LIIS JAKS

Stockholm 2014

School of Information and Communication Technology
Kungliga Tekniska Högskolan
This thesis was supported by the Estonian national scholarship program Kristjan Jaak, which is funded and managed by Archimedes Foundation in collaboration with the Estonian Ministry of Education and Research.
Abstract

A modern vehicle is controlled by a distributed network of embedded devices - Electronic Control Units. The software of these devices is updated over an easily accessible and standardised diagnostic interface. Their hardware capabilities are very low, and thereby the security implementations are fairly minimalistic. This thesis analyses the Electronic Control Units used in the heavy-duty vehicle company Scania for security vulnerabilities. First, a list of security requirements was compiled. The implementation of these requirements was verified on several Electronic Control Units by the application of software testing methods. Testing identified two potentially dangerous shortfalls: short encryption seeds used in the authentication challenge, and a lack of reliable software source verification. These vulnerabilities were validated by performing experimental attacks. A brute-force attack was performed on a device with 2-byte seeds and keys. Next, an active man-in-the-middle attack was successfully carried out to bypass authentication and flash the Electronic Control Unit with arbitrary software. Additionally, a passive man-in-the-middle attack was performed to sniff and store software files. The final attack was a combination: a valid seed and authentication code pair was sniffed over a flashing session, followed by using the pair to gain access later. To mitigate these attacks, it is most important to use long authentication seeds and keys, and implement all security standards. Public-key cryptography may also be an alternative for authentication. Software data encryption could be considered for integrity and confidentiality. A less computation-intense solution would be adding cryptographic signatures to messages.
Abstrakt

Acknowledgements

I would like to dedicate this thesis to my family - the ones that are close, and the ones that are close to my heart; those who are with me, and those who have passed on. I owe my education and all success in life to their constant support, and to the amazing job that my mother and father did raising me. There is hardly anything more valuable in the world than a family that sticks together, even when they are far apart.

My gratitude goes out to the helpful experts in Scania: my supervisor Niklas, who was always enthusiastic about this research and whose constant attention quickly helped me up the learning curve; Ellef, who would not let a day go by without asking me, whether my work was going smoothly; Anders, who was always busy with critical tasks, and yet never failed to find a moment to help me clear out sudden blockages; and Crister, who warmly welcomed me to Scania and thanks to whom I have a dream Master’s thesis topic.

I am also thankful to my examiner Anne for her invaluable feedback. Likewise, I am thankful to all the other influential teachers over the many years and across three countries, who have taught me everything I know about security, information technology, and learning in general.

Tack, kiitos, aitäh!
# Contents

1 Introduction 1
   1.1 Background 1
   1.2 Problem Description 2
   1.3 Purpose 3
      1.3.1 Goals, Ethics and Sustainability 3
   1.4 Methods 4
   1.5 Delimitations 5
   1.6 Outline 6

2 Theoretical Background 7
   2.1 Technological Concepts 7
      2.1.1 Electronic Control Units 7
      2.1.2 Controller Area Network 8
   2.2 Vehicle Diagnostics 10
      2.2.1 Data Link Connector 10
      2.2.2 Software Update Process 11
      2.2.3 Diagnostic Communication Protocols 11
      2.2.4 Flashing Process 12
   2.3 Security Concepts 12
   2.4 Security Mechanisms in Vehicle Diagnostics 13
      2.4.1 Security in the Controller Area Network 14
      2.4.2 Authentication and Authorisation in Diagnostics Standards 15
      2.4.3 Encryption 16
   2.5 Possible Attackers 17
   2.6 Related Work 17
      2.6.1 Bypassing Authentication 18
      2.6.2 Deviation from Standards 18
   2.7 Application 19
      2.7.1 Experiments for Confirming Potential Vulnerabilities 19

3 Methods 21
   3.1 Academic Research Methods 21
   3.2 Application of Methods 22
   3.3 Quality Assurance 22
   3.4 Software Testing Methods 24
      3.4.1 Verification and Validation 24
      3.4.2 Positive and Negative Testing 24
      3.4.3 Integration Testing 25
CONTENTS

3.4.4 Grey-box Testing ............................ 25
3.4.5 Test Automation ............................ 26

4 Test Plan 27
4.1 Attack scenarios ............................. 27
4.2 Security Requirements ....................... 28
4.3 Technical background ....................... 29
4.4 Test Cases ....................................
  4.4.1 Requirement 1: Unlocking ................ 30
  4.4.2 Requirement 2: Flashing Operations .... 31
  4.4.3 Requirement 3: Reading Memory .......... 31
  4.4.4 Requirement 4: Seed Unpredictability ... 32
  4.4.5 Requirement 5: Long Seed ............... 32
  4.4.6 Requirement 6: Time Delay ............... 33
  4.4.7 Requirement 7: Time Delay After Power-Up 34
  4.4.8 Requirement 8: Changing Sessions ....... 34
  4.4.9 Requirement 9: Multiple Testers ....... 35
  4.4.10 Requirement 10: Data Encryption .... 36

5 Identified Vulnerabilities 37
5.1 Test Set-Up ...................................
  5.1.1 The Hardware ............................ 37
  5.1.2 The Software ............................ 38
5.2 Test Results ..................................
  5.2.1 Results for ECUs A, B and G ............. 39
  5.2.2 Results for ECU C ........................ 41
  5.2.3 Results for ECU D ........................ 41
  5.2.4 Results for ECU E ........................ 41
  5.2.5 Results for ECU F ........................ 42
5.3 Summary ..................................... 43

6 Experimental Attacks 45
6.1 Brute-Force Attack ............................ 45
6.2 Man-in-the-Middle Attacks .................... 46
  6.2.1 Passive Attack: Recording Software Files 47
  6.2.2 Active Attack: Replacing ECU Software .... 48
6.3 Combined Attack ............................... 51
6.4 Impact of Vulnerabilities ..................... 52

7 Solutions 55
7.1 Proposed Solutions ............................ 55
  7.1.1 Sufficiently Strong Implementation of Security Standards 55
  7.1.2 Public-Key Authentication ................ 56
  7.1.3 Software Data Encryption ................ 56
  7.1.4 Cryptographic Signatures on Messages .... 57
  7.1.5 Session Key Exchange ..................... 57
7.2 Evaluation of Solutions ....................... 58
8 Conclusions

8.1 Identified Vulnerabilities ........................................... 62
8.2 Performed Attacks ...................................................... 62
  8.2.1 Brute-Force Attack ............................................. 63
  8.2.2 Man-in-the-Middle Attack ..................................... 63
  8.2.3 Combined Attack ................................................. 64
8.3 Proposed Solutions .................................................... 64
8.4 Constraints ............................................................. 65
8.5 Evaluation of the Research ........................................... 66
  8.5.1 Validity .............................................................. 66
  8.5.2 Reliability .......................................................... 66
  8.5.3 Replicability ....................................................... 67
  8.5.4 Ethics ............................................................... 67
  8.5.5 Sustainability ...................................................... 68
8.6 Discussion .............................................................. 68
8.7 Future Work ............................................................ 69
List of Figures

2.1 Architecture of the CAN network [41]. . . . . . . . . . . . . . . . . 9
2.2 SecurityAccess service [42] [44]. . . . . . . . . . . . . . . . . . . . . 16
3.1 White-box, black-box and grey-box testing. . . . . . . . . . . . . . . 26
5.1 The testbed for testing security requirements. . . . . . . . . . . . 38
6.1 The testbed set-up for the man-in-the-middle attacks. . . . . . . . . 47
7.1 Session key agreement, using symmetric cryptography. . . . . . . 59
List of Tables

4.1 The ECU’s possible security-related negative response codes for requests by the tester [42] [44]. ........................................ 30

5.1 Test results for ECUs A, B and G. ................................. 39
5.2 Test results for ECU C. .............................................. 40
5.3 Test results for ECU D. .............................................. 41
5.4 Test results for ECU E. .............................................. 42
5.5 Test results for ECU F. .............................................. 43
## List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN</td>
<td>Controller Area Network</td>
</tr>
<tr>
<td>DH</td>
<td>Diffie-Hellman</td>
</tr>
<tr>
<td>DLC</td>
<td>Data Link Connector</td>
</tr>
<tr>
<td>ECU</td>
<td>Electronic Control Unit</td>
</tr>
<tr>
<td>EMS</td>
<td>Engine Management System</td>
</tr>
<tr>
<td>KWP2000</td>
<td>Keyword Protocol 2000</td>
</tr>
<tr>
<td>OBD-II</td>
<td>On-Board Diagnostics II</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RSA</td>
<td>Rivest-Shamir-Adleman</td>
</tr>
<tr>
<td>UDS</td>
<td>Unified Diagnostic Services</td>
</tr>
<tr>
<td>VCI</td>
<td>Vehicle Communication Interface</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

xviii
Chapter 1

Introduction

Modern heavy-duty vehicles rely greatly on software running the embedded devices controlling their different modules - Electronic Control Units (ECUs). This thesis analyses the process of updating software on ECUs for security flaws, and is conducted in the global heavy-vehicle company Scania. This chapter introduces the background, purpose and general outline of the thesis project.

1.1 Background

Scania Group [40] is a global company, which produces a wide variety of trucks, buses and engines. They provide a complete technical, educational and financial service of managing an inventory of motor carriers. Among Scania’s services, workshop maintenance is the one relevant to this thesis. It is during a maintenance checkup that the latest software is installed on the vehicle’s computational units. Scania is responsible for the correct functioning of the vehicle, which additionally translates to the safety of its driver and passengers. Therefore, it is in Scania’s interest to ensure the integrity of the installed software.

A modern vehicle holds in itself an array of separate modules: the engine, breaks, air conditioner, locking, and dozens of others. Each module in a vehicle consists of mechanical components, as well as electrical sensors and actuators. An Electronic Control Unit (ECU) is an embedded system, which manages and controls a module’s electrical components by running software stored in its non-volatile memory [20] [22]. The ECUs in a vehicle communicate over a Controller Area Network (CAN), a message-based communication protocol [9]. This network can be accessed externally via a standard connector, which is commonly located under the dashboard [34]. During routine checkup, the CAN is connected to for diagnostic purposes: reading usage and performance data, and updating the software of ECUs [43] [40]. To make these processes
easier, specific application-level communication standards are in use: Keyword Protocol 2000 (KWP2000) [44] and Unified Diagnostic Services (UDS) [42]. The hardware for connecting to a vehicle’s internal electrical system is available and respective diagnostic communication standards are open as well. Therefore, anyone can set up an interaction with a vehicle that they have physical access to. KWP2000 and UDS define security measures, in order to guarantee that only authorised persons get access to read and write ECU software data.

1.2 Problem Description

ECUs are responsible for most of the functionality of a vehicle. Software bugs and miscalculated settings in an ECU may easily cause a vehicle to malfunction. This does not only pose an inconvenience for the business it serves, but could also potentially endanger the safety of its transportees or other travellers on the road. Therefore, it is of great importance that before a vehicle is handed over to the customer, the functioning and co-operation of all its modules is thoroughly tested.

Some parties may have interest in altering the software of an ECU - for example, a truck owner wishing to improve some performance indicators, or a criminal intending financial or physical harm. Unexpected behaviour of a vehicle resulting from untested software alterations may then lead to legal warranty cases. In such situations it may be difficult to prove that any manipulation has occurred, and the responsibility would lie on the manufacturer. Therefore, it is in the interest of the manufacturing company to minimize the possibility of ECU software being altered outside its own production units and verified workshops.

The industry standards concerning ECU software updating feature several security measures, but the degree of their implementation varies from device to device. Previous research has already demonstrated cases of overriding ECU software security in automobiles. However, lack of overview exists when it comes to the security of ECUs used in trucks and buses. This thesis poses three questions:

1. Which security vulnerabilities are present in the current solutions for downloading software updates to ECUs used in Scania trucks, buses and engines?

2. How can these vulnerabilities be exploited?

3. What are possible solutions for eliminating the vulnerabilities?
CHAPTER 1. INTRODUCTION

1.3 Purpose

The purpose of the thesis is to present a security evaluation of the process of performing software updates on different Scania ECUs. This includes a concise report of vulnerabilities, their impact, and possible solutions. Although security measures are programmed into ECUs to protect against unauthorised software installations, their sufficiency has been unclear so far. To find out if there is any need for improving the ECU security measures in the future, the current ones should be thoroughly mapped and challenged with experimental attacks.

1.3.1 Goals, Ethics and Sustainability

The evaluation of security is performed by testing the software update interface of an ECU for the presence of a set of security requirements. The thesis intends to bring the significance of the found vulnerabilities to the industry’s attention by means of experimentally demonstrating possible attack scenarios. It also aims to provide suggestions for improving the current situation. The project has three sequential goals:

1. Identify exploitable vulnerabilities in the ECU software update process
2. Demonstrate or explain, how such vulnerabilities can be practically exploited
3. Propose solutions that would eliminate the vulnerabilities

Since this is a thesis that performs security testing, ethical hacking principles are followed.

The thesis has potential to result in positive ethical consequences, in line with its main motivations. Any security vulnerabilities in downloading software on the control units of a vehicle or an engine pose a threat to its driver, passenger, operator, or other people travelling on the road. If software can be loaded on an ECU from other sources than Scania, then it must be considered that this altered software may not have gone through sufficient testing to be proven safe. It may be that the software is modified in a way, which makes the vehicle’s properties and capabilities illegal its country of operation. In extreme cases the software alterations can even be downright malicious.

Also, if such software does turn out to be harmful, it might be impossible to prove that it was indeed not put there initially by the manufacturer, and legal blame could be placed incorrectly. If vulnerabilities will be found in the course of this project, further steps can be taken by Scania to avoid all these unwanted consequences, and make their machinery safer to operate.
CHAPTER 1. INTRODUCTION

As a tradeoff, eliminating the possibility to make software alterations to ECUs takes away a degree of creative freedom from truck owners. Also, the vehicle owner may achieve a more powerful or capable vehicle by altering ECU software, which is something that the vehicle manufacturer would normally ask a higher fee for. The topic of sharing and copyrighting intellectual property is sensitive for both sides of the equation, and should not be discarded lightly. The arguments of potential hacking and harm to other travellers on the road oppose to this freedom. Nevertheless, when considering possible solutions to vulnerabilities, as a part of this thesis or something loosely connected, the interests and rights of both sides should be kept in mind.

Implementing stronger security solutions in ECUs requires extra work, and potentially more capable hardware. These are investments that companies may not necessarily be willing to make, unless risks are very high. When suggesting possible solutions within this thesis, the security/cost tradeoff must be considered, in order to arrive at realistic conclusions.

1.4 Methods

Research projects are classified as quantitative or qualitative, based on whether they are respectively numerical or non-numerical. In quantitative research, experiments are performed to verify or falsify measurable hypotheses, or a computer system’s functionalities. Qualitative research aspires to understand meanings, opinions and behaviours, in order to develop new hypotheses [18]. There are several abstraction levels, when it comes to specific research methods, and each has several methods, ranging between the ones used in quantitative and qualitative research.

Firstly, a philosophical standpoint is taken. The choice ranges from positivism, which assumes that the reality is independent of the observer, to criticalism, which sees the reality as constantly reproduced by people. Secondly, a research method is chosen. Different research methods either focus on collecting and analysing data, solving known problems or developing new ideas. Thirdly, it is necessary to understand how the research is approached. A deductive approach starts with a hypothesis and continues to either verify or falsify it. And inductive approach means that observations are made first, and followingly, new theories are formulated [18].

This thesis project is a quantitative research, since it aims to validate an ECU’s software update interface in terms of security. It is conducted from the philosophical standpoint of positivism: the general understanding is that ECUs either have specific security measures implemented or not. The security measures are posed as hypotheses, and presented in the form of software require-
ments. They are verified using experimental research methods: this method studies the cause-and-effect between variables [18]. In this thesis it is used to understand the effect that specific features (or lack thereof) in the studied ECU interface has on the security of the software update process. A deductive approach is taken to perform the study. The research involves testing ECU software to verify and validate the implementation of security requirements. It also involves performing experiments to explore the possibility for performing specific attacks, given that some tests on security implementations fail.

1.5 Delimitations

This thesis does not cover the whole ECU software update process. It focuses on the software interface of ECUs for performing diagnostic communication with an external node. Matters concerning cryptographic key management, as well as the distribution, production and signing of software files are not examined. This is so mostly because details of these topics are confidential, and cannot be the subject of a public research.

Due to a confidentiality agreement, some additional details have been left out from the report of experiment results. Namely, it is the calculated estimations on how much time it would take for particular attacks to succeed, as well as measured timespans of experimental attacks. Disclosing this information could pose a security risk on Scania ECUs.

To perform a thorough security analysis, the experiments performed in this thesis build upon several Scania-internal specification documents. These would not necessarily be available to a real adversary, who would have to spend considerable amount of time on figuring out the details of communicating to an ECU.

Also, for the sake of convenience, the research uses some software code developed internally in Scania to perform accurate software updates. The focus of this research is mainly on whether it is possible for an adversary to bypass security measures, and other aspects are optimised to save time.

Several attacks are performed as a part of this thesis. However, none of them attempt to attain the most valuable data - secret encryption keys. This is a very interesting and important question in terms of the purpose of this research. Building upon the results of this thesis, it could be explored as the next step.

The goal of the experimental attacks described in Chapter 6 is to program the ECU with arbitrary software. However, two of the three attacks (see brute-force attacks in Section 6.1 and Section 6.3) only focus on bypassing the authentication mechanism, and do not follow up with actual software updating.
Theoretically, the success of these attacks implies that software updating can be performed by an adversary, but it is not demonstrated.

1.6 Outline

Chapter 2, introduces the background for this thesis. It explains the concept of an ECU, as well as how its software is updated. Also, previous ECU security related work is mentioned, as well as how it is used in this research.

Chapter 3 explains general academic research methods, and how they are used in this quantitative study.

This thesis has three stages, corresponding to its goals. Firstly, Chapter 5 describes the process of identifying vulnerabilities. Several ECUs are tested, which are commonly used in Scania trucks. Test cases are based on the security requirements of relevant standards - KWP2000 and UDS. Also, related research and known security problems are taken into account. Tests are conducted to confirm or disprove the presence of potential issues. This involves using Scania internal general-purpose software and programming test scripts.

Secondly, vulnerable ECUs are used in the second stage to demonstrate the significance of found security holes. More complex experiments are devised and developed, than in the previous stage, to present a realistic hacking scenario and visible results. These experiments are presented in Chapter 6.

Thirdly, Chapter 7 proposes solutions to identified vulnerabilities, based on common security practices. Applicable protection mechanisms are found with the aid of literature study.

Finally, Chapter 8 concludes the findings of the research, evaluates them and outlines topics for future work.
Chapter 2

Theoretical Background

This thesis deals with Electronic Control Units (ECUs) [20] [22], the embedded devices responsible for the functioning of different modules in a vehicle. ECUs are connected in a distributed network called Controller Area Network (CAN) [46], which is a standardised protocol. The ECUs need to be accessed externally in the workshop, to perform diagnostic operations, as well as updating ECU software [33]. In order to keep an open market for workshop services, diagnostic communication protocols are also standardised [12] [13]. To ensure the integrity of ECU software and thereby road safety, these standards also define some security measures that can optionally be implemented by vehicle manufacturers [42] [44]. Their implementation, however, is not always sufficient, and previous research has already uncovered some vulnerabilities, as well as proposed possible solutions [35] [31] [28] [32].

2.1 Technological Concepts

Electronic Control Units (ECUs) are like small computers, controlling the components of a vehicle. It is necessary for them to communicate information between each other, as well as externally. This communication is standardised in the message-based Controller Area Network (CAN) protocol [46], which does not have any built-in security [32].

2.1.1 Electronic Control Units

ECUs [20] [22] are embedded systems, used in a vehicle to regulate and control its electrical subsystems. These modules are responsible for engine management, steering and breaking, vehicle performance and driveability, driver and passenger safety and comfort, and information provision. An ECU manages an
electrical system by gathering information from various sensors placed inside the vehicle, processing the received data, and sending instructions to other controllable elements of the system. For example, the ECU of an electronic clutch control system would receive readings from the engine speed sensor, process the data, and send instructions to the devices responsible for operating the clutch [20] [22]. ECUs communicate with each other and other devices over a Controller Area Network, a common communication standard in automotive applications (see Section 2.1.2) [22] [9].

The central element of the common ECU is the microprocessor (also referred to as central processing unit, CPU). The microprocessor’s task is to process raw data from sensors, as well as data and instructions from the memory, the driver’s console, or other ECUs. The CPU can be accompanied by several types of memories. Firstly, non-volatile memory stores core data and built-in instructions [22]. Secondly, volatile Random Access Memories (RAM) are used for storing disposable data during program execution [22].

The capabilities of ECUs are extremely limited, compared to desktop computers. For example, the Qorivva MPC5566 [14], a microcontroller designed for engine management, has a 132 MHz architecture, 3 MB of non-volatile flash memory, and 128 KB of RAM. Due to limited hardware, an ECU’s programming is very minimalistic, and computation-intensive security solutions are not preferred.

2.1.2 Controller Area Network

Controller Area Network (CAN) [46] is a standard used for communicating between intelligent electronic devices in the vehicle. It enables maintaining a distributed control system in a vehicle, where each ECU is independently responsible for controlling a separate module. It is a message-based protocol, using serial buses for data transfer. The advantages of CAN have contributed to its adoption in applications outside the automotive industry, as well. For example, CAN is used in railway, aerospace, medical and several non-industrial applications [9] [52].

Serial Bus

In CAN, data is transferred over a serial bus [9]. A serial bus is a communication cable, which transfers data one bit at a time, over one data line (as opposed to parallel communication, where several bits are sent simultaneously, over multiple data lines) [3]. The vehicle’s electronics are grouped together, and only one bus runs through each group, significantly reducing the amount of necessary wiring. The ECUs on one bus receive all the messages sent on that bus (see Figure 2.1).
CHAPTER 2. THEORETICAL BACKGROUND

[22]. This setup also allows the ECU to have a single CAN interface, instead of separate inputs for every device it needs to communicate with [52].

Figure 2.1: Architecture of the CAN network [41].

Messages

CAN is a message-based protocol [52]. A CAN message has a message identifier, as opposed to a device identifier. All nodes on the bus receive the message and can decide for themselves, whether it is of interest. The message identifier is also used to prioritise between messages, if several devices start transmission at the same time. Lower value, in this case, indicates higher priority. [9]

Since there are multiple devices connected to a single bus in CAN, it uses the CSMA/CD (Carrier Sense Multiple Access with Collision Detection) method [52] to avoid and tackle transmission collisions. “Carrier sense” means that a device needs to monitor the network and make sure the bus is not actively transmitting, before sending a message. “Multiple access” stands for the equal opportunity of every device on the network to transmit a message during a period of inactivity. “Collision detection” indicates that it is possible for nodes to detect if two devices are transmitting at the same time, and take appropriate action [52].

CAN uses the following message types [52] [9]:

- data frame - transmitting information to other nodes;
- remote frame - a request for data;
- error frame - informing of protocol errors defined by CAN;
• overload frame - notifying that the sending node requires more time to process received messages;

• interframe space - separating data or remote frames.

2.2 Vehicle Diagnostics

The vehicle’s computer system constantly generates and stores data about its functioning. These logs contain, for example, trouble codes and real-time data. This information can be used to troubleshoot any problems in the workshop, and subsequently perform repair work [33].

The system which performs the diagnosis, test, inspection or monitoring functions on the ECU is called a tester [42]. This may be a scan, test or software update tool used in the workshop. Most relevant to this thesis is the client function [42] of the tester, which makes use of diagnostic services. Respectively, the ECU implements a server function, which provides the diagnostic services. The security of ECU software update process is primarily a matter of the communication between the client and the server.

The physical means for communicating with the computer system (the Data Link Connector - see Section 2.2.1), as well as the abstract (the diagnostic communication protocols - see Section 2.2.3), are standardised in the industry. The main reason for standardising this connector is that the European law [12] [13] requires for diagnostic information to be readily accessible. It is the vehicle owner’s duty to fix any emission-related and safety-critical problems as soon as possible, and for that purpose no vehicle manufacturer can monopolise the market on the repair shops for their vehicles. Hence, all information related to the diagnostic communication setup needs to be standardised, or readily available. This standardised diagnostics system is called On-Board Diagnostics II (OBD-II) [33].

Having this access to the vehicle’s internal computer system also opens up the possibility of updating the software in an ECU’s programmable flash memory (flashing - see Section 2.2.2), without removing it from the vehicle, or having to replace it when a software fault is discovered [48].

2.2.1 Data Link Connector

The vehicle’s internal CAN network can be accessed externally through a standardised data link connector (DLC) [34] [45]. It is located in the passenger compartment, most often under the dashboard, near the steering wheel. The DLC enables physically interfacing with the vehicle’s on-board computing net-
work, to query diagnostic information and real-time data, and as a side-function - programming ECUs.

2.2.2 Software Update Process

The process of ECU software updating is referred to as *flashing*. Flashing is normally done either before the ECUs are placed into the vehicle, or when the vehicle is brought into a workshop for service, and there are software updates available for any ECUs. During flashing the *application software* in the ECU’s non-volatile (flash) memory is (re-)programmed [2]. Application software is what normally runs on a working ECU and performs its main functionality. It may be required by the manufacturer that ECU reprogramming could only be performed by *boot software*. Boot software is stored in the boot memory partition, which is protected from inadvertent erasure or modification. It supports a very minimal instruction set. In addition to the flashing process itself, the boot server is also responsible for verifying the presence of a valid application software and launching it after ECU power-up or restart. [42]

The software update process does not only involve the protocol of communication between the tester and the ECU. It also comprises of the management of authentication passwords and the security policy of the workshop computer. These matters are, however, internal to the vehicle manufacturer. In the case of Scania they are publicly undisclosed for security reasons. Therefore, they will not be touched upon in this thesis. Instead, the focus is on the implementation of communication protocols in the software of different ECUs.

2.2.3 Diagnostic Communication Protocols

Unified Diagnostic Services (UDS) [42] and Keyword Protocol 2000 (KWP2000) [44] are protocols, which define the communication between a tester and an ECU. When implemented on CAN, KWP2000 covers the application layer and UDS covers the application and session layers of the OSI (Open Systems Interconnection) model [23]. The protocols define methods for the following functional units [42] [44]:

- *Diagnostic and communication management* - managing the states of the diagnostics session
- *Data transmission* - low-level data access on the ECU
- *Stored data transmission* - accessing stored sets of diagnostic information on the ECU
- *Input/output control* - substituting ECU input or output values
CHAPTER 2. THEORETICAL BACKGROUND

- *Routine* - remote control of a routine
- *Upload/download* - transferring larger amounts of data

### 2.2.4 Flashing Process

The flashing process consists of several steps, each of which has a corresponding service in both standards [42] [44]:

**Step 1:** Setting the application session from DefaultSession to ProgrammingSession

**Step 2:** Going to boot mode (may be included in the request of the previous step)

**Step 3:** Setting the boot session from DefaultSession to ProgrammingSession

**Step 4:** Initialising data transmission parameters

**Step 5:** Transferring the application’s binary data from the tester to the ECU

**Step 6:** Terminating data transfer

The exact implementation of these steps may be according to standard, or vendor-proprietary, but the general outline remains the same.

### 2.3 Security Concepts

In computer science, security is a discipline focusing on the dependability of a system in the face of malice, error or mischance. It involves tools, processes and methods used to design, implement and test complete systems [1]. This section describes the security methods and attack types addressed in this research.

To grant access to sensitive operations or data in a computer system, a user needs to be authenticated and authorised. Authentication means ensuring that the user is who they claim to be. Authorisation means granting the right to access the requested resource to the authenticated user [6].

Sniffing [50] means monitoring the data passing across a network. It requires a sniffing tool to be installed into the network, which gathers sensitive data, for use by the attacker. Encryption is commonly used to protect data against effective sniffing [50].

Brute-force attacks are performed against authentication mechanisms. It involves trying out all possible passwords, secret keys or calculated authentication codes, until authentication is performed successfully. If the attack is performed against a live authentication server, it is called an online brute-force attack. If the attacker has attained some additional data about the password, they can perform an offline brute-force attack, by simulating the calculations...
done on the password to verify whether it is a correct match. Online brute-force attacks can be slowed down by adding time delays between sequential authentication attempts. Offline brute-force attacks can be rendered infeasible by using long passwords. Then there will be so many possible passwords, that trying them all out would take an unrealistic amount of time [24].

In a man-in-the-middle attack, two nodes appear to be communicating with each other directly, whereas in reality, their communication data is being sent and received by a malicious node in between them. This attack is called a passive man-in-the-middle attack, when the malicious node only relays the messages between the communicating nodes, and either analyses or stores them. Alternatively, it is called an active man-in-the-middle attack, when the malicious node alters the message contents, before sending it on [6].

Reverse engineering [11] is the process of extracting knowledge about the design of a system. Software can be reverse engineered for various purposes, including acquiring valuable information from proprietary products. To reverse engineer software, the executable file is processed with various tools, like disassemblers, decompilers and debuggers. The aim is to extract the sequence of instructions coded into the program, and thereby understand the algorithm used. Sometimes, sensitive data can also be uncovered, if it is hardcoded into the software [11].

### 2.4 Security Mechanisms in Vehicle Diagnostics

The ECUs used in Scania vehicles come from different manufacturers, such as Stoneridge [49], Wabco [51] and Bosch [16]. They have varying processing capabilities and memory capacity. Also, some ECUs have more security-sensitive tasks than others. For example, an engine management system (EMS) has the very critical task of combusting fuel and transferring energy [26], and vehicle owners may be rather interested in tweaking its functionality change its performance. However, the resulting performance indicators may be illegal and unsafe, or the desired functionality might be provided by the vehicle manufacturer for an increased price. Therefore it is very undesirable to allow vehicle owners to change the programming of an EMS. ECUs with such important tasks tend to have a stronger security implementation. On the other hand, the ECU controlling the speed and operation of windshield wipers, for instance, might not need to waste resources on processing security functions, as it being exploited does not bear high risks.

There is no concise overview of different security mechanisms in use in Scania ECUs. Therefore they need to be experimentally mapped by testing ECUs for common criteria.
In computer science, three classical security requirements are always considered [17]:

- **confidentiality** - sensitive info is not disclosed to unauthorised parties;
- **integrity** - sensitive info cannot be modified by unauthorised parties;
- **availability** - information and resources are not withheld from legitimate users.

From the perspective of ECU software update, the question of integrity is the most prevalent. It is desirable that the software that ends up running a large vehicle on the road is the same as intended and tested by Scania. If the software gets modified in any way, it immediately poses a potential safety, legislative or financial risk.

Ciniglio et al. define the following security goals specifically for the ECU flashing process [7]:

- **error detection** - the ECU must be able to check if the flashed software is corrupted due to transmission problems;
- **authenticity** - it should be guaranteed that the software comes from a legitimate source;
- **copy protection** - the software should be copied only on a determined ECU;
- **confidentiality** - no unauthorised party should be able to read the software;
- **authorisation** of the external programming tool (diagnostic tester) towards the ECU - the diagnostic tester should be suitably recognised by the ECU, in order to decide if the flashing has to be permitted or not.

Some of these matters - like error detection, confidentiality and authorisation - are covered by security suggestions in diagnostic communication standards. The extent and details of their implementation, however, is entirely up to the vehicle manufacturer. Although it is in the manufacturer’s interest to have proper security measures in place, it may not always be financially cost-efficient, and the standards freely allow for a completely unsecured implementation.

### 2.4.1 Security in the Controller Area Network

The CAN standard does not define any inherent security mechanisms. Messages between ECUs are sent in plaintext and no prior authentication is required. This
opens the possibility for a malicious node in the network to masquerade itself as a legitimate node. In the message-based CAN, this means that a malicious node is sending messages, which it is not supposed to be entitled and configured to send [32].

Furthermore, since all nodes receive messages sent on the bus, then anyone can sniff on the communication, given they have physical access to the network. A malicious node can listen on the bus and store messages from legitimate nodes, and replay the messages later by transmitting exact copies of them to the network [32].

In the case of ECU software update, the original CAN standard is only used on the physical and data link layer [46]. Some security mechanisms are added in the application layer, by the UDS and KWP2000 protocols (see subsection 2.4.2).

2.4.2 Authentication and Authorisation in Diagnostics Standards

Anybody can connect a device to the CAN network of a vehicle, and communicate with ECUs using messages according to the UDS or KWP2000 standard. Therefore, some services and memory locations can be secured - they can only be accessed, if the diagnostic session has entered the unlocked state from the default locked state. Different security levels can be defined for different services, only one of which can be active at a time.

Entering the unlocked state is only authorised, if an authentication challenge is successfully passed. For authentication, the SecurityAccess service is used. As presented in Figure 2.2, the client first requests a seed from the ECU (server) with the requestSeed subfunction. The ECU replies with a seed. The client, then, calculates an authentication code from the seed sent (using a cryptographic function of the vehicle manufacturer’s choice, which may utilise a secret key shared between the tester and the ECU). It sends the calculated authentication code back to the ECU, which then compares it to an internally stored or calculated value. If they match, then security access to the requested security level is unlocked.

The length of seeds and keys, as well as the algorithm for authentication code calculation is not standardised. It is normally only a few bytes, due to the generally low processing capabilities of ECUs [8] [31] [35]. To avoid brute-force attacks on such short keys, the server may implement time delays before responding positively to a requestSeed message. This should be done after a server power-up/reset, and after a certain number of false access attempts.

Additionally, an attempt counter can be implemented - after a certain num-
ber of unsuccessful attempts to gain security access, the tester will automatically be denied access. On the other hand, it is also allowed to use static or semi-static seeds, which may make it easier to break in to the secure mode. The specific implementation of these possibilities is completely up to the manufacturer. [42] [44]

Figure 2.2: SecurityAccess service [42] [44].

Technically, it is possible to use a static authentication code, which is the result of mapping, rather than a mathematic calculation. Common solutions, however, use symmetric-key cryptography for authentication code calculation. This means that the ECU and the tester share one secret key, which is used both for encrypting and decrypting [1]. The modern alternative to symmetric cryptography is asymmetric cryptography (also called public-key cryptography), which employs different keys for encrypting and decrypting [1]. Public-key infrastructure (PKI) may be too cumbersome for the limited processing power and memory capacity of an ECU. At least one PKI solution has been suggested [7], but generally PKI has not made it to wide use in ECUs at the time of this writing.

2.4.3 Encryption

Encryption involves disguising information, so that the result would be illegible to a bystander [27]. It is commonly used to ensure the confidentiality of information.

For encrypting sensitive data, the UDS protocol provides the possibility to execute services through the SecuredDataTransmission service [42]. It requires implementing a security sub-layer in the application layer of both the tester and the server. The security sub-layer is responsible for encrypting and decrypting the whole message of the intended service. The encrypted message is encapsulated in the data field of a SecuredDataTransmission message, which is sent as a regular UDS service request or response [42].
CHAPTER 2. THEORETICAL BACKGROUND

Theoretically, this could strengthen the confidentiality and integrity of software data while flashing. In practice, however, the flashing process rarely employs the SecuredDataTransmission service.

2.5 Possible Attackers

Before starting a security analysis, it is necessary to define, who might have an interest in performing an attack, what are their motivations and what attributes do they have available.

Vehicle owners are probably the most likely to be tinkering with ECUs and their software. The most interesting ECU for them must be the Engine Management System (EMS). To comply with the European law [12] [13], the EMS software sets restrictions to the engine power, which ensure that emissions stay within the allowed limits. A truck owner might want to disable those restrictions, to increase the engine power. To do that, they would have to make respective changes in the EMS software. Normally they would be able to purchase all necessary hardware to connect to the CAN network via the DLC. However, they would not be able to attain the SecurityAccess secret keys.

Workshops do not have direct access to secret keys either, but they are able to use the Scania software, which performs SecurityAccess. Because of the anti-monopoly law [12] [13], there are numerous workshops that are independent from Scania [40], and yet are provided the tools necessary to perform all operations on Scania ECUs. They might be able to use these privileges to flash other software on ECUs, than the official updates. If so, they could earn some extra income from interested truck owners.

A different kind of an attacker would be someone whose intention is to do harm to the vehicle, whatever the underlying reason. This could be accomplished by changing the functionality of the Brake Management System ECU, for example. Such an attacker has no immediate access to ECU update software nor SecurityAccess secret keys. Additionally, they have great difficulty getting physical access to a vehicle that does not belong to them.

2.6 Related Work

Several successful attempts have been made to take control over critical services of an ECU. From the perspective of protection, ECU flashing is no different from any other sensitive function. Therefore, the following attacks help in defining relevant vulnerabilities for the goals of this thesis.
2.6.1 Bypassing Authentication

Miller and Valasek [35] recently demonstrated how to take control over some functionalities of Ford Escape and Toyota Prius cars, like steering, accelerating and disabling the brakes. They also managed to reprogram some ECUs of the named vehicles. The authors use the following methods to bypass SecurityAccess:

- \textit{Brute-force attack} - In the described version of this attack, the goal is to uncover the value of the authentication code, not the secret key. The attacker repeatedly requests a seed from the ECU, and responds with a different authentication code value in the \texttt{sendKey} message every time, until access is granted. Given that the attacker has no information on the algorithm for encrypting the seed, this attack is only possible if the seed is static (i.e. it never changes) and short. The seed being static means that the expected authentication code is also static. If all possible authentication code values are tried, the attacker eventually comes across one that is correct. Other sources [31] claim that the authentication code is likely to match the length of the seed, and a seed-key pair of 4 and more bytes is infeasible to brute-force, due to the anti-hacking timing delays implemented on the SecurityAccess service by vehicle manufacturers. The given example shows a seed and authentication code of 3 bytes.

- \textit{Extracting and reverse engineering the ECU firmware} - In the case of dynamic and/or longer seeds, the secret key is needed, as well as the algorithm for calculating the authentication code. The ECU would either store both of these, or the calculated authentication code itself. The authors suggest extracting the firmware from the ECU and reverse engineering it to uncover the necessary information.

- \textit{Reverse engineering diagnostic software} - The secret keys and algorithms are also stored in the official diagnostic tool of the specific vehicle. It might be easier to reverse engineer that software instead, if one has access to it. The authors preferred this method to the previous one.

2.6.2 Deviation from Standards

Koscher et al. performed a similar study to explore the ways how a car could be subjected to a hacker’s will [28]. Their results are mostly based on the observation that several standard security requirements are not (sufficiently) implemented in the cars chosen for the study. Some of these observations may be relevant to this thesis, and it may be worth verifying whether these shortcomings are present on Scania vehicles and engines as well:
• It was possible to enter the reflashing mode of the ECU controlling the engine, while driving, which resulted in the engine stopping.

• Some sensitive services, such as flashing an ECU, did not actually require authentication at all.

• Neither was authentication performed when reading the ECU memory, including the memory areas where authentication codes were stored.

2.7 Application

This thesis will expand on the previous works done in the field of ECU security. The purpose is to compile a fuller picture of the vulnerabilities specific to the ECU software update process, particularly in the ECUs used in Scania trucks, buses and engines.

As explained in Section 2.4, Ciniglio et al. define these five security goals for the ECU flashing process [7]: error detection, authenticity, copy protection, confidentiality and authorisation. In the process of producing results fitting the purposes and preconditions for this thesis project, all these security requirements are touched upon, using previous theoretical and experimental research.

2.7.1 Experiments for Confirming Potential Vulnerabilities

Many vulnerabilities are the result of the insufficient implementation of suggested security features, as described in subsection 2.6. Therefore, it would be the first order of business to test all the security requirements and suggestions of UDS/KWP2000 on a couple of Scania ECUs that might be of interest to hackers.

An experiment will be planned to explore the possibility of a man-in-the-middle attack in the process of ECU flashing. As pointed out by Lin and Sangiovanni-Vincentelli [32], CAN is susceptible to replay. Standards [42] [44] rely on the SecurityAccess service for ensuring authenticity and performing authorisation. Thereby, if one of these security requirements is compromised, then so is the other. To bypass authentication and flash an ECU with their own software, a malicious party can possible use a man-in-the-middle attack or a brute-force attack, both of which are tested in this research. Although the diagnostic communication standards [42] [44] have error detection mechanisms in place in the form of message checksums, they are not intended to provide protection from a man-in-the-middle attack - rather from accidental errors - and can be easily bypassed.
CHAPTER 2. THEORETICAL BACKGROUND

A man-in-the-middle attack could also compromise confidentiality, as it enables the adversary to read the data that is being transferred between the programming tool and the ECU. Once the adversary learns the contents of the flash files, they could flash them on a different ECU (provided that they can bypass authentication again). This, however, is a violation of the copy protection requirement. And so it is that all five security requirements are explored within the experiments.
Chapter 3

Methods

The aim of this thesis is to find which security vulnerabilities exist in the software update process of Scania ECUs. Appropriate security measures should prevent the specific event of an unauthorised entity being able to update an ECU with arbitrary software. Whether such security measures are in place, is determined with a quantitative study. Experimental research determines, whether a list of security requirements is fulfilled in the case of several ECUs. To carry out the experiment, software testing methods are used. From the test results, the strength of ECU security is deducted.

3.1 Academic Research Methods

This thesis applies quantitative research [18]: a method, which aims to verify or falsify measurable hypotheses. In terms of computer science, quantitative research is used to verify a computer system’s functionalities and interfaces. Quantitative research can be designed either on experiments, surveys or case studies. The chosen research methodology directly implies the data collection method [18]. Experimental methodology [18] is the best for verifying hypotheses about the correlation of independent and dependent variables. Experiments are performed to collect data about the effects of changing variable values. Surveys [18] help describe relationships between variables in phenomena that cannot be directly observed. In a survey, questionnaires are used to ask questions and thereby collect data indirectly. Case studies [18] focus on one particular instance of a phenomenon, and enable mixing in elements of a qualitative research. Data is collected by studying one or a couple of participants in depth.

Once data is collected, it is analysed to make conclusions and bring the research to a result. To turn qualitative data gathered in surveys into quantitative, it can be categorised by coding [18]. Numerical or coded data gathered over
CHAPTER 3. METHODS

a large population can be analysed by statistical means [18], which enables making conclusions across a sample. On strictly numerical data, it may be necessary to use computational mathematics - apply algorithms, perform calculations and create models and simulations [18].

The goals of this thesis include finding security vulnerabilities in the software update interfaces of multiple ECUs, and verifying that they pose a threat. This can be best done by experimental means: testing the ECUs for specific security requirements and collecting data about the degree of their implementation. Calculations are performed on the collected data to evaluate the feasibility of an attack.

3.2 Application of Methods

In determining the strength of an ECU’s security, there are several interdependent variables at stake. For example the length of the cryptographic seed in authentication and the number of allowed failed authentication attempts in combination determine the amount of time that a brute-force attack would take on average. It is the combination of different variables that gives the security heuristic, not necessarily the specific values on their own. As these variables are directly measurable, then experiments are the most appropriate methodology for the goals of this research.

The ECUs to be tested are not chosen randomly. The aim is to focus on ECUs that might pose a greater interest to hackers, as well as to find a variety of security strengths. The goal is not to provide a statistical overview of security methods used, but rather to emphasize the advantages and disadvantages of different implementations. The results for each tested ECU are analysed and presented separately. Computations are performed to determine the feasibility of several proposed attacks on each test subject.

3.3 Quality Assurance

Once the research data is collected and analysed, the process and results need to be validated and verified. The quantitative research method requires the evaluation of the following quality criteria [18]: validity, reliability, replicability and ethics.

Validity means that the research has measured what was expected to be measured [15]. In the case of this research, the strength of security is measured via variables, which combine into security heuristics. The validity of research results depends on whether security vulnerabilities deducted from measurements
actually open up the way for realistic attacks. To ensure the validity of the test results as such, any found security vulnerabilities will be further explored with experimental attacks. If the attacks succeed, it confirms the test results in terms of security strength (or lack thereof).

Reliability refers to the stability of measurements [15]. In measuring an ECU’s security implementation with automatic means there is little room for error. However, implementing an attack on the (lack of) security measures has a very high random component for reasons explained in Chapters 5 and 6. Therefore, the experimental attacks need to be conducted several times over to ensure reliability of results.

Replicability is the possibility for someone else to perform the research and arrive at the same results [5]. It requires for the author to include enough information in the research report, so that it could be repeated and verified externally. For the purpose of replicability, some factors contributing to the security of the ECU software update process are left out of this research, as they are internally defined in Scania, and classified to the general public. This includes, most importantly, the secret key management system for authentication. Instead, the implementation of publicly defined diagnostics protocols is focused on. The research does use some information that cannot be disclosed, as well as code developed internally in Scania, but care is taken that the research questions are posed and answered in a way that can be replicated without access to similar information, albeit in a slower pace.

Ethics is the application of moral principles in every aspect of the research [36]. Since this is a thesis that performs security testing, the author puts herself in the role of a hacker. This means that ethical hacking principles should be followed. This includes obtaining permission from the organisation whose devices or networks are tested, respecting non-disclosure agreements, reporting all found vulnerabilities, and causing no deliberate harm [4]. As for the impact of the research, the research is evaluated in terms of whether it contributes to road safety and avoiding the incorrect placement of legal blame, as well as restricting the creative freedom of vehicle owners and protecting the intellectual rights of the vehicle manufacturers.

Additionally, proposed solutions are evaluated in terms of sustainability. The evaluation focuses on the question, whether the proposed solutions are feasible to be implemented, and how likely it is to carry out the suggestions resulting from this research.

The research will be evaluated based on these criteria in Section 8.5.
3.4 Software Testing Methods

Although diagnostics standards may prescribe a fairly secure system, the real security lies in the implementation. As previous research suggests, serious vulnerabilities may be found in a faulty or insufficient implementation. Therefore, software testing methods are used in this research for data collection and analysis.

Software testing is “an activity used to reduce risk and improve quality by finding defects” [19]. This section introduces the main concepts and methods used in software testing, and how they are applied in this research.

3.4.1 Verification and Validation

A software product is developed to meet certain user needs. These needs are used as a basis to engineer the future product’s requirements. The requirements are then used to design and develop the product [30].

When testing, essentially one of two things is being done: verification or validation [19]. Verification means checking whether a product meets the official requirements in the specification document. In the case of this research, the tested security requirements are defined in the KWP2000 and UDS standard documents. Validation means checking whether the product actually meets user needs. This research validates if the selected ECUs are secure enough, so that an unauthorised party would not be able to alter their software.

Verification against standards is a very good starting point, as standards define a fairly complete security system. However, they also leave details of many requirements open (e.g. cryptographic seed length). Validation helps to confirm whether a specific implementation actually meets the real need for security.

3.4.2 Positive and Negative Testing

For testing every requirement or need, several test cases are designed. A requirement of a function normally states some input and a respective output. A positive test runs the function with an input stated in the requirements, and sees whether it results in the respective output. It is called “positive”, since it tests what the program is supposed to do. A negative test sees what happens, when the function is run with an unexpected input. It is “negative” in the sense that the test case tests what happens when the user tries to make the program behave in a way that it is not supposed to behave. [10]

For example, consider testing a key and a lock. A positive test would involve trying if a correct key opens the lock, when turned counterclockwise in the
keyhole, and locks it, when turned clockwise. A negative test would involve inserting a different key in the lock, and turning it both ways, or trying to pick the lock with a piece of wire - inputs that are not in conformance with what the lock has to do in a usual case, but which the complete product has to handle anyway. [10]

3.4.3 Integration Testing

Integration testing is a term used for the testing of interfaces - the boundaries of information exchange for different components of a computer system [10] [38]. An ECU has an external interface for communicating with another node on the CAN network - meaning that the software running on the ECU reacts to a specific set of commands sent to it from the CAN (such as commands to program the ECU with new software). In this thesis, this interface is tested with regard to the software update process.

In integration testing, all interactions between the communicating nodes need to be tested. Some of these interactions are documented (explicit) and some are not (implicit) [10]. In the case of ECUs, interfacing is achieved with a communication protocol (KWP2000 or UDS). The explicit interactions are documented in public standards specifications, which are available for anyone to study. However, there are also implicit interactions. These protocol details are internal to the vehicle manufacturing company. This does not mean that these interactions cannot be used by an adversary. It only means that in an everyday situation, the adversary is assumed to not know about them. This is “security through obscurity”, and should not be relied on.

3.4.4 Grey-box Testing

A software tester may perform tests acting as an end user of the product, or they may instead go through the program code. In black-box testing, the tester has no access to the code - the focus is on the software’s external functionality. White-box tests are based on an analysis of the internal code structure of the software. Grey-box testing refers to a mixture of white-box testing and black-box testing. Figure 3.1 illustrates these different methods, treating the software as a box with code inside, and externally usable functions (illustrated as F1 and F2). Grey-box testing is performed mostly on the external functions, but sometimes parts of the internal structure are also studied for clarity and to make further decisions [21] [10].

When testing ECU security, the role of a hacker is assumed. Normally, they would not have any knowledge of the software’s inner workings, and treat the interface as a black box. However, there are two reasons why it is a good
idea to not only perform black-box testing within this research, but to mix in some white-box testing as well. Firstly, since the research is carried out on site in Scania, then internal documentation and software code is available for use. This reduces the need for a trial-and-error search for answers about the interface functionality, and thereby saves time. Secondly, security through obscurity cannot be relied on - if a functionality or a bug is present, it can theoretically be found and used, even when it is not publicly documented.

3.4.5 Test Automation

In some cases it is better to automate the testing process with software tools, instead of performing tests manually [21]. This involves testing activities which

- need to be repeated many times over,
- are very slow to carry out manually or
- cannot be performed accurately manually.

One could buy an appropriate tool, or develop one on their own. Sometimes, open-source options are available, which can be modified to the tester’s needs. In this research, the code of Scania internal software is used and modified for testing purposes. This option allows for speedy development, and is very flexible.
Chapter 4

Test Plan

Security vulnerabilities may lie in faulty or insufficient implementation of security requirements. Security testing employs classical software testing methods to find any such vulnerabilities, if present. Several ECUs are tested against the KWP2000 and UDS standard documents. The aim is to collect data enabling to determine whether all suggested security measures are implemented sufficiently well to deter possible attacks.

The following subsections describe which security requirements are tested, why and how. Some of them have been chosen on the basis of the results of previous research, and others represent the remaining security requirements in the standards. The results of executing these test cases are described in Chapter 5.

4.1 Attack scenarios

There are several possible attack scenarios, which could be tested if an ECU with weak enough security is found. Here, these scenarios are defined generally. Once the security mechanisms of each ECU are mapped, it is possible to define how exactly each attack could be implemented.

In scenario 1, there is no effective authorization mechanisms in place to stop the attacker from simply connecting with the ECU and loading new software on it. In scenario 2, the attacker reads the ECU software from memory, reverse engineers it and recovers secret keys. Scenario 3 is a brute-force attack where the attacker chooses a value to use as the authentication code. This value is given to every received seed, until a seed is received, to which this authentication code happens to correspond to. Then, access is granted. Scenario 4 is also a brute-force attack. In this one, the attacker has prior knowledge of a valid seed and authentication code pair. They will query seeds without replying, until the
known seed is received. Then, the known authentication code is sent as a reply, and the higher security level is unlocked.

The following security requirements are chosen for testing, as they could prevent these attack scenarios. Poor implementation may, however, open the possibility for them to be executed.

4.2 Security Requirements

The first experiment focuses on finding flaws in the implementation of standard security features. Based on the standard specifications [42] [44] and experimental results from other sources [28] [35] [31], the following requirements will be tested:

1. The ECU is unlocked only if a SecurityAccess service is called and successfully responded to.

2. Flashing operations on the ECU can only be performed in the unlocked state.

3. Reading sensitive information (e.g. secret cryptographic keys) from the ECU memory is functionally impossible or possibly only in an unlocked state.

4. The SecurityAccess seed is non-static and random.

5. The SecurityAccess seed, key and authentication code are so long that any brute-forcing would be rendered infeasible. This means that trying through all possible seeds or keys would take so much time that the brute-force method could not possibly be of benefit.

6. A time delay of sufficient length to mitigate brute-force attacks is implemented after a certain amount of failed SecurityAccess attempts.

7. A time delay of sufficient length to mitigate brute-force attacks is implemented for running the SecurityAccess service after ECU power up and reset.

8. Effective time delay is unaffected by changing sessions.

9. Time delay is equally effective on requests coming from all addresses.

10. Software data is transferred in encrypted form with the SecuredDataTransmission service.
CHAPTER 4. TEST PLAN

4.3 Technical background

Different sessions may have different seed and secret key characteristics, and authentication code calculation algorithm. The testing is focused on the Programming-Session, in both application and boot mode.

The impact of the aforementioned security requirements is very interdependent. Therefore, to evaluate the security of a specific ECU, all relevant aspects are tested first without making a final judgement as to the suitability of the specific results. Following the tests, experiments are performed to see if the particular combination of security measures can be bypassed to gain access to sensitive data and functions. Also, calculations are made to see how long it would take to gain access using brute-force methods. Based on these experiments and calculations, recommended values can be named for the length of cryptographic seeds and keys, as well as time delays.

Testing the items in the list of security requirements include sending the SecurityAccess service requests to the ECU and analysing the responses. The responses may include seed values or negative response codes. The latter help determine whether security requirements have been implemented (see Table 4.1).

4.4 Test Cases

The security requirements listed in Section 4.2 are tested with the following test cases. There is one or more test cases corresponding to each requirement. Some of the requirements are tested only with negative test cases because it is important to approach those points from a hacker’s perspective. A hacker does not approach the ECU interface with the desire to go through the security controls as is stated in a use case for updating ECU software, because they are unlikely to have the necessary secret keys right away. Instead, they try to see if there is any other possible sequence of actions that will eventually give them access to the software updating functions. Therefore, some functions are tested with unexpected input to see whether the security still holds.

Each test case consists of a sequence of steps and an expected result. The expected result is what a well enough secured system would give, if the described steps are followed. If the given steps have the expected result, the test is considered passed, otherwise the test is failed. The test case may also define measuring something, the value of which will play a role in determining the result of the test. The measurements are used for calculating the feasibility of the brute-force attack scenario (scenario 3 described in Section 4.1). The results of carrying out these test are presented in Chapter 5.
<table>
<thead>
<tr>
<th>Preceding request</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[any]</td>
<td>generalReject</td>
<td>A general error message</td>
</tr>
<tr>
<td>[any]</td>
<td>securityAccessDenied</td>
<td>The specific request would require the ECU to be in an unlocked state, but the needed security level is locked</td>
</tr>
<tr>
<td>[any]</td>
<td>invalidFormat</td>
<td>The format of the parameters does not match the prescribed format for the requested service</td>
</tr>
<tr>
<td>[any]</td>
<td>subFunctionNotSupported</td>
<td>The sub-function parameter (security level) is not supported</td>
</tr>
<tr>
<td>[any]</td>
<td>incorrectMessageLength</td>
<td>The length of the message is wrong</td>
</tr>
<tr>
<td>SecurityAccess</td>
<td>invalidKey</td>
<td>If the sendKey message contains an unexpected authentication code value</td>
</tr>
<tr>
<td>SecurityAccess</td>
<td>exceededNumberOfAttempts</td>
<td>Delay timer is active due to exceeding the maximum number of allowed false access attempts</td>
</tr>
<tr>
<td>SecurityAccess</td>
<td>requiredTimeDelayNotExpired</td>
<td>Delay timer is active</td>
</tr>
</tbody>
</table>

Table 4.1: The ECU’s possible security-related negative response codes for requests by the tester [42] [44].

4.4.1 Requirement 1: Unlocking

This is a negative test case to see if the server state will remain locked after sending the wrong authentication code within the SecurityAccess service.

Test Case 1: Unlocking

Step 1: Enter ProgrammingSession

Step 2: Send SecurityAccess.requestSeed message

Step 3: Send SecurityAccess.sendKey message with random authentication code

Step 4: Send SecurityAccess.requestSeed message. Expected result: server should reply with a new seed, or an error message referring to time delay being activated. If security is unlocked, then by standard, the ECU places 0 instead of the seed value in the securitySeed message.
CHAPTER 4. TEST PLAN

4.4.2 Requirement 2: Flashing Operations

If requesting sensitive services really does require a certain security level to be unlocked, then the ECU is expected to reply such requests in a locked state with a `securityAccessDenied` message. To determine whether flashing is properly secured, this experiment will test certain functions listed in subsection 2.2.4 in the presented order. The aim is to find out which services will be denied with the `securityAccessDenied` message, and which will be allowed. This requirement features one test case for each critical flashing-related service.

Test Case 2.1: Going to Boot Mode

Step 1: Enter `ProgrammingSession` in application mode.

Step 2: Go to boot mode. Expected response: `securityAccessDenied`.

Test Case 2.2: Initialising Parameters

It may happen in some cases that when the ECU starts up in application mode, then it will only require one round of `SecurityAccess` before going to boot mode, and no extra rounds before performing flashing activities in boot mode. In that case it must be checked that the ECU is in a locked state also when it has started up in boot mode.

Step 1: Make sure there is no application installed on the ECU, so that it starts up in boot mode.

Step 2: Enter `ProgrammingSession` in boot mode.


Test Case 2.3: Transferring data

Step 1: Make sure there is no application installed on the ECU, so that it starts up in boot mode.

Step 2: Enter `ProgrammingSession` in boot mode.

Step 3: Transfer data from the tester to the ECU. Expected response: `securityAccessDenied`.

4.4.3 Requirement 3: Reading Memory

Both UDS and KWP2000 define a service for reading the ECU memory, which takes the starting address and data size as parameters: `ReadMemoryByAddress`. If implemented and unprotected, this function may allow reading software bytecode from the ECU memory. This, in turn, can be reverse engineered to uncover
encryption keys and algorithms. Similarly to testing the previous requirement, this experiment will also perform the function in a locked state to see if it will be replied with a securityAccessDenied message.

Test Case 3: Reading Memory

Step 1: Enter ProgrammingSession in application mode.

Step 2: Send the ReadMemoryByAddress request. Expected response: securityAccessDenied.

4.4.4 Requirement 4: Seed Unpredictability

So as to mitigate brute-force attacks on the answer to the SecurityAccess challenge, the seed should be unpredictable. Firstly, it should not be static, since a static seed implies a static authentication code. This means that not only is the authentication code easy to brute-force for a persistent hacker, but it also suffices to just sniff a SecurityAccess service once to reveal it for further use. Secondly, a seed should be random enough to be non-repeating and unpredictable. If the seed is random, then it would be useful to know, how long one would theoretically have to query for seeds, until one repeats.

Test Case 4.1: Seed Unpredictability

Step 1: Enter ProgrammingSession

Step 2: Send SecurityAccess.requestSeed message (should be repeated for about 20 times in order to determine how random or static the seed is). Expected result: all seeds are different and appear not to have an obvious pattern.

Test case 4.2: Time to Query Seeds

Step 1: Enter ProgrammingSession

Step 2: Send SecurityAccess.requestSeed message (repeat 10000 times)

Step 3: Measure time passed. Expected result: querying many seeds in a row takes so long that coming across a repeated value in feasible time is improbable.

4.4.5 Requirement 5: Long Seed

The seed should be so long that, taking into account time delays after failed access attempts and reset (if present), it would be infeasible to brute-force the
secret key or authentication code. Whether or not a brute-force attack is feasible will be calculated and evaluated individually for each tested ECU.

Test Case 5: Long Seed

Step 1: Enter ProgrammingSession

Step 2: Send SecurityAccess.requestSeed message. Expected result: a seed is acquired, which is so long that a brute-force attack is rendered infeasible.

4.4.6 Requirement 6: Time Delay

If security access attempts are not bound with sufficient time delays, then brute-force attacks even on long seed-authentication code pairs may become feasible.

Test Case 6.1: Presence of Time Delay

Step 1: Enter ProgrammingSession

Step 2: Send SecurityAccess.requestSeed message

Step 3: Send SecurityAccess.sendKey message with random authentication code

Step 4: Keep repeating last 2 steps. Expected result: at some point, the server should reply with the requiredTimeDelayNotExpired message (subFunctionNotSupported, invalidFormat or generalReject also possible).

Test Case 6.2: Failed Attempt Count

Step 1: Repeat the steps of Test Case 6.1.

Step 2: Measure the amount of allowed failed attempts.

Test Case 6.3: Length of Time Delay

Step 1: Repeat the steps of Test Case 6.1.

Step 2: Keep sending SecurityAccess.requestSeed and SecurityAccess.sendKey messages. Measure the length of time delay. Expected result: Given the time delay and seed length, a brute-force attack as per scenario 3 is rendered infeasible.
4.4.7 Requirement 7: Time Delay After Power-Up

Some ECUs may take as little as milliseconds to be reset. In that case it may be easier for an attacker to reset the ECU, rather than wait for time delay to expire. Therefore, a time delay should be implemented on the SecurityAccess service also right after powering up the ECU.

Test Case 7.1: Presence of Time Delay After Power-Up

**Step 1:** Enter ProgrammingSession

**Step 2:** Send SecurityAccess.requestSeed message

**Step 3:** Send SecurityAccess.sendKey message with random authentication code and verify that it was not accepted

**Step 4:** Reset the ECU

**Step 5:** Enter ProgrammingSession after ECU has powered up again

**Step 6:** Send SecurityAccess.requestSeed message immediately. **Expected result:** the server should reply with the requiredTimeDelayNotExpired message (subFunctionNotSupported, invalidFormat or generalReject also possible).

Test Case 7.2: Length of Time Delay After Power-Up

**Step 1:** Repeat the steps of Test Case 7.1.

**Step 2:** Keep repeating. **Measure** the length of time delay. **Expected result:** the time delay after power-up is at least as long as measured in Test case 6.3.

4.4.8 Requirement 8: Changing Sessions

In preliminary testing it appeared that sometimes the ECU replied with a new seed earlier, if the session was changed to DefaultSession and then back to ProgrammingSession. The reason for such behaviour is unknown, and the presence of it indicates that this may be a way to bypass time altogether.

Test Case 8: Changing Sessions

**Step 1:** Enter ProgrammingSession

**Step 2:** Send SecurityAccess.requestSeed message
CHAPTER 4. TEST PLAN

Step 3: Send `SecurityAccess.sendKey` message with random authentication code and verify that it was not accepted.

Step 4: Repeat requesting seeds and sending invalid keys until the request is replied with `exceededNumberOfAttempts` or other general error message.

Step 5: Enter `DefaultSession`.

Step 6: Enter `ProgrammingSession`.

Step 7: Repeat requesting seeds until the ECU replies with a new seed. **Measure** the length of time delay. **Expected result:** Time delay is equal to the one measured with Test Case 6.

### 4.4.9 Requirement 9: Multiple Testers

This test aims to see how the ECU reacts if there are multiple testers connected to the CAN bus, each with a different address, and each querying for seeds. If the time delays only apply to one address at a time, then by adding another tester to the bus, it may be possible to double brute-forcing speed.

**Test Case 9: Multiple Testers**

**Step 1:** Connect two testers on the CAN bus and follow the next steps on both of them.

**Step 2:** Enter `ProgrammingSession`.

**Step 3:** Send `SecurityAccess.requestSeed` message.

**Step 4:** Send `SecurityAccess.sendKey` message with random authentication code and verify that it was not accepted.

**Step 5:** Repeat requesting seeds and sending invalid keys until the request is replied with `exceededNumberOfAttempts` or other general error message.

**Step 6:** Keep sending `SecurityAccess.requestSeed` messages. If a seed is received, reply it with a random authentication code and verify that it was not accepted. Keep repeating until 3 time delays have passed.

**Step 7:** **Measure** the total amount of seeds received on the two testers per length of time delay. **Expected result:** the measurement equals the number measured in Test Case 6. I.e. the ECU does not differentiate requests coming from different addresses - it starts the time delay when it has received a total of the number of seeds measured in Test Case 6, and the time delay is effective on requests from all addresses.
4.4.10 Requirement 10: Data Encryption

Any data can be downloaded on the ECU with the TransferData service, but to make any effective use of it, it also needs to be “understood” by the ECU. If the ECU expects encrypted data, then it will always perform the decryption operation on it. To load an effective program on it, in that case, it needs to be encrypted first. That can only be done by a legitimate end node, and thereby no other party can load effective programs on the ECU. If the ECU does not expect encrypted data, then an illegitimate node can also use the TransferData service to load arbitrary software on it.

Also, due to the characteristics of the CAN network, any device connected to a CAN bus can sniff all messages sent on it. If the flashing process is sniffed and the software data would not need decrypting with the secret key, it could be reverse engineered. It is safer to protect proprietary software with encryption.

Test Case 10: Data Encryption

Step 1: Flash an ECU, using Scania flashing software.

Step 2: Log the messages sent. Expected result: Either the SecuredDataTransmission service is used to encrypt the TransferData messages, or the software data itself in TransferData messages is encrypted.
Chapter 5

Identified Vulnerabilities

All test cases described in Chapter 4 were executed on different ECUs with varying strengths of security. The aim was to verify or falsify the presence of ample security mechanisms. This chapter describes the results and their potential impact: each failed test indicates the presence of a potential security vulnerability. The most significant vulnerabilities are chosen to be demonstrated in more elaborate attacks, described in Chapter 6.

5.1 Test Set-Up

The security tests are performed on 7 ECUs, which are chosen to present a variety of different security implementations, and also because their functionality is potentially more interesting to hackers, as per industry opinion. Within one vehicle, there may be ECUs with very strict authentication, authorisation and integrity mechanisms, as well as those with virtually non-existent security. With this research it is intended to both test the most secure ECUs, as well as to point out why security vulnerabilities in the weaker ones are dangerous.

To test an ECU, it is connected to a computer. The computer runs customised software that sends test-case-specific CAN messages to the ECU, and listens for the replies.

5.1.1 The Hardware

The experiments are carried out in a small testbed, which contains the minimal elements for communicating with an ECU. In a vehicle, the ECU is connected directly to the CAN network. In the testbed (see Figure 5.1), the CAN buses are emulated by a CAN-specific breakout box. This device is used to test ECUs outside of trucks [25]. It has three rows of paired connectors, each row electrically
interconnected to emulate one CAN bus. All devices connected to the same row receive all the messages from each other, as on a real CAN bus. The breakout box also has one row of connectors, which provide electricity to the connected devices.

![Figure 5.1: The testbed for testing security requirements.](image)

The computer is connected to the breakout box with a Vehicle Communication Interface (VCI) device. The VCI connects to the computer’s Universal Serial Bus (USB) port, and to the breakout box’s Data Link Connector (DLC). In a vehicle, the DLC is directly wired to one specific bus (referred to as the green bus), and the same applies to the breakout box. Therefore, the computer communicates with the ECUs connected to the breakout box’s green bus. Alternatively, the breakout box may have no DLC, but the VCI is connected directly to one of the three buses with a crossover cable.

5.1.2 The Software

For developing test scripts, Scania internal diagnostics software was used. The software itself utilizes the Kvaser CANlib [29] Software Development Kit (SDK). CANlib is a product which provides and application programming interface for accessing the CAN network. It supports a variety of programming languages and compilers. In these experiments, Microsoft C# [47] is used. CANlib automates all the lower-level communication, and test scripts only need to define the UDS or KWP2000 commands to be sent.

5.2 Test Results

The test results are grouped by tested ECU to give a complete picture of security in one device, except for the ones that bore very similar results. A test is considered passed, if the results matches the expected result described in the respective test case in Section 4.4. Otherwise, the test is considered failed. Due to a confidentiality agreement with Scania, some information is not hereby presented. The names of ECUs are concealed, and represented with single letters instead. The results of test cases that required measurements are omitted.
5.2.1 Results for ECUs A, B and G

The test results for ECUs A, B and G are presented together in Table 5.1, as they vary little. Differing results are presented as separated by a slash, identical results are presented jointly. The results show that all sensitive functions are properly secured with the SecurityAccess service. In fact, the ReadMemoryByAddress function is not even implemented, so it couldn’t be used even if SecurityAccess was bypassed. Therefore, attack scenarios 1 and 2 are ruled out.

<table>
<thead>
<tr>
<th>Test case no.</th>
<th>Question</th>
<th>Answer for ECU A/B/G</th>
<th>Result for ECU A/B/G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Is SecurityAccess required for unlocking?</td>
<td>Yes</td>
<td>Passed</td>
</tr>
<tr>
<td>2.1</td>
<td>Can boot mode be entered in locked state?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>2.2</td>
<td>Can download parameters be initialised in locked state?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>2.3</td>
<td>Can data be transferred in locked state?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>3</td>
<td>Can ECU memory be read in locked state?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>4.1</td>
<td>Is seed predictable?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>4.2</td>
<td>What is the time to query 10000 seeds?</td>
<td>[omitted]</td>
<td>[omitted]</td>
</tr>
<tr>
<td>5</td>
<td>How long is the seed?</td>
<td>8 bytes</td>
<td>Passed</td>
</tr>
<tr>
<td>6.1</td>
<td>Is there time delay after failed SecurityAccess attempts?</td>
<td>Yes</td>
<td>Passed</td>
</tr>
<tr>
<td>6.2</td>
<td>How many attempts are allowed before time delay?</td>
<td>[omitted]</td>
<td>[omitted]</td>
</tr>
<tr>
<td>6.3</td>
<td>How long is the time delay?</td>
<td>[omitted]</td>
<td>[omitted]</td>
</tr>
<tr>
<td>7.1</td>
<td>Is there time delay after ECU reset?</td>
<td>Yes / No / No</td>
<td>Passed / Failed / Failed</td>
</tr>
<tr>
<td>7.2</td>
<td>How long is reset + time delay?</td>
<td>[omitted]</td>
<td>[omitted]</td>
</tr>
<tr>
<td>8</td>
<td>Is time delay affected by changing sessions?</td>
<td>Yes / No / Yes</td>
<td>Failed / Passed / Failed</td>
</tr>
<tr>
<td>9</td>
<td>Is time delay affected by requests from different addresses?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>10</td>
<td>Is software data transferred in encrypted form?</td>
<td>No</td>
<td>Failed</td>
</tr>
</tbody>
</table>

Table 5.1: Test results for ECUs A, B and G.

The software data is transferred in unencrypted form, which means that the data can be sniffed by another device on the CAN bus. The SecurityAccess seed is 8 bytes long in all three ECUs and appears to be random. There is a time delay after a failed SecurityAccess attempt. In ECUs A and G, the time delay is infinite, unless the session is changed to DefaultSession and then back to ProgrammingSession. Once this change is done, the delay is finite. In ECU
### Chapter 5. Identified Vulnerabilities

<table>
<thead>
<tr>
<th>Test case no.</th>
<th>Question</th>
<th>Answer</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Is SecurityAccess required for unlocking?</td>
<td>Yes</td>
<td>Passed</td>
</tr>
<tr>
<td>2.1</td>
<td>Can boot mode be entered in locked state?</td>
<td>Boot mode not available</td>
<td>N/A</td>
</tr>
<tr>
<td>2.2</td>
<td>Can download parameters be initialised in locked state?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>2.3</td>
<td>Can data be transferred in locked state?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>3</td>
<td>Can ECU memory be read in locked state?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>4.1</td>
<td>Is seed predictable?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>4.2</td>
<td>What is the time to query 10000 seeds?</td>
<td>[omitted]</td>
<td>[omitted]</td>
</tr>
<tr>
<td>5</td>
<td>How long is the seed?</td>
<td>2 bytes</td>
<td>Failed</td>
</tr>
<tr>
<td>6.1</td>
<td>Is there time delay after failed SecurityAccess attempts?</td>
<td>Yes</td>
<td>Passed</td>
</tr>
<tr>
<td>6.2</td>
<td>How many attempts are allowed before time delay?</td>
<td>[omitted]</td>
<td>[omitted]</td>
</tr>
<tr>
<td>6.3</td>
<td>How long is the time delay?</td>
<td>[omitted]</td>
<td>[omitted]</td>
</tr>
<tr>
<td>7.1</td>
<td>Is there time delay after ECU reset?</td>
<td>Yes</td>
<td>Passed</td>
</tr>
<tr>
<td>7.2</td>
<td>How long is reset + time delay?</td>
<td>[omitted]</td>
<td>[omitted]</td>
</tr>
<tr>
<td>8</td>
<td>Is time delay affected by changing sessions?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>9</td>
<td>Is time delay affected by requests from different addresses?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>10</td>
<td>Is software data transferred in encrypted form?</td>
<td>No</td>
<td>Failed</td>
</tr>
</tbody>
</table>

Table 5.2: Test results for ECU C.

B, the time delay is always finite. There is also a time delay after resetting ECU A. When resetting ECUs B and G, there is no extra time delay before querying seeds is allowed. This is a minor security bug, since it allows for shortening the time delay. If two VCIs are used to attack any of the ECUs, then the time delay parameters remain the same - only one access attempt is allowed, no matter which address it came from.

The size of the seed pool is $2^{64}$. If an adversary were to attack as per scenario 3, then they would find a match in about a couple of trillion ($10^{12}$) years, taking into account the time delays. It is safe to say that attack scenario 3 would be infeasible to perform on these three ECUs, due to the long seed.

If the attacker wished to focus on one seed only, as per attack scenario 4, they would have to keep querying seeds until the one comes up that they are interested in. Querying seeds with `SecurityAccess.requestSeed` without sending any additional `SecurityAccess.sendKey` messages does not invoke a time delay. Still, going through the seed pool of $2^{64}$ to find a repeated seed once would take about a few billion ($10^9$) years. This attack can hence also be counted as infeasible.
CHAPTER 5. IDENTIFIED VULNERABILITIES

5.2.2 Results for ECU C

The test results for ECU C are presented in table 5.2. In contrast to the previously presented ECUs, the SecurityAccess seed and authentication code are only 2 bytes long, which means there is a very small pool of 65536 seeds. Thereby, both scenario 3 and 4 are feasible to carry out. These attacks are experimentally conducted on ECU C (see results in Section 6.1 and Section 6.3).

5.2.3 Results for ECU D

Test subject D is an example of a completely unsecured ECU. As shown in Table 5.3, the SecurityAccess service is not implemented at all - no authentication is required for performing flashing operations. Any adversary would have free access to flash this ECU with arbitrary software, as per attack scenario 1. As one secure feature, the service for reading memory contents is not implemented in this ECU, therefore attack scenario 2 cannot be followed.

<table>
<thead>
<tr>
<th>Test case no.</th>
<th>Question</th>
<th>Answer</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Is SecurityAccess required for unlocking?</td>
<td>No</td>
<td>Failed</td>
</tr>
<tr>
<td>2.1</td>
<td>Can boot mode be entered in locked state?</td>
<td>Yes</td>
<td>Failed</td>
</tr>
<tr>
<td>2.2</td>
<td>Can download parameters be initialised in locked state?</td>
<td>Yes</td>
<td>Failed</td>
</tr>
<tr>
<td>2.3</td>
<td>Can data be transferred in locked state?</td>
<td>Yes</td>
<td>Failed</td>
</tr>
<tr>
<td>3</td>
<td>Can ECU memory be read in locked state?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>4 - 9</td>
<td>SecurityAccess not implemented</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Is software data transferred in encrypted form?</td>
<td>No</td>
<td>Failed</td>
</tr>
</tbody>
</table>

Table 5.3: Test results for ECU D.

5.2.4 Results for ECU E

ECU E has a long seed and authentication code, as well as properly implemented time delays (see test results in Table 5.4). Interestingly, it was noted during the experiments that when requests were coming from two different addresses, and they were exactly synchronous, then each address received a seed 3 times before
time delay was activated. Therefore 6 seeds were received in total, instead of 3. Although this does not drastically reduce the time of any proposed attack scenarios, this phenomenon is worth mentioning for future reference.

<table>
<thead>
<tr>
<th>Test case no.</th>
<th>Question</th>
<th>Answer</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Is <strong>SecurityAccess</strong> required for unlocking?</td>
<td>Yes</td>
<td>Passed</td>
</tr>
<tr>
<td>2.1</td>
<td>Can boot mode be entered in locked state?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>2.2</td>
<td>Can download parameters be initialised in locked state?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>2.3</td>
<td>Can data be transferred in locked state?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>3</td>
<td>Can ECU memory be read in locked state?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>4.1</td>
<td>Is seed predictable?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>4.2</td>
<td>What is the time to query 10000 seeds?</td>
<td>[omitted]</td>
<td>[omitted]</td>
</tr>
<tr>
<td>5</td>
<td>How long is the seed?</td>
<td>8 bytes</td>
<td>Passed</td>
</tr>
<tr>
<td>6.1</td>
<td>Is there time delay after failed <strong>SecurityAccess</strong> attempts?</td>
<td>Yes</td>
<td>Passed</td>
</tr>
<tr>
<td>6.2</td>
<td>How many attempts are allowed before time delay?</td>
<td>[omitted]</td>
<td>[omitted]</td>
</tr>
<tr>
<td>6.3</td>
<td>How long is the time delay?</td>
<td>[omitted]</td>
<td>[omitted]</td>
</tr>
<tr>
<td>7.1</td>
<td>Is there time delay after ECU reset?</td>
<td>Yes</td>
<td>Passed</td>
</tr>
<tr>
<td>7.2</td>
<td>How long is reset + time delay?</td>
<td>[omitted]</td>
<td>[omitted]</td>
</tr>
<tr>
<td>8</td>
<td>Is time delay affected by changing sessions?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>9</td>
<td>Is time delay affected by requests from different addresses?</td>
<td>Yes</td>
<td>Failed</td>
</tr>
<tr>
<td>10</td>
<td>Is software data transferred in encrypted form?</td>
<td>No</td>
<td>Failed</td>
</tr>
</tbody>
</table>

Table 5.4: Test results for ECU E.

### 5.2.5 Results for ECU F

The test results for ECU F are presented in Table 5.5. In the case of flashing this ECU, unlocking is required only for entering boot mode. If the ECU starts up in boot mode, then all flashing functions can be performed without passing **SecurityAccess** a single time.

The seed is only 2 bytes long, similarly to ECU C, and time delays short enough for brute-force attacks to be feasible as well. Groups of subsequent seeds seemed to occasionally appear in the same order, hinting that the random number algorithm may not output random enough results.

ECU F cannot be restarted without being unlocked first. Therefore, time
CHAPTER 5. IDENTIFIED VULNERABILITIES

<table>
<thead>
<tr>
<th>Test case no.</th>
<th>Question</th>
<th>Answer</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Is SecurityAccess required for unlocking?</td>
<td>Yes</td>
<td>Passed</td>
</tr>
<tr>
<td>2.1</td>
<td>Can boot mode be entered in locked state?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>2.2</td>
<td>Can download parameters be initialised in locked state?</td>
<td>Yes</td>
<td>Failed</td>
</tr>
<tr>
<td>2.3</td>
<td>Can data be transferred in locked state?</td>
<td>Yes</td>
<td>Failed</td>
</tr>
<tr>
<td>3</td>
<td>Can ECU memory be read in locked state?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>4.1</td>
<td>Is seed predictable?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>4.2</td>
<td>What is the time to query 10000 seeds?</td>
<td>[omitted]</td>
<td>[omitted]</td>
</tr>
<tr>
<td>5</td>
<td>How long is the seed?</td>
<td>2 bytes</td>
<td>Failed</td>
</tr>
<tr>
<td>6.1</td>
<td>Is there time delay after failed SecurityAccess attempts?</td>
<td>Yes</td>
<td>Passed</td>
</tr>
<tr>
<td>6.2</td>
<td>How many attempts are allowed before time delay?</td>
<td>[omitted]</td>
<td>[omitted]</td>
</tr>
<tr>
<td>6.3</td>
<td>How long is the time delay?</td>
<td>[omitted]</td>
<td>[omitted]</td>
</tr>
<tr>
<td>7.1</td>
<td>Is there time delay after ECU reset?</td>
<td>Resetting requires unlocking</td>
<td>Passed</td>
</tr>
<tr>
<td>7.2</td>
<td>How long is reset + time delay?</td>
<td>Resetting requires unlocking</td>
<td>N/A</td>
</tr>
<tr>
<td>8</td>
<td>Is time delay affected by changing sessions?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>9</td>
<td>Is time delay affected by requests from different addresses?</td>
<td>No</td>
<td>Passed</td>
</tr>
<tr>
<td>10</td>
<td>Is software data transferred in encrypted form?</td>
<td>No</td>
<td>Failed</td>
</tr>
</tbody>
</table>

Table 5.5: Test results for ECU F.

delay cannot be bypassed by restarting the ECU, when performing a brute-force attack. This is a feature, which might serve as a positive example on its own, although in the current case it doesn’t definitively protect against a brute-force attack.

5.3 Summary

As presented in Section 5.2, the security of ECUs varies widely. ECU D has no authentication mechanisms implemented - an adversary can simply connect to the CAN network of the vehicle and immediately flash the ECU with any software. As there is no security to speak of, then this ECU will not be further
CHAPTER 5. IDENTIFIED VULNERABILITIES

focused on. Rather it will be tested whether attacks can be performed on the security mechanisms that do exist.

ECUs C and F have a very short authentication seed, which potentially makes the ECUs vulnerable to brute-force attacks. Such an attack is experimentally carried out and described in Section 6.1. As there is no message sender authentication and all messages are sent in plaintext, then ECUs C and F are also vulnerable to man-in-the-middle attacks. Combining these two vulnerabilities, attack scenario 4 is also carried out, and the experiment is described in Section 6.3.

ECUs A,B and E have an authentication mechanism that is safe against brute-force attacks described in scenarios 3 and 4, but similarly to all the other ECUs, there is no effective authentication performed on separate messages. Also, none of the ECUs encrypt the software data transferred in the update process. This means that even these stronger ECUs are potentially vulnerable to man-in-the-middle attacks. Several experiments are conducted to demonstrate how such attacks can be performed on ECU A to serve different purposes: flashing the ECU with arbitrary software, sniffing proprietary flash files and sniffing seed-authentication code pairs (see Section 6.2).

Although there are additional tests that failed, it does not mean that the faulty implementation in these places forms a security vulnerability. If the SecurityAccess seed and keys are 8 bytes long, and there is no time delay for unlocking the server after restart (as is with ECUs B and G), it does not open up a door to an attack, as brute-forcing is mitigated with just the seed/key length. The same goes for the fact that failed access time delay is reducible by changing the session in ECUs A and G.
Chapter 6

Experimental Attacks

The test results presented in Chapter 5 uncovered several potentially exploitable vulnerabilities. To validate, whether it is in actuality possible to perform attacks on them, additional experiments were performed. The experiments simulate attacks, which aim to gain unauthorised access to flashing functions.

A brute-force attack was successfully implemented against short authentication seeds. An active man-in-the-middle attack was performed, resulting in flashing the ECU with arbitrary software. A combination of a man-in-the-middle and a brute-force attack against short authentication seeds was carried out to get quick access to the unlocked state of an ECU.

6.1 Brute-Force Attack

The shortest observed seed and authentication code were 2 bytes long. This length makes a brute-force attack feasible, as per scenario 3. The aim of this attack is to get one-time access to ProgrammingSession, using the brute-force approach. The experiment is conducted on ECU C.

Course of the Experiment

The attack consists of querying SecurityAccess seeds and always responding with a pre-chosen authentication code (from hereon referred to as authentication_code_A). It is likely that at some point, authentication_code_A will be accepted as the correct reply to a seed. The purpose of this experiment is to explore the possibility of such an attack, and to measure the timespan until the attack succeeds. To determine some repeatability, this test is conducted twice. Given the time expectations of this experiment, repeating it as many times as would suffice for making statistical conclusions would be too demanding within
CHAPTER 6. EXPERIMENTAL ATTACKS

the time scope of carrying out this thesis research. The course of the experiment is as follows.

• Beforehand, one random 2-byte combination is chosen as the value of *authentication_code_A*.

• *ProgrammingSession* is entered. Time measurement begins.

• *SecurityAccess.requestSeed* message is sent. If a negative response is received (i.e. time delay is on), repeat the message until the ECU sends a positive response (i.e. until the time delay is over).

• If a *SecurityAccess.securitySeed* message is received, a *SecurityAccess.sendKey* message is sent with *authentication_code_A* as its content.

• If the ECU responds negatively to *authentication_code_A*, repeat the process beginning from the *SecurityAccess.requestSeed* message.

• If the ECU accepts *authentication_code_A* and grants access to the unlocked state, stop time measurement.

Results

The experiment was performed twice. On both occasions, a seed appeared, which accepted the random authentication code. The amount of time needed for the attack may be considered feasible, but not short. Theoretically, this attack is realistic to some extent, but it only provides one-time security access (instead of uncovering a secret key, which would provide access for all future occasions). If the details of the authentication code calculation algorithm are known to the attacker, the uncovered legitimate seed-authentication code pair may, however, be subjected to a much more speedy offline brute-force attack to find out the secret key.

6.2 Man-in-the-Middle Attacks

This section describes the man-in-the-middle attacks performed on ECU A. This ECU is an example of the strongest security implementation, yet it has a weakness which enables inserting a “man in the middle” of the communication. This would be a device on the other side of the DLC, non-visible to the person who is updating ECU software in the workshop. The device acts as a relay for all the messages sent between the CAN network and the tester. It receives the messages, analyses them and possibly modifies them, and finally forwards them.

The testbed set-up is illustrated on Figure 6.1, with the name of each device is on the left, and the name of the device it simulates is in quotation marks on
CHAPTER 6. EXPERIMENTAL ATTACKS

the right. A laptop acts as the workshop computer. It is connected to a breakout box, which simulates the DLC - the external physical interface of the vehicle’s CAN. This, in turn is connected to a PC, which acts as the attacker’s device. This is connected to another breakout box, simulating the coordinator for the CAN network. An ECU is connected to the breakout box. In combination, these devices mimick a real vehicle.

Figure 6.1: The testbed set-up for the man-in-the-middle attacks.

6.2.1 Passive Attack: Recording Software Files

One way of carrying out a man-in-the-middle attack is for the attacker to be a passive observer and just sniff the messages sent over the communication channel. By doing that, the attacker is able to gain knowledge of the ECU-specific flashing protocol, and additionally record the software data transferred from the tester to the ECU. This data could be used for reverse engineering or extracting secret keys. The software could also just be stored for future use, e.g. for loading it on an ECU on a different vehicle, which is not supposed to have some specific functionality that this software provides.

Note, that to perform this attack, the attacking device does not need to be a “man in the middle”. Due to the characteristics of CAN, it can be any node on the bus.
CHAPTER 6. EXPERIMENTAL ATTACKS

Course of the Experiment

The course of the experiment is as follows.

- The attack device starts listening on both channels - the tester side and vehicle CAN side.

- Each message received on one channel is forwarded out the other channel (this includes the SecurityAccess challenge and response).

- Before the attack software forwards the message, it reads it and detects the service being called.

- It records the contents of every TransferData message into a file. Each of these messages contains a part of software data.

Results

This experiment resulted in the successful recording of the software binary file downloaded on ECU A. As a part of this experiment, the binary file was scanned to see if it contains the secret keys used to calculate the SecurityAccess authentication code. However, they were not found in plaintext. Further comprehensive reverse engineering was not performed due to time constraints.

6.2.2 Active Attack: Replacing ECU Software

A man-in-the-middle attack can also be carried out actively. This means that the attacker alters or replaces some messages in the communication flow. The attack performed in this experiment aims to update the ECU with the software provided by the attacker. This is done by replacing the messages, which carry software data (that is, messages of the TransferData service). Every specific amount of TransferData blocks is followed by a RequestTransferExit message, which requests for a checksum of the sent data blocks. If the intention of the attacker is not to alarm the tester of their presence, then the reply to this request cannot be forwarded from the ECU - the software data sent by the attacker does not add up to the same checksum as the data sent by the tester. For this reason, the attack software must calculate the checksums for the tester simultaneously with sending data to the ECU. In a normal software update process, the tester restarts the ECU at the end of data transfer and writes additional informational metadata to the ECU. This may include the fingerprint (some identifier value) of the programming tool, which reprogrammed the ECU software. Then, the ECU is restarted again and the same metadata is queried. The lack of a fingerprint may raise suspicion in whoever might check upon the ECU at some later stage, or cause the workshop...
tool to give an error notification. It is beneficial to start forwarding messages between the tester and the ECU again after data transfer is done, so that a legitimate fingerprint would be added.

For simplicity, this experiment uses partial code from a Scania proprietary software tool to perform the data transfer between the attacker and the ECU. The code snippet does not perform any security functions, but it does take care of communicating the data transfer parameters with the ECU, dividing the software file into data blocks of appropriate size and format, and using correct messages as per the Scania internal diagnostics protocol. In theory, such code could be developed, using CANlib [29], by an attacker who has been able to sniff on a flashing session (as described in the previous experiment) and reverse engineer the Scania diagnostics protocol, or has otherwise detailed knowledge of it.

**Course of the Experiment**

The course of the experiment is as follows.

- The attack device starts listening on both channels - the tester side and vehicle CAN side.

- Each message received on one channel is forwarded out the other channel (this includes the SecurityAccess challenge and response).

- Before the attack software forwards the message, it reads it and detects the service being called.

- When the first RequestDownload [42] [44] message is received (this happens after the ECU has gone to boot mode and all instances of SecurityAccess have been passed), the attacker stops forwarding messages. Instead, the attack software starts executing in two separate threads: one for responding to the tester, and one for flashing the ECU with arbitrary software. Once both threads have come to the point where all data is transferred and the ECUReset service is called to restart the ECU, they finish execution and message forwarding is resumed.

- The attack program starts its first thread of execution, which responds to all messages from the tester with a positive response message.
  - For each TransferData block, the checksum is calculated, and added to the previously stored checksum.
  - Each RequestTransferExit is responded with the currently stored checksum. Then the stored checksum is nullified, so as to begin calculating a new sum for the next blocks.
All other messages are responded with the defined positive response message [42] [44].

If the ECUReset service is called, the thread responds with a positive response message and finishes. The program resumes to forwarding messages between the tester and the ECU.

• At the same time, the attack program starts its second thread execution, which simultaneously flashes the ECU with software provided by the attacker. This is done by code loaned from Scania proprietary flashing software. Note that none of the following actions require additional SecurityAccess challenges, because the ECU has already entered an unlocked state.

• Respective memory regions are erased [42].

• For each set of data blocks, download parameters are initialised (RequestDownload service [42] [44]), data is transferred (TransferData service [42] [44]) and data is programmed into memory (RequestTransferExit service [42] [44]).

• Once all data is transferred, the ECUReset service is called to reset the ECU. The thread finishes and the program waits for the first thread to finish as well, before resuming to forwarding messages between the tester and the ECU.

• All following messages are forwarded between the tester and the ECU. This will include writing metadata to the ECU, and checking it.

Results

Within this thesis an attack program was developed, which follows these steps and successfully manages to flash ECU A with arbitrary software. The workshop flashing software finishes its procedures without error and the ECU appears to be running the software loaded on it by the workshop tool.

The attack program was tested in every step to ensure correct behaviour, and thereby it was also confirmed that the software that was programmed into the ECU could not have sourced from the workshop tool, and was indeed the software transferred to it by the attack program. Afterwards the ECU was moderately tested and the software loaded in the course of this attack was confirmed to run without obvious errors.
6.3 Combined Attack

The test results for ECUs C and F (presented in Section 5.2) showed that the size of their seed pool is only $2^{16}$. The time to come across any one specific seed, when querying them without sending replies that would invoke a time delay, is thereby fairly short. If one was able to sniff on a SecurityAccess message exchange, using a passive man-in-the-middle attack (as described in Section 6.2.1), then they would have a valid seed-authentication code pair. It would not take the attacker a long time to send SecurityAccess.requestSeed messages until they receive a seed equal to the one sniffed. That seed can then be replied with the sniffed authentication code, which would grant access to the unlocked state of the ECU, and thereby all flashing functions. This hypothesis is tested experimentally on ECU F.

Note again, that to sniff on software update messages, the attacking device does not need to be a “man in the middle”, but can be any node connected to the CAN bus, due to the characteristics of CAN.

Course of the Experiment

The course of the experiment is as follows.

- A passive man-in-the-middle attack (similarly as described in Section 6.2.1) is carried out to sniff on a flashing session. Instead of the TransferData service, the messages of the SecurityAccess service are logged.
- Later, a valid seed (from hereon referred to as seed_A) and a corresponding authentication code (authentication_code_A) are extracted from the logs.
- ProgrammingSession is entered. Time measurement begins.
- The brute-force method is used: SecurityAccess.requestSeed messages are sent to the ECU repeatedly, until the ECU sends a SecurityAccess.securitySeed in response, which contains a seed equal to seed_A. (Note that no SecurityAccess.sendKey messages are sent to the ECU, therefore no time delay is activated.)
- A SecurityAccess.sendKey message is sent to the ECU, containing a authentication code equal to authentication_code_A.
- If the server responds with a positive response message, then time measurement is stopped.

Results

The experiment was successfully run a total of 5 times with different seed and authentication code pairs, each taking a short amount of time. In an attack
situation, the \texttt{SecurityAccess} challenge only needs to be passed once or twice, therefore this attack is feasible. When performing this attack on an ECU with an 8-byte security seed and key, such as ECU A, then gaining access would take several billion years.

In the workshop, one flashing session likely includes calling the \texttt{SecurityAccess} service more than once. Therefore, more than one seed and authentication code pair can be uncovered over the course of one passive man-in-the-middle attack. This would reduce the time required get past one \texttt{SecurityAccess} challenge, as there would be more seed-key pairs stored for use over the same seed pool.

\section*{6.4 Impact of Vulnerabilities}

As demonstrated by the experiments described in this chapter, some ECUs have realistically exploitable vulnerabilities, which may allow for an adversary to alter their software.

Standards of ECU diagnostic communication (KWP2000 \footcite{44} and UDS \footcite{42}) define security measures to protect the flashing functions from being executed by unauthorised parties. However, their implementation is up to the manufacturer. In some ECUs, no security measures are implemented at all, leaving the ECU vulnerable to any software alterations by whoever has physical access to the vehicle.

Some ECUs do have security measures in place, but their implementation is faulty, and they can be bypassed in certain situations. For example, testing determined that in one ECU make no authentication is required for flashing when the ECU starts up in boot mode. Therefore, if for some reason the ECU would be left without a functional application flashed on it, then the next time it starts up, there are no restrictions to who can flash it.

If security measures are indeed effectively implemented on ECUs, they also need to be sufficiently strong. The seed, secret key and authentication code need to be long enough to mitigate both online and offline brute-force attacks. Additional protections needs to be in place in the form of time delays after a certain number of false security access attempts, as well as after restarting the ECU. If these numbers (seed/key length, number of allowed failed access attempts, length of delay) form a weak combination, then it is possible to gain access to sensitive ECU services in a reasonable amount of time simply by trying all the different seeds or keys.

Diagnostics messages sent on the CAN bus do not include any sender authentication, and thereby replay and man-in-the-middle attacks are possible. This means that a device is placed between the workshop computer and the vehicle’s internal CAN network, enabling it to relay, alter, block and insert messages
in the ongoing communication. The experimental attacks showed that such a
device can successfully and unnoticably flash even a relatively secure ECU with
arbitrary software. It can also sniff on the communication to collect sensitive
data. The data can be later used for reverse engineering (which may include
extracting secret keys) or a faster brute-force attack.

It can be imagined that the attacking device is remotely controlled. Using
wireless transmission, it could send sniffed data to the person who implanted
it. Alternatively, the implanter could use wireless connection to send additional
instructions to the device.

Such attacks can be avoided by implementing additional or stronger security
measures. The next chapter will introduce some suggestions for improving ECU
software update security, based on common network and information security
standards.
Chapter 7

Solutions

7.1 Proposed Solutions

Most vulnerabilities and attacks described previously can easily be avoided by implementing security measures listed in standards [42] [44], using strong enough variables. The more complex attacks, however, may be avoided only by making additions to the current implementation. The memory capacity of an ECU is extremely limited, and hence the suggested solutions are kept as minimalistic as possible.

7.1.1 Sufficiently Strong Implementation of Security Standards

Locked and unlocked state. It is essential that there is a distinction between a locked and an unlocked state in the ECU software. Flashing-related operations should only be available in an unlocked state, which can be entered only after successfully passing the SecurityAccess challenge. The ECU needs to start up in a locked state both when starting up in app and boot mode.

Long seeds and keys. The SecurityAccess challenge itself needs to be secured with a long, unrepeating seed. As testing showed, an 8-byte seed and key proved sufficiently strong. This seed/key length makes gaining access with an online brute-force method infeasible already by itself, as it can hardly be performed any faster than the ECU’s capability to send replies to SecurityAccess.requestSeed (and SecurityAccess.sendKey) messages. The resilience towards an offline brute-force attack depends on the length of the secret key, which can be different from the length of the seed.

Time delays. If the ECU cannot handle performing cryptographic computations with long seeds and keys, then online brute-forcing as per scenario 55.
3 can always be prolonged by increasing the time delay after a failed security access attempt. In the case of scenario 4, however, no time delays are implemented, drastically reducing the time required to gain access. This behaviour is according to standards, which suggest only implementing a time delay when one or several access attempts have failed. In a legitimate situation, the tester does not need to query more than one SecurityAccess seed, without replying with a correct authentication code in between. Therefore it is hereby suggested to implement the time delay right after a seed is queried.

7.1.2 Public-Key Authentication

A lightweight authentication mechanism has been proposed [7], based on the RSA (Rivest-Shamir-Adleman) encryption algorithm [39]. The proposition suggests storing the vehicle manufacturer’s public key on the ECU [7]. This way, any secret keys would not be stored in the ECU memory nor in its program code. Therefore, secret keys would not be vulnerable when software update data gets exposed during transfer.

Authentication would be performed in the place of the current SecurityAccess authentication service - before executing flashing operations. Further data encryption is not suggested [7].

The software module of the authentication system is optimised by using processor-specific instructions, to be as efficient as possible. Verifying authentication data on the ARM Cortex M3 processor, which is widely used in embedded applications, takes an average of 9.66 milliseconds, using a 32-byte key. The size of the program itself is 8 kilobytes [7].

7.1.3 Software Data Encryption

Using the UDS SecuredDataTransmission service [42] for software data transmission can avoid man-in-the-middle attacks. Encrypting all data would ensure both confidentiality and integrity - without knowing the decryption key, the adversary would not be able to either understand the flash data nor liberally change it on-line.

The strength of the SecuredDataTransmission service would lie in its security sub-layer, and namely in the encryption methods used. The UDS and KWP2000 standards do not define what algorithm to use, or how to agree upon the encryption key. Encrypting all software update data with the shared secret key would mean that the key gets a lot of exposure when the transmission is sniffed. This means that a cryptanalyst would have a great amount of material to attempt to uncover the secret key. Therefore it should be encrypted with a one-time shared session key instead.
7.1.4 Cryptographic Signatures on Messages

Encrypting all the transmitted software data may produce too much overhead, therefore less computation intensive solutions are also considered.

The man-in-the-middle attack performed in this thesis is possible, because the sender of messages is not verified. To prove that the message comes from a legitimate source, the tester should add a signature to the message. To compose a digital signature, the message is passed through a hash function, and the resulting hash is encrypted [1].

Another problem, which should be addressed, is the possibility that TransferData messages are sniffed and replayed at some other instance to install the same software on a different ECU. Therefore, message freshness should also be verified by adding some data to the hash before encrypting it, which is associated with the current time. Normally, such data could be a timestamp [1], but since an ECU does not have an internal clock, it has no way of verifying a timestamp. Alternatively, this data could be a nonce - a number used only once [1]. Using a true nonce would mean that the ECU would have to store all previously used numbers, which would probably be too resource-demanding. Instead, a random number could be used, as it is unlikely to ever repeat, given that it is long enough.

The assumed adversary - the tester - should not have any role in initialising the nonce, because otherwise they could use the sniffed nonces and replay the message signatures as well. Therefore, it should be the ECU deciding its value before every TransferData message (possibly as a part of the RequestDataTransfer message, or the reply to the previous TransferData message).

The choice of a suitable secure hash function and nonce size should be left up to the vehicle manufacturer.

7.1.5 Session Key Exchange

Since the signature is added to every TransferData message, then the key used for encrypting it gets a lot of exposure. This would be undesirable for the same secret key, which is used in the SecurityAccess challenge. Therefore, a disposable session key should be agreed upon. The session key agreement protocol should be resistant to the attacks described earlier.

Using Asymmetric Cryptography

The common simple solution for exchanging a session key is the Diffie-Hellman (DH) protocol [37]. It is secure against man-in-the-middle attacks only if it uses...
public-key cryptography to authenticate both parties of the exchange [37] [1]. As suggested in Section 7.1.2, using the asymmetric RSA encryption algorithm may be feasible.

Mutually authenticated DH would require for the ECU to authenticate itself, as well. This means that the proposed RSA solution [7] would need to be extended. The ECU would need to store its own secret key, and perform additional cryptographic operations to authenticate itself to the tester. This is undesirable, as it would probably double the amount of computations needed, in comparison to the original proposition.

In the context of the man-in-the-middle attack presented in this thesis it is only absolutely necessary to authenticate the tester. Therefore, for agreeing on the session key, DH could be used, with the addition of the tester’s cryptographic signatures on all DH messages.

Using Symmetric Cryptography

Developers may still find public-key cryptography too cumbersome to be used in the resource-limited ECUs. To enable considering a simpler and even more lightweight option, hereby an alternative solution is proposed.

To make key agreement as simple and light on the memory as possible, the same mechanism may be used, as for SecurityAccess, with minor modifications (see Figure 7.1):

1. The ECU sends a random number $r$ to the tester.
2. Both the ECU and the tester encrypt $r$ with the shared key.
3. Instead of sending the encrypted $r$ back to the ECU, as in the SecurityAccess challenge, it is used as the session key.

7.2 Evaluation of Solutions

Previous research shows that insufficient implementation of security standards is the main source of vulnerabilities in the diagnostic interface of an ECU (see Section 2.6). Using an authentication mechanism with a long seed and an appropriate time delay after an invalid access attempt is the essential step towards securing an ECU. However, even the ECUs that employ these mechanisms properly, are vulnerable to man-in-the-middle attacks.

The strongest security methods for data transmission can hardly be considered for ECUs, as the devices are very limited in terms of memory, storage and computational capabilities. Decrypting all the software data sent to the ECU
Figure 7.1: Session key agreement, using symmetric cryptography.

is a very costly operation, and SecuredDataTransmission is therefore not the most optimal solution.

The public-key solution based on the RSA algorithm, which has been proposed in previous solutions, only authenticates the tester. It uses optimised program code, and is claimed to be lightweight enough to be considered as a realistic solution. Although the view of developers is that public-key cryptography is too computation-intensive for ECUs, it may be worthwhile to implement the proposed solutions for performance testing purposes.

Adding cryptographic signatures on messages is a step towards message authentication, which can defer an active man-in-the-middle attack. However, it does not protect the software data from being stored by the means of a passive man-in-the-middle attack, and later reverse engineered.

The proposed session key exchange protocols do not perform mutual authentication. They only authenticate the tester to the ECU and not vice versa. In the proposed attack models it is always the ECU that is the attack target, because eventually the altered software needs to end up programmed on it. Therefore, authenticating the ECU to the tester is not as important, and it can be trusted. This opens up the possibility to give full responsibility for choosing the session key seed to the ECU. If the ECU is the only one deciding the nonce for the session key, then an adversary cannot use masquerade or replay attacks to present itself as a legitimate node.
Chapter 8

Conclusions

Modern vehicles are controlled by a distributed computer system of embedded devices - Electronic Control Units (ECUs). Customising the functionality of a vehicle comes down to changing the software on ECUs. To avoid problems with road safety, as well as legal issues, it is desirable that only the vehicle manufacturer is able to make software alterations to ECUs controlling critical modules. This goal requires for appropriate security measures to be in place, so that external parties would be unable to get access to an ECU’s self-reprogramming functionality. Previous research has already identified several potentially exploitable vulnerabilities in the diagnostic interfaces of ECUs used in automobiles. Also, general-purpose attacks have been demonstrated, mostly related to sending immediately executable commands to ECUs.

This thesis continued the research by moving the focus to ECUs used in heavy-duty vehicles, and specifically the reprogramming functionality of the ECUs’ diagnostic interfaces. The purpose was to evaluate the process of performing software updates on different Scania ECUs from the security perspective. As a result, it reports security vulnerabilities, which may lead to an unauthorised person flashing ECUs with arbitrary software. To thoroughly present vulnerabilities and their impact, the thesis has three consecutive goals: to identify vulnerabilities, demonstrate attacks, and propose solutions. The research was carried out as a quantitative study, using experimental research methods and a deductive approach. Experimental methodology was used to test ECUs and calculations were performed to arrive at conclusions. To evaluate the severity of the vulnerabilities found, the results were validated by performing experimental attacks.

To start off the security evaluation and meet the first goal - identifying vulnerabilities - formal software testing methods were used. The abstraction level tested was the integration of the ECU and a diagnostic node. The subject of
testing was the application-level diagnostic interface of each chosen ECU. The vulnerabilities pointed out in previous research, as well as the security requirements defined in the UDS and KWP2000 standards documents, were built upon to compile a set of test cases. Each test aimed to verify the correct or sufficient implementation of a functional security requirement. Positive tests were executed to verify the presence of standardised security requirements, and negative testing was added to identify any security bugs. As the specification documents and software code used internally in Scania was available for this study, then grey-box testing methods were applied to save time. To make accurate measurements of time delays and query speeds, some tests were automated with scripts.

8.1 Identified Vulnerabilities

The results of the security evaluation reveal several exploitable vulnerabilities in ECUs with varying security strength. The main identified problems were the following:

- No security implementation whatsoever. These systems were not chosen to perform further attacks on.
- Short seed and authentication code used in authentication. It is feasible to perform a brute-force attack on an ECU with a seed and authentication code length of merely 2 bytes, and an average-length failed access attempt delay.
- No authentication or encryption of messages carrying flashing data. Since CAN is a message-based protocol, then lack of higher-level authentication and sending plaintext messages makes it vulnerable to man-in-the-middle attacks.

The found vulnerabilities are in line with previous research, which emphasizes similar or related problems in ECUs used in automobiles. To determine and demonstrate the significance of these vulnerabilities, they were further experimented with.

8.2 Performed Attacks

Experimental attacks were devised to meet the second goal of the thesis. The aim of this was to validate whether the identified vulnerabilities could realistically be exploited to flash an ECU with arbitrary software. The goal of the attacks was to gain unauthorised access to an ECU’s flashing functionality.
CHAPTER 8. CONCLUSIONS

Three different types of attacks were successfully conducted, implying that the identified vulnerabilities pose a realistic threat.

8.2.1 Brute-Force Attack

Previous research suggests that short seeds and keys might only provide a temporary protection from malicious security access attempts. As in the course of testing an ECU was found, which featured a 2-byte seed and authentication code, the realistic feasibility of a brute-force attack could be verified.

A random 2-byte combination was chosen and repeatedly used as the authentication code. New seeds were queried, until the authentication code was accepted, and a higher security level was unlocked. This experiment was conducted twice, and succeeded in a feasible amount of time on both occasions.

Gaining access to a higher security level makes it possible to successfully send flashing commands to an ECU. However, flashing itself was not performed as a part of this experiment - bypassing authentication and unlocking the ECU was the sole goal.

8.2.2 Man-in-the-Middle Attack

Previous research also points out that the message-based nature of the CAN network, and the lack of encryption in ECU diagnostic interface implementations makes it susceptible to message replaying. It was the intention of this research to verify whether it could be exploited to perform a man-in-the-middle attack.

This experimental attack was performed on an ECU, which in the course of testing proved to be one of the most secure ones. However, flashing messages are sent without separate authentication of each message, and software data is unencrypted. This means that another device can be placed between the communication line of the tester and the ECU, to relay messages between them, and change or drop them at will.

The testbed consisted of 2 computers - one of them acting as the tester, and one of them the adversary - and an ECU. The adversary managed to successfully relay messages to the ECU until authorisation was successfully finished. Then, it dropped the following messages from the tester, and replied to them with positive response messages and calculated checksums. At the same time, it sent messages containing arbitrary software data to the ECU. Once the tester and the adversary had finished sending flashing data, the adversary continued relaying messages from the tester to the ECU, in order to finalise the flashing process, and not arise immediate suspicion. As a result, the legitimate workshop flashing tool finished its task without giving error messages, and the ECU was actually flashed with arbitrary software provided by the adversary.
8.2.3 Combined Attack

The previous attack is defined as an active man-in-the-middle attack, since the adversary intercepts messages. However, the intention of the adversary may simply be to sniff on the communication between the tester and the ECU. This way they could find out a legitimate seed-authentication code pair, and use it in a sped-up brute-force attack to unlock a higher security level. The combined attack is suggested as a part of this research.

The same test set-up was used as in the previous attack. The ECU used was the same as in the first brute-force attack. This time, message relaying was not stopped. Instead, when the tester and ECU performed a SecurityAccess challenge, the seed and authentication code were sniffed and stored. Later, the ECU was queried for seeds without sending any response, until the sniffed seed appeared. Then it was replied to with the sniffed authentication code. Since no false authentication codes were sent, then time delay was never activated. The higher security level was successfully unlocked with substantially reduced time in comparison to the previous brute-force attack.

8.3 Proposed Solutions

As the identified vulnerabilities proved to be exploitable, solutions were also proposed to avoid such attacks on ECUs in the future.

First, it is essential that ECU software implements the security functionality suggested in the UDS and KWP2000 standards. Flashing functions should only be available after unlocking a higher level of security via a successful completion of the SecurityAccess challenge. The seed and key used in the challenge should be 8 bytes long to mitigate brute-force attacks. In case of a shorter key, a time delay should be implemented after a failed security access attempt, before a new seed can be queried, as well as after every boot. To prolong the time required to performed the combined attack, a time delay should be activated right after a SecurityAccess seed is queried.

Secondly, the ECU software update process would benefit from using the SecuredDataTransmission service for sending software data to the ECU in an encrypted form. This would protect the proprietary software of the vehicle manufacturer from being reverse engineered. Also, it would be impossible to reuse the software data to illegitimately flash another ECU. Additionally, the software data messages could not be altered by a “man in the middle”.

Since decryption on a large scale might be too labour-intensive for an ECU, an alternative protection method against an active man-in-the-middle attack would be adding a signature on each message containing software data, con-
firming that it came from a legitimate source. The signature would consist of a message hash and a nonce, encrypted with a session key. The session key would be agreed upon with the Diffie-Hellman algorithm, where the tester is authenticated with public-key cryptography. Alternatively, if symmetric cryptography is preferred, the session key could be a random number chosen by the ECU, encrypted the shared secret key used for SecurityAccess.

8.4 Constraints

The research only focused on one part of the ECU software update process - the security implementations in the ECU’s self-reprogramming interface. In that component of the whole process several vulnerabilities were found and demonstrated. It may very well be that additional security holes lie in the other elements of the process, such as the storage and distribution of cryptographic keys or production and signing of flash files. Additionally, no offline brute-force attacks were performed on the cryptographic algorithms to uncover secret authentication codes, although they may have been feasible. Therefore, this research does not present a complete picture, regarding the security of the ECU software update process.

The experiment performed in the course of this thesis relied greatly on the internal specification documents and proprietary software code of Scania. The documentation was used to build an understanding on the Scania-specific details of the application-level communication protocols used in updating the software of different ECUs. These details could theoretically be reverse engineered by an adversary, who has sniffed on a flashing procedure, therefore this should not completely nullify the reproducibility of the research. Scania’s proprietary software code was used in experiments to take care of establishing a connection with the ECU, as well as for performing the flashing itself, after authentication was bypassed in the course of the man-in-the-middle attack. As Scania software is built on the publicly available Kvaser CANlib library [29], then, given sufficient time and knowledge about the flashing protocol, these scripts could also be reproduced without access to proprietary software.

The brute-force attack and combined attack did not follow up with software updating after authentication was bypassed. Thereby, the attacks are slightly incomplete, although the potential threat has been demonstrated.

All tests and experimental attacks were performed in a testbed, not on a real vehicle. ECUs were treated as standalone devices, not as a part of a complete system. Adding a “man in the middle” to a truck may mean inconspicuously inserting a pre-programmed device in its internal electrical system. Also, additional restrictions may apply to perform these attacks or make arbitrary software
CHAPTER 8. CONCLUSIONS

The results of Test case 4.1 were only concerned with randomness as perceived by a human observer - that seeds did not have an obvious pattern. Given the context, this may have been a sufficient criterion, but it would have been more comprehensive to analyse the pseudo-random number generation algorithm used, to see whether its results can be predicted.

8.5 Evaluation of the Research

In this section, the quality of the thesis is evaluated, based on criteria listed in Section 3.3.

8.5.1 Validity

Several test cases, which were based on select security requirements, failed in the phase of identifying vulnerabilities. Those security holes were built upon to perform full attacks. Three attacks were successfully planned and carried out, implying that the choice of security requirements and execution of tests was valid.

The perceived validity of the thesis may be affected by some numerical results being left out of the report, due to the confidentiality agreement.

8.5.2 Reliability

In both the brute-force attack, as well as the combined attack, there is a high random component, in the form of the random authentication seed. Therefore, the attacks needed to be carried out several times over, in order to ensure the reliability of time measurements, as well as success.

The brute-force attack was carried out only twice. Although it is a feasible attack, it could not be carried out fast enough to be repeated many times over. Therefore, a scientifically reliable evaluation could not be compiled, to give an overview of the average time required to perform this attack. Also, it cannot be stated that this attack will always work, or that there are no additional hurdles that may hinder its success.

The combined attack was successfully carried out five times. This is enough to ensure that it is a possible attack, and to calculate a loose average of the time required.
8.5.3 Replicability

This report presents a fairly detailed description of the tested security requirements, test cases, and attack plans. Based on this documentation, it should be possible to repeat the course of the experiments.

However, due to a confidentiality agreement, many measurements regarding the time required to perform attacks could not be published. Therefore, based on the presented data, it is difficult to evaluate, whether an independently repeated experiment arrives at the same results. Also, it may prove difficult to use non-numerical results in a future public research, to decide upon the feasibility of an attack.

8.5.4 Ethics

This research was carried out, keeping ethical hacking principles in mind. Scania was previously aware of security testing and experimental attacks being carried out on ECUs used in their organisation. The found security vulnerabilities were not exploited to cause deliberate and unwanted malfunctioning of devices, or any physical harm or financial loss. All found vulnerabilities were reported to Scania, before publishing them in this thesis report. Additionally, all information, which could be treated as company secrets, was removed from the public report, as requested by Scania.

This thesis demonstrated that ECUs are vulnerable to several attacks, which open up the possibility to flash them with arbitrary software. A tech-savvy truck owner could perform them to add desirable functionality to their vehicle. A dishonest independent workshop could easily make a business out of changing the functionality of ECUs, or making upgrades without paying commission to the manufacturer. In extreme cases, hackers could alter the software of critical ECUs to cause physical harm to the passengers in the vehicle or on the road, or financial loss. With regard to these matters, this thesis has ethically positive intentions.

However, the thesis suggests implementing features, which block the ECUs to software updates from any other sources than the official manufacturer. This restricts consumer freedom to do as they please with the hardware equipment they have purchased. Today, the topic of intellectual freedom and ownership in technology is becoming very poignant, therefore both sides of this equation need to be considered.
8.5.5 Sustainability

In suggesting solutions to problems, it has been kept in mind that an ECU’s computational capabilities, as well as permanent and temporary storage capacity are very modest. However, some suggestions (e.g. encrypting and decrypting all transmitted software data) may still prove extensively cumbersome to perform. Also, the total investment required to implement improved security mechanisms may prove too much for an organisation, if the threats are not evaluated to pose a big risk. Therefore, the more computation-intensive suggestions do not seem very promising with respect to sustainability.

However, more lightweight solutions were also proposed - sufficient implementation of security standards, adding signatures, and a simple session key exchange protocol. These are less costly and more easily implemented, not requiring a great deal of rework in the whole software update process.

8.6 Discussion

As a starting point for the research, formal software testing methods were used to identify vulnerabilities. Test cases were compiled, based on the security recommendations in related industry standards, and positive tests were executed based on them. However, positive testing can only cover the functionality that is supposed to be present. Unexpected security vulnerabilities appear in the unlimited space of possibilities outside the requirements specifications, which is explored by means of negative testing. Exhaustive testing of that space is impossible, and time restrictions limit the amount of negative test cases that sheer creativity can produce. To maximise test coverage, additional test cases were compiled, inspired by previous research - both theoretical and practical - on the topic of ECU security.

Multiple attacks were successfully performed against ECUs within this thesis, based on the vulnerabilities identified in testing. As a minor improvement, the brute-force attacks could have been tested in full scale. This would include gaining access and immediately flashing the ECU with arbitrary software. It may prove necessary to perform some additional functions after flashing, to make the software functional afterwards.

All of the attacks rely on physical access to the vehicle. Furthermore, the man-in-the-middle attacks would require a device to be inconspicuously inserted into the vehicle’s internal electrical network, which would perform the sniffing, fooling and flashing operations. An attacker would have to go through a lot of trouble to achieve such access.

In the context of this research, the types of adversaries exist, who do in-
deed have physical access to the vehicle. This includes the vehicle owner and independent workshops. They can take as much time as they would like to perform a brute-force attack. Also, they would not have to worry about hiding the man-in-the-middle device, so they can use a regular computer for that purpose. Also, they would not need to “fool” the workshop tool into thinking that the flashing succeeded, and can break the communication right after bypassing authentication. Therefore, it can be argued that the described attack scenarios are likely and realistic enough to be considered.

Coming from a different perspective, is it really necessary to forbid the owner of a truck from modifying the functionality of their vehicle? In some cases it poses a risk to road safety. If the programming of an engine or gearbox management system is modified by an external individual, then the changes are not properly tested, as is done over the official development process. The alterations may carry software bugs, which cause the vehicle to act abnormally, and pose a danger to its passengers and other travellers on the road. Furthermore, they may cause the engine to perform in a way that does not comply with the environmental requirements regulated by the law. However, some ECUs, such as the audio management system, do not have such critical functions at all. They could be left open for modifications - given that there is a way to determine, whether the current software on the ECU is the original or not, so as to avoid false warranty claims. Moreover, since any extra functionalities are very costly on ECUs, it would be financially straining to protect all of them.

8.7 Future Work

In future research, the other parts of the ECU software update process could also be evaluated in terms of security. This includes the management and distribution of secret keys, as well as the production, signing and distribution of flash files. These topics include a wider subject area, spanning across multiple computer systems in the vehicle manufacturing organisation. However, that would most likely be a Scania internal research, rather than a public study.

An important security aspect that went completely undiscussed in this project is the algorithm for generating random numbers in ECUs. What was verified in Test case 4.1, was that the seed is not static or that their pattern is not obvious to the eye. However, white-box testing the algorithm would reveal, whether the input used for calculating the seed is predictable. A predictable input also makes the seed predictable, given that the adversary knows the seed generation algorithm.

To comprehensively validate these attacks, they should be tested on a real vehicle. Recreating these attacks on a complete system may uncover restrictions,
which went unnoticed in this research.
Bibliography


[41] Scania internal documentation.


