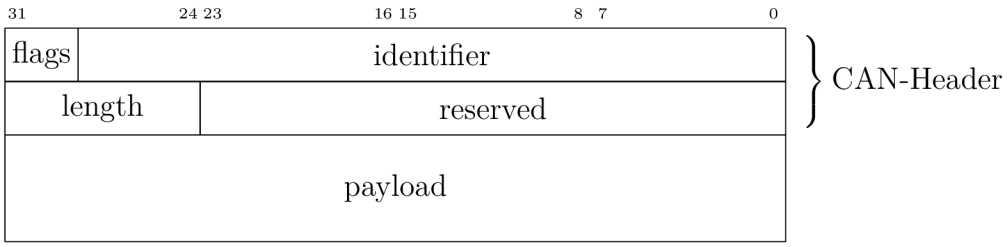

Figure 2.8: CAN frame defined by SocketCAN [1]

Figure 2.7 and figure 2.8 are using different field names which can be mapped as follows: *Arbitration Field* maps to *identifier*, *Control* to *flags* and *length*, *Data* to *payload*. The comparison of figure 2.7 and figure 2.8 clearly shows the loss of information during the CAN frame processing from a physical layer driver. Almost every CAN driver acts in the same way, whether an application code runs on a microcontroller or a Linux kernel. This also means that a standard application does not have access to the Cyclic Redundancy Check (CRC) field, the acknowledgment bit, or the end-of-frame field.

Through the CAN communication in a vehicle or a separated domain, ECUs exchange sensor-data and control inputs; this data is mainly not secured and can be modified by assailants. Attackers can easily spoof sensor values on a CAN bus to trigger malicious reactions of other ECUs. Miller and Valasek described this spoofing attack during their studies on automotive networks [36]. To prevent attacks on safety-critical data transferred over CAN, Automotive Open System Architecture (AUTOSAR) released a secure onboard communication specification [4].

2.3.2 ISO-TP (ISO 15765-2)

OSI Transport Layer protocol for addressed communication in CANs

The CAN protocol supports only eight bytes of data. Use-cases like diagnostic operations or ECU programming require much higher payloads than the CAN protocol supports. For these purposes, the automotive industry standardized the Transport Layer (ISO-TP) (ISO 15765-2) protocol [26]. ISO-TP is a transportation layer protocol on top of CAN. Payloads with up to 4095 bytes can be transferred between ISO-TP endpoints fragmented in CAN frames. The ISO-TP protocol handling requires four special frame types.

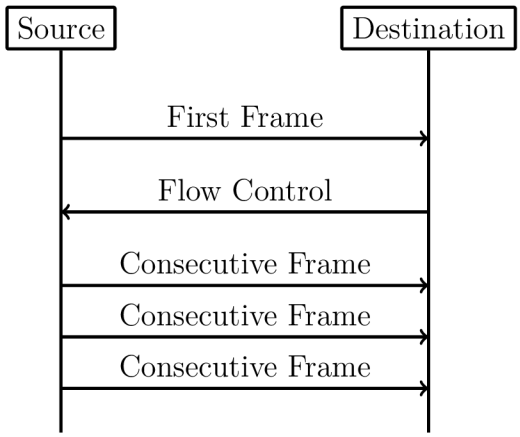

Figure 2.9: ISO-TP fragmented communication

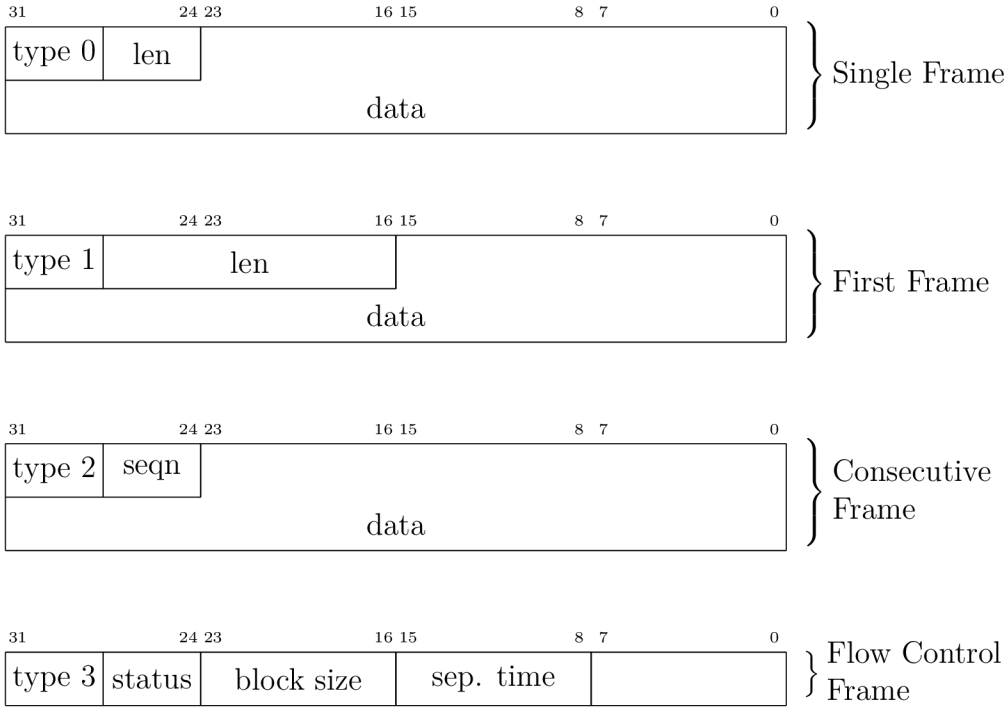

Figure 2.10: ISO-TP frame types

The different types of ISO-TP frames are shown in figure 2.10. The payload of a CAN frame, shown in figure 2.8, gets replaced by one of the four ISO-TP frames from figure 2.10. Each individual ISO-TP frame type has a different purpose. A single frame can transfer between 1 and 7 bytes of ISO-TP message data. The `len` field of a Single Frame or a First Frame indicates the ISO-TP message length. Every message with more than 7 bytes of payload data must be fragmented into a First Frame, followed by multiple Consecutive Frames. This communication is illustrated in figure 2.9. After the First Frame is sent from a sender, the receiver has to communicate its reception capabilities through a Flow Control Frame to the sender. Only after this Flow Control Frame is received, the sender is allowed to communicate the Consecutive Frames according to the receiver's capabilities.

ISO-TP acts as a transport protocol with the support of directed communication through addressing mechanisms. In vehicles, ISO-TP is mainly used as a transport protocol for diagnostic communication. In rare cases, ISO-TP is also used to exchange larger data between ECUs of a vehicle. Security measures have to be applied to the application layer protocol transported through ISO-TP since ISO-TP has no capabilities to secure its transported data.

2.3.3 DoIP

OSI Transport Layer protocol for diagnostic communication over IP-based protocols

Diagnostic over IP (DoIP) was first implemented on automotive networks with a centralized gateway topology. A centralized GW functions as a DoIP endpoint that routes diagnostic messages to the desired network, allowing manufacturers to program or diagnose multiple ECUs in parallel. Since the Internet Protocol (IP) communication between a repair-shop tester and the GW is many times faster than the communication between the GW ECU and a target ECU connected over CAN, the remaining bandwidth of the IP communication can be used to start further DoIP connections to other ECUs in different CAN domains. DoIP is specified as part of AUTOSAR and in ISO 13400-2. Similar to ISO-TP, DoIP does not specify special security measures. The responsibility regarding secured communication is delegated to the application layer protocol.

2.3.4 Diagnostic Protocols

OSI Application Layer protocols

Two examples of diagnostic protocols are General Motor Local Area Network (GMLAN) and Unified Diagnostic Service (UDS) (ISO 14229-2). The General Motors Cooperation uses GMLAN. German OEMs mainly use UDS. Both protocols are very

Table 129 — Request message definition

A_Data byte	Parameter name	Cvt	Hex value	Mnemonic
#1	ReadDataByIdentifier Request Service Id	M	22	RDBI
#2 #3	dataIdentifier[] #1 = [byte#1 (MSB) byte#2]	M M	00-FF 00-FF	DID_ HB LB
:	:	:	:	:
#n-1 #n	dataIdentifier[] #m = [byte#1 (MSB) byte#2]	U U	00-FF 00-FF	DID_ HB LB

Figure 2.11: UDS ReadDataByIdentifier service definition [22]

Table 80: ReadDataByParameterIdentifier Request Message

Data Byte	Parameter Name [Note 1]	Cvt	Hex Value	Mnemonic
#1	ReadDataByParameterIdentifier Request Service Id	M	22	SIDRQ
#2 #3	parameterIdentifier #1 = [byte 1 (MSB) byte 2 (LSB)]	M	00 thru FF 00 thru FF	PID_ B1 B2
#4 #5	parameterIdentifier #2 = [byte 1 (MSB) byte 2 (LSB)]	U	00 thru FF 00 thru FF	PID_ B1 B2
:	:	:	:	:
#n-1 #n	parameterIdentifier #k = [byte 1 (MSB) byte 2 (LSB)]	U	00 thru FF 00 thru FF	PID_ B1 B2

Note 1: MSB = Most Significant Byte, LSB = Least Significant Byte.

Figure 2.12: GMLAN ReadDataByParameterIdentifier service definition [15]

similar from a specification point of view, and both protocols use either ISO-TP or DoIP messages for a directed communication with a target ECU. Since different OEMs use UDS, every manufacturer adds its custom additions to the standard. Also, every manufacturer uses individual ISO-TP addressing for the directed communication with an ECU. GMLAN includes more precise definitions about ECU addressing and an ECUs internal behavior compared to UDS.

UDS and GMLAN follow a tree-like message structure, where the first byte identifies the service. Every service is answered by a response. Two types of responses are defined in the standard. Negative responses are indicated through the service `0x7F`. Positive responses are identified by the request service identifier incremented with `0x40`. Figure 2.11 and figure 2.12 are given as an example of how similar both diagnostic protocols are. The service identifier and the further specification of these request messages are identical.

2.3.5 SOME/IP

OSI Application Layer protocols

Replacement for CAN in Ethernet-based networks

Scalable service-Oriented MiddlewarE over IP (SOME/IP) defines a new philosophy of data communication in automotive networks. As described in section 2.2.3, SOME/IP is used to exchange data between network domain controllers in the latest vehicle networks. SOME/IP supports subscription and notification mechanisms, allowing domain controllers to dynamically subscribe to data provided by another domain controller dependent on the vehicle's state. SOME/IP transports data between domain controllers and the gateway that a vehicle needs during its regular operation. The use-cases of SOME/IP are similar to the use-cases of CAN communication. The main purpose is the information exchange of sensor and actuator data between ECUs. This usage emphasizes SOME/IP communication as a rewarding target for cyber-attacks.

2.3.6 CCP/XCP

OSI Application Layer protocol

Universal Measurement and Calibration Protocol (XCP), the CAN Calibration Protocol (CCP) successor, is a calibration protocol for automotive systems, standardized by ASAM e.V. in 2003. The primary usage of XCP is during the testing and calibration phase of ECU or vehicle development. CCP is designed for use on CAN. No message in CCP exceeds the 8-byte limitation of CAN. To overcome this restriction, XCP was designed to aim for compatibility with a wide range of transport protocols. XCP can be used on top of CAN, CAN FD, Serial Peripheral Interface (SPI), Ethernet, USB, and FlexRay. The features of CCP and XCP are very similar; however, XCP has a larger functional scope and optimizations for data efficiency.

Protocols for ECU development and configuration

Both protocols have a session-based communication procedure and support authentication through seed and key mechanisms between a master and multiple slave nodes. A master node is typically an engineering Personal Computer (PC). In vehicles, slave nodes are ECUs for configuration. XCP also supports simulation. A vehicle engineer can debug a MATLAB Simulink model through XCP. In this case, the simulated model acts as the XCP slave node. CCP and XCP can read and write to the memory of an ECU. Another main feature is data acquisition. Both protocols support a procedure that allows an engineer to configure a so-called data acquisition list with memory addresses of interest. All memory specified in such a list will be read periodically and be broadcast in a CCP or XCP Data Acquisition (DAQ) packet on the chosen communication channel. Figure 2.13 gives an overview of all supported communication and packet types in XCP. In the Command Transfer Ob-

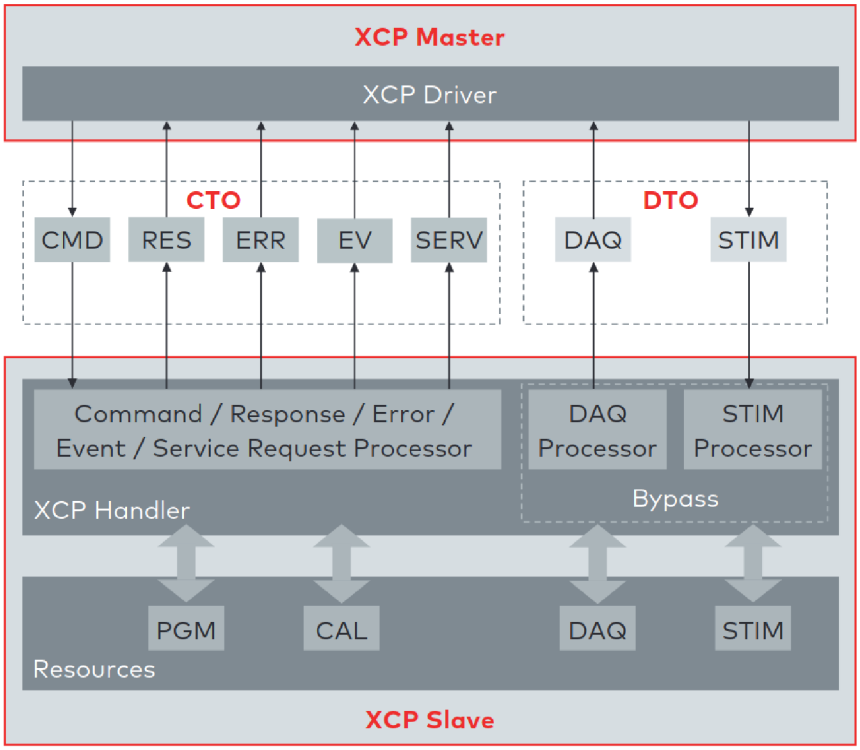

Figure 2.13: XCP communication model between XCP Master and XCP Slave. This model shows the communication direction for CTO/Data Transfer Object (DTO) packages [14].

ject (CTO) area, all communication follows a request and response procedure always initiated by the XCP master. A Command Packet (CMD) can receive a Command Response Packet (RES), an Error (ERR) packet, an Event Packet (EV), or a Service Request Packet (SERV) as a response. After the configuration of a slave through CTO CMDs, a slave can listen for Stimulation (STIM) packets and periodically send configured DAQ packets. The resources section of figure 2.13 indicates the possible attack surfaces of this protocol (Programming (PGM), Calibration (CAL), DAQ, STIM) which an attacker could abuse. It is crucial for a vehicle's security and safety that such protocols, which have their use only during calibration and development of a vehicle, are disabled or removed before a vehicle is shipped to a customer.

Part II

Manual Security Investigations of Safety-Critical Systems

Chapter 3

Investigation-Process and Vulnerability Metrics for Automotive Systems

Without a defined process, a security investigation of a target system is neither comprehensive nor comparable. This chapter introduces a black-box investigation process for embedded safety-critical systems customized for automotive components. Following manual investigations will be based on this investigation process. Preparatory to this general investigation process of automotive components, a model for threat analysis and risk estimation of ransomware for automotive systems was developed and published [O5]. In this publication, a theoretical risk model was verified through a showcase implementation of automotive ransomware.

3.1 Investigation Process

Identifies vulnerabilities in automotive systems

The goal of later investigations, and therefore the goal of this process, is the comprehensive identification of vulnerabilities in automotive components. The necessary steps of this process are defined on an abstracted automotive system, which guarantees the applicability to any safety-critical automotive system. Figure 3.1 shows an abstracted automotive system and all relationships of its hardware- and software-components to generic properties, later called ECU properties. A high-level (system-level) overview is shown on the left side of figure 3.1, which treats the investigated component as a black-box, and only considers interfaces from a vehicle network or a wireless network to the investigation target. Furthermore, an inspection of an ECU's

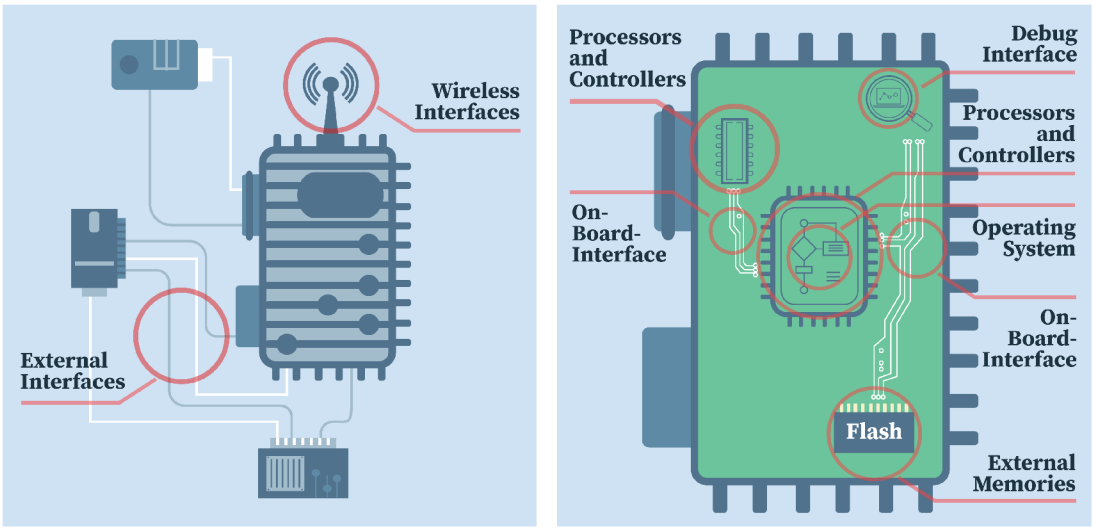

Figure 3.1: **Left:** System-level overview of an automotive component and the indication of ECU properties inside the investigation process. **Right:** Component-level overview of an automotive component and the indication of ECU properties inside the investigation process.

properties on a component-level is necessary to perform a comprehensive analysis. An abstracted ECU with its component-level properties is illustrated on the right side of figure 3.1. These generic ECU properties are a key element for a subsequent attack surface and vulnerability analysis since, through their broad definition, they can be applied to any component. An overview of the entire process is given in figure 3.2.

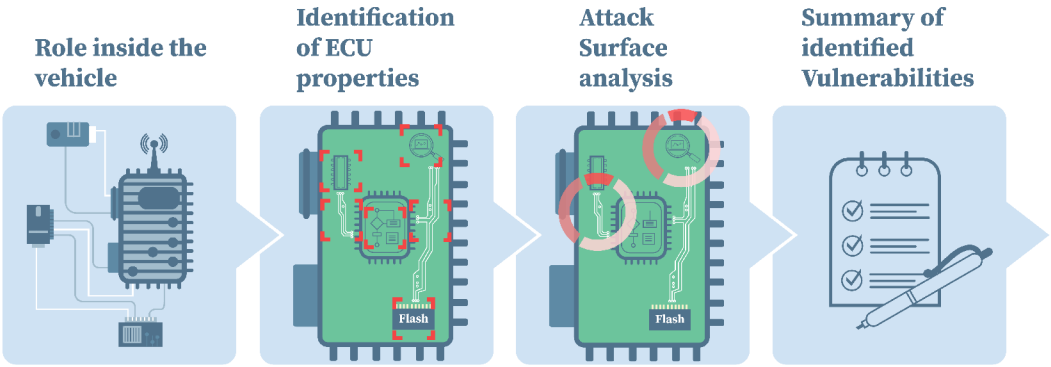

Figure 3.2: Overview of the investigation process for automotive components.

The process contains four steps. It starts with a high-level perspective on the entire investigation component and their duties in a complex safety-critical network. Next, all generic ECU properties are mapped to actual hard- and software components and interfaces, which results in a comprehensive list of all security-relevant entities. In the following step, each entity's possible attack surfaces are analyzed, and finally, vulnerabilities of each component can be identified. Each investigation of a unique ECU will be performed according to this process which guarantees comparable and comprehensive results. Table 3.1 contains a detailed description of the individual process steps.

Table 3.1: Definition of the investigation process for automotive components.

Step 1: Role inside the vehicle
Initially, it is necessary to understand the capabilities and responsibilities of a component inside the vehicle network. This analysis helps to estimate the possible impacts of vulnerabilities on the overall safety-critical system. In step four of this process, the impact of identified vulnerabilities can be estimated, taking safety- and security-critical side effects onto the entire vehicle into account.
Step 2: Identification of ECU properties
All technical information of the component is gathered, which guarantees the completeness of the attack surface analysis's in step three.
Processors and Controllers
All active components on an ECU.
External Memories
All memories, which are connected through an On-Board Interface to a Processor or Controller. Usually, these memories are external Electrically Erasable Programmable Read-Only Memory (EEPROM) Integrated Circuits (ICs) or Flash memory ICs, which may store security-relevant data.
Debug Interfaces
These interfaces give access to special debug functionalities and may leak sensitive information. Such interfaces are usually only used during the development process, and their functionality should be removed in a release-version of an ECU.
On-Board Interfaces
Communication interfaces between processors, controllers, or memories. These interfaces are necessary for information exchange or control functionalities between active components on a Printed Circuit Board (PCB).

External Interfaces
Wired connections for communication with other ECUs or diagnostic systems. These interfaces can be accessed over a network. No physical access to the PCB of an ECU is required.

Wireless Interfaces
Interfaces for communication with wireless networks. These interfaces are accessible without physical access to the PCB of an ECU.

Operating System
Used software architecture and operating system structure of an ECU, including all individual software and firmware components, such as application software, bootloader software, and even device drivers for special hardware.

Step 3: Attack Surface analysis
Based on the collection of ECU properties, possible attack surfaces are analyzed.

Remote attack surfaces
Remote attack surfaces can be exploited over a remote connection, wired or wireless. No physical access to the component itself is required.

Local attack surfaces
Local attack surfaces require physical access to the component. Modifications may need to be applied to exploit a local attack surface.

Step 4: Summary of identified Vulnerabilities
A fact sheet describes every identified vulnerability. Such a list of vulnerabilities for an ECU does not necessarily contain all possible vulnerabilities. Every investigation is performed within a limited time frame. Therefore only vulnerabilities found during the available investigation time will be listed. Finally, the impact of every vulnerability is evaluated with respect to the components role inside the vehicle.

3.2 Vulnerability Scoring System

Rates vulnerabilities, allows comparisons

Security-metrics are a challenging research field on their own [28]. A simple and satisfying solution for all aspects of a system is not easily achievable, which should be considered for exotic computer systems and networks found in the automotive domain. To further analyze identified vulnerabilities, a definition of the exploitation risk will be based on the Common Vulnerability Scoring System (CVSS), the *de-facto* standard for software vulnerabilities [35]. In recent years, automotive devel-

opers started to use the Threat Assessment & Remediation Analysis (TARA) system for risk assessments during their development processes. Since the conducted investigations during this research were performed as black-box analyses of automotive components, proprietary system knowledge required to perform a TARA was not available. The exploitation risk of a vulnerability is dependent on the exploitability and the impact factor. A general description of these factors is provided by Younis et al. [58]. The impact factor describes the possible consequences of an existing vulnerability, and the exploitability factor describes the accessibility of a vulnerability to an attacker. For the application to automotive systems, the impact factor is defined in table 3.2, the exploitability factor in table 3.3. The multiplication of both factors can obtain a score for the exploitation risk.

Table 3.2: Definition of a suitable impact score for a black-box analysis of automotive components, allowing the application of a basic vulnerability scoring system during later investigations.

Score	**Definition**
(1) Low	An attacker gains internal information to prepare for further attacks.
(2) Medium	An attacker gains privileges for local code execution or gains access to one further possible target or attack surface.
(3) High	An attacker gains privileges for local code execution and access to multiple possible targets or control over Cyber-Physical Systems (CPSs).

Table 3.3: Definition of a suitable exploitability score for a black-box analysis of automotive components, allowing the application of a basic vulnerability scoring system during later investigations.

Score	**Definition**
(1) Low	An attacker needs physical access to the target.
(2) Medium	An attacker needs physical access to a network connected to a target or wireless access and user interaction.
(3) High	An attacker can exploit this vulnerability through wireless access and without user interaction.

3.3 Vulnerability Description

Defined documentation of vulnerabilities

As a final step of the introduced investigation process, every identified vulnerability should be described in a standardized way. A template fact sheet for identified vulnerabilities is shown in table 3.4. This fact sheet shows the minimum of required information to describe the effects and the location of a vulnerability. For every vulnerability, a description explains technical details. **Preconditions** necessary for exploitation of the vulnerability help to understand **Impact** and **Exploitability** ratings. If these two ratings do not follow the standard definitions or require additional considerations, the **Rating Explanation** contains further information and explains exceptions from the definitions. Essential for later analysis is an attribution of the vulnerability to an **Attack Surface**. The fact sheet for vulnerability chains will be identical to the fact sheet of single vulnerabilities, except for one additional field. Since vulnerability chains consist of multiple vulnerabilities, the additional field **Involved Vulnerabilities** contains a list of all related vulnerabilities of a chain.

Table 3.4: Template for a vulnerability fact sheet

V0	**Vulnerability Name**		
Vulnerability description containing further details.			
Preconditions:	Preconditions, required to abuse this vulnerability.		
Impact:	(2) Medium	**Exploitability:**	(2) Medium
Rating Explanation:	Further information on why certain ratings for the impact and exploitability factors were chosen.		
Attack Surface:	The attack surface on which the vulnerability can be abused. This surface is based on one ECU characteristic out of the list of ECU properties.		

Chapter 4

Security Investigation and Survey of Safety-Critical Systems

This chapter examines safety-critical components in automotive systems and their possible attack surfaces. Manual investigations were performed to identify individual attack surfaces and exemplary vulnerabilities in four different ECUs, each with a unique role in an automotive network. Additionally, publicly available information on proven attacks against automotive systems was collected and analyzed to extract further vulnerabilities and vulnerability chains. During this information gathering step, the most relevant attack surfaces in automotive systems are discussed with real-world examples. Technical insights show the implications of certain attack surfaces on the security of the entire vehicle. All identified vulnerabilities are rated with the previously defined vulnerability scoring system.

4.1 Manual Investigations of Electronic Control Units

This section provides an overview of manually investigated components inside vehicle networks and shows investigation results for each component. Four different ECU types were chosen for manual analysis to demonstrate the variety of automotive systems. Every ECU type represents an ECU category with a unique role inside a vehicle's network. The performed analyses are based on the proposed investigation process from section 3.1; ratings of vulnerabilities follow the definitions in section 3.2. All manual investigations cover an ECU's embedded-security analysis, identify possible attack surfaces, and an exemplary list of identified vulnerabilities, documented

according to the definitions in section 3.3.

4.1.1 Central Gateway Controller

Bundles the entire communication of a vehicle

As a representative for gateway ECUs in vehicular networks, a gateway ECU for BMW cars was investigated. This ECU's main application is in lower volume BMW models, for example, the i3, X1, and Mini cars. Independently from this investigation, KeenLabs also published an analysis of this ECU [32].

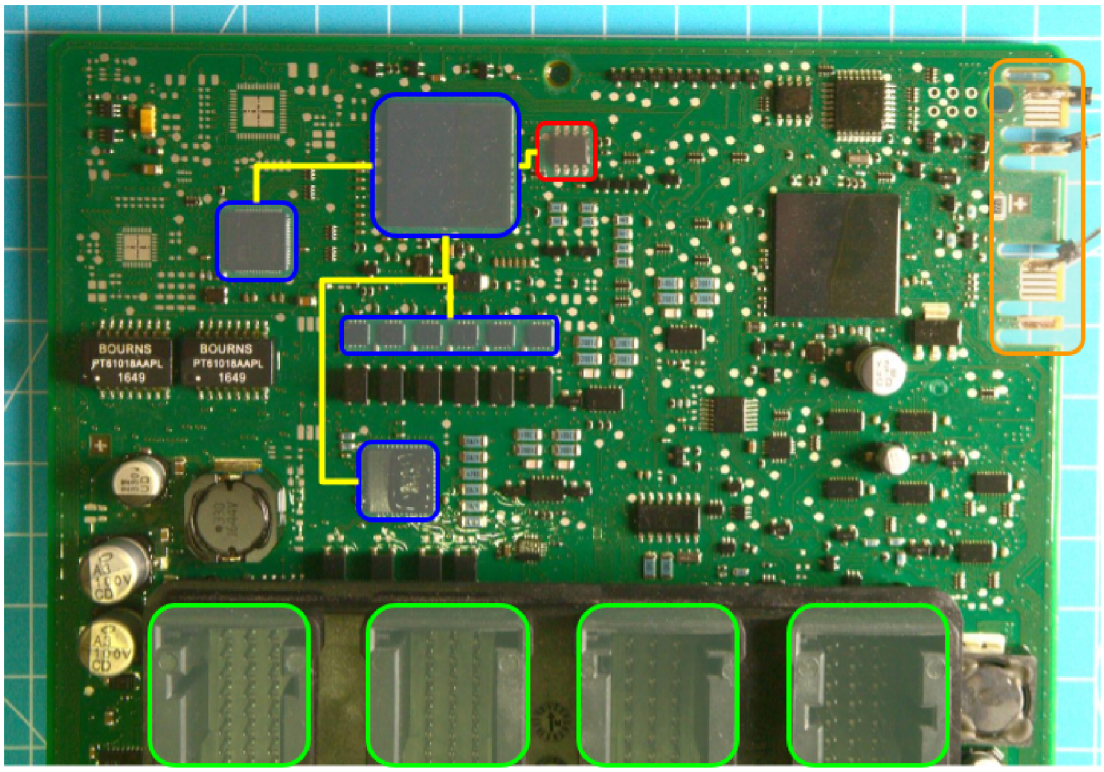

Figure 4.1: PCB of the Body Domain Controller (BDC) and Central Gateway Controller (CGW) ECU, model LR01. Only the components of the CGW are highlighted. The following color scheme is used for the indication of ECU properties. *Processors and Controllers* are blue, *External memories* are red, *Debug Interfaces* are orange, *On-Board Interfaces* are yellow, and *External Interfaces* are green.

Table 4.1: Investigation of the BMW CGW, following the defined process.

Step 1: Role inside the vehicle

This ECU fulfills two different functionalities inside the vehicle. The PCB contains two processors, which act as two independent ECUs in the same housing. One processor is used as CGW; the other processor acts as BDC. The BDC part of this ECU is in charge of controlling all body electronics components inside the car, for example, the lighting system, the horn, the wash and wipe system, and power window regulators. The following security analysis focuses on the functionalities of the CGW part of this ECU.

The main purpose of this controller is the network separation and message routing between individual networks. All communications from one subnetwork to another subnetwork are routed through the CGW. Furthermore, the CGW can act as a firewall to block illegitimate communication between networks. Through a dedicated diagnostic network connected to the On-Board Diagnostic (OBD) interface, the CGW provides diagnostic access from OBD to all subnetworks, dependent on the target ECU location for diagnostic communication. As a gateway control unit, this ECU has access to all communication on the vehicle's internal networks. It fulfills safety-critical functionalities through its control over network communication and acts as a FlexRay master node. Two IEEE 802.3 Ethernet interfaces are connected to an Ethernet-Switch controller. One Ethernet line is connected to the vehicle's OBD connector; the other line is connected to the vehicle's Multimedia Unit (MMU). The CGW can route IP traffic from the OBD connector to the MMU, which is useful for large software updates of the vehicle's multimedia system, for example, navigation system data. Furthermore, the CGW can route ISO-TP messages from any subnetwork of the vehicle to the OBD connector. ISO-TP messages get encapsulated in the BMW proprietary protocol High-Speed-Fahrzeug-Zugang (High-Speed Car Access) (HSFZ) and sent as a Transmission Control Protocol (TCP) packet to a repair shop tester.

Step 2: Identification of ECU properties

The following properties could be identified.

Processors and Controllers

This ECU contains one FlexRay controller, one Ethernet-Switch controller, and seven CAN controllers. The central processor is an NXP MPC5668G controller with a Power Architecture e200z6 core. One Atmel microcontroller is used for the passive entry system of the vehicle.

External Memories

Per processor, one external EEPROM could be identified. These memories store persistent data such as mileage values.

Debug Interfaces

The PCB contains two debug headers, one header for each processor. These debug headers expose a serial connection with a command-line interface and a Joint Test Action Group (JTAG) interface.

On-Board Interfaces

One CAN interface used for the on-board communication between the two processors. SPI and Inter-Integrated Circuit (I2C) are used for communication with various peripheral ICs. The communication on these interfaces is mainly for safety-related information, like status information of output drivers and relays-switches.

External Interfaces

Four individual FlexRay networks, six CANs, and two Ethernet interfaces are connected to this ECU. One Ethernet and one CAN interface is dedicated to diagnostic communication. The second Ethernet interface is directly connected with the MMU of the vehicle. FlexRay communication is mainly used for safety-critical data transfer, such as driver assistant systems and steering control units. Also, the CGW acts as a master node for 14 different LIN networks.

Wireless Interfaces

This ECU contains interfaces for multiple passive-entry antennas.

Operating System

Inside the ECU, an embedded operating system and a bootloader are used.

Step 3: Attack surface analysis

The following attack surfaces can be considered for further security analysis.

Remote attack surfaces

Ethernet TCP and IP-based communications have a complex driver stack. The BMW-specific diagnostic protocols can offer privileged functionalities. Due to the higher complexity of the used protocols, this is the likeliest interface for implementation errors in network drivers. Software updates of the entire vehicle are forwarded from HSFZ to the desired ECU in a subnetwork.

CAN This attack surface is the second likeliest for remote attacks. Most ECUs in the car are connected via CAN to the CGW. To offer the required diagnostic functionalities, this ECU can forward ISO-TP communication into subnetworks. UDS is implemented to allow diagnostic operations and software updates on this ECU.

LIN Various peripheral components are connected over LIN to this ECU. LIN does not support complex protocols on this ECU. Attacks over LIN that could compromise this ECU are unlikely.

Local attack surfaces

JTAG JTAG access is not possible without further hardware attacks. JTAG would be a rewarding attack surface to compromise this ECU and extract sensitive data, such as keys of the vehicles' immobilizer system or secrets for encrypted communication between ECUs.

Debug Interfaces This ECU has two serial debug interfaces, one per processor. Usually, these interfaces are used during development. Logging functionalities can help during the information gathering to better understand the internal behavior of the ECU. Vulnerabilities on the debug interface of this processor can be used to compromise the entire ECU.

External Memories Data that is often manipulated during the ECUs' lifetime is stored in external EEPROMs. Stored data in these memories can be manipulated if the ECU does not encrypt or authenticate this memory. This attack surface could be used to spoof incorrect data for internal processing.

Chip-Internal Memories Very sensitive data, for example, the cryptographic material for the vehicles' immobilizer system, is stored in internal memories of the processor. Advanced hardware attacks are required to attack the processor's internal storage. Colin O'Flynn's publication gives a detailed impression of the required effort for such hardware attacks [41].

Step 4: Summary of identified vulnerabilities

Within the available time frame for this investigation, five different vulnerabilities were found. All exemplary vulnerabilities are shown in tables A.1, A.2, A.3, A.4, and A.5.

Since this ECU acts as the central component for all vehicle-internal communication, external interfaces are the most critical attack surface for this ECU. The Ethernet interface can be attacked from vehicle-internal communication by the MMU or by a malicious attacker through the OBD connector. Since most ECUs of this vehicle are connected via CAN with the CGW, internal attacks over CAN are likely. The offered diagnostic services and flashing mechanisms are a rewarding target. If an attacker manages to compromise the CGW, they can manipulate the vehicle's entire communication and trigger safety-critical commands. Attacks on the debug interfaces or the ECU's external memories are interesting for attackers who either want to manipulate the immobilizer system, the mileage counter, or the permanent pairing of ECUs to a vehicle. In general, attacks on the PCB can reveal the firmware and the ECU's internal behavior, which might be the first step to prepare remote attacks.

4.1.2 Body Domain Controller

Controls all actuators in the vehicle body

BDCs control all types of body-electronics in the vehicle. This includes the entry- and immobilizer-system, power supply for body-electronic circuits, and all kinds of actuators, for example, the vehicle's horn, the wash- and wipe-system, and power window regulators. As a representative, a BDC for vehicles manufactured by GM is chosen for a manual investigation.

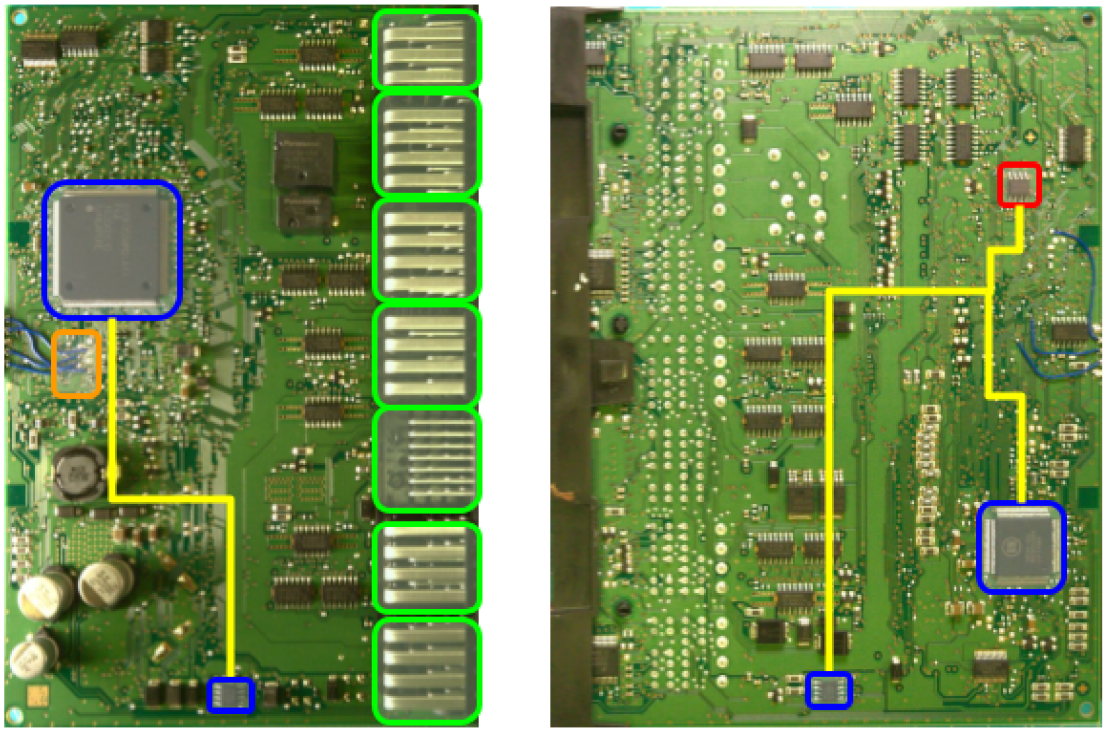

Figure 4.2: PCB of BDC manufactured by GM. The following color scheme is used for the indication of ECU properties. *Processors and Controllers* are blue, *External memories* are red, *Debug Interfaces* are orange, *On-Board Interfaces* are yellow, and *External Interfaces* are green.

Table 4.2: Investigation of a GM BDC, following the defined process.

Step 1: Role inside the vehicle
As a central component, this ECU plays a crucial role in the vehicle's functionality. Security-related information for the vehicle's immobilizer system is stored in this ECU.

Therefore this component is in charge of unlocking the whole vehicle when a driver wants to start the vehicle's engine. Besides the control over the immobilizer, the steering wheel has to be unlocked as well, and additionally, multiple other ECUs inside the car need to be supplied with power and woken up.
Therefore, this ECU controls several relays to supply different circuits during the vehicle's operation. Since this ECU is part of two different CAN networks, a high-speed CAN and a low-speed Single Wire CAN (SWCAN), it can route messages from one network to another. The investigated vehicle uses a network architecture consisting of two Line-Bus networks (see 2.2.1) for the entire in-vehicle communication. The BDC has full access to the vehicle's internal networks and can directly communicate with any other ECU inside the car.

Step 2: Identification of ECU properties

The following properties could be identified.

Processors and Controllers

The central processor is a Renesas uPD70F3558 microcontroller with internal flash memory and Random Access Memory (RAM). A proprietary Application-Specific Integrated Circuit (ASIC) on the PCB's backside shown in figure 4.2 provides multiple safety-critical functionalities to the main microcontroller. A data-sheet of this ASIC is not publicly available, but manual reverse engineering showed that this ASIC consists of multiple LIN transceivers, a hardware watchdog to reset the main microcontroller, and multiple Universal Asynchronous Receiver Transmitter (UART) and SPI connections. Such ASICs are often referenced as System Basis Chip (SBC).

External Memories

This ECU contains one external EEPROM.

Debug Interfaces

A JTAG-debug header for flashing of the central processor was found on the PCB. Serial interfaces with debug information could not be identified.

On-Board Interfaces

The central processor and the SBC for safety-critical functionality are connected via multiple interfaces. Run-time data of the processors operating system is exchanged over UART with the hardware watchdog of the SBC. Multiple different SPI interfaces are connected to the SBC as well.

External Interfaces

This ECU has multiple LIN interfaces, a high-speed CAN interface (500 kbps), and a low-speed SWCAN interface (33,3 kbps). These two CAN networks are the main communication system for vehicle-internal communication.

Wireless Interfaces

This ECU does not have any wireless interfaces.

Operating System

Inside the ECU, an embedded operating system and a bootloader are used.

Step 3: Attack surface analysis

The following attack surfaces can be considered for further security analysis.

Remote attack surfaces

CAN Since this ECU is directly connected to all other ECUs and the OBD interface, the high-speed CAN connection poses the biggest attack surface. The diagnostic protocol GMLAN is supported and used for diagnostic operations and software updates.

SWCAN The low-speed SWCAN interface of this ECU poses a smaller attack surface than the high-speed CAN interface. This ECU is mainly sending data into this network. No diagnostic protocol is supported on SWCAN by this ECU.

LIN LIN is used to communicate with peripheral components; for example, the immobilizer Radio-Frequency Identification (RFID) coils or systems like power mirror controls and power window controls. Because of the small communication payloads in LIN, attacks which lead to security vulnerabilities, like buffer overflows or remote code execution, are very unlikely.

Local attack surfaces

JTAG The JTAG interface of this ECU is locked. For example, a sophisticated hardware attack or voltage-glitching attacks, or electromagnetic-glitching attacks, are required to access the debug interface of this ECU.

External Memories Run-time data that needs to be changed often over the ECUs lifetime is usually stored in an EEPROM. These memories are usually not protected and can be read through physical connections.

Step 4: Summary of identified vulnerabilities

Within the available time frame for this investigation, three different vulnerabilities and one vulnerability chain were found. All exemplary vulnerabilities are shown in tables A.6, A.7, A.8, and A.9.

The identified vulnerabilities in this ECU allow remote code execution over the high-speed CAN network of the vehicle. Malicious actors with temporary access to the OBD interface of a vehicle can execute arbitrary code on the BDC. This can allow attackers to disable the immobilizer system of the vehicle. Safety-critical functions can be triggered, or malfunctions provoked. These vulnerabilities were used to demonstrate the capabilities of automotive ransomware [O5][P5].

4.1.3 Telematics Control Unit

Acts as mobile-communication gateway of the vehicle network

Modern vehicles have to provide a Global System for Mobile Communications (GSM) and Global Positioning System (GPS) connection to fulfill the European regulation eCall [10]. In case of a serious road accident, a modern vehicle has to transmit its current position automatically to emergency services. Vehicle manufacturers have to fulfill these requirements to be able to sell their vehicles inside the European Union. Most manufacturers added a new ECU with GSM and GPS connectivity to their existing vehicle architectures and network topologies. From the requirements of eCall, these Telematics Control Units (TCUs) need to have interfaces for both GSM and GPS in one single component. As a representative for this investigation, a TCU manufactured by LG was chosen. This TCU is used for telematic services of vehicles manufactured by GM.

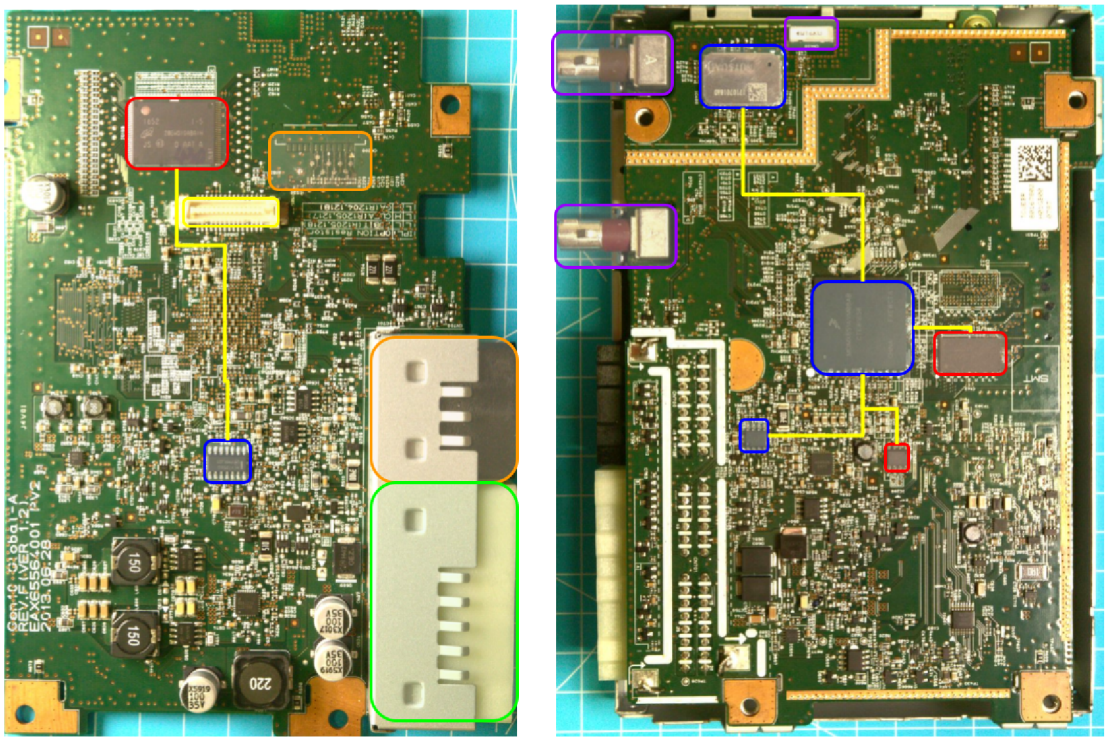

Figure 4.3: PCB of a TCU used in GM vehicles. The following color scheme is used for the indication of ECU properties. *Processors and Controllers* are blue, *External memories* are red, *Debug Interfaces* are orange, *On-Board Interfaces* are yellow, *External Interfaces* are green, and *Wireless Interfaces* are purple.

Table 4.3: Investigation of a GM TCU, following the defined process.

Step 1: Role inside the vehicle

This specific TCU provides remote connectivity for the entire vehicle. The TCU has a connection to every other ECU in the vehicle and is able to perform software updates of the entire vehicle. A WLAN access point hosted by this TCU allows passengers to use the vehicle's Long Term Evolution (LTE) connection for internet access.

Step 2: Identification of ECU properties

The following properties could be identified.

Processors and Controllers

The central processor of this TCU is an i.MX6 single core application processor from Freescale/NXP. As a WLAN controller, a Broadcom BCM4330 chipset module is used. The Network Access Device (NAD) daughterboard for the LTE and GSM connectivity uses a Qualcomm MDM9215 LTE chipset. Also, an embedded Subscriber Identity Module (SIM) card is located on the NAD board.

External Memories

The i.MX6 central processor of this ECU has an external flash and RAM.

Debug Interfaces

This TCU contains various debug interfaces. A USB On-The-Go (USB-OTG) interface to the central processor, a serial connection, and a second USB connection is available on an external connector. The second USB connection can be transformed into an Ethernet connection if a USB to Ethernet adapter with ASIX chipset is connected. On the PCB, a JTAG header, both for the central processor and the WLAN module, was found.

On-Board Interfaces

The central processor connects via USB to the NAD daughter board and the WLAN module.

External Interfaces

This ECU has a high-speed CAN interface (500 kbps) and a low-speed SWCAN interface (33,3 kbps). These two CAN networks are the main communication system for all vehicle-internal communication.

Wireless Interfaces

This ECU has the following wireless interfaces: LTE, GPS, and WLAN.

Operating System

A QNX operating system operates the central processor. The WLAN-module uses an embedded real-time operating system from Broadcom, and the NAD is driven by an embedded real-time operating system from Qualcomm.

Step 3: Attack surface analysis

The following attack surfaces can be considered for further security analysis.

Remote attack surfaces

WLAN A malicious attacker could use the vehicle's internal WLAN hotspot to attack the entire vehicle. An attacker needs to know the correct credentials to access the WLAN network, or a pre-authentication vulnerability in the WLAN-chipset needs to be present. This interface exposes a remote attack surface, which authenticated or unauthenticated attackers could potentially attack.

LTE The LTE connection itself is encrypted and authenticated. Every SIM card is registered and bound to a specific vehicle. Vulnerabilities inside the LTE chipset firmware could allow remote attacks from an attacker within the LTE or even GSM communication range. An attacker with physical access could use the SIM card to access the communication endpoints inside the Access Point Name (APN) network.

GPS GPS is a receive-only communication interface. The TCU uses the GPS signal to determine the vehicle's position. GPS can also be used as a source for the current time. Both signals, time and position, can be spoofed by malicious actors. If the information from the remote interface GPS is trusted and used for internal and security-sensitive operations, such as certificate validation, an attacker is able to compromise those.

CAN The TCU uses a powerful application processor and a complex operating system, which requires more memory compared to a firmware on embedded processors. Because of the limitations of CANs transfer rate, software updates over CAN are unlikely, which excludes this attack vector. CAN might be an attack surface to reconfigure the TCU or to cause a DoS. A reduced set of diagnostic services can be expected on the CAN interface. It might also be possible that other ECUs could use CAN communication to ex-filtrate data over the Internet connection of the TCU.

SWCAN This remote interface is probably the least rewarding for attacks. It is unlikely that this interface offers diagnostic services since this ECU has a faster CAN connection. Also, on this interface, there is a possibility that other ECUs could ex-filtrate information.

Local attack surfaces

JTAG All three processors (i.MX6, BCM4330, MDM9215) have JTAG interfaces. The interface of the MDM9215 processor is not exposed. All other interfaces can be used to debug the processor and to read out the firmware.

External Memories The i.MX6 central processor has an external flash that could be read out or modified.

Debug Interfaces Multiple debug interfaces were found on the central processor. The i.MX6 has a UART, USB, and even an Ethernet-over-USB interface exposed on a dedicated external connector of this ECU.

Step 4: Summary of identified vulnerabilities

Within the available time frame for this investigation, three different vulnerabilities were found. All exemplary vulnerabilities are shown in tables A.10, A.11, and A.12.

This TCU has multiple remote attack surfaces and an extraordinary position in the vehicle architecture, as it is directly connected to all other ECUs of the vehicle. A successful remote attack of this ECU would allow an attacker to control the entire vehicle. Vulnerabilities to allow such an attack could not be identified. The identified local attack surfaces allow comprehensive information gathering about the internal functionalities of this ECU. Access to debugging interfaces can help attackers to prepare remote attacks.

4.1.4 Airbag Control Unit

This ECU controls the airbag and restraint system of vehicles from GM.

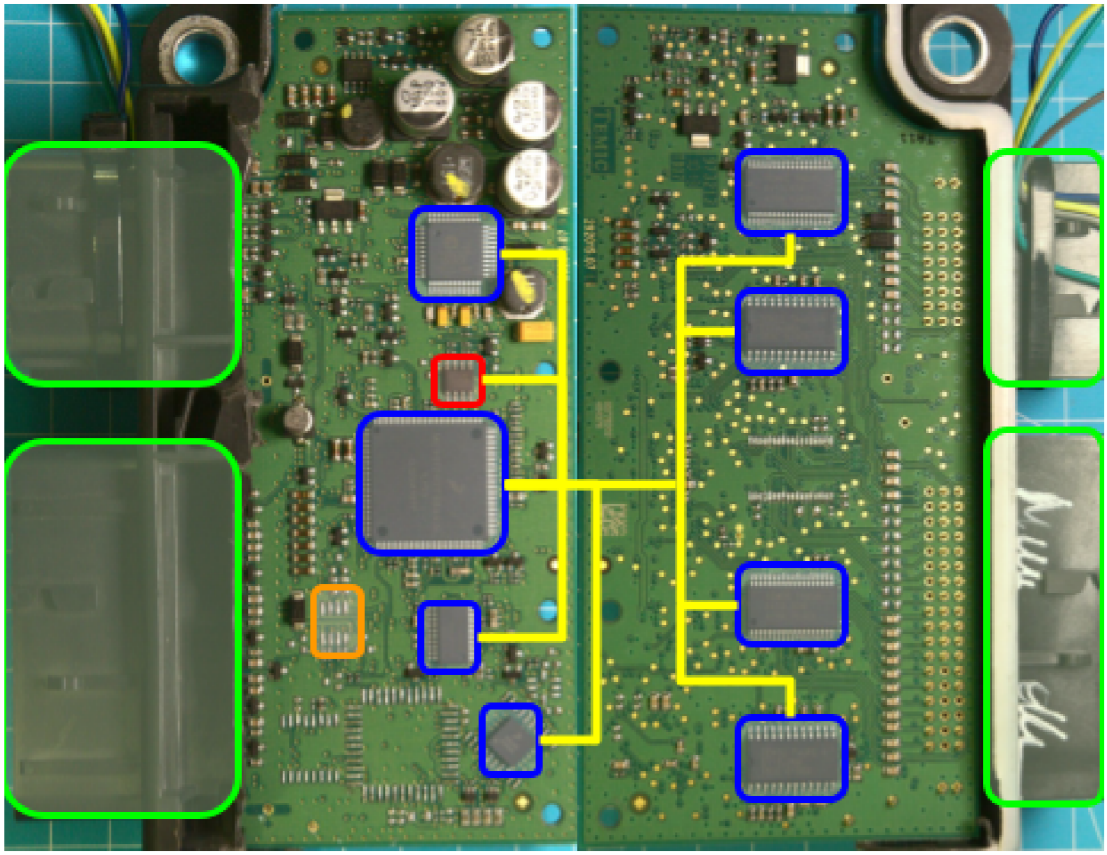

Figure 4.4: PCB of an airbag and restraint system ECU used in GM vehicles. The following color scheme is used for the indication of ECU properties. *Processors and Controllers* are blue, *External memories* are red, *Debug Interfaces* are orange, *On-Board Interfaces* are yellow, and *External Interfaces* are green.

Table 4.4: Investigation of a GM Airbag Control Unit (ACU), following the defined process.

Step 1: Role inside the vehicle

This ECU controls lifesaving systems as the airbag- and the passenger-restraint-system. Furthermore, this ECU can store pre-crash vehicle data in an internal memory. The vehicle driving data is received on a low-speed CAN connection. Since the airbag- and the passenger-restraint-system are very safety-critical components, security vulnerabilities can have a devastating impact. Outgoing communication from this ECU is only emitted if internal errors are detected.

Step 2: Identification of ECU properties

The following properties could be identified.

Processors and Controllers

The central processor is a 16-bit Freescale MC9S12XS processor, which uses a CPU12X instruction set architecture. Two SPI airbag squib driver ICs (TEMIC TAURI 4 E113.0xx) are used to launch the vehicle's airbags. Furthermore, two Micro-Electro-Mechanical System (MEMS) sensors guarantee redundant acceleration data.

External Memories

This ECU does not have external memories.

Debug Interfaces

No debug interfaces could be identified on this ECU. Reverse-engineering of the traces on the PCB is necessary to identify a debug interface of the processor.

On-Board Interfaces

The central processor connects through multiple SPI interfaces to all other controllers.

External Interfaces

This ECU has a low-speed SWCAN interface (33,3 kbps).

Wireless Interfaces

This ECU has no wireless interfaces.

Operating System

A real-time operating system for embedded systems is used.

Step 3: Attack surface analysis

The following attack surfaces can be considered for further security analysis.

Remote attack surfaces

SWCAN Since this interface is the only connection to the vehicle network, all diagnostic- and software-update-services have to be served there. This is a possible attack surface for exploitation.

Local attack surfaces

SPI This on-board interface can be used to gain information about the internal functionality of the airbag squib driver ICs.

Step 4: Summary of identified vulnerabilities

Within the available time frame for this investigation, two different vulnerabilities and one vulnerability chain were found. All exemplary vulnerabilities are shown in tables A.13, A.14, and A.15.

This ECU's connectivity is very minimal, but the safety impact a software vulnerability can have is tremendous. Code execution can potentially allow an attacker to launch airbags. The exposed diagnostic interface over CAN does not show any security hardening, and very weak security access algorithms are used.

4.2 Survey of Published Attacks on Automotive Systems

In recent years, automotive systems were regularly targeted by various cyber-attacks. This section summarizes publications with the most impact related to passenger safety. All presented attacks will be analyzed to highlight the used attack surfaces and understand the required vulnerabilities and vulnerability chains.

4.2.1 Dieter Spaar: Beemer, Open Thyself!

Remote attack of BMW's locking system

Dieter Spaar demonstrated an attack against BMW's remote control features [52]. Next to multiple other comfort features, owners can use a smartphone application to lock and unlock their vehicles. Shared cryptographic secrets and implementation flaws in the Next Generation Telematics Protocol (NGTP) communication protocol allowed Spaar to open arbitrary vehicles through a malicious Base Transceiver Station (BTS). This attack required remote services to be enabled on a victim's car. During his research, he even managed to remotely change a victim's car's configuration to enable the required remote features from his malicious BTS, allowing him to perform his attack on every vehicle from BMW, which had the proper TCU built-in. For this research, two different vulnerabilities and one vulnerability chain could be identified. All vulnerabilities and vulnerability chains are shown in tables A.16, A.17, and A.18.

4.2.2 Miller & Valasek: Remote Exploitation of an Unaltered Passenger Vehicle

Famous Jeep Cherokee hack from 2015

Miller & Valasek were able to gain full control over a vehicle through a remote attack [38]. The targeted vehicle exposed highly sensitive services on various ports. These ports were accessible through the vehicle's IP address. The absence of an APN for the vehicle's cellular connection allowed Miller & Valasek to connect to vulnerable cars over the Internet. The exposed services allowed, for example, software updates of arbitrary ECUs in the vehicle. Through malicious firmware modifications of an ECU, they could gain remote access to the vehicle's CAN bus, allowing them to control any cyber-physical function of an attacked vehicle. For this research, two different vulnerabilities and one vulnerability chain could be identified. All vulnerabilities and vulnerability chains are shown in tables A.19, A.20, and A.21.

4.2.3 Nie et al.: Free-Fall - Hacking Tesla from Wireless to CAN Bus

Security researchers from Keen Security Lab analyzed a Tesla Model S for remote exploitability [50]. Based on an already known browser exploit, they crafted an attack chain to compromise the entire vehicle. A local privilege escalation bug, introduced by an outdated Linux kernel version, allowed them to gain control over the entire MMU. This unit is also connected to the vehicle gateway and has the ability to provide software updates. The lack of signed firmware updates allowed them to craft their manipulated gateway firmware, which provides full remote access to the vehicle's CAN buses. At this point, they could exploit further ECUs through insecure UDS routines and update mechanisms. For this research, four different vulnerabilities and one vulnerability chain could be identified. All vulnerabilities and vulnerability chains are shown in tables A.22, A.23, A.24, A.25, and A.26.

4.2.4 Cai et al.: 0-days & Mitigations - Roadways to Exploit and Secure Connected BMW Cars

The same group of security researchers that already analyzed the cars from Tesla started a follow-up research project on multiple cars from BMW [6]. In this publication, they showed a remote exploit of an unaltered vehicle from BMW. The researchers crafted two very complex attack chains, consisting of vulnerabilities in multiple different ECUs. The first attack chain used a web browser exploit on the MMU as a remote entry. A time-of-check to time-of-use (TOCTOU) attack against internal diagnostic services allowed them to send arbitrary UDS messages onto the vehicle's internal CAN bus. Implementation flaws in the UDS protocol of the CGW allowed the escalation to all internal communication systems. Their second attack chain did not involve any user interaction. Similar to the attack of Spaar, they used the provisioning feature of the NGTP protocol. A buffer overflow in the TCU allowed them to obtain remote code execution. A vulnerable diagnostic service allowed them to send arbitrary messages onto the vehicle's CAN bus. From here onward, the same flaws in the CGW could be used to get access to the entire vehicle. For this research, four different vulnerabilities and two vulnerability chains could be identified. All vulnerabilities and vulnerability chains are shown in tables A.27, A.28, A.29, A.30, A.31, and A.32.

4.2.5 Computest: The Connected Car - Ways to get unauthorized access and potential implications

The Dutch security company Computest conducted an investigation of Audi and Volkswagen vehicles as an R&D project [5]. Within their research, they discovered open ports on the vehicle's MMU. A vulnerability in the operating system QNX allowed them to open a shell through a WLAN or mobile data connection. Depending on the country in which the vehicle is operated, the remote connection might be secured by an APN from the telecommunication service operator. Further vulnerabilities in the QNX system and internal services of the MMU offer a local privilege escalation vulnerability. Through a firmware modification of the CAN Microcontroller Unit (MCU) inside the MMU, they obtained arbitrary write access to the vehicle's CAN bus. At this point, they stopped their research. An attack of the vehicle gateway ECU would have been necessary to compromise the entire vehicle. For this research, three different vulnerabilities and one vulnerability chain could be identified. All vulnerabilities and vulnerability chains are shown in tables A.33, A.34, A.35, and A.36.

Chapter 5

Analysis of Identified Vulnerabilities and Attack-Surfaces

To conclude part II of this thesis, an evaluation of the performed ratings will be conducted, aiming to reveal common properties of each identified attack surface and defense strategies to lower the exploitation risk of vulnerability chains. In preparation for chapter 6, each attack surface is analyzed for its test-ability and its automation capabilities for security tests on a component level.

This chapter targets the following research questions:

- **Q1:** Which impact on the overall safety-critical system do individual vulnerabilities and vulnerability-chains have?
- **Q2:** Which attack surfaces can be analyzed automatically?

The following contributions will answer these questions:

- **A1:** An evaluation of all vulnerabilities and vulnerability chains regarding their exploitation risk and mitigation strategies (5.1).
- **A2:** An overview of capabilities for automated discovery and testing of attack surfaces (5.2).

At the end of this chapter, common attack surfaces in automotive components are identified, their impact on the overall safety-critical system is estimated through a scoring system, and the possibility for automated security tests on individual attack surfaces is evaluated.

5.1 Evaluation of Identified Vulnerabilities

The evaluation of all identified vulnerabilities is structured in two parts. First, average scores of all vulnerabilities clustered by their attack surface and vulnerability chains scores are compared. Secondly, vulnerability chains are analyzed, and the impact of mitigation strategies is discussed.

5.1.1 Analysis of Vulnerability Ratings

Section 4.1 and section 4.2 show a collection of 28 different vulnerabilities and eight different vulnerability chains. Every vulnerability and vulnerability chain is rated with an impact and exploitability factor, according to the definitions in table 3.2 and table 3.3.

Table 5.1 shows the average impact and exploitability factors of all identified vulnerabilities clustered by the attack surface. Besides evaluating single vulnerabilities, vulnerability chains need to be considered independently to better understand the differences in the exploitation risk. All vulnerability chains are shown in table 5.2.

The identified vulnerability chains consist of two to four individual vulnerabilities. The combination of multiple vulnerabilities leads to a higher impact and exploitability factors. Table 5.2 shows that the average impact and exploitability factor increased for all vulnerability chains than average values of single vulnerabilities.

Table 5.1: Summary of vulnerabilities grouped by attack surfaces. The ratings Low, Medium, and High correspond to the numeric values 1, 2, and 3.

No.	Name	Impact	Exploit.
Attack Surface: External Memories			
V1	Unprotected external memory	Low	Low
V6	Unprotected external memory	Low	Low
V9	Unprotected external memory	Medium	Low
V14	Shared cryptographic secrets in TCU	Low	Low
	Total:	**1.25**	**1**

Attack Surface: Debug Interfaces			
V4	Serial debug interface available	Low	Low
V5	Insecure MCU internal bootloader	Medium	Low
V10	Unprotected JTAG interface	Medium	Low
V11	External Debug-Port	Medium	Low
	Total:	**1.75**	**1**
Attack Surface: On-Board Interfaces			
V17	Insecure software update mechanisms	High	Low
V20	Unauthenticated software update mechanisms	High	Low
V28	Insecure software update mechanisms	Medium	Low
	Total:	**2.67**	**1**
Attack Surface: External Interfaces			
V2	Diagnostic message routing	Medium	Medium
V3	Development functionalities	Low	Medium
V7	Insecure Security Access mechanism	Medium	Medium
V12	Weak Security Access algorithm	Medium	Medium
V21	Insecure UDS protocol	High	Medium
V24	Insecure UDS message routing	High	Medium
	Total:	**2.17**	**2**
Attack Surface: Wireless Interfaces			
V15	Information leaks in NGTP	Low	High
V16	Exposed services on remote interface	Medium	High
V18	Browser exploit	Medium	High
V22	Browser exploit	Medium	Medium
V25	Buffer overflow in the NGTP protocol	Medium	High
V26	Open ports on the vehicles network interfaces	Low	High
	Total:	**1.67**	**2.83**
Attack Surface: Operating System			
V8	Insecure software update mechanism	High	Low
V13	Remote Code Execution (RCE) vulnerability	High	Low
V19	Linux kernel exploit	Medium	Low
V23	TOCTOU attack	Medium	Low
V27	QNX vulnerability	Medium	Low
	Total:	**2.4**	**1**

Table 5.2: Summary of vulnerability chains. The ratings Low, Medium, and High correspond to the numeric values 1, 2, and 3.

No.	Vulnerabilities	Impact	Exploitability	Ref.
C1	V7, V8	High	Medium	A.9
C2	V12, V13	High	Medium	A.15
C3	V14, V15	High	High	A.18
C4	V16, V17	High	High	A.21
C5	V18, V19, V20, V21	High	High	A.26
C6	V22, V23, V24	High	Medium	A.31
C7	V24, V25	High	High	A.32
C8	V26, V27, V28	Medium	High	A.36
	Total:	**2.88**	**2.62**	

Figure 5.1 visualizes the average exploitation risk of all vulnerability chains and all vulnerabilities grouped by attack surfaces. With an average exploitability score of 7.55 (2.88×2.62), vulnerability chains have the highest exploitation risk. Another remarkable outcome of this analysis is the distribution of impact and exploitability factors across the individual attack surfaces' vulnerabilities. An attack surface is either more likely to be exploited but has a lower impact factor or is unlikely to be exploited but has a higher impact factor.

Wireless Interfaces have a medium to high exploitability factor (2.83) combined with a low to medium impact factor (1.67), resulting in an exploitability risk score of 4.73. This relatively low impact factor is an effect of the applied hardening mechanisms onto the vehicle architectures, as described in section 2.2.2.

External Interfaces, from an ECU's point of view, include all vehicle-internal interfaces. This attack surface has a medium to high impact (2.17) and a medium exploitability (2). With an exploitability risk score of 4.34, this attack surface is the second likeliest entry point for exploitation. The impact factor would be higher on this attack surface if OEMs did not already apply physical hardening mechanisms through network separations.

The attack surfaces External Memories, Debug Interfaces, and On-Board Interfaces require physical access for any exploitation attempt, which is the main reason

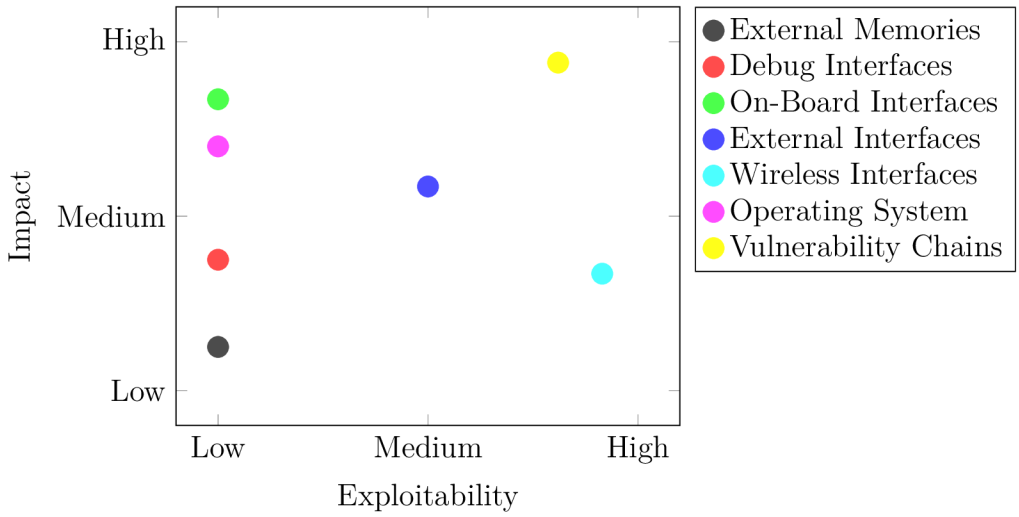

Figure 5.1: Distribution of average exploitability factors and average impact factors for each attack surface and all vulnerability chains.

for the low exploitability factor. The attack surface Operating System requires access to the system, usually unprivileged code execution, resulting in the low exploitability factor for automotive systems. These four attack surfaces' impact factors vary from low to high, depending on the possibilities an attacker gains from a successful attack. In all investigated systems, code execution was not easy to achieve.

5.1.2 Analysis of Vulnerability Chains

All published attacks on automotive systems used a more or less complex vulnerability chain. The chaining of vulnerabilities is necessary to overcome architectural mitigation strategies, OEMs applied to their cars. A common goal of every attack is unlimited access to the vehicle's internal network, which results in a high impact factor of a vulnerability chain.

The vulnerabilities V8, V13, V17, V20, and V28 are targeting software update mechanisms of vehicle components which are connected to the vehicle-internal networks. In all cases, these vulnerabilities require an additional vulnerability to get access to the interface on which the safety-critical components receive their updates. In the analyzed chains, these were either CAN networks or On-Board networks on the ECU's PCB.

To better understand the impact of vulnerability chains and the defense capabilities if these vulnerability chains could be broken up, the following hypothetical assumption is made:

Assuming that the software update mechanisms would not have any security flaws, V8, V13, V17, V20, and V28 would not be present.

This assumption would drastically change the impact factors of the vulnerability chains C1, C2, C4, C5, and C8, which then would have a medium instead of a high impact factor. Furthermore, this would result in a lower exploitation risk of a car. The majority of the collected vulnerability chains (5 out of 8) would be affected by this assumption which shows that this defense strategy would lower exploitation risk in most cases.

Table 5.3: Comparison of real exploitation risk (on the left side) with the hypothetical exploitation risk (on the right side). The ratings Low, Medium, and High correspond to the numeric values 1, 2, and 3.

	Real Risk		**Hypothetical Risk**	
No.	**Impact**	**Expl.**	**Impact**	**Expl.**
C1	High	Medium	Medium	Medium
C2	High	Medium	Medium	Medium
C4	High	High	Medium	High
C5	High	High	Medium	High
C8	Medium	High	Medium	High
Total:	**2.8**	**2.6**	**2**	**2.6**
Risk:	**7.28 / 9**		**5.2 / 9**	

Table 5.3 shows that the lowering of one factor for the exploitation risk calculation results in a decrease in the overall exploitation risk. This hypothetical example should highlight the capabilities of security measures that target the decreases of either the impact or the exploitability factor. Such mitigation, which increases the complexity of successful exploitation, is based on the *Defense in Depth* hardening strategy introduced by the U. S. National Security Agency [2]. Especially in automotive systems, components have either high exploitation factors or high impact factors (see figure 5.1). This circumstance fits perfectly to lower the overall exploitation risk

through a *Defense in Depth* strategy. The *Defense in Depth* strategy can break up vulnerability chains by improving individual components' security, which delivers a cost-efficient method to harden a complex automotive system against safety-critical cyber-attacks.

5.2 Analysis of Automation Capabilities

Automated testing is crucial to improve the cost efficiency of security investigations and penetration tests. Software systems, nowadays, have overall good automation capabilities for quality assurance tests. Also, the research field of automated security tests for software products delivers promising results. On embedded systems, the same progress has not taken place yet. Automated functional tests for hardware systems are well understood and performed on all automotive components, but security-related automated tests are an exception and only performed on a very limited set of systems.

The studied publications demonstrated the automated execution of all vulnerability chains. In some publications, the usage of automated tools for vulnerability discovery was explicitly mentioned. Miller & Valasek and researchers from Computest used the tool *Nmap* during their investigations to identify the vulnerabilities V16(A.19) and V26(A.33) [34].

To better understand hardware-related security testing difficulties, every attack surface will be examined for its automation capabilities while also discussing required challenges to be solved. Furthermore, the possibility of automated identification and exploitation of vulnerabilities will be discussed.

5.2.1 External Memories

Low automation capabilities

All identified vulnerabilities related to the attack surface External Memories are listed in Table 5.1. The average exploitability factor of 1 already indicates difficulties for automated security tests. Spaar described the entire process for his attack [52]. This clearly shows that the process of memory extraction from a sophisticated target PCB has no capabilities for automation. On the other hand, if targets were more basic, which mainly describes that the form factor of external memory is easier to access, open-source tools like *flashrom* can be used for semi-automated memory extraction [12].

In contrast, the analysis of extracted memory can be partly automated to extract specific information. Binary analysis tools, for example, *binwalk*, can identify data of interest [46]. Entropy analysis of the memory content will immediately highlight cryptographic material. All under the assumption, the extracted data is not encrypted. In general, the automation capabilities for the attack surface External Memories are low.

5.2.2 Debug Interfaces

Automated vulnerability assessments of Debug Interfaces on PCBs face similar difficulties as described for External Memories. On automotive systems, mainly UART and JTAG interfaces are used for debugging purposes (see table 5.1). In most cases, these interfaces do not provide an easy to access connector. Therefore, manual modifications need to be made to a target PCB. Tools like the *JTAGulator* can assist during the identification of these interfaces [53]. Vulnerability assessments are very target-specific and need to be done manually. In summary, the automation capabilities for the attack surface Debug Interfaces are low.

Low automation capabilities

5.2.3 On-Board Interfaces

On-Board Interfaces are the third attack surface that requires physical access to a PCB. These interfaces are a great source to gather internal information about a target. An example of how On-Board Interfaces can be used to attack embedded targets was shown in a presentation at the Troopers Conference 2019 and in the publication *Extending Vehicle Attack Surface Through Smart Devices* [O4][P3]. The same difficulties as on Debug Interfaces and External Memories also apply for On-Board Interfaces. Manual investigation and modifications of the PCB are required in order to get access to them. The automation capabilities of this attack surface correlate with their low exploitability.

Low automation capabilities

5.2.4 External Interfaces

External Interfaces stand for all wired interfaces that connect with an ECU. The most relevant physical protocols for security investigations are explained in detail in section 2.1, and important communication protocols are summarized in section 2.3. In recent years, many open-source tools for penetration attempts in automotive networks were published. In cooperation with E. Pozzobon, a list of open-source

High automation capabilities

tools for automotive network penetration testing was published [O1, table 1 on page 3]. All identified vulnerabilities on the attack surface External Interfaces (see table 5.1) could have been discovered automatically. From the perspective of a security researcher or penetration tester, the physical interfaces CAN, Automotive Ethernet, and Ethernet are easy to access. These interfaces provide stable communication since the communication medium is not shared. All together, vulnerability discovery on External Interfaces has a high potential for automated testing.

5.2.5 Wireless Interfaces

Low automation capabilities for Wireless Interface in general

The attack surface Wireless Interfaces describes a wide variety of different communication technologies. A detailed collection of wireless interfaces is given by Checkoway et al. [7, section 3.2 and 3.3]. Only the mobile data connection and WLAN connections were attacked (see table 5.1). This attack surface has two negative aspects in terms of automation. First, the huge variety of communication standards require special testing equipment for each interface. Second, every standard has its unique protocol structure, which requires individual effort to establish a connection. These two aspects result in very complex and expensive test setups. All wireless interfaces use the shared communication medium air. External interference will additionally increase the difficulties for automated test setups. The automation capabilities of the entire attack surface Wireless Interfaces have low capabilities for automated vulnerability assessments. Nevertheless, it is essential to mention that a large amount of tools for individual communication standards exists.

Individual wireless technologies can have higher automation capabilities

5.2.6 Operating Systems

Low automation capabilities

This attack surface summarizes all vulnerabilities which require an implementation bug or logic error in the operating system or bootloader of an ECU. The analyzed systems showed a huge variety, from a bare-metal real-time operating system to a full-fledged Linux operating system. This circumstance makes automated analysis of vulnerabilities complicated since the systems are fundamentally different. For those reasons, the automation capabilities of the attack surface Operating Systems are rated low.

5.3 Summary

Safety-critical vulnerabilities in automotive systems are realistic, and various researchers were able to demonstrate multiple successful attacks. In all real-world scenarios, vulnerability chains were required to overcome architectural difficulties and mitigation strategies. The final goal of every exploitation is unlimited access to vehicle-internal networks with connections to CPSs. Through a *Defense in Depth* strategy, the exploitation risk can be reduced by interrupting the necessary vulnerability chains. The positive effects of hardening strategies applied to the vehicle network architectures could already be demonstrated by an in-depth analysis of required vulnerability chains in published attacks. Furthermore, it was demonstrated that hardening efforts applied to individual components are an efficient mitigation strategy to lower the automotive system's overall exploitation risk.

Defense in Depth strategies already secure automotive architectures

A cost- and time-efficient solution to increase a systems security level is automated security testing. All identified attack surfaces were analyzed for their automation capabilities. The most suitable attack surfaces for automated security testing are wired vehicle-internal networks (External Interfaces of an ECU). Such automated tests have the potential to harden the vehicle's internal networks against exploitation sustainably and lower the overall impact factor, resulting in a lower overall exploitation risk of an entire vehicle.

Wired vehicle networks have high automation capabilities for security tests

Part III

Automated Security Investigations of Safety-Critical Systems

Chapter 6

Tools for Security Investigations of Safety-Critical Networks

Part II provided two essential insights. First, vehicle-internal networks contribute disproportionately to the impact of security vulnerabilities. Second, these networks have outstanding potential for automated security tests and vulnerability scans.

This chapter will focus on tools and methodologies for automated security evaluations in wired vehicular networks. As the first step, a suitable open-source software framework for tool development will be chosen. Second, the diagnostic protocol stack in automotive networks is introduced. Further, this stack will be dissected into its layers, and opportunities for automated testing on each layer will be discussed.

In summary, this chapter contains the following contributions:

- A summary of the implementation of an open-source software framework for automotive penetration testing.
- A description of contributed tools for every layer in the automotive diagnostic protocol stack.
- A novel approach for ISO-TP endpoint identification on CAN-based networks.

6.1 Selection of a Software Framework

To avoid reinvent the wheel, an existing software framework should be used as a starting point for creating a toolbox to support automated vulnerability scanning in

automotive networks. Such a toolbox should fulfill the following requirements:

1. **Open-Source Software:** All tools are based on open-source software to ensure freedom of use and to leverage an already existing software basis.

2. **Portability:** The tools are usable on all major operating systems.

3. **Community:** An active community ensures longevity.

As the first step, the following open-source tools were selected for further analysis of suitability: Busmaster [45], CANalyzat0r [49], cantoolz [51], Caring Caribou [47], Scapy [43], and Metasploit [44].

Fundamental analysis of open-source tools can be done by evaluating the version control system's history. All chosen software frameworks use `git` for this purpose. Most important for a project's vitality are the age of a project and the number and the frequency of contributions, which quickly reveal if an existing community actively maintains a project.

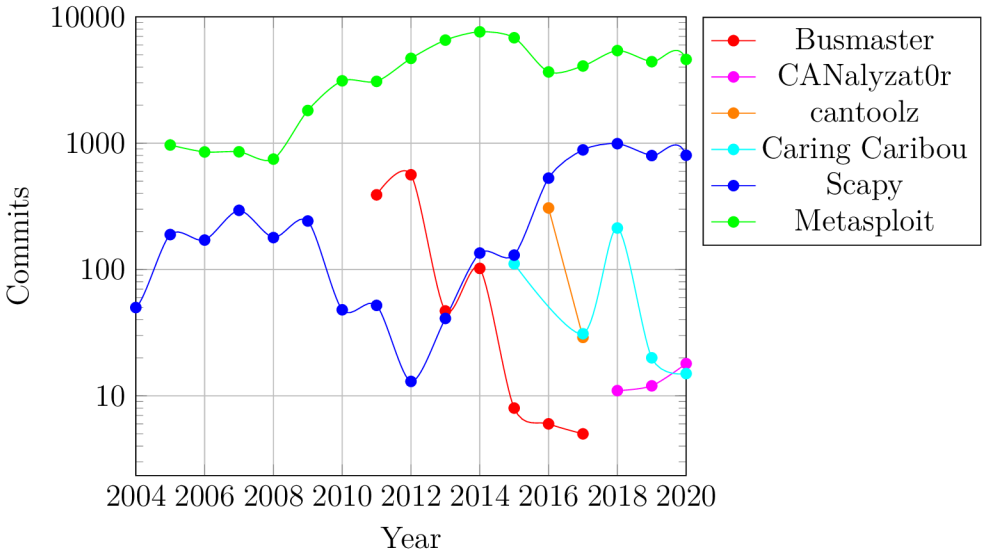

Figure 6.1: Evaluation of commits per year for the open-source projects Busmaster [45], CANalyzat0r [49], cantoolz [51], Caring Caribou [47], Scapy [43] and Metasploit [44].

The commits per year plot in figure 6.1 clearly shows significant differences between the individual projects. The projects cantoolz and Busmaster did not receive any contribution since 2018. The projects CANalyzat0r and Caring Caribou are still under development, but the number of commits is low. This indicates that no active community for these projects exists. As a counterexample, the projects Scapy and Metasploit show the contribution behavior of an active community. Over the last five years, the project Scapy received more than 800 commits on average per year, the project Metasploit more than 5000.

Because of the given requirements, the only projects worth considering are *Scapy, a powerful Python-based interactive packet manipulation program and library*, and *Metasploit, The world's most used penetration testing framework*. Scapy was chosen for further development between these two projects because of its outstanding capabilities for IEEE 802.3 based network penetration testing. The communication systems for vehicle-internal networks will develop into this technological direction, which makes *Scapy* a future safe choice for a network penetration testing tool and framework. *Metasploit*, on the other hand, mainly focuses on penetration testing of web-applications, end-user computer systems, and servers, which does not make it a perfect fit for the desired use-cases in automotive penetration testing.

The following sections will reference contributions to this open-source project as part of the efforts to form a general and open toolkit for automotive penetration testing in vehicular networks.

6.2 Dissection of the Automotive Diagnostic Protocol Stack

Recent developments in the automotive industry show the transition from CAN-based networks to IEEE 802.3 based networks. Therefore, the focus of this chapter lies on application layer protocols, for example, diagnostic protocols. This allows the application of the following tools and methodologies, which only dependent on an application layer protocol, on top of both protocol stacks (CAN and IEEE 802.3) if the abstraction to the transport layer is implemented transparently. For comparison, these stacks are shown in figure 2.5.

6.2.1 Media Access Layer Contributions

In 2018, an overview of media access solutions for CAN penetration testing was published in cooperation with Enrico Pozzobon [O1]. The release "A Survey on Media Access Solutions for CAN Penetration Testing" defined the general requirements of software tools for penetration testing in vehicular networks. Furthermore, a performance analysis of different media access solutions was conducted. The used test setups were designed to benchmark use cases, common for penetration test applications. Furthermore, this publication demonstrates that open-source software and hardware media access solutions can provide a decent performance compared to expensive professional media access equipment.

Another outcome of this paper was mapping open-source tools to the used software framework for media access [O1, Table 1]. Most of the presented tools use either Linux *SocketCAN* [18] or *python-can* [55] as a media access framework. Each framework follows its individual design goals. The *python-can* framework is highly flexible due to its support of all major operating systems and CAN bus interfaces. A communication object in *python-can* does not have an intermediate layer for message queuing. *python-can* provides a standardized software interface to a CAN network device. Linux *SocketCAN*, on the other hand, provides a full-featured socket interface to a CAN network device. This demands the usage of Linux kernel functions for message queuing and filtering. Both frameworks have their advantages and disadvantages.

6.2.2 Transport Layer Contributions on CAN

As mentioned in section 2.3.2, ISO-TP is used to exchange payloads greater than eight bytes between two ISO-TP endpoints. A set of communication parameters is required to establish communication with an ISO-TP endpoint successfully. A novel approach to automatically detect ISO-TP endpoints in CANs was published at the Computer Science in Cars Symposium [O2].

Once the necessary communication parameters are known, one can establish communication with an ISO-TP endpoint. A special kernel module for ISO-TP communication can be used on Linux-based systems to achieve a socket-like communication [16]. A comparable solution does not exist for Windows or OSX-based systems.

The novel ISO-TP scanner utility is an important key to enable automated security tests in CAN-based networks. This utility's unique capability is to identify ISO-TP endpoints independently from the application layer protocol on top. However, security vulnerabilities are more likely to exist in an application layer protocol (see section 7.1.2). After the identification of ISO-TP endpoints, additional tools can perform additional security tests on the application layer.

6.2.3 Transport Layer Contributions on IP networks

The latest vehicles use HSFZ or DoIP for diagnostic communication. As shown in figure 2.5, these two protocols are located in the automotive diagnostic protocol stack's transport layer. Increasing bandwidth demands made the use of IEEE 802.3 based communication necessary. Since this communication technology is widely used, various tools from classical IT-systems can now be used for information gathering and analysis of automotive networks. Two remarkable open-source tools are:

- **Wireshark** is the world's foremost and widely-used network protocol analyzer [9]. The latest version of Wireshark supports the automotive protocols DoIP and UDS. Wireshark allows to analyze any network traffic exchanged with a vehicle.
- **Nmap** is a port scanner for IP networks [34]. With Nmap, open sockets on ECUs can be identified, which helps to find endpoints for DoIP or HSFZ communication if they do not reveal themselves through announcement messages.

Both protocols can act as a routing protocol for UDS communication into vehicle subnetworks. This requires additional address information similar to source and destination addresses in the ISO-TP communication protocol.

To enable automated security scanning of application layer protocols on top of these communication protocols, protocol descriptions and application layer sockets were contributed to the *Scapy* project. The protocol descriptions allow packet creation and fuzzing on the transportation layer. Transparent application layer sockets allow the application layer to application layer communication to support protocol stack independent diagnostic protocol examinations.

6.2.4 Application Layer Contributions

The groundwork for each protocol's basic support was necessary to allow the creation of automated scanners for application layer protocols (GMLAN, UDS, XCP,

OBD). My additions to the *Scapy* project enable network packet creation and traffic interpretation for these automotive diagnostic protocols.

6.3 Summary of Contributions to Scapy

The entire groundwork for advanced scanning techniques of automotive diagnostic protocols is summarized in table B.1. All self-developed contributions are open-source, targeting the objective to support free automotive security research and development.

Chapter 7

Automated Security Investigations of Safety-Critical Networks

This chapter shows a novel approach for threat analysis and measurement of the possible attack surfaces in automotive diagnostic protocols. With an automated scanner algorithm and its implementation, in-depth information gathering on automotive systems is demonstrated on real-world automotive components.

In summary, this chapter contains the following contributions:

- Threat definitions and a threat model of the possible attack surface of automotive components.
- An algorithm for automated system state reverse-engineering of hidden system states in automotive diagnostic protocols.
- A new approach for attack surface estimation of automotive diagnostic protocols over a vehicle's lifetime.

7.1 Threats for Automotive Diagnostic Protocols

This section introduces novel threat definitions for automotive diagnostic protocols based on proven and demonstrated attacks. With an open-source scanner, introduced in section 7.2, automated information gathering in vehicular networks can be realized. This scanner uses automated system state reverse engineering to obtain a system state machine of an automotive system. Combining threat definitions with a reverse-engineered system state machine forms a novel threat model for the potential

attack surface of automotive systems. Parts of the following sections were published at the Embedded Security in Cars Workshop (ESCAR) USA in May 2021 [O3].

The following implementation of a diagnostic protocol scanner will focus on UDS and GMLAN, widely used in the automotive industry and supported by almost every car. Both of these diagnostic protocols allow various commands for diagnosis, programming, or even safety-critical operations on an ECU [20]. Vulnerabilities and implementation flaws in both protocols have been used in the past to attack vehicles. These attacks will be discussed, and general threat definitions for these protocols' services will be presented. During this research, custom services for development or provisioning purposes from OEMs could be identified. In general, undocumented UDS services are a good indicator of hidden functionalities, often with extensive privileges.

7.1.1 Demonstrated attacks

Some vulnerabilities that were presented in chapter 3 were already related to diagnostic protocols. The concrete abuse of these vulnerabilities will be evaluated to determine generic threats for a protocol's features.

- **V3** (A.3) The OEM added proprietary additions to the UDS protocol implementation. Attackers can leak sensitive information, read memory, and access services for message routing.
- **C1** (A.9) An insecure implementation of GMLANs authentication mechanism allows attackers to obtain extensive execution privileges. Combined with authentication issues in the software update mechanism, a remote code execution attack over CAN was demonstrated [P5].
- **C2** (A.15) This vulnerability chain is very similar to C1, which proves the repetition of vulnerability patterns in different ECUs.
- **V21** (A.25) Researchers could execute safety-critical functionalities of UDS without authentication.
- **V24** (A.29) Similar to V3, logical issues in the message routing of the gateway ECU exposed safety-critical functionalities of UDS to attackers.
- Miller & Valasek conducted an in-depth security analysis of diagnostic protocols [36]. They have proven the safety-criticality of various features in diagnostic protocols for two different vehicles.

- Garcia et al. focused on security access mechanisms and analyzed remote code execution capabilities of eleven different ECUs [57]. Their research was focused entirely on diagnostic protocols.
- Pareja and Cordoba demonstrated hardware attacks on automotive diagnostic protocols [42]. Fault injection attacks allowed them to bypass authentication mechanisms and arbitrary memory extraction to obtain an ECU's firmware.

7.1.2 Threat Definitions

The demonstrated attacks (section 7.1.1), combined with a threat analysis of the protocols GMLAN and UDS, will be used to develop general threat definitions for automotive diagnostic protocols. The proposed threat definitions are based on the Common Vulnerabilities and Exposures (CVE) flaw terminology [40]. Since CVE flaw types are initially designed for web applications, office applications, 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 7.2, 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 in-depth impact analysis of a diagnostic protocol implementation. Table 7.1 proposes a measurand and a measuring unit for every flaw type to achieve a comparable rating. [O3]

Table 7.1: 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`

Table 7.2: 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 explains why a certain flaw type is possible. A list of all service names of UDS and GMLAN can be found in appendix C in table C.1.

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 (DoS) individual ECUs [39, p. 9][50, 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 [6, 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 [42].
`27h`	`27h`	*crypt*	Van den Herrewegen et al. and Dürrwang et al. demonstrated impacts of weak cryptographic implementations [57, 11].
		pass	Miller & Valasek revealed many hard-coded cryptographic secrets inside an ECUs firmware [36, p. 46].
		rand	Nie et al. analyzed weak security access implementations and showed the lack of random seed creation [50, 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 [29, p. 7].
2Dh		*int-overflow*	This service specification describes two possible use-cases, clearing of non-volatile memory and changing of calibration values [22, p. 147]. Both use-cases can be used to cause program flow corruptions, e.g. integer- or buffer-overflows.
		buf	See above. Identical to *int-overflow*.
2Eh	3Bh	*int-overflow*	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.
		phys	Cai et al. demonstrated the manipulation of the driver's seat position through this service [6, p. 8].
		buf	Payloads can contain complex data, e.g. certificates or ring buffer contents. Increasing data size and complexity leads to a higher 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 [36, p. 15].
31h		*dos-flood*	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 [39, p. 12].
		buf	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 TOCTOU attack, which led to code execution [6, p. 8].

		phys	RoutineControl jobs can be used to control actuators on a vehicle. Miller & Valasek were able to kill a vehicle's engine [36, p. 51]. Dürrwang et al. showed the deployment of airbags through insecure implementations of RoutineControl jobs [11].
		infoleak	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 [39, 57].
`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*	Koscher et al. demonstrated the possibility of triggering physical actions on ECUs [29, p. 8].
		dos-flood	The GMLAN standard describes the possibility to trigger an ECU reset [15].

7.2 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. [O3]

7.2.1 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, the bootloader and the application software are 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. [O3]

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. [O3]

7.2.2 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 7.3 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. [O3]

Table 7.3: Summary of state modifying services in UDS and GMLAN. [O3]

	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

7.2.2.1 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. [O3]

7.2.2.2 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.[O3]

7.2.2.3 Security Access Testing

System states which grant 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 Dürrwang et al. and Nie et al. [11, 31].

$$key = \neg seed \tag{7.1}$$

- **Mathematical Operations**
  The security access mechanism of one analyzed OEM relies on complex mathematical operations. To obtain a key, 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:

$$key = (seed * secret1 + secret2) \bigoplus (seed * secret3 + secret4) \bigoplus secret5 \tag{7.2}$$

- **Proprietary XOR-Shift-Loop**
  Security access algorithms for this group were analyzed in-depth by Van den Herrewegen et al. [57]. 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:

$$key = RSA_{sign}(MD5(seed \mid salt), private\_key) \tag{7.3}$$

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, 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. [O3]

7.2.2.4 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. [O3]

7.2.2.5 Summary

The explained properties of system state graphs in automotive diagnostic protocols can be defined as follows: [O3]

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 transition 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.*

7.2.3 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 was executed. This is necessary to know if an Enumerator has finished its scan of a service. [O3]

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. [O3]

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. [O3]

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 classified information under Non-Disclosure Agreements (NDAs). `SecurityAccess` algorithms are one example of such classified information. 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. [O3]

7.3 The Attack Surface Model for Automotive Diagnostic Protocols

Combining the previously introduced threat definitions (table 7.2) 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). [O3]

Definition 2 *A threat 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 7.2, 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 $F_{min}(t) = 1 - (1 - F_1(t))(1 - F_2(t))$ [3]. 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. [O3]

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 threat model resulting from the automated scan. The necessary steps are the following: [O3]

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. [O3]

Chapter 8

Evaluation

This chapter discusses gathered results from thirteen analyzed ECUs.

8.1 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 [16]. 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 TCP sockets were sufficient. 13 different ECUs (shown in table 8.1) 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. For every investigated ECU, the following manual installation steps were necessary: [O3]

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.

Table 8.1: Overview of investigated ECUs. A label **E{x}** is assigned to each ECU for later reference. [O3]

Ref.	OEM	ECU Type	Part No.
E1	BMW AG	Gateway ECU	LR-01
E2	BMW AG	BDC	LR-01
E3	BMW AG	Gateway ECU	9243211
E4	BMW AG	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

8.2 Scan Duration

First, the scan algorithms' general runtimes are discussed to provide a basic overview of the performed scans. Table 8.2 shows the number of sent, received, and missed packets, as well as the average response time per request. These measurements are used to compute a theoretical scan duration based on the average response time and the number of answered requests. Next to this theoretical scan duration, the scan algorithm's actual execution time is shown in table 8.3. Interesting in this table is the computed scan overhead. This overhead is the difference between the actual execution time and the computed scan duration. During this overhead time, the scan algorithm performs resets, security access authentications, and state changes of the scan targets. Compared to the actual scan time, the high overhead is mainly caused by the necessary delays of certain ECUs. Every state change interrupts a scan execution and requires a reset of the scan target.

Table 8.2: Captured runtime metrics of all performed scans. This table shows the number of requests (REQ.), responses (RESP.), and unanswered or timed-out requests (TMO.). Furthermore, the computed average response time (AVG. RESP.) from all answered requests is shown.

	REQ.	RESP.	TMO.	AVG. RESP.
E1	3528349	3528209	140	0.711 ms
E2	2038900	2038859	41	1.547 ms
E3	2040410	2040298	112	1.726 ms
E4	1711320	1711302	18	1.572 ms
E5	1720970	1720408	562	9.675 ms
E6	1665907	1664711	1196	4.909 ms
E7	735608	735591	17	19.65 ms
E8	549750	542980	6770	53.51 ms
E9	97988	97950	38	19.396 ms
E10	2672670	2672449	221	4.447 ms
E11	2055886	2054380	1506	7.742 ms
E12	3359694	3359392	302	1.388 ms
E13	1026945	1026925	20	8.65 ms

8.3 Automated System State Reverse-Engineering

The scan algorithm was able to identify multiple different system states for each ECU. All ECUs showed individual system state machines, even if the same manufacturer developed them. As an example, figure 8.1 shows two different system state machines with all transitions. This gives an impression of how different system state graphs can be. Table 8.4 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. [O3]

8.4 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 were able enter the bootloader if the correct sequence of commands is sent. ECUs which implement the GMLAN protocol showed

Table 8.3: Captured duration metrics of all performed scans. This table compares the measured duration (DUR.) with the computed duration (CDUR.) of all performed scans. The computed duration is the product of all answered request times, the average response time from table 8.2. The scan overhead (OVHD.) and the proportional overhead (POVHD.) are given from the difference of the measured duration and the computed duration.

	DUR.	CDUR.	OVHD.	POVHD.
E1	14 500 s	2 509 s	11 991 s	83 %
E2	10 162 s	3 154 s	7 008 s	69 %
E3	10 700 s	3 522 s	7 178 s	67 %
E4	8 739 s	2 690 s	6 049 s	69 %
E5	18 909 s	16 650 s	2 259 s	12 %
E6	18 822 s	8 178 s	10 644 s	57 %
E7	76 169 s	14 455 s	61 714 s	81 %
E8	74 467 s	29 417 s	45 050 s	60 %
E9	4 679 s	1 901 s	2 778 s	59 %
E10	21 734 s	11 885 s	9 849 s	45 %
E11	22 466 s	15 917 s	6 549 s	29 %
E12	15 718 s	4 663 s	11 055 s	70 %
E13	11 188 s	8 883 s	2 305 s	21 %

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. [O3]

Negative response messages have a fixed size and fit in a single CAN frame. Therefore the timings of negative responses, shown in table 8.5, 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 8.5 show a significant change in an ECUs communication behavior, caused by the execution of different firmware. Since the bootloader has a smaller codebase and less parallel tasks 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

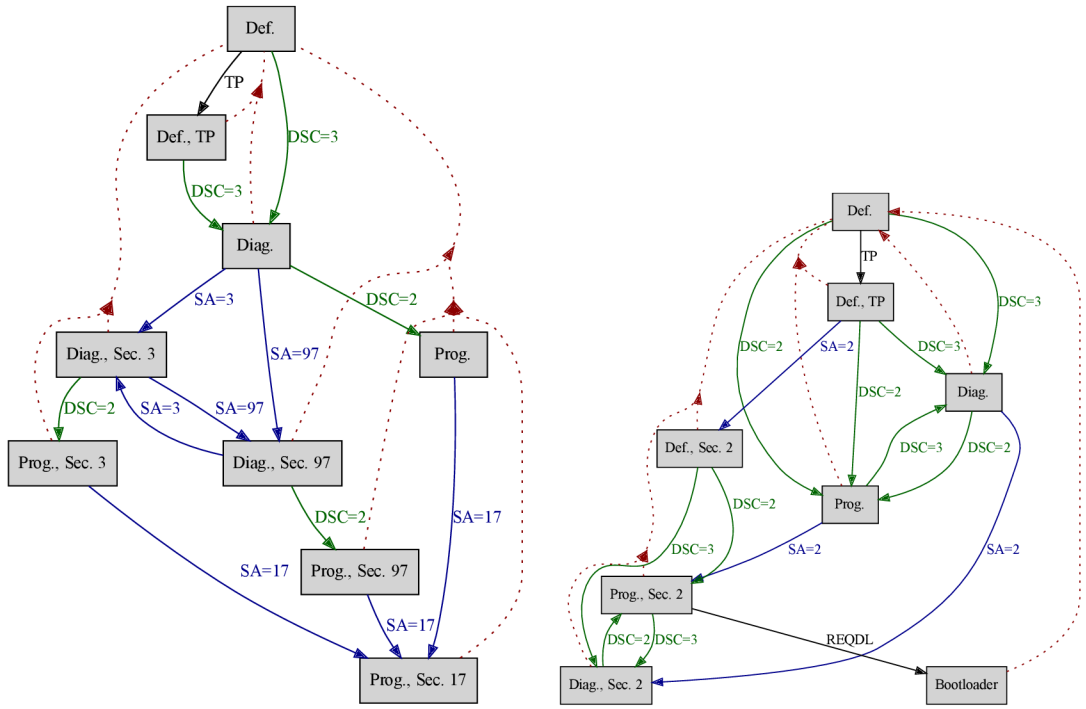

Figure 8.1: ***Left:*** Automatically reverse-engineered system state graph of ECU E1. ***Right:*** Automatically reverse-engineered system state graph of ECU E7. ***Both:*** Reset through the power cycle is represented by the red dotted lines. Blue lines indicate Security Access (SA) authentication. The green lines show Diagnostic Session Control (DSC) transitions. TP stands for Tester Present and REQDL for Request Download. [O3]

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. [O3]

8.5 Attack Surface Increase

The scan results of ECU E10 in table 8.6 are discussed, and a more general overview of the results of all tested ECUs is provided in table 8.7 to demonstrate that the proposed scan algorithm can automatically explore an increased attack surface on

Table 8.4: 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. [O3]

	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

Table 8.5: Average response time for negative responses and number of samples. Timings of five different ECUs in the default and the bootloader state. [O3]

	Firmware	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

diagnostic protocols. Figure 8.2 shows all system states of ECU E10 with its possible transitions. For each system state, table 8.6 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 8.6 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 8.7 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. [O3]

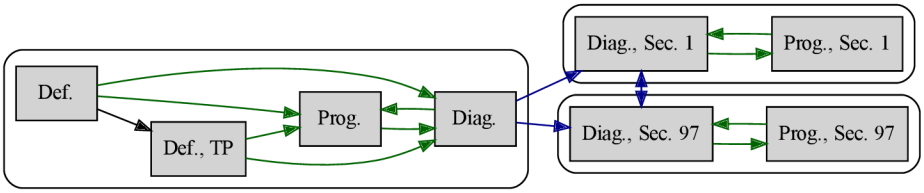

Figure 8.2: Automatically reverse-engineered system state graph for ECU E10 without reset transitions. Colors and abbreviations are identical to figure 8.1. Clusters indicate sub-graphs with unique security access levels. [O3]

Table 8.6: The detailed threat model for ECU E10. The abbreviation rpc stands for rand/pass/crypt. [O3]

State	**buf**	**dos-fl.**	**infoleak**	**int-ov.**	**phys**	**rpc**
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 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, the number of sub-functions that could trigger physical actions (column `phys`) increased system states' injection. [O3]

8.6 Threats over Lifetime

As a final evaluation, the proposed threat model's capabilities for lifetime security analysis will be discussed. ECU E1 supports three different security access levels.

Table 8.7: Overview of identified flaws per ECU. Rows with state s_0 stand for default session, rows with state S describes the combination of all identified system states. The abbreviation rpc stands for rand/pass/crypt. [O3]

ECU	States	buf	dos-fl.	infoleak	int-ov.	phys	rpc	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 8.8: Attack surface metrics protected by individual security access levels for ECU E1. [O3]

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

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 8.8 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 [3]. 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. [O3]

Figure 8.3 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 8.8. 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. [O3]

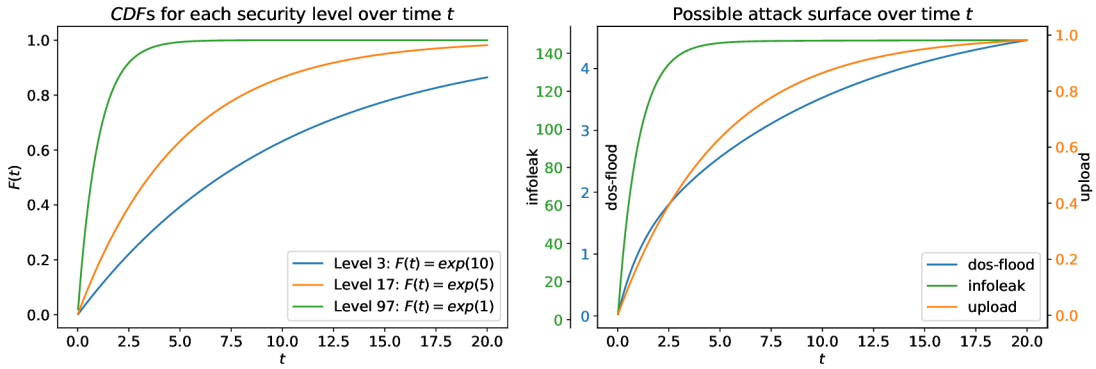

Figure 8.3: ***Left:*** Three CDFs of different exponential distribution functions to model the meantime t until a successful attack of the corresponding security access algorithm for three different security access levels. ***Right:*** Evaluation of the attack surface over time t for measured attack 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 $CDFs$. [O3]

8.7 Summary

This chapter demonstrates automation capabilities for different protocol layers in the automotive diagnostic protocol stack. It was possible to realize automated tools because of comprehensive groundwork by the contribution of automotive protocol implementations to the open-source software framework *Scapy*.

Based on proven security incidents, it was possible to establish a comprehensive risk model for threat estimations of automotive diagnostic protocols. Furthermore, it was possible to prove that ECUs have a different attack surface, depending on their internal system state. Active automata reverse engineering techniques enable the implemented scan algorithms to discover system states during a black-box scan automatically. Through active detection and stimulation of system states, the algorithms can perform a more comprehensive analysis of an ECU's or a vehicle's attack surface. The introduced metric helps security researchers to rate and compare the possible attack surfaces of ECUs and their evolutions over a system's lifetime. Additionally, the risk model can point researchers or penetration testers to safety- and security-critical services. By publishing these tools as open-source software, automated attack surface discovery for automotive diagnostic protocols will speed up,

aiming to lower the attack surface of future vehicles.

The developed scan algorithms can 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. Furthermore, reverse engineered system state machines can deliver valuable input for smart fuzzing approaches of automotive diagnostic protocols.

Chapter 9

Conclusion

This chapter summarizes the main contributions of this thesis, collects open issues, and proposes future work. Since the security of safety-critical systems is a rapidly growing field, it will more and more affect our all daily life's. This thesis builds a foundation for future security research in safety-critical automotive networks based on free and open software tools.

9.1 Open Issues

The following issues in the field of security testing for safety-critical systems, specifically automotive systems, remain.

9.1.1 Proprietary Systems and Security by Obscurity

One, interested in the field of automotive security research, is facing multiple obstacles in the beginning. Entire vehicles for research require a high initial investment, on the other hand, single components are very hard to investigate, without a possibility to observe their normal behavior. Documentation about a system is usually kept secret, and necessary information to understand the internals of a component needs to be reverse-engineered. Furthermore, keeping an automotive system in an operational state, while exploring its internals, is a very challenging task on its own. In recent years, educational systems for automotive security research have become available, and more researchers document their efforts publicly. Unfortunately, no public information exchange on security topics with OEMs or suppliers exists.

Many automotive systems contain custom or obscured hardware components. Part numbers of controllers and processors are customized, or datasheets and software libraries are only available under NDA. These measures increase difficulties for researchers but do not provide any real security enhancements.

Possible solution. The automotive industry needs a change in its security culture, to address upcoming security challenges. The latest vehicles already contain open-source software to provide their functionalities. Similar cultural changes need to take place for security-related topics, and more first-hand information for security research is required.

9.1.2 Custom Implementations of Diagnostic Protocols

Every analyzed ECU contains a unique implementation of the diagnostic protocol. This ECU-specific protocol follows the definitions in the standards, but the custom additions of OEMs are kept secret. More public knowledge about the actual functionalities in automotive systems would allow more comprehensive security investigations.

Possible solution. Parts of this information can be extracted from the software of repair shop testers. Parser to extract protocol information would be required, to enhance security investigations.

9.2 Future Work

On top of this work, automotive security research can build up in three possible directions:

- *Comprehensive analysis of custom diagnostic protocols.* Parsers for OEM-specific diagnostic protocol description files could extract detailed information for advanced scanners of diagnostic protocols. This information could speed up scans and would lead to a more detailed attack surface and threat analysis.

- *Automated fuzz testing of automotive systems.* Reverse-engineered system state machines would allow more effective smart fuzzing of automotive diagnostic protocols. Additional performance increases could be obtained by proprietary protocol information, as described previously.

- *System-state reverse engineering for testing of state-full automotive software.* State-full software, present in the update mechanisms of secure automotive bootloaders, could use identical system state reverse-engineering techniques for automated and explorative software testing.

9.3 Final Conclusion

This thesis covers a study of published security incidents of automotive systems and in-deep security analysis of four different ECUs. The security analysis follows a defined process for black-box security investigations and a comprehensive vulnerability rating of automotive components. Based on the analysis results, the automation capabilities of the attack-surface "External Interfaces" are leveraged for the creation of an open-source software framework for security testing of automotive systems. On top of this framework, tools for automated and semi-automated testing in automotive networks were created and published. Finally, a novel attack surface model for automotive diagnostic protocols is defined, including an automated scanner with system state reverse engineering capabilities.

9.3.1 Major Contributions

The main contributions of this thesis are:

- an **investigation process** for black-box security analysis of automotive components,
- an **open-source software framework** for manual and semi-automated security testing in automotive networks, and fully-automated security testing of automotive diagnostic protocols,
- a novel **attack-surface model** for automotive diagnostic protocols.

9.3.2 Review of Aims of the Ph.D. Thesis

The following tasks are taken from the author's Ph.D. thesis exposé (Weiss, 2020):

- Evaluations of open-source software projects suitable for security tests in safety-critical networks.

This goal was accomplished. Open- and closed-source media access solutions were analyzed [O1], and existing software frameworks, suitable for security testing in automotive networks, were evaluated in section 6.1.

- Analysis of security flaws in current automotive systems.
- Evaluation of automation capabilities for security tests in current automotive systems.

These two goals were covered in Part II of this thesis and therefore completed. A novel investigation process for automotive systems was proposed and applied to four different ECUs. Results were presented at the Troopers19 conference and to the Automotive Security Research Group [P3, P4]. Next to this manual investigation, published research on security incidents of automotive systems was studied and analyzed. Based on the defined process, automation capabilities for individual attack surfaces in automotive systems were evaluated.

- Implementation of software tools to support the penetration testing process in safety-critical networks.

This goal was achieved by a comprehensive extension of the open-source software tool "Scapy" in three stages. First, all proprietary automotive protocols and communication technologies were implemented and presented to the security research community [P1, P2]. Second, a tool and methodology for the automated detection of possible attack targets in vehicular networks were published [O2]. Third, a novel threat model, a scan algorithm with system-state reverse-engineering, and an open-source software tool were published [O3].

- General consideration of what kind of security flaws can be detected through automated testing.

This goal was accomplished for automotive diagnostic protocols. In Chapter 7 of this thesis, a novel attack surface model introduces a mapping of security flaw types to individual functions of automotive diagnostic protocols. This mapping describes what kind of security flaws can be detected in individual features of automotive diagnostic protocols.

List of Authors Publications

[O1] Enrico Pozzobon, Nils Weiss, Sebastian Renner, and Rudolf Hackenberg. A Survey on Media Access Solutions for CAN Penetration Testing. In *ACM Computer Science in Cars Symposium (CSCS)*, CSCS '18, New York, NY, USA, 09 2018. Association for Computing Machinery.

[O2] Nils Weiss, Sebastian Renner, Jürgen Mottok, and Václav Matoušek. Transport layer scanning for attack surface detection in vehicular networks. In *Computer Science in Cars Symposium*, page 1–8. ACM, Dec 2020.

[O3] Nils Weiss, Sebastian Renner, Jürgen Mottok, and Václav Matoušek. Automated threat evaluation of automotive diagnostic protocols. In *Proceedings of the Embedded Security in Cars Workshop (ESCAR)*, page 1–16, May 2021.

[O4] Nils Weiss, Sebastian Renner, Enrico Pozzobon, and Rudolf Hackenberg. Extending Vehicle Attack Surface Through Smart Devices. In *The Eleventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)*, Rome, Italy, 09 2017.

[O5] Nils Weiss, Markus Schrötter, and Rudolf Hackenberg. On Threat Analysis and Risk Estimation of Automotive Ransomware. In *ACM Computer Science in Cars Symposium*, CSCS '19, New York, NY, USA, 2019. Association for Computing Machinery.

List of Authors Presentations

[P1] Nils Weiss. Automotive Penetration Testing with Open Source Software. Open Source Specialist Group - A Specialist Group of the British Computer Society, 2021.

[P2] Nils Weiss and Enrico Pozzobon. Automotive Penetration Testing with Scapy. IT Security Conference Troopers19, 2019.

[P3] Nils Weiss and Enrico Pozzobon. Iot Backdoors in Cars. IT Security Conference Troopers19, 2019.

[P4] Nils Weiss and Enrico Pozzobon. Reverse Engineering and Weaponizing OBD Dongles. In *Automotive Security Research Group, Meeting 22*, Stuttgart, Germany, 2019.

[P5] Nils Weiss and Enrico Pozzobon. From Blackbox to Automotive Ransomware. DEF CON SAFE MODE Hacking Conference. Virtual Conference, 2020.

Bibliography

[1] Linktype_can_socketcan - packet structure, 2020. https://www.tcpdump.org/linktypes/LINKTYPE_CAN_SOCKETCAN.html (accessed 2021-04-14).

[2] U.S. National Security Agency. Defense in Depth - a practical strategy for achieving Information Assurance in today's highly networked environments. https://apps.nsa.gov/iaarchive/library/ia-guidance/archive/defense-in-depth.cfm (accessed 2021-04-14).

[3] Florian Arnold, Holger Hermanns, Reza Pulungan, and Mariëlle Stoelinga. *Time-Dependent Analysis of Attacks*, volume 8414 of *Lecture Notes in Computer Science*, page 285–305. Springer Berlin Heidelberg, 2014.

[4] AUTOSAR. *Specification of Secure Onboard Communication*, 2020. https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_SecureOnboardCommunication.pdf (accessed 2021-04-14).

[5] Computest Services B.V. *The Connected Car - Ways to get unauthorized access and potential implications*, Apr 2018. https://www.computest.nl/wp-content/uploads/2018/04/connected-car-rapport.pdf (accessed 2021-04-14).

[6] Zhiqiang Cai, Aohui Wang, and Wenkai Zhang. 0-days & Mitigations: Roadways to Exploit and Secure Connected BMW Cars. In *BlackHat USA*, pages 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).

[7] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 1–6, USA, 2011. USENIX Association.

[8] Kyong-Tak Cho and Kang G. Shin. Error Handling of In-Vehicle Networks Makes Them Vulnerable. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1044–1055, New York, NY, USA, 2016. Association for Computing Machinery.

[9] Gerald Combs. Wireshark is the world's foremost and widely-used network protocol analyzer. `https://www.wireshark.org/` (accessed 2021-04-14).

[10] European Commission. *Intelligent transport systems - The interoperable EU-wide eCall*. `https://ec.europa.eu/transport/themes/its/road/action_plan/ecall_en` (accessed 2021-04-14).

[11] Jürgen Dürrwang, Johannes Braun, Marcel Rumez, Reiner Kriesten, and Alexander Pretschner. Enhancement of Automotive Penetration Testing with Threat Analyses Results. *SAE International Journal of Transportation Cybersecurity and Privacy*, 1(2):91–112, Nov 2018.

[12] flashrom.org. flashrom is a utility for identifying, reading, writing, verifying and erasing flash chips. `https://flashrom.org/` (accessed 2021-04-14).

[13] Association for Standardization of Automation and Measuring Systems. The Universal Measurement and Calibration Protocol Family. Standard ASAM MCD-1 XCP, Association for Standardization of Automation and Measuring Systems, Germany, DE, 2003. `https://www.asam.net/standards/detail/mcd-1-xcp/` (accessed 2021-04-14).

[14] Vector Informatik GmbH. Xcp – the standard protocol for ecu development, 2020. `https://assets.vector.com/cms/content/application-areas/ecu-calibration/xcp/XCP_ReferenceBook_V3.0_EN.pdf` (accessed 2021-04-14).

[15] General Motors Worldwide (GMW). General Motors Local Area Network Enhanced Diagnostic Test Mode Specification. Standard GMW3110, General Motors Worldwide (GMW), 2018.

[16] Oliver Hartkopp. *Linux Kernel Module for ISO 15765-2:2016 CAN transport protocol*, 2020. `https://github.com/hartkopp/can-isotp` (accessed 2021-04-14).

[17] Oliver Hartkopp. *Readme file for the Controller Area Network Protocol Family (aka SocketCAN)*, 2020. `https://www.kernel.org/doc/Documentation/networking/can.txt` (accessed 2021-04-14).

[18] Oliver Hartkopp. *SocketCAN userspace utilities and tools*, 2020. `https://github.com/linux-can/can-utils` (accessed 2021-04-14).

[19] IEEE. *IEEE Standard 802.3-2018 - Standard for Ethernet*, 2018.

[20] ISO Central Secretary. Road vehicles – End-of-life activation of on-board pyrotechnic devices – Part 3: Tool requirements. Standard ISO 26021-3:2009, International Organization for Standardization, Geneva, CH, 2009.

[21] ISO Central Secretary. Road vehicles – Implementation of World-Wide Harmonized On-Board Diagnostics (WWH-OBD) communication requirements – Part 3: Common message dictionary. Standard ISO 27145-3:2012, International Organization for Standardization, Geneva, CH, 2012.

[22] 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.

[23] 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.

[24] ISO Central Secretary. Road vehicles – Communication between vehicle and external equipment for emissions-related diagnostics – Part 5: Emissions-related diagnostic services. Standard ISO 15031-5:2015, International Organization for Standardization, Geneva, CH, 2015.

[25] ISO Central Secretary. Road vehicles – Controller area network (CAN) — Part 1: Data link layer and physical signalling. Standard ISO 11898-1:2015, International Organization for Standardization, Geneva, CH, 2015.

[26] 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.

[27] 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.

[28] Wayne Jansen. Directions in security metrics research. Technical Report NIST IR 7564, National Institute of Standards and Technology, 2009. `https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7564.pdf` (accessed 2021-04-14).

[29] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental Security Analysis of a Modern Automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462, May 2010.

[30] Sekar Kulandaivel, Tushar Goyal, Arnav Kumar Agrawal, and Vyas Sekar. CANvas: Fast and Inexpensive Automotive Network Mapping. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 389–405, Santa Clara, CA, August 2019. USENIX Association.

[31] Tencent Keen Security Lab. *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).

[32] Tencent Keen Security Lab. *New Vehicle Security Research by KeenLab: Experimental Security Assessment of BMW Cars*, 2020. `https://keenlab.tencent.com/en/2018/05/22/New-CarHacking-Research-by-KeenLab-Experimental-Security-Assessment-of-BMW-Cars/` (accessed 2021-04-14).

[33] Pico Technology Ltd. *Complete CAN data frame structure*, 2020. `https://www.picotech.com/images/uploads/library/topics/_med/CAN-full-frame.jpg` (accessed 2021-04-14).

[34] Gordon Lyon. Nmap: the Network Mapper - Free Security Scanner. `https://nmap.org/`(accessed 2021-04-14).

[35] Peter Mell, Karen Scarfone, and Sasha Romanosky. A Complete Guide to the Common Vulnerability Scoring System Version 2.0. `https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=51198` (accessed 2021-04-14).

[36] Dr. Charlie Miller and Chris Valasek. 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).

[37] Dr. Charlie Miller and Chris Valasek. A Survey of Remote Automotive Attack Surfaces. DEF CON 22 Hacking Conference. Las Vegas, NV: DEF CON, August 2014.

[38] Dr. Charlie Miller and Chris Valasek. Remote Exploitation of an Unaltered Passenger Vehicle. DEF CON 23 Hacking Conference. Las Vegas, NV: DEF CON, August 2015.

[39] Dr. Charlie Miller and Chris Valasek. Advanced can injection techniques for vehicle networks. In *BlackHat USA*, Aug 2016. `http://illmatics.com/can%20message%20injection.pdf` (accessed 2021-04-14).

[40] The MITRE Corporation (MITRE). Vulnerability Type Distributions in CVE – Flaw Terminology. `https://cve.mitre.org/docs/vuln-trends/index.html` (accessed 2021-04-14).

[41] Colin O'Flynn. BAM BAM!! On Reliability of EMFI for in-situ Automotive ECU Attacks. Cryptology ePrint Archive, Report 2020/937, 2020. `https://eprint.iacr.org/2020/937` (accessed 2021-04-14).

[42] Ramiro Pareja and Santiago Cordoba. *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).

[43] Pierre Lalet Philippe Biondi, Guillaume Valadon and Gabriel Potter. *Scapy*, 2018. `http://www.secdev.org/projects/scapy/` (accessed 2021-04-14).

[44] Inc. Rapid7. *metasploit - The world's most used penetration testing framework*, 2020. `https://www.metasploit.com/` (accessed 2021-04-14).

[45] RBEI and ETAS. *BUSMASTER*, 2017. `https://github.com/rbei-etas/busmaster/` (accessed 2021-04-14).

[46] Inc. ReFirm Labs. Binwalk is a fast, easy to use tool for analyzing, reverse engineering, and extracting firmware images. `https://github.com/ReFirmLabs/binwalk` (accessed 2021-04-14).

[47] Christian Sandberg, Kasper Karlsson, Tobias Lans, Mattias Jidhage, Johannes Weschke, and Filip Hesslund. *Caring Caribou*, 2018. `https://github.com/CaringCaribou/caringcaribou` (accessed 2021-04-14).

[48] Ankita Sawanta and Lenina Svb. CAN, FlexRay, MOST versus Ethernet for Vehicular Networks. *International Journal of Innovations & Advancement in Computer Science*, 04 2018.

[49] Philipp Schmied. *CANalyzat0r*, 2018. `https://github.com/schutzwerk/CANalyzat0r` (accessed 2021-04-14).

[50] Yuefeng Du Sen Nie, Ling Liu. *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).

[51] Alexey Sintsov. *CANToolz - framework for black-box CAN network analysis*, 2017. `https://github.com/CANToolz/CANToolz` (accessed 2021-04-14).

[52] Dieter Spaar. *Beemer, Open Thyself! – Security vulnerabilities in BMW's ConnectedDrive*, February 2015. `https://www.heise.de/ct/artikel/Beemer-Open-Thyself-Security-vulnerabilities-in-BMW-s-ConnectedDrive-2540957.html` (accessed 2021-04-14).

[53] Grand Idea Studio. Jtagulator. a tool to assist in identifying on-chip debugging (ocd) and/or programming connections from test points, vias, or component pads on a target piece of hardware. `https://github.com/grandideastudio/jtagulator` (accessed 2021-04-14).

[54] Junko Takahashi, Yosuke Aragane, Toshiyuki Miyazawa, Hitoshi Fuji, Hirofumi Yamashita, Keita Hayakawa, Shintarou Ukai, and Hiroshi Hayakawa. Automotive Attacks and Countermeasures on LIN-Bus. *Journal of Information Processing*, 25:220–228, 02 2017.

[55] Brian Thorne. *python-can*, 2020. `https://github.com/hardbyte/python-can` (accessed 2021-04-14).

[56] Ken Tindell. *CAN Bus Security - Attacks on CAN bus and their mitigations*, 2019. `https://canislabs.com/wp-content/uploads/2020/12/2020-02-14-White-Paper-CAN-Security.pdf` (accessed 2021-04-14).

[57] Jan Van den Herrewegen and Flavio D. Garcia. *Beneath the Bonnet: A Breakdown of Diagnostic Security*, volume 11098 of *Lecture Notes in Computer Science*, page 305–324. Springer International Publishing, 2018.

[58] Awad Younis, Yashwant K. Malaiya, and Indrajit Ray. Assessing vulnerability exploitability risk using software properties. *Software Quality Journal*, 24(1):159–202, Mar 2016.

Appendix A

Identified Vulnerabilities

This appendix lists detailed information of all identified vulnerabilities in the investigated automotive components. All vulnerabilities are documented with the defined vulnerability fact sheet.

A.1 Central Gateway Controller

Table A.1: Summary of vulnerability *V1*

V1	**Unprotected external memory**		
The external EEPROM can be red or manipulated. Data inside this memory is not authenticated nor encrypted.			
Preconditions:	Physical access to the PCB		
Impact:	Low	**Exploitability:**	Low
Rating Explanation:	An attacker needs physical access and code execution can't be obtained.		
Attack Surface:	External Memories (EEPROM)		

Table A.2: Summary of vulnerability *V2*

V2	**Diagnostic message routing**		
All diagnostic communication is forwarded into the vehicle-internal subnetwork. No plausibility checks are applied. No authentication is required.			
Preconditions:	Access to the diagnostic interface		
Impact:	Medium	**Exploitability:**	Medium
Rating Explanation:	Physical network access is required and an attacker gains access to multiple other targets but no priviledges for code execution.		
Attack Surface:	External Interfaces (CAN)		

Table A.3: Summary of vulnerability *V3*

V3	**Development functionalities**		
BMW proprietary UDS commands to access special functionalities for development purposes are available. Some of these services can be used without authentication.			
Preconditions:	Access to the diagnostic interface		
Impact:	Low	**Exploitability:**	Medium
Rating Explanation:	Development functionalities can be accessed with physical access to the diagnostic CAN network. Code execution privileges are not obtained.		
Attack Surface:	External Interfaces (CAN)		

Table A.4: Summary of vulnerability *V4*

V4	**Serial debug interface available**		
A serial interface (UART interface) with debug functionalities can be accessed via a debug header on the PCB. A command line terminal on this interface can be used to run debug and development functions.			
Preconditions:	Physical access to the PCB		
Impact:	Low	**Exploitability:**	Low
Rating Explanation:	Internal information can be gathered, if an attacker has physical access to the target.		
Attack Surface:	Debug Interfaces (UART)		

Table A.5: Summary of vulnerability *V5*

V5	**Insecure MCU internal bootloader**		
A hardware attack on the internal bootloader was demonstrated by Colin O'Flynn [41].			
Preconditions:	Physical access to the PCB		
Impact:	Medium	**Exploitability:**	Low
Rating Explanation:	An attacker can obtain code execution or access all secrets stored in internal memories of this MCU.		
Attack Surface:	Debug Interfaces (UART)		

A.2 Body Domain Controller

Table A.6: Summary of vulnerability *V6*

V6	**Unprotected external memory**		
The external EEPROM can be red or manipulated. Data inside this memory is not authenticated nor encrypted.			
Preconditions:	Physical access to the PCB		
Impact:	Low	**Exploitability:**	Low
Rating Explanation:	Physical access to the target is required. No code execution can be obtained from this vulnerability.		
Attack Surface:	External Memories (EEPROM)		

Table A.7: Summary of vulnerability *V7*

V7	**Insecure Security Access mechanism**		
The GMLAN protocol specification defines a key length of two bytes for the security access service [15, p. 125]. The seed which need to be answered with the correct key is constant. A brute force attack on the security access mechanism is feasible under laboratory conditions. After obtaining security access on this ECU, an attacker can enter the bootloader and execute dangerous functionalities, for example flashing of the ECU.			
Preconditions:	Access to the high-speed CAN network, for example through the vehicles OBD interface.		
Impact:	Medium	**Exploitability:**	Medium
Rating Explanation:	Physical access to the cars internal network is required. The bootloader as further attack surface can be accessed.		
Attack Surface:	External Interfaces (CAN)		

Table A.8: Summary of vulnerability *V8*

V8	**Insecure software update mechanism**		
After obtaining security access, one can access the bootloader functionalities of this ECU. The bootloader is implemented in an insecure way. A bootloader service, called *Transfer Data*, allows the transfer and the execution of arbitrary code to the ECUs RAM [15, p. 161]. No cryptographic signatures are checked by the bootloader.			
Preconditions:	Access to the high-speed CAN network and security access.		
Impact:	High	**Exploitability:**	Low
Rating Explanation:	An attacker with access to the bootloader can obtain remote code execution over the car internal network and therefore trigger physical actions in the car.		
Attack Surface:	Operating System (Bootloader)		

Table A.9: Summary of vulnerability chain *C1*

C1	**Vulnerability Chain: Insecure Security Access mechanism and insecure software update mechanism**		
An attacker obtains remote code execution and access to physical systems through access to the cars internal CAN network, for example through the OBD interface.			
Preconditions:	Access to the vehicle network through OBD		
Impact:	High	**Exploitability:**	Medium
Rating Explanation:	Two vulnerabilities can be joined together to obtain a vulnerability chain with higher exploitability factor and impact factor as the individual vulnerabilities.		
Attack Surface:	External Interfaces (CAN)		
Involved Vulnerabilities:	V7, V8		

A.3 Telematics Control Unit

Table A.10: Summary of vulnerability *V9*

V9	**Unprotected external memory**		
The external flash memory of the main processor can be red or manipulated. Data inside this memory is not authenticated nor encrypted. An attacker can read or modify password hashes for the passwords in the QNX operation system. Cryptographic data to secure the back-end connections can be red.			
Preconditions:	Physical access to the PCB		
Impact:	Medium	**Exploitability:**	Low
Rating Explanation:	An attacker can gain local code execution through manual data tampering. This attack requires physical modifications on the PCB.		
Attack Surface:	External Memories (Flash)		

Table A.11: Summary of vulnerability *V10*

V10	**Unprotected JTAG interface**		
The main processor can be debugged through the external JTAG interface. The firmware inside the flash memory can be red and modified. Run-time data in the RAM memory can be red and manipulated.			
Preconditions:	Physical access to the PCB		
Impact:	Medium	**Exploitability:**	Low
Rating Explanation:	Full control over the processor can be achieved from an attacker. This requires physical access an modifications on the PCB.		
Attack Surface:	Debug Interfaces (JTAG)		

Table A.12: Summary of vulnerability *V11*

V11	**External Debug-Port**		
The i.MX6 main processor can be booted over USB-OTG. This USB port is available on an external connector. An attacker can use this debug port to compromise the entire ECU.			
Preconditions:	Physical access to the ECU		
Impact:	Medium	**Exploitability:**	Low
Rating Explanation:	This vulnerability gives an attacker identical privileges as the previous two vulnerabilities, but no physical modifications to the PCB are required.		
Attack Surface:	Debug Interfaces (USB-OTG)		

A.4 Airbag Control Unit

Table A.13: Summary of vulnerability *V12*

V12	**Weak Security Access algorithm**		
The security access algorithm of this ECU follows the examples in ISO 26021 [20].			
Preconditions:	Access to SWCAN		
Impact:	Medium	**Exploitability:**	Medium
Rating Explanation:	A weak security access algorithm allows an attacker to access a new attack surface over the vehicles network.		
Attack Surface:	External Interfaces (SWCAN)		

Table A.14: Summary of vulnerability *V13*

V13	**RCE vulnerability**		
After a successful security access authentication, the bootloader of this ECU is accessible. This bootloader contains a vulnerability which allows an attacker to execute arbitrary code.			
Preconditions:	Access to the ECUs bootloader through security access		
Impact:	High	**Exploitability:**	Low
Rating Explanation:	If an attacker gains access to the bootloader, a remote code execution vulnerability can be triggered. This gives an attacker the possibility to cause physical actions from a remote interface.		
Attack Surface:	Operating System (Bootloader)		

Table A.15: Summary of vulnerability chain *C2*

C2	**Vulnerability Chain: Weak Security Access algorithm and RCE vulnerability**		
An attacker can trigger a RCE vulnerability in the airbag ECU bootloader from the vehicles OBD interface. This vulnerability gives an attacker control over physical systems.			
Preconditions:	Access to the vehicle network		
Impact:	High	**Exploitability:**	Medium
Rating Explanation:	The combination of these two vulnerabilities increase the impact and the exploitability compared to both individual vulnerabilities.		
Attack Surface:	External Interfaces (SWCAN)		
Involved Vulnerabilities:	V12, V13		

A.5 Dieter Spaar: Beemer, Open Thyself!

Table A.16: Summary of vulnerability *V14*

V14	**Shared cryptographic secrets in TCU**		
Shared secrets allow an attacker to authenticate itself as legitimate communication partner to a TCU.			
Preconditions:	Physical access to one TCU.		
Impact:	Low	**Exploitability:**	Low
Rating Explanation:	The gained information from this vulnerability allow an attacker to prepare further attacks. Physical modification to the PCB were necessary to extract these shared secrets.		
Attack Surface:	External Memories (Flash)		

Table A.17: Summary of vulnerability *V15*

V15	**Information leaks in NGTP**		
Leaked information allow preparations for remote attacks on the communication protocol NGTP.			
Preconditions:	Vehicle needs to connect to a malicious BTS.		
Impact:	Low	**Exploitability:**	High
Rating Explanation:	An attacker gains necessary information to craft malicious commands for a targeted vehicle.		
Attack Surface:	Wireless Interfaces (Mobile Data Connection)		

Table A.18: Summary of vulnerability chain *C3*

C3	**Vulnerability Chain: Shared secrets and information leaks in NGTP**		
Leaked information combined with cryptographic secrets for authentication can be used to trigger physical actions on the car.			
Preconditions:	Vehicle needs to connect to a malicious BTS.		
Impact:	High	**Exploitability:**	High
Rating Explanation:	An attacker can authenticate itself onto a arbitrary car and execute legitimate physical actions from a wireless connection.		
Attack Surface:	Wireless Interfaces (Mobile Data Connection)		
Involved Vulnerabilities:	V14, V15		

A.6 Miller & Valasek: Remote Exploitation of an Unaltered Passenger Vehicle

Table A.19: Summary of vulnerability *V16*

V16	**Exposed services on remote interface**		
Various open ports for connections from the Internet existed on the vehicles MMU.			
Preconditions:	Internet connection and IP-address of the vehicle.		
Impact:	Medium	**Exploitability:**	High
Rating Explanation:	Attackers can execute arbitrary code on the MMU over an Internet connection. This exposes the CAN controller of the attacked ECU as additional attack surface.		
Attack Surface:	Wireless Interfaces (Mobile Data Connection)		

Table A.20: Summary of vulnerability *V17*

V17	**Insecure software update mechanisms**		
Exposed services allow software updates to the MMU of the vehicle. No authentication checks were performed on software updates. A custom firmware allows arbitrary read and write access to the vehicles CAN network.			
Preconditions:	Local code execution on the MMU.		
Impact:	High	**Exploitability:**	Low
Rating Explanation:	A custom firmware update allows read and write access to the vehicles internal CAN network. Through this connection, physical actions on the vehicles actuators can be triggered from the MMU main processor.		
Attack Surface:	On-Board Interfaces (SPI)		

Table A.21: Summary of vulnerability chain *C4*

C4	**Vulnerability Chain: Exposed services and insecure software update mechanisms**		
The combination of these two vulnerabilities allows attackers a remote exploitation of the entire vehicle.			
Preconditions:	Internet connection to the vehicle		
Impact:	High	**Exploitability:**	High
Rating Explanation:	An attacker can execute arbitrary code and trigger physical actions on the vehicles actuator from an Internet connection.		
Attack Surface:	Wireless Interfaces (Mobile Data Connection)		
Involved Vulnerabilities:	V16, V17		

A.7 Nie et al.: Free-Fall - Hacking Tesla from Wireless to CAN Bus

Table A.22: Summary of vulnerability *V18*

V18	**Browser exploit**		
Once a vehicle connects to a malicious WLAN access point, a browser exploit can be triggered.			
Preconditions:	Established WLAN connection to malicious WLAN access point.		
Impact:	Medium	**Exploitability:**	High
Rating Explanation:	A browser exploit could be triggered without user interaction. This exploit gives privileges for code execution. An attacker has access to a greater attack surface.		
Attack Surface:	Wireless Interfaces (WLAN)		

Table A.23: Summary of vulnerability *V19*

V19	**Linux kernel exploit**		
This vulnerability allows attackers to gain root privileges on a target system. This exploit is triggered after a successful browser exploit.			
Preconditions:	Unprivileged code execution		
Impact:	Medium	**Exploitability:**	Low
Rating Explanation:	An attacker needs unprivileged code execution in order to escalate privileges through this vulnerability. Privileged code execution exposes a greater attack surface on the MMU.		
Attack Surface:	Operating System (Linux)		

Table A.24: Summary of vulnerability *V20*

V20	**Unauthenticated software update mechanisms**		
A malicious firmware update of the vehicles internal gateway controller allows arbitrary read and write access to the vehicles CAN network.			
Preconditions:	Privileged code execution on the MMU		
Impact:	High	**Exploitability:**	Low
Rating Explanation:	An attacker with root privileges on the MMU can trigger a software update of a malicious firmware to the MMUs network processor. An attacker obtains arbitrary read and write access to the vehicles CAN bus.		
Attack Surface:	On-Board Interfaces (unknown)		

Table A.25: Summary of vulnerability *V21*

V21	**Insecure UDS protocol**		
In-proper security mechanism on the UDS protocol allows access to dangerous services on various ECUs in the vehicle.			
Preconditions:	Read and write access to the vehicle-internal CAN network.		
Impact:	High	**Exploitability:**	Medium
Rating Explanation:	Once an attacker has access to the vehicles CAN network, he can abuse the UDS protocol to trigger physical actions or reprogram further ECUs in the vehicle.		
Attack Surface:	External Interfaces (CAN)		

Table A.26: Summary of vulnerability chain *C5*

C5	**Vulnerability Chain: Browser exploit, Kernel exploit and unauthenticated software updates**		
This vulnerability chain, consisting of three different vulnerabilities gives an attacker full remote access to the vehicles intern CAN network.			
Preconditions:	Connection to a malicious WLAN access point.		
Impact:	High	**Exploitability:**	High
Rating Explanation:	An attacker can trigger physical actions through a wireless connection.		
Attack Surface:	Wireless Interfaces (WLAN)		
Involved Vulnerabilities:	V18, V19, V20, V21		

A.8 Cai et al.: 0-days & Mitigations - Roadways to Exploit and Secure Connected BMW Cars

Table A.27: Summary of vulnerability *V22*

V22	**Browser exploit**		
Once the web browser accesses a malicious website, a browser exploit can be triggered.			
Preconditions:	Access of a website by the cars web browser.		
Impact:	Medium	**Exploitability:**	Medium
Rating Explanation:	This exploit requires user interaction to load a malicious website. An attacker gains code execution privileges on the MMU.		
Attack Surface:	Wireless Interfaces (Mobile Data Connection)		

Table A.28: Summary of vulnerability *V23*

V23	**TOCTOU attack**		
This vulnerability allows attackers to send arbitrary UDS messages onto the vehicles internal networks.			
Preconditions:	Code execution.		
Impact:	Medium	**Exploitability:**	Low
Rating Explanation:	An attacker is able to send UDS messages to all ECUs in the same subnetwork. In this special case, the impact is rated medium, since the vehicle uses a CGW to separate networks. An attacker can not access the entire vehicle network through this vulnerability. ECUs that control CPSs are located in different subnetworks.		
Attack Surface:	Operating System (QNX)		

Table A.29: Summary of vulnerability *V24*

V24	**Insecure UDS message routing**		
A logic error in the vehicle-internal gateway allows attackers to send UDS commands from one subnetwork to any other subnetwork.			
Preconditions:	Access to one of the vehicles subnetworks.		
Impact:	High	**Exploitability:**	Medium
Rating Explanation:	This vulnerability allows the attackers to access ECUs that control CPS. Physical actions can be triggered through this vulnerability.		
Attack Surface:	External Interfaces (CAN)		

Table A.30: Summary of vulnerability *V25*

V25	**Buffer overflow in the NGTP protocol**		
Through a provisioning feature of the NGTP protocol, a buffer overflow could be triggered. This allows arbitrary code execution on the TCU of the vehicle, which involves access to the vehicles internal subnetwork on which the TCU is connected to.			
Preconditions:	GSM or Enhanced Data Rates for GSM Evolution (EDGE) connection to a malicious BTS.		
Impact:	Medium	**Exploitability:**	High
Rating Explanation:	This vulnerability has a medium impact, since the TCU of this vehicle is in the same subnetwork as the MMU. Access to ECUs with control over CPSs is not granted by this vulnerability.		
Attack Surface:	Wireless Interfaces (Mobile Data Connection)		

Table A.31: Summary of vulnerability chain *C6*

C6	**Vulnerability Chain: Browser exploit, TOCTOU attack and insecure UDS message routing**		
This vulnerability chain allows remote exploitation of the entire vehicle.			
Preconditions:	Access of a malicious website.		
Impact:	High	**Exploitability:**	Medium
Rating Explanation:	The victim of this attack has to trigger an exploit by accessing a malicious website.		
Attack Surface:	Wireless Interfaces (Mobile Data Connection)		
Involved Vulnerabilities:	V22, V23, V24		

Table A.32: Summary of vulnerability chain *C7*

C7	**Vulnerability Chain: NGTP buffer overflow and insecure UDS message routing**		
The buffer overflow vulnerability combined with the insecure UDS message routing vulnerability allow the exploitation of the entire vehicle.			
Preconditions:	GSM/EDGE connection to a malicious BTS.		
Impact:	High	**Exploitability:**	High
Rating Explanation:	This vulnerability chain doesn't require user interaction. An attacker can compromise the entire vehicle and gains access to CPSs.		
Attack Surface:	Wireless Interfaces (Mobile Data Connection)		
Involved Vulnerabilities:	V24, V25		

A.9 Computest: The Connected Car - Ways to get unauthorized access and potential implications

Table A.33: Summary of vulnerability *V26*

V26	**Open ports on the vehicles network interfaces**		
Open ports expose services of the ECUs operating system to the Internet.			
Preconditions:	Internet connection		
Impact:	Low	**Exploitability:**	High
Rating Explanation:	Attackers can remotely access services of the ECUs operating system.		
Attack Surface:	Wireless Interfaces (WLAN or mobile data connection)		

Table A.34: Summary of vulnerability *V27*

V27	**QNX vulnerability**		
A vulnerable service on an open port allows attackers to trigger a vulnerability of the QNX operating system.			
Preconditions:	Access to vulnerable service.		
Impact:	Medium	**Exploitability:**	Low
Rating Explanation:	An attacker can use this vulnerability to obtain code execution. The vehicle network can't be accessed from the exploitable MMU.		
Attack Surface:	Operating System (QNX)		

Table A.35: Summary of vulnerability *V28*

V28	**Insecure software update mechanisms**		
An insecure software update mechanism of the MMU allows privilege escalation to a second processor in the MMU. A custom firmware allows arbitrary read and write access to the vehicles CAN network.			
Preconditions:	Code execution on the MMU processor.		
Impact:	Medium	**Exploitability:**	Low
Rating Explanation:	An attacker can escalate his privileges to obtain access to a CAN subnetwork for multimedia ECUs. Access to ECUs with control over actuators is not possible from this vulnerability.		
Attack Surface:	On-Board Interfaces (unknown)		

Table A.36: Summary of vulnerability chain *C8*

C8	**Vulnerability Chain: Open ports, QNX vulnerability and insecure software updates**		
The combination of these vulnerabilities allow attackers a remote exploitation with access to the vehicles CAN subnetwork for multimedia ECUs			
Preconditions:	Internet connection		
Impact:	Medium	**Exploitability:**	High
Rating Explanation:	This vulnerability chain misses one more vulnerability to gain remote access to ECUs with control over CPSs.		
Attack Surface:	Wireless Interfaces (WLAN or Mobile Data Connection)		
Involved Vulnerabilities:	V26, V27, V28		

Appendix B

Contributions to Scapy

Table B.1: Summary of contributions to the *Scapy* project in order to form a versatile open-source software framework for automotive penetration testing. On each contribution, a note identifier specifies the author's involvement. 1: The author implemented this contribution; 2: This contribution was implemented in cooperation with another contributor; 3: The author supported the implementation of this contribution; 4: The author has extended an existing implementation.

CAN - Media Access Layer
`CAN` Packet[4] Added support for handling of CAN packets received from Linux SocketCAN sockets.
`NativeCANSocket`[1] Standardized Scapy SuperSocket object for communication over a Linux SocketCAN socket.
`PythonCANSocket`[1] Standardized Scapy SuperSocket object for communication over any python-can Bus object. This SuperSocket adds internal message queues to mimic the behavior of Linux SocketCAN sockets for python-can Bus objects.
`SignalFields` and `SignalPacket`[1] A new class of Scapy Fields to support Data Base CAN (DBC) signal packets for CAN networks. `SignalPackets` and `SignalFields` allow straightforward translation of DBC packet definitions to Scapy Packet classes.

`CandumpReader`[1]
: A helper class for parsing and interpretation of candump log-files into Scapy Packets.

ISO-TP - Transportation Layer

`ISOTP` Packet[1]
: A special Scapy Packet class for ISO-TP messages. This class allows defragmentation of CAN frames into an `ISOTP` message and fragmentation of an `ISOTP` message into a list of CAN frames.

`ISOTPHeader` and `ISOTP_{SF, FF, CF, FC}` Packets[1]
: Special Scapy Packet classes to dissect and build individual ISO-TP frames for CAN communication.

`ISOTPMessageBuilder`[3]
: A helper class for defragmentation of CAN messages to ISO-TP messages.

`ISOTPSoftSocket`[2]
: A special Scapy SuperSocket class for ISO-TP communication over a `NativeCANSocket` or a `PythonCANSocket`. All underlying communication handling is done in Python.

`ISOTPNativeSocket`[1]
: A Scapy SuperSocket for interaction with the ISO-TP Linux kernel module. This allows ISO-TP communication over Linux ISO-TP sockets from Python. All underlying communication handling is done in the Linux kernel.

`ISOTPScan`[2]
: An ISO-TP endpoint scanner for CANs. This utility automatically identifies possible endpoints and all communication parameters necessary to establish an ISO-TP communication.

HSFZ - Transportation Layer

`HSFZ` Packet[1]
: A special Scapy Packet class to support dissection and building of packets for BMWs HSFZ protocol.

`HSFZSocket`[1]
: A pre-configured Scapy StreamSocket for HSFZ communication over TCP.

`ISOTP_HSFZSocket`[1]
: A transparent `HSFZSocket` to allow application layer communication with an ECU, based on a logical address. All necessary wrapping of messages to send an application layer packet over HSFZ to an ECU is handled transparently.

DoIP - Transportation Layer

`DoIP` Packet[1]
: A special Scapy Packet class to support dissection and building of DoIP packets.

`DoIPSocket`[1]
: A pre-configured Scapy StreamSocket for DoIP communication over TCP.

`UDS_DoIPSocket`[1]
: A transparent `DoIPSocket` to allow application layer communication with an ECU, based on a logical address. All necessary wrapping of messages to send an application layer packet over DoIP to an ECU is handled transparently.

Diagnostic Protocols - Application Layer

`UDS`[1]
: Implementation of Scapy Packet classes for dissection and building of UDS packets. UDS packets are a derivation class of `ISOTP` packets.

`UDS_Scanner`[1]
: Implementation of a scanner utility specifically for attack-surface exploration of UDS implementations.

`GMLAN`[1]
: Implementation of Scapy Packet classes for dissection and building of GMLAN packets. `GMLAN` packets are a derivation class of `ISOTP` packets.

`GMLAN` utilities[2]
: A set of helper functions for easy execution of complex GMLAN functions on ECUs.

`GMLAN_Scanner`[1]
: Implementation of a scanner utility specifically for attack-surface exploration of GMLAN implementations.

`OBD`[2]
: Implementation of Scapy Packet classes for dissection and building of OBD packets. `OBD` packets are a derivation class of `ISOTP` packets.

`OBD_Scanner`[2]
: A scanner utility to automatically gather all available OBD information from an ECU.

`CCP`[1]
: Implementation of Scapy Packet classes for dissection and building of CCP packets.

`XCP`[3]
: Implementation of Scapy Packet classes for dissection and building of XCP packets.

`XCPOnCANScanner`[3]
: A scanner utility to identify `XCP` endpoints in CANs.

Communication Protocols - Application Layer

`SOME/IP`[3]
: Implementation of Scapy Packet classes for dissection and building of `SOME/IP` packets.

Application Layer Utilities

`Ecu` and `EcuSession`[1]
: Helper object to analyze diagnostic layer communication.

`EcuAnsweringMachine`[1]
: Helper object to simulate an ECU.

Appendix C

UDS / GMLAN Service Request Identifiers

Table C.1: List of hexadecimal service identifiers and according service names of UDS and GMLAN

UDS	UDS Service Name	GMLAN	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