Jürgen Wurzinger, BSc

Data Security in Vehicular Control Units and Communication Systems

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Development and Business Management

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Ph.D. Roderick Bloem

Institute of Applied Information Processing and Communications
Head: O.Univ.-Prof. Dipl.-Ing. Dr.techn. Reinhard Posch

Second Supervisor
Dipl.-Ing. Daniel Hein

Graz, April 2016

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis dissertation.

Date

Signature

Abstract

With the introduction of sophisticated, networked entertainment and safety features in modern cars, like for instance Vehicle-To-Everything technologies, a large number of new attack surfaces evolved. In the last few years many researchers focused on the exploitation of this interconnection of modern cars. They showed that almost all cars are vulnerable to possible hacking attacks. This fact highlights the demand that besides securing the interfaces also the internal vehicular networks need protection.

Most research in this domain focuses on revealing new ways of hacking a car or on the development of possible countermeasures. Little is done to develop suitable security testing environments to complement existing safety and functional tests. However, as research showed those traditional tests cannot satisfy security needs of modern cars. With this thesis we want to introduce an application called CAN Communication Tester (CAN-CT) which addresses this problem for the Controller Area Network.

The CAN-CT is an application that makes use of known hacking attacks and injects, replays and invalidates messages systematically. We show that we can successfully spoof messages, suppress all communication on the bus and bring electronic control units in error states. The CAN-CT is capable of learning from the traffic on the bus and can execute targeted attacks. By attacking an electronic control unit the same way a hacker would, we are able to examine the implementation of protection mechanisms in an intuitive yet effective way. Furthermore this approach allows us to test the proper working of possible attack detection techniques as well.

Using Hardware in the Loop Systems we tested the CAN-CT with actual state of the art electronic control units. We revealed vulnerabilities like lacks in the plausibility checking of messages or completely omitted examinations of the frame structure. Those flaws opened up serious threats to hackers. Exploiting those vulnerabilities we showed that we were able to take over message IDs owned by another electronic control unit and hence achieve targeted manipulations.

Revealing weaknesses in real-world electronic control units allowed us to demonstrate the applicability and the impact of applications like the CAN-CT for automotive systems. Our results highlighted the need for security tests to complement traditional testing environments.

Kurzfassung

Durch die Einführung komplexer, vernetzter Unterhaltungs- und Sicherheitsfunktionen in modernen Autos, wie beispielsweise Vehicle-To-Everything Technologien, entstand eine Vielzahl neuer Angriffsvektoren. In den letzten Jahren stand der Fokus vieler Wissenschaftler dieses Bereichs auf der Ausnutzung von Schwachstellen in der Vernetzung moderner Fahrzeuge. Es wurde gezeigt, dass jedes untersuchte Auto anfällig gegenüber möglichen Hacking-Angriffen ist. Dieser Umstand unterstreicht die Forderung, neben der Sicherung der Schnittstellen mit der Außenwelt auch die internen Fahrzeugnetzwerke zu schützen.

Der Großteil aktueller Forschungen in diesem Bereich beschäftigt sich mit der Aufdeckung neuer Möglichkeiten, ein Fahrzeug zu hacken und geeignete Gegenmaßnahmen dafür zu finden. Im Gegensatz dazu mangelt es jedoch an der Erforschung entsprechender Security-Testumgebungen, um bestehende funktionssicherheits-relevante und funktionale Tests zu ergänzen. Wie Forschungen jedoch gezeigt haben, genügen diese traditionellen Testumgebungen den wachsenden Security-Anforderungen nicht mehr. Mit dieser Arbeit stellen wir eine Applikation, genannt CAN Communication Tester (CAN-CT), vor, die genau dieses Problem für das Controller Area Network behandelt.

Der CAN-CT ist eine Applikation, die bekannte Hacking-Angriffe verwendet, um Controller Area Network (CAN) Nachrichten gezielt zu manipulieren und eine andere Herkunft der Nachrichten zu imitieren. Wir demonstrieren, dass wir erfolgreich Nachrichten fälschen, jegliche Kommunikation am Bus unterdrücken sowie Steuergeräte in Fehlerzustände bringen können. Der CAN-CT lernt vom Verkehr am Bus und nutzt dieses Wissen, um ge-

zielte Attacken auszuführen. Durch die Art, Steuergeräte auf dieselbe Weise anzugreifen, wie es ein Hacker würde, können wir die Implementierung von Schutzmaßnahmen in einem intuitiven und doch effektiven Weg testen. Weiters erlaubt es uns diese Vorgehensweise, die einwandfreie Funktion möglicher Angriffserkennungstechniken zu überprüfen.

Mit Hilfe von Hardware in the Loop Systemen testeten wir den CAN-CT mit aktuellen Steuergeräte. Wir konnten Schwachstellen aufdecken, beispielsweise bei der Überprüfung der Plausibilitätschecks sowie eine gänzlich unterlassene Überprüfung der Nachrichtenstruktur. Diese Fehler ermöglichen ernsthafte Bedrohungen durch Hacker. Durch die Ausnutzung dieser Schwachstellen konnten wir beispielsweise Nachrichten-IDs von anderen Steuergeräte übernehmen und somit gezielte Manipulationen erreichen.

Mit der Aufdeckung solcher Schwachstellen in modernen Steuergeräte konnten wir die Anwendbarkeit und die Bedeutung von Programmen wie des CAN-CTs aufzeigen. Unsere Ergebnisse unterstreichen somit die Notwendigkeit von Security-Tests als Ergänzung von traditionellen Testumgebungen.

Acknowledgments

First of all I would like thank my supervisors Roderick Bloem and Daniel Hein for their great support. Their advices, reviews and comments guided me through this work and helped me overcome all challenges.

This thesis would not be possible without the support and contribution of the AVL LIST GmbH Graz. I wrote it during my employment in the global research and technology department of this company. Here I would like to explicitly thank Peter Priller for supporting me not only during this work but rather throughout my whole employment at the AVL LIST GmbH. Without his precious assistance this work would not be possible at all. Furthermore his great advices and his efforts in establishing the connections needed for our test setups laid the foundation for this work.

This leads me to the great support of all members of the HiL-Team (PTE/DFM) at AVL LIST GmbH with Michael Kordon, Ales Kolar and Mohan Swaminathan leading the way. Without their help and contribution I would not have been able to test my work in a real-world scenario.

Besides these persons I want to thank my colleagues and friends for supporting and motivating me. Here I would like to mention Markus Nager, who's ideas and advices during the one or the other coffee break helped me to overcome certain challenges in my work.

Finally, I would particularly like to thank my family for their invaluable support. They always encouraged and enabled me to pursue my goals. A special thanks goes to my partner, Lisa Kandlhofer, for her precious support and motivation during everyday life.

Contents

Abstract iii

Kurzfassung v

Acknowledgments vii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Problem . . . 3
- 1.3 Solution . . . 4
  - 1.3.1 Objectives . . . 5
  - 1.3.2 Outlook on Results . . . 5
- 1.4 Outline . . . 6

2 Preliminaries 7
- 2.1 Terminology and Definitions . . . 7
  - 2.1.1 Electronic Control Unit . . . 7
  - 2.1.2 Hardware in the Loop System . . . 8
  - 2.1.3 Attack . . . 8
  - 2.1.4 Test Case . . . 9
  - 2.1.5 Plan of Attack . . . 9
- 2.2 In-Vehicle Communication Systems . . . 9
  - 2.2.1 Overview of Current Automotive Networks . . . 10
  - 2.2.2 Limitations of Current Automotive Networks . . . 11
- 2.3 Controller Area Network . . . 12
  - 2.3.1 History and Standardization . . . 12
  - 2.3.2 Characteristics . . . 13
  - 2.3.3 Physical Layer Attributes . . . 14
  - 2.3.4 Frame Formats . . . 15

2.3.5 Arbitration . . . 18
2.3.6 Error Handling . . . 18
2.3.7 Higher Layer Protocols . . . 20
2.3.8 Current Protection Mechanisms . . . 22
2.4 Car Hacking . . . 23
2.4.1 Attack Vectors and Security Threats . . . 23
2.4.2 Attacks . . . 25

3 Related Work **27**
3.1 Car Hacking Techniques . . . 27
3.2 Attack Detection and Prevention Approaches . . . 29
3.3 Security Testing . . . 30
3.4 Conclusion . . . 32

4 CAN Communication Tester (CAN-CT) **33**
4.1 Overall Concept . . . 33
4.1.1 Underlying Attack Scenario . . . 34
4.2 Software and Hardware . . . 35
4.3 Architecture . . . 35
4.4 Overview of Application . . . 39
4.4.1 CAN Monitor . . . 39
4.4.2 Capturing Tool . . . 40
4.4.3 Attack Features . . . 41
4.4.4 CAN Network Simulator . . . 44
4.5 Attack Templates . . . 46
4.5.1 Replay/Spoofing Attack . . . 46
4.5.2 Bus Off Attack . . . 48
4.5.3 Denial of Service Attack . . . 49
4.6 Observability . . . 50
4.6.1 Log Files . . . 50
4.6.2 Observation via Oscilloscope . . . 51
4.6.3 Debugger/Calibration Software . . . 51
4.6.4 Analysis of other CAN Messages . . . 51
4.6.5 Attack-Based Detection . . . 52
4.6.6 Higher Layer Protocols . . . 53
4.6.7 Conclusion . . . 53
4.7 Extensibility . . . 53

4.8 Limitations and Solutions . . . 54
4.8.1 Hardware Delay . . . 54
4.8.2 Message Receipt . . . 55
4.8.3 Proprietary Higher Layer Protocols . . . 55
4.9 Conclusion . . . 56

5 Results **57**
5.1 Overview of Test Systems . . . 57
5.2 Laboratory Experiments . . . 60
5.2.1 Test Setup . . . 61
5.2.2 Event-Based Replay/Spoofing Attack . . . 64
5.2.3 Time-Based Replay/Spoofing Attack . . . 65
5.2.4 Bus Off Attack . . . 67
5.2.5 Denial of Service Attack . . . 69
5.2.6 Conclusion . . . 70
5.3 Heavy Duty Vehicle . . . 71
5.3.1 Test Setup . . . 72
5.3.2 Message Spoofing . . . 76
5.3.3 Resilience Against Bus Off Attacks . . . 78
5.3.4 Behavior during Denial of Service Attacks . . . 80
5.3.5 Correctness of Time Out Window . . . 81
5.3.6 Compliance with Expected Message Format . . . 83
5.3.7 Plausibility Checks for Illegal Values . . . 85
5.3.8 Conclusion . . . 86
5.4 Passenger Car . . . 86
5.4.1 Test Setup . . . 87
5.4.2 Plausibility Checks . . . 91
5.4.3 Alive Counter . . . 94
5.4.4 Additional Cyclic Redundancy Check . . . 95
5.4.5 Conclusion . . . 96
5.5 Interpretation of Results . . . 97
5.5.1 Lack of "Real" Security Approaches in CAN Communication . . . 98
5.5.2 Universal Validity . . . 98
5.6 Conclusion . . . 99

6 Outlook and Conclusion **101**

Bibliography **105**

List of Figures

2.1 CAN signal interference filtering according to ISO 11898-2 (Mischo et al., 2015) 14
2.2 CAN 2.0a Data Frame (Mayer, 2006) 17
2.3 Depiction of the arbitration process between two CAN nodes (Mayer, 2006) 19
2.4 CAN error handling state machine 21
2.5 Digital I/O channels in a modern car and corresponding attack vectors (Checkoway et al., 2011) 24

4.1 Basic concept of the CAN-CT and its underlying attack scenario 34
4.2 Simplified UML2 class diagram of the CAN-CT 38
4.3 Monitoring the CAN bus with the CAN-CT 40
4.4 CAN-CT's CAN capturing functionality 41
4.5 CAN-CT's plan of attack definition 42
4.6 Customized attacks via the CAN-CT's scripting engine 44
4.7 USB to CAN adapter configuration and CAN node simulator 45

5.1 Pictures of the heavy duty vehicle Hardware in the Loop System setup 59
5.2 Architecture of the laboratory experiments test setup 62
5.3 Frame structure of the targeted CAN frame used in the laboratory setup 63
5.4 Frame structure of the attacking CAN frame used in the laboratory setup 63
5.5 Event-based replay attack observed via oscilloscope in the laboratory setup 64
5.6 Overview of time-based spoofing attacks with a different offset in the laboratory setup 66

5.7 Overview of CAN frames sent during a bus off attack in the laboratory setup 68
5.8 Active and passive error frames captured during a bus off attack in the laboratory setup 69
5.9 Denial of Service attack with an ID of 00Dh in the laboratory setup 70
5.10 Overview of the VGS at high and at low engine speeds 73
5.11 Architecture of the heavy duty vehicle HiL setup 74
5.12 Example communication between the VGS controller and the actuator in the heavy duty vehicle HiL system setup 75
5.13 Time-based spoofing attacks with an offset of 1.5 ms and a target value of zero respectively 30 percent in the heavy duty vehicle HiL system setup 77
5.14 Bus off attack with an DLC of 5 and the target value set to 30 percent in the heavy duty vehicle HiL system setup 79
5.15 DoS attack with ID 1BEh in the heavy duty vehicle HiL system setup 81
5.16 Overview of different time offset values and their success on spoofing the target position of the VGS in the heavy duty vehicle HiL system setup 82
5.17 Overview of different frame structures with regard to their data length code and their impact on the acceptance by the targeted ECU in the heavy duty vehicle HiL system setup . . 84
5.18 Reaction of the observed ECU when confronted with illegal values in the heavy duty vehicle HiL system setup 85
5.19 Network architecture of the second Hardware in the Loop System setup, a passenger car 89
5.20 Speed related CAN messages with example values used for our attacks in the passenger car HiL system setup 90
5.21 Plausibility checks for the internally calculated vehicle speed based on the received wheel speeds in the passenger car HiL system setup 93

Index of Abbreviations

CAN	Controller Area Network	101
CAN-CT	CAN Communication Tester	102
CAN-FD	Controller Area Network Flexible Data Rate	11
CiA	CAN in Automation	20
CRC	cyclic redundancy check	75
DCU	dosing control unit	72
DLC	data length code	62
DoS	Denial of Service	102
DUT	device under test	71
ECU	electronic control unit	101
GUI	graphical user interface	36
HiL	Hardware in the Loop System	102
IDE	identifier extension	16
LIN	Local Interconnect Network	10
MOST	Media Oriented Systems Transport	10
NRZ	non-return-to-zero	15
OBD-II	On-Board Diagnostics II	35
QPC	QueryPerformanceCounter	37
RTR	remote transmission request	16
SAE	Society for Automotive Engineers	10
SUT	system under test	57
VGS	variable geometry turbocharger system	72

1 Introduction

1.1 Motivation

A few years ago vehicles were mere mechanical machines, containing but a few electronic devices communicating in a closed network. However, as time changed so did vehicles. Nowadays premium cars contain up to 100 electronic control units (ECUs) with almost 100 million lines of code (Charette, 2009). These ECUs are tightly connected via different bus systems and are continuously exchanging information with each other in a real time setup. Upcoming, next-generation features like autonomous driving, vehicle to everything (V2X) communications or the already existing interconnectivity via 3G/4G, WiFi, Bluetooth etc. are changing our mobility significantly.

Even though these new technologies improve safety and comfort, they are also accompanied by a vast number of security-related threats. Due to the continuously increasing complexity and the growing number of external interfaces of the vehicle's internal network with its environment, new attack surfaces emerge.

Manufacturers have seen cars as closed networks without the primary need of security mechanisms in the automotive networks for too long. Many researchers already demonstrated the serious lack of security in modern cars. Already in the year 2006 Wolf, Weimerskirch, and Paar (2006) highlighted a major security issue of vehicular networks in their paper "Secure In-Vehicle Communication." They showed that the Controller Area Network (CAN) is inherently vulnerable to possible hacking attacks, as it connects the most critical components (for instance brakes, engine, powertrain) with hardly any security features implemented.

By exploiting this lack of security mechanisms on the CAN bus, researchers showed that taking over control of a car via sending malicious messages on the bus has become reality (Koscher et al., 2010; Miller and Valasek, 2013; Hoppe, Kiltz, and Dittmann, 2011; Staggs, 2013). In fact, by taking advantage of modern technologies like for example the brake or parking assist systems serious attacks are possible.

These results already highlighted the severe vulnerabilities of the internal vehicular network and the urgent need of implementing countermeasures. However, as all of these attacks were executed with direct, physical access to a cars internal network, these works were still not considered as realistic threats (BBC News, 2010; Leyden, 2010).

However, Checkoway et al. (2011) showed that it is possible to hack a car, compromise the attacked ECU and take over control of the vehicle via almost every externally facing I/O interface. They were able to control arbitrary vehicular functions even from a distance. In 2015 Valasek and Miller (2015) exploited a vulnerability in a car's cellular interface. This vulnerability allowed them to hack and remotely control every affected car just by knowing its IP address. No changes to the manufacturer's configuration had to be made beforehand.

This research clearly demonstrates the urgent need to secure cars. It is vital to establish ways to protect those networks from hackers. However, just securing the external interfaces of a car is not enough. It is crucial to also find ways to impede and to detect malicious messages within an internal vehicular network.

Although various approaches with regard to encryption, authentication and attack detection mechanisms were published in the last few years (see Section 3.2), none of them were officially accepted in the automotive sector so far (Navet and Simonot-Lion, 2013). Hardware limitations, the heterogeneous networks consisting of distributed ECUs from various vendors and proprietary higher layer protocols, impeded an integration of new security techniques. As a result there is still no common, standardized solution for authentication, authorization or encryption within the CAN.

Car hacking research highlighted that the automotive industry needs tools and techniques to protect their CAN communication mechanisms. Besides sophisticated security measures it needs tools to efficiently test ECUs for possible vulnerabilities as well.

With this thesis we want to introduce a tool that efficiently tests ECUs for such vulnerabilities. By carrying out various hacking attacks we are trying to manipulate or to take over control of an ECU. This approach allows us to detect implementation flaws of the CAN communication mechanisms and thus helps assuring a safe and secure CAN bus. We called our solution the CAN Communication Tester (CAN-CT), which we are going to introduce in the following chapters.

1.2 Problem

The inherent issue with today's vehicles is the ECU communication via a non-secure bus. Almost all crucial ECUs are connected via the CAN bus. Before the recent development security measures and possible hacking attacks were hardly a concern. CAN communication was just tested for safety and functional requirements but not for malicious behavior.

However, as research demonstrated hacking a vehicle is reality. Securing the I/O channels of a car is not sufficient. For instance considering the attack scenario that a hacker was able to take over control of an ECU via a remote attack; without any further security mechanisms on the vehicle's internal communication systems, he or she would be in a position to manipulate arbitrary ECUs with facing hardly any obstacles.

This scenario shows, that manufacturers have to secure their vehicular communication systems as well. Stronger emphasis is and will be put on more secure CAN communication. Yet due to the growing complexity of modern ECUs and in-vehicle communications it is very likely that flaws in the implementation of such safety and security measures exist. This implies that we need a testing environment to assure the proper working of these techniques. We are convinced, the one key factor of such a security testing environment is the verification of these security measures through the

simulation of hacking attacks. This requirement illustrates that such a testing environment differs in its fundamental testing concept from traditional safety and functional tests. We feel certain, that such a testing environment would be ideally suited to complement those existing environments to ensure a higher level of safety and security.

1.3 Solution

With this thesis we develop a software application prototype named CAN Communication Tester (CAN-CT) to efficiently test vehicular electronic control units (ECUs) for security (and potentially safety) vulnerabilities.

The CAN-CT is based on the attack scenario that a hacker was able to gain full access to the CAN bus by compromising an ECU via an I/O channel. Based on this scenario the CAN-CT behaves like an actual attacker on the bus. This approach allows us to examine the behavior of ECUs during the execution of arbitrary attacks and thus lets us analyze ECUs and their CAN networking in exceptional states.

We validate the correct implementation of CAN networking including safety measures like sequence numbers, application layer checksums, time outs, plausibility checks and error/failure handling mechanisms. We will test these mechanisms by carrying out hacking attacks like invalidating, manipulating, replaying or injecting messages into the CAN bus.

The CAN-CT will be based on the assumption that sufficient knowledge of the CAN communication is available and that direct and full access to the bus exists. It can be understood as an application of the Man-in-the-Middle attack model with the aim to highlight faulty implementations of safety and security measures. Attack vectors to gain access to the CAN bus are not examined within this thesis.

1.3.1 Objectives

Our goals for the CAN-CT are:

- Testing the correct implementation of CAN networking including current protection mechanisms
- Supporting for attacks that invalidate, manipulate, replay and inject CAN messages
- Supporting various higher layer protocols for analysis and attacks
- Testing the implementation against an unknown ECU in a real world scenario
- Implementing the prototype as a learning system by analyzing the traffic on the CAN bus directly. We determine important message IDs and their respective purpose in the CAN communication and use them as the input for further attacks

1.3.2 Outlook on Results

In this thesis we show that we were able to prove the CAN-CT working properly in a laboratory setup as well as in real world scenarios. For the latter we made use of two different Hardware in the Loop Systems (HiLs). One represented a heavy duty vehicle while the other simulated a passenger car. Both HiLs consisted of multiple hardware ECUs, sensors and actuators. For our tests we mainly focused on the engine control unit as well as on turbocharger control units.

These test setups allowed us on the one hand to demonstrate the proof of concept of being able to hack a vehicular communication network with the CAN-CT, while on the other hand we detected serious weaknesses in the implemented protection mechanisms of the targeted ECUs.

In more detail these flaws consisted of serious lacks in the plausibility checking of messages, of a too loose time out window for incoming messages, a not strict enough alive counter checking procedure and a completely omitted examination of the correct frame structure. We detected all of those weaknesses by attacking the targeted ECU with the implemented hacking attacks. These attacks consisted mainly of series of impersonation attacks and special forms of Denial of Service (DoS) attacks.

The flaws we identified in the protection mechanisms make it easier for attackers to manipulate ECUs. We exploited those flaws to take over message IDs owned by another ECU. For instance with our attacks we were able to fully control the turbocharger. Thus we were able to define the air pressure in the combustion chamber. Causing the turbocharger to generate a too high air pressure over a longer period of time might finally lead to damages of the engine.

In summary, with this thesis we demonstrate, that with the CAN-CT we are able to manipulate ECUs, suppress all communication on the bus and bring ECUs into error states.

1.4 Outline

This thesis starts with preliminaries (Chapter 2). This chapter provides a basic understanding of the technologies, protocols and other measures needed for this thesis. Chapter 3 gives an overview of related work. It is divided into the three main areas that influenced this thesis: car hacking techniques, attack detection mechanisms and attack prevention measures. In Chapter 4 we introduce our tool - the CAN Communication Tester (CAN-CT). Chapter 5 then gives a detailed overview of the real world applicability and impact of the CAN-CT. The thesis concludes with Chapter 6. Here the main points are summarized and possible future work is discussed.

2 Preliminaries

This chapter introduces the major technologies, protocols and concepts on which this thesis is based. We define important terms and give an overview of relevant topics with regard to automotive communication systems.

2.1 Terminology and Definitions

In this section describe and define different terms and concepts used within this thesis.

2.1.1 Electronic Control Unit

Modern cars consist of a large number of electronic devices that control almost every aspect of a car. These devices are self-contained automotive embedded systems called electronic control units (ECUs) (Koscher, 2014).

Virtually every component of a car, including the breaks, the engine, the throttle, lighting controls, entertainment system and so on are governed by these embedded systems (Koscher, 2014). This leads to almost 100 ECUs in a modern luxury sedan (Charette, 2009). These computers monitor and control not only sensors and actuators but also technologies supporting the driver, the passengers or even certain driving situations. Those features encompass technologies like ABS, ESP or further advanced driver assistant systems.

Historically the first ECU in a car was in charge of controlling and monitoring the engine. Therefore many researchers still define the acronym ECU as Engine Control Unit. However, because these control units are now operating various aspects of a car - and not only the engine - we are defining the term ECU as *electronic control unit*.

2.1.2 Hardware in the Loop System

A Hardware in the Loop System (HiL) can be understood as a test framework where parts of the system are simulated while other parts are actual hardware devices. This approach allows to test a hardware in a true-to-nature test condition without the need of the actual system to be built (Ren, Steurer, and Woodruff, 2007).

Furthermore the actual hardware can be tested in extreme conditions, which helps identifying hidden defects. This is especially useful for a cost and risk effective development of the investigated system.

A vehicle represents one use case for a HiL. A HiL therefore helps developing certain parts of the vehicle without the need of the actual vehicle. The missing hardware components are just simulated by the HiL.

The hardware ECU under test gets all the information from the HiL as if it was coming from the actual vehicle components. Therefore for the device under test (DUT) there is little to no difference between the HiL and nature.

In a HiL it is possible to manually put the DUT in different states and imitate arbitrary situations.

2.1.3 Attack

For our tests in Chapter 5 we define an *attack* as follows:
An attack consists of at least one injected message. If it is composed of multiple messages, these messages must have the same message ID (see Section 2.3.4) as well as the same kind of attack described in Section 2.4.2.

2.1.4 Test Case

We define the term *test case* as follows:
A test case consists of at least one attack and serves exactly one purpose.

An example of such a test purpose can be the verification of certain plausibility checks of an ECU. Such a verification can consist of multiple attacks but will be treated as one test case.

2.1.5 Plan of Attack

We define a *plan of attack* as follows:
A plan of attack consists of at least one test case. If it combines multiple test cases, it further states how and when they are executed.

An example of such a plan of attack is the consecutive execution of multiple test cases. For instance, one can base the execution of the second test case on the success of the previous one. This means the second test case might need a prior exploitation of a vulnerability caused by the first test case to work.

Thus, we can say, a plan of attack includes one or more test cases which in turn use at least one attack to reach their purpose.

2.2 In-Vehicle Communication Systems

Until the beginning of the 1990s most ECUs were connected via point to point links (Navet and Simonot-Lion, 2013). The growing number of ECUs made it necessary to use more sophisticated communication systems.

The varying characteristics of a state of the art vehicle demanded various attributes of such a system. For example, fast communication for multimedia services or safe real time connections for critical power train functions. This requirement led to multiple, separate vehicular networks in state of the art vehicles (Tuohy et al., 2015). These network technologies encompass protocols like CAN, LIN, FlexRay or MOST.

For example, the BMW 7 series, launched in 2008, contains four Controller Area Network (CAN) buses, a FlexRay bus, a Media Oriented Systems Transport (MOST) bus, multiple Local Interconnect Network (LIN) buses, an Ethernet bus and several wireless interfaces. Most of these different networks are connected via a central gateway (Navet and Simonot-Lion, 2013).

2.2.1 Overview of Current Automotive Networks

The reason why modern cars use many different communication systems lies in the varying types of data that has to be transmitted. Based on the requirements on transmission speed and function, the Society for Automotive Engineers (SAE) established a classification for automotive communication protocols (Navet and Simonot-Lion, 2013). This classification divides the requirements for data transmission into four categories: Class A, Class B, Class C and Class D.

Class A networks are characterized by cost efficient technologies for low bandwidth transmissions ($<$ 10 kbit/s). This is suitable for simple control data like a car's lighting, the seat control or the door lock. For these kind of data rates mostly LIN is used.

A class B network operates at a rate between 10 kbit/s and 125 kbit/s. Its main purpose is to reduce the number of sensors by sharing information between ECUs. In most cars this is realized with a low-speed CAN bus.

High speed real-time communications happen at speeds of 125 kbit/s to 1 Mbit/s and are subsumed by class C. Data transmitted on class C networks have high demands regarding real-time and failure resistance. In modern cars the powertrain and the chassis domain communicate with networks in this class. Mostly a high-speed CAN bus is used for this purpose.

The last class introduced by the SAE is Class D. It requires speeds of more than 1 Mbit/s. These high speed networks are mostly used for multimedia data or as gateways between other networks. The common protocol for this class is MOST. This protocol can reach a maximum bandwidth of 150 Mbit/s with the standard MOST150 (Tuohy et al., 2015).

Besides these protocols there are also other communication systems like FlexRay, Automotive Ethernet or Controller Area Network Flexible Data Rate (CAN-FD). The latter two especially were developed as next generation protocols to overcome limitations of current systems.

FlexRay was released in 2004 with the aim to replace the CAN protocol. Its main advantages are a higher data rate (10 Mbit/s), its deterministic time-triggered time division multiplexed access (TDMA) media sharing and its high flexibility (Tuohy et al., 2015). However, due to higher costs of the nodes it was not able to gain wide acceptance.

With the introduction of the CAN-FD in 2012, we believe it is most likely that CAN-FD will be the CAN protocol's successor.

2.2.2 Limitations of Current Automotive Networks

An increasing number of data-intensive functions changed the requirements of vehicular communication systems. Due to its limitations the CAN protocol for instance cannot fulfill all demands of modern ECUs. It is necessary to process larger quantities of data at real-time while working more energy efficiently. Furthermore it is crucial to solve the inherent security issues of current protocols (Navet and Simonot-Lion, 2013).

Especially these vulnerabilities to attacks became a great concern of many manufacturers. As research in the last few years demonstrated the CAN protocol with its payload of eight bytes is too limited to provide sufficient security techniques. Happel, 2014 states that CAN-FD can overcome this issue. Due to the increased payload of 64 bytes CAN-FD meets the requirements of modern vehicular communication systems while allowing for the integration of security mechanisms as well.

2.3 Controller Area Network

Since 1994-1995 the CAN bus is the most widely used communication system for automotive applications (Lawrenz, 2013). Besides its application in Class B networks it is mostly used for Class C data transmission (see Section 2.2.1). As it connects the most critical components within a car, it is one of the most important buses.

2.3.1 History and Standardization

The CAN protocol was first introduced in 1986 at an SAE conference in Detroid (Lawrenz, 2013). In 1991 it was the first bus system that was implemented in a vehicle in mass production (Mischo et al., 2015).

The current version of the CAN protocol is the version 2.0. This version is divided into two parts - 2.0a and 2.0b. Both specifications are identical except for the additional support of 29-bit message identifiers in version 2.0b, whereas 2.0a just uses 11-bit identifiers (Navet and Simonot-Lion, 2013). Due to the larger number of different message IDs version 2.0b is mostly used for heavy-duty vehicles, while 2.0a principally is used for passenger cars.

The CAN protocol is standardized since 1993. It is defined in the ISO Standard 11898 which is split into five parts, where the first three parts are the most relevant for the automotive industry. ISO 11898-1 defines the CAN protocol and the entire data link layer as well as parts of the physical layer. The second part (ISO 11898-2) defines the physical layer behavior for the High-Speed CAN, whereas the third part describes the Low-Speed CAN bus (Mayer, 2006). ISO 11898-4 defines a time-triggered CAN based communication protocol (TTCAN) and ISO 11898-5 outlines a high-speed medium access unit with low-power mode (Lawrenz, 2013). Navet and Simonot-Lion, 2013, however, note that the time-triggered CAN protocol has not been used in production cars so far.

Besides these ISO standards, the CAN protocol is also standardized by other organizations like the SAE (J2284-1 to 3).

In this thesis we limit the discussion to the CAN 2.0a version and the ISO 11898-2 standard, as this is the most widely used standard for CAN networks (Di Natale et al., 2012).

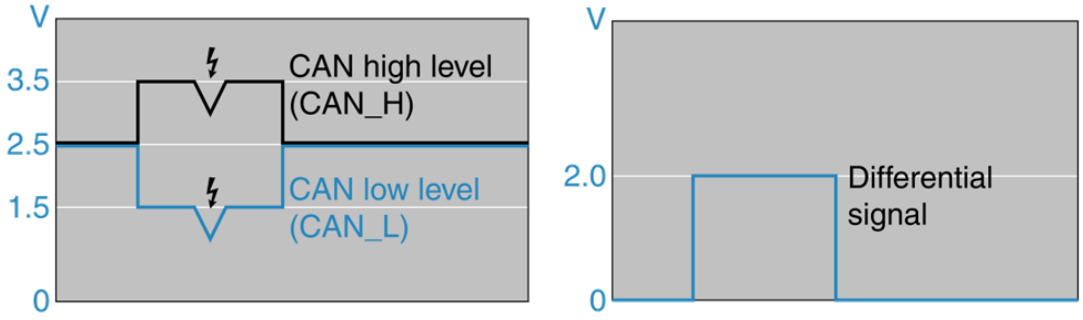
2.3.2 Characteristics

The main goal of the CAN protocol is to efficiently support distributed real time control with a very high level of security even in large and complex networks. The CAN specification in its version 2.0 (Bosch, 1991) states the following points as main characteristics:

- prioritization of messages based on their identifier,
- guarantee of maximum latency times,
- configuration flexibility,
- multicast reception with time synchronization,
- system wide data consistency,
- multimaster bus access,
- error detection and signaling,
- automatic retransmission of corrupted messages as soon as the bus is idle again, and
- distinction between temporary errors and permanent failures of nodes and autonomous switching off of defect node.

The CAN bus is a broadcast medium that consists of nodes with equal rights (multimaster). Every node has the same right to send a message. The prioritization mechanism helps avoiding possible collisions on the medium. The node sending a message with a lower ID (higher priority) will prevail on the bus. The other node has to wait. If, however, a collision still occurs, the error handling techniques signal an error and the failed message will be retransmitted. In this way the protocol assures a system wide data consistency.

In the further sections we will detail most of those points again.



(a)CAN-H and CAN-L data transmission according to ISO 11898-2

(b)Differential amplifier subtraction of CAN-H and CAN-L signals according to ISO 11898-2

Figure 2.1: CAN signal interference filtering according to ISO 11898-2 (Mischo et al., 2015)

2.3.3 Physical Layer Attributes

According to the ISO 11898-2 standard - the most common standard for CAN networks (Di Natale et al., 2012) - the network is based on a two-wire differential bus. These wires are named CAN-H (CAN High) and CAN-L (CAN Low).

The idle state of both wires lies around 2.5 V (Mischo et al., 2015). This voltage is also the recessive state of the CAN bus and represents a binary 1. In the dominant state both wires change their voltage by 1 V. For the CAN-H this means a dominant state of 3.5 V and for the CAN-L a drop to 1.5 V. The dominant state of the CAN bus represents the binary 0. If two or more nodes simultaneously transmit different messages, the dominant bit always overwrites the recessive bit. This behavior is also used for the CAN protocols prioritization mechanism described in Section 2.3.5.

By comparing the symmetry of both signal flanks the CAN controller is resistant against interferences. A differential amplifier subtracts the CAN-L voltage from the CAN-H and converts it back into logical states (Mischo et al., 2015). This approach is depicted in Figure 2.1.

Table 2.1: Bit rates, maximum bus lengths and corresponding bit times according to CANOpen (Mischo et al., 2015)

Bit Rate	Bit Time	Bus Length
1 Mbit/s	1 μs	25 m
800 kbit/s	1.25 μs	50 m
500 kbit/s	2 μs	100 m
250 kbit/s	4 μs	250 m
125 kbit/s	8 μs	500 m
62.5 kbit/s	16 μs	1000 m
20 kbit/s	50 μs	2500 m
10 kbit/s	100 μs	5000 m

In an ISO 11898-2 CAN network data rates up to 1 Mbit/s are possible. Dependent on the speed, the maximum bus length differs. Di Natale et al., 2012 cite the speeds, bus lengths and bit times from CANopen listed in Table 2.1.

The CAN bus uses non-return-to-zero (NRZ) as the encoding method for data transmission. This means there is no compulsory return to zero between two equal states. This method provides the advantage of requiring only a minimum bandwidth (Lawrenz, 2013). After the transmission of five consecutive bits of the same value, a stuff bit of the inverted value is inserted.

Based on the edges from the recessive to the dominant state (and optionally vice versa in case of low bit rates) the CAN controllers synchronize their internal bit time (Bosch, 1991).

2.3.4 Frame Formats

The CAN protocol differentiates four types of frames - data frame, remote frame, error frame and overload frame. The data frame is the only frame that actually transmits message data. The other frames are used to request the transmission of a data frame, to allow for fault confinement, for synchronization and for flow control (Lawrenz, 2013; Di Natale et al., 2012).

In the following section we are going to detail the data frame. The error frame will be described in Section 2.3.6.

Data Frame

The data frame is used to transmit information on the bus. As the CAN network is a broadcast medium, every node receives every message. The CAN protocol does not use any source or destination address. Instead it uses an identifier which defines the message's content (for instance the vehicle speed, engine parameters and so on). Based on this identifier each CAN node decides if the message will be further processed or if it will be ignored (Mischo et al., 2015).

This approach allows for a high flexibility of the network. It does not need any information or configuration of the nodes. A new node just connects to the network and it can send and receive messages immediately. However, without any prior knowledge of the messages sent on the bus, a new node cannot determine their meaning or ascertain which messages are relevant.

The data frame in the CAN version 2.0a (Bosch, 1991) is depicted in Figure 2.2. After a Start of Frame bit (SOF) the data frame starts with an ID of eleven bits. After the ID section, the frame has a remote transmission request (RTR) bit. This bit states, if the frame is a data frame (binary value 1), or if it is a remote frame (binary value 0). Combined with this RTR bit the ID gives the arbitration field which will be detailed in the next section.

The subsequent control field is composed of an identifier extension (IDE), a reserved bit and the data length code (DLC). The IDE represents the current version of the frame. A binary 1 represents the version 2.0a and a binary 0 refers to the version 2.0b. The DLC then states how many data bytes the message uses. The minimum value is 0 - no data at all - and the maximum length is eight (8 bytes).

After the control field the actual data is transmitted. Based on an optional higher layer protocol each bit of this section can have additional meanings. The CAN protocol per se does not make any assumptions regarding this part.

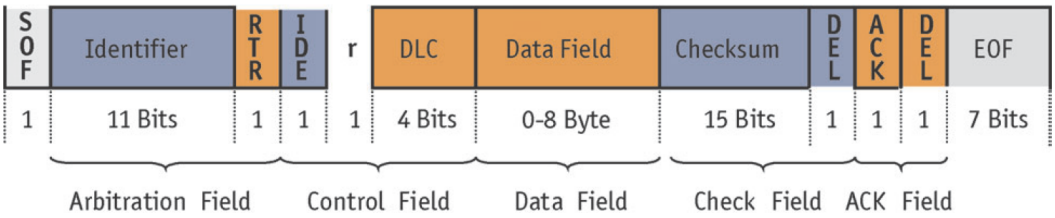


Figure 2.2: CAN 2.0a Data Frame (Mayer, 2006)

The data frame uses a 15 bit checksum to provide a check for the correctness of the received frame. This cyclic redundancy check (CRC) is obtained by defining the input polynomial, the coefficients of which are given by the destuffed stream of bits starting from the Start Of Frame bit to the data bytes (if present) (Bosch, 1991). The resulting polynomial is then divided by the following generator-polynomial:

$$X^{15} + X^{14} + X^{10} + X^8 + X^7 + X^4 + X^3 + 1$$

The remainder of this polynomial division is the CRC sequence. More details about this calculation can be found in the CAN specification (Bosch, 1991).

After this checksum, the frame consists of a delimiter and an acknowledgment bit. This bit is sent out with a value of 1 (recessive state). Every node that correctly receives the message overwrites this bit with a binary 0 (dominant state). This means the initial recessive state for this bit, will be set to the dominant state by every recipient that received the message correctly. This way the sender notices if its message was received correctly by at least one node. It is, however, not possible to determine if every node received the message correctly (overwriting a dominant bit with a recessive state is not possible).

The frame concludes with an acknowledgment delimiter and 7 recessive bits representing the end of frame (EOF). After these bits an interframe spacing of another 3 bits is used. This means the next message can be send after ten recessive bits of waiting.

2.3.5 Arbitration

The CAN protocol uses the carrier sense multiple access/collision avoidance (CSMA/CA) technique to avoid collisions on the bus. This approach is realized by the arbitration mechanism (Johansson, Törngren, and Nielsen, 2005). For this purpose the arbitration field of a CAN frame is used to determine who is allowed to send.

Arbitration is a technique to handle possible bus access conflicts. Whenever two or more nodes start transmitting a frame at the same time, it is used.

Every node transmits its message bit per bit while simultaneously watching the actual signals on the bus. If the signals received correspond with the signals sent, the node continues transmitting; if not, it stops immediately. A difference between those values occurs when a sent recessive bit (binary 1) was overwritten by the dominant bit (binary 0) of another message. The node then has lost arbitration and must withdraw without sending one more bit (Bosch, 1991).

Due to the dominant state of 0 the node sending a message with the lowest ID gets the highest priority. It will prevail in the arbitration process and can transmit its message.

Figure 2.3 illustrates this approach. Two nodes are sending at the same time. The first three bits of both messages are equal. Both nodes continue sending. At bit four the binary 1 (recessive state) of sender C is overwritten by the binary 0 (dominant state) of sender A. Sender A won the arbitration. Sender C thus stops transmitting.

2.3.6 Error Handling

The CAN protocol knows five different types of errors (Bosch, 1991):

- *Bit-Error:* A sent bit differs from the actual bit on the bus (after arbitration),
- *Stuff-Error:* More than five consecutive bits of the same value were sent (except for end of frame and interframe spacing),
- *CRC-Error:* The CRC contained in the transmitted frame is not correct,

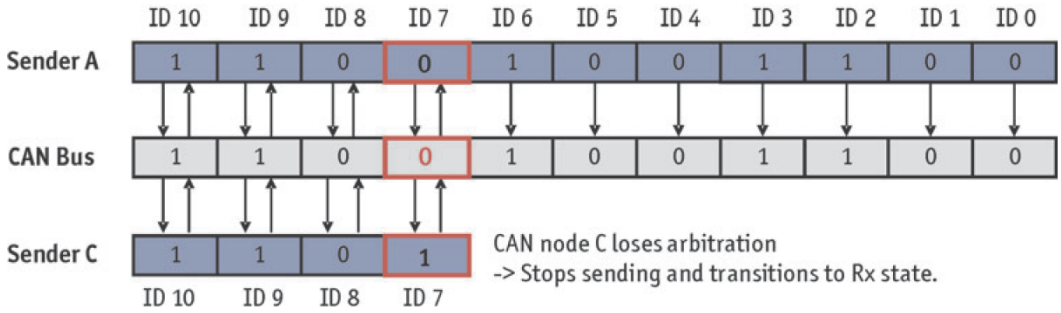


Figure 2.3: Depiction of the arbitration process between two CAN nodes (Mayer, 2006)

- *Form-Error*: The frame format (delimiters, end of frame) is corrupted,
- *Acknowledgment Error:* The acknowledgment bit of the message remains in the recessive state - this implies no other node received the message correctly.

When any of those errors are detected by either the sending node or the receiving nodes, it immediately sends an error frame. The error frame with an active error flag consists of six consecutive dominant bits. This frame overwrites the currently transmitted frame that caused the error. Due to the violation of the bit-stuffing rule caused by the error frame other CAN nodes start sending an error frame as well (Lawrenz, 2013). This leads to a possible superposition of error frames for up to twelve dominant bits. After the error frame each node sends eight recessive bits (error delimiter).

To confine faulty CAN nodes the protocol defines three operation states, which will be detailed in the following paragraph (Johansson, Törngren, and Nielsen, 2005):

1. *Error Active*: a node in this state works properly. If an error is detected it sends an error frame with active error flag.
2. *Error Passive*: in this state the node is seen as potentially faulty. It still can receive and send messages but an error will be signaled with an passive error flag (six recessive - not dominant - bits).
3. *Bus Off*: a node that entered the bus off state is not allowed to participate on the bus anymore. It is seen as defective.

To enter these states, each node uses two counters (transmit error counter (TEC) and receive error counter (REC)). A detected/signaled error increases the receive/transmit error counter by eight; a correctly received/sent message reduces it by one. If one of those counters reaches a value of more than 127 the node enters the error passive state. In this state the node uses six recessive bits to signal an error. This serves the purpose to not destroy messages sent by other nodes with this error flag. The reason is that the error passive CAN controller might be faulty itself. Only after both counters dropped under the limit of 127 the node is allowed to send active error flags again.

If the transmission error counter is raised to a value of more than 255 the node has to enter the bus off state. It is not allowed to send nor to receive messages on the bus. This means no messages will be further processed by the CAN controller or forwarded to higher layer applications. The CAN controller ignores all traffic on the bus. Only after reseting its counters the node can participate again. If, however, a CAN controller does not conform to the CAN protocol's error handling procedure, there is no possibility to isolate it. Thus, the CAN bus relies on the conformity with the protocol of each node.

These state transitions are depicted in Figure 2.4

2.3.7 Higher Layer Protocols

The CAN protocol per se just defines the lower two OSI layers. For higher layer applications a large number of protocols exist. According to Kvaser, 2012 there are several dozens of protocols. Many of those are standardized by different organizations. Examples are the SAE J1939 (defined by the SAE), the CANopen standardized by the CAN in Automation (CiA) or the DeviceNet protocol (Johansson, Törngren, and Nielsen, 2005).

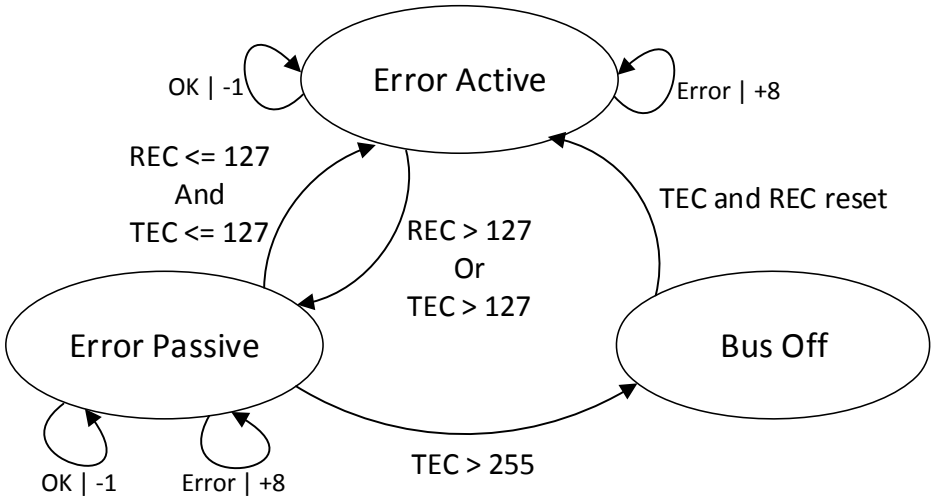


Figure 2.4: CAN error handling state machine

Besides these three protocols Kvaser, 2012 lists the following protocols as the most common ones:

- CANKingdom,
- CCP/XCP,
- MilCAN,
- NMEA2000,
- OSEK/VDX, and
- SDS.

These protocols differ in their characteristics regarding the targeted application they should support. This is reflected in their real-time control abilities, their flexibility regarding network configuration or their data rates (Johansson, Törngren, and Nielsen, 2005). Some protocols are used especially for diagnostic purposes (for instance ISO 14229, ISO 14230) while others are used for normal communication between ECUs.

To harmonize the communication techniques of different vendors a common automotive industry standard architecture named AUTOSAR was developed (Lawrenz, 2013). However, as research showed (Miller and Valasek, 2013; Koscher, 2014; Staggs, 2013; Tuohy et al., 2015) most manufacturers still use proprietary extensions, modifications or higher layer services. Koscher et al., 2010 even states that many ECUs deviate from their own protocol standards as well.

2.3.8 Current Protection Mechanisms

Vehicular communication networks followed a security by isolation approach. As, however, this isolation is not existent anymore, new security techniques are needed. So far no standardized security measures for CAN communication exist. Staggs, 2013 considers the security through obscurity approach still as the prevailing technique to protect cars. Message IDs and content encoding are confidential. This correlates with current car hacking research (Valasek and Miller, 2015; Miller and Valasek, 2013; Koscher, 2014). However, they clearly demonstrated that these techniques fail to secure a car.

Miller and Valasek, 2013 showed that normal CAN communication is not protected at all. Besides the nondisclosure of the message IDs and the encoding of the content some manufacturers use additional CRC checks, alive counters and time out windows - but no security measures.

Application layer CRCs use a simple calculation over the data bytes and the ID to assure the integrity of the message. Alive counters serve as a sequence number which increases with every sent message. This assures the receipt of the messages in the right order and lets the ECU determine if a message got lost. A time out window defines the time when a message is allowed to be received. If it arrives before or after this window it is rejected.

Authentication mechanisms are only implemented for diagnostic purposes, like flashing the ECU (Koscher, 2014; Miller and Valasek, 2013; Valasek and Miller, 2015). Koscher et al. (2010) state, in their examinations these mechanisms use weak keys, a fixed challenge (seed) and are both just 16 bits long.

Summing up, the current way to protect the CAN bus against unauthorized manipulation is the nondisclosure of message IDs and the proprietary encoding of data, the usage of mechanisms like application layer CRC, alive counter and time out windows as well as a (weak) authentication for diagnostic applications (Tao, Antunes, and Aggarwal, 2014). Besides those techniques, ECUs use plausibility checks to determine the correctness of a received message. These plausibility checks verify the validity of a received CAN signal by comparing it to other signals. These other signals can be generated by any sensor, sent by another ECU or calculated from other similar values.

2.4 Car Hacking

As research shows, CAN buses have been treated as closed networks for too long. Nowadays these buses are connected to a broad number of interfaces to the outside world. This reveals a large number of possible attack vectors. Research demonstrated in the last few years that car hacking is reality; for instance, Valasek and Miller (2015) were able to fully remote control a car just by knowing its IP address.

2.4.1 Attack Vectors and Security Threats

After being criticized by the media for building their research on an unrealistic attack scenario (Leyden, 2010; BBC News, 2010), Checkoway et al., 2011; Valasek and Miller, 2015 decided on focusing on possible attack surfaces. While Checkoway et al., 2011 demonstrated that they were able to take over a car via almost every connection with its outside world, Valasek and Miller, 2015 revealed an exploit in the cellular interface of a car that would let them remote control up to 471,000 vehicles. Both showed that even though these interfaces use sophisticated security mechanisms, there are also vulnerabilities.

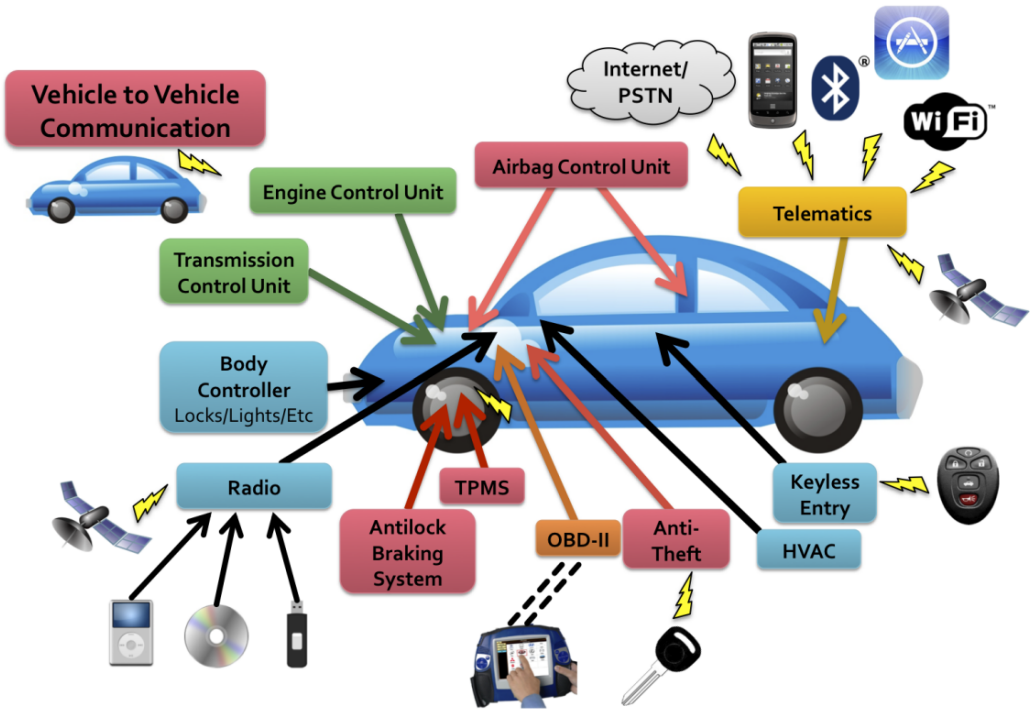


Figure 2.5: Digital I/O channels in a modern car and corresponding attack vectors (Checkoway et al., 2011)

Figure 2.5 depicts typical connection of a modern car with its outside world. Checkoway et al. (2011) roughly grouped the ECUs according to their function (indicated by their color). In their paper they analyzed the vulnerability of those I/O channels. Checkoway et al. demonstrated that they could gain full control of all connected ECUs on the respective CAN bus and, by transitivity, of all ECUs in the vehicle by exploiting those interfaces. Their examination encompassed hacks via the car's radio through a specially prepared audio CD, via the OBD-II channel (a diagnosis port for mechanics), the bluetooth interface, the tire pressure monitoring system (TPMS) or even the cellular modem by calling it and playing a crafted audio signal.

2.4.2 Attacks

Each attack surface needs individual examination for vulnerabilities. The focus of this thesis is already one step further. We are dealing with possible hacking attacks when an attacker already gained access to the CAN bus.

Here we distinguish between the following types of attacks.

Replay Attack

C.-W. Lin and Sangiovanni-Vincentelli, 2012 define a replay attack as a subcategory of a masquerade attack. This means the attacker sends a message in which he claims to be a node other than himself. For a replay attack this means that the message is not altered at all. The exact copy of a previously received message will be sent out by the attacker.

Knapp and Langill, 2015 state that a replay attack consists of two tasks: recording and replaying. The attacker is able to listen on the bus and starts recording interesting packets; for instance, when an authentication mechanism takes place on the network. By replaying the recorded message the attacker then might be able to gain access to certain resources.

Spoofing Attack

The spoofing attack is similar to the replay attack. It is also a form of a masquerade attack. The difference is that instead of replaying a previously recorded message, an altered or completely different message will be sent. Yet, the attacker still claims to be a node other then himself.

Based on the results of Koscher et al. (2010), Koopman and Szilagyi (2013) consider a spoofing attack as a way to make an embedded system unsafe in essentially limitless ways.

Denial of Service Attack

Knapp and Langill, 2015 define a Denial of Service (DoS) attack as a malicious attempt to make a resource unavailable. This type of attack encompasses ways of inhibiting or crashing a particular service in a device, suppressing all communication with a device or every other way of hampering an attacked device to communicate.

In traditional business systems, Knapp and Langill state, this does not typically cause significant negative consequences if resolved in time. However, in industrial environments this attack can lead to serious consequences if not managed accordingly.

3 Related Work

The underlying research area of this thesis can be divided into 3 sections. Our thesis is influenced and motivated by the results of current car hacking research. Based on these findings a lot of research was done to protect against those kind of attacks. Here we distinguish between attack detection and attack prevention mechanisms. We took these approaches into consideration for the design of the application we are going to introduce with this thesis. The third part, which influenced our research, are current efforts regarding automotive security testing approaches.

3.1 Car Hacking Techniques

In the last few years extensive academic as well as non-academic research was done to determine the vulnerability of vehicles against hacking attacks. Koscher et al. (2010) were one of the first to test this vulnerability comprehensively in a real-world scenario. Before most research dealt with more theoretical questions regarding possible hacking threats and their investigation (Larson and Nilsson, 2008; Wolf, Weimerskirch, and Paar, 2004; Wolf, Weimerskirch, and Paar, 2006; Nilsson, Phung, and Larson, 2008).

For instance Nilsson, Phung, and Larson (2008) discussed the specifics of performing a digital forensic investigation of cyber attacks on vehicular networks. Based on an attacker model similar to ours (see Section **??**) Nilsson, Phung, and Larson developed requirements for a digital investigation. In contrast to our work this paper states guidelines to carry out forensic investigations of car hacking attacks. It does not aim to systematically reveal possible vulnerabilities in automotive ECUs as we do.

Similar to Koscher et al. Hoppe, Kiltz, and Dittmann (2011) did a thorough examination of security threats to Controller Area Network (CAN) buses. Besides a theoretical analysis of possible vulnerabilities and selected countermeasures, Hoppe, Kiltz, and Dittmann provided practical examples in a self-made testing environment. In their tests they demonstrated various attacks on the CAN bus. Those attacks, however, where achieved by sending special commands to a targeted electronic control unit (ECU). No systematic bus off, spoofing of recurring messages, or Denial of Service (DoS) attacks were carried out, as we introduce with this thesis. Furthermore their paper aimed to point out consequences and possibilities of hacking attacks and did not systematically examine vulnerabilities in CAN communication protection mechanisms.

Especially the results published by Koscher et al. led to high attention in the academic as well as the non-academic sector. In 2013 Miller and Valasek (2013) published a detailed analysis about a CAN bus car hack. They showed that they could control almost every functionality when physically connected to the car. As opposed to Hoppe, Kiltz, and Dittmann (2011) Miller and Valasek used more sophisticated attacks and exploited diagnostic protection mechanisms as well. The spoofing attack described in their paper, however, used a simpler approach than ours. Miller and Valasek just flooded the bus with their attack, whereas we are injecting just one, precisely timed, additional message right after the original one. This way, we believe, our attack would not been detected by their proposed detection mechanism.

Similar to the research of Miller and Valasek further CAN hacks followed (Evenchick, 2015; Staggs, 2013; Valasek and Miller, 2014). All those papers followed the same goal, to point out what an attacker can do when connected to the CAN bus.

As, however, the need for a physical connection of CAN hacks was considered to be an unrealistic threat many researchers focused on the exploitation of remote interfaces of a car. Miller and Valasek (2014) carried out a broad survey of remote attack surfaces. They examined dozens of cars based on predefined parameters for their hackability.

Checkoway et al. (2011) demonstrated that they were able to take over control via almost every interface of the vehicle with its outside world (see Section 2.4.1). In 2015 Valasek and Miller (2015) were able to reveal a serious vulnerability in a car's cellular interface which let them completely remote control affected cars just by knowing their IP address.

Besides these hacking attacks, Rouf et al. (2010) used the tire pressure monitoring system (TPMS) for their attacks. Andy Davis (Vallance, 2015) was able to hack a car by sending data via digital audio broadcasting (DAB) radio signals. While all these papers revealed vulnerabilities in a car's I/O channels, we are validating CAN communication protection mechanisms directly on the bus.

Other than these approaches we are focusing not on ways how to hack a car, but we are using CAN attacks as the basis for our systematical tests to efficiently detect possible vulnerabilities of the CAN communication by penetrating the system. Thus, these papers motivated our work and laid the foundations for our hacking attacks.

3.2 Attack Detection and Prevention Approaches

As no common security standard for CAN communication was found so far, we are focusing on the communication mechanisms of current vehicles in this thesis. However, based on the large number of proposed attack detection and prevention approaches and the need to protect the CAN bus in the near future, we designed our application to prepare for possible future adaptations to integrate such techniques easier.

For that reason we want to give a quick overview of current efforts in this field in the following paragraphs:

Based on their own car hacking findings Hoppe, Kiltz, and Dittmann (2011) as well as Miller and Valasek (2013) proposed possible ways of detecting hacking attacks on the bus. They mainly searched for abnormalities and for a higher traffic rate on the network. Other than that Larson and Nilsson (2008) proposed a specification-based approach to detect attacks. With our application we can validate if these approaches can spot and prevent our attacks.

In "Securing Embedded Systems: Analyses of Modern Automotive Systems and Enabling Near-Real Time Dynamic Analysis" Koscher (2014) started a discussion whether detection or prevention approaches are more applicable to secure a vehicular network. While some researchers suggest external means to secure automotive communication systems (Wolf and Gendrullis, 2012; Koscher, 2014; Tao, Antunes, and Aggarwal, 2014; Kammerer, Frömel, and Wasicek, 2012; Cassettari, Fanucci, and Boccini, 2014), other efforts are paid to secure the CAN communication per se by using various types of security mechanisms (Bittl, 2014; C. W. Lin et al., 2013; C.-W. Lin and Sangiovanni-Vincentelli, 2012; Groza, S. Murvay, et al., 2012; Groza and P.-s. Murvay, 2012; Groza and S. Murvay, 2013).

We believe that the modular design and the generic attack definition of our implemented application, makes it combinable with most of the above listed approaches. Thus, these approaches should not influence our work directly.

3.3 Security Testing

As every security mechanism is just as good as its implementation, it still needs testing environments to assure their correct implementation and behavior. At the moment the prevalent approach to validate an ECU are traditional safety and functional tests. Although these tests work properly to validate an ECU's normal functioning, they cannot verify, if the tested ECU is vulnerable to hacking attacks.

Marinescu et al. (2014), for instance introduced a model-based testing framework. Model-based testing can be understood as a way to model the software and/or its environment. Based on these models test cases can be extracted. Even though this approach validates internal functions in detail, it still cannot sufficiently simulate the malicious intelligence of a hacker. We believe that this is also true for other traditional testing methodologies.

In 2008 Armengaud, Steininger, and Horauer (2008) introduced methods to systematically test the FlexRay communication mechanisms of embedded systems. Based on a careful and detailed analysis of the bus traffic and the exploitation of the clock synchronization service they were able to validate the correct behavior of the DUT. As this work exploited details of the time triggered FlexRay protocol, while our work is based on the event triggered CAN protocol, little correlation exists. Furthermore our application focuses on the revealing of weaknesses in higher protocol layers.

Bayer et al. (2014) already dealt with the topic of a systematical execution of automotive security evaluations and penetration tests on a theoretical level. Based on above mentioned examples of car hacking they highlighted the importance of such a system. However, no actual implementation was introduced.

In his master's thesis Talebi (2014) follows a similar approach as Bayer. He studied the field of security evaluations by looking at the CAN bus as a fully simulated environment. The practical tests described in his theses were based on the network simulation software CANoe. In his thesis Talebi used a similar strategy for vehicular security testing as we did. However, while Talebi used a simulation software to carry out his tests, we are actually introducing a software application prototype for the CAN bus that follows this approach of a systematical execution of hacking attacks to reveal vulnerabilities.

Although the papers of Talebi and Bayer et al. already stress the urgent need of a testing infrastructure for a correct and secure CAN communication, there is no actual implementation realized so far.

In October 2015, Jenik (2015) presented a black box testing approach based on a concept called exhaustive fuzzing. Similar to this approach also our implemented application can be used for black box testing; however, instead of fuzzing we learn from the traffic on the bus and execute targeted attacks. We further implemented special attacks that let attackers suppress all communication on the bus or systematically kick an ECU off the bus. Our approach further offers the possibility to examine not only the correct communication of ECUs but also to test potential attack detection mechanisms.

3.4 Conclusion

The application we introduce in this thesis makes use of the findings of modern car hacking research activities. We are creating attacks similar to those described in car hacking research papers like in *Adventures in Automotive Networks and Control Units* from Miller and Valasek (2013). These attacks will then be executed to test the correct implementation of CAN communication mechanisms. As until now no security standard was specified, we prepared for possible future adaptations with the design of our application. In this thesis, however, we are focusing on the communication mechanisms of currently available vehicles.

This attack-based testing approach differs from previously introduced functional or model-based testing frameworks by its *systematic and comprehensive execution of hacking attacks*. Although Bayer et al. as well as Talebi already outlined similar concepts on a more theoretical level, to the best of our knowledge, a fully working prototype of such a testing environment has not been presented so far. Other than Jenik we introduce an application that uses attack techniques to determine the proper working of ECUs and of possible attack detection mechanisms.

4 CAN Communication Tester (CAN-CT)

This chapter introduces the CAN Communication Tester (CAN-CT). The CAN-CT is an application we developed to automatically examine Controller Area Network (CAN) communication protection techniques for possible vulnerabilities.

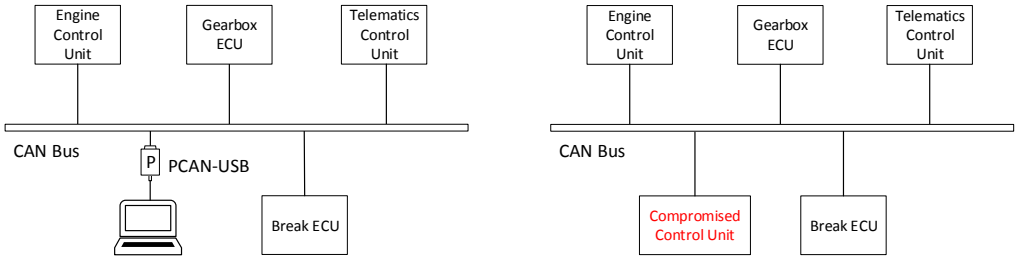
We will present the CAN-CT and its functionality in more detail. Furthermore we will provide a critical analysis of arisen challenges and possible solutions.

4.1 Overall Concept

The inherent goal of the CAN-CT is to check CAN communication protection mechanisms of vehicular electronic control units (ECUs) for possible weaknesses.

As opposed to functional testing of the device under test (DUT) we are targeting a more generic approach. CAN-CT executes hacking attacks on the CAN bus. Various kinds of attacks (see Section 4.5) are implemented, can be parameterized, combined and will then be sent on the bus. Our aim is causing unexpected behavior of the targeted ECU provoked by the executed attacks to unearth flawed implementations of the CAN communications.

Besides the possibilities to carry out hacking attacks a key component of the CAN-CT is to monitor the success of the executed attacks. Based on different requirements for each attack, several techniques are used (see Section 4.6).



(a)Basic concept of the CAN-CT (b)Underlying attack scenario

Figure 4.1: Basic concept of the CAN-CT and its underlying attack scenario

In other words the overall concept of the CAN-CT can be understood as a way to automatically carry out various hacking attacks on the CAN bus by simultaneously watching the reaction of the targeted ECU. Potential implementation flaws in the ECU shall be highlighted by the CAN-CT.

4.1.1 Underlying Attack Scenario

As the CAN-CT aims at detecting weaknesses of the observed ECU by automatically executing hacking attacks, it is based on a realistic attack scenario. For a real world setting such an underlying attack scenario is best described by understanding the CAN-CT as a compromised ECU. This means the CAN-CT has full access to the vehicle's CAN bus and thus, is capable of reading and writing arbitrary CAN messages sent on the bus. As the CAN bus is a broadcast medium deleting messages sent between two nodes is not possible.

Our attack scenario assumes that the attacker already got full access to the CAN bus by, for instance, exploiting a vulnerability in an ECU. This means we are not targeting ways of gaining access to the vehicle's CAN bus but rather we are establishing the assumption that we, as an attacker, already found a vulnerability in an ECU, that has full access to the CAN bus. By taking over this ECU we are now in a position to sent arbitrary messages on the bus. This concept is visualized in Figure 4.1.

Besides the hostile takeover there are also other realistic scenarios conceivable that suit our concept. One for example is the physical connection of a malicious device to the CAN bus via for example the On-Board Diagnostics II (OBD-II) port. This port is typically used for diagnostic sessions to analyze, update or fix a vehicle's ECUs.

As a consequence the attack scenario can be understood as either a compromised ECU or a third-party hacking device physically connected to the CAN bus.

4.2 Software and Hardware

The key component of the CAN-CT is the connection of the application to the CAN bus. For this task we made use of the PCAN-USB[1] adapter. This adapter connects an USB port to a CAN network. With this adapter it is possible to read and write CAN message from a PC.

To communicate with the adapter we used the basic library provided by PEAK-System[2]. This library offered us the core functionality to configure and connect to the adapter as well as to send and receive messages on the CAN bus.

We implemented the features described in the following in the C# programming language.

4.3 Architecture

For the CAN-CT's architecture we focused on a modular design and an easy extensibility. With the realized architecture we tried to avoid any dependencies on the GUI, the used hardware to connect to the CAN bus or the attacking types. We concentrated on a clear software architecture and adapted various design patterns.

[1] see `http://www.peak-system.com/PCAN-USB.199.0.html?&L=1`
[2] see `http://www.peak-system.com/PCAN-Basic.239.0.html?&L=1`

We summarize our goals for the CAN-CT as follows:

1. Easy extensibility and modular design
2. Avoidance of dependencies regarding GUI and USB to CAN adapter
3. Highly performant execution of attacks
4. High accuracy ($< 100\ \mu$s)
5. Great flexibility in the plan of attack definition
6. In-depth analysis of existing CAN traffic

A simplified class diagram of the CAN-CT is depicted in Figure 4.2. One of the core components is the *ControllerManager*. This class is based on the singleton pattern and holds all references to the other controllers. This allows us to access every controller instance without creating any dependencies. Thus it helps fulfilling the first goal.

Besides this the *ControllerManager* also takes care of providing a reference to the used GUI. The graphical user interface (GUI) itself has to implement the interface *IGui*. This offers us the possibility to easily separate the view from our logic (second goal).

The other controllers are in charge of controlling specific functionalities and instances. For example the *HWController* takes care of communicating with the PCAN-USB as well as with the nodes that use this hardware handle to send and receive messages through it. This approach allows us to be more flexible when exchanging the used USB to CAN adapter (second goal).

A node can be seen as either the attacker that consists of functions necessary for executing the attacks or as a simulated node used for the CAN network simulator. These different types are subtypes of the *Node* class.

Similar also the *CANMsg* class is divided into the subclasses *IncCANMsg* (incoming message) and *OutCANMsg* (outgoing message), which in turn is being extend by the *AttackCANMsg* - a message that holds additional information for an attack. The *MsgController* then takes care of creating, maintaining, amending and sending these messages.

The plan of attack is managed by the *AttackController*. This controller treads every kind of attack the same way via its *IAttack* interface. This makes it easy to extend the CAN-CT for possible further attacks (first goal).

Besides these core components the CAN-CT consists of multiple other classes. These classes are needed to carry out supporting tasks like for instance the capturing functionality. Classes that realize this feature, provide sophisticated measures of analyzing the CAN traffic. We can extract information about incoming messages regarding the cycle time of recurring messages, their attributes (ID, data length code (DLC), payload) and if and where additional protection mechanisms like an application layer cyclic redundancy check (CRC) or an alive counter are placed within the message (sixth goal). This knowledge can then be used as the basis of further attacks.

The definition of our attacks can be done either with the CAN-CT or with MS Excel. Here we took care for a larger range of different types of attacks and parameters to define them as well as an intuitive and convenient way to create the plan of attack within the CAN-CT. All generated attacks can then be exported or imported as a MS Excel file (xlsx). Furthermore the implemented C# scripting/dynamic compilation engine provides the user with additional and mere limitless ways to define attacks. With this engine it is possible to generate complex attacks, compile the code directly within the CAN-CT and monitor its success (fifth goal).

In addition to these functions we implemented complex calculations to allow cyclic executions of attacks with a high performance and accuracy (third and fourth goal). Especially the latter is very important as our attacks need an accuracy of a few microseconds. This was realized by using the QueryPerformanceCounter (QPC) API which guarantees high resolution and accuracy of time stamps (<1 μs). This QPC uses the performance counter of the processor to obtain these high-resolution time stamps.

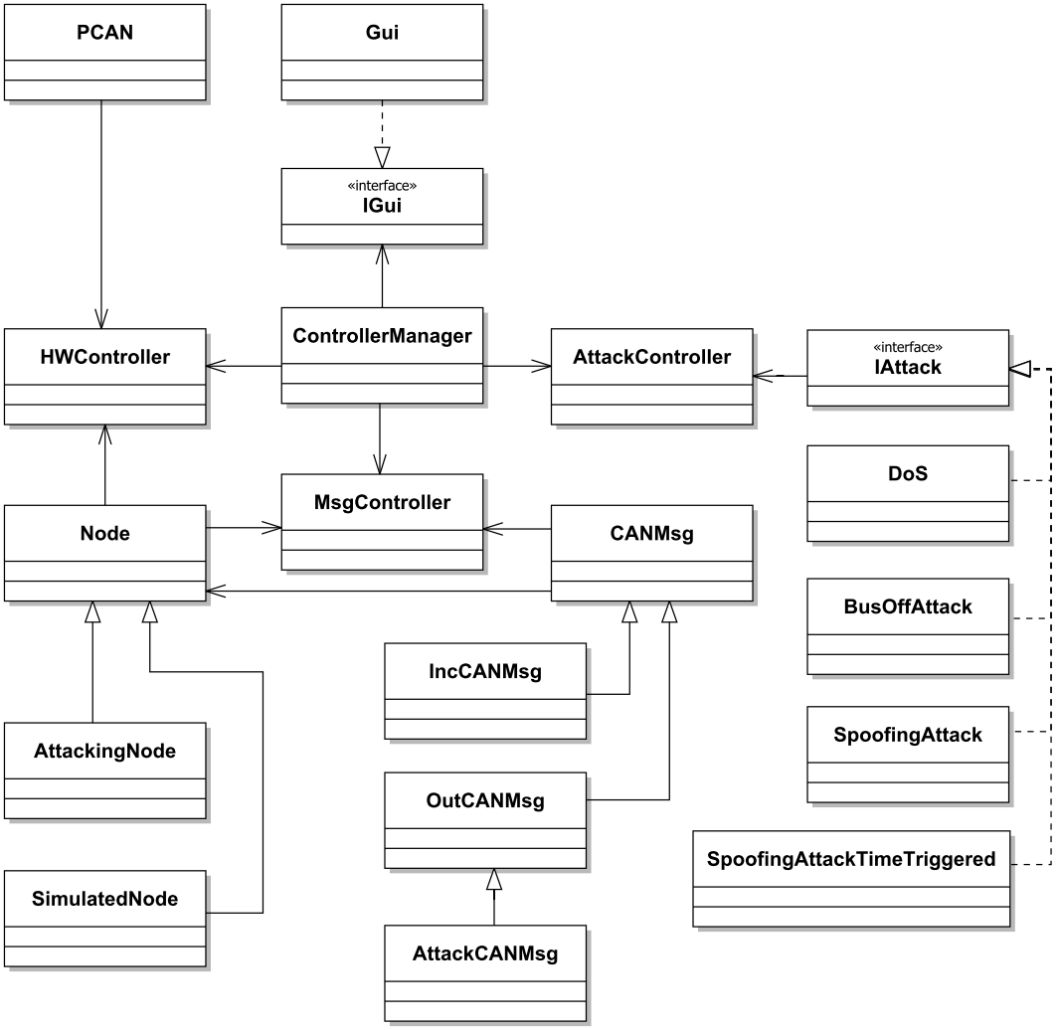


Figure 4.2: Simplified UML2 class diagram of the CAN-CT

4.4 Overview of Application

The CAN-CT can be divided into four main parts:

- CAN Monitor
- Capturing Tool
- Attacking Features
- CAN Network Simulator

Even though the attacking part serves as the central component of the CAN-CT, all other areas are needed to support the functionality or to make the interaction with the CAN-CT as well as the execution of attacks more convenient. In the following sections these parts will be explained in more detail.

4.4.1 CAN Monitor

One of the basic components of the CAN-CT is the CAN monitor. The CAN monitor allows the user to watch all traffic on the bus. All incoming and outgoing messages are listed and extended with important parameters like cycle time and message count (see Figure 4.3). Furthermore it is possible to limit the incoming messages to a certain ID range which offers a comfortable possibility when dealing with a large amount of messages on the bus.

Besides the mere visualization of the events on the bus it is also possible to directly send simple messages from this area.

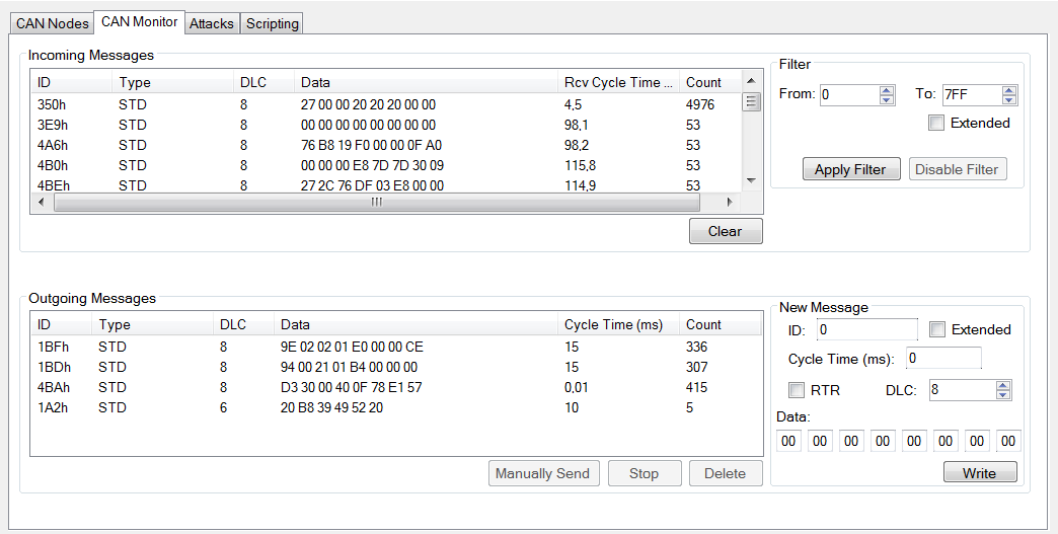


Figure 4.3: Monitoring the CAN bus with the CAN-CT

4.4.2 Capturing Tool

The capturing tool analyzes the incoming messages in more depth. Every received message will be examined for the presence of specific protection mechanisms like application layer CRCs or alive counters. By comparing the content of the currently analyzed message with its corresponding, preceding messages, we can figure out if and in which data byte such protection measures are used. Based on the time difference between the reception of the current message and its preceding, the cycle time is calculated as well.

The results of these analyses are then displayed in the table depicted in Figure 4.4. The gathered intelligence is of great importance for further attacks, as it offers a profound knowledge about possible targets. Each row in the table represents the general description of all messages belonging to one message ID. This information can directly be used to start an attack.

Furthermore it is possible to export the capture data to and import it from Microsoft Excel.

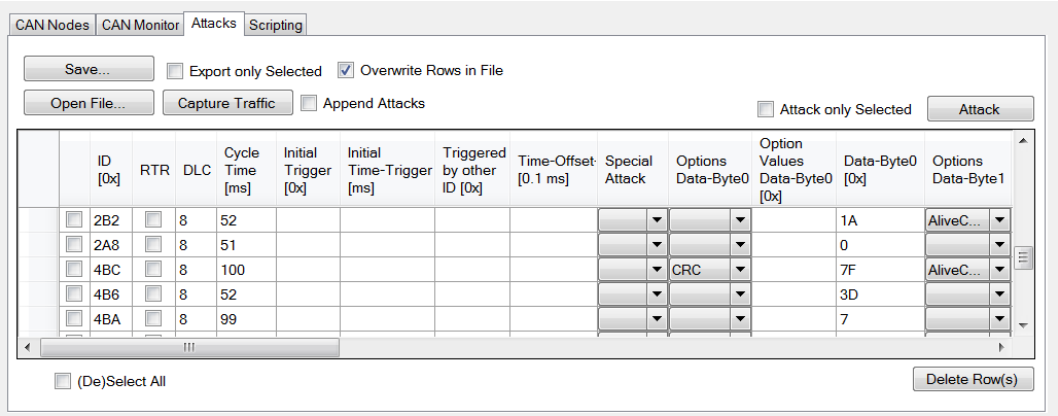


Figure 4.4: CAN-CT's CAN capturing functionality

4.4.3 Attack Features

Based on the information captured with the functionality described above we can define our plan of attack (see Section 2.1.5). We can choose which messages we want to use for our attacks. It is also possible to define attack messages completely by hand; however, the capturing tool provides a more convenient way of specifying these. The CAN-CT further allows for importing and exporting the description of those attacks as Microsoft Excel files (xlsx).

As we can see in Figure 4.5 we can define multiple options for the each message. These options include the DLC, various settings for every data byte (CRC, Alive Counter, randomization, etc.) and parameters which describe the characteristics of the attack per se. The plan of attack depicted in this figure consisted of attacks like a bus off attack, a spoofing attack, a replay attack and a Denial of Service (DoS) attack. Furthermore we specified when the attack was triggered (after a certain time or at the first occurrence of a specific message ID).

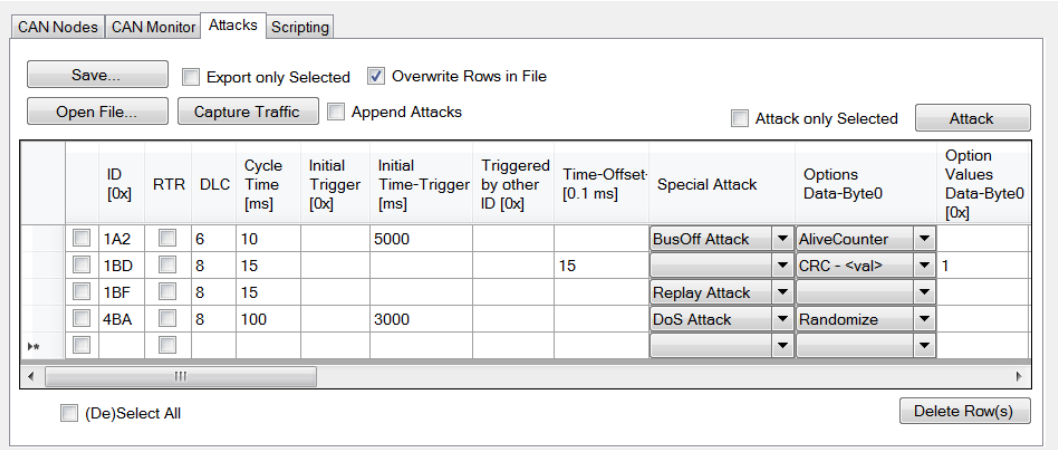


Figure 4.5: CAN-CT's plan of attack definition

We divided these triggers into event-based and time-based triggers:

Event-Based Attack Trigger

An event-based attack is triggered by the receipt of a specified ID. This ID can be either the same as the attack message itself or another one. The latter is especially interesting for attacks that should be executed whenever a certain message is received for instance as a form of a reply or a feedback.

Time-Based Attack Trigger

The second trigger we implemented is a time-based trigger. Here we monitor the target message, calculate its cycle time (the time span between each receipt of a periodic message) and anticipate the time of receipt of the next corresponding message. We then define an offset to this point in time when we want our attack to be executed.

This approach allows us to place our attack for instance directly before or after the target message without any delay. This also offers us the possibility to circumvent a delay caused by the used hardware - the PCAN-USB.

Further Options

To allow for more complex attacks, we implemented additional options that can be used together with any of the attacks described in Section 4.5.

The first option is a start trigger. This option allows us to execute our attack or any subsequent attack just when a specific event occurs. These start triggers are similar to the above described attack triggers. However, this event occurs just once with the goal to start the attack. The attack in turn can use one of the above described triggers for its execution additionally.

The start triggers can be either time triggered, for instance the attack starts after a specified time, or they can be triggered by the receipt of a specific message. The latter for example allows us to start a second attack as soon as the first one was successful. This can be triggered for example by the receipt of a status message caused by a bus off attack.

Additionally every data byte of the attack message can be defined with specific options. These options allow to adapt the message dynamically during the attack. Every data byte can either be specified by a fixed value or it can use any of these options:

- calculation of the application layer CRC
  - correct CRC
  - correct CRC minus a specified value
  - correct CRC plus a specified value
  - correct CRC minus a random value
  - correct CRC plus a random value
- calculation of the alive counter and optionally synchronization with the target
  - correct alive counter
  - correct alive counter minus a specified value
  - correct alive counter plus a specified value
  - correct alive counter minus a random value
  - correct alive counter plus a random value

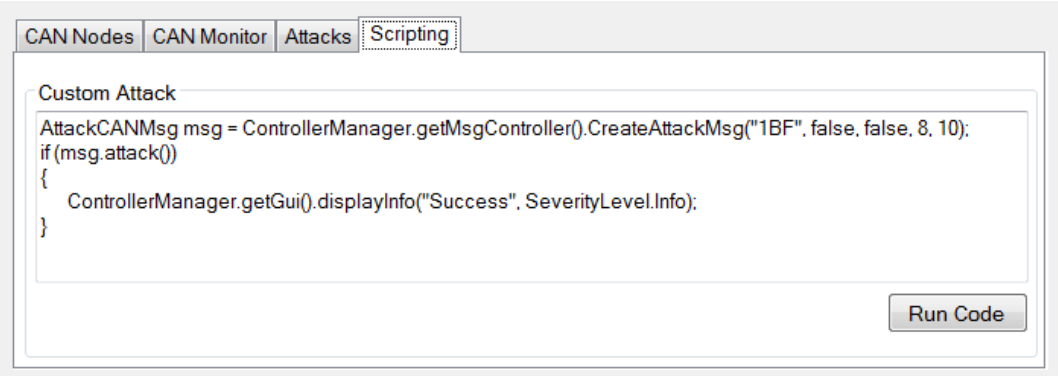


Figure 4.6: Customized attacks via the CAN-CT's scripting engine

- increasing an initial value to an end value by a given step
- decreasing an initial value to an end value by a given step
- randomizing the data byte

C# Scripting Engine

If the above described options are not sufficient we can also program our own attacks with the built-in C# scripting engine. Figure 4.6 depicts some easy sample code which creates an attack message and displays an information on the GUI when the attack was carried out successfully.

4.4.4 CAN Network Simulator

To verify the proper functioning of the CAN-CT we implemented a CAN network simulator. This functionality allows us to use one or more PCAN-USBs additionally to our attacking PCAN-USB adapter. These additional adapters are then connected with each other and used to simulate actual ECUs on the CAN bus. The description of the outgoing messages sent by

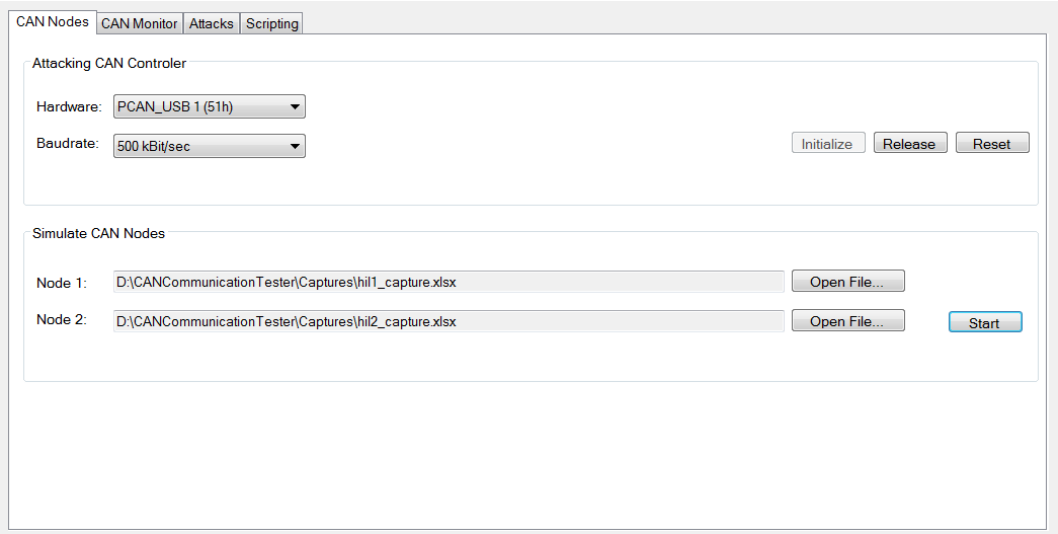


Figure 4.7: USB to CAN adapter configuration and CAN node simulator

such a simulated CAN node is specified in a Microsoft Excel or CSV file. These files use the same format as the captures generated by the CAN-CT or as the plan of attack without attack information. The additional PCAN-USBs are then used to send the messages according to their specification in the provided files.

Thus, these CAN nodes, used to simulate ECUs, are controlled by the CAN-CT as well. These nodes, however, do not run any model of an actual ECU. They cyclically send the messages defined by the user. The attacking PCAN-USB adapter then can execute its attacks based on the traffic generated by these CAN nodes.

Figure 4.7 shows the CAN network simulator and CAN controller configuration tab. This tab allows us to specify and configure the available hardware as well as to choose the file on which the simulated CAN nodes are based.

This setup was used for our laboratory experiments described in Section 5.2.

4.5 Attack Templates

To fully support a simple yet efficient way of testing the safety and security mechanisms of the observed ECU the CAN-CT offers a number of attack templates out of the box. By specifying different parameters various attacks can be carried out.

These attacks can be divided into

- replay and spoofing attacks,
- bus off attacks, and
- Denial of Service attacks.

Besides these attack templates, the user can create additional attacks with the provided C# scripting engine.

In the following sections we will introduce the above mentioned attacks in more detail.

4.5.1 Replay/Spoofing Attack

Replay as well as spoofing attacks are masquerade attacks (see Section 2.4.2). In both cases the attacker claims to be a node other than itself. On a CAN bus, this masquerade is performed by sending messages of a certain ID that is originally sent by another node. The difference between a replay and a spoofing attack is just the content of the respective message that is sent along with this attack.

While the spoofing attack just uses the message ID that belongs to another node and defines parts of the message anew (payload, DLC...), the replay attack copies the message as a whole and sends it back again. This means the whole message of a replay attack is identical to the original message, whereas the spoofing attack has just the ID in common with it.

We used spoofing attacks to falsify messages to manipulate control units. With replay attacks we tried to identify possible weaknesses in the implemented protection mechanisms of an ECU.

Event-Based

In the CAN-CT we implemented two different forms of the replay/spoofing attack. The difference here is the trigger that executes the attack.

The first version we implemented is the event-based attack. This kind of attack triggers the replay or spoofing attack by the receipt of a specified ID.

For a replay attack we normally use the corresponding ID as trigger. As soon as the targeted message arrives, it will be copied as a whole and immediately sent back on the bus. The same is true also for spoofing attacks, with the exception that not the received message will be transmitted but the predefined one.

Time-Based

The time-based attack can be triggered even before the actual receipt of the target message. This implies that the replay attack is not an actual replay anymore. As basis for our attack we take the lastly received message, analyze it for special contents like an application layer CRC or alive counter, and adapt these components accordingly. Due to the adaption of these components the message should be valid. However, as we cannot predict, whether the remaining payload changed, we cannot guarantee the replay attack to be an exact copy of the original message.

Therefore a real replay attack is just possible when it is triggered by the corresponding ID. This circumstance, however, does not interfere with the spoofing attack. As we define the message content for this attack by ourselves, it is not dependable on the previously received message.

4.5.2 Bus Off Attack

The bus off attack we implemented takes advantage of the CAN protocols error handling mechanisms (see Section 2.3.6). It aims to cause transmission errors whenever the targeted ECU sends a message. This approach increases the targeted ECU's error counter until it enters the bus off state. When the bus off state is reached, the ECU is no longer allowed to send nor to receive messages. Thus, it got successfully kicked off the bus.

In more detail, we anticipate the time of receipt of the targeted message with the goal to send our attacking message at the exact same time. This attack then causes a collision on the bus and destroys the originally sent message. The attacked controller notices that the signals on the bus do not correspond with the ones sent. For that reason it sends an error frame and stops transmitting its current frame.

After a certain time the attacked controller tries to sent anew, but the message is overwritten again immediately by our attack. This approach causes the ECU's transmission error counter to increase. This firstly makes the ECU under attack to enter the error active, then the error passive and finally the bus off state. In this state the node does no longer receive nor send messages on the bus.

The message used for this attack has to conform to certain rules. First of all it has to use the same ID as the target. Otherwise arbitration would handle a simultaneous transmission of two frames.

Secondly due to the dominant state of zero on the bus (see Section 2.3.2) the first difference between the attacking and the target frame has to be a binary one in the target frame and a zero in the attacking frame. If it is vice versa the target would overwrite our message and therefore we would be kicked off the bus. This happens as the PCAN-USB conforms to the same rules as the other nodes do.

It makes no difference if the collision occurs in the DLC field or the payload. Therefore the attacking message must be of the same length or shorter than the targeted one. For instance a DLC of 8 in binary code equals 1000 whereas a DLC of 7 is represented by 0111. As this shorter DLC has a zero in the beginning (dominant state) it overwrites a message with a longer one already in this part of the message.

To increase the probability of causing a collision we use a time-based attack. Here our attack actually uses three messages right before and after the anticipated time of receipt. This way we can cope with possible inaccuracies of the targeted node.

4.5.3 Denial of Service Attack

A Denial of Service attack is characterized by the malicious attempt to make a resource unavailable (see Section 2.4.2). On a CAN bus this goal can be achieved by flooding the bus with high prior messages. The consequence of a successful DoS attack is the suppression of a part or of all the communication on the bus. For instance, this attack allows us to completely disable this communication system.

Due to the arbitration process (see Section 2.3.5) the message with the highest priority (lowest ID) prevails. This means if we continuously send messages with any specified ID with the shortest possible intervals, no messages with a higher ID (lower priority) can be transmitted anymore. Thus a resource can be made unavailable.

The content of the attacking message used, is not relevant as long as there is no other ECU trying to send a message with the same ID. Otherwise collisions will occur.

The best option for suppressing all communication on the bus is by using the ID 0h. This ID has the highest priority as it consists of just zeros (the dominant state) and therefore wins in every arbitration.

4.6 Observability

Besides the actual execution of the attack it is also crucial to monitor whether an attack was successful or not. We use multiple observation approaches for this purpose. These approaches can be divided into two main tasks - obtaining and analyzing data.

We used the following approaches to gather information about our attacks:

1. CAN traffic related log files generated by the CAN-CT,
2. CAN traffic capturing with an oscilloscope, and
3. observation via a debugging/calibration tool.

Based on the characteristics of the test case we then analyzed this data by hand or processed it with the CAN-CT. One interesting use case is the analysis of this data with regard to specific CAN messages (see Section 4.6.4).

Besides these mere data gathering techniques the CAN-CT uses following approaches that combine both tasks:

1. Automatic recognition of success based on the characteristics of the current attack
2. Use of (diagnostic) higher layer protocols

All these approaches will be explained in the following.

4.6.1 Log Files

One of the simplest yet not most efficient or accurate ways to determine the success of an attack is the analysis of the CAN traffic log files generated by the CAN-CT. Here we examine the received and sent messages. Based on these log entries we make conclusions about success; for instance the time offset between the target and the attack message or the mere (non)existence of a message can be indicators whether an attack was successful.

4.6.2 Observation via Oscilloscope

The second method to gather data about the attacks is the observation of the bus with the oscilloscope. The analysis of this data may not provide us with details whether the attack was successful in manipulating the targeted ECU or in causing internal errors, but it shows if the attacks were carried out the way they should be. Therefore the analysis with the oscilloscope allows us to examine, if attacks like for example DoS are working properly or if the time offset we specified is adhered to. The latter is important to determine the compliance of the actual time offset with the specified one, even when we have to cope with time variances between the supposed and the actual receipt of the target message caused by inaccuracies of the ECU under attack.

The main difference between the CAN-CT's log files and the oscilloscope observations is the examination of the results on a different ISO/OSI layer. While the CAN-CT provides logs generated by the application itself, the oscilloscope shows the actual physical signals on the bus. This more in-depth analysis of the oscilloscope provides more accurate results without any delays caused by software or hardware components. Furthermore in the laboratory setup (see Section 5.2) it allowed us to verify if the CAN-CT's logging function is working properly.

4.6.3 Debugger/Calibration Software

Another method of collecting information about the attacks is using a debugger respectively a special calibration software connected to the targeted ECU. With such a tool we are able to monitor almost all details about the internal behavior of the attacked ECU. A calibration tool for instance is also integrated in a Hardware in the Loop System (HiL) setup.

4.6.4 Analysis of other CAN Messages

Based on the data gathered by one of the above described methods we can look for messages of particular interest.

Sometimes we can ascertain the success by looking at other related messages on the CAN bus. An example of this is given in Section 5.3. Here we make use of the knowledge that one message provides information about certain values (observation message) while another message allows to set these values (attack message).

Even though this way works pretty well and provides us with accurate details whether the targeted ECU accepted the spoofed messages, it requires significant knowledge about the underlying system. Furthermore very often such a feedback message does not exist.

4.6.5 Attack-Based Detection

Whereas the approaches described above were mere information gathering techniques, which in most cases needed a manual analysis of the data, the CAN-CT can automatically determine the success for certain attacks based on their characteristics. These attacks are:

- *DoS:* Only messages with lower ID than the attack message should be present on the bus.
- *Bus-Off:* After executing the attack no more messages from the attacked ECU should be received.

In both cases the CAN-CT is able to monitor the bus at the same time it transmits messages. This allows us to determine whether our attacks are successful or not.

However, not every attack is suitable to determine its success automatically with the CAN-CT. Spoofing or replaying a message is a simple task with the CAN-CT but proving that this message was accepted by the attacked ECU is not as trivial. Thus, if we want to know whether it was possible to overwrite original values with our attack we have to use a different means.

4.6.6 Higher Layer Protocols

Another promising approach is the use of diagnostic or proprietary higher layer protocols. Many protocols allow for requesting information about the internal variables or states of an ECU. By sending particular messages we can gain knowledge if the attack was able to manipulate the target.

This approach is similar to the approach described in Section 4.6.4; however, instead of taking advantage of the normal communications of an ECU under attack, we are using diagnostic messages to request information or read memory addresses.

Due to the huge number of different higher layer protocols, we decided on not implementing any of those in the current version of the CAN-CT (see Section 4.8.3). Instead we used a generic design of the CAN-CT to be able to handle most requirements for these protocols. This means, although a higher layer protocol is not implemented so far, the user can manually send diagnostic messages with the CAN-CT. A build-in solution is subject to future work.

4.6.7 Conclusion

As it was shown there are a number of ways to determine the success of an attack. Which way is best suited always depends on the attack that should be monitored, the knowledge of the underlying system and the means that are available. Except for an observation via a higher layer protocol all options were used in our practical examination of the CAN-CT.

4.7 Extensibility

As stated in Section 4.3 we focused on a modular design that allows for easy extensibility of the CAN-CT. This decision was made due to the current lack of more sophisticated protection mechanisms in modern vehicular communication networks. Therefore we prepared the CAN-CT for future extensions with regard to the implementation of further attacks.

Due to the communication between the *AttackController* and the actual attacks through an interface (see Figure4.2) we can easily extend the set of attacks. This allows us to prepare for potential changes in the requirements of hacking attacks caused by newly implemented security mechanisms.

Nevertheless, we think that the currently implemented set of attacks already serve as a sound base for hacking future security techniques as well. Based on the characteristics of the particular security mechanism the CAN-CT will have to be adapted, but in our opinion for most attacks the basis functionality of the tool will remain the same. We will still have to attack the target via one of the above described attack mechanisms.

As we outline in the next section we had to cope with limitations of the PCAN-USB. We can exchange the PCAN-USB with another CAN controller in a future version of the CAN-CT. This would provide us with a shorter hardware delay as well as with the possibility to carry out attacks without having to conform to the CAN protocol. To realize such a change, we would have to adapt the *HWController* class. Only minor changes would be necessary for the CAN message classes. All other functionality is fully decoupled and will continue to work.

4.8 Limitations and Solutions

Although we were able to implement almost all initially planned functionalities, we still had to cope with some unforeseen circumstances. These challenges will be described in the following sections.

4.8.1 Hardware Delay

One problem we discovered during the implementation and the testing of the CAN-CT was the high delay between the receipt of a message and the subsequent transmission of our attacking frame (see Section 4.5.1).

With the CAN-CT we measured a mean delay of 1.1 ms for this task. More than 95 percent of this delay was caused by the hardware used to connect to the CAN bus - the PCAN-USB.

To overcome this delay we implemented time-based attacks which anticipate the time of receipt of the next target message (see Section 4.5.1). This allows us to time our messages right before or right after the target message without any delay.

4.8.2 Message Receipt

The PCAN-USB and its corresponding API provided us just with the incoming message as a whole. It was not possible to read the incoming signals bit per bit. This limitation made on the fly manipulations of incoming messages impossible.

To overcome this issue it would be necessary to program a CAN controller ourself or respectively to use another one that is capable of carrying out this task like for instance a Field Programmable Gate Array (FPGA) based solution.

4.8.3 Proprietary Higher Layer Protocols

Although proprietary higher layer protocols are still prevalent in the automobile industry, we decided on not implementing any of these. This decision was made as we did not want to focus on just one or a few manufacturers. However, we created the application in a way that it can handle most requirements of such protocols as we will demonstrate in Chapter 5. Furthermore due to the design of the CAN-CT future extensions can easily be integrated into the existing application.

Besides these proprietary higher layer protocols there exist standards especially with regard to diagnostic protocols as well. These protocols were not implemented so far and are subject to future work.

4.9 Conclusion

With the implementation of the CAN-CT we were able to cover almost all goals set in Section 1.3.1.

We were able to implement a tool that can be used to efficiently test vehicular ECUs against some security (and potentially safety) vulnerabilities. The correct implementation of CAN networking including safety measures like sequence numbers, application layer checksums, time outs, plausibility checks etc. can be examined with the CAN-CT.

All these mechanisms can be tested by carrying out attacks like invalidating, replaying, or injecting messages to the CAN bus. While the last two points can fully be achieved with the implemented replay and spoofing attacks, invalidating a message is just partly possible. We can invalidate messages on the bus, but by causing a collision. A manipulation of the transmitted frame on the fly is not possible with the CAN-CT.

Because we use the PCAN-USB adapter for our attacks we did not have to implement the CAN controller ourself, but it limited our possibilities. We had to find workarounds to overcome the delay caused by the PCAN-USB, we were not able to manipulate messages on the fly, and furthermore all our attacks had to conform to the CAN protocol. This means we could not send random signals or invalid frames on the bus.

These limitations made us face challenges, but for most issues we were able to find a solution. We were still fully able to prove the CAN-CT working as required and achieved almost all goals for this thesis, as we will demonstrate in the following chapter.

5 Results

This chapter states the results of a practical application of the CAN Communication Tester (CAN-CT) explained in Chapter 4. To fully examine the functionality of the CAN-CT we validated it in three different test setups.

The first setup was a laboratory test, in which we installed a Controller Area Network (CAN) bus and simulated traffic according to the test cases. The attacks described in Section 4.5 were verified by oscilloscope. The objective of this experiment was to test all implemented attacks in detail. We were able to show the functioning of the CAN-CT.

In the second and third test scenario we made use of different Hardware in the Loop Systems (HiLs). We showed that the CAN-CT was capable of properly spoofing messages on an actual vehicular CAN bus and hence bringing the system under test (SUT) in a manipulated state. Furthermore we were able to highlight weaknesses in the implementation of mechanisms to secure the communication of the attacked electronic control unit (ECU) on the CAN bus.

5.1 Overview of Test Systems

To verify the general functioning of the CAN-CT and to prove its applicability we examined it in multiple setups. We carried out the first task in a laboratory setup.

For the laboratory setup we created our own network based on CAN consisting of three PCAN-USB adapters. All those adapters were controlled by the CAN-CT. The CAN-CT as a network simulator on the one side generated traffic, which simulated actual ECU communication, while on the other side it attacked one of these CAN controllers. With this experiment we verified that all implemented attacks work properly. A more detailed insight will be provided in Section 5.2.

After showing the general functioning of the CAN-CT we proved the applicability and importance of such a tool for actual vehicular ECUs and communication systems. For this purpose we decided to make use of HiLs (see Section 2.1.2).

We used two different HiLs to verify the CAN-CT. AVL LIST GmbH[1] provided us with both HiLs and offered great support for our tests. The first HiL was a heavy duty vehicle HiL while the other one was a passenger car HiL. Each HiL consisted of at least one real hardware ECU (see Figure 5.1). The detailed architectures of those systems are described in Section 5.3.1 and Section 5.4.1. All details regarding make and model of these vehicles and manufacturers are confidential. However, all used ECUs are state of the art.

To validate the CAN-CT we applied it to real life ECUs, like turbocharger and engine control units, to demonstrate that it can find weaknesses in the implemented communication techniques. We chose the tested ECUs to efficiently support the proof of concept of the CAN-CT. The targeted ECUs have interesting characteristics like for instance, possibilities to observe the success of an attack by monitoring certain CAN messages (see Section 5.3) or additional safety/security mechanisms (see Section 5.4). We explicitly want to mention that to the best of our knowledge these targets do not consist of any fundamental differences to any other vehicular control units.

Furthermore all measurements were carried out at a bit rate of 500 kbit/s - the standard bit rate for most vehicle CAN buses. This implies that one bit takes 0.002 ms (2 μs) on the bus.

[1] see `https://www.avl.com/`

(a)Photo of the heavy duty vehicle Hardware in the Loop System setup

(b)Connecting the CAN-CT via the PCAN-USB adapter

Figure 5.1: Pictures of the heavy duty vehicle Hardware in the Loop System setup

5.2 Laboratory Experiments

The goal of the laboratory experiments was to test the proper functioning of the CAN-CT. Thus, we aimed to answer the following questions:

- Can we analyze and learn from the traffic on the bus?
- Do all attack templates, options and parameter settings work as expected?
- Can we use the CAN-CT to cause an ECU under attack to enter the bus off state?
- Can we suppress all/part of the communication on the bus?
- Does the CAN-CT's attack logging functionality comply with the physical signals on the bus?

To answer these questions we examined the following attacks in detail:

- event-based replay/spoofing attack,
- time-based replay/spoofing attack,
- bus off attack, and
- Denial of Service attack.

Furthermore we tested functionalities like the monitoring, capturing and simulation features of the CAN-CT.

Using an oscilloscope we were able to check if the spoofed messages are sent as expected. Furthermore we could validate the compliance of the log files generated by the CAN-CT with the actual signals on the bus. The key task was the manual inspection of the electric signals on the CAN bus. Here we could examine the order of the arrival of the messages as well as the time offset between the target and the attacking message(s).

5.2.1 Test Setup

As stated above, for the laboratory experiments we made use of three PCAN-USB adapters, which were described in section 4.2. We used two of those adapters as "normal" CAN nodes to simulate actual ECUs. These nodes can be understood as any vehicular ECUs. The third adapter served as the attacking node. In our attack scenario (see Section 4.1.1) this node can be seen as a compromised ECU.

We then carried out the validation of the results in multiple ways. First of all we logged the sent and received messages with the CAN-CT. To verify these logs we also made use of the log functionality of the PCAN-View[2] application. However, as these logs were just software-based and therefore may not fully reflect the physical signals on the bus, we used an oscilloscope to monitor the actual events on the CAN bus. This approach provided us with a more accurate way of validating the CAN-CT. For instance we could see the exact number of bits between two messages or the physical signals during a collision on the bus.

Architecture

The architecture of this setup is depicted in Figure 5.2.

As stated above, we used one PCAN-USB adapter as a compromised, malicious ECU. The tasks of this attacker can be divided into three areas:

1. learning the communication on the bus,
2. carrying out attacks based on the previously gained knowledge, and
3. optionally observing the success of the attack.

We carried out all these tasks as well as the simulation of the other two CAN nodes with the CAN-CT. Furthermore also the initial validation of the results - the logging - was done by the CAN-CT. By connecting the oscilloscope to the CAN bus' high and low wires we were able to fully monitor the events on the bus and prove the correctness of the CAN-CT.

[2] see `http://www.peak-system.com/PCAN-View.242.0.html?&L=1`

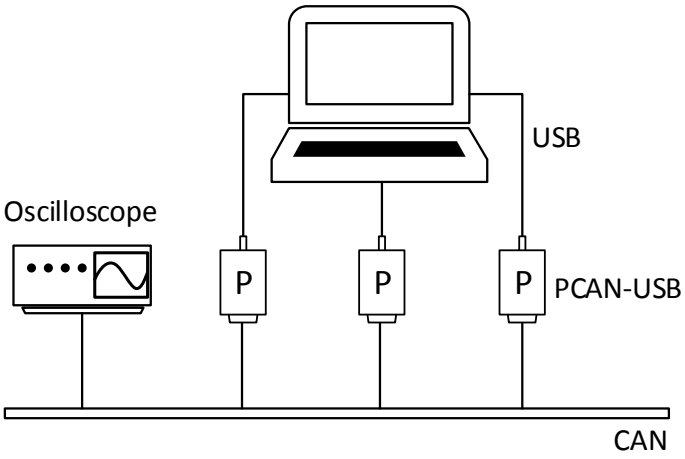


Figure 5.2: Architecture of the laboratory experiments test setup

CAN Communication Characteristics

The simulated traffic on the CAN bus was generated to fit the needs of the current test case. It did not reflect actual CAN messages captured in a vehicle. For most test cases neither the actual message format nor the structure of the payload was relevant. The actual payload of a message has no impact on the success of an attack and thus, does not interfere with the proper functioning of the CAN-CT in a real life scenario. Therefore we used short IDs as well as a simple payload for most messages within this setup. This decision was made to focus the attention on the essential characteristics.

In most test cases the target and the corresponding attack frame followed the frame structure depicted in Figure 5.3 and Figure 5.4. Both messages used the ID 00Dh and a data length code (DLC) of eight. The data of the target frame consisted of eight bytes with the hex value AA. The attacking frame used all zeros as payload. The additional, inverted bits in some parts of the frames are stuff bits. Please refer to Sections 2.3.3 and 2.3.4 for a description of the encoding as well as of the other parts of the message.

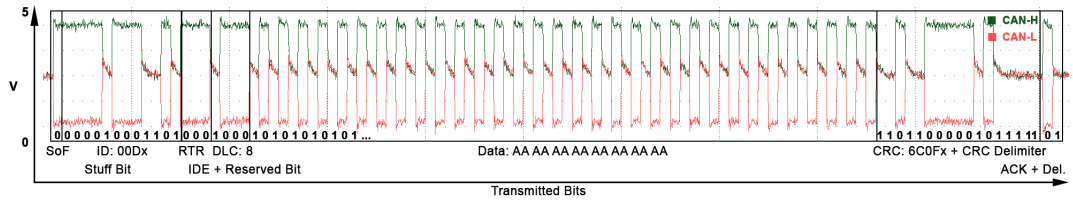


Figure 5.3: Frame structure of the targeted CAN frame used in the laboratory setup

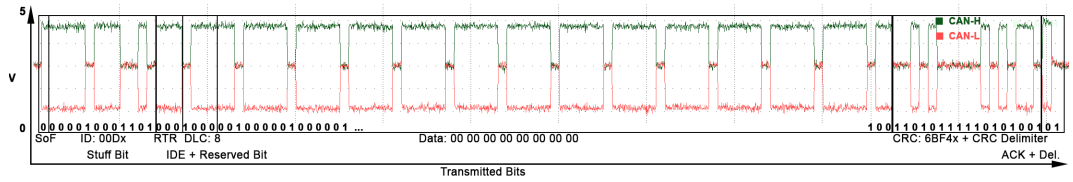


Figure 5.4: Frame structure of the attacking CAN frame used in the laboratory setup

Detecting a Successful Attack

As described in Section 4.6 there exist a number of possible ways to ascertain the success of an attack. To prove the correct and proper functioning of the CAN-CT we decided on using the following means of observation:

- log files,
- attack-based detection, and
- observation via oscilloscope.

By choosing these approaches we focused on the lower layers of the network protocol as well as on the characteristics of the particular attack. Other means introduced in Section 4.6 were not used as all of the CAN nodes were simulated and no actual vehicular communication took place in this setup.

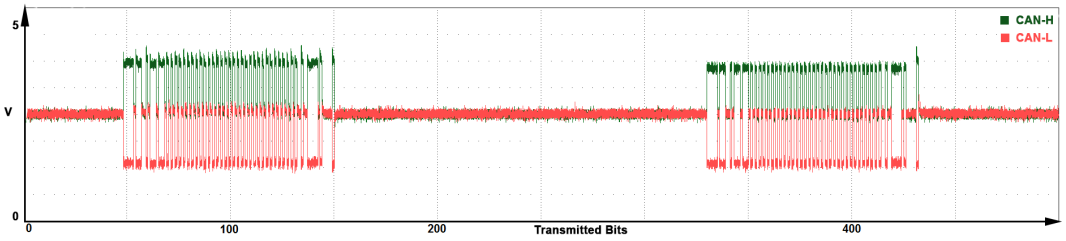


Figure 5.5: Event-based replay attack observed via oscilloscope in the laboratory setup. Both frames depicted are identical. The left frame represents the original message, whereas the right frame is the replay attack message.

5.2.2 Event-Based Replay/Spoofing Attack

For the purpose of testing the correct functioning of the CAN-CT we categorized attacks based on their underlying functionality. As replay and spoofing attacks just differ in their payload we do not differentiate between them in this section. Instead our focus is on the time it takes between receiving the respective message and placing the attack on the bus. This time offset examination allowed us to determine the performance and accuracy of our attacks.

The mean time between the receipt of the targeted message and the placement of the attack message is 1.3 ms at a sample size of 104 measured with the oscilloscope. This correlates also to the measurements done by CAN-CT itself. We noted an average delay of 1.1 ms at a bit rate of 500 kbit/s. A more detailed analysis of this delay highlighted that more than 95 percent of this delay were caused by the PCAN-USB whereas the internal processing of the CAN-CT took just around 0.05 ms.

The calculated standard deviation of 0.337 ms of this delay was mainly based on limitations of the used system. Due to the communication via USB and the nondeterministic characteristics of the system's scheduler we could not achieve more stable results.

Figure 5.5 visualizes a measurement of the replay attack with a short delay of 0.36 ms. The first frame is the targeted frame which triggered the attack within the CAN-CT. After the receipt of this frame (identified by its ID) the CAN-CT copied the message content and sent it back again on the bus. The results are two identical frames on the CAN bus.

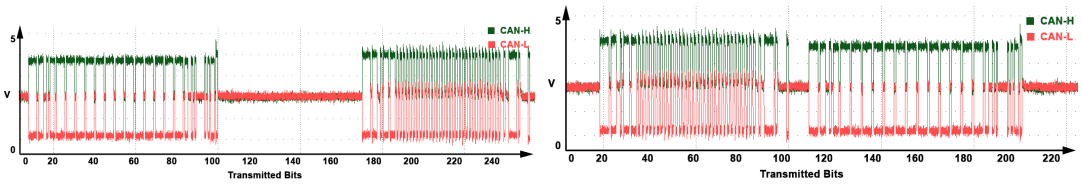
As shown with the CAN-CT we were able to spoof or replay messages triggered by the respective receipt. Even though the execution of the attack took around 1.3 ms the attacks per se worked as expected.

5.2.3 Time-Based Replay/Spoofing Attack

Similar to the above attack also this test case does not distinguish between replay and spoofing attacks. The main advantage of this kind of attack, in contrast to the event-based attack, is that it is possible to overcome the delay caused by the PCAN-USB.

As described in Section 4.5 this attack learns from the incoming messages and anticipates the time of receipt of the next message. Based on this feature it is possible to add or subtract an arbitrary time offset to this point in time. This means, we try to send the message sooner, later or at the same time as the original message.

To validate the correct time calculation of this feature the tests consisted of various time offsets. We were able to show that the attack works properly. This can be seen in Figure 5.6. The left figure depicts a time offset attack of -0.1 ms. This allows the attacker to place the malicious message right before the original. The right figure demonstrates the attack with a time offset of 0 ms. This means the attack will either be received just before the original message or right after it, as shown here. Furthermore it is also possible that this attack will cause an collision on the bus.



(a)Time-based spoofing attack (left frame) with an offset of -0.1 ms

(b)Transmission of target (left frame) and attack message (right frame) at the same time (time offset of 0 ms)

Figure 5.6: Overview of time-based spoofing attacks with a different offset in the laboratory setup

The problem with these time offset attacks close to the original receipt is, that it is quite likely that such collisions occur on the bus. Generally the CAN protocol's arbitration mechanism should take care of collision avoidance. As mentioned in Section 2.3.5 arbitration just works for the ID-part of a message. The placement of two or more messages with the same ID but different payload data at the exact same time will result in a collision.

Therefore, if our goal is to spoof messages and not to destroy other frames, these collisions can hamper the success of the attack. However, as we detected in the real world test setups (see sections 5.3.2 and 5.4.2) the ECUs under attack accept messages until a predefined time out window is reached. We discovered that this time out window is large enough to use a time gap of around 1 ms between the target and our attack message to still be able to successfully spoof messages.

In this test setup we observed that the PCAN-USB adapters are not perfectly accurate. The expected receipt of the target message differed by up to 0.05 ms. This made it possible to use an automatic adaption of the calculated point in time (see Section 4.5.1). As a consequence, the desired time offset value can not be satisfied at all times.

5.2.4 Bus Off Attack

The bus off attack aims to directly cause a CAN controller to fail. By anticipating the receipt of a recurring message, the CAN-CT tries to send its message with the same ID as the target at the exact same time. When carried out correctly this leads to a collision on the bus.

We were able to show that the targeted CAN controller can be put into the bus off state immediately after executing the attack. The CAN-CT anticipates the time of receipt of the target message, sends the attack message at the exact same time as the target and hence causes a collision on the bus.

However, overwriting a value on the bus only works for values that are smaller than the original value because 0 is the dominant state (see Section 2.3.3). We found out that this problem can be circumvented by using a different DLC which leads to an earlier collision and hence allows for spoofing higher values as well. This means we do not have to prevail by transmitting zeros with the payload but instead we overwrite the original message already at the DLC. As we will show in Section 5.3.6 spoofing a message with a different length does not lead to rejection by the ECU.

For this test we used the same frames as introduced before (see Section 5.2.1). In Figure 5.7 these frames are depicted once again. In comparison with the active error frame caused by the bus off attack, this figure offers an overview of why and where the attack occurs. Here we can see that right after the message header - when the payload starts - the collision occurred. This is illustrated by the error frames depicted in this graphic. We can distinguish between two separate error frames. The first is sent by the targeted CAN controller that detects a deviation of the signals sent and read back (bit error). The second error frame then gets sent by the other CAN nodes on the bus, that discover more than five consecutive bits of equal value (stuff error).

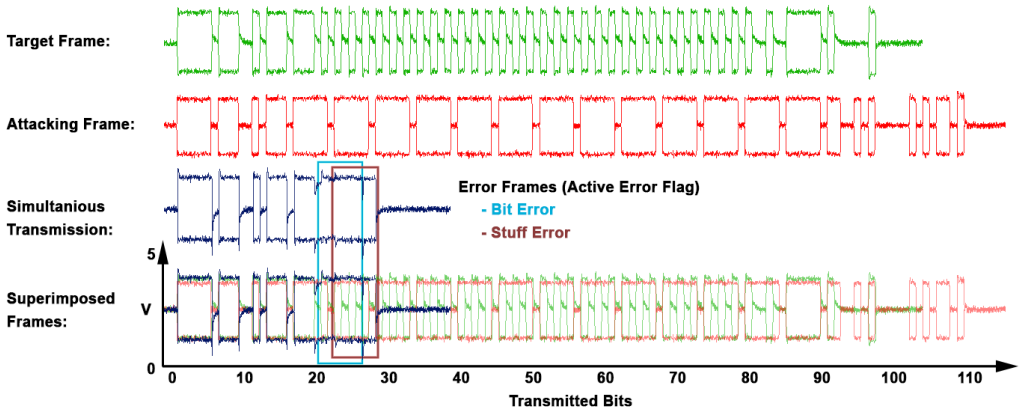


Figure 5.7: Overview of CAN frames that are present during a bus off attack. The first two frames reflect the target and the attack message. When they are sent simultaneously, a collision occurs and an error frame will be sent. The last trace depicts a symbolic overlay of all these frames.

Whenever such an error is detected, the affected CAN controller increases its error counter (see also Section 2.3.6). After reaching a counter value of 128 the controller enters the error passive state. Now for every collision the controller sends a passive error flag (six recessive bits) which do not destroy the traffic on the bus (see Figure 5.8). Instead we just see the general characteristics that happen during an arbitration process. Additionally the internal error counter gets increased further. After raising the counter above 255 it will finally enter the bus off state. The absence of message IDs belonging to the target allows us to detect this state.

With this attack we are able to put the targeted controller into the bus-off state within seconds. This can be interesting, for example, when we want to occupy IDs owned by this controller to spoof messages in the future.

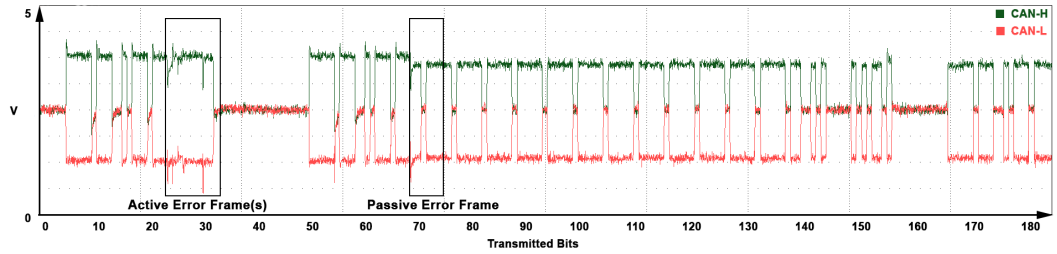


Figure 5.8: Active and passive error frames captured during a bus off attack in the laboratory setup. The passive error frame starts at the exact same bit position as the active error frame. We detect the passive error as the voltage level decreased. This means just one ECU (the attacker) keeps sending bits in dominant state. The target sends recessive bits to signal the error. These bits, however, are not visible in this figure as they are overwritten by the message of the attacker immediately.

5.2.5 Denial of Service Attack

The last kind of attack we verified within the laboratory test setup was the Denial of Service attack. Here we examined whether we can suppress all communication with higher IDs than the attack message. Due to the arbitration process of the CAN protocol, only messages with a lower ID than the one that is flooding the bus should prevail.

We examined this behavior again with the oscilloscope and the log files created by the CAN-CT. We could determine that the CAN-CT completely flooded the bus. Between two messages just eleven recessive bits remained on the bus. This is the minimum between two frames defined in the protocol (1 bit acknowledgment delimiter, 7 bit end-of-frame, 3 bit interframe spacing).

Therefore, we were able to demonstrate the intended behavior. As it is depicted in Figure 5.9 the attack was carried out with an attacking frame of ID 00Dh and the payload of all zeros. The only remaining communication on the bus were frames below this value. Here we can see a message with the ID of 00Ah prevail in the arbitration. All other communication with a higher ID is blocked due to our attack.

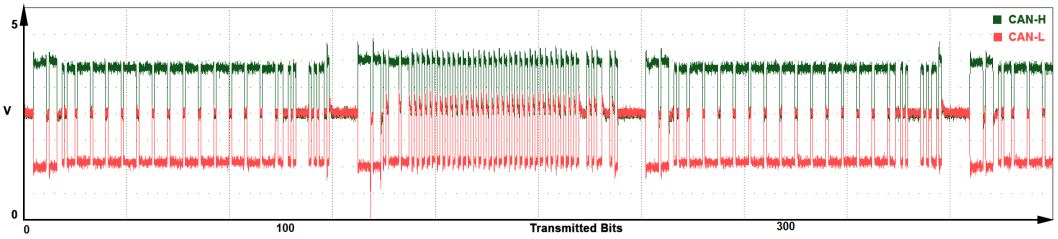


Figure 5.9: Denial of Service attack with an ID of 00Dh in the laboratory setup

5.2.6 Conclusion

Summing up we demonstrated that all implemented attacks work properly within the bounds of the PCAN-USB. We showed that although there are limitations, like delays and inaccurate transmission times, caused by this adapter, we can find ways to circumvent those.

As, however, the cycle time of a message depends on multiple factors like the respective CAN controller or the bus load an exact prediction of the target message is not always possible. Nevertheless as we showed during the bus off attacks this precision is still accurate enough to cause collisions on the bus. Based on this observation we think that every implemented attack can be successfully carried out with the CAN-CT.

For a real world scenario this means that the CAN-CT offers

- multiple ways to spoof a message,
  - event-based attacks,
  - time-based attacks,
  - bus off attacks,
- possibilities to kick an ECU off the bus and take over its IDs in further consequence, and
- to suppress all/part of the communication on the bus.

However, not all kind of attacks are equally effective for all use cases. Therefore based on the particular behavior we want to test and the aim we are trying to achieve, we have to choose the most suitable attack.

5.3 Heavy Duty Vehicle

To prove that the CAN-CT works properly in real world scenarios, we made use of a HiL system. In this test setup the HiL-System simulated a heavy duty vehicle. A HiL system combines actual hardware with simulated components. The goal is that the tested hardware behaves exactly the same as it would in a complete vehicle.

For our purpose this setup suits us perfectly as we have a real vehicular bus system as well as actual ECUs to test. The main advantage here is that these ECUs are all monitored by the HiL setup. Hence, it was possible to read internal variables and states of the device under test (DUT).

With this setup we aimed to answer the following questions:

- Can we use the CAN-CT to test real world ECUs or are there differences to the laboratory test setup?
- Can we circumvent current communication protection mechanisms with the CAN-CT?
- Can we use the CAN-CT to cause an actual ECU under attack to enter the bus off state?
- Can we suppress all/part of the communication on a vehicular CAN bus?
- Can we reveal weaknesses in the communication mechanisms of the targeted ECU?

Thus, the goal of this setup was to verify the implemented attacks for their real world applicability and impact. We aimed to apply the CAN-CT to reveal weaknesses in the communication mechanisms of the targeted ECU by executing the following attacks:

- spoofing attacks (time- and event-based),
- bus off attacks, and
- Denial of Service attacks.

By applying these attacks we monitored protection mechanisms like the time out window, plausibility checks or the general behavior of the ECU in certain situations (for instance during a Denial of Service (DoS) or a bus off attack).

5.3.1 Test Setup

The setup of this HiL system consisted of three separate CAN buses which connected a dosing control unit (DCU), an engine control unit, an urea tank, a variable geometry turbocharger system (VGS), and a controller for the VGS (see Figure 5.11).

After a comprehensive analysis of this setup we decided to bring the VGS into focus of our examinations. The VGS and its corresponding controller were very promising as they offered a comfortable feedback channel for the observation of our success.

In the following paragraphs we introduce these components in more detail.

Architecture

The VGS is the turbocharger of the vehicle under test. Its main task is to regulate the boost pressure in the combustion chamber. The amount of pressure generated depends on internal computations within the VGS Controller and is mainly influenced by the engine speed. It is determined by the position of the vanes of the VGS (see Figure 5.10). The result, that is the command of how much boost pressure should be generated (the position of the vanes), is then transferred via CAN from the VGS Controller to the actual VGS - an actuator that consisted of a CAN controller as well.

After an internal plausibility check of the received values, the VGS moves its vanes to the position that is requested. The actual actuator position is then sent back on the CAN bus in form of a feedback.

By taking advantage of this communication mechanism we can influence the performance of the engine. Furthermore a continuously too high pressure can cause serious damage to the VGS or even to the engine. As the VGS is just connected via the CAN bus to its control unit, it does not know, if the engine, or any other component, signals a dangerous state, when we suppress or falsify those messages.

(a)VGS with partially closed vanes (normally at a low engine speed). Higher boost pressure will be generated.

(b)VGS with fully opened vanes (normally at a high engine speed). Lower boost pressure will be generated.

Figure 5.10: Overview of the VGS at high and at low engine speeds[3]

Figure 5.11 visualizes this setup. The grayed out components were not directly attacked during these tests as we focused on the engine CAN bus. This bus system connects VGS, its Controller as well as, the engine control unit. The engine control unit per se was not used for spoofing messages but it was tested for bus off and DoS attacks. By making use of the HiL PC and its calibration access to the ECUs we were able to monitor the internal states of the targeted hardware devices at all times.

The *AVL PUMA Open*[4] and the HiL system itself is needed for the simulation of the rest of the vehicle as well as for modeling the respective environment. The *AVL PUMA Open* is the global industry standard for testbed automation and serves as the basis for virtual environments and HiL testbeds. With these tools various driving maneuvers as well as system operations can be simulated. For our tests these components were needed to run the simulation.

[3]see `http://www.autozine.org/technical_school/engine/Forced_Induction_2.html`

[4]see `https://www.avl.com/-/avl-puma-open-automation-platform`

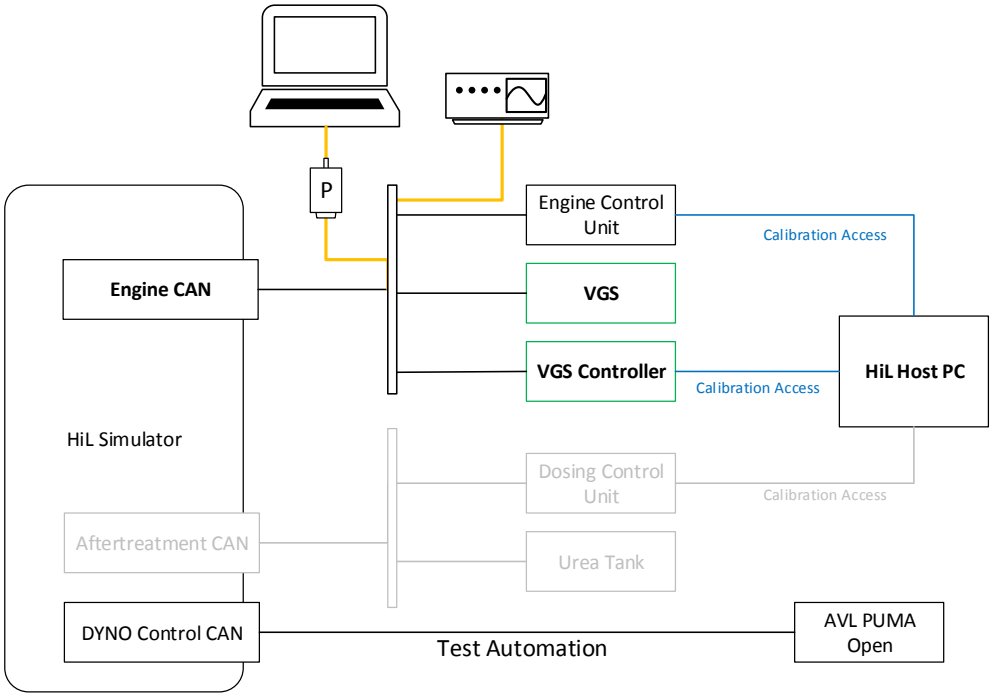


Figure 5.11: Architecture of the heavy duty vehicle HiL system setup

As depicted in Figure 5.11 we connected our tool via the PCAN-USB adapter (bold, yellow lines). The monitoring of the test cases was then done with the HiL PC as well as via the oscilloscope.

CAN Communication Characteristics

Our main focus was on the communication between the VGS and its controller. We made use of the two messages depicted in Figure 5.12.

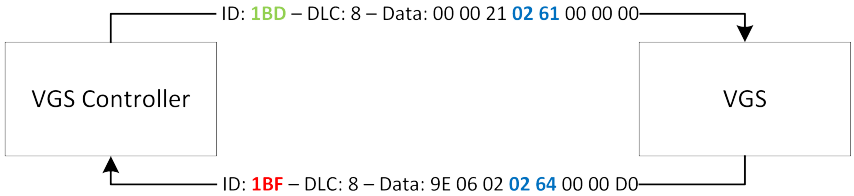


Figure 5.12: Example communication between the VGS controller and the actuator in the heavy duty vehicle HiL system setup. The bold, blue values represent the target respectively the actual position of the VGS

The first message (ID: 1BDh) was sent by the VGS controller every 15 ms and consisted of information about different states as well as the target position of the VGS. As the states contained within the message just defined basic characteristics which were not relevant for our tests we will not detail these here.

The bold values shown in the figure are the ones that are of special interest to our tests. These values specify the requested target position calculated within the controller. The hex value 0261 corresponds to the decimal value 609 which divided by ten gives the percentage 60.9. This percentage stands for the requested position of the VGS.

The second message (ID: 1BFh) gets sent back by the VGS actuator itself. It contains, besides other status information, the actual position of the VGS. Normally this should match the target position within a small tolerance. In this example we have the hex value 0264 which represents 61.2 percent. We can see that this value differs from the target position just by 0.3 percent.

As, however, the heavy duty vehicle does not use any safety/security measures besides internal plausibility checks and bus load monitoring, additional techniques like an ancillary application level cyclic redundancy check (CRC) or alive counter are not used in this setup.

Detecting a Successful Attack

We carried out the monitoring of the success of our attacks mainly via the back channel messages of the VGS actuator. Together with the HiL PC we were able to watch internal variables that were relevant for determining error states or bus off conditions.

Besides these techniques an oscilloscope as well as the CAN-CT's ability to observe the success and create log files were used.

5.3.2 Message Spoofing

The first test case aimed to examine whether we can successfully spoof messages on the bus. For that reason we tried to falsify the messages with the ID 1BDh to change the target position of the VGS.

The main task in this context was finding out how the VGS actuator determines whether a message is valid and should be accepted. We found different ways to reach this goal. However, not all approaches were equally efficient and successful.

The first option was kicking the actual controller for the VGS off the bus. However, as we will explain in detail in Section 5.3.3 this attack did not meet our expectations.

The second possibility was to make use of an event-based attack (see Section 4.5.1). We triggered our attack on the detection of the target message (1BD). As stated in Section 5.2.2 this leads to a mean delay of around 1.3 ms.

Although this strategy already yielded consistent results, as a next step, we tried using the time-based attack mechanism of the CAN-CT as well. Here we discovered that the actuator always accepted the last incoming message as valid. This implied that the previously received messages will be overwritten in the receive buffer of the targeted ECU before they are actually used. However, as we detected that the time variance between the actual and the supposed receipt of the original message differed significantly

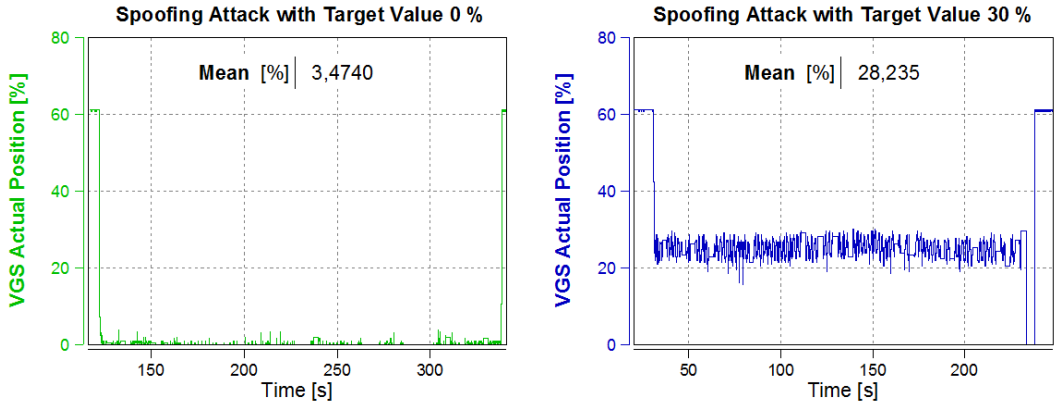


Figure 5.13: Time-based spoofing attacks with an offset of 1.5 ms and a target value of zero respectively 30 percent in the heavy duty vehicle HiL system setup. The originally sent target value was 61 percent. During our attack we successfully blocked this value and overwrote it with our target value. At the moment we stopped the attack the value was set back to the initial value.

(around 1.5 ms) it was not trivial timing the manipulated message right after the original. We were always susceptible to the threat of being overwritten by the real CAN controller or of causing a collision. Thus we found that a time offset of 1.5 ms is the most suitable time offset value for this kind of attack.

Figure 5.13 shows the time offset attack with this delay which sets the target value to zero respectively to 30 percent. The fluctuations in the charts are based on internal calculations of the VGS. We can see that the originally sent target value of 61 percent is successfully blocked by our attack and the actual values are set according to our attack.

In summary we can say that the time offset attack of 1.5 ms as well as the event-based attack achieved consistent success. Therefore we were able to prove that we can manipulate an ECU over the CAN bus with the CAN-CT.

5.3.3 Resilience Against Bus Off Attacks

As already mentioned in the previous section the bus off attack was just partly successful. On the one hand we were able to prove that we can cause a bus off error of the targeted ECU, but on the other hand this attack is not suitable for message spoofing. We can kick the target off the bus - at least for a certain amount of time - but as the tests showed the recipient of those messages also noticed the bus error and thus, did not accept the spoofed messages either in most cases.

This circumstance is illustrated by Figure 5.14. We tried to use the bus off attack to, on the one hand, cause the ECU under attack to enter the bus off state and on the other hand, to take over this message ID and set the position of the vanes to 30 percent. In just a few points in time the VGS actuator accepted the spoofed value that was sent along with this attack. In most cases a bus error was detected. The VGS then ignored all other messages with this ID and set the value to zero percent to signal an error state.

The fact, that the VGS noticed the bus error, did not allow us to spoof messages along with this attack; nevertheless, we were able to make the ECU under attack unavailable for all communications. During our tests we were able to highlight that we could turn the targeted ECU into a bus off error state whenever we wanted just by causing collisions each time the respective ECU tried to sent.

However, as opposed to the laboratory setup, both tested ECUs (VGS, engine control unit) did not give in that easily. The target can be put into the bus off state but it resets its error counter on a regular basis. The first time it is attacked it is reseted within a few hundred milliseconds. After a longer time executing this attack these periods increased. The longest bus off state measured was over three seconds. The problem, however, is, that once the target is able to place a correct message on the bus these periods are decreased again.

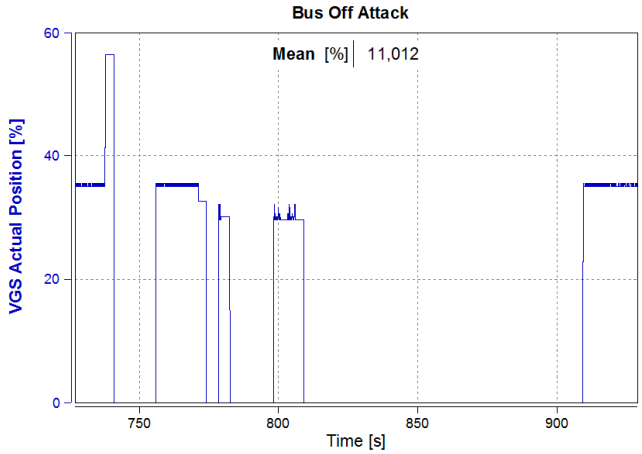


Figure 5.14: Bus off attack with an DLC of 5 and the target value set to 30 percent in the heavy duty vehicle HiL system setup. The position before and after the attack was 35 percent. During our attack the position fell to zero percent as a bus error was detected (error state). Just at the beginning the targeted ECU was able to prevail (between 760 s - 770 s). Then we were able to spoof our target value for a short time (between 780 s and 810 s). After that neither we nor the actual ECU could transmit the target value anymore. However, the target was successfully kicked off the bus.

Summing up, we were able to prove that we can turn all tested ECUs into a bus off state as well as to cause the internal reset counter of the target to overrun. All these attacks conform to the CAN protocol and did not need any modification of the CAN controller. Combining the bus off attack with a spoofing attack, however, did not work as expected. The target can be made unavailable on the bus, but other ECUs might notice this error state as well.

5.3.4 Behavior during Denial of Service Attacks

To test the VGS during a Denial of Service attack we chose the ID 1BEh as the attacking message to flood the bus. The reason was that this ID lies between the two IDs in charge of the VGS control. This way we can test if we can suppress part of the communication while other messages can still be transmitted.

The target position of the VGS (ID 1BDh) should still be received by the actuator but the feedback message (ID 1BFh) is blocked on the bus. This behavior is illustrated in Figure 5.15. Here we can see, that, when the attack started at around 90 seconds, no more feedback messages were received (no fluctuations anymore). Just in the middle of the attack the actuator was able to prevail over the attack.

During the DoS attack we were able to observe no error states or failures of the monitored devices. This implies that the ECUs continue their work with the lastly received values. However, when all communication on the bus is blocked we would have expected the ECU to enter a safety mode or at least display a warning. Yet we were not able to observe any of those. In a real world hacking attack this could cause serious threats to the driver as no communication over this CAN bus is possible anymore. Thus each ECU knows just what it gets from its own sensors. This, however, might not be enough to assure safety in all situations.

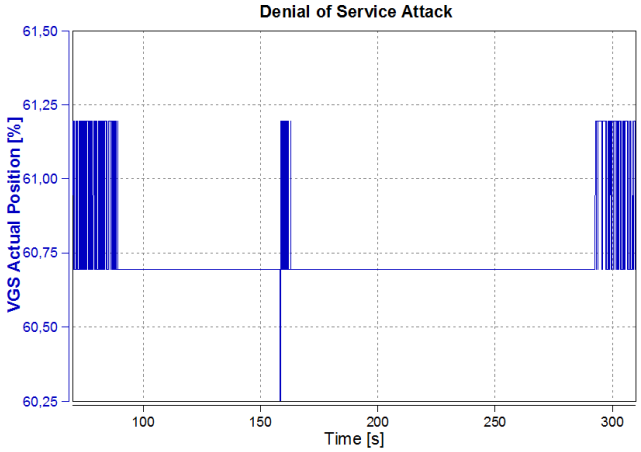
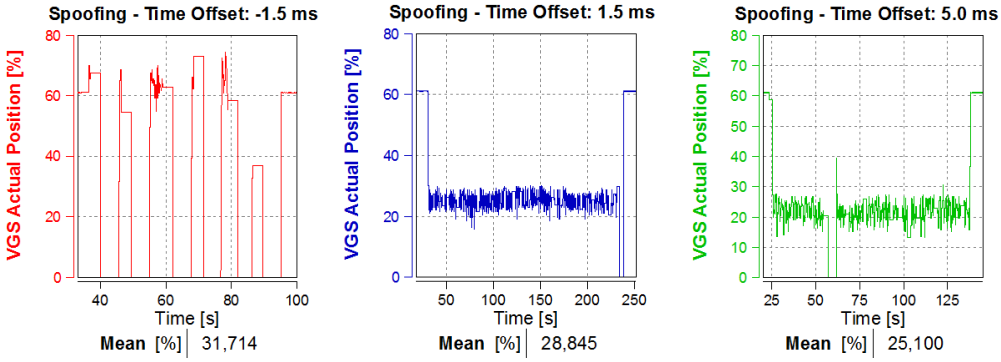


Figure 5.15: DoS attack with ID 1BEh in the heavy duty vehicle HiL system setup

5.3.5 Correctness of Time Out Window

Most vehicular ECUs rely on periodically incoming status messages. These messages have a predefined cycle time that states how many time passes, till the next message will be sent. However, a higher bus load or other problems may cause the transmitting ECU to not exactly adhere to this defined cycle time. For instance, during our tests we discovered, that the hardware in this test setup fluctuated by almost 1.5 ms. This circumstance makes it necessary for the ECUs to also accept messages further away from the expected time of receipt. As a consequence, ECUs use a specified time tolerance to determine the validity of a cyclic message. This tolerance is called *time out window*.

To determine the impact and the consequences of the window size we carried out a few time-based spoofing attacks with different time offset values. The cycle time of both VGS messages is 15 ms.



(a)Spoofing attack with a time offset of -1.5 ms

(b)Spoofing attack with a time offset of 1.5 ms

(c)Spoofing attack with a time offset of 5 ms

Figure 5.16: Overview of different time offset values and their success on spoofing the target position of the VGS in the heavy duty vehicle HiL system setup. The spoofed target position was 30 percent, the original target position was 61 percent.

Figure 5.16 shows the success rate of three different time offset values. All three aimed to set the target value of the VGS to 30 percent. The first chart depicts a time offset attack with a value of -1.5 ms (see Figure 5.16a). This means that the attack message is received before the original message. We can see that almost at all points in time our attack was ignored or caused a collision on the bus. These collisions in turn caused a target value of zero, which indicates a bus error.

The middle chart shows the attack with a time offset of 1.5 ms. This means our message should arrive 1.5 ms after the original message was received. In this figure we can see that our attack was successful in almost all instances. Just at the end of the attack a short collision happened on the bus (value of zero at second 240). The deviation of the mean value from its expected target value and the fluctuations depicted in the chart, are caused by internal computations of the VGS actuator itself. This noise during the attack suggests, that the original messages were sometimes able to prevail. Instead of moving its vanes directly to this higher percentage, the VGS

actuator might use an interpolation technique to adapt to these changes; however, before the actually requested value could be reached, the spoofed messages lower this value again. Thus, the mean value of 28.845 percent suggests that we cannot guarantee an entirely accurate spoofed value at all time (see Figure 5.16b).

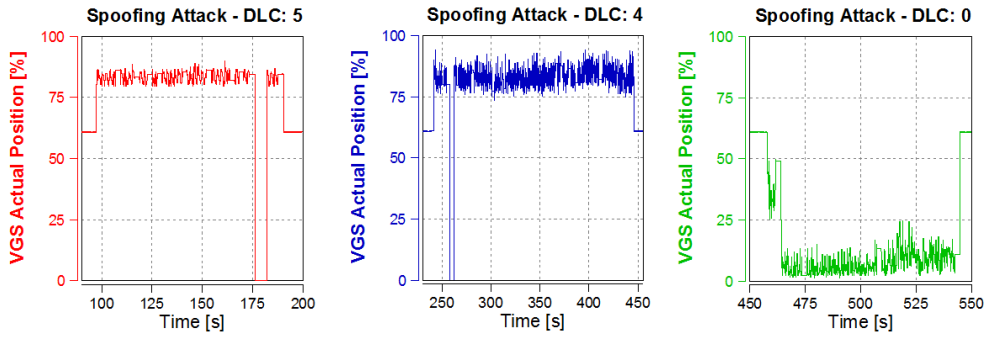
The last chart of Figure 5.16 visualizes the attack with a time offset of 5.0 ms. Here we can see that the attack was still successful, but the noise is higher than in the previously described attack. Due to the long time offset the actuator did not always accept the injected value and adapted its position more frequently (see Figure 5.16c).

As a consequence we can say that the best results were achieved with a time window of 1.5 ms. Nevertheless the targeted ECU still allowed us to send messages even later than that. The example showed that the success rate of the attack decreased but it was still possible to attack the target (Figure 5.16c). Therefore the time out window seems to be larger than it has to be. This makes it easier to spoof messages without being susceptible to the threat of being overwritten by the real CAN controller.

5.3.6 Compliance with Expected Message Format

The next test we carried out was to validate how well the ECUs are checking their incoming messages with regard to their frame structure. Here we noticed that the VGS did not carefully verify the messages. Messages with a shorter data length code (DLC) were equally accepted as messages without any data at all. Missing values were just treated as zeros.

This behavior is demonstrated in Figure 5.17. The picture shows a spoofing attack with different DLC values. The target value changed for each plot due to the shorter data length. For the first plot (DLC: 5) it was set to 80 percent (hex 0320), the second with the DLC of 4 already missed the last part of the target position. So instead of the hex value 0320 we just sent 03 which equals 76.8 percent when completed with zeros to the value 0300. The last graphic shows the attack without any data (DLC: 0). Instead of detecting this message as faulty it is again completed with all zeros which led to a target position of zero percent.



(a)Spoofing attack with a data legth code of 5

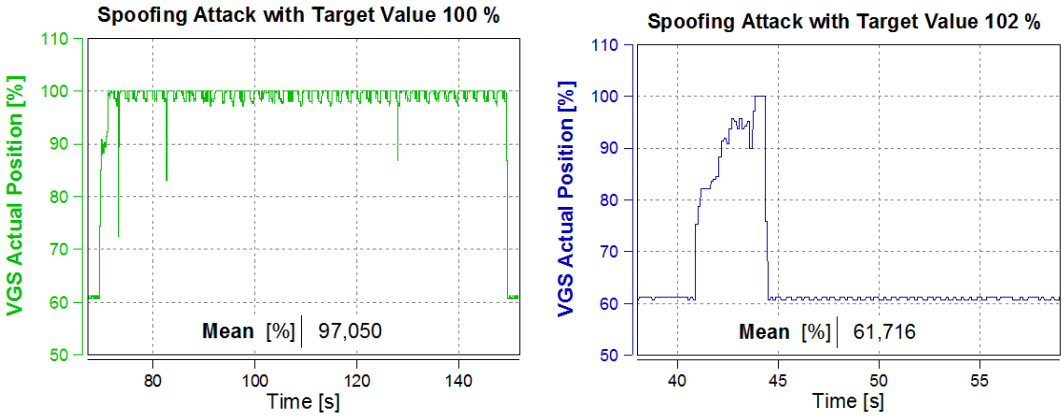
(b)Spoofing attack with a data legth code of 4

(c)Spoofing attack with a data legth code of 0

Figure 5.17: Overview of different frame structures with regard to their data length code and their impact on the acceptance by the targeted ECU in the heavy duty vehicle HiL system setup

As a message without any data does not contain any information, it should under no circumstances be complemented with zeros and treated as valid. This implies that the VGS lacks a tight enough verification of incoming messages. This makes it easy for attackers to spoof messages without the need of any detailed knowledge of the underlying system.

As stated above (see Section 5.3.5), the noise, especially in Figure 5.17b, is caused by a not perfectly successful spoofing attack. The VGS Controller was able to place some messages in between our spoofed messages. This caused an adaption of the VGS actuator which resulted in the noise depicted in this chart.



(a)Spoofing attack which increased the target position to 100 percent

(b)Spoofing attack which increased the target position to 102 percent

Figure 5.18: Reaction of the observed ECU when confronted with illegal values in the heavy duty vehicle HiL system setup

5.3.7 Plausibility Checks for Illegal Values

Another important test was to see how the observed ECU reacts when illegal values are sent with the attack.

We sent messages to the ECU with a target value of more than 100 percent (see Figure 5.18b). The left plot (Figure 5.18a) shows the spoofing of 100 percent whereas the right graph shows the error handling of the ECU. At the moment when the target value exceeded 100 percent the ECU ignored the message and accepted the original messages as valid again.

As we already demonstrated with the tests in the previous sections, the VGS did not make any checks whether a rapid change from 60 percent to 0 percent is plausible or if extreme positions like 0 or 100 percent are valid. Thus, we can say that the VGS might lack plausibility checks for the positioning of the vanes.

5.3.8 Conclusion

We were able to prove that every functionality of the CAN-CT worked properly also in a HiL setup. Furthermore, we highlighted the importance for such a tool by pointing out weaknesses of the observed ECU's communication mechanisms.

By using the CAN-CT's functionality to spoof messages we were able to demonstrate on the one hand the simplicity of manipulating a vehicular control unit while on the other hand we could reveal serious shortcomings regarding the plausibility checks of incoming messages.

Although, we observed a big deviation between the supposed and the actual receipt of the original message, the time out window of the tested ECU seemed to be too large. Even after a time offset of 5 ms our message was accepted by the VGS. Furthermore, the expected frame structure of the messages was not checked at all.

All the weaknesses we revealed with the CAN-CT ease potential hacking attacks as no knowledge of the underlying system is needed. As the common security principle for normal CAN communication in a vehicle is still best described by the security through obscurity approach, these flaws even weaken this protection mechanism and offer susceptibilities for a number of attacks.

5.4 Passenger Car

In the last two test setups we were already able to on the one hand prove the CAN-CT working properly and on the other hand highlighting its importance for securing the CAN communication by pointing out serious lacks of the implemented safety/security measures.

However, the heavy duty vehicle did not use all possible techniques to protect CAN controllers from malicious communication. For that reason we decided to also test a vehicle that implemented ancillary checks like additional cyclic redundancy checks or alive counters.

5.4.1 Test Setup

The underlying vehicle for this HiL setup was a modern passenger car. It differed from the heavy duty vehicle as it made use of the following techniques:

- integration of an additional *application layer CRC* to protect the message payload,
- use of an *alive counter*, which serves as a sequence number to assure the correct receipt of a series of messages, and
- more sophisticated *plausibility checks*.

Based on these ancillary protection mechanisms we aimed to answer the following questions:

- Can these protection mechanisms hamper our attacks?
- Can the CAN-CT be used to reveal implementation flaws in these techniques?
- Are all previously introduced attacks also working for this test setup?

As a consequence, our goal was to verify whether these protection measures were implemented correctly by making use of the CAN-CT. In detail we used the following attacks to test for unexpected behavior of the targeted ECU:

- time-based spoofing attacks,
- event-based replay attacks, and
- bus off attacks.

With these attacks we aimed to manipulate the vehicle speed up to 250 km/h.

We carried out DoS attacks as well. However, as the result was similar to the DoS attack described in the previous section (see Section 5.3.4) and thus did not reveal any new findings, we are not detailing this attack any further.

Architecture

The architecture of this HiL setup was similar to the one described in Section 5.3.1. However, as we see in Figure 5.19 it was less complex. Here we were using just one CAN bus (engine CAN) that connected the engine control unit to the HiL system.

We again connected the CAN-CT via the PCAN-USB adapter to the engine CAN. To observe our attacks we made use of the oscilloscope and the HiL-PC to monitor internal states of the engine control unit.

The other components depicted in Figure 5.19 fulfilled the same tasks as in the previous setup. The HiL Simulator as well as the *AVL PUMA Open* were used to run the simulation and to model the environment. These components assure the proper working of the DUT.

We used the engine control unit as the recipient of our attack messages. We monitored, if we were able to manipulate the engine control unit and if we can cause it to enter error states. The falsified messages used for these tests are originally coming from the break control unit. This control unit periodically provides the engine control unit, among others, with information about the speed of each wheel and the vehicle speed in general. As, however, this control unit is not physically attached to the CAN bus in this setup, it is simulated by the HiL system.

CAN Communication Characteristics

As stated above, we made use of the communication between the HiL and the engine control unit. Here we identified the two messages controlling the vehicle's speed, originally sent from the break control unit, as possible targets. Due to the absence of the break control unit in this setup, both messages are sent by the HiL simulator. These messages are then received by the engine control unit, which in turn took these values as input to calculate the vehicle speed.

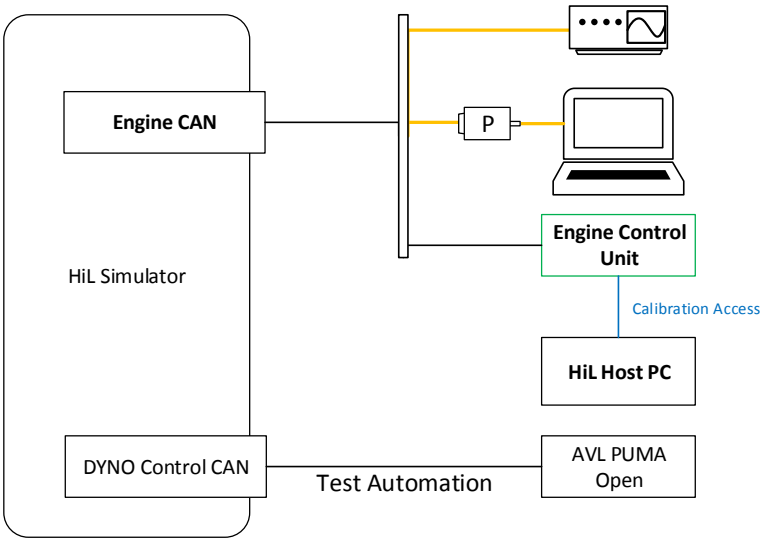


Figure 5.19: Network architecture of the second Hardware in the Loop System setup, a passenger car

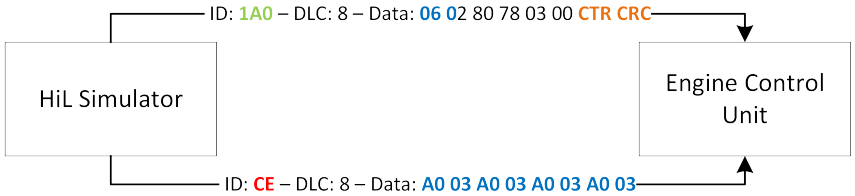


Figure 5.20: Speed related CAN messages with example values used for our attacks in the passenger car HiL system setup. The bold blue numbers within the payload represent the bytes that control the vehicle speed.

The first message we focused on had the ID 1A0h. This message got sent every 20 ms from the HiL simulator and consisted of information about the vehicle speed (bold, blue numbers in Figure 5.20), the state of the vehicle (parking position, driving forward/backward and so on) and other speed related information. This message used an application layer CRC as well as an alive counter to protect the message.

The second message we used for our attacks (ID CEh) provided the ECU with speed related information as well. This message, however, did not contain an additional CRC or an alive counter as the transmitted data required the full payload. This data is divided in four sections of two bytes each. It consisted of the current wheel speed for each wheel separately. Based on these four wheel speeds the ECU then calculated the average speed, checked its plausibility and used it as the internal vehicle speed.

Figure 5.20 depicts this traffic in more detail. For our test cases we mainly focused on the bold, blue data values of the messages illustrated in the figure. These values were used by the ECU to calculate the internal vehicle speed and to make certain actions plausible. For example the internally calculated speed will then be used for applications like the cruise control, the turbocharger, the gearbox, or the engine per se.

Detecting a Successful Attack

The detection of a successful attack was carried out the same way as in the previous setup (see Section 5.3.1). The oscilloscope as well as the HiL PC's monitoring functionality was used along with the CAN-CT's ability to determine success and to create log files.

5.4.2 Plausibility Checks

Our tests showed that the CAN-CT's spoofing attacks work in this setup the same way as described in the previous sections. Thus, in the following tests we concentrated on the validation of the implemented measures to protect the communication.

Our first test was the examination of the internal plausibility checks of the ECU. By trying different time offsets for our spoofing attack, we achieved the highest success rate with a time offset of 1.0 ms.

Rejection of Values

Our assumption was that we have to make the manipulated increase of speed plausible for the ECU to accept it. Thus, we spoofed the speed rise from the ground speed to the desired speed using multiple messages over several seconds.

By comparing these results with a spoofing attack that directly set the speed to the desired level we discovered that the ECU's internal calculation of the vehicle speed took care of making changed values plausible by itself. This means a direct rise from zero to 250 km/h (respectively vice versa the fall of this speed) within one spoofed message for example was internally spread over a few steps to the manipulated value. The then following messages with the same spoofed value were made plausible by the ECU without our contribution. This behavior is depicted in Figure 5.21b. Here we can see that the attack stopped around the time 43,25 seconds. The following speed decrease was then spread over several steps.

We further tested if we could use this behavior to overcome possible problems during our attack. We intentionally weakened our attack to allow the original sender of the speed messages to prevail sometimes. The interesting situation that occurred, was that the ECU actually helped us to cope with these problems by ignoring these (original) values. This was done by the ECU as it assumed that based on previous signals these values are potentially faulty. The circumstance that our attack was ever successful is based on the large number of manipulated messages coming after the first one that also stated this changed value. The ECU could not ignore these messages and had to adapt. On the contrary, occasionally received messages, stating a contradicting value, are ignored. This fact helped us when we had to cope with problems to take over the targeted message.

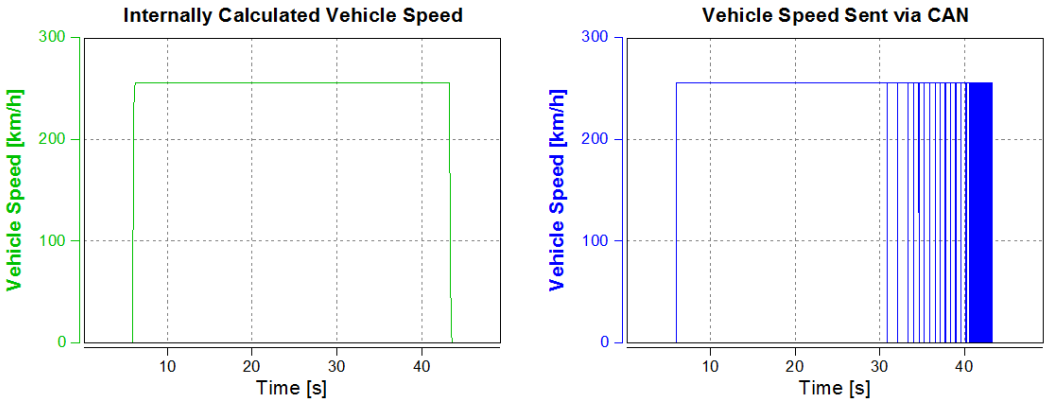
The result of this test can be seen in Figure 5.21. Figure 5.21a shows the attack as a whole. Especially interesting is the behavior of the ECU depicted in Figure 5.21b. Here we can see the difference between the internal calculation of the vehicle speed and the actually received values in more detail.

Combination of Speed Related Messages

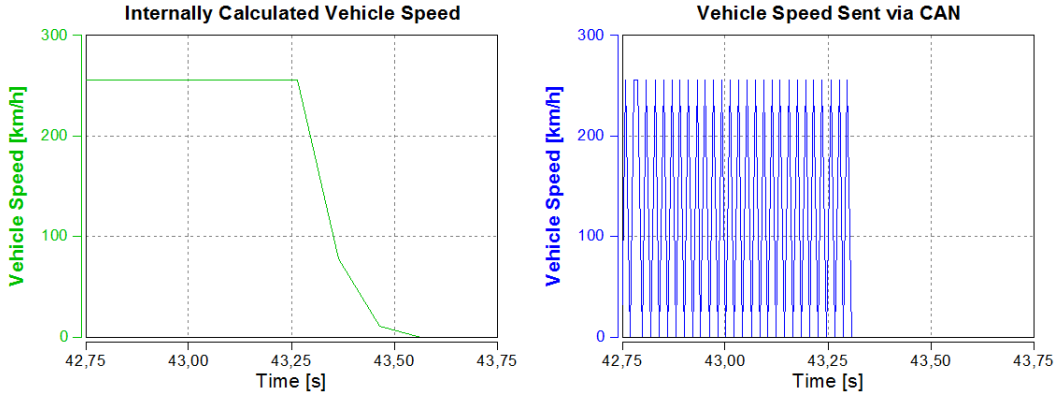
For the attack described above we just needed to spoof the message with the ID CEh. However, for some internal calculations or plausibility checks, for instance the calculation of the vehicles acceleration, both messages combined are necessary.

Hence we had to spoof both messages at the same time. The task per se did not differ much from a simple spoofing attack. However, it needs more sophisticated knowledge about the dependencies of the underlying system.

If an attacker has this knowledge, combining multiple messages to one attack is not a problem and thus does not protect from being hacked.



(a)Overview of the validation of the internal vehicle speed during disturbances of the spoofing attack



(b)Detailed plot of the validation of the internal vehicle speed during disturbances of the spoofing attack

Figure 5.21: Plausibility checks for the internally calculated vehicle speed based on the received wheel speeds in the passenger car HiL system setup. The attack started at around second 6 and ended at second 43,30. The value of 250 km/h reflects our spoofed signal; the value of 0 km/h the actual vehicle speed.
At the end of the attack (starting at around second 31) the sender was able to continuously prevail over our attack (right plot); however, based on the previous contradicting signals these disturbances were ignored by the receiving ECU (left plot) .
Left: internally calculated vehicle speed based on the received wheel speeds, *Right:* actually received wheel speeds over CAN

5.4.3 Alive Counter

For the next part we focused our attention on the alive counter. The alive counter is used to help the recipient determine whether there exists a problem with the transmitter. Every message sent increases the counter by one. The alive counter is part of the message's payload and therefore gets transmitted every time (see *CTR* part in data-byte 7 of message ID 1A0h in Figure 5.20). Based on this mechanism the recipient then knows if all messages where received correctly and arrived in the expected sequence.

As we used message ID CEh, which did not contain an alive counter, in the previous test, we did not have to care about this protection mechanism before. For that reason we tried to figure out whether this protection measure can prevent us from spoofing messages, or if we can discover flaws in the implementation to cause unexpected behavior of the attacked ECU.

Dependence on Previous Value

For these tests we started by capturing the current value of the alive counter. We then calculated the next time of receipt of this message, adapted the alive counter (increased it by one) and sent our messages with a time offset of 1.0 ms after the next planned time of receipt. This means we complied with the correct alive counter and overwrote the original message in the recipients receive message buffer. This way were able to successfully spoof messages protected by the alive counter as well.

During our tests we noticed that we do not actually have to be in sync with the alive counter of the sender. The first message may have been rejected by the recipient but as soon as it recognized that the next message's alive counter equaled the last received message's counter increased by one it accepted it. We were able to show that as long as the sent message depends on the previous message we do not even need to be in sync with the last original message.

Omission of Alive Counter

After we examined the behavior of the alive counter's checking procedure in more detail we further wanted to analyze the reaction of the ECU when the alive counter is completely omitted or placed in another data-byte.

However, we were not able to detect any weaknesses. If the alive counter is placed in another byte of the payload or completely omitted, the message will be rejected.

This makes it at least harder for attackers without any knowledge to cause harm to the system. Nevertheless when analyzing the traffic on the bus an alive counter can be detected quite easily as shown with the CAN-CT.

5.4.4 Additional Cyclic Redundancy Check

Many manufacturers use an application layer cyclic redundancy check (CRC) besides the CAN protocol's CRC to protect important message content. The application layer CRC is part of the message and can be transmitted in any data byte of the payload (see *CRC* part in data-byte 8 of message ID 1A0h in Figure 5.20). The CRC used in this test setup is similarly calculated as the CAN protocol's CRC (see Section 2.3.4). However, instead of using the CAN frame's control fields as well for its computation, it just made use of the ID and the remaining data bytes.

We aimed to examine if an application layer CRC can hamper our attacks or if we can take advantage of any flaws in its implementation. We made use of the CAN-CT's functionality to determine which data byte contained the CRC. As we figured out that just one of our two observed messages (ID 1A0h) contained an additional CRC, all further described tests were based on this message.

By adding a correctly calculated CRC to our spoofed message, the ECU accepted it without any problems. Therefore, if one knows how to calculate the CRC, the containing data byte can be identified. Thus, this approach does not protect from malicious messages either.

Off by X

The first test we carried out to identify possible weaknesses in the implementation of the CRC, was to see, if we can detect any unexpected behavior of the ECU when sending lower or higher CRCs than expected. At first we calculated the CRC for our spoofed messages correctly and then increased respectively decreased a given value from this figure.

However, every message that did not comply with the correct CRC got rejected by the ECU. The spoofed messages were just ignored when the CRC was not calculated properly.

Omission of CRC

The same result was observable when we omitted the CRC completely respectively placed it in a wrong data byte. Here again our attack did not cause any effect and was ignored.

As a consequence we can say that the CRC checks are properly implemented. We were not able to reveal any weaknesses regarding this protection mechanism.

5.4.5 Conclusion

With this test setup we were able to successfully extend our tests in a HiL-based vehicular communication network. In addition to the already introduced attacks in the previous section we were able to test the effect of ancillary safety measures like additional CRC and alive counter.

We examined if these measures are representing any obstacles for our attacks. However, we were able to automatically identify these mechanisms within a message easily with the CAN-CT's capturing function (see Section 4.4.2) and furthermore could integrate it into our attacks without any problems.

On the contrary we further were able to prove for the observed ECU that we do not even have to be in sync with the last original alive counter value when starting our attack. We just have to spoof our message with an alive counter at the right data byte position of the message and increase it with every message. The ECU then just checks whether the new message's value correlates to the previous one.

Besides the examination of these techniques we further figured out that the plausibility checks of the observed ECU actually support our attack as they made implausible values plausible over time. In other words, if we started our spoofing attack without increasing it slowly over a certain period of time, and instead setting the speed directly to a given value, the attack was not rejected. On the contrary the ECU even adapted its internal vehicle speed to comply with our spoofed values.

This test setup allowed us again to successfully reveal weaknesses or at least strange behavior in the implemented protection mechanisms of the CAN communication.

5.5 Interpretation of Results

Due to the HiL setups it was possible to test the CAN-CT in a scenario that is very similar to a real vehicle. The only difference is that there might be less traffic on the bus due to simulated ECUs. Most HiL setups implement only those messages that are necessary for their tasks. This, however, should not interfere with our tests. Messages of these simulated ECUs are just not implemented if and only if their absence does not impair the actual hardware ECU's working. Furthermore to the best of our knowledge the reduced bus load does not make any differences either. As long as the bus is not working to capacity no divergences should be present. Although, it might be possible, that a message cannot always adhere to its cycle time, as we demonstrated, the CAN-CT is capable to dynamically adapt to those delays in the time of receipt.

Besides this a HiL offers a complete setup of actual vehicle ECUs communicating with each other the same way as they do in a real vehicle. The monitoring functionality of this setup further gave us a detailed insight how and when messages are accepted from the CAN bus.

5.5.1 Lack of "Real" Security Approaches in CAN Communication

Even though we tested three separate ECUs, just one implemented slightly more sophisticated protection mechanisms for the CAN communication. These included an additional application layer CRC, more sophisticated plausibility checks and an alive counter to determine a message's validity. These measures are not an obstacle for an attacker. One must have a more profound knowledge about the system but it is still not enough to hamper hacking attacks in general.

The underlying problem here is that, in contrast to some diagnostic protocols, there is no protection of the normal CAN communications with security mechanisms like encryption, authentication or attack detection techniques (see Sections 2.3.8 and 3.2).

Nevertheless, as stated in Section 4.7 the CAN-CT was developed considering possible extensions for future security mechanisms. Due to the current lack of more sophisticated security approaches in modern vehicles we were not able to take these techniques already into account, but prepared for future adaptations.

5.5.2 Universal Validity

In our test setups we mainly focused on one (passenger car HiL system) or two (heavy duty vehicle HiL system) ECUs. To the best of our knowledge there do not exist fundamental differences to any other vehicular control units. These ECUs were chosen based on their special characteristics and interesting applications. Furthermore we were able to demonstrate that our tests were applicable for two different vehicle types as well.

5.6 Conclusion

The aim of this chapter was the practical validation of the developed tool - the CAN Communication Tester - detailed in Chapter 4. We were able to validate the CAN-CT in a laboratory setup as well as in real world scenarios. Furthermore besides the proof of concept of being able to hack a vehicular communication network we even could point out serious weaknesses in the implemented protection mechanisms of the targeted ECUs.

In more detail these flaws consisted of serious lacks in the plausibility checking of messages, of a too loose time out window for incoming messages, a not strict enough alive counter checking procedure and a completely omitted examination of the correct frame structure. Regarding the latter we were able to show that even messages that did not consist of any data, were still seen as valid and accepted by the targeted ECU without any problems. This shortcoming poses a serious vulnerability to a number of attacks, as no underlying knowledge is needed to manipulate an ECU.

Another flaw we want to highlight is that neither the heavy duty vehicle nor the passenger car made any checks whether the received values are plausible or not. Especially the latter even supported our attacks as it made implausible values plausible by itself. As we showed in Section 5.4.2 this fact can help for instance when we have to face problems in prevailing over the original ECU.

In addition to revealing these weaknesses we were able to show that every attack described in Section 4.5 was carried out successfully. We were able to kick a targeted ECU off the bus and even causing the internal reset counter to overrun. We proved how to spoof messages even when they were protected by an application layer CRC and alive counter and further we suppressed the whole or specific parts of the CAN communication with our DoS attacks.

Nevertheless not all implemented attacks are suitable for every purpose the same way. This may be caused by the characteristics of the particular attack per se, by limitations of the PCAN-USB, or by a different behavior of the attacked real world ECUs to the simulated CAN nodes. Especially the latter showed us that other than our expectations the ECU would not give in, even when consequently kicked off the bus.

However, all things considered we can say that we were able to successfully prove every aspect of the CAN-CT. In fact by revealing different weaknesses in the communication techniques of state of the art ECUs we were able to highlight the urgent need for such an application to test against malicious behavior.

6 Outlook and Conclusion

Modern cars consist of up to 100 electronic control units (ECUs) communicating over various vehicular networks (Charette, 2009). With the interconnection with their environment vehicles introduced advances regarding safety and entertainment. However, these features opened up a large number of new attack surfaces. The prevalent security by isolation approach is broken and the internal communication systems are no closed networks anymore (Koscher, 2014).

Even though manufacturers use strong security measures to protect those interfaces from hackers, there is no 100 percent secure system. This fact was demonstrated by many researchers, who were able to exploit almost every interface of a car with its outside world (Checkoway et al., 2011; Valasek and Miller, 2015). This circumstance makes it necessary that besides the protection of those interfaces also the communication on internal vehicular networks has to be secured. This is especially true for the Controller Area Network (CAN) bus, as it connects the most critical components in a vehicle.

The inherent problem, however, is, that the CAN protocol per se does not define any standards regarding security. Although researchers proposed various means to protect those networks (see Section 3.2), so far there is no commonly realized mechanism. Currently, most vehicles use safety mechanisms like application layer CRCs, alive counter, time out windows and internal plausibility checks to protect their communication. These means, combined with the effort to keep message IDs and data encoding confidential, hamper attacks, but do not provide sufficient protection.

As car hacking becomes a serious threat, it is getting increasingly important to ensure a correct and secure behavior of every single ECU. Protection mechanisms are just as good as their implementation. To test if an ECU is vulnerable to an attack, it is necessary to attack it the same way as a hacker would; while simultaneously tracking whether undesired or unspecified behavior was detected.

Although we are aware that no actual security measures are implemented for normal communication on vehicular CAN buses so far, it is still important to assure the proper working of currently implemented protection techniques. Therefore, with the CAN Communication Tester (CAN-CT) we implemented an application that allows for detecting weaknesses in those protection techniques by penetrating the system. We are using CAN attacks as the basis for our systematic tests to efficiently detect possible vulnerabilities of the CAN communication.

With the CAN-CT we developed an application that learns from the traffic on the network and automatically detects possible protection mechanisms. This knowledge is then used as the basis for further attacks to inject, replay, spoof and invalidate CAN messages.

By making use of the Hardware in the Loop Systems (HiLs) we were able to test the impact and applicability of the CAN-CT in close to real world scenarios. Here we revealed various weaknesses and strange behaviors of the ECUs under test. For instance these flaws encompassed serious lacks in the plausibility checking of messages, a too loose time out window for incoming messages, a not strict enough alive counter checking procedure and a completely omitted examination of the correct frame structure.

Besides the attack types to invalidate, replay and inject messages set out as objectives for this thesis we further implemented attacks to suppress all communication on the bus (Denial of Service (DoS)) as well as a method to specifically kick ECUs off the bus. However, we were not able to realize attacks to manipulate transmitted messages on the fly. These attacks would have needed special hardware that would be able to dismiss rules of the CAN protocol.

As we made use of a third party USB to CAN adapter (PCAN-USB) all our actions had to conform to the CAN protocol. This limitation as well as a high delay caused by the hardware hampered the development of the CAN-CT. The latter, however, we were able to circumvent by implementing time-triggered attacks. These attacks anticipate the next time of receipt of a target message and react accordingly.

Due to the large number of different higher layer protocols we decided to use a more generic approach in the CAN-CT. We did not implement any higher layer protocol so far. Yet based on its flexible design it is possible to define attacks in a way to meet most requirements of higher layer protocols. The design of the CAN-CT further allows to extend it to support future functions and attacks.

However, in this version of the CAN-CT all our attacks mostly follow a unidirectional approach. We can define specific replies to certain messages but we do not cover the full complexity needed for more sophisticated interactions used in some higher layer protocols (for instance diagnostic protocols). Adding a higher layer protocol to the CAN-CT can be done by implementing the *IAttack* interface (see Section 4.3). All attacks that implement this interface can easily be integrated into the CAN-CT. Another option is making use of the C# scripting engine of the CAN-CT to implement higher layer protocol attacks directly in the CAN-CT. This approach, however, allows just for temporary attacks and might be too limited for complex tasks.

The implementation of more sophisticated interactions between ECUs as well as the support for diagnostic protocols would be subject for future work. By supporting those protocols it would be possible to examine the correct protection mechanisms for certain commands like re-flashing an ECU. Another interesting complement to the CAN-CT would be the use of a specifically designed hardware to connect to the CAN bus. By not conforming to the CAN protocol many more attacks are possible.

Summing up, we demonstrated that we successfully reached almost all of the goals for this thesis outlined in Section 1.3.1. In the following we listed each goal with additional information to which extend we were able to accomplish it:

- Testing the correct implementation of CAN networking including current protection mechanisms ✓
- Supporting for attacks that
  - invalidate, ✓
  - manipulate *(manipulation of target via spoofing, not manipulation of original message on the fly)*, ∼
  - replay, and ✓
  - inject CAN messages. ✓
- Supporting various higher layer protocols for analysis and attacks *(generic design to be protocol independent)* ∼
- Testing the implementation against an unknown ECU in a real world scenario ✓
- Implementing the prototype as a learning system by analyzing the traffic on the CAN bus directly.
  - We determine important message IDs and their respective purpose in the CAN communication *(the purpose of the message cannot be determined with the CAN-CT, but all protection mechanisms are identified)* ∼
  - and use them as the input for further attacks. ✓

With the CAN-CT we implemented an application that is capable of attacking ECUs the same way a hacker would do. This allows us to test the correct implementation of protection mechanisms on the one hand, while on the other hand future attack detection techniques can be examined this way as well. With the use of the HiLs we were able to prove the CAN-CT working properly also in a real-world scenario. We revealed many vulnerabilities in the CAN communication and thus showed its applicability and impact for current automotive systems. For that reason we want to highlight once again the importance of a security testing framework to complement existing safety and functional testing environments.

Bibliography

Armengaud, Eric, Andreas Steininger, and Martin Horauer (2008). "Towards a systematic test for embedded automotive communication systems." In: *IEEE Transactions on Industrial Informatics* 4.3, pp. 146–155. ISSN: 15513203. DOI: 10.1109/TII.2008.2002704 (cit. on p. 31).

Bayer, Stephanie, Thomas Enderle, Dennis-Kengo Oka, and Marko Wolf (2014). "Security Crash Test - Practical Security Evaluations of Automotive Onboard IT Components." In: *Automotive - Safety and Security 2014 (2015), Sicherheit und Zuverlässigkeit für automobile Informationstechnik, Tagung, 21.-22.04.2015, Stuttgart, Germany*, pp. 125–139 (cit. on pp. 31, 32).

BBC News (2010). *Hack attacks mounted on car control systems - BBC News*. URL: http://www.bbc.com/news/10119492 (visited on 04/30/2015) (cit. on pp. 2, 23).

Bittl, Sebastian (2014). "Attack potential and efficient security enhancement of automotive bus networks using short MACs with rapid key change." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8435 LNCS, pp. 113–125. ISSN: 16113349. DOI: 10.1007/978-3-319-06644-8_11 (cit. on p. 30).

Bosch (1991). *CAN Specification Version 2.0* (cit. on pp. 13, 15–18).

Cassettari, Riccardo, Luca Fanucci, and Giorgio Boccini (2014). "A new hardware implementation of the advanced encryption standard algorithm for automotive applications." In: *2014 10th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, pp. 1–4. ISBN: 978-1-4799-4994-6. DOI: 10.1109/PRIME.2014.6872672 (cit. on p. 30).

Charette, Robert N. (2009). *This car runs on code*. URL: http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code (visited on 04/30/2015) (cit. on pp. 1, 7, 101).

Checkoway, Stephen, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno (2011). "Comprehensive Experimental Analyses of Automotive Attack Surfaces." In: *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings* (cit. on pp. 2, 23, 24, 29, 101).

Di Natale, Marco, Haibo Zeng, Paolo Giusto, and Arkadeb Ghosal (2012). *Understanding and Using the Controller Area Network Communication Protocol: Theory and Practice*. Springer-Verlag New York. ISBN: 978-1461403135. DOI: 10.1007/978-1-4614-0314-2 (cit. on pp. 13–15).

Evenchick, Eric (2015). *An Introduction to the CANard Toolkit* (cit. on p. 28).

Groza, Bogdan and Pal-stefan Murvay (2012). "Broadcast Authentication in a Low Speed Controller Area Network." In: *E-Business and Telecommunications*, pp. 330–344. ISBN: 978-3-642-35755-8. DOI: 10.1007/978-3-642-35755-8_23 (cit. on p. 30).

Groza, Bogdan and Stefan Murvay (2013). "Efficient protocols for secure broadcast in controller area networks." In: *IEEE Transactions on Industrial Informatics* 9.4, pp. 2034–2042. ISSN: 15513203. DOI: 10.1109/TII.2013.2239301 (cit. on p. 30).

Groza, Bogdan, Stefan Murvay, Anthony Van Herrewege, and Ingrid Verbauwhede (2012). "LiBrA-CAN: A lightweight broadcast authentication protocol for controller area networks." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7712 LNCS, pp. 185–200. ISSN: 03029743. DOI: 10.1007/978-3-642-35404-5_15 (cit. on p. 30).

Happel, Armin (2014). "Secure communication for CAN FD." In: *CAN Newsletter* 4/2014, pp. 1–3 (cit. on p. 11).

Hoppe, Tobias, Stefan Kiltz, and Jana Dittmann (2011). "Security threats to automotive CAN networks - Practical examples and selected short-term countermeasures." In: *Rel. Eng. & Sys. Safety* 96.1, pp. 11–25. DOI: 10.1016/j.ress.2010.06.026 (cit. on pp. 2, 28, 30).

Jenik, Aviram (2015). "Increasing resilience by finding unknown vulnerabilities." In: *iCC 2015 Proceedings - 15th international CAN Conference, Vienna, Austria, October 27-28, 2015*, pp. 06-1–06-5 (cit. on p. 32).

Johansson, Karl Henrik, Martin Törngren, and Lars Nielsen (2005). "Vehicle Applications of Controller Area Network." In: *Handbook of Networked and Embedded Control Systems*. Vol. VI, pp. 741–765. DOI: 10.1007/0-8176-4404-0_32 (cit. on pp. 18–21).

Kammerer, Roland, Bernhard Frömel, and Armin Wasicek (2012). "Enhancing security in CAN systems using a star coupling router." In: *7th IEEE International Symposium on Industrial Embedded Systems, SIES 2012 - Conference Proceedings*. Karlsruhe: IEEE, pp. 237–246. ISBN: 9781467326841. DOI: 10.1109/SIES.2012.6356590 (cit. on p. 30).

Knapp, Eric D. and Joel Thomas Langill (2015). "Hacking Industrial Control Systems." In: pp. 171–207. DOI: 10.1016/B978-0-12-420114-9.00007-1 (cit. on pp. 25, 26).

Koopman, Philip and Christopher Szilagyi (2013). "Integrity in embedded control networks." In: *IEEE Security and Privacy* 11.3, pp. 61–63. ISSN: 1540-7993. DOI: 10.1109/MSP.2013.61 (cit. on p. 25).

Koscher, Karl (2014). "Securing Embedded Systems: Analyses of Modern Automotive Systems and Enabling Near-Real Time Dynamic Analysis." PhD thesis. University of Washington. URL: https://digital.lib.washington.edu/researchworks/bitstream/handle/1773/26024/Koscher%5C_washington%5C_0250E%5C_13805.pdf?sequence=1 (cit. on pp. 7, 22, 30, 101).

Koscher, Karl, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage (2010). "Experimental Security Analysis of a Modern Automobile." In: *31st IEEE Symposium on Security and Privacy, S and P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*, pp. 447–462. DOI: 10.1109/SP.2010.34 (cit. on pp. 2, 22, 25, 27, 28).

Kvaser (2012). *CAN Protocol Tutorial* (cit. on pp. 20, 21).

Larson, Ulf E. and Dennis K. Nilsson (2008). "Securing Vehicles Against Cyber Attacks." In: *Proceedings of the 4th Annual Workshop on Cyber Security and Information Intelligence Research: Developing Strategies to Meet the Cyber Security and Information Intelligence Challenges Ahead*. CSIIRW 08. Oak Ridge, Tennessee, USA: ACM, 30:1–30:3. ISBN: 978-1-60558-098-2. DOI: 10.1145/1413140.1413174 (cit. on pp. 27, 30).

Lawrenz, Wolfhard (2013). *CAN System Engineering*. Springer-Verlag London. ISBN: 978-1-4471-5612-3. DOI: 10.1007/978-1-4471-5613-0 (cit. on pp. 12, 15, 19, 22).

Leyden, John (2010). *Boffins warn on car computer security risk - The Register*. URL: http://www.theregister.co.uk/2010/05/14/car%5C_security%5C_risks/ (visited on 12/01/2015) (cit. on pp. 2, 23).

Lin, Chung Wei, Qi Zhu, Calvin Phung, and Alberto Sangiovanni-Vincentelli (2013). "Security-aware mapping for CAN-based real-time distributed automotive systems." In: *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, pp. 115–121. ISSN: 10923152. DOI: 10.1109/ICCAD.2013.6691106 (cit. on p. 30).

Lin, Chung-Wei and Alberto Sangiovanni-Vincentelli (2012). "Cyber-Security for the Controller Area Network (CAN) Communication Protocol." In: *2012 International Conference on Cyber Security* SocialInformatics, pp. 1–7. DOI: 10.1109/CyberSecurity.2012.7 (cit. on pp. 25, 30).

Marinescu, Raluca, Mehrdad Saadatmand, Alessio Bucaioni, Cristina Seceleanu, and Paul Pettersson (2014). "A Model-Based Testing Framework for Automotive Embedded Systems." English. In: *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, pp. 38–47. ISBN: 978-1-4799-5795-8. DOI: 10.1109/SEAA.2014.70 (cit. on p. 31).

Mayer, Eugen (2006). *Serial Bus Systems in the Automobile*. Vector Informatik GmbH, pp. 1–6 (cit. on pp. 12, 17, 19).

Miller, Charlie and Chris Valasek (2013). *Adventures in Automotive Networks and Control Units*. URL: http://can-newsletter.org/assets/files/ttmedia/raw/c51e81bf7c09578c37e3f7a1f97c197b.pdf (cit. on pp. 2, 22, 28, 30, 32).

Miller, Charlie and Chris Valasek (2014). *A Survey of Remote Automotive Attack Surfaces*. URL: http://illmatics.com/remote%20attack%20surfaces.pdf (cit. on p. 28).

Mischo, Stefan, Jörn Stuphorn, Rainer Constapel, Peter Häussermann, Alexander Leonhardi, Heiko Holtkamp, Norbert Löchel, Stefan Powolny, and Hanna Zündel (2015). "Bus systems." In: *Automotive Mechatronics*, pp. 70–143. ISBN: 978-3-658-03974-5. DOI: 10.1007/978-3-658-03975-2_6 (cit. on pp. 12, 14–16).

Navet, Nicolas and Françoise Simonot-Lion (2013). "In-vehicle communication networks - a historical perspective and review." In: *Industrial Communication Technology Handbook, Second Edition*. Ed. by Richard Zurawski. CRC Press Taylor&Francis (cit. on pp. 2, 9–12).

Nilsson, Dennis K., Phu Phung, and Ulf E. Larson (2008). "Vehicle ECU classification based on safety-security characteristics." In: *Road Transport Information and Control - RTIC 2008 and ITS United Kingdom Members' Conference, IET*, pp. 1–7. ISSN: 0537-9989 (cit. on p. 27).

Ren, Wei, Michael Steurer, and Steve Woodruff (2007). "Accuracy evaluation in power hardware-in-the-loop (PHIL) simulation center for advanced power systems." In: *Proceedings of the 2007 Summer Computer Simulation Conference, SCSC 2007, San Diego, California, USA, July 16-19, 2007*, pp. 489–493. DOI: 10.1145/1357910.1357987 (cit. on p. 8).

Rouf, Ishtiaq, Rob Miller, Hossen Mustafa, Travis Taylor, Sangho Oh, Wenyuan Xu, Marco Gruteser, Wade Trappe, and Ivan Seskar (2010). "Security and Privacy Vulnerabilities of In-car Wireless Networks: A Tire Pressure Monitoring System Case Study." In: *Proceedings of the 19th USENIX Conference on Security*. USENIX Security 10. Washington, DC: USENIX Association, pp. 21–21. ISBN: 888-7-6666-5555-4 (cit. on p. 29).

Staggs, Jason (2013). *How to Hack Your Mini Cooper : Reverse Engineering CAN Messages on Passenger Automobiles* (cit. on pp. 2, 22, 28).

Talebi, Sareh (2014). "A Security Evaluation and Internal Penetration Testing Of the CAN-bus." Master of Science Thesis. Chalmers University of Technology (cit. on pp. 31, 32).

Tao, Zhang, Helder Antunes, and Siddhartha Aggarwal (2014). "Defending Connected Vehicles Against Malware: Challenges and a Solution Framework." In: *IEEE Internet of Things Journal* 1.1, pp. 10–21. DOI: 10.1109/JIOT.2014.2302386 (cit. on pp. 23, 30).

Tuohy, Shane, Martin Glavin, Ciarán Hughes, Edward Jones, Mohan M. Trivedi, and Liam Kilmartin (2015). "Intra-Vehicle Networks: A Review." In: *IEEE Transactions on Intelligent Transportation Systems* 16.2, pp. 534–545. DOI: 10.1109/TITS.2014.2320605 (cit. on pp. 9–11, 22).

Valasek, Chris and Charlie Miller (2014). *Car Hacking : For Poories*. URL: http://www.ioactive.com/pdfs/IOActive_Car_Hacking_Poories.pdf (cit. on p. 28).

Valasek, Chris and Charlie Miller (2015). *Remote Exploitation of an Unaltered Passenger Vehicle*. URL: http://www.ioactive.com/pdfs/IOActive%5C_Remote%5C_Car%5C_Hacking.pdf (cit. on pp. 2, 22, 23, 29, 101).

Vallance, Chris (2015). *Car hack uses digital-radio broadcasts to seize control - BBC News*. URL: http://www.bbc.com/news/technology-33622298 (visited on 07/27/2015) (cit. on p. 29).

Wolf, Marko and Timo Gendrullis (2012). "Design, Implementation, and evaluation of a vehicular hardware security module." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7259 LNCS, pp. 302–318. ISSN: 03029743. DOI: 10.1007/978-3-642-31912-9_20 (cit. on p. 30).

Wolf, Marko, André Weimerskirch, and Christof Paar (2004). "Security in automotive bus systems." In: *Proceedings of the Workshop on Embedded Security in Cars (escar) 2004* (cit. on p. 27).

Wolf, Marko, André Weimerskirch, and Christof Paar (2006). "Secure In-Vehicle Communication." In: *Embedded Security in Cars*. Berlin/Heidelberg: Springer-Verlag, pp. 95–109. DOI: 10.1007/3-540-28428-1_6 (cit. on pp. 1, 27).