Automatically Detecting Cryptographic Algorithms in Firmware Binaries

Alexandru Blinda
1581951
Supervised by Dr.-Ing. David Oswald

University of Birmingham
School of Computer Science
April 9, 2017

This dissertation is submitted for the degree of *BSc Computer Science*

Abstract

Over the past decades, advances in all types of technology have touched every aspect of our lives. The automotive industry has been rapidly evolving, continuously shifting from mechanical and primitive electric units to cutting-edge electronically controlled units. The modernisation of cars provides numerous benefits such as comfort and reliability - however, this occurs at the expense of more complex connections between its internal components and various external devices. As more data transfers are needed between different components of a car, its confidentiality and integrity must be protected using various cryptographic algorithms.

This paper provides an analysis of automotive firmware binaries conducted on data collected from topic-related forums. It aims to serve as a clear overview of what cryptographic algorithms are used to protect all manipulations of car data. The project consists of a data collection script and the analysis of the gathered data.

Keywords: Firmware Binaries, Cryptographic Algorithms, Automatically Detecting, Automotive Security, Web Scraping

All code and reports associated with this project are available at
`git-teaching.cs.bham.ac.uk/mod-ug-proj-2017/axb1080`

Acknowledgements

I would like to thank Dr.-Ing David Oswald for his interesting final year project idea and his invaluable support throughout the development of this project.

I would also like to thank my amazing girlfriend who spent countless hours proofreading this report and for her moral support that kept me going.

Contents

List of Figures

Glossary

AES Advanced Encryption Standard.

API Application Programming Interface.

BAT Binary Analysis Tool.

CAPTCHA Completely Automated Public Turing test to tell Computers and Humans Apart.

CASE Computer-Aided Software Engineering.

CRC Cyclic Redundancy Check.

CSS Cascading Style Sheets.

DEAL Data Encryption Algorithm with Larger blocks.

DER Distinguished Encoding Rules.

DES Data Encryption Standard.

DOM Document Object Model.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

IDE Integrated Development Environment.

MD4 Message-Digest Algorithm 4.

MD5 Message-Digest Algorithm 5.

PDF Portable Document Format.

PKCS Public Key Cryptography Standards.

RC2 Rivest Cipher 2.

RC5 Rivest Cipher 5.

RC6 Rivest Cipher 6.

RIPEMD RACE Integrity Primitives Evaluation Message Digest.

RKE Remote Key-less Entry.

RSA Rivest–Shamir–Adleman.

SHA Secure Hash Algorithm.

SSH Secure Shell.

SSL3 Secure Sockets Layer 3.

TEA Tiny Encryption Algorithm.

URL Uniform Resource Locator.

XPath XML Path Language.

Chapter 1

Introduction

The concept of 'digital' is widely accepted as the new status quo of the society. The steadily increasing interconnectivity between devices has broken the communication barriers, becoming the main driving force behind globalisation. As a result, data flows more than ever across private and public networks. Cyber security, especially cryptography, plays a crucial role in protecting this information ensuring its confidentiality, integrity and authenticity.

The automotive industry is one of the many fields influenced by the latest advances in technology. Automobiles have been progressing from mechanical units to electronically controlled units for the past 20 years. Today's cars are relying heavily on using embedded devices to produce basic capabilities - from controlling the doors to assisting with transmission control and even braking. Considering that in average half of the population of the European Union owns a vehicle (Eurostat 2017), the security of such devices is critical. The safety requirements of automobiles are continuously augmented as the world is rapidly sailing towards the era of autonomous cars.

Cryptography plays a crucial role in securing the various electronic components found in automobiles from malicious attackers. According to Schütze (2011), it is in particular used for RKE, electronic immobilisers, secure access to multimedia instruments, tuning protection and odometers protection. Unfortunately, most of these electronic systems are proprietary. Therefore, almost no information has been publicly revealed regarding cryptographic algorithms used in automobile industry.

1.1 Contributions

In summary, this paper makes the following contributions to its corresponding academic research field:

- It provides information regarding the design and underlying implementation of a data collection script which was used to gather 17.1 GB of data.

- It presents the design and describes the underlying implementation of a data analysis script that extracted 15576 firmware binaries from the collected data set and produced a final report summarising the output of algorithm signature detection.
- It presents a formal and comprehensive analysis of the aforementioned report, which includes a description of the various cryptographic algorithms found and a summary of the patterns emerging from the distribution of these algorithms among the gathered data.

1.2 Report Structure

This report follows the following logical structure:

- **Technical Background** This chapter provides a high-level background and description of key technical and domain-specific terms.
- **Related Work** This chapter presents relevant work conducted in the past related to this particular research area.
- **Data Collection** This chapter presents the process conducted to acquire a large data set, including the design and implementation details of the automated data collection script.
- **Data Analysis** This chapter presents the tools employed in the analysis of the gathered data.
- **Results** This chapter examines the obtained algorithm signatures, providing in-depth investigation of the results.
- **Project Management** This chapter describes the project management practices and principles followed throughout the delivery of this project.
- **Summary and Conclusion** This chapter describes the personal achievements of this project, further work that can be conducted in the future and formulates the conclusion of this report.

Furthermore, support material can be found at the end of the report. This includes a detailed explanation of the contents of the submitted zip file and instructions to run the software developed as part of this project.

Chapter 2

Technical Background

2.1 Firmware

Firmware is computer software specifically designed to perform well-defined tasks and incorporated into the hardware of a device. Thus, its executable code and data is stored in a binary file and can be easily analysed. It is usually stored in non-volatile[1] memory and provides low-level control for the hardware. Although the term is not commonly used outside the technology world, firmware is found in any electronic device such as smartphones, laptops and even TV remote controls.

2.2 Web Scraping

Web scraping can be defined as the process of gathering data of interest from websites. Even though this task can be manually performed by a person, the term generally refers to the automated process. A web scraper can access a website either directly, using HTTP, or via the inherent capabilities of a web browser. Conceptually, the process is simple and similar to the steps a human takes when browsing. The first step implies fetching the web page, which is exactly the same action a browser does when a person visits a website. Once the page is downloaded, the content is analysed through processing techniques such as parsing, searching, copying or reformatting depending on the aim of the task. Sometimes web scraper systems need to bypass checks enforced by websites in order to prevent automated data gathering scripts, such as CAPTCHA codes. This is achieved by using advanced technology solutions based on computer vision or natural language processing.

Glez-Peña et al. (2013) describes three main approaches in developing a web scraper based on either libraries for programming languages, dedicated frameworks or desktop

[1] A form of digital memory whose contents are saved even if the device is turned off or loses external power supply

software.

2.2.1 Libraries

This technique for building a web scraper relies entirely on the basic functionalities implemented in every modern programming language, such as HTTP handling, HTML DOM parsing and regular expressions. There are also sophisticated third-party libraries that may speed up these processes or use more efficient extraction methods - for instance XPath[2] and CSS selectors[3]. Although this technique offers full control over the design and capabilities of the automated script, it represents a highly laborious and time-consuming approach. Moreover, scrapers are sensible pieces of software that must be updated periodically because they heavily depend on changeable HTML code. In the case of a compiled language, such as Java, every rework of the program will result in a costly recompilation. Therefore, using libraries to develop web scrapers is typically not the best option as the drawbacks overrun the benefits in a general purpose context.

2.2.2 Frameworks

Frameworks save implementation time and facilitate easy integration with existing systems. However, this occurs at the expense of losing the ability to customise the functionalities of the scraper. There are plenty of frameworks that can be used for web scraping depending on the preferred programming language and the extraction options. Such a framework is represented by Selenium, which is presented later in this report.

2.2.3 Software

This option involves using dedicated software for scraping, thus removing the need for any programming knowledge. Unfortunately, using such software limits the data gathering capabilities as the system cannot be altered to satisfy specific user requirements.

2.3 Cryptographic Algorithms

Cryptography is conceptually defined as the study of methods for achieving secure communication and, more generally, for protecting private data. At the heart of modern cryptography lie key properties such as data confidentiality, data integrity and data authentication. These particular requirements are critical for a cryptographic algorithm.

[2]Check `https://www.w3schools.com/xml/xpath_intro.asp` for information about XPath

[3]Check `https://www.w3schools.com/cssref/css_selectors.asp` for more information about CSS selectors

Cryptographic algorithms are built on on mathematical problems considered to be infeasible - that is, they cannot be solved using current technical capabilities. Perfect cryptographic systems exist, such as the one time pad[4], but these are tough to implement and are not adopted for general purpose use.

Based on their design and purpose, cryptographic algorithms classify in symmetric-key algorithms, public-key algorithms and cryptographic hash functions.

2.3.1 Symmetric-Key Algorithms

Symmetric-key algorithms are defined as cryptographic algorithms that make use of an identical secret key both for encrypting the plaintext and decrypting the cyphertext. Therefore, the key must be shared between parties using a protected channel to securely communicate. This represents the main drawback of symmetric-key encryption, since they require additional techniques to share the key securely.

Symmetric-key cryptography involves two cypher types: stream cyphers and block cyphers.

2.3.1.1 Stream Cyphers

A stream cypher is a cryptographic cypher which encrypts one plaintext bit at a time using the secret key. It is faster than a block cypher, but it leads to significant security vulnerabilities when used incorrectly.

2.3.1.2 Block Cyphers

A block cypher is a cryptographic cypher which takes a number of bits (i.e. a block) and treats them as a single unit. If there are fewer bits than the size of the block, the plaintext is padded according to some rules. Block cyphers are widely used in the design of modern cryptographic algorithms.

2.3.2 Public-Key Algorithms

Public-key systems, also known as asymmetric algorithms, are cryptographic algorithms based on pairs of keys formed of a public key, which is widely available and a private key, kept as a secret. Secure channels are not required in public-key encryption key exchange. The public key is not suitable for decrypting encrypted messages and so it is useless in the hands of possible attackers. Only the private key can be used to decrypt messages - therefore it is essential to properly protect it.

[4] An uncrackable encryption method that relies on a random one-time pre-shared key at least the size of the message.

Due to the design considerations behind it, public-key cryptography provides two functionalities. Firstly, it provides digital signature for messages, effectively ensuring authentication and integrity. To digitally sign a message, the sender of the message encrypts the hashed message with their private key and attaches this encrypted hash to the message. Then, the recipient checks the authenticity of the message by decrypting the attached signature using the public key of the sender. If the obtained hashed value from the signature is equal to the hash of the received message, the integrity and authenticity of the data are proven. Secondly, public-key cryptography ensures secrecy by encrypting plaintext messages using the public key of the recipient. Only the person which possesses the private key of the recipient will be able to read the encrypted message.

Compared to symmetric-key systems, asymmetric-key systems have enormous computational complexity and are much slower. Thus, they are generally used for data which is smaller in size. In practice, both symmetric-key cryptography and asymmetric-key cryptography are combined to obtain unbreakable secure communication. This is done through encrypting the messages using symmetric-key encryption, since this approach is faster. Then the symmetric-key is encrypted using public-key encryption and it is securely shared between the parties.

2.3.3 Cryptographic Hash Functions

Cryptographic hash functions are mathematical functions that map data of random size to a fixed sized array of bits and are designed to be almost impossible to invert. The only way to retrieve the data from a hashed value is to search for all possible initial messages that may have generated it or use already computed tables of matched hashes to speed up the searching process. The cryptographic hash functions are usually used for digital signatures and storing data that should be inaccessible such as passwords.

Chapter 3

Related Work

Firmware is ubiquitous in today's world, since it can be found in every electronic device and automated component. Thus, there is a constant need for analysing firmware binaries in order to discover security vulnerabilities and develop patches that mitigate these dangers. Much of the firmware source code is proprietary and it is not publicly available. Therefore, the only way to analyse pieces of firmware in the absence of source code is to reverse engineer the binary code. Disassembling binaries and manually turning the assembly code in a higher-level language, or even analysing it, is a laborious and time-consuming task and it is infeasible when large amounts of data are involved. As a solution for this issue, methods to automatically reverse engineer binaries have been developed.

3.1 Reverse Engineering Software

As stated earlier in this paper, firmware is essentially software with a well-defined purpose embedded in hardware. Automatically reverse engineering binary programs is a problem researchers have sought to resolve for a long time. Cifuentes & Gough (1995) identify the critical issues in developing such tools. Some of these problems are the indistinguishable representation between data and instructions in modern computer's architecture, the huge number of extra subroutines introduced by the compiler in the produced binary and the differences in libraries enforced by the distinction between operating systems. The same publication presents the first successful automated decompiler with its underlying design and limitations that handles the presented problems. It also recognises the importance of automatic reverse engineering in checking software for malicious code. The described decompiler targets the devices using Intel 80286 architecture and running the DOS operating system, taking a binary program as input and generating a C program as output. The decompiling system employs a module that stays at the heart of modern firmware reverse engineering programs. At the heart of this module is an automatic signature generator, which creates signatures specific for each compiler and library function and

stores them into a database. A library function for a specific compiler is determined by a unique set of instructions that represents the library signature. Based on the signatures stored in the database, the decompiler checks whether a subroutine is a library function.

Even though the first successful compiler has been described a long time ago, people started focusing on lower-level control software recently. RivNIC, a tool for reverse engineering device drivers, is presented in Chipounov & Candea (2010). The research describes a technique to automatically decompile a device driver. It also recognises the implications the tool provides for mitigating security vulnerabilities in drivers. The program takes as input a closed-source device driver binary and produces higher-level code that implements the same hardware protocol as the original driver. Moreover, the tool is able to produce code implementing the same functionality, but targeting a different operating system than the original binary code. It achieves this by closely recording the driver activity in relation to its hardware, the instructions it executes and the memory it accesses.

3.2 Reverse Engineering Firmware

Delugré (2010) is one of the first works in reverse engineering firmware binaries and continues in the direction set by Chipounov & Candea (2010), based on analysing network drivers' firmware. Considering the continuous development of embedded devices and increase of computer interconnectivity, more research has been done in reverse engineering and analysing firmware. An interesting and useful paper is Zaddach & Costin (2013) which summarises all the relevant work regarding automatic reverse engineering for firmware binaries. It also briefly presents the most commonly used hardware architectures in embedded devices, touching processor architectures, communication lines, operating systems, libraries used etc. Moreover, it provides a comprehensive list of state of the art community tools for reverse engineering firmware binaries and embedded devices such as binwalk, signsrch, BAT and many others.

Up to this point all the firmware security analysis and reverse engineering has targeted only specific devices. Costin et al. (2014) is the first public large-scale analysis of firmware binaries. The paper presents the number of vulnerabilities found by automatically reverse engineering 32 thousand firmware binaries and a comparison between different reverse engineering tools. Unfortunately, there is no information or analysis regarding cryptographic algorithms in the data sample. The paper also introduces the concepts of static analysis and dynamic analysis. Dynamic analysis refers to performing analysis of embedded devices by running the device, but it is rarely performed as it is infeasible to gather a large amount of data. For this reason, static analysis is more common as it scales better and does not require the actual device.

3.3 Cryptographic Analysis of Automotive Firmware

Vehicles undoubtedly established themselves as indispensable in our perception of the modern world. Ever since technology began to shape the design of automobiles there was a need for exhaustive analysis of embedded systems in cars, especially from a security viewpoint.

One of the first attacked system is the RKE which, in some cases, suffered of serious security vulnerabilities due to use of weak proprietary cryptography. One of the most known examples is the KeeLoq RKE system, used in some cars. There are various papers describing attacks against KeeLoq such as Indesteege et al. (2008), Bogdanov (2007) and Eisenbarth et al. (2008). Garcia et al. (2016) presents an analysis of the Hitag2 RKE scheme - another infamous example implemented in millions of vehicles.

The Car2X communication system is quite recent. It provides car-to-car short-communication and car-to-infrastructure short-communication. Schütze (2011) introduces a formal analysis of the cryptographic algorithms used in this system. It also states that vulnerabilities in this system would allow external information to affect the behaviour of the car.

Even though countless security analysis have been conducted on individual automobile systems, no publication focuses on producing a formal large-scale analysis of cryptographic algorithms used in automotive firmware. Thus, the need for research in this particular field arises.

Chapter 4

Data Collection

A suitably extensive data set is required to be processed as part of the analysis stage for this research to reveal applicable and valuable insights in the discussed topic. However, trying to manually gather a large amount of firmware binaries is both very inefficient and infeasible in view of this reports' time frame.

As a result, devising an automatic method of collecting data becomes a necessary step. Since topic-related websites represent the main sources containing references to automobile firmware binaries, the aforementioned issue can be solved through developing a web scraping script.

4.1 Script Specification

The data collection script must be built in such a way that it will provide the core functionality of web scraping. Moreover, it should support execution on general-use PC operating systems since computers provide more power and larger storage units than a mobile device. Robustness represents a critical requirement of the script due to the highly responsive nature of web scrapers to any change in the website structure. Subsequently, the tool should provide other relevant capabilities that wrap around the core features and serve as improvements to the overall performance of the product.

The web scraping script has been built based on the following comprehensive list of requirements.

4.1.1 Functional Requirements

- The script must keep records of each source thread where the collected data resides. More specifically, the script must render the website in PDF format and save the URL of the thread.
- The script must keep accurate and separate logs for each running scraper.

- The script must run undetectably by simulating human behaviours accepted by corresponding remote servers.
- The script must download relevant data matching supplied search terms and store it in a predefined format.
- The script must be able to provide *save* and *load* options for configuration purposes.

4.1.2 Non-functional Requirements

- Adaptability: The script must be easily extendable to scrap any given website.
- Scalability: The script must support a given number of web scrapers running in parallel on different source websites.
- Security: The script must preserve the anonymity of the user and the privacy of the connection as much as possible.

4.2 Software Architecture

The architectural design of the web scraping script must be chosen such that it is able to offer both the functional and non-functional requirements mentioned in the previous section. To satisfy all the constraints, the script has been built following the simple plugin oriented design, which is described in Figure 4.1.

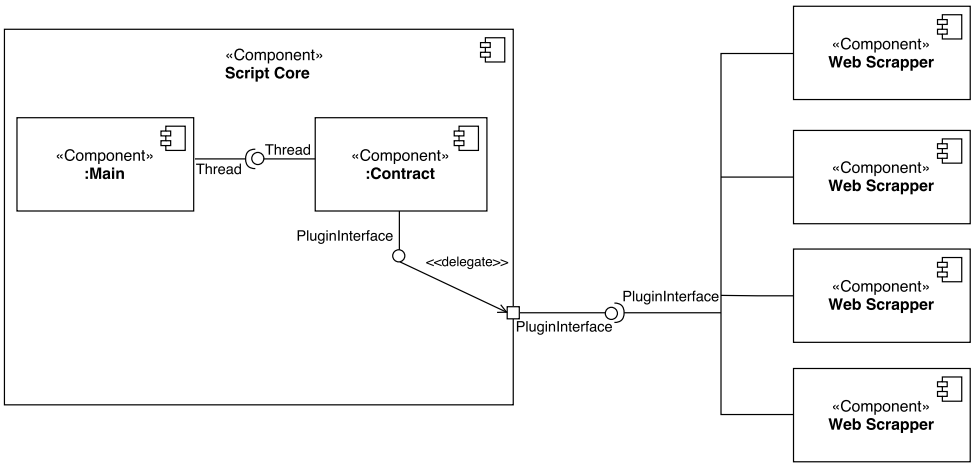

Figure 4.1: The component diagram of the web scraping script.

The core of the script, which represents the entry point of the software, handles the configuration and starts each of the scrapers. These scrapers are external files that can be added or removed from the core, acting as plugins for the main application.

For a file to be considered a valid scraper, it must respect the rules imposed by a contract defined in the core of the script. As long as the scrapers abide these constraints, the core does not look into the particular intricacies of their implementation. This means that the plugins can be acquired or developed independently and that they can follow different implementation approaches as long as they match the contract provided by the script core.

The chosen architecture effectively ensures robustness, as the scrapers do not directly interact with each other. Should one of the scrapers encounter an error, its behaviour will not be propagated to the other plugins. Furthermore, the design facilitates adaptability by allowing the addition of new web pages. For any such website, a new scraper that obeys the contract can be developed and added to the script without changing the core. Lastly, adding and deleting the scraping modules happens at runtime, which provides system flexibility.

4.3 Script Implementation

The script provides straightforward core functionality - however, the implementation becomes quite complex in the light of the described additional features. Although the need for extra capabilities could be argued, they offer great benefits for the users of the system. For user experience considerations, the script has been designed as a command line software that is intuitive and quite easy to use.

Implementation of the tool and its emerging functionalities is based on a combination of numerous state of the art technologies.

4.3.1 Python

The key step was choosing a programming language that would support development of described functionalities, while also offering portability and easy deployment on various platforms.

The script is entirely implemented in Python 3.6 (*Welcome to Python.org* 2018). Python is an easy-to-use programming language with a large library base – the standard libraries and some third party libraries provide everything needed for the script. Moreover, since it is an interpreted programming language, it suits the development of a web scraping script perfectly. Changes in the web scraper that reflect variations in website code do not result in recompiling any of the code. It also provides script portability on all major operating systems, such as Windows, Mac OS, Linux and Unix. The decision to use Python was also based on the ease of package-management provided by pip [1] and the ability to pack

[1] Read more about pip `https://docs.python.org/3/installing/`

required libraries in virtual environments, thus achieving efficient deployment.

Python has been primarily used to combine capabilities of all other technologies. Moreover, the Python standard library has provided methods of resolving the functional requirements. For example, the contract between the core and the scrapers has been developed using the *thread* standard library and the object oriented features of Python. Secondly, the configuration and menu of the script have been implemented using the *configparser* module, which is also contained in the same library.

4.3.1.1 Anaconda Virtual Environment

Python applications generally make use of third party packages. These packages can sometimes rely on distinct versions of the same library. As Python cannot maintain two different versions of the same package in an environment, it will consider some applications unable to run. Virtual environments are a feature implemented in Python to mitigate such possible conflicting situation by providing environments that hold the necessary packages for each application. As a bonus, virtual environments provide powerful deployment support – the application can be deployed with its underlying environment, which guarantees all required libraries are automatically installed. Unfortunately, an issue arises when running the script on Windows, since pip cannot manage the installation of some required packages in this particular environment.

To overcome this problem, Anaconda (*:: Anaconda Cloud* 2018) has been used. Anaconda is a free package management tool that provides virtualisation of environments and is fully supported by Windows, Mac OS and Linux machines.

4.3.2 Selenium

Glez-Peña et al. (2013) present various approaches to developing a web scraper. The scope of the software represents a key factor in the choice of a particular web scraper technology. In the context of this script, the ability to replicate human behaviour was a critical aspect to consider.

Selenium (*Selenium - Web Browser Automation* 2018) is described by its developers as a suite of free browser automation tools for testing purposes. However, the capabilities offered by Selenium are not restricted to testing purposes. The Selenium WebDriver API - part of the Selenium tool suite - ensures the ability to automate any major browser using a programming language that supports it (such as Python) and a controller, which is also called a web driver. Therefore, a web scraping script can be easily developed based on its features. This API interacts with web pages through browsers and it can select elements using advanced features such as XPath and CSS selectors. Furthermore, it can interpret and execute JavaScript, while preserving a human appearance.

Unfortunately, the Selenium WebDriver API lacks file downloading capabilities - the need for a workaround arises, which is described later on in the report.

4.3.3 Google Chrome

There is certainly no shortage of web browsers to choose from. The obvious option would consist of a widely-supported, sophisticated, yet easy to use browser. For this project, Google Chrome (*Chrome Web Browser* 2018) has been used to scrap the chosen websites inasmuch as it is one of the most advanced browsers at the present moment, free to use and available for most operating systems. The initial setup is straightforward and its simplicity improves overall time performance.

4.3.3.1 Chrome Driver

The Selenium WebDriver API interacts with the web browser through the aforementioned controller, also called a web driver. The web driver implementation depends on the chosen browser and it is not part of the Selenium suite tools. Since Google Chrome is used for the web scraping script, ChromeDriver (*ChromeDriver - WebDriver for Chrome* 2018) must be used. ChromeDriver is available for all platforms where Google Chrome is also available. It provides advanced features such as tunneling Google Chrome connections via specific ports and connecting to websites without visually rendering them.

4.3.4 BeautifulSoup

The technologies described so far are enough to build a basic scraper that would crawl given websites, either sequentially or in parallel, and extract information from their content. Unfortunately, this functionality does not include downloading the detected data. Therefore, the solution to this issue has been devised through a combination of two well-known Python libraries.

The first step to downloading a data file from the website is to get the link to that resource. Most of the forums mark download links in a specific way - for example *url?action=dlattach;topic=35.0;attach=39*. Thus, the links corresponding to downloading relevant data can be extracted by searching for specific keywords or patterns. In the example given above, the keyword for downloading is *dlattach*.

BeautifulSoup (*Beautiful Soup: We called him Tortoise because he taught us.* 2018) is the first library in the two-step data downloading process. It provides parsing capabilities for markup languages, as well as effective methods for traversing, searching and modifying the tree obtained through parsing. These can be achieved through regular expression pattern matching, using CSS selectors or XPath traversals. As HTML represents a simple markup language, this library is the perfect tool to quickly retrieve information embedded

in the website. In this context, BeautifulSoup has been used to find the described download links in scraped web pages.

4.3.5 requests

After gathering a list of links pointing to corresponding data files, the next step is actually downloading these files onto a machine.

Even though there are numerous libraries that offer suitable download capabilities, *requests* (*Requests: HTTP for Humans — Requests 2.18.4 documentation* 2018) has been chosen to complete the second step of the data downloading process. Requests is a HTTP library for Python that provides advanced features, such as tunneling connections via proxies with no noticeable performance issues.

In the script, the *requests* library uses the cookies stored by Selenium to connect via HTTP to the links retrieved by BeautifulSoup. The result of the connection is then returned and the content is saved in a predefined file structure, which is enforced through Python.

4.3.6 wkhtmltopdf

As part of running the data collection script, the files are downloaded automatically in such a way that the user hardly notices the running process. In the end, when the results are analysed, a need to check the source of the gathered data might arise. To improve the speed of this process, the script renders the website where data has been found and saves it as a PDF file, alongside the URL of the website. Even though saving the URL is straightforward, there is no capability of rendering the HTML to PDF in Python. To overcome this, a combination of third-party software and Python library has been employed.

wkhtmltopdf (*wkhtmltopdf* 2018) is an external open source tool that serves to render HTML into PDF. It supports all major operating systems, thus preserving the portability of the script. It also provides advanced features useful for the data collection script, such as tunneling the connection through proxies and using cookies to access websites that require login.

4.3.6.1 pdfkit

Fortunately, the usage of the previously described tool is facilitated by a Python library that acts as its controller. *pdfkit* (*pdfkit* 2018) is an adapted version of the homonymous Ruby library, representing a Python wrapper designed for *wkhtmltopdf*.

In the script, *pdfkit* is used throughout the downloading stage, alongside the *requests* library. Whenever a file is downloaded, *pdfkit* renders the HTML of the source page and

saves it to a local directory when the corresponding PDF does not already exist.

4.3.7 Tor

Anonymity and secrecy are crucial properties of the data collection script, as the identity of the user must be protected in case of script detection. Furthermore, the contents of the connections must be kept secret. Ensuring these two properties brings forward an invaluable benefit for the user - they cannot get permanently blocked from a website for scraping activities.

To achieve these security requirements, Tor (*Tor Project — Privacy Online* 2018) has been used to devise a suitable approach. The Tor project is a network designed to protect the security and privacy of individuals on the Internet by making use of well-proven encryption algorithms. The network can be accessed through different software, such as the Tor Browser (Tor incorporated in Mozilla Firefox) or Tor Expert Bundle, which is a command line tool designed to access the network. When Tor is used, the connection is tunneled through a specific port. In the case of the data collection script, the chosen connection tool is Tor Expert Bundle, as there is no need for an actual web browser. This represents another third-party piece of software which will need to be configured and managed as part of the core script.

4.3.7.1 Stem

The answer to using Tor from the data collection script is Stem (*Welcome to Stem! — Stem 1.6.0 documentation* 2018) - a controller Python library part of the Tor project. Using this library gives the script total control over the Tor process, providing starting, configuring and stopping capabilities.

When the *start* command is run, the data collection script will activate the Tor process using the configuration contained in the script - this occurs only if the option to use Tor is set to 'true'. When the script connects to the Tor network successfully, the scrapers will be immediately started. Throughout the entire session, all connections made between the script and the websites are tunneled through the Tor network. This is the reason behind using the inherent tunneling functionalities of the presented libraries.

4.4 Results

All requirements have been successfully implemented and the commands mapped to each of the features are described in a comprehensive list, which can be seen by running the command *help* in the script (see Figure 4.2).

```
Please input a command:
-help
Here is a list of commands that can be used:
-help                                   Print the help panel.
-start                                  Start the script. Make sure it has at least one module added.
-add -scrapper [MODULE_NAME] [CLASS_NAME]   Add a scrapper to the script with the given scrapper name and class name.

-add -search_term [SEARCH_TERM]         Add a search term to the script.
-delete -scrapper [MODULE_NAME]         Delete a scrapper that is not currently running from the scrappers list.
-delete -search_term [SEARCH_TERM]      Delete a search term from the script.
-set -threads [POSITIVE INTEGER]        Set the max threads that can be run at once to another positive integer.
-set -chrome_path [CHROME PATH]         Set the path to the chromedriver binary.
-set -wkhtmltopdf_path [WKHTMLTOPDF PATH]   Set the path to the wkhtmltopdf binary.
-set -headless [BOOLEAN]                Set whether to run chromedriver headless or not.
-set -tor [BOOLEAN]                     Set whether to use tor or not.
-set -tor_path [TOR PATH]               Set the path to Tor bundle binary.
-set -socks_p [POSITIVE INTEGER]        Set the socks port for Tor. Note it must be a valid port.
-set -control_p [POSITIVE INTEGER]      Set the control port for Tor. Note it must be a valid port.
-export                                 Export the current configuration into the default file 'config.ini' so it
 can later be loaded.
-load                                   Load the 'config.ini' if it exists.
```

Figure 4.2: The output of the *help* command.

Figure 4.3 describes the configuration to fire up the data collection tool. The script gathered data from three topic-related forums. It searched all the posts related to the supplied keywords *firmware, flash, dump* and it identified downloadable data in all found threads. The data has been saved in a well-structured file tree, containing separate directories for each website name. These are in turn composed by folders corresponding to each thread name.

```
Please input a command:
-config
Current configuration is:
General Configuration
chromedriver_path = res/chromedriver.exe
wkhtmltopdf_path = res/wkhtmltopdf/bin/wkhtmltopdf.exe
run_headless = False
max_threads = 1
search_terms = firmware,dump,flash
Tor Configuration
use_tor = True
tor_path = res/tor/Tor/tor.exe
socks_port = 9050
control_port = 9051
Scrappers
scrappers = nefariousmotorsports_module digitalkaos_module ecuconnections_module
```

Figure 4.3: The configuration used to run the script.

As a result, the script managed to gather 17.2GB of data comprised of different archive types, image files, binary files and other text file types.

Chapter 5

Data Analysis

As a result of the collection process, a large data set comprising 356 plain binary files and a further 26,710 archives has been acquired. Taking into account the number of files, the analysis process must be accelerated through some type of automation method. Thus, a number of scripts have been developed to aid the analysis of the collected data samples.

5.1 Data Extraction

The gathered data is presented in a raw format. It contains a combination of different file types such as archives, binaries, images and text files. Therefore, the data must be effectively filtered to obtain a plain, uncompressed set of binaries.

To achieve this, a number of technologies described in the following subsections have been employed.

5.1.1 PowerShell

Taking into consideration the fact that the entire project has been developed on a machine running Windows, the obvious choice of a suitable scripting language is Windows PowerShell (*PowerShell Scripting — Microsoft Docs* 2018). PowerShell is a scripting language based on the .NET Framework. It offers a command-line shell which is similar to the Linux alternative, with the difference that PowerShell is object oriented (i.e. running a command outputs an object).

PowerShell has been used throughout the entire data analysis stage to develop numerous helper scripts. For instance, this language represented the code base of the data extraction script. More specifically, it was used to implement the core tasks of this script: filtering the archives and plain binaries from the rest of the data, controlling the extraction of the .rar or .zip files and moving all filtered firmware binaries to a new directory named *bins*, where the actual script resides.

5.1.2 WinRAR

Even though PowerShell is able to manage most of the required tasks, it lacks inherent extracting capabilities. To solve this shortfall, the WinRAR (*WinRAR archiver, a powerful tool to process RAR and ZIP files* 2018) tool has been utilised. WinRAR is one of the most complex archive managers designed for Windows, capable of processing various archive formats. It is also available as command line only for Linux and Mac OS.

In this particular helper script, PowerShell stores a list of all the archive files available in the data set. One by one, each archive is passed to the WinRAR process which specifically detects binary files. If a binary file is found in the current archive, then it is extracted. There are some cases when the archives are password-protected - in this situation, WinRAR has been configured to ignore these files, which will remain stored in the data set for future analysis.

5.2 signsrch Analysis Script

At this stage, the data has been successfully filtered and the plain binaries are stored in a clean directory. The next step is to analyse the firmware binaries with the purpose of identifying special algorithms, such as compression algorithms, cryptographic algorithms, encoding algorithms and others. There are methods of automatically detecting algorithm signatures in firmware binaries - check chapter 3 for papers that discuss such options. However, since a Windows environment is used for this project, the number of available tools is limited, rendering the choice much easier.

signsrch (*Luigi Auriemma* 2018) is a widely-acclaimed and free to use Windows-only tool created by Luigi Auriemma. The author defines its main feature as being able to "recognize tons of compression, multimedia and encryption algorithms and many other things like known strings and anti-debugging code", considering it useful "for figuring or having an initial idea of what encryption/compression algorithm is used for a proprietary protocol or file".

Running *signsrch* on a firmware binary produces a well-structured report that describes what algorithm signatures have been found, if such any signature exists. This is very useful as a starting point for an exhaustive analysis. Unfortunately, as the size of the data sample collected in this project is quite large, it is very time-consuming and inefficient to manually look through all the produced reports.

For this reason, *signsrch* has been embedded in a PowerShell script that stores a list of all the existing firmware binaries and passes each of them to *signsrch*. After *signsrch* produces corresponding reports for all the firmware binaries received, PowerShell checks each report to see if an algorithm has been found. If nothing has been detected in the report, it will disregard the binaries that do not contain any algorithm signature.

The output of the script is a final report that summarises information regarding the total number of binaries analysed, the number of binaries which contain at least one algorithm signature and the full path of each binary and its underlying signatures. This report was instrumental in the analysis stage, as it encompasses all key aspects of the data sample in a single file.

5.3 binwalk Analysis Script

Tools designed to automatically detect algorithms in binaries are not without fault. This is due to the intricacies of source code compilation on modern computer architectures. Thus, there is always the possibility of obtaining a false positive (i.e. observing a false algorithm) or disregarding a true positive (i.e. bypassing a suitable algorithm), caused by slight variations in the signatures that define an algorithm. In order to minimise the risk of such flaws impacting the accuracy of the results, a second binary analysis tool has been used on the same data set.

binwalk (*ReFirmLabs binwalk* 2018) is a Python-based "fast, easy to use tool for analyzing, reverse engineering, and extracting firmware images". *binwalk* works in a similar way as *signsrch* - it outputs a report corresponding to every analysed binary, which contains information about the signatures found. The *binwalk* analysis has been conducted in an almost identical way as the presented *signsrch* analysis. The only notable difference is that the *binwalk* analysis makes use of two scripts to obtain a final report.

Since *binwalk* has been implemented in Python, it can be executed directly through Python or it can be easily imported in a Python script. Therefore, another script has been implemented in this programming language with the purpose of creating the desired report for each binary file. The script stores a list of all the firmware binaries in the data set and passes each one of them to the *binwalk* module, which in turn searches for specific algorithm signatures.

Once the reports have been produced, purely for the ease of implementation, a PowerShell script has been developed. This script checks the outputs of *binwalk* and outputs a final report which follows the same structure as the *signsrch* deliverable.

Considering that *signsrch* has been designed especially for Windows environments, the results produced by this tool will represent the primary source of the analysis process. Conversely, *binwalk* has been built targeting Linux and is considered to have experimental capabilities on Windows. Therefore, *binwalk* will be solely used with the aim of minimising false positives in the case of contradictory results.

Chapter 6

Results

The data extracting script described in chapter 5 managed to retrieve 15576 firmware binaries, which sum up approximately 6GB of data. As it has been previously mentioned, the analysis has been centred on the results returned by *signsrch*. When further information was needed to validate the accuracy of a result, *binwalk* has been employed to examine that particular signature.

6.1 Preliminary Results

From a total number of 15576 firmware binaries, *signsrch* has found 2874 binaries containing at least one algorithm signature (Figure 6.1).

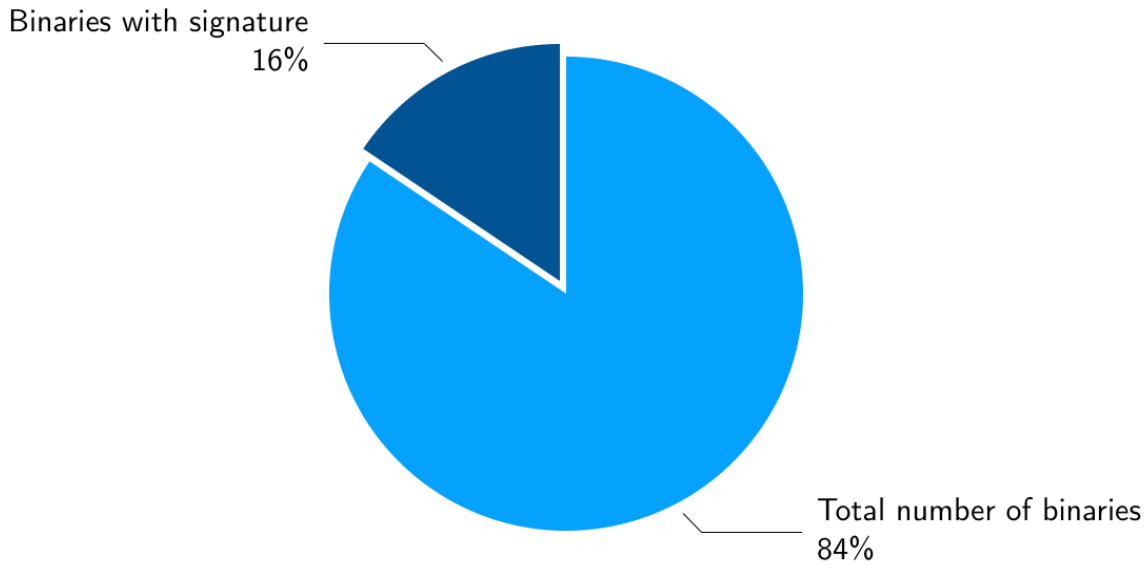

Figure 6.1: Pie chart showcasing the firmware binary distribution.

A good variety of signatures can be observed in the resulted binaries, which contain 300

distinct signatures. The wide range of signatures is instrumental in defining an algorithm distribution that will reveal interesting patterns. However, there is no exact mapping between a signature and an algorithm - that means that an algorithm can be represented by more than one distinct signature. This phenomenon occurs whenever the algorithm is defined by different signatures for both little-endian[1] and big-endian[2] machines or due to increased complexity. In the latter case, each function implemented in the algorithm may have its own distinct signature that will be detected by *signsrch*. After careful analysis and filtering of the signatures, 68 distinct algorithms have been identified. These algorithms can be classified into seven groups:

- Data Encoding and Decoding Algorithms, such as base64 encoding;
- Image Processing Algorithms, represented by image compression algorithms such as Jpeg;
- Data Compression Algorithms, including algorithms and libraries such as zlib, zinflate, unlzx;
- Error Detection and Correction Algorithms, containing mainly CRCs;
- Audio and Video Processing Algorithms, such as audio and video encoding algorithms and audio and video compression algorithms;
- Cryptographic Algorithms, including cryptographic related algorithms and libraries;
- Other algorithms, encompassing utility algorithms such as precomputed power tables and random number generators.

Figure 6.2 showcases the distribution of the different groups of algorithms presented in the above classification.

In the context of the present research it is very fortunate that the cryptographic algorithms have the highest diversity among all the algorithms in the analysed data. Table 6.1 summarises the findings related to this algorithm class.

Even though at a first glance the obtained results are encouraging, it is highly unlikely that some of the identified cryptographic algorithms are indeed present amongst them. The best example is the appearance of *padding* used in hash algorithms in 1962 out of 2874 binaries containing signatures. Furthermore, there are some surprising results, such as the presence of MD4 and Windows Cryptographic Hash Functions. To minimise the risk of reporting erroneous results, a further analysis based on *binwalk* has been utilised.

[1]Refers to systems in which the least significant value of a multi-byte data is stored at the lowest storage address

[2]Refers to systems in which the most significant value of a multi-byte data is stored at the lowest storage address

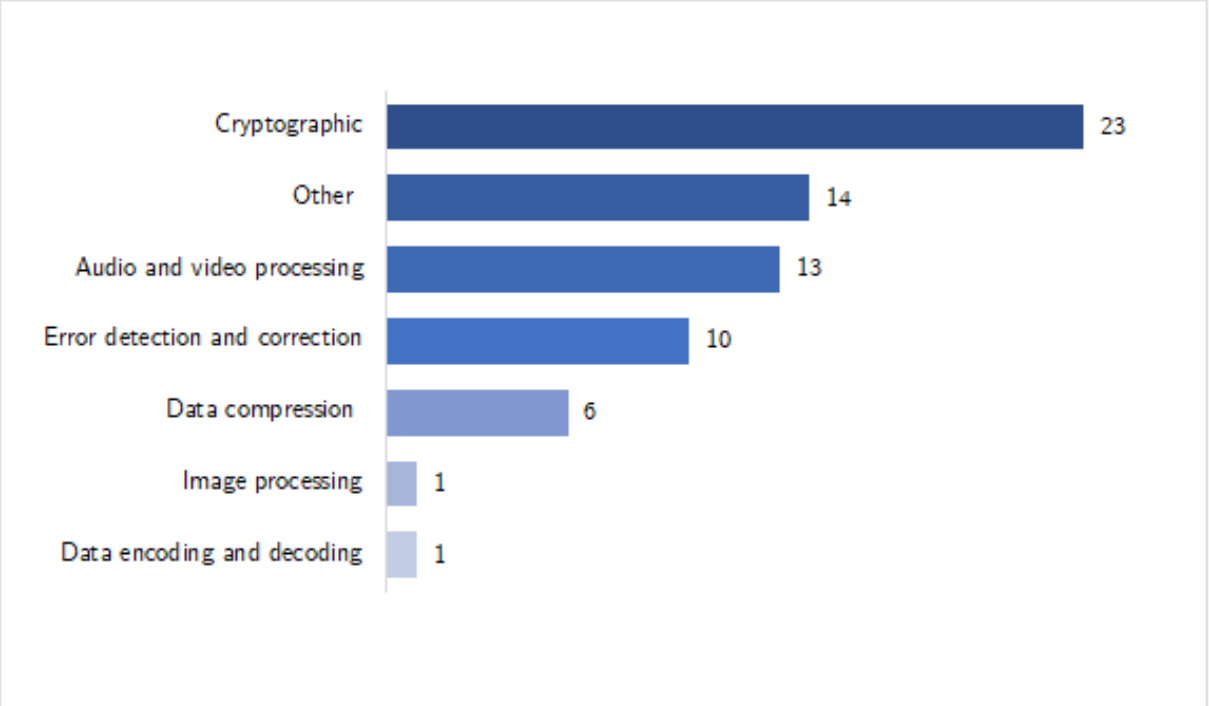

Figure 6.2: Bar chart describing the algorithms distribution among the data.

The outcome of this additional analysis step outlines approximately a half of the number reported by *signsrch* - 1317 binaries containing at least one signature. Despite the fact that this tool has identified a smaller number of signatures, it has proven valuable as a suitable method of ruling out some of the inaccurate data.

6.2 ACSS

ACSS is one of the functions named in the previous table that certainly draw attention. Even though 16 binaries have been reported to use this algorithm, there is actually little mention of its capabilities on the Internet. This cryptographic algorithm seems to be related to SSH as found in (*Ghudson SSHWC ACSS* 2018) and (*Obfuscated OpenSSH ACSS* 2018).

A thorough check of the *binwalk* output revealed a relation between the presence of ACSS in firmware binaries which contain operating systems or file systems.

6.3 MD4

Although MD4 is one of the well known cryptographic hash functions, its ability of providing security has been severely affected. Dobbertin (1996) describes an attack designed to find a collision for MD4 which can be executed in just a few seconds on a computer. The

Cryptographic Algorithm	Number of Binaries
ACSS	16
AES	139
Blowfish	27
DEAL	2
DES	104
Gost	162
Haval	27
Lucifer	32
MBC2	199
MD4	13
MD5	17
Misty	159
Noekeon Nessie	15
padding used in hash algorithms	1962
PKCS padding	86
RC2	7
RC5 and RC6 magic values	3
RIPEMD	127
SSH RSA	3
SSL3	41
TEA	45
Windows Crypto Hash Functions	10
ZipCrypto	1

Table 6.1: Cryptographic algorithms found

processed results have revealed a concerning presence of MD4 hash functions in modern firmware binaries.

After careful analysis of the binaries in question, it seems that the algorithms are not actually MD4 functions. The binaries that contain an MD4 signature are in fact some of the binaries that have also been associated with RIPEMD. This irregularity in *signsrch* detection has occured inasmuch as the RIPEMD algorithm is essentially based on MD4.

6.4 Windows Cryptographic Hash Functions

Another peculiar appearance in the obtained firmware binaries is represented by the Windows cryptographic hash functions. These are functions employed in Windows operating systems with the purpose of hashing distinct data. They make use of widely-spread hashing algorithms such as the SHA hash family.

A close inspection of the firmware binaries containing this signature has revealed that they are associated with Windows CE - an operating system from Microsoft that targets embedded devices.

6.5 Padding used in Hashing Algorithms

Probably the most considerable surprise is represented by the presence of 1962 binaries affiliated with *padding* used in hash functions. This appears to be very unlikely from a logical point of view, as the number of identified hashing functions is much lower.

Unsurprisingly, a second *binwalk* analysis was able to pinpoint the erroneous results. Furthermore, distinct patterns can be observed by inspecting the connection between the *signsrch* results and the *binwalk* ones.

Whenever *padding* used in hash functions appears in isolation, the *binwalk* tool either reports the presence of x509 certificates in DER format or it does not detect any algorithm in the supplied binaries. Conversely, if *padding* appears alongside MBC2 and Gost, *binwalk* identifies a cryptographic algorithm, without actually specifying the type. Last but not least, when *padding* is contained next to MBC2, Gost and SSL3, *binwalk* reports the cryptographic algorithm to be represented by Blowfish.

In the light of the presented irregularities, it can definitely be inferred that the results produced by such analysis tools are not precisely accurate. Unfortunately, the lack of documentation describing MBC2 makes it difficult to reach a conclusion based on the observed patterns - at the moment, (*MBC2 Readme* 2018) is the only reference that names MBC2 as a cryptographic algorithm.

6.6 AES

AES is a state of the art cryptographic algorithm, representing the current cryptography standard in all industries. AES has substantially increased its popularity over the recent years, defining a rising trend which is expected to appear in the discussed data set. Even though inaccuracies in the detection algorithm have compromised certain results, AES has been accurately indentified in the corresponding firmware binaries. This can be easily validated as multiple phases of the AES algorithm are present in the file. Following a manual analysis of all the binaries containing AES and a further check of their data source, information related to the car manufacturer and manufacturing date has been retrieved. Figure 6.3 describes the pattern in AES usages in automobiles which has been revealed by data sample containing time references.

Through looking at the data set, it can be noticed that 2007 is the earliest year in which an automobile component using AES has been produced. Furthermore, from 2007 to the present year, AES has been utilised constantly in the automotive industry.

There has been a high level of difficulty in finding automobile date references. Nonetheless, this has allowed retrieval of more information about the actual car manufacturers using AES in the analysed firmware binaries. Out of 139 firmware binaries containing AES,

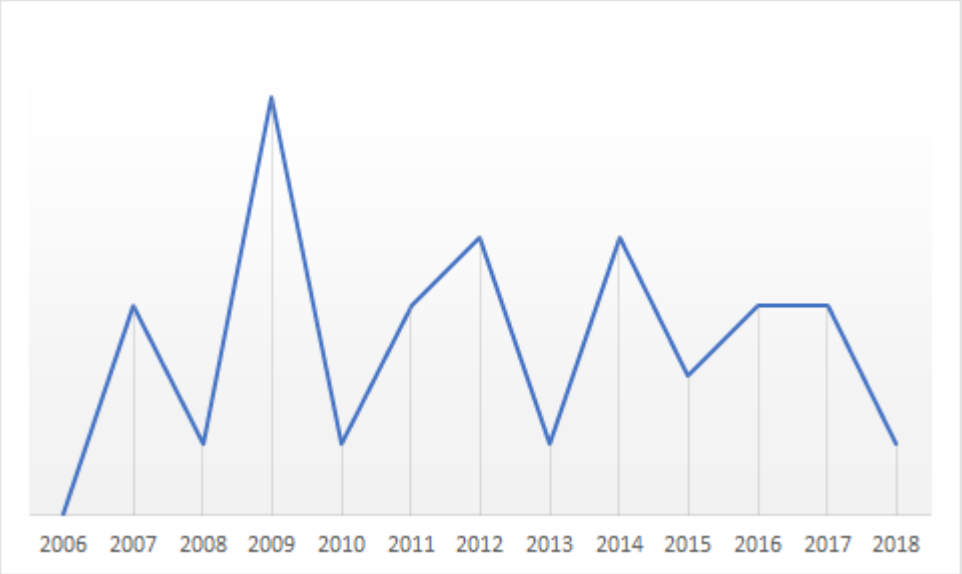

Figure 6.3: Line chart showing the year distribution of AES among the data set.

more than half of them (approximately 73) have been mapped to a car maker. Table 6.2 summarises these findings. Note that these findings cannot be transformed to a larger-scale due to the lack of data and this purely refers to the analysed data sample.

Automobile Maker	Number of Binaries
Audi	24
BMW	3
Fiat	2
Ford	3
Honda	1
Jaguar	1
Nissan	6
Porsche	2
Renault	3
Seat	5
Skoda	5
Volkswagen	17
Volvo	1

Table 6.2: Automobile manufacturers using AES

6.7 DES

DES is another common and well-known symmetric-key algorithm, representing the previous standard for encryption up until 2001. DES has been accurately reported by the reverse engineering tools - this affirmation is enforced by the fact that multiple steps of DES

have been detected in the algorithm. Unfortunately, no data related to the manufacturing year of these binaries has been retrieved. This lack of information did not compromise a further analysis, where, 72 out of 104 binaries with DES have been mapped to a car manufacturer (Table 6.3).

Automobile Maker	Number of Binaries
Audi	2
Ford	60
Mini	1
SsangYong	4
Volkswagen	5

Table 6.3: Automobile manufacturers using DES

It can be clearly seen from the data sample that Ford dominates the firmware binaries usages which involve DES. After careful examination, it was discovered that these binaries are designed for older Ford components, such as Ford 6000CD, thus belonging to cars produced between 2002-2007.

Chapter 7

Project Management

Effective management is a critical aspect of any successful, large-scale project. Therefore, various project management principles and practices have been applied to guarantee timely delivery of high quality work.

During the first weeks, a Gantt chart has been created to serve as a roadmap against which progress can be easily tracked. It clearly describes the development stages of the project (see Figure 7.1). The project has been split in such a way that milestones correspond to the three major deadlines: the inspection week of the first term, the demonstration week of the second term and the report submission.

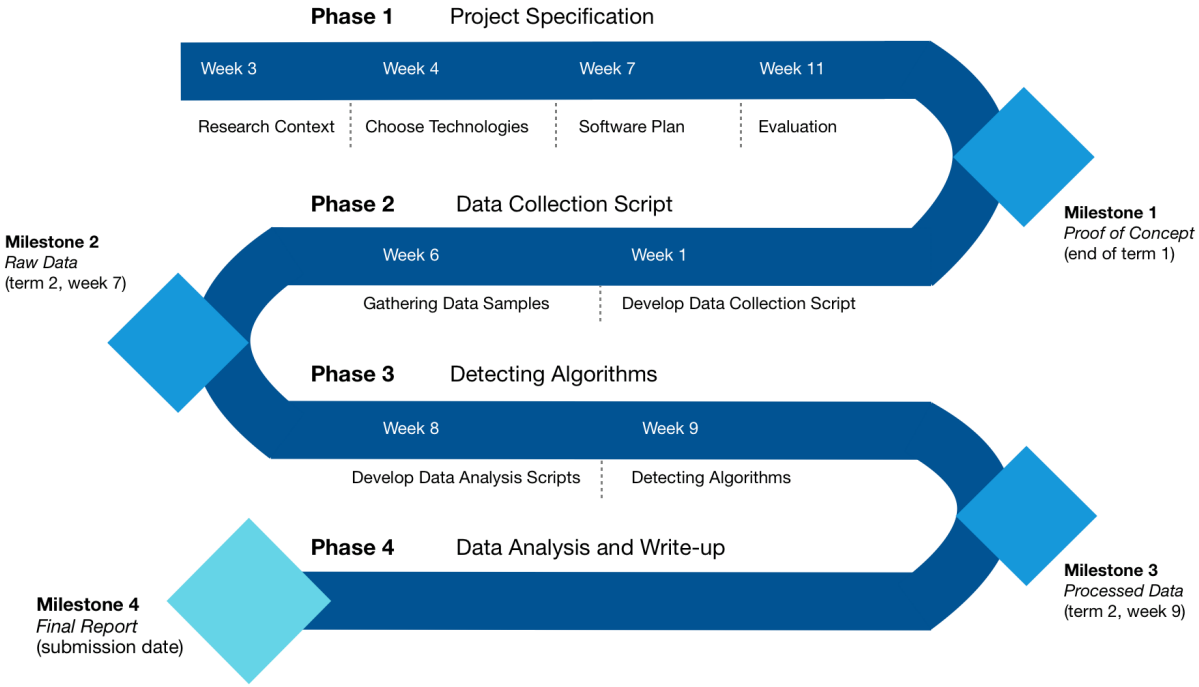

Figure 7.1: Gantt chart in the form of a project roadmap.

The first phase spans over the entire first term and includes the research needed to

build a good foundation for the development plan - from understanding the project context and the impact of its contributions to pinpointing technologies that would ensure effective delivery of a data collection script. The stage results in a proof of concept, which was presented in the inspection week.

The next five phases spread over the second term and describe the development of the data collection script, the actual data collection stage, the development of the data analysis scripts and the detection of algorithm signatures contained in the data set. Before demonstration week, all the scripts are expected to be fully functional, with preliminary results available for discussion. The project finalises with the analysis of the obtained results and the write-up, thus spanning to the report submission deadline.

7.1 Software Development Methodology

The project resource consisted of a single developer. Consequently, the underlying development methodolgy has adopted only context-relevant agile software development principles and practices.

Although a 'traditional' agile methodology has not been followed, the deployed software development methodology adhered to most of the agile mindset[1]. The software delivery aspect of this project centred on simplicity. Therefore, the tools have been designed and implemented in such a way that they achieve the minimum needed to complete the task, following the agile principle of 'good enough' development.

An iterative development approach ensured a working version was always available, alongside offering a real measure of progress. The iteration period has been mainly a one week time frame, ending with the weekly supervisor meetings. The iterative approach also represented a prompt and fast way of adapting system specifications to any requirement changes. Therefore, unnecessary delays were completely avoided when the feedback revealed possible enhancements to the software.

The iterative development methodology includes the well-known *develop - test - release* cycle. This pattern ensured continuous development and testing of the script, with improvements made every time a problem had been detected or a change had emerged from a review meeting.

7.2 Task Management

The project phases described in the Gantt chart represent a general-view roadmap. These phases have been further split into more detailed tasks to provide clear progress tracking

[1]Read more about the agile principles `https://www.smartsheet.com/comprehensive-guide-values-principles-agile-manifesto`

capabilities. Moreover, various tasks have been grouped together to fit in one week time frames - this effectively maps task management to the chosen iterative development approach.

Working on such a vast project resulted in a backlog containing a large number of tasks. With the purpose of tracking personal progress while working on those features, Trello (*Trello* 2018) has been used during all project phases. Trello is a project management platform designed to manage organising and prioritising projects in a flexible way, making use of boards, cards and lists.

Figure 7.2 shows the management process of the tasks using a Kanban Flow[2] where three categories appear. The *To do* category stores cards describing tasks that have not been started yet. Tasks that have been started, but have not been finished are marked as *In progress*. Last but not least, finished tasks are moved in the *Done* section.

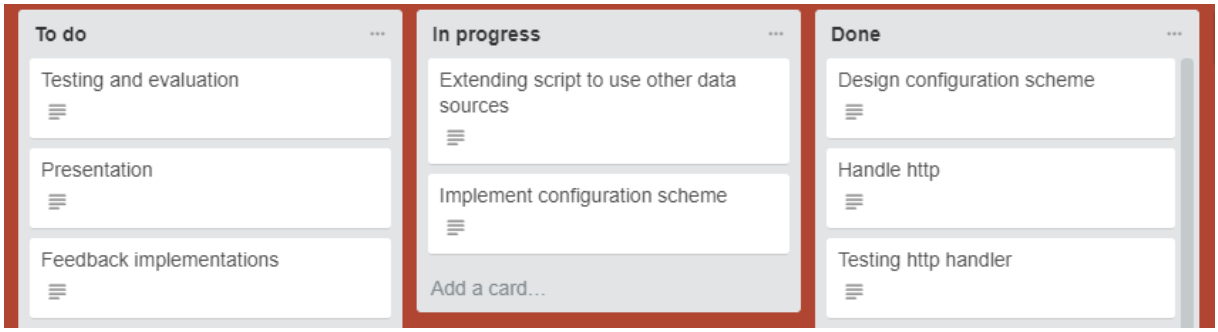

Figure 7.2: Preview of the Trello board with some tasks.

7.3 Version Control

Version control must be employed for any large scale project. It safely stores current versions of software, protecting users from possible data loss and hardware failures. Moreover, version control systems showcase work progress, facilitate sharing data between different work stations and provide the ability to roll back to previous versions of a project.

This project has been developed using the university Git server as a version control tool.

7.4 CASE Tools

CASE tools are pieces of software that offer valuable support in the design and development process of a project. Although software is solely used as a means to achieving the end goal of the research, final deliverables include complex tools that have been developed

[2]Read more about Kanban `https://www.versionone.com/what-is-kanban/`

throughout all stages of the project. The usage of CASE tools during the development of these applications has drastically improved overall work velocity.

The architecture for the data collection script has been clearly outlined in a diagram produced through *draw.io* (*draw.io* 2018), an easy to use flowchart maker. Furthermore, the code base of the developed scripts has been written in either Python or Windows PowerShell. On one hand, the Python implementation has been achieved with the help of Pycharm (*PyCharm: Python IDE for Professional Developers by JetBrains* 2018), one of the most powerful and complete IDEs at the present moment. On the other hand, the PowerShell scripts have been developed using Windows PowerShell ISE (*Introducing the Windows PowerShell ISE — Microsoft Docs* 2018), a free and comprehensive tool from Microsoft.

7.5 Supervisor Meetings

The progress of the project has been closely overseen and discussed during the weekly supervisor meetings. Each meeting marked the end of an iteration where the latest working version of the software was showcased. Following the demonstration, any feedback or changes in the requirements would be addressed in the next iteration.

Chapter 8

Summary and Conclusion

8.1 Further Work

This paper offers a high-level presentation of the process used to gather and analyse a data set consisting of automotive firmware binaries. However, it should be noted that work in this area is not completed. Future research can be continued in the direction set by this report through exploring other methods suitable for improving result accuracy.

A first possible stage could involve acquiring a larger number of data samples from additional data sources. This would guarantee that the outcome will contribute even more substantially to the research area.

Even though various approaches to minimise noise and remove erroneous data have been taken, a further filtering stage could be a next step towards validating the results. It would be highly recommended to run this phase on a Linux machine since data could be further processed using tools that are not available on Windows, such as BAT[1]. Through comparing the output from a wide variety of binary analysis tools, the number of false positives would certainly be reduced, which would determine an increase in the overall result accuracy.

Probably the most important contribution of this paper is the fact that it lays the foundation for in-depth security analysis of automobiles by exploiting attacks against weak cryptographic algorithms. Such discoveries would have a massive impact on the automotive industry and the security provided by these ubiquitous vehicles to passengers.

8.2 Personal Achievements

I would like to take the time to thank Dr.-Ing David Oswald one more time for offering me the excellent opportunity of working on this challenging final year project idea.

[1] http://www.binaryanalysis.org/

This project has enriched my knowledge in a wide variety of computer science topics – from learning unfamiliar frameworks, to using analytical tools and discussing cyber security concerns. On one hand, developing the data collection script offered me the chance to improve my then-basic Python skills and acquire knowledge about advanced technologies such as web scraping. Moreover, through implementing such a large-scale software project, I had the chance to apply the software engineering practices that I have learned in my degree, thus ensuring timely releases of high quality deliverables.

On the other hand, the analytical aspect of the report proved to be immensely valuable, as it added much more insight to the limited cyber security knowledge provided by a one-term module. On account of this opportunity, I became familiar with numerous uncommon cryptographic algorithms and their specific role in the industry. As a bonus, I have gained abilities in using two major reverse engineering tools, alongside learned a powerful scripting language (i.e. Windows PowerShell).

8.3 Conclusion

In conclusion, this paper presented a formal analysis of cryptographic algorithms detected in automotive firmware binaries. It began by outlining similar work conducted in this domain, alongside recognising the lack of such research in the automotive industry. It then described the approach used to gather large amounts of data necessary for formulating relevant conclusions. Taking into consideration the number of data samples, an overview of the filtering and processing of such data has also been presented. The final chapters offer an exhaustive analysis where peculiar cryptographic algorithms appearances and distributions of such algorithms are discussed.

Bibliography

:: Anaconda Cloud (2018).
URL: *https://anaconda.org/*

Beautiful Soup: We called him Tortoise because he taught us. (2018).
URL: *https://www.crummy.com/software/BeautifulSoup/*

Bogdanov, A. (2007), Linear slide attacks on the keeloq block cipher, *in* 'International Conference on Information Security and Cryptology', Springer, pp. 66–80.

Chipounov, V. & Candea, G. (2010), Reverse engineering of binary device drivers with revnic, *in* 'Proceedings of the 5th European conference on Computer systems', ACM, pp. 167–180.

ChromeDriver - WebDriver for Chrome (2018).
URL: *https://sites.google.com/a/chromium.org/chromedriver/*

Chrome Web Browser (2018).
URL: *https://www.google.com/chrome/*

Cifuentes, C. & Gough, K. J. (1995), 'Decompilation of binary programs', *Software: Practice and Experience* **25**(7), 811–829.

Costin, A., Zaddach, J., Francillon, A., Balzarotti, D. & Antipolis, S. (2014), A large-scale analysis of the security of embedded firmwares., *in* 'USENIX Security Symposium', pp. 95–110.

Delugré, G. (2010), 'Closer to metal: reverse-engineering the Broadcom NetExtreme's firmware', *Hack. lu* **10**.

Dobbertin, H. (1996), Cryptanalysis of md4, *in* 'International Workshop on Fast Software Encryption', Springer, pp. 53–69.

draw.io (2018).
URL: *https://www.draw.io/*

Eisenbarth, T., Kasper, T., Moradi, A., Paar, C., Salmasizadeh, M. & Shalmani, M. T. M. (2008), On the power of power analysis in the real world: A complete break of the keeloq code hopping scheme, *in* 'Annual International Cryptology Conference', Springer, pp. 203–220.

Eurostat (2017), *Energy, transport and environment indicators — 2017 Edition*, Vol. 25.
URL: *http://ec.europa.eu/eurostat/documents/3217494/8435375/KS-DK-17-001-EN-N.pdf*

Garcia, F. D., Oswald, D., Kasper, T. & Pavlidès, P. (2016), Lock it and still lose it-on the (in) security of automotive remote keyless entry systems., *in* 'USENIX Security Symposium'.

Ghudson SSHWC ACSS (2018).
URL: *http://web.mit.edu/ghudson/dev/sshwc/acss.c*

Glez-Peña, D., Lourenço, A., López-Fernández, H., Reboiro-Jato, M. & Fdez-Riverola, F. (2013), 'Web scraping technologies in an api world', *Briefings in bioinformatics* **15**(5), 788–797.

Indesteege, S., Keller, N., Dunkelman, O., Biham, E. & Preneel, B. (2008), A practical attack on keeloq, *in* 'Annual International Conference on the Theory and Applications of Cryptographic Techniques', Springer, pp. 1–18.

Introducing the Windows PowerShell ISE — Microsoft Docs (2018).
URL: *https://docs.microsoft.com/en-us/powershell/scripting/core-powershell/ise/introducing-the-windows-powershell-ise?view=powershell-6*

Luigi Auriemma (2018).
URL: *http://aluigi.altervista.org/mytoolz.htm*

MBC2 Readme (2018).
URL: *ftp://zedz.net/pub/security/cryptography/algorithms/mb/mbc2/mbc2-README.txt*

Obfuscated OpenSSH ACSS (2018).
URL: *https://github.com/brl/obfuscated-openssh/blob/master/acss.c*

pdfkit (2018).
URL: *https://pypi.python.org/pypi/pdfkit*

PowerShell Scripting — Microsoft Docs (2018).
URL: *https://docs.microsoft.com/en-us/powershell/scripting/powershell-scripting?view=powershell-6*

PyCharm: Python IDE for Professional Developers by JetBrains (2018).
URL: *https://www.jetbrains.com/pycharm/*

ReFirmLabs binwalk (2018).
URL: *https://github.com/ReFirmLabs/binwalk/wiki*

Requests: HTTP for Humans — Requests 2.18.4 documentation (2018).
URL: *http://docs.python-requests.org/en/master/*

Schütze, T. (2011), Automotive security: Cryptography for car2x communication, *in* 'Embedded World Conference', Vol. 3.

Selenium - Web Browser Automation (2018).
URL: *https://www.seleniumhq.org/*

Tor Project — Privacy Online (2018).
URL: *https://www.torproject.org/*

Trello (2018).
URL: *https://trello.com/*

Welcome to Python.org (2018).
URL: *https://www.python.org/*

Welcome to Stem! — Stem 1.6.0 documentation (2018).
URL: *https://stem.torproject.org/*

WinRAR archiver, a powerful tool to process RAR and ZIP files (2018).
URL: *https://www.rarlab.com/download.htm*

wkhtmltopdf (2018).
URL: *https://wkhtmltopdf.org/*

Zaddach, J. & Costin, A. (2013), 'Embedded devices security and firmware reverse engineering', *Black-Hat USA* .

Appendix A

Project File Structure

```
Blinda1581951
    git-repository.txt
    report.pdf
├──binwalk-analysing-env
├──binwalk-analysing-script
├──data-collection-env
├──data-collection-script
    └──res
├──extracting-script
├──results
└──signsrch-analysing-script
    └──res
```

- **git-repository.txt** holds the address of the Gitlab repository
- **report.pdf** represents the report of the project
- **binwalk-analysing-env** holds the .yml file for duplicating the Anaconda environment of the binwalk analysing script
- **binwalk-analysing-script** contains the Python script to analyse firmware binaries using binwalk and the PowerShell script to merge the individual reports into one final report.

- **data-collection-env** holds the .yml file for duplicating the Anaconda environment of the data collection script
- **data-collection-script** stores the source code for the Python script that was used to gather the data of this project
- **extracting-script** represents the PowerShell script used to extract the firmware binaries from the archived data and to copy unarchived binaries to the destination directory.
- **results** includes the final reports created by the signsrch analysing script and the binwalk analysing script; the full individual reports are available on the Gitlab of this project.
- **signsrch-analysing-script** holds the PowerShell script used to analyse the individual firmware binaries using signsrch and to create the final signsrch report
- **res** contains, in both cases, different resources necessary for running the software on Windows

Appendix B

Data Collection Script Instructions

In order to run the data collection script, some preparatory steps must be taken. The guide to running the data collection script is also available on the Gitlab repository with links to downloading the required resources. The downloading links are also available in the Bibliography.

1. Install Anaconda (*:: Anaconda Cloud* 2018) and open the Anaconda Prompt
2. Use the data-collection-env.yml file from the datadata-collection-env directory to create the environment: `conda env create -f data-collection-env.yml`
3. Activate the new environment: `activate data-collection-env`
4. Check that the new environment has been installed correctly: `conda list`
5. Go to the data-collection-script directory and start the script: `python main.py`

This guide and all the resources are targeting Windows. Please check the websites of the resources for other operating systems.

Appendix C

binwalk Analysis Script Instructions

In order to run the binwalk analysis script, some preparatory steps must be taken. The guide to running the binwalk analysis script is also available on the Gitlab repository with links to downloading the required resources. The downloading links are also available in the Bibliography.

1. Download binwalk (*ReFirmLabs binwalk* 2018)
2. Install Anaconda (*:: Anaconda Cloud* 2018) and open the Anaconda Prompt
3. Use the binwalk-env.yml file from the binwalk-analysing-env directory to create the environment: `conda env create -f  binwalk-env.yml`
4. An error about binwalk will appear. At this point follow the installation steps from the binwalk developer website.
5. Activate the new environment: `activate binwalk-env`
6. Check that the new environment has been installed correctly: `conda list`
7. Go to the binwalk-analysing-script directory and start the script: `python main.py`