Espen Taftø Vestad, Amar Licina, Abdulfatah Abdi-Salah

Automated dynamic malware analysis of ELF-files

Bachelor's project in Digital Infrastructure and Cyber Security
Supervisor: Ernst Gunnar Gran
May 2021

Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Kunnskap for en bedre verden

Espen Taftø Vestad, Amar Licina, Abdulfatah Abdi-Salah

Automated dynamic malware analysis of ELF-files

Bachelor's project in Digital Infrastructure and Cyber Security
Supervisor: Ernst Gunnar Gran
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Abstract

In today's digital landscape there is need for more information security research. In order to help secure crucial information and digital systems there needs to be reliable tools, frameworks and platforms for information security analysts and experts. As the threat landscape regarding Linux systems becomes greater, the need for accurate malware analysis of ELF-files increases. This was one of the greatest motivators for working with this project. Throughout this thesis project, we have developed a solution which will hopefully contribute towards evolving research on these topics.

Sammendrag

I dagens digitale landskap er det mer nødvendig med økt forskning innen informasjonssikkerhet. For å sikre viktig informasjon i digitale systemer er det essensielt med pålitlige verktøy og rammeverk for informasjonssikkerhets-forskere. Siden trusselbilde for Linux systemer stadig vokser, er det i høyere grad nødvendig med nøyaktige og effektive analyser av potensiell skadevare i ELF-filer. Gjennom dette prosjektet har vi utviklet en løsning som forhåpentligvis vil bidra mot et mer utviklet sikkerhetsmiljø innen Linux-skadevare.

Foreword

This thesis was written by students from the Department of Information Security and Communication Technology at the Norwegian University of Science and Technology. The students were:

- Espen Taftø Vestad
- Amar Licina
- Abdulfatah Abdi-Salah

We wish to extend a thanks to the project supervisor Ernst Gunnar Gran, for providing guidance throughout the project-period, and helping improve the end-product. We also wish to thank the project owner Lasse Øverlier for their cooperation and accommodation by providing the testing material. Thanks to Lars Erik Pedersen for providing access to the NTNU cloud services and enabled nested virtualisation. Finally we would like to thank NTNU security for allowing us to test malware sample.

Contents

Abstract . . . . **iii**
Sammendrag . . . . **v**
Foreword . . . . **vii**
Contents . . . . **ix**
Figures . . . . **xiii**
Tables . . . . **xv**
Code Listings . . . . **xvii**
Acronyms . . . . **xix**
Glossary . . . . **xxi**
1 Introduction **1**
1.1 Project description 1
1.2 Motivation 2
1.3 Target audience 2
1.4 Scope 2
1.4.1 Problem statement 2
1.4.2 Objective and goals 2
1.5 Project Group 3
1.5.1 Previous knowledge 4
1.6 Project process and Thesis layout 4
2 Requirements **7**
2.1 Functional requirements 7
2.2 Operational requirements 7
2.3 External requirements 8
3 Theory and technology **9**
3.1 Malware and reverse engineering 9
3.2 The Executable Linkable Format 9
3.3 Definition 10
3.3.1 Class 11
3.3.2 Data 11
3.3.3 Versions 11
3.3.4 OS/ABI 11
3.3.5 ABI version: 12
3.3.6 Machine 12
3.3.7 Type 12

3.3.8 Program headers and section headers in the ELF-header . . . 12
3.3.9 Static and dynamic binaries . . . 12
3.4 File Data . . . 13
3.4.1 Program Header . . . 14
3.4.2 Section headers . . . 15
3.5 Malware in ELF-files . . . 15
3.6 Malware analysis methods . . . 15
3.6.1 Static analysis . . . 15
3.6.2 Dynamic analysis . . . 16
3.6.3 Memory Analysis . . . 16
3.7 Virtualisation . . . 16
3.8 Obfuscation . . . 17
3.9 Sandboxing . . . 17
3.10 The Limon sandbox . . . 17
3.10.1 Dynamic analysis tools . . . 18
3.10.2 Static analysis tools . . . 19
4 Design . . . 23
4.1 Researching available solutions . . . 23
4.2 Functionality Design . . . 23
4.3 Application design . . . 25
4.4 Architecture design . . . 25
4.5 Network design . . . 26
4.6 Sequence diagram . . . 27
5 Implementation . . . 31
5.1 Methodology . . . 31
5.2 Implementation repository . . . 32
5.3 Infrastructure Configuration . . . 32
5.3.1 Initial configuration . . . 32
5.3.2 Instance deployment . . . 32
5.4 Installing cuckoo . . . 33
5.5 Limon setup . . . 33
5.5.1 Sandbox outline . . . 33
5.5.2 Host OS configuration . . . 34
5.5.3 Guest OS configuration . . . 36
5.5.4 Sandbox network configuration . . . 38
5.5.5 Configuring the Limon script . . . 43
5.6 Scripting the Limon setup . . . 43
5.6.1 Installing Limon from the thesis repository . . . 43
5.7 Limon modifications . . . 45
6 Analysis and testing . . . 47
6.1 Limon usage . . . 47
6.2 Analysis output . . . 47
6.3 Performance . . . 47
6.3.1 Execution performance . . . 48

6.3.2 Functionality testing . . . 48
6.3.3 Successful to unsuccessful sample execution ratio . . . 50
6.4 Examples . . . 54
6.4.1 Tsunami malware execution . . . 54
6.4.2 Rootkit . . . 57
6.4.3 Privilege escalation . . . 58
7 Discussion . . . 61
7.1 Unmet functional requirements . . . 61
7.1.1 Logging of executed code or Assembly instructions . . . 62
7.2 Useful malware indicators in ELF-files . . . 63
7.2.1 Symbols . . . 63
7.2.2 Segments . . . 63
7.2.3 Run-time indicators . . . 64
7.3 Improvements and further work . . . 64
7.3.1 Additions . . . 65
7.3.2 Modifications . . . 66
7.3.3 fixes . . . 68
7.4 The importance of combined analysis . . . 70
7.5 Protecting systems against ELF-malware infection . . . 70
8 Conclusion . . . 71
8.1 Project assessment . . . 71
8.2 Learning outcome and evaluation . . . 72
8.3 Results . . . 72
Bibliography . . . 73
A Project plan . . . 81
B Project agreement . . . 97
C Meeting schedule . . . 103
D Working hours . . . 109
E Developed installation scripts . . . 129
F Thesis project repository . . . 137
G Malware test samples . . . 139

Figures

3.1 Simplified topology of an ELF-file 10
3.2 ELF header 11
3.3 File data illustrated schematically. [61] 13
3.4 The program header of an ELF-file [58] 14
3.5 Types of analysis in Limon 17

4.1 Application interaction diagram 25
4.2 Architecture design 26
4.3 Network design 27
4.4 Sequence diagram 29

5.1 Trello Kanban board 31
5.2 Architecture design 34
5.3 Adding a host-only-network (vmnet10). 39
5.4 Assigning the custom vmnet to the virtual instance. 39
5.5 Confirming that host OS acts as default gateway for the guest OS sandbox 40
5.6 Setting preferred DNS visually. 41
5.7 Confirming changes in preferred DNS. 41
5.8 Adding static IP address to the sandbox instance. 41
5.9 Choosing network services to be simulated by INetSim. 42
5.10 Network design 44

6.1 Reasons regarding failed execution of samples during performance test. 51
6.2 Mole scanner sample terminated because of missing arguments. . . 52
6.3 Mole scanner sample executed successfully passing required arguments when running Limon. 52
6.4 Readelf reading past end of file for section headers. 53
6.5 Stripped ELF-binary without call trace output. 53
6.6 Tsunami VirusTotal detection 55
6.7 Possible file creation of */tmp/cron*, string comparisons and memory allocation. 55

6.8 Possible tampering with **crontab** to schedule downloads from malicious domain. 56
6.9 Deletion of previously created file. 56
6.10 Observed credentials are written to disk. 56
6.11 SMTP network activity logged by inetsim. 56
6.12 Inspecting network activities in wireshark output. 57
6.13 RootKit VirusTotal detection 57
6.14 Options for Root Kit 58
6.15 The FucKit RK script repeated itself 176 times 58
6.16 The figure shows the VirusTotal output from the static analysis . . . 59
6.17 malloc function example from call trace 59

Tables

3.1 Features provided by Limon, according to default configuration. . . 21

4.1 Tools and functionalities planned for implementation. 24

6.1 Features used by Limon. 48
6.2 Description of output-files and which tools generate them. 49

7.1 Representation of whether functional requirements has been covered. 61

Code Listings

5.1 Installing VMware on Linux 35
5.2 Reinstallation of Strace 37
5.3 Library packages for executing 32bit binaries on 64bit architectures 37
5.4 Adding *malware_analysis* directory to */etc/environment* 38
5.5 Assigning appropriate variables for Limon in *conf.py* 42
6.1 Simple Python script for running Limon with several malware samples. 50
6.2 Running a malware sample along with required arguments in Limon. 52
7.1 Pseudo-code for automatic unpacking solution for Limon in *statan.py* 65
E.1 Limon installation script for configuring the host machine. 134
E.2 Limon installation script for configuring the guest machine. 135

Acronyms

ABI Application Binary Interface. 11, 12

API Application Programming Interface. 8, 20, 21, 23, 25, 66

CLI Command line interface. 18, 36, 50

DHCP Dynamic Host Configuration Protocol. 41

DNS Domain Name System. 40

ELF Executable Linkable Format. v, xxiii, 1–4, 7–15, 17, 20, 25, 28, 37, 47, 50, 53, 54, 61, 63, 64, 70–72

GUI Graphical user interface. 18, 35, 38, 66, 70

I/O Input/Output. 18, 61, 72

NTNU Norwegian University of Science and Technology. 1, 4, 32, 71

OS operating system. 1, 2, 10, 11, 19, 24, 32, 33, 53, 67

RAM random access memory. 67

SMTP Simple Mail Transfer Protocol. 64

SOC Security Operations Center. 2, 66

VM virtual machine. 16, 18, 19, 32, 33, 35, 38, 41, 44, 67

Glossary

Assembly A low level programming language used by the processor. Assembly is also used to compile higher level programming languages [1]. 7, 54, 61, 62, 66

client A person or organization which hands out a project or task to recipients. 33, 36

Command and Control Server A server which is controlled by an attacker that is then used to send commands to a system in order to perform actions such as stealing information, control botnets and conduct DDoS attacks [2].. 18, 24, 52, 64

Cuckoo An open source software for automating malware analysis [3]. 23, 33

debug A process of identifying a problem and removing those errors [4]. 14, 16

dynamic analysis The process of analyzing a program by running the program on a real or virtual environment. 1, 3, 15–18, 24, 25, 33, 50, 55, 57, 59, 62, 63, 66–72

egress A point which allows a framework to communicate outside via the egress point. 32

embedded system A system comprised of both computer software and hardware which has a specific purpose [5]. 1

exploit A program or code which is used to take advantage of a vulnerability in a system or application [6]. 52

firewall A security device which monitors traffic and allows or denies access based on security rules [7]. 32

fuzzy hash A compression method which compares the similarities in digital files [8]. 20, 21, 24, 36

guest OS An operating system installed on a exsisting operating system by using for example a virtual machine [9]. xiii, 17, 32, 33, 35–38, 40, 43, 44, 68

hash A hash function converts the value of input into a hash value which can only be decoded by comparing input with values in the hash table [10]. 20, 21

Hexdump Hexdump is an utility that displays the content of a binary file in hexadecimal, octal, decimal or ASCII. The utility for inspection comes in good use for data recovery, programming or reverse engineering [11]. 48

host OS The operating system which communicates with the underlying hardware [12]. xiii, 19, 25, 32–38, 40, 41, 43, 44, 65

hypervisor A virtual machine monitor. Software used to host virtual machines [13]. 16, 19, 25, 32, 35

indicator of compromise The traces left by a potential attacker that are uncovered by a forensic analysis [14]. 15, 16, 20

InetSim A tool used to simulate network services commonly used by malware [**inetstim**]. 18, 21, 24, 25, 27, 28, 36, 38, 41, 49, 56

ingress A point of access which grants access to services outside of a framework. 32

Kanban Kanban is a lean work management method which uses a Kanban board in order to organizes tasks. 31, 72

LDD A tool used to print the shared objects/libraries of a file [15]. 20, 21, 24, 35

Limon A sandbox used to analyze and report the run time indications of Linux malware. [16]. xvii, xxiii, xxiv, 17–20, 23–26, 28, 32, 33, 35–37, 40, 42, 43, 45, 47, 48, 50, 51, 54, 55, 62, 63, 65–72, 137

Ltrace Troubleshooting software which shows calls to shared libraries [17]. 19, 25, 28, 38, 47–49, 53–56, 59, 66, 67, 69

Lubuntu A lightweight Linux distribution based on ubuntu [18]. 35

malware Intrusive software which aims to disrupt, destroy or steal data from a target [19]. 1–5, 8, 9, 15–21, 23–25, 33, 35–38, 45, 47–56, 61–72

malware sample An instance of malicious code which can be used for analysis in a secure environment. vii

memory analysis Analysis of the volatile memory in a computers memory dump [20]. 3, 15–17, 24, 33, 36, 37, 69, 70

nested virtualisation A complex process that involves running a virtual machine within a virtual machine [21]. vii, 16, 25

open source Code which is accessible for all to use, edit or enhance. [22]. 17, 18, 20, 33, 36, 43

Open Stack An open source platform which provides cloud computing infrastructures [23]. 25–27, 32

packer Method used to hide malware and make them appear as new by using run time encryption [24]. 7, 20, 35, 54, 62, 65, 66

Phishing A social engineering attack with often aims to steal user data or other information by masquerading as a trusted entity to gain the victims trust[**Phishing**]. 70

Pillow A python imaging library [25]. 19, 21, 24

ransomware Ransomware is an attack which encrypts the files on a device rendering them unusable. The attacker then demands payment in order to decrypt the encrypted files [26]. 1

Readelf A tool for displaying the information of one or more ELF-files [27]. 11, 20, 21, 24, 35, 53, 63

Remnux A reverse engineering toolkit used on Linux to analyze malware [28]. 18, 23, 36

sandbox A safe isolated environment where code can be run and analyzed [29]. xiii, 2–4, 8, 17, 21, 23, 24, 33, 36–38, 40, 45, 47–50, 61, 67, 68, 71, 72

security rule Firewall settings for allowing or denying traffic from a network. 32

snapshot Storing the state of a machine at a certain point in time in order to return to it if an error occurs [30]. 40, 45, 67

Ssdeep A tool used for fuzzy hashing [31]. 20, 21, 24, 36, 49

SSH A protocol used to secure access the command-line on another machines [32]. 32

static analysis Analysis method which analyses the source code and tests it for vulnerabilities [33]. xiv, 3, 8, 15–17, 19, 20, 24, 25, 33, 35, 47, 53, 59, 62

Strace Tool used for tracing system calls. This tool will be used by Limon in order to trace system calls made by ELF-file [34]. xvii, 19, 21, 24, 25, 28, 37, 38, 47–49, 53, 62, 66, 67, 69

Strings A tool which finds and prints embedded strings in binary files [35]. 20, 49

Sysdig A tool used for event monitoring and run time threat detection [36]. 21, 24, 44, 45, 69

TCPdump A command-line packet analyzer used by Limon in order to analyze network traffic [37]. 18, 21, 24, 25, 27, 28, 35, 49

threat actor An actor or a group who pose a threat to the assets [38]. 1, 62, 64, 68

threat intelligence Information used to understand threats that have, will or are currently at place [39]. 2

tshark A network protocol analyser that allows one to collect data from a live network or read packets from a formerly saved capture files. [40]. 21, 24

virus Type of a malicious code or program coded with an intent to alter a device, or spread to another by inserting or attaching itself to a legitimate programs [41]. 9

VirusTotal An online scanner engine that uses many different antivirus scans which the users own antivirus scan may have missed [42] . xiv, 20, 21, 24–26, 54, 57–59

VMrun A command that can be used to for example create files, delete files, create directories in virtual environments [43]. 38, 44

VMware Workstation Workstation is a virtualization software developed by VMware [44]. 17, 19, 25–28, 32, 33, 35, 38, 40, 41, 44, 45

Volatility A memory analysis and forensics tool used to analyze memory dumps [20]. 36, 37, 44, 69

wireshark Network protocol analyser that allows an analyst to see what is happening on a network at microscopic level[45]. 18, 56

Yara A tool used by Limon to detect packers and the capabilities of malware using Yara rules [46]. xxiv, 20, 21, 24, 35, 54, 62, 65

Chapter 1

Introduction

Malware in computer systems are constantly evolving. New pieces of malicious code and applications are being distributed to computer systems around the world frequently. Sophisticated threat actors are continuously developing new malware methods to harm, abuse or control today's modern systems.
Linux is widely used as the operating system in servers and cloud infrastructure world wide. In the past years, there has been an increase in malware campaigns targeting Linux systems. Examples include the famous ransomware "RansomExx" [47] and the potentially Chinese state sponsored "RedXOR" malware [48]. At the same time, Linux and embedded systems are widely dependent on ELF-files, which is short for *Executable Linkable Format*.

Based on this increase, dynamic analysis techniques to identify behaviour of ELF-malware will be relevant for the coming years. Through this thesis, methods and technologies for dynamically analysing malware in ELF-files will be investigated and implemented.

1.1 Project description

Considering the fact that malware in ELF-files now are an increasing threat, project client Lasse Øverlier at NTNU in Gjøvik, has provided the group with the task of exploring how to dynamically analyse these binary files. Throughout the project period, it is desired that a secure sandbox environment is implemented. This environment should take advantage of one or several methods of performing dynamic malware analysis that provides useful output regarding how ELF-files behaves during execution. Information regarding library calls, network activity, disc activity and logging of executed code are desired in the output that the analysis platform should produce. Discussing other parameters that might contribute to identify ELF-malware are also relevant for the thesis.

1.2 Motivation

This thesis may be of interest for those with attentiveness for topics such as malware analysis, cybercrime and criminality in general. Working towards a more secure everyday is something that motivates many to pursue a career in information security. Being capable of analyzing malware, creates the feeling of contribution towards an important community, by bringing cybercrime to a more manageable level. People with an interest in applying technical skills related to information security may also find this thesis interesting as it also touches other topics such as threat intelligence, Python programming and Bash scripting.

1.3 Target audience

The target audience for this report is mainly people that are interested in dynamically analysing Linux binary files for possible malware, for instance security analysts/researchers, SOC-operators and threat hunters. The reader should have some previous knowledge about information security and the Linux operating system in order to fully comprehend the project report.

1.4 Scope

This thesis project has implemented a secure sandbox environment in order to automatically run and execute malicious ELF-files, returning relevant output for further analysis. Which output is relevant is determined by the project client, Lasse Øverlier, and is further specified as a part of the problem statement. To determine the scope of this thesis, the research question along with appropriate goals, objectives and delimitations will be defined through this chapter.

1.4.1 Problem statement

Based on the task description along with the guidance given by the project supervisor and clarifications given by the client, the following problem statement has been developed:

The goal of the project is to establish a secure sandbox environment which executes and analyses ELF-files dynamically, producing behaviour reports for further analysis. The project will also explore what parameters are relevant in order to identify ELF-malware, and behavioural reports should include data regarding performed library calls, disk access, network activity, and code logging.

1.4.2 Objective and goals

This section defines the goals in which the project aims to achieve through this thesis, both on a short and long term basis.

Effect goals

The effect goals describe the implementation's anticipated long term impact, along with potential for desired changes from how things currently operate.

- Make ELF-file analysis more effective for security analysts.
- Implement a method which, in the long run, may provide indications of potential malware contained in ELF-files for the target audience.

Achievement goals

Achievement goals refers to objectives to be achieved during the thesis project period.

- Implement an automated method for dynamic malware analysis of ELF-files.
- Explore different sandbox technologies which might be used for dynamic analysis of ELF-files, and study how this can be implemented in a secure way.
- Discuss useful parameters that might be used to classify an ELF-file as malicious.

Delimitations

Delimitations describe the focus area and boundaries of the project based on the requirements provided by the client, Lasse Øverlier, in order to make an accurate and complete solution.

- Methods used to perform dynamic analysis of ELF-files might be commercial, preexisting, or custom methods developed specifically for the thesis.
- If commercial methods are available, their functionality might be described rather than being implemented in the environment.
- The project focuses mainly on dynamic analysis methods, whereas static- or memory analysis methods are not considered. However, some static analysis methods might still be relevant for the final result.
- The final project solution will not determine whether analysed ELF-samples contains malware. The user only receives information about what the ELF-binary performs during execution, providing indications on whether the file might be malicious or not.

1.5 Project Group

The thesis participants has discussed which roles are appropriate and necessary to have for the project. The list below contains the roles for the project:

- Espen Taftø Vestad: Group-leader, Contact person, and timekeeper.
- Amar Licina: Second group-leader, Facilitator and Secretary.

- Abdifatah Abdi-salah: Secretary.

1.5.1 Previous knowledge

All thesis participants have experience regarding information security, programming and network. They have taken several relevant courses at NTNU and have worked with related activities in their spare time. These courses range from artificial intelligence and algorithmic methods to ethical hacking and reverse engineering courses. The list below illustrates all the courses the thesis participants have taken while studying at NTNU.

- IMT1031 - Fundamental Programming
- IMT1003 - Introduction to IT-Operations and Information Security
- REA1101 - Mathematics for computer science
- IMT2006 - Computer Networks
- IMT2243 - Software Engineering
- IMT1082 - Object-oriented Programming
- IMT2007 - Network Security
- IMT2571 - Data Modelling and Database Systems
- IMT2021 - Algorithmic Methods
- IMT2008 - ITSM, Security and Risk Management
- IMT2282 - Operating Systems
- IMT3003 - Service Architecture Operations
- IMT3673 - Mobile/Wearable Programming
- IMT3004 - Incident Response, Ethical Hacking and Forensics
- IMT3005 - Infrastructure as Code
- IMT3104 - Artificial Intelligence
- IMT4116 - Reverse Engineering and Malware Analysis
- IMT2291 - Web Technology
- IMT3501 - Software Security

1.6 Project process and Thesis layout

The reader will be approached with theories that intend to solve the problem statement, followed by the design and implementation carried out to achieve the final results. Initially, the project consist of a requirement specification, followed by a theory chapter explaining various concepts and technologies used throughout the thesis.

A secure sandbox environment for analysing potential malware in ELF-files has been designed and implemented according to requirement specification. Tests of real-world malware samples have also been conducted to verify that the solution meets the requirements set by the project description.
This section briefly describes how the thesis report is structured for the reader's simplicity.

Introduction

Chapter 1 gives the reader an overall overview of the thesis, introducing the purpose and goals of the project.

Requirements

Chapter 2 specifies requirements regarding the thesis implementation. This includes functional, operational and external requirements.

Theory and technology

Chapter 3 will explain the theory behind the different aspects and technologies used in this thesis. This includes aspects relevant for malware analysis and descriptions of the different tools used to implement the solution.

Design

Taking into consideration the specified requirements from chapter 2, chapter 4 covers how the solution has been designed for further implementation.

Implementation

Chapter 5 covers the technical implementation of the solutions designed in chapter 4.

Analysis and testing

Chapter 6 covers testing the implementation described in chapter 5, as well as describing achieved results during this thesis.

Discussion

Chapter 7 further discusses the findings and results from chapter 6, along with describing measures of improvements and further work.

Conclusion

Chapter 8 will provide a short overview of the thesis project, describing how things were carried out, learning outcome and things to consider in the future.

Chapter 2

Requirements

This chapter describes the functional, operational and external requirements which need to be met in order to complete the project and achieve desired results.

2.1 Functional requirements

The functional requirements describes the different functionalities that the solution needs to provide. The project description presented desired functionalities in order to solve the task, which are further discussed throughout this section.

The framework is created with the target audience in mind, the information security analysts which needs to test ELF-files. It is important that the user of this framework has experience in information security analysis in order to use the output that is returned.

The main functional requirements for the framework are:

- Returning information regarding how ELF-files behaves during execution. More specifically information regarding network access, disk-access, library calls, detection of potential packers, and logging of executed code or Assembly instructions.
- Returning output in human-readable format.
- Return output to the command line as well as to file.
- Creating an isolated environment which securely and stealthy allows for execution of ELF-malware.

2.2 Operational requirements

Operational requirements refers to requirements which must be met in order to run the implementation. The project description does not specify any operational requirements, but there has been certain measures taken in order to make the analysis process as seamless as possible.

- The framework needs to be developed for Linux platforms considering the ELF-file format.
- The sandbox must support both x86 and x86-64 architectures to run different types of malware samples.

2.3 External requirements

The external requirements describe the requirements which appear outside of the system framework. The environment depends on external resources for fetching malware samples in order to conduct accurate system tests. Network access is also necessary in order to connect to the API used by certain tools in the framework, which is done during the static analysis portion of the analysis process.

Chapter 3

Theory and technology

This chapter will describe different theoretical concepts and technologies relevant for this thesis. The overall purpose of this chapter is to prepare the reader for the coming chapters by discussing fundamental aspects of importance.

3.1 Malware and reverse engineering

Malware or malicious software is a huge threat to everyone, and are mainly responsible for most computer intrusions and incidents [49]. A malware identifies as something that has ability harm or damage a computer or a network. Malware is usually identified as type of malicious software, regardless of how it works, how it's distributed or it's intent. Examples of malware types include Trojan Horses [50], rootkits [51], scareware [52], spyware [53], and worms [54]. A virus is a specific type of malware. A computer virus is designed to copy itself and spread to other devices whenever it gets the chance [41]. Reverse engineering is all about disassembling and breaking down a binary file to investigate how it's built. In the case of a malware sample, reverse engineering might help identifying the programs intent and how it works [55].

3.2 The Executable Linkable Format

The following section addresses theoretical aspects regarding the Executable Linkable Format (ELF), including definitions and the file structure. While an in-depth understanding of this format is out of scope for this thesis implementation, the reader is encouraged to posses some information on the subject.
as seen in figure 3.1 this is a simplified version of the structure of an ELF-file. An ELF-file consist of an ELF-header and file data. File data is composed of program headers, section headers and data.

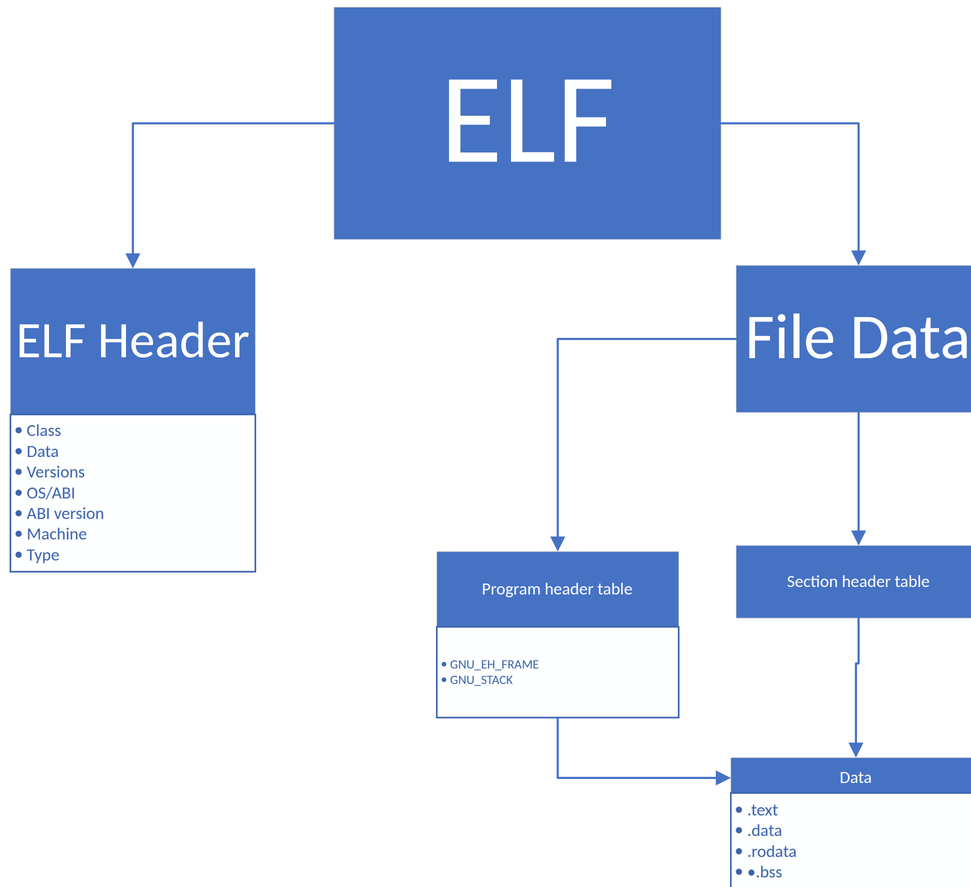


Figure 3.1: Simplified topology of an ELF-file

3.3 Definition

ELF stands for Executable Linkable Format. The ELF-file defines structures for binaries, libraries and core files. It is used for executable files, relocatable object files, shared libraries, and core dumps. Many operating systems today are heavily dependant on ELF-files, as for example Linux, Solaris/Illumos, Android. The file format is also used within several game consoles, such as PlayStation portable, Dreamcast and Wii [56]. The structure of an ELF-file consist of the *ELF-header* and *file data*. These structure components is further described throughout this chapter.

The ELF-header, as shown in figure 3.2, is 32 bytes long and provides file information. The header starts with a sequence of four unique bytes which as you can see above, translates to E, L, F. With the prefixed 7f value. [57]

- 0x45 = E
- 0x4c = L
- 0x46 = F

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x80483e0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          3532 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         6
  Size of section headers:           40 (bytes)
  Number of section headers:         26
  Section header string table index: 25
```

Figure 3.2: ELF header

3.3.1 Class

Class determines if the architecture of the ELF-file is either 32-bit(=01) or 64-bit(=02). (=01) and (=02) are translated by the Readelf command as either *ELF32* or *ELF64*. As seen in figure 3.2, this file is a 32-bit file(=01)

3.3.2 Data

The Data field can be two different options. 01 stands for LSB (Least Significant Bit), which is refers to as Little-endian [58]. The other possible option is 02 which defines MSB (Significant Bit) which refers to as big-endian [58].

3.3.3 Versions

The version field provides us with which version number the ELF-file has. There are only two possible versions numbers: *Current* and *None*. These values are displayed as "1" or "0" in the ELF-header. "1" translates to *current*, while "0" translates to *none*. Figure 3.2 displays an ELF-file with version set as *current*. [57].

3.3.4 OS/ABI

Every OS may come across overlaps in terms of duplicate functions, hence some functions are identical and some has minor differences [56]. The definition of the relevant set is done with an Application Binary Interface (ABI). This mitigates the chance for overlaps, and supports the OS ABI to know how functions are forwarded [59].

3.3.5 ABI version:

This section can provide information regarding which version of the ABI is specified for the file.

3.3.6 Machine

This field shows which expected machine type (CPU-architecture) and specifies what CPU-architecture is required running the ELF-file. Figure 3.2 specifies "Intel 80386" for this particular file [60].

3.3.7 Type

This field identifies what object type the file has [57]. Examples of object types include:

- REL Relocatable file (value 1)
- Executable file (value 2)
- Shared object file (value 3)
- Core file (value 4)

3.3.8 Program headers and section headers in the ELF-header

An ELF-file can consist of multiple program headers and section headers [61]. The list below describes other important fields within the ELF-header, which are also displayed in figure 3.2

- Number of program headers: Identifies how many program headers there is in the ELF-file.
- Number of section headers: Identifies how many section headers there is in the ELF-file.
- Start of program headers: Identifies the start of the program headers with bytes into the ELF-file.
- Start of section headers: Identifies the start of the section headers with bytes into the ELF-file.
- size of section headers: Identifies the size of the section headers that is in the ELF-file.
- size of program headers: Identifies the size of the program headers that is in the ELF-file.

3.3.9 Static and dynamic binaries

There are two types of ELF binaries: Either static or dynamic, which refers to its respectable library. [58]

- Dynamic binaries: Needs external components to be executed correctly, and commonly contains functions such as creating network socket, or opening

files etc.
- Static binaries: Has all libraries included within the file.

3.4 File Data

As seen in figure 3.3. The file data area consist of three parts, as seen in figure 3.3.

- Program Headers: Or Segments (describes zero, or more segments)
- Section Headers: Or Sections (describes zero, or more sections)
- Data: (referred to by entries in the program header, or section header table)

Every segment contains information that is important for the file's run-time execution, while sections contain important data for linking and relocation. To get a better understand of how an ELF-file structure looks like, figure 3.3 represents everything that has been discussed.

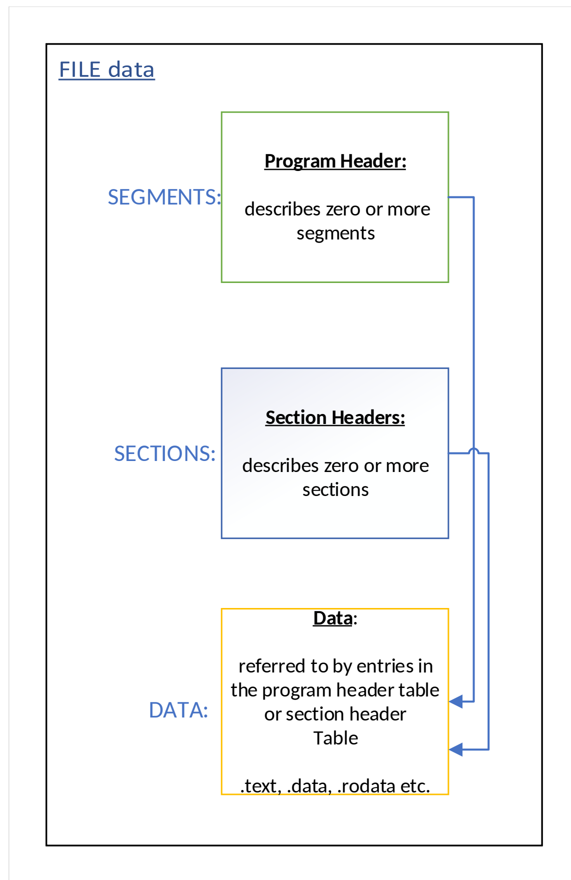


Figure 3.3: File data illustrated schematically. [61]

3.4.1 Program Header

An ELF-file can consist of zero or more segments, and it shows the segments which are used in run-time execution to create a process/memory image. The kernel can access the segments and map them into a virtual address space by using mmap system calls [58]. It converts predefined instructions to a memory image. An ELF-file needs a program header table in order to be executed if it is a normal binary, if not it will not run. These headers are used along with its underlying data structures to create a process.

```
Elf file type is EXEC (Executable file)
Entry point 0x402ba8
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x00000000000001f8 0x00000000000001f8  R E    8
  INTERP         0x0000000000000238 0x0000000000400238 0x0000000000400238
                 0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x0000000000015144 0x0000000000015144  R E    200000
  LOAD           0x0000000000015e00 0x0000000000615e00 0x0000000000615e00
                 0x00000000000005f8 0x00000000000214e8  RW     200000
  DYNAMIC        0x0000000000015e18 0x0000000000615e18 0x0000000000615e18
                 0x00000000000001e0 0x00000000000001e0  RW     8
  NOTE           0x0000000000000254 0x0000000000400254 0x0000000000400254
                 0x0000000000000044 0x0000000000000044  R      4
  GNU_EH_FRAME   0x0000000000012c84 0x0000000000412c84 0x0000000000412c84
                 0x000000000000071c 0x000000000000071c  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
  GNU_RELRO      0x0000000000015e00 0x0000000000615e00 0x0000000000615e00
                 0x0000000000000200 0x0000000000000200  R      1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version
.gnu.version_r .rela.dyn .rela.plt .init .plt .text .fini .rodata .eh_frame_hdr .eh_frame
   03     .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
   08     .init_array .fini_array .jcr .dynamic .got
```
linux-audit.com

Figure 3.4: The program header of an ELF-file [58]

As seen in figure 3.4, there are 9 program headers within the ELF.

- **GNU_EH_FRAME**: Within a segment there is a GNU_EH_FRAME, as seen in figure 3.4. GNU_EH_FRAME shows how the segment uses the GNU C compiler (gcc) as sorted queues to sort exception handlers. If something were to go wrong, the debug information can be displayed here [58].
- **GNU_STACK**: There is also a GNU_STACK within a segment. This field stores information about the stacks. The stack is a buffer where the items are stored, such as variables. The sorting method that is used is LIFO (Last In, First Out). Stacks should not be executable, therefor this may introduce security vulnerabilities by manipulation of memory [58].

3.4.2 Section headers

The section headers define all the sections within an ELF-file. In the section header the data is linked and relocated. The section header table describes zero or more sections that are followed by data which are referred to by entries from the program header table, or section header table [56]. The following table illustrates the four main sections in the section header table:

- .text contains executable code, which will be packed into a segment with read and execute access rights. Which is only loaded once, as the contents will not change.
- .data: Initialized data with read/write access rights
- .rodata: Initialized data with read access rights only
- .bss: initialized data with read/write access rights

3.5 Malware in ELF-files

ELF-malware are ELF-files which contain code that serves a malicious purpose. Infected ELF-files or processes might in some cases behave abnormally, having contents that the victim cannot detect nor see.
Most ELF-malware are based on the "Silvio Cesare File Virus"[62]. Silvio Cesare is an Australian security researcher known for his work with ELF-virus for UNIX-like operating systems. [63]
ELF-malware can be sorted into two categories: [62]

- First: where a malicious code can attach itself to the start of Innocent executable.
- Second: where a malicious code can injects itself into text or data segment of innocent executable.

3.6 Malware analysis methods

This section describes different methods for analysing malware samples. Three analysis methods will be covered here: Static analysis, dynamic analysis and memory analysis. The reader is encouraged to possess knowledge regarding all of these methods, especially dynamic analysis since this concept is crucial for this thesis.

3.6.1 Static analysis

Static analysis is performed in a non-runtime environment, which involves statically analysing software without execution the program. This is done through examining the source code, byte code and application binary for indicators of compromise. This is most easily achieved by using different static analysis tools. When statically analysing a binary file, the internal structure of the file, such as instructions, addressing, is checked rather than observing the behaviour by running

the program. [55]

3.6.2 Dynamic analysis

Unlinke static analysis, dynamic analysis involves executing the binary file and examining its behavior in a run-time environment. Dynamically analysing the malware allows the analyst to debug and observe the malware's behaviour during execution while examining the impact on the different system components and network. An analyst can debug the process while it is running to examine the malware in a running state for observing potential outcomes, getting a better understanding regarding the intentions of the malware. [55]

3.6.3 Memory Analysis

Memory analysis may be referred to as *memory forensics*. Memory analysis is conducted in order to investigate whether malware exists within the computers memory after being compromised. Volatile data captured from a computers memory dump is analysed in order to find or identify most malicious behaviors which are hard to detected on the computer's storage device. Volatile data is referred to as temporary memory stored on a computer at run-time. Once the computer is shut down, the volatile data is gone. Examples of volatile data includes chat messages, clipboard contents and running processes [64].

3.7 Virtualisation

Virtualization is the process of creating a virtual instance or environment which is separate from the hardware on a physical machine. The VM is run by the hypervisor which creates and monitors the VM [13]. A virtual machine provides the same functionalities as a physical machine, and will have its own systems and programs detached from the host machine. The virtual machine will also have a virtual network interface and limited access to the host machine's CPU. Virtual machines usually have a set of functionalities which can be accessed by the host machine in order to manipulate, recover or modify the virtual machine. Using virtual environments reduces the amount of equipment needed since a single piece of infrastructure can run multiple virtual instances.

These are some of the typical functionalities of a VM:

- Snapshots: Saving the current state of a virtual machine in order to return to said state if it becomes necessary. [30]
- Migration: The process of moving a virtual machine instance or snapshot from one physical machine to another is known as migration. [65]
- Nested virtualization: Running a virtual machine within another virtual machine is known as nested virtualisation. [21]

3.8 Obfuscation

Obfuscation is a programming technique used to intentionally obscure code in order to make reverse engineering more difficult, and to make code unclear for anyone except the programmer. Reverse engineering techniques rely on the clarity of the code when copying a program. There are certain methods available that make it possible to analyze obfuscated code, one of these methods being slicing. Slicing is a method used to simplify obfuscated code in order to make it comprehensible, and makes the functionalities of a program easier to find. [66]

3.9 Sandboxing

A sandbox is an isolated testing environment for malware analysts. This environment allows an analyst to run and execute suspicious files without the risk of harming the application, system, network, or underlying platform. By using virtualisation software, the sandbox can revert back to a clean state for each analysis in order to understand the malware's purpose while the malware is running or after it has been run. This is also done in order to avoid alerting the creator of the malware since the malware is being tested in an isolated environment without direct internet access. [29]

3.10 The Limon sandbox

Limon is a sandbox solution designed to analyse ELF-files for potential malware before (static analysis), during (dynamic analysis), and after (memory analysis) execution, as seen in figure 3.5 [16].

Figure 3.5: Types of analysis in Limon

In fact, Limon itself is a script [67] that utilizes other open source malware analysis tools, and automates the process. It was created by Monnappa K A using Python and was presented on Black Hat 2015 [68]. The concept has received little attention since it's presentation. The Limon script is installed and configured on a Linux host machine. The host machine runs a VMware Workstation guest machine, also known as the sandbox. Please be advised that through this thesis, the sandbox will also be referred to as the analysis environment and the guest OS. Table 3.1

describes the different functionalities and tools used by Limon, along with their role and purpose in the analysis phase. Note that this thesis will not fully implement all of the functionalities used by Limon. Only the most relevant features for solving the problem description will be implemented. This is further discussed in chapter 4 *Design*.

3.10.1 Dynamic analysis tools

Limon relies on several open source tools to conduct malware analysis. This subsection will shortly describe some of the dynamic analysis tools which Limon is dependent on. Only tools relevant for this thesis implementation is discussed below.

INetSim and Remnux

InetSim is a malware analysis tool used to simulate common network services in a lab environment [69]. The software makes it possible to analyse network behaviour of malware samples in environments without a network connection. This is useful in order for analyst to remain stealthy when analysing malware which relies on external resources, such as Command and Control Server servers for instance.

While InetSim is a standalone tool, Remnux is a reverse engineering toolkit consisting of multiple tools used for malware analysis [28]. There are several ways to setup Remnux. It can be installed as a virtual machine or added to an existing Ubuntu system. If preferred, Remnux also offers containers which the analysis tools are able to run in. InetSim is included in the Remnux distribution by default, and thus installing Remnux are considered an alternative for installing InetSim if desired.

TCPdump

TCPdump is a CLI packet analyser which displays network packets received or transmitted over a network interface in the computer [37]. It utilizes a library called *libpcap* [37] to capture network packets and dump the results to a *pcap-file*. The *pcap-file* can be opened in a packet analyser software GUI, for instance wireshark, for further analysis of captured network traffic. In order to capture network packets residing from the Limon analysis environment, TCPdump sniffs traffic on the virtual network adapter of the VM. This is further described in chapter 5, *Implementation*.

Strace

One of the functional requirements defined in chapter 2, *Requirements*, describes that the system should be capable of recording the malware's I/O activity. Execut-

ing a malware sample using Strace, enables tracing of system calls carried out by the sample. A system call is described as *a way for programs to interact with the operating system* [70]. It is performed when a program makes a request to the operating system kernel. System call examples includes operations such as *write()* (input), *read()* (output), and *wait()* (sleep for a given amount of time).

Strace itself is a tool for recording such kernel interactions, and is widely used by system administrators and trouble-shooters to find problems in their programs [34]. Malware analysts can benefit from Strace as it provides useful information regarding which system calls are carried out during execution on a low level. When executing a malware sample with Strace using Limon, output reports regarding executed system calls are generated for further analysis.

Ltrace

While Strace records system calls, Ltrace is another debugging tool used to trace and record dynamic library calls carried out by a program [17]. These are more high-level function calls from shared libraries. Ltrace is also able to record system calls such as Strace, in addition to library calls. However, system call tracing in Ltrace is not as accurate as it is in Strace.

VMware Workstation

VMware Workstation is a host hypervisor for creating and running virtual machines [44]. In short, virtual machines are virtual computer instances running on a physical host machine. The host OS are able to communicate with the virtual machines managed by the hypervisor, also known as the guests. Running a virtual machine inside another virtual machine, known as *nested virtualisation*, is also possible. This usually reduces the performance of the guest machine [21], and are not compatible for all types of system or hardware.

Pillow

Pillow is a Python imaging library which is capable of capturing screen images, whereas several image file formats are supported [25]. Limon uses Pillow for capturing a desktop screenshot in the analysis environment desktop before execution of the running malware sample has ended. Although most malwares probably runs stealthy in the background (not visually), there are cases where samples created additional files on the desktop or leaves terminal windows open.

3.10.2 Static analysis tools

The following subsection describes the different static analysis tools used by Limon to analyse malware samples without execution. Every static analysis tool

mentioned in the Limon documentation [16] are relevant for this thesis implementation.

Yara

Yara is a open source tool for malware analysts to identify and/or classify malware samples [46]. The tool utilizes patterns, rules and expressions from common malwares, statically comparing these to the sample of choice. Limon uses Yara rules to determine the category of the malware and whether the sample has been packed with a packer.

Virustotal API

VirusTotal is a threat intelligence platform used to share and search suspicious files and domains [42]. One of its most useful features, is the ability to display suspicious detections from other security firms based on the file's hash. Using VirusTotal's public API, Limon is able to fetch these detections through the terminal by automatically submitting the hash value of a malware sample.

Ssdeep

Ssdeep is a program for computing *context triggered piecewise hashes* (CTPH) [31], also called fuzzy hashes. The program utilizes this to compare similarities in malware samples. For each analysis, Limon will use Ssdeep to compare the current sample to previously analysed samples. Data regarding previous samples are stored in a Ssdeep master file.

Strings

Strings is a program included by default in Linux distributions, determining the contents of and extracting text from binary files [35]. The tool is useful for malware analyst to quickly identify possible indicator of compromise. Limon always initiates an analysis by running Strings on the sample.

ldd

LDD, short for *List Dynamic Dependencies*, is a utility for printing shared library dependencies for each program or shared library specified [15]. The output of LDD is part of the static analysis results when conducting an analysis with Limon.

Readelf

Readelf display information about ELF-files and their header [27]. Figure 3.2 shows the output of running Readelf on an ELF-file. Limon always outputs the header information of an ELF-file using Readelf at the start of an analysis.

Functionality	Description	Tool
System call tracing	Records low level system calls carried out by the malware	Strace, Sysdig
Network simulation	Simulates various network services to replicate a real world scenario in a secure way	InetSim
Network packet sniffing	Records network activity in the analysis environment by sniffing traffic on the virtual network interface	TCPdump, tshark
Analyse memory	Captures and analyses memory image after malware execution.	Volatility
Fuzzy hash	Used to determined whether two inputs (malware samples) are similar, rather than identical [8]	Ssdeep
Malware engine detections	Reports malware detection by other engines through open sources based on the sample's hash value.	VirusTotal public API.
Print hexdump	Prints hexdump as part of the call trace	Strace
Extract strings	Dumping the binary's strings to a txt-file	Strings
Detections of packers and capabilities	Benefits from Yara-rules in order to detect malware behaviour and possible use of packers.	Yara.
Print ELF-header information	Displays in-depth static information regarding the header of the ELF-file	Readelf.
Printing shared library dependencies	Displays the shared libraries that the malware sample requires	LDD.
Internet mode	Connects the sandbox to the internet, allowing the malware sample to communicate with external resources	N/A.
Capture screenshot	Captures a desktop screenshot of the analysis environment upon ended analysis	Python Pillow.

Table 3.1: Features provided by Limon, according to default configuration.

Chapter 4

Design

This chapter outlines how the entire framework for the project is designed. A description of the tools used as well as the network infrastructure, application design, and an overview of the entire system with a sequence diagram will be provided in this chapter.

4.1 Researching available solutions

In order to meet the desired requirements for this thesis, a design phase has been carried out before conducting the actual implementation. There are already several sandbox technologies for malware analysis available. When conducting the implementation design, avoiding re-creating the wheel is preferable. Thus, existing technology combined with custom modifications is of interest. The following sandboxes and/or malware analysis frameworks are considered relevant towards solving the problem description:

- **Cuckoo Sandbox** [3].
- **IRMA:** Incident Response and Malware Analysis [71].
- **R2pipe:** API for scripting Radare2 with Python [72].
- **Remnux:** Linux toolkit for malware analysis [28].
- **Limon sandbox** [16].

In this particular case, Limon became the sandbox solution of choice, because of it's simplicity and the fact that it is tailored for analysing ELF-files within Linux distributions.

4.2 Functionality Design

As some of the features included in Limon's default configuration are out of scope for this thesis, some functionality will not be implemented. On the other hand, there are functionalities required to solve the task that are not a part of Limon by default, e.g. the ability to trace library calls and logging of packed code. These are

features that needs to be integrated with the Limon tool manually.
This thesis focuses on dynamic analysis of malware, and thus, features related to memory analysis will be excluded. Although static analysis methods are out of scope as well, static functionalities will be implemented, as Limon always performs static analysis regardless of chosen arguments.

Internet mode is excluded by choice in this case. The problem description states the requirement for a secure and isolated sandbox environment. A sandbox environment designed this way, should be isolated both from the underlying OS and networks. Isolating the sandbox from the internet is crucial to prevent executed malware from reaching malicious networks, domains or Command and Control Server server. Failing to do so, might also impact other devices on the local network in case of propagation, e.g. worm. From an incident response team or threat hunter perspective, isolating network traffic is key to avoid detecting when analysing sophisticated malwares. As most malware nowdays relies on an internet connection, simulation of such services is still required to replicate a real-world scenario. Limon utilizes InetSim in order to to simulate these services without actually connecting to the internet.

Table 4.1 displays the tools and functionalities designed and implemented in this thesis based on the above discussion. The table also includes custom functionalities which are needed in order to cover the functional requirements. Text colored in blue indicates custom functionality manually integrated.

Tool / Functionality	**Method**	**Implementation**
Internet mode	Dynamic	No
Network simulation (InetSim)	Dynamic	Yes
Network packet capture (TCP-dump/tshark)	Dynamic	Yes
Sysdig	Dynamic	No
Strace	Dynamic	Yes
Volatility	Memory	No
Strings utility	Static	Yes
Hexdump	Static	Yes
Fuzzy hashing (Ssdeep)	Static	Yes
VirusTotal detection	Static	Yes
Readelf	Static	Yes
LDD (Shared library dependencies	Static	Yes
Python Pillow (Screenshot grabber)	Dynamic	Yes
Yara-rules	Static	Yes
ltrace (for tracing library calls)	Dynamic	Yes

Table 4.1: Tools and functionalities planned for implementation.

4.3 Application design

Figure 4.1, *Application interaction diagram*, briefly displays the interaction between Limon and the different tools that the script is dependent on. The host machine takes care of all static analysis operations, since execution of the malware sample is not necessary at this stage. Thus, all of the static analysis tools are installed on the host. The VirusTotal public API is the only application requiring an internet connection in order to work properly.

Regarding dynamic analysis, VMware Workstation is used as hypervisor for the analysis environment. Strace and Ltrace are installed within this environment in order to trace and record system and library calls during execution of the malware sample. InetSim and TCPdump however, is installed in the host OS. This way, TCP-dump might sniff network packets on the virtual network interface, while InetSim simulates network services for traffic transmitted on it.

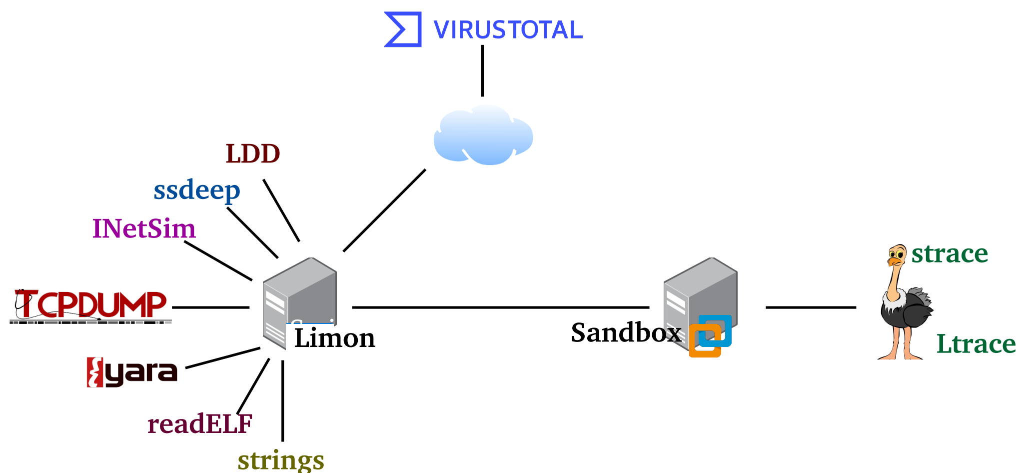


Figure 4.1: Application interaction diagram

4.4 Architecture design

Figure 4.2, shows how the architecture of the project is set up. This framework is built on a virtual machine in Open Stack and uses nested virtualisation in order to create a sandbox within Open Stack. In the Open Stack instance the Limon script initiates a VMware Workstation instance, which is the sandbox used in this project, where all the testing of the ELFs is done. The Limon script returns the results from the ELF-file activity and creates output-files which will be stored on the Open Stack instance.

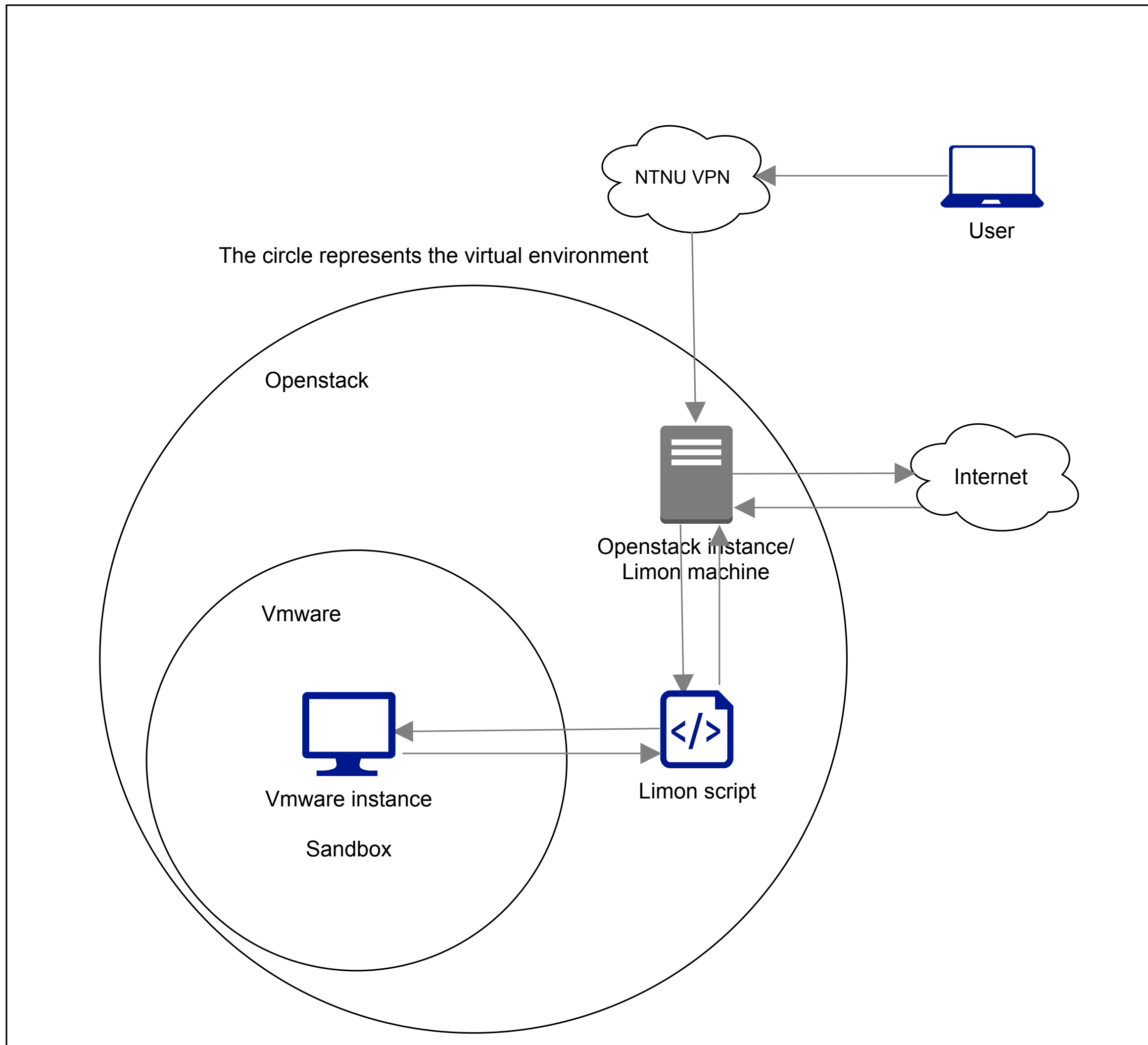


Figure 4.2: Architecture design

4.5 Network design

In the internal virtual network, shown in figure 4.3, the Open Stack instance is the default gateway for the VMware Workstation instance. The Open Stack machine has a connection to the Internet and can communicate outside of the internal network. This is necessary because Limon uses VirusTotal to calculate the threat level of a virus during the static analysis phase. Network traffic from the Open Stack

instance will be routed via the Open Stack network, and the network from the VMware Workstation instance need to be isolated. InetSim on the host machine will simulate network services for the VMware Workstation instance. TCPdump is used in order to sniff network traffic residing from the virtual network, recording the traffic between the host machine and the guest machine.

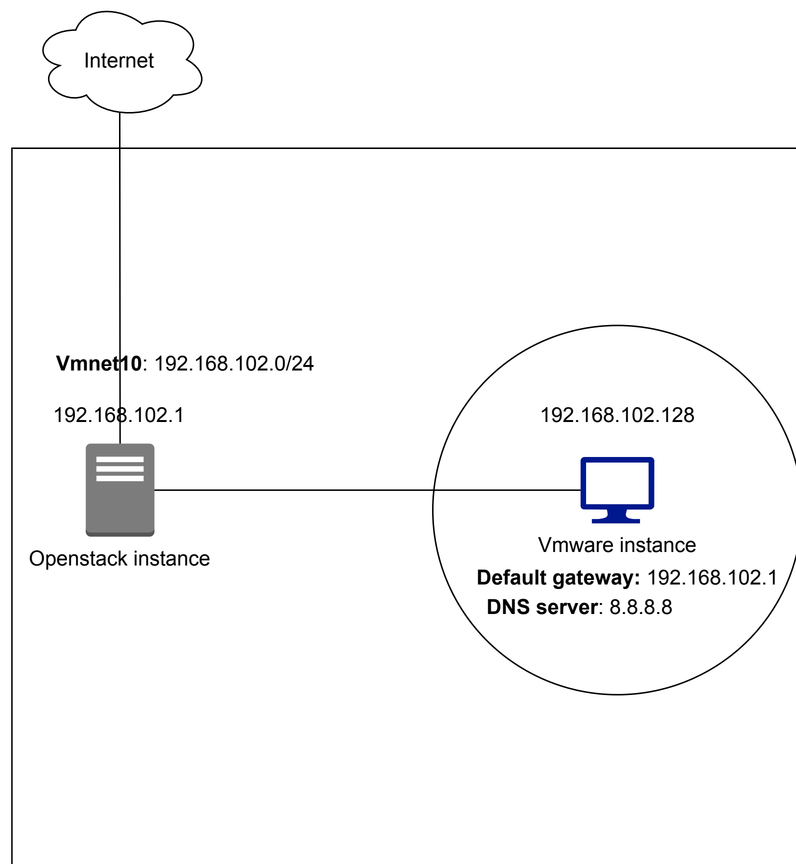


Figure 4.3: Network design

4.6 Sequence diagram

The diagram, 4.4, describes how the framework will operate while running, and how different components in the framework work to complete the analysis process. The diagram focuses mainly on the dynamic analysis aspects of the project

framework, although the framework also performs static analysis of the uploaded ELF. The results and outputs returned have been illustrated as a single arrow in the diagram, but the outputs will also be available separated by each tool as well.

Below is a description of the steps taken in the sequence diagram. The numbers correspond to an arrow in the sequence diagram:

- 1: The user uploads the Executable Linkable Format (ELF) to the Host OS.
- 2: The Host OS runs the Limon script which analyses the uploaded ELF.
- 3: Begin the static analysis.
- 4: After the static analysis the script initiates the dynamic analysis process.
- 5: The Limon script reverts the VMware Workstation instance to a clean snapshot before initiating the test.
- Limon transfer the ELF to the VMware Workstation instance
- 7: Starts TCPdump and InetSim. These tools then begin sniffing packets and monitoring the network activity of the ELF-file.
- Points 8 to 10 happen in parallel but are displayed as sequential for readability purposes.
- 8: In the VMware Workstation instance the Strace monitors the system calls made by the ELF-file. Ltrace is also initiated to monitor the ELF library calls.
- 9: The ELF-file generates network traffic which then is tracked and monitored by InetSim and TCPdump
- 10: The Limon script takes screenshots of the VMware Workstation desktop in order to detect creation of files or started programs.
- 11, 12, 13, 14: When the testing phase is completed the Limon script ends the sandbox and returns the results to the host machine. Each tool returns output to a file which is located on the Host machine where the user can then inspect the results.

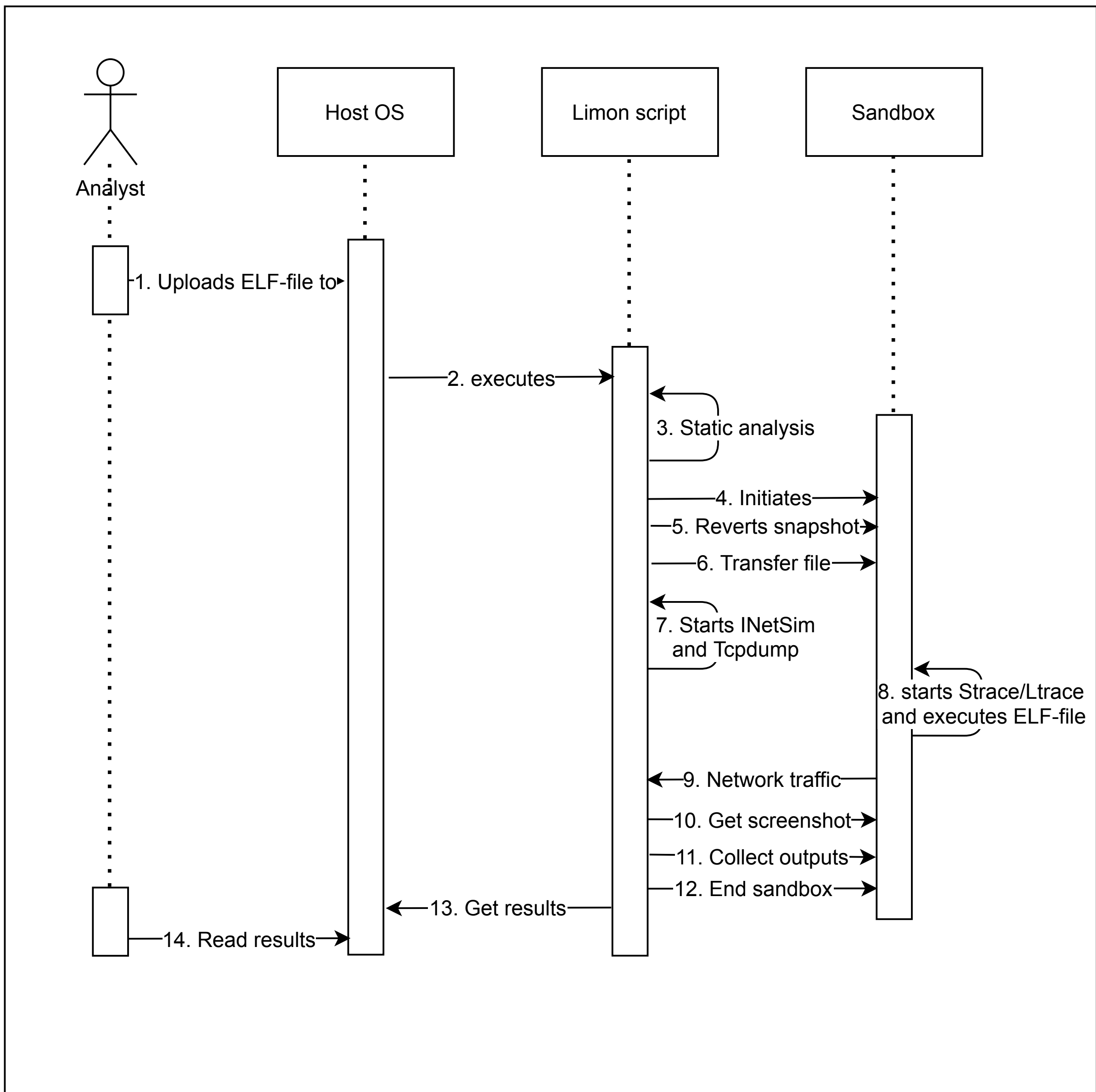


Figure 4.4: Sequence diagram

Chapter 5

Implementation

This chapter will explain the implementation process of the architecture designed in the previous chapter. The chapter is mostly focused on the practical aspects towards answering the research question, but the methodology strategy used during this thesis project will be briefly touched. Furthermore, the implementation regarding the different tools and solutions used will be shown and discussed. Each tool is explained in detail along with a justification on why the chosen tool contributes to solve the problem in question.

5.1 Methodology

The pre-project period, as seen in appendix A, introduced the use of Kanban methodology in combination with a Gantt schema [73] that describes the different phases of the thesis project. Figure 5.1 shows a snippet of the Kanban board in use can be seen in use.

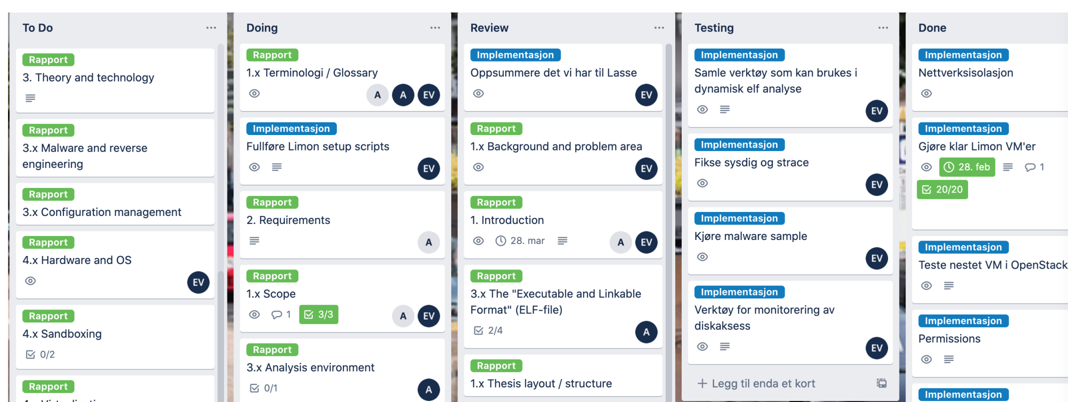


Figure 5.1: Trello Kanban board

5.2 Implementation repository

A BitBucket repository containing this thesis implementation has been created, and is available for cloning [74]. The repository includes an installation script. When ran, it automatically configures Limon on the host OS and configures the guest OS, including the different tools required. Please be advised that references regarding the *BitBucket repository* refers to this thesis development of Limon [74] F. Instructions on how to install Limon from this script is further described in chapter 5.6.

5.3 Infrastructure Configuration

The implementation of Limon may take place on a infrastructure of interest. This could for instance be a physical computer or a VM if nested virtualisation is supported. Regardless of underlying infrastructure, VMware Workstation Workstation is required as hypervisor for the guest OS. As NTNU's Open Stack infrastructure SkyHigh [75] offers plenty of available resources and ease of accessibility, this became the natural choice of infrastructure to implement this thesis. Naturally, as Open Stack is a VM manager [23], nested virtualisation will be required for implementation. The nested virtualisation feature was enabled in SkyHigh after requesting this to the DevOps team.

5.3.1 Initial configuration

In order to use SkyHigh as the underlying infrastructure, initial configuration is required according to the SkyHigh documentation [75]. This includes creating a subnet, configuring a router for external access, a firewall with appropriate security rules and creation of SSH key-pairs for authentication. In this case, security rules allowing egress and ingress IPv4 and IPv6 traffic were created allowing incoming and outgoing traffic to the internet. Internet connection is required in order to download necessary tools and packages. Creating ingress security rules for SSH and ICMP traffic enables remote administration and ensures that the virtual machines in SkyHigh are able to ping each other. Prerequisites needed in other infrastructures may vary from one to another. When implementing using a physical computer, an internet connection and sufficient hardware (as describes in chapter 2, *Requirements*) is required.

5.3.2 Instance deployment

According to the Limon documentation [16], Limon is configured on an Ubuntu operating system. This is also the case regarding the analysis environment. The Ubuntu versions mentioned in the documentation, *15.04 LTS* for the host OS and *12.04 LTS* for the guest OS, are outdated. To replicate a real-world scenario as close as possible, a more updated version of the OS should be considered. Ubuntu

18.04 LTS was chosen for the analysis environment for this thesis implementation, as *18.04 LTS* is widely used at the time of writing. *18.04 LTS* was also chosen as the operating system for the host OS for simplicity.

5.4 Installing cuckoo

Primarily, Cuckoo became the sandbox solution of choice for this thesis, because of it's popularity and recommendation from the client. Cuckoo was installed according to their documentation [3], along with VirtualBox [76] to host the guest OS for malware analysis.
After successfully installing Cuckoo, all of its dependencies and the virtual environment, it came clear that Cuckoo only supported Windows as the analysis environment of malware by default. Thus, the decision was made to implement Limon instead of Cuckoo.

5.5 Limon setup

As mentioned earlier in the report, the Limon sandbox solution is a project presented at BlackHat Europe 2015 [68]. The project is open source and documented to a certain extent. One of the bigger challenges regarding the project is the fact that is has been discontinued since 2016. No commits has been performed since that year. This causes some risk, as the Limon script might not work properly if the dependant tools used by the script aren't compatible with their updated versions. This section describes how Limon and the dependant tools are configured. Code listings with commands and small scripts used in the configuration will be shown throughout this section. Setting up Limon and the dependant tools is a tedious process. Therefore, a Bash-script wrapping up the whole installation has been created, and found in are found in appendix E.1 and E.2 . The reader is still encouraged to read through the whole implementation chapter, in order to gain proper understanding regarding the underlying technology.

5.5.1 Sandbox outline

Figure 5.2 *Architecture design*, displays the overall architecture to be implemented. This figure was initially shown in chapter 4, and is repeated here for the reader's convenience. The overall construction consists of a host system with Limon, VMware Workstation and some monitoring tools installed. When analysing a malware sample with Limon, this machine will conduct static analysis on the sample. Memory analysis will also be performed from this machine if memory forensic tools are installed. However, memory analysis methods is out of scope this thesis project. The analysis environment VM will receive the malware sample from the host OS to perform dynamic analysis, generating reports for further study upon ended execution. A more detailed description on how an analysis is conducted with Limon can be seen in figure 4.4, *Sequence diagram*.

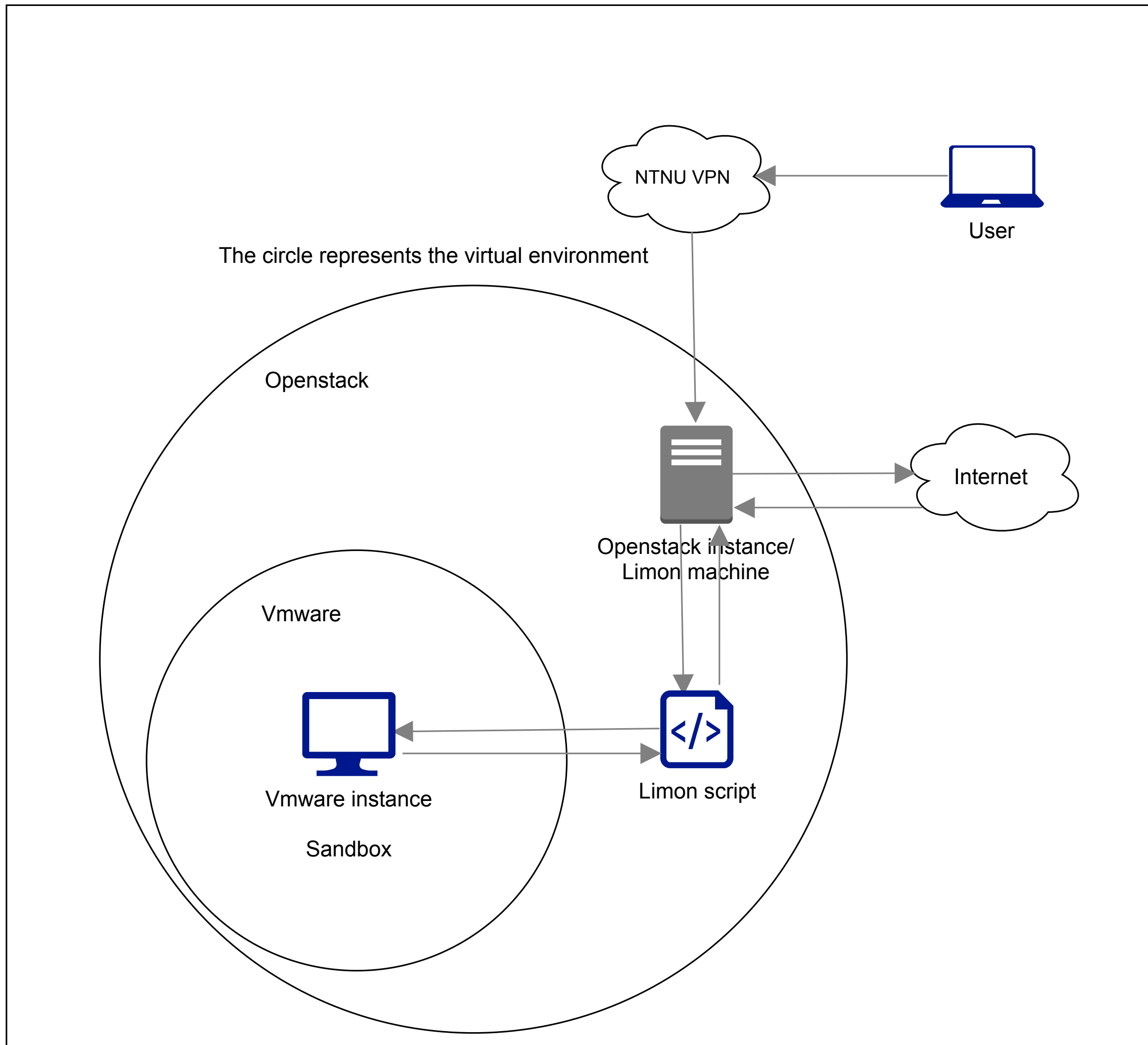


Figure 5.2: Architecture design

5.5.2 Host OS configuration

This sub chapter will explain how the host machine was implemented, documenting the installation of the tools required. As previously mentioned, the host OS is running Ubuntu *18.04 Server*. The tools listed below might be downloaded to the directory of choice. For simplicity, all of the tools required were downloaded to the */home/**current-user*** directory in this case.

Installing desktop environment

A graphical user interface is required to properly complete the implementation of Limon. If the host system is a server distribution without a GUI, a light weight desktop environment needs to be installed. This could for instance be Lubuntu desktop, which is very simple to install [77]. Other popular desktop environments includes, for instance, *KDE* and *Mate Core* [77].

VMware Workstation and guest operating system

Limon is pre-configured to use VMware Workstation as the hypervisor for analysis, as the script is dependent on *vmrun* for executing commands and transferring files between the host OS and guest OS. *vmrun* is VMware Workstation's command line utility for managing VMs [43]. The installation script developed during this thesis, as shown in appendix E.1 and E.2, automatically installs the Linux version of VMware Workstation. Code listing 5.1 displays how the installation is carried out. The installation script E.1 also downloads an Ubuntu 18.04 LTS image from *TrendSigma* [78] to be used as the guest OS.

Code listing 5.1: Installing VMware on Linux

```
#!/bin/bash
sudo ./VMware-Workstation-Full-16.0.0-16894299.x86_64.bundle --console
```

To verify that the VMware Workstation image is working properly, the reader is encouraged to run the *sudo vmrun start <path_to_vmx-file>* command. This will start the downloaded virtual instance in VMware Workstation, ensuring that it is working properly.

Static analysis tools

Some of the tools mentioned in chapter 3, *Theory and technology*, used in Limon's static analysis process are already included in Ubuntu by default. These tools includes LDD, Readelf, TCPdump, and the strings utility. Others require manual installation.

Limon relies on Yara and Yara-Python for detecting packers and further detect capabilities of the executed malware. The Limon documentation [16] dictates the installation of Yara version 1.7.2. Since this version is outdated, a newer version should be considered. This implementation relies on Yara version 4.0.5, which is installed according to their updated documentation [46].

Yara-rules are red by Limon from the */root/yara_rules* directory, and thus this directory should be created to store these rules. Rules are available from this GitHub repository [79]. By default, Limon is configured to detect malware capabilities and packers, by applying corresponding rules. However, which rules to match, might

be changed by applying other rule sets if needed.

The fuzzy hashing tool, Ssdeep, are available for download using the command *apt-get install ssdeep*.

INetSim

Limon relies on InetSim to simulate network services for the sandbox. The Limon documentation [16] refers to the official InetSim documentation page [69] for downloading and installing the software. The installation and configuration of InetSim is poorly documented, and might therefore appear as challenging. As an alternative approach, Remnux CLI can be downloaded [28], considering the fact that InetSim is included in Remnux. Remnux is an open source toolkit for malware-analysis. This eases the installation process of InetSim, as Remnux is straight forward to setup. It also ensures that InetSim is correctly configured, as the process is automated. For the sake of simplicity and security, Remnux CLI has been implemented to ensure correct configuration of InetSim in this thesis project. The reader should bear in mind that installing Remnux includes a lot of other tools not need for this implementation, using unnecessary storage space.

Volatility

Limon depends on Volatility [20] for performing memory analysis. Memory analysis is out of scope for this thesis, and thus, Volatility is not implemented.

Analysis directory

Limon requires a directory for storing outputted reports upon ended analysis. According to the Limon configuration file, *conf.py*, reports are stored within */root/linux_reports*, and thus, this directory needs to be created. If a non-root directory is preferable, *conf.py* needs to be updated with the appropriate directory.

5.5.3 Guest OS configuration

This sub section focuses on configuring the guest OS analysis environment (also referred to as the sandbox). According to the Limon documentation [16], root login should be enabled for executing malware samples in this environment. However, as requested by the project client, it is desired that malware samples are executed with lower privileges, in order to fully observe what the executed malware sample might achieve.

As mentioned in the host OS configuration, Ubuntu 18.04 will be used as the guest OS for this implementation. If implementing is done through the automated setup scripts E.1E.2, the guest OS configuration below is part of the automated process.

Strace

Strace is a tool that traces system calls [34]. The Limon documentation [16] mentions that some Strace versions included in Ubuntu by default might not work properly with Limon. To ensure proper functionality, Strace should be re-installed as version 5.11, as referred to in code listing 5.2.

Code listing 5.2: Reinstallation of Strace

```
# Strace
echo "password" | sudo -S apt-get remove -y strace
wget https://strace.io/files/5.11/strace-5.11.tar.xz
tar -xf strace-5.11.tar.xz
cd strace-5.11
./configure --disable-mpers
make
make install
cd ..
rm strace-5.11.tar.xz
```

PHP CLI

Limon does support analysis of PHP-scripts. This is out of scope for this thesis project. If the reader wishes to implement this feature, this can easily be done by running the command *apt-get install -y php-cli*.

Running 32 bit executables on 64 bit Ubuntu

There are cases where one might need to run and analyse 32bit executable ELF-files, which might not work by default on a 64bit architecture. The following library packages should be installed according to the Limon documentation to ensure proper functionality in these cases, as specified in code listing 5.3

Code listing 5.3: Library packages for executing 32bit binaries on 64bit architectures

```
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386
```

Volatility profile creation

Memory analysis is, as mentioned earlier, out of scope for this thesis and will not be implemented. In cases where memory analysis is preferred, a Volatility profile of the guest OS needs to be created and transferred to the host OS. This is further described in the documentation [16], and will not be elaborated on in this report.

Miscellaneous

malware samples transferred to the sandbox needs to be stored inside the */home/* ***your_user****/malware_analysis* directory. In the default Limon source code, samples

are stored under */root/malware_analysis*. However, since this implementation focuses on executing malware with lower privileges, the directory should be created somewhere with proper user permissions, such as the path mentioned above. The *malware_analysis* directory should also be added to the end of *PATH* in */etc/environment*, as shown in code listing 5.4.

The */home/**your_user**/logdir* directory needs to be created for the sandbox to store analysis result reports. Strace and Ltrace output will be stored here upon ended analysis, before being transferred to the host system.

The guest OS configuration is completed by clearing the Bash history, which is done by running the command *history -c && history -w*. This will clear both the Bash history residing in memory and on the disk.

Code listing 5.4: Adding *malware_analysis* directory to */etc/environment*

```
mkdir malware_analysis
mkdir logdir
PATH=$PATH:~/home/user/malware_analysis
touch path.txt
echo -n 'PATH="' >> path.txt
echo -n $PATH >> path.txt
echo '"' >> path.txt
cat path.txt > /etc/environment
rm path.txt
```

5.5.4 Sandbox network configuration

As VMrun networking commands are only supported in *VMware Fusion Pro* for MacOSX, network configuration has to be performed through the VMware Workstation GUI. To ensure that the sandbox is completely isolated, it should be placed on a custom virtual network without access to the internet. Network traffic originating from the sandbox will be routed to the host OS, which runs InetSim to simulate network services. A more detailed step-by-step explanation of the network configuration is described in appendix E.

Creating a custom virtual network

A custom virtual network can be created within the *Virtual Network Manager* in the VMware Workstation GUI. A new *host-only-network* with a subnet mask of 255.255.255.0 should be created, as seen in figure 5.3. The *Subnet IP* field will be blank when the network is created, and should be filled with the network address of choice (e.g. 192.168.102.0). A static IP address for the VM will be assigned further on in the configuration process.

Next, the newly created network needs to be added to the sandbox instance. This is done by entering the instance settings and selecting the newly added custom

network, as shown in figure 5.4.

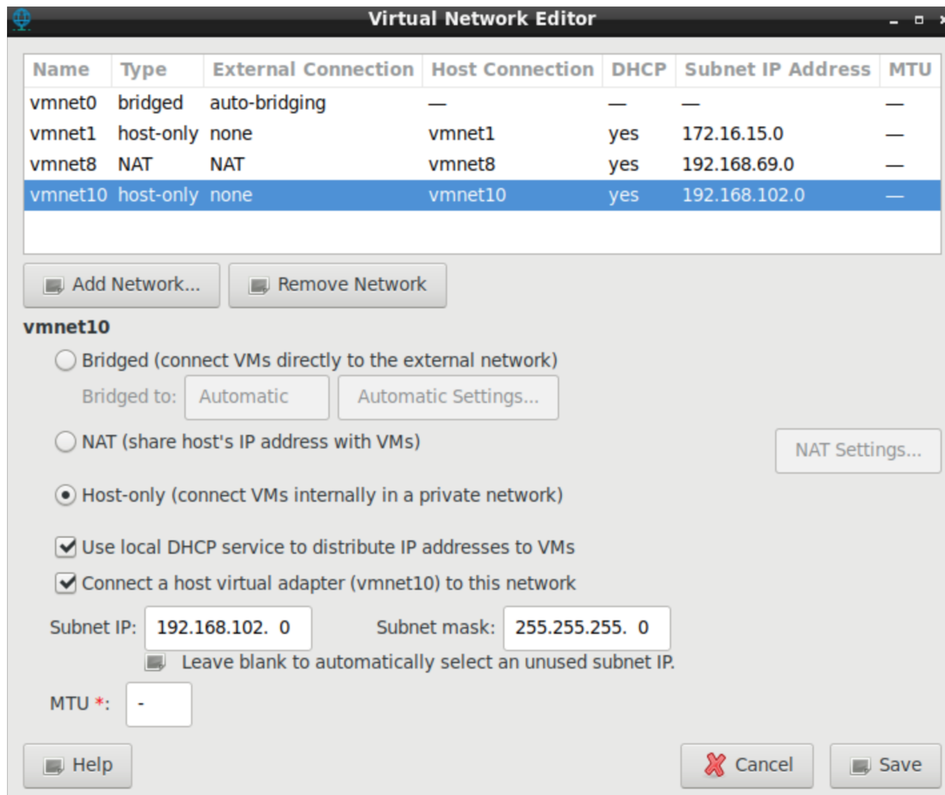


Figure 5.3: Adding a host-only-network (vmnet10).

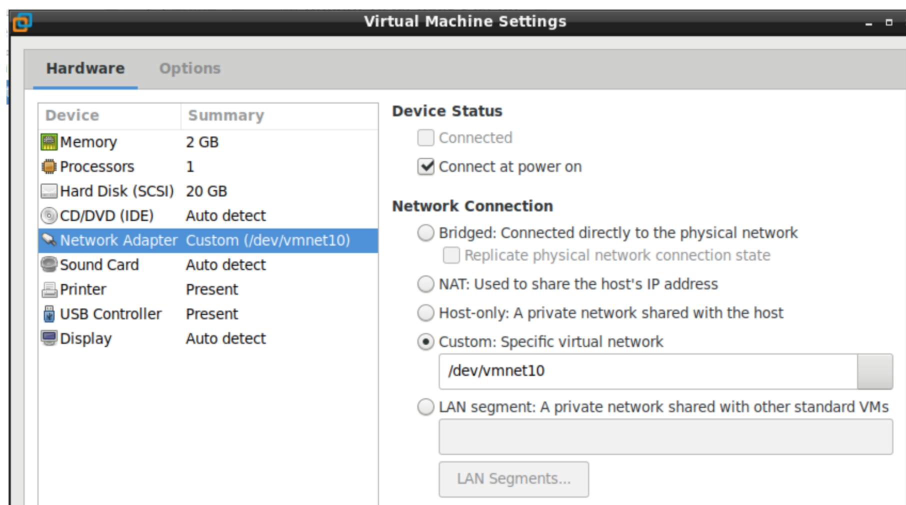


Figure 5.4: Assigning the custom vmnet to the virtual instance.

Setting default gateway and preferred DNS on guest instance

As mentioned previously, network traffic originating from the sandbox are to be routed to the host OS. To achieve this, the default gateway of the sandbox instance is statically set as the IP address of the host OS on the custom *vmnet*. This is achieved by running the command *route add default gw <IP address> <INTERFACE-NAME>*, where *IP address* represents the host OS IP address and *INTERFACE-NAME* is the name of the guest network interface (this can be checked by running the command *ip addr* on the sandbox instance), e.g. *route add default gw 192.168.102.1 ens33*. The action might be verified by running the command **/sbin/route -n**, as shown in figure 5.5.

```
user@ubuntu:~$ /sbin/route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 ens33
192.168.102.0   0.0.0.0         255.255.255.0   U     100    0        0 ens33
```

Figure 5.5: Confirming that host OS acts as default gateway for the guest OS sandbox

Next, the preferred DNS address will be statically set. Since the virtual network that the sandbox is connected to is isolated from the internet, the IP address of the host OS should be set as the preferred DNS. This is achieved by editing the network settings visually in the guest OS, as seen in figure 5.6. Verification of updated DNS resolvers may be performed by running the command *systemd-resolve –status | grep 'DNS Servers' -A2*, as seen in figure 5.7. If the system's DNS settings are cached on the system itself, a reboot might be required to verify these changes.

Assigning static IP address to guest instance

A static IP address for the guest OS sandbox running on the custom virtual network must be specified. This is important as the IP address of the sandbox instance needs to be set manually in the Limon configuration file, *conf.py*. Since Limon reverts snapshot for each analysis, the IP address of the sandbox might change if done otherwise. This will result in the configuration not being able to listen to network traffic correctly.

This has to be changed in the VMware Workstation *dhcpd* configuration file on the host OS: */etc/vmware/vmnet<YOUR_VMNET>/dhcpd/dhcpd.conf*. There should be an entry for the custom *vmnet* previously created in this file. Below this entry, a new entry (also with the name of the custom *vmnet*) needs to be added. Inside this entry, *hardware ethernet* should be equal to the sandbox instance's MAC-address, and *fixed-address* equal to the IP address of choice within the previously defined subnet. An example is shown in figure 5.8.

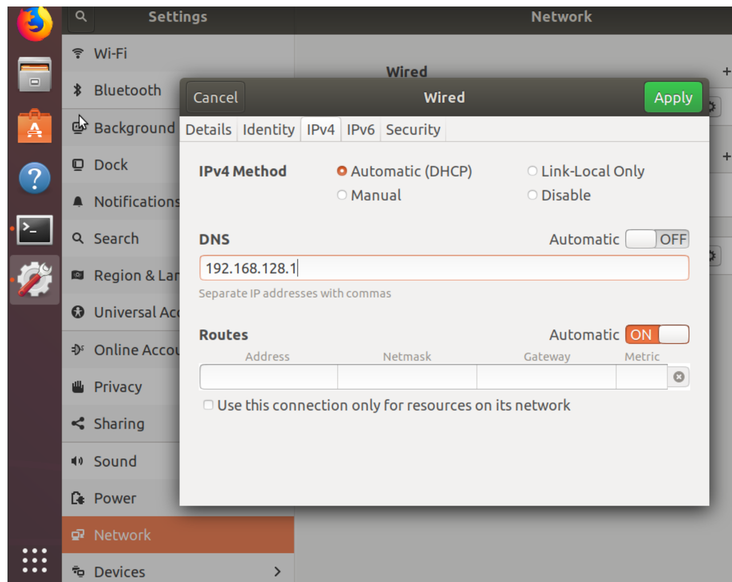


Figure 5.6: Setting preferred DNS visually.

```
user@ubuntu:~$ systemd-resolve --status | grep 'DNS Servers' -A2
            DNS Servers: 192.168.102.1
```

Figure 5.7: Confirming changes in preferred DNS.

```
host vmnet10 {
        hardware ethernet 00:50:56:C0:00:0A;
        fixed-address 192.168.102.1;
        option domain-name-servers 0.0.0.0;
        option domain-name "";
}
# Ubuntu 1804 guest IP
host vmnet10 {
        hardware ethernet 00:0C:29:95:05:99;
        fixed-address 192.168.102.128;
}
```

Figure 5.8: Adding static IP address to the sandbox instance.

After the above configuration has been implemented, VMware Workstation's DHCP service should be restarted using the commands *net stop vmnetdhcp* and *net start vmnetdhcp* on the host OS.

Lastly, one should configure InetSim in */etc/inetsim/inetsim.conf* on the host OS. Uncomment the network services of choice to be simulated by InetSim, as shown in figure 5.9. The options for *service_bind_address* and *dns_default_ip* needs to be uncommented, and the value of these two options should be set equal to the host OS IP address on the custom VM network.

```
start_service dns
start_service http
start_service https
start_service smtp
start_service smtps
start_service pop3
start_service pop3s
start_service ftp
start_service ftps
#start_service tftp
#start_service irc
#start_service ntp
#start_service finger
#start_service ident
#start_service syslog
```

Figure 5.9: Choosing network services to be simulated by INetSim.

This concludes the network configuration for the Limon architecture. Figure 5.10, *Network Architecture*, is repeated here for the sake of simplicity.

Code listing 5.5: Assigning appropriate variables for Limon in *conf.py*

```
##############[general variables]################################
py_path = r'/usr/bin/python'
report_dir = r'/home/ubuntu/linux_reports'
virustotal_key = "my_VirusTotal_API_key"

###############[vm variables]#####################################
host_analysis_vmpath = r'/home/ubuntu/Ubuntu1804/Ubuntu.vmx'
host_vmrunpath = r'/usr/bin/vmrun'
host_vmtype = r'ws'
analysis_username = "user"
analysis_password = "password"
analysis_clean_snapname = "cleansnapshot"
analysis_mal_dir = r"/home/user/malware_analysis"
analysis_py_path = r'/usr/bin/python3'
analysis_perl_path = r'/usr/bin/perl'
analysis_bash_path = r'/bin/bash'
analysis_sh_path = r'/bin/sh'
analysis_insmod_path = r'/sbin/insmod'
analysis_php_path = r'/usr/bin/php'

################[static analyis variables]##########################
yara_packer_rules = r'/home/ubuntu/yara_rules/packers_index.yar'
yara_rules = r'/home/ubuntu/yara_rules/capabilities/capabilities.yar'

#################[network variables]################################
analysis_ip = "192.168.102.128"
host_iface_to_sniff = "vmnet10"
host_tcpdumppath = "/usr/sbin/tcpdump"

#######################[memory anlaysis variables]##################

vol_path = ''
```

```
mem_image_profile = ''

######################[inetsim variables]########################
inetsim_path = r"/usr/bin/inetsim"
inetsim_log_dir = r"/var/log/inetsim/"
inetsim_report_dir = r"/var/log/inetsim/report"

######################[monitoring varibales]########################
analysis_sysdig_path = ''
host_sysdig_path = ''
analysis_capture_out_file = ''

analysis_strace_path = r'/usr/local/bin/strace'

strace_filter = r"-etrace=fork,clone,execve,chdir,open,creat,close,socket,connect,
accept,bind,read,write,unlink,rename,kill,pipe,dup,dup2"

analysis_strace_out_file = r'/home/user/logdir/trace.txt'
analysis_ltrace_path = r'/usr/bin/ltrace'
analysis_ltrace_out_file = r'/home/user/logdir/ltrace.txt'
analysis_log_outpath = r'/home/user/logdir'
```

5.5.5 Configuring the Limon script

The Limon script itself needs to be downloaded to the host OS. Everything needed is included in the implementation's BitBucket repository [74]. The Limon configuration file, *conf.py*, needs to be configured properly for the script to work. This process consists of updating variables in *conf.py* with relevant parameters and paths to the tools installed above on both the host OS and guest OS. Code listing 5.5 contains all the variables that needs to be updated and set in *conf.py* (the configuration file has several other entries than listed in code listing 5.5. Only variables requiring assignment has been included). Calling the Bash command *which* on a program, e.g. *which strace*, is useful in order to identify path of the programs.

5.6 Scripting the Limon setup

The above installation is a tedious process that involves installing several individual open source tools that together shapes the foundation of Limon. Thus, the reader might run the custom Limon setup script developed in this thesis to ease the implementation process. Running the setup script, *limon_setup.sh* E.1, installs all of the tools which Limon is dependent on, along with creating appropriate directories for analysis. All scripting material are also provided in the BitBucket repository developed through this thesis [74].

5.6.1 Installing Limon from the thesis repository

To install Limon by using the installation script developed through this thesis, clone the BitBucket repository [74] on the host OS. The following files are relevant

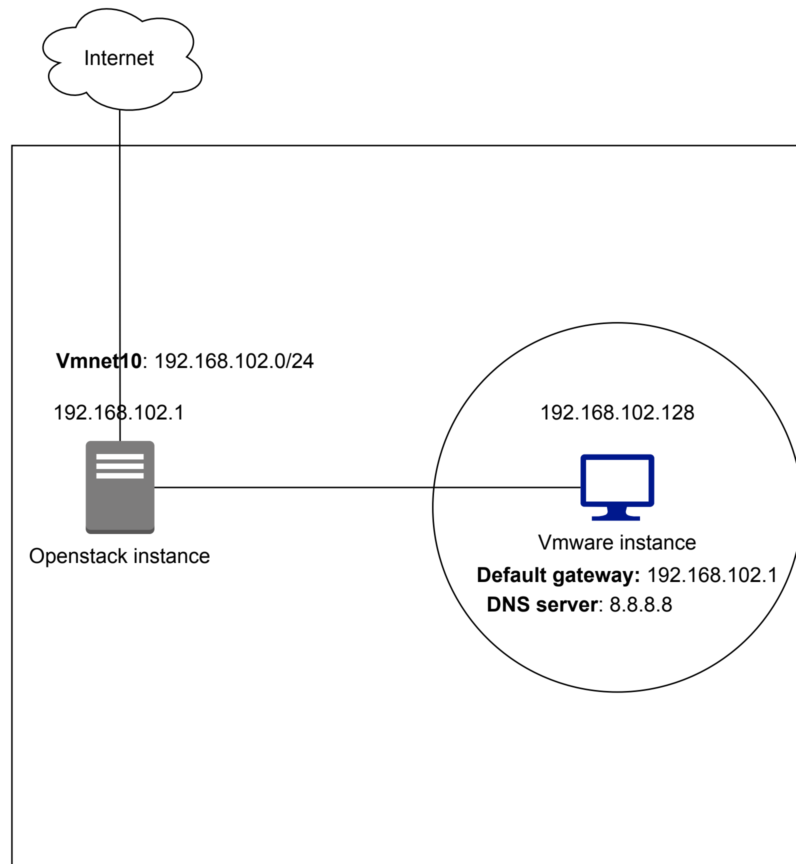


Figure 5.10: Network design

for the installation:

- **limon_setup.sh**: Configures the host OS
- **analysis_host_setup.sh** Configures the analysis machine (guest OS). Automatically invoked by *limon_setup.sh*.

The configuration is initiated by running *limon_setup.sh*. This will configure the host OS and install the dependent tools, including VMware Workstation and the Ubuntu VM image needed for the guest OS. Using VMrun, *analysis_host_setup.sh*, are copied and executed within the guest OS, configuring this machine as well. Lastly, the Sysdig scripts are transferred to the guest OS. These are not executed in the installation, and are used when running an analysis. The Volatility image from the guest OS are also created and copied to the host OS. When execution of the script has ended, the host OS is rebooted in order for changes to take effect.

Please be advised: As seen in code listing 5.5, configuring the Limon configuration file, *conf.py*, has to be done manually. This applies for the network configurations shown in chapter 5.5.4 as well. When these configurations are considered completed, a snapshot of the analysis environment needs to be created. This is achieved in VMware Workstation's snapshot manager menu. The name of the snapshot created should also set in the *conf.py* file.

5.7 Limon modifications

In order to accomplish all of the required functionalities for this thesis implementation, some modifications deviating from the official documentation took place. The documentation does encourage the reader to run root as default in the sandbox. This implementation runs analysis as a normal low privileged user in order to fully capture the capabilities of analysed malware. In addition, the ability to trace malware library calls is not a feature in the official Limon implementation [67]. Thus, this particular Limon implementation is modified and able to record library calls using the *ltrace* tool [17].

For the sake of security, internet mode have been excluded. As the various tools mentioned through this chapter has the ability to record both system and network activity, the implementation of Sysdig has been excluded as well. Approaches and argumentation regarding excluded tools and functionalities are further discussed in chapter 7.

Chapter 6

Analysis and testing

This chapter will describe results from some analysis tests, as well as the different outputs being produced when conducting an analysis.

6.1 Limon usage

Table 6.1 shows the options that can be given when running Limon. Each flag (except the timeout flag) uses a tool which will impact how the analysis is conducted. If the Ltrace flag is not added to the command when running Limon, then the Strace tool will be used instead for the testing process. The script is simply executed by running the following command from inside the cloned directory:

sudo python limon.py -t 60 <args> <path-to-malware-sample> <malware-args>

6.2 Analysis output

After successfully conducting an analysis of an ELF-sample, Limon produces output files containing the reported results. This includes reports related to system activities, network activities, static analysis data and a summary. A screenshot of the sandbox desktop is also captured. Depending on the chosen arguments, different output files might also be produced. Table 6.2 describes the output files along with their relation to the corresponding tools.

6.3 Performance

This section briefly covers different performance aspects regarding the implementation of Limon in this thesis. Performance regarding malware execution, accuracy and mass testing are considered.

Script	Description	Tool
–timeout, -t	Set desired timeout	N / A
–libtrace, -l	Traces library calls.	Ltrace
–libstrace, -L	Traces library calls and system calls.	Ltrace
–printhexdump, -x	Limon prints the hex dump in call trace.	Hexdump
–python, -P	Additional python script to run in the sandbox.	python
–perl, -p	Additional perl script to run in the sandbox.	perl
–shell, -s	Runs additional shell script.	shell
–php, -z	Runs additional php script.	php
–Bash, -b	Runs additional bash script.	bash
–ufctrace, -C	Unfiltered call trace (full trace).	Strace

Table 6.1: Features used by Limon.

6.3.1 Execution performance

When analysing a malware sample with Limon, the user chooses a timeout which defines the amount of seconds for the sandbox to run. After the desired amount of seconds has been reached, associated processes are killed and the sandbox is suspended. The sandbox continues to run until the desired timeout has been reached, regardless of whether execution of the malware sample has finished or not. This allows recording of post execution network activities.

6.3.2 Functionality testing

In order to ensure proper functionality and performance of the implementation over time, a mass-test was conducted. This test sequentially carried out malware analysis of 100 unique samples G from *VirusShare* [80]. Each sample was ran with the -L parameter in Limon for tracing both library and system calls with Ltrace.

Listing 6.1 displays how such a test might be carried out with Limon. The scripts loops through the malware sample directory on the host machine, running an analysis with Limon for each sample. Please be advised that this test should not be considered as a "heavy load" test. The test was conducted in order to observe Limon's functionality in different scenarios, ensuring that the script is working properly along the way.

Output-file	**Description**	**Tool**
final_output.txt	This file contains a summary of both the dynamic and static analysis as well as the network traffic and the system calls.	Ssdeep, InetSim, TCPdump, Strace
ltrace.txt	Contains output from the Ltrace tool.	Ltrace
trace.txt	Contains output from Strace tool which is used when the Ltrace is not used.	Strace
desktop.png	Takes screenshot of the desktop to see whether the malware sample has created files on the desktop.	Pillow
output.pcap	Dump of the network traffic.	TCPdump
strings_unicode.txt, strings_ascii.txt	Extracts the texts strings embedded in the program by using the Strings software. Both ascii and unicode output is produced.	Strings

Table 6.2: Description of output-files and which tools generate them.

For this functionality test, a timeout of 60 seconds was set. More precisely: The sandbox will run for 60 seconds per malware sample before being suspended and reverted for analysing the next sample. 1 minute of execution should make room for plenty of time for the malware samples to carry out its actions, and should be enough margin for observing potential network activities after execution.

Taking into consideration the timeout (t), additional time (a) and the amount of malware samples to analyse (m), please consider the following time calculation formula for the functionality test:

$$minutesInTotal = \frac{(t + a)m}{60} \tag{6.1}$$

Considering equation 6.1 above, the amount of time it takes to conduct the performance test is calculated in equation 6.2:

$$minutesInTotal = \frac{(60 + 15)100}{60} = 125 \tag{6.2}$$

As shown above, a functionality test for 100 malware samples using a 60 second timeout would take approximately 2 hours in total. The functionality test carried out in this thesis implementation was considered successful. No errors or abnormalities were encountered during the performance test, and reports were generated for each sample tested.

Code listing 6.1: Simple Python script for running Limon with several malware samples.

```
import os

for i in os.listdir("/home/ubuntu/malware_samples/VirusShare_samples/"):
        os.system('sudo python limon.py -L -t 60
        /home/ubuntu/malware_samples/VirusShare_samples/'+str(i))
        print("\n")
```

6.3.3 Successful to unsuccessful sample execution ratio

Although no errors were raised during the functionality test, there were some malware samples that did not execute properly during the analysis. The subject area of ELF-files is complex, compared to other binary file types in terms of strcture. Thus, there are several reasons for why an ELF-file might not execute properly when analysing with Limon. Figure 6.1 shows an overview of the different reasons regarding failed executions observed in the functionality test. In total, 38% executed successfully, while the remaining 62% were considered as failures. Successfully executed malware samples are considered as sample which executed without returning any errors in the call trace activity, potentially achieving desired results. The aspects regarding failed sample execution are further discussed below. It is important to investigate these aspects, in order to fully understand factors that might affect proper dynamic analysis of ELF-files.

Root permissions required

As described in the previous chapters, the sandbox in this thesis implementation runs as a normal user with standard privileges. Software needing root permissions are not able to run unless the correct password is specified using sudo. A lot of malware samples require root permissions in order to execute properly, resulting in failed execution if otherwise. The user should bear this in mind when analysing a malware sample using this particular implementation of Limon. 11% of the failed samples during the functionality test failed based on this.

Faulty usage

One of the major factors regarding unsuccessful sample execution had to do with faulty "user interaction" with the sample. When running a program from the CLI, the user often has to pass arguments along with the command in order to select

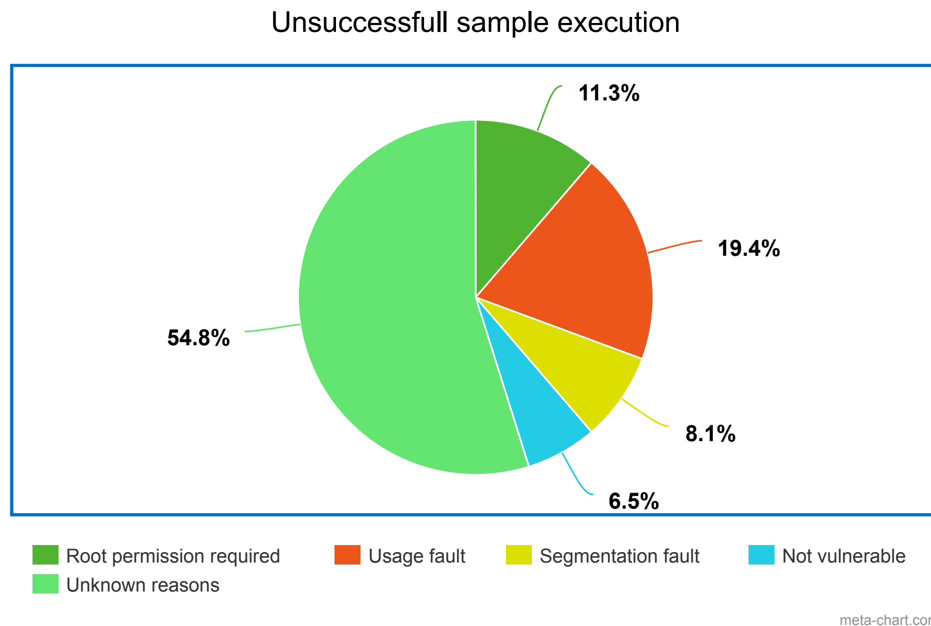


Figure 6.1: Reasons regarding failed execution of samples during performance test.

desired program functionality.

There are several ways to display the arguments of a program. For instance, when running a program without arguments, it might display the arguments available along with required usage, before terminating entirely. In other cases when executing a program, the user is able to choose arguments while the program is hanging and waiting for further input.

Consider comparing figure 6.2 and figure 6.3. Figure 6.2 shows and example of a malware sample which terminated since the amount of arguments required to proceed execution were not met. However, Limon are able to handle arguments from the user when running an analysis. Listing 6.2 shows how analysing the sample shown in 6.2 with Limon using the correct amount of arguments might be carried out. Figure 6.3 shows the output of this sample when executed correctly with arguments. Since different malware samples in many cases requires different amounts of arguments, this has not been possible to demonstrate through the functionality test. Analysing concrete and individual malware-samples are recommended.

Code listing 6.2: Running a malware sample along with required arguments in Limon.

```
#!/bin/bash
sudo python limon.py -t 60 /home/ubuntu/malware_samples/mole_scanner.elf 5 8080 3
```

```
__libc_start_main(0x80488e0, 1, 0xffac7b14,
__register_frame_info(0x804a4b0, 0x804a5e4,
printf("Usage: %s <b-block> <port> [c-bl"...
```

Figure 6.2: Mole scanner sample terminated because of missing arguments.

```
9794 time(0)                                        = 1620826935
9794 time(0)                                        = 1620826935
9794 time(0)                                        = 1620826935
9794 printf("\n[+] mole2 scan completed in %u "..., 1, 0) = 54
9794 fclose(0x82a4160 <unfinished ...>
```

Figure 6.3: Mole scanner sample executed successfully passing required arguments when running Limon.

Malware comes in various variants with different use cases. Some are used to exploit the system [81], while others are used to penetrate further into an already compromised system. This is also known as post-exploitation [82]. Post-exploitation malware often requires input from the threat actor to run desired options after being deployed on the compromised system. This might for instance be options such as attempting privilege escalation, start enumeration/scanning, or establishing Command and Control Server communication. Upon analysing results from the functionality test, this seemed to be the cause in many cases based on observations of functions printing these options during execution.

Some popular examples of such post-exploitation software include Cobalt Strike [83] and Mimikatz [84].

System not vulnerable

In order for a malware to successfully exploit a system, the system needs to be vulnerable. If otherwise, the exploitation attempt will fail. This was the case for at least 6,5% of the samples executed during the functionality test. These cases were identified from print functions in the call trace activities. Please be advised that these numbers might not be exact, as this cause might be the case for other malware samples which failed execution as well. Only cases where this fact is certain are considered in this calculation.

Segmentation fault

Approximately 8% of analysed samples failed execution because of a segmentation fault error, which were specified in the call trace activities. A segmentation fault,

also known as segfault, is described as a memory error in a program which tries to access a memory address that does not exist, or without proper access rights [85]. The most common causes for the error includes poor programming, incorrect pointer manipulation or invalid assumptions regarding shared libraries [86]. In some cases, a segfault might also occur if a vulnerability has been patched in the operating system.

Unknown failure reasons

Over 54% of the malware samples that executed incorrectly produced empty call trace outputs, resulting in troubles regarding determining the cause. However, re-running an analysis for some of these samples using Strace instead of Ltrace produced the same segfault error as mentioned earlier. It is thus possible that some of these samples experienced segfault issues, which Ltrace was not able to properly detect. This also proves the fact that Strace is more accurate on system call tracing than it's brother, Ltrace, which is mainly for tracing library calls.

The static analysis part proved to be useful in determining possible unknown execution issues. Figure 6.4 displays an error message from Readelf, which was frequently encountered for these samples. This might indicate that the ELF-file is corrupt, either as a result of poor programming or as a measure to slow down analysts. It might also indicate that the sample is stripped. A sample compiled with a strip flag instructs the compiler to discard debugging symbols, reducing disk size and making it harder to debug or reverse engineer [87]. As shown in figure 6.5, this might be verified by running the command *file* on the sample. Based on this intelligence, this is most likely the case for the analysis results where Ltrace were not able to produce any output.

```
----------------------------------------
readelf: Error: Reading 1856 bytes extends past end of file for section headers
readelf: Error: the dynamic segment offset + size exceeds the size of the file
Section Header Information:
There are 29 section headers, starting at offset 0x6740:

----------------------------------------
readelf: Error: Reading 1856 bytes extends past end of file for section headers
readelf: Error: the dynamic segment offset + size exceeds the size of the file
Symbol Information:
```

Figure 6.4: Readelf reading past end of file for section headers.

```
ubuntu@limon4:~/malware_samples/moved_files$ file VirusShare_1c2c9f1af3cd38775278dfc5712692e6

VirusShare_1c2c9f1af3cd38775278dfc5712692e6: ELF 64-bit LSB shared object, x86-64, version 1
(SYSV), BuildID[sha1]=ff39ec0381339448b48588501f6c5d3db1788ed3, dynamically linked, stripped
```

Figure 6.5: Stripped ELF-binary without call trace output.

There is also a chance that these samples were obfuscated using packers. However, Yara did not indicate detection of any packers for these particular malware samples. Please be advised that the possibility of packed malware is still existing, regardless of Yara-detection. This thesis implementation of Limon does not have the ability to trace Assembly instructions or to log executed code. Some solutions to this are further discussed in chapter 7 Discussion.

Unsuccessful sample execution summary

As discussed above, there are several discovered failure reasons for the malware samples tested in the functionality test. These findings has been elaborated on in detail in order to enlighten the reader as much as possible on these topics. Unknown reasons and Segfaults are considered the most serious failure reasons towards the technical implementation. However, there are measures towards these problems which is further described in chapter 7. Please be advised that these encounters might also be dependant on the sample that is analysed, in terms of poor programming or memory based reasons.

Cases regarding usage faults, required root permissions or the system not being vulnerable, are considered less serious as these encounters are individual and sample specific errors. It is possible that some of the samples falling under the "segfault" or "unknown reasons" category would have fallen under other categories if more output were produced. Since there are so many factors affecting how an ELF-file is properly executed, mass-testing of ELF-samples in a real-world scenario is a challenging task.

6.4 Examples

This section describes examples from some of the analysis results to illustrate achieved functionality with Limon. All of the malware samples described below were executed with Ltrace for 60 seconds, providing output about library calls and system calls. The purpose of this section is not to illustrate knowledge within malware analysis, but to demonstrate how Limon output might be used in practical scenarios by security analysts or researchers.

6.4.1 Tsunami malware execution

As seen in figure 6.6, *Tsunami VirusTotal detection*, this sample was identified as Tsunami trojan/backdoor by several engines on VirusTotal. Yara also detects the sample as *network irc*.

```
----------------------------------------
Malware Capabilities and classification using YARA rules:
        [network_irc]
----------------------------------------
Virustotal:

          ALYac ==> Gen:Variant.Backdoor.Linux.Tsunami.1
          AVG ==> ELF:Tsunami-A
          Ad-Aware ==> Gen:Variant.Backdoor.Linux.Tsunami.1
          AegisLab ==> Trojan.Linux.Tsunami.4!c
          AhnLab-V3 ==> Linux/Tsunami.Gen
          Antiy-AVL ==>
          Arcabit ==> Trojan.Backdoor.Linux.Tsunami.1
          Avast ==> ELF:Tsunami-A
          Avast-Mobile ==>
          Avira ==> BDS/Katien.R
          Baidu ==>
          BitDefender ==> Gen:Variant.Backdoor.Linux.Tsunami.1
          BitDefenderTheta ==>
          Bkav ==>
          CAT-QuickHeal ==>
          CMC ==>
          ClamAV ==> Unix.Malware.Agent-7438859-0
          Comodo ==> Malware@#3t5ddzbeperta
          Cynet ==> Malicious (score: 85)
          Cyren ==> E64/Backdoor.GYKE-
          DrWeb ==> Linux.BackDoor.Tsunami.346
```

Figure 6.6: Tsunami VirusTotal detection

Moving on to the dynamic analysis results, the call trace activity recorded by Ltrace shows interesting finds which illustrates how Limon might be used to inspect malware behaviour. Figure 6.7 indicates creation of a file called *cron* in the */tmp/* directory, followed by some string compare operations and memory allocation. This figure also illustrates how Ltrace accurately traces high level library calls.

```
9748 strlen("touch /tmp/cron")                   = 15
9748 strcmp("touch", "for")                      = 14
9748 strcmp("touch", "until")                    = -1
9748 strcmp("touch", "in")                       = 11
9748 strcmp("touch", "then")                     = 7
9748 __ctype_b_loc()                             = 0x7f5ccef304d8
9748 __ctype_b_loc()                             = 0x7f5ccef304d8
9748 __ctype_b_loc()                             = 0x7f5ccef304d8
9748 __ctype_b_loc()                             = 0x7f5ccef304d8
9748 __ctype_b_loc()                             = 0x7f5ccef304d8
9748 malloc(16)                                  = 0x55f5281a1f20
9748 strcspn("touch", "\210\203\201\202\204\207") = 5
```

Figure 6.7: Possible file creation of */tmp/cron*, string comparisons and memory allocation.

Several interesting observations from Ltrace are observed in the output report. As seen in figure 6.8, the malware sample tries to tamper with *cron*, which is a time-based job scheduler in Linux [88]. Based on the above observations, the sample apparently attempts to schedule a weekly download from the domain *stablehost[.]us*. Searches in open sources indicates that this domain is related to malicious activity. Further Ltrace observations indicates that the attempt of writing a scheduled task to *cron* failed, since *root* permissions are required to perform this operation. Figure 6.9 indicates removal of the created *cron* file in the */tmp/* directory. This is likely a part of clearing tracks.

```
9747 feof(0x1378260)                              = 1
9747 strcat("* * * * * ", "/home/user/VirusShare_8f19484738"...) = "* * * * * /home/user/VirusShare_"...
9747 strcat("* * * * * /home/user/VirusShare_"..., " >/dev/null 2>&1\n") = "* * * * * /home/user/VirusShare_"...
9747 sprintf("@weekly wget http://stablehost.u"..., "@weekly wget http://stablehost.u"..., "/home/user/VirusShare_8f19484738"...,
"/home/user/VirusShare_8f19484738"..., "/home/user/VirusShare_8f19484738"...) = 228
9747 strcat("* * * * * /home/user/VirusShare_"..., "@weekly wget http://stablehost.u"...) = "* * * * * /home/user/VirusShare_"...
9747 fclose(0x1378260 <unfinished ...>
9747 SYS_close(4)                                 = 0
9747 <... fclose resumed> )                       = 0
9747 fopen("/tmp/cron", "a" <unfinished ...>
```

Figure 6.8: Possible tampering with **crontab** to schedule downloads from malicious domain.

```
9747 system("crontab /tmp/cron;rm -rf /tmp/cr"... <unfinished ...>
```

Figure 6.9: Deletion of previously created file.

A password, *authenticationpassword*, along with the username *NICK*, are further observed written to disk in 6.10. These credentials are also observed within the recorded network activity. The InetSim log data, as shown in figure 6.11, shows communication on port 25 (SMTP). Figure 6.12 shows a snippet from the *output.pcap* file, which might be used to further analyse captured network activity in wireshark.

```
9753 vsprintf(0x6068c0, 0x405b90, 0x7ffca05ff0e0, 0x1378490) = 82
9753 strlen("PASS authenticationpassword\nNICK"...) = 82
9753 write(4, "PASS authenticationpassword\nNICK"..., 82 <unfinished ...>
9753 SYS_write(4, "PASS authenticationpassword\nNICK"..., 82) = 82
```

Figure 6.10: Observed credentials are written to disk.

```
[2021-04-23 19:18:32] [430] [smtp_25_tcp 474] [192.168.102.132:43758] connect
[2021-04-23 19:18:32] [430] [smtp_25_tcp 474] [192.168.102.132:43758] send: 220 mail.inetsim.org INetSim Mail Service
[2021-04-23 19:18:33] [430] [smtp_25_tcp 474] [192.168.102.132:43758] recv: PASS authenticationpassword
[2021-04-23 19:18:33] [430] [smtp_25_tcp 474] [192.168.102.132:43758] send: 500 5.5.1 Error: unknown command
[2021-04-23 19:18:33] [430] [smtp_25_tcp 474] [192.168.102.132:43758] recv: NICK DLXZKDI
[2021-04-23 19:18:33] [430] [smtp_25_tcp 474] [192.168.102.132:43758] send: 500 5.5.1 Error: unknown command
[2021-04-23 19:18:33] [430] [smtp_25_tcp 474] [192.168.102.132:43758] recv: USER FTMWRLK localhost localhost :KAQSYW
[2021-04-23 19:18:33] [430] [smtp_25_tcp 474] [192.168.102.132:43758] send: 500 5.5.1 Error: unknown command
```

Figure 6.11: SMTP network activity logged by inetsim.

Source	Destination	Protocol	Length	Info
192.168.102.132	192.168.102.1	DNS	84	Standard query 0xcc89 A linksys.secureshellz.net
192.168.102.1	192.168.102.132	DNS	100	Standard query response 0xcc89 A linksys.secureshellz.net A 192.168.102.1
192.168.102.132	192.168.102.1	TCP	74	43758 → 25 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=3278359929 TSecr=0 WS=1
192.168.102.1	192.168.102.132	TCP	74	25 → 43758 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1951245408 T
192.168.102.132	192.168.102.1	TCP	66	43758 → 25 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=3278359931 TSecr=1951245408
192.168.102.1	192.168.102.132	SMTP	116	S: 220 mail.inetsim.org INetSim Mail Service ready.
192.168.102.132	192.168.102.1	TCP	66	43758 → 25 [ACK] Seq=1 Ack=51 Win=29312 Len=0 TSval=3278359938 TSecr=1951245417
192.168.102.132	192.168.102.1	SMTP	148	C: PASS authenticationpassword | NICK DLXZKDI | USER FTMWRLK localhost localhost :KAQSYW

Figure 6.12: Inspecting network activities in wireshark output.

6.4.2 Rootkit

As seen in figure 6.13, this sample was identified as Trojan.Linux.Rootkit by several engines on VirusTotal:

```
----------------------------------------
Malware Capabilities and classification using YARA rules:
        []
----------------------------------------
Virustotal:

          ALYac ==> Trojan.Linux.Rootkit.E
          AVG ==> ELF:Agent-FM [Rtk]
          Ad-Aware ==> Trojan.Linux.Rootkit.E
          AegisLab ==> Trojan.Linux.Agent.5!c
          AhnLab-V3 ==>
          Antiy-AVL ==> Trojan[Rootkit]/Linux.Agent.e
          Arcabit ==> Trojan.Linux.Rootkit.E
          Avast ==> ELF:Agent-FM [Rtk]
          Avast-Mobile ==>
          Avira ==> TR/RKit.Linux.A.E.2
          Baidu ==>
          BitDefender ==> Trojan.Linux.Rootkit.E
```

Figure 6.13: RootKit VirusTotal detection

In the dynamic analysis section, as shown in figure 6.14, the file outputs four options. This indicates that the file requires user input. The options given by the file are as follows:

- (L)ocal port configuation
- (R)emote configatuin
- (B)ackdoor password
- (H)idden program configuation
- (E)XIT

In figure 6.14 there is also the output "invalid option" and "system unfinished" which may occur due to the file not receiving any input from a user. There is also no indication of an internet connection being established. This sample does not act independently which means that a user needs to establish a remote connection to access the device with the file sample, or to access a device directly.

```
9740 printf("\nFucKit RK configuration tool by"... <unfinished ...>
9740 SYS_fstat64(2004, 0xffae6ebc, 0xf7df01a5, 0xf7ee0860) = 0
9740 SYS_brk(0xf7ee2000)                          = 0x9906000
9740 SYS_brk(0xf7ee2000)                          = 0x9927000
9740 SYS_brk(0xf7ee2000)                          = 0x9928000
9740 <... printf resumed> )                       = 41
9740 printf("= Version: %s\n\n", "0.2")           = 16
9740 printf("=================================" ...) = 48
9740 printf("[*] (L)ocal ports configuration." ...) = 33
9740 printf("[*] (R)emote ports configuration" ...) = 34
9740 printf("[*] (B)ackdoor password.\n")       = 25
9740 printf("[*] (H)idden programs configurat" ...) = 37
9740 printf("[*] (E)xit.\n")                     = 12
9740 printf("================================"...) = 48
```

Figure 6.14: Options for Root Kit

Rootkit no connection to host

The creator of the malware has removed the creator of the configuration tool, as seen at the top of figure 6.14. The rootkit configuration tool sample is called "FuckIt". No external IP addresses was in the logs, this may be because the malware was unsuccessful since this malware did not receive any valid input which may have resulted in the output "Invalid option" and "System("clear" <unfinished...>, as seen in the bottom of figure 6.14.

RootKit Loop

This malware has repeated itself 176 times in a run-time of 60s after not receiving any input, as shown in figure 6.15

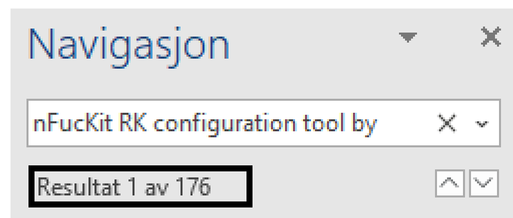


Figure 6.15: The FucKit RK script repeated itself 176 times

6.4.3 Privilege escalation

In the static analysis section VirusTotal, as seen in figure 6.16, returns the malware samples vulnerability details. The details described by the CVE returned by VirusTotal, CVE-2010-3301, are how the file attempts to manipulate an error which exists in the Linux kernel by triggering an out-of-bounds access to the system call table.

```
---------------------------------------
Malware Capabilities and classification using YARA rules:
        []
---------------------------------------
Virustotal:

          ALYac ==> Trojan.Linux.GenericA.54324
          AVG ==> ELF:Exploit-T [Expl]
          Acronis ==>
          Ad-Aware ==> Trojan.Linux.GenericA.54324
          AegisLab ==>
          AhnLab-V3 ==>
          Antiy-AVL ==> Trojan[Exploit]/Linux.CVE-2010-3301.c
          Arcabit ==> Trojan.Linux.GenericA.DD434
          Avast ==> ELF:Exploit-T [Expl]
          Avast-Mobile ==>
```

Figure 6.16: The figure shows the VirusTotal output from the static analysis

As seen in the dynamic analysis section, in figure 6.17, under call trace activities, which records the program activity using Ltrace, the activities performed by the file can be inspected. Here the program starts by allocating memory with the *malloc* function. That a file allocates memory is normal, but in this instance there is an abnormal amount of lines designated to memory allocation, the sequence shown in figure 6.17 repeats several times in the dynamic analysis section.

```
9748 malloc(32)                                         = 0x556bacefc320
9748 __ctype_b_loc()                                    = 0x7fbaa82fd4d8
9748 __ctype_b_loc()                                    = 0x7fbaa82fd4d8
9748 __ctype_b_loc()                                    = 0x7fbaa82fd4d8
9748 __ctype_b_loc()                                    = 0x7fbaa82fd4d8
9748 __ctype_b_loc()                                    = 0x7fbaa82fd4d8
9748 malloc(32)                                         = 0x556bacefc350
9748 __ctype_b_loc()                                    = 0x7fbaa82fd4d8
9748 __ctype_b_loc()                                    = 0x7fbaa82fd4d8
9748 __ctype_b_loc()                                    = 0x7fbaa82fd4d8
9748 __ctype_b_loc()                                    = 0x7fbaa82fd4d8
9748 __ctype_b_loc()                                    = 0x7fbaa82fd4d8
9748 __ctype_b_loc()                                    = 0x7fbaa82fd4d8
9748 __ctype_b_loc()                                    = 0x7fbaa82fd4d8
9748 __ctype_b_loc()                                    = 0x7fbaa82fd4d8
9748 malloc(32)                                         = 0x556bacefc380
```

Figure 6.17: malloc function example from call trace

At the end of the call trace section of the dynamic analysis the file exits without any errors. By inspecting the Ltrace outputs at the end of the file it is possible that the program finishes successfully. In this case the possibility of getting an indication regarding the program gaining any escalated privileges is low, but since the program prints "*Process finished*" at the end of the call trace output it is possible that it has finished successfully.

Chapter 7

Discussion

This chapter further discusses the results and findings from chapter 6 *Analysis and testing*. Further work and recommended measures will be discussed, along with recommended additions, modifications and fixes that might improve the implementation.

7.1 Unmet functional requirements

As specified in chapter 2 *Requirements*, several functional requirements are required in order to successfully solve the task and answer the problem statement. Table 7.1 repeats these requirements, stating which ones are considered covered.

Functional requirement	**Achieved**
Internet access	Yes
Disk access (I/O)	Yes
Library call tracing	Yes
Packer detection	Yes
Output in human-readable format	Yes
Display analysis output in the terminal	Yes
Store analysis output in files	Yes
Creating an isolated sandbox environment for secure and stealthy ELF-malware execution.	Yes
Logging of executed code or Assembly instructions	No

Table 7.1: Representation of whether functional requirements has been covered.

Considering table 7.1 above, one of the functional requirements has not been implemented as part of the final solution. Research and concepts regarding this requirement will be further discussed throughout this section.

7.1.1 Logging of executed code or Assembly instructions

Obfuscating malware is done in order to confuse analysts or victims by making textual and binary data more difficult to read and understand [24]. The term is further discussed in chapter 3. Obfuscation techniques might make static analysis useless, as the analyst is not able to properly extract static information regarding the software in human-readable format. In some cases, obfuscation might lead to difficulties in debugging malware during run-time (dynamic analysis) as well.

The principle of malware packing is a subset of obfuscation. A packer is a piece of software that compresses a malware sample in order to make its code and data unreadable [24]. At run-time, the packed program is decompressed in memory using a wrapper program. Please be advised that packers have legitimate use cases as well, for instance compressing software to more acceptable sizes [24]. Threat actors often abuse these packers to avoid detection and behaviour identification. There are several packer software available, as for instance *UPX, Themida* or *The Enigma Protector*.

Conducting dynamic analysis on packed malware samples introduces difficulties, as debugging tools analysing the sample (e.g. Strace) might fail executing a compressed sample. Packed samples need to be unpacked in order to be analysed properly, and might only be unpacked by using the same software which packed it. This introduces the need for identifying which packer has been used to pack the sample. This implementation of Limon does have the ability to detect potential known packers using Yara-rules. If the packer used to pack the sample is correctly identified, the malware analyst might manually unpack the sample using the identified packer software (e.g. *UPX*). Thus, the analyst might run a new analysis in Limon using the unpacked sample to get accurate run-time results.

Please be advised that sophisticated malware might benefit custom developed packers not publicly available. In such cases, the security analyst must manually reverse engineer the binary in order to identify how it has been packed. [24]. Thus, logging of executed code or Assembly instructions being carried out might be useful in order to determine what is happening behind the curtains. Successfully obtaining this knowledge in such cases are key to unpack the sample for further analysis.

In some cases, sophisticated malware might also combine the usage of a custom developed software packer in addition to a popular one (e.g. *UPX*) [24]. Limon lacks the ability to automatically unpack packed malware samples. Implementation of such functionality has been attempted in this thesis without success. Suggestions for how this might be implemented are briefly discussed down the line in chapter 7.3, *Improvements and further work*.

7.2 Useful malware indicators in ELF-files

This section will further discuss indicators of importance regarding malware in ELF-files. Both static and dynamic parameters useful to identify malware are considered.

7.2.1 Symbols

Symbols describe data types such as functions and variables stored in the source code, which might be exported for debugging and linking purposes [87]. By inspecting symbols, an analyst might uncover which functions and variables has been created by the developer, thus gaining a better understanding regarding the binary's functionality. Some of these findings might also be searched online in open sources to find existing indicators. The *Readelf* command combined with the *-s* argument is useful for examining an ELF-file's symbol information.

When inspecting symbol information, one will most likely come across tables called *.dynsym* and *.symtab*. *.dynsym* contains dynamically linked symbols, while the *.symtab* table contains all symbols, including the dynamically linked ones [87]. The difference between statically- and dynamically-linked binaries are defined in chapter 3.3.9. It is important to inspect these symbol tables, as they may reveal function calls or variables of importance which might substantiate a hypothesis of potential malware. Some of the most important symbol types include the following:

- **OBJECT:** Declared global variables.
- **FUNC:** Declared functions.
- **FILE:** Source files compiled into the library. This is also a debug symbol, meaning that stripped binaries won't contain these symbol types.

The analyst should also bear in mind that statically linked binaries are usually bigger in file size, as they most likely contain larger amounts of code related to the dependant libraries and not the actual logic of the file [87]. This may make it harder to understand the malware's purpose and functionality, compared to dynamically linked binaries where most of the file's contents are related to it's logic.

7.2.2 Segments

Segments (also known as program headers [87], further described in chapter 3) are important to examine for potential anomalies. If anomalies are discovered in the segment structure, it might indicate that the file has been modified, infected or packed. Figure 6.4 displays an example of such a case which was discovered during the Limon functionality test. The dynamic analysis process for this particular malware sample were not successful as no call trace activity were recorded. This substantiates the hypothesis that the sample potentially is packed, or that the

segments structure has been tampered with.

In short, information about sections might be useful for an analyst to examine in order to determine knowledge regarding the following:

- Identifying entry points and other useful information regarding the executable code.
- Mapping between segments and sections.
- Discover anomalies in the ELF-header structure.
- Identify whether the file has been stripped.
- Identify whether the file has been infected or potentially packed.

7.2.3 Run-time indicators

There are also certain run-time indicators to be aware of when analysing ELF-malware. The */tmp/* directory in Linux is widely used by malicious programs as a directory for creating or downloading files from the internet. As the directory has global read and write permissions regardless of which user carries out the desired operation, */tmp/* is a natural directory for performing malicious operations. A threat actor having obtained a foothold in the system, might abuse this directory for his or hers advantage.

Abnormal network activities should also be examined if suspicion regarding ELF-malware has been raised. In enterprise infrastructure, on-premise Linux servers often serve specific services, as for instance a mail server, web server or storage. Depending on the internal configuration, a mail server might only expect communication over SMTP port 25. If a running ELF-process on the mail server system starts transmitting abnormal network traffic on other network ports, it might indicate potential malicious activity residing from the process. Malicious activity might also take place in the expected service, and thus proper monitoring of network services in Linux are necessary in order to track down potential ELF-compromises.

ELF-malware often tends to tamper with Linux's task scheduled system, also known as *crontab*. Abnormalities in *crontab* might indicate that a Linux-malware has scheduled malicious tasks, such as retrieving commands from a Command and Control Server-server or perform other malicious operations regularly. Figure 6.8 shows how a malware sample from the functionality test, mentioned in chapter 6, tries to tamper with *crontab* in order to weekly perform downloads from a malicious website.

7.3 Improvements and further work

The following section covers recommended further work for this thesis implementation and mentions potential for improvements. The technical suggestions

mentioned are divided in three parts: *Additions, Modifications* and *Fixes*.

7.3.1 Additions

Several additions to improve Limon and this thesis implementation has been specified below. These include additions to cover unmet functional requirements and features considered as relevant to support the project in general.

Unpacking function

Limon lacks the feature to unpack potentially packed malware samples. The implementation does, however, have the ability to detect potential packers using Yara. To further extend Limon's capabilities, the packer detection results might be used to unpack the sample for further analysis. By installing the most popular packers on the host OS, a Python *switch statement* might unpack the sample based on the detection returned from Yara. Thus, if the packed sample is unpacked successfully using the detected packer software, Limon can continue with it's normal procedures. Code listing 7.1 suggests a pseudo-code example which may be used to implement this functionality.

Code listing 7.1: Pseudo-code for automatic unpacking solution for Limon in *statan.py*

```
# statan.py

...

def check_for_packer(malware_sample):
        packer = run_yara_comparison(malware_sample)
        return packer

if (result = check_for_packer(malware_sample)):
        unpacked_sample = ""
        if (result == "UPX"):
                unpacked_sample = upx_unpack(malware_sample)
        if (result == "Themida"):
                unpacked_sample = themida_unpack(malware_sample)

        ...

        if (unpacked_sample):
                run_analysis(unpacked_sample)
        else:
                print("Potential unknown packer detected")

else:
        print("No packer detected")

...
```

During the functionality test, packer detection using Yara delivered mixed results. This might for instance occur as a results of an outdated Yara rule-set. To

ensure proper packer detection, it is advisable to implement an additional packer detection technique. *Detect It Easy* (DiE) [89] is a signature-based packer detection tool which detects a file's compiler, linker and packer. The tool is available for Linux and can be downloaded from their official GitHub repository [90]. By implementing another packer detection method, such as *DiE*, the likelihood for Limon to detect potential packing of samples is greatly increased by using different sources.

Logging of executed code or Assembly instructions

Logging the Assembly instructions that a malware carries out might be useful in cases were the malware sample is packed with an unknown packer in order to successfully unpack the sample. The feature might also provide more enrich analysis output to support a hypothesis regarding what the malware is doing.

The Radare2 framework [91], as further described in chapter 4, might be used to conduct analysis on Assembly level. The R2pipe [72] API makes it possible to script Radare2 operation, thus making it possible to perform dynamic analysis automatically. This requires an in-depth understanding of Assembly, and R2pipe is also poorly documented.

Please be advised that this thesis implementation do have some approaches considered relevant for this functional requirement. Strace and Ltrace does record information regarding executed code in terms of system and library codes. These logging techniques illustrates system and program operations accurately on their own. Based on this, one might argue that the functional requirement for logging executed code is covered to some extent.

Visual indicators and GUI

This implementation of Limon provides output from several analysis tools and methods, which helps security analyst to form a hypothesis and investigate malware behaviour. Although the analysis output files provides useful and informational results, there are no visual or concrete malware classifier such as graphs, entropy or "confidence score". As mentioned in chapter 1.4.2 *Delimitations*, this is out of scope for this thesis. However, this might become a handy feature in Limon when implemented in a SOC for instance.

7.3.2 Modifications

Running Strace and Ltrace simultaneously

As mentioned repeatedly throughout this thesis, both Strace and Ltrace has been implemented in order to trace both system- and library calls that the malware carries out during execution. However, currently there are no way for executing

a malware sample with both tools simultaneously. Please consider the use case below:

Jim wants to analyse a malware sample with Limon using a 60 second timeout, tracing both system- and library calls. Jim is aware that Ltrace is able to trace both call types, but wishes to use Strace to trace system calls for accuracy reasons. In order to retrieve both types of results, Jim must run two separate analysis on the same sample, doubling the conduction time.

The problem exists since the sample is ran using one of the tools within the sandbox. The sample to analyse acts as an arguments for either Strace or Ltrace. There are several approaches to this issue. For instance, an additional sandbox instance might be added, which runs simultaneously with the already existing one. Using duplicate instances, one could have been configured to run Ltrace while the other one conducts analysis using Strace, thus generating reports from both tools in one analysis. However, this is more resource heavy and reserves more storage space in terms of snapshots and VM images.

The more suitable approach for this case, is probably to eliminate the ability for Ltrace to trace system calls. Thus, a Limon argument entered by the user might be used to determine whether to use both Ltrace and Strace. Based on whether this argument is chosen, an if-statement might sequentially perform both operations if desired. This doubles the time it takes to conduct dynamic analysis, but decreases the amount of required resources and user interaction.

Hardware and virtualisation

Throughout this thesis implementation, nested virtualisation have been used as the underlying system infrastructure. Many computers does not have hardware support for nested virtualisation, while others might need to enable this feature in the computer's BIOS settings. The latter option might also be considered as a policy breach on work computers in some organisations.

The hardware used for the host system in this thesis includes the following:

- 8 Virtual CPU cores.
- 40 GB of disk space.
- 16 GB of RAM.

There are no specified hardware requirements for Limon to run properly. In the case of nested virtualisation, sufficient hardware should be ensured. The recommended approach for setting up Limon is to use a dedicated desktop with Ubuntu as the operating system. This will ensure proper performance for the sandbox, and is less dependant on the physical hardware. Having Limon installed on a physical laptop might also be the most convenient alternative in an incident response or digital forensics case.

Migrating from Python2 to Python3

Limon is written in Python2, which is discontinued and obsolete. In terms of building new functionalities on top of Limon, re-writing the code to Python3 might be more convenient for future development. As most analysis tools built with Python today are built with version 3, interference in interaction between Limon and an analysis tool is possible. Updating to Python3 is also important in order to mitigate risk regarding potential vulnerabilities in older versions of Python.

A safer internet mode

As described in 4, Limon includes an *internet mode* functionality by default. This feature was not implemented in this thesis implementation in order to create the safest and most isolated sandbox environment possible. Allowing sophisticated malware samples to access the internet during dynamic analysis might expose the threat actor to information about the analyst.

If *internet mode* is to be implemented, there are measures that might help mitigating the risk of exposure. For instance, by configuring *proxy chains* in the sandbox, internet traffic residing from the malware are routed through several proxy servers before reaching the end destination. This adds some layers of anonymity for the analyst compared to the default *internet mode* in Limon. An alternative might also be to route all internet traffic redesign from the malware through the TOR-network [92]. Regardless of which, performing dynamic analysis of sophisticated malware in an offline environment is always recommended if possible.

Changing default *sudo* password

As mentioned in chapter 5, an Ubuntu 18.04 LTS .vmx image from *TrendSigma* [78] was used as the guest OS. This particular image's default user and password consisted of *user:password*, which in many cases are typical default credentials. Such credentials are easily guessed by some malware to run programs with higher privileges. Depending on the scenario the analyst wishes to simulate, changing default credentials might be key to avoid this outcome. However, this might not be the case if the analyst wishes to recreate a scenario where default credentials has been used.

7.3.3 fixes

This section will describe recommended fixes regarding the Limon implementation. This includes fixes for this particular thesis implementation, as well as Limon in general.

Sysdig implementation

Sysdig, which is a system monitoring tool for Linux, is not part of this particular thesis implementation of Limon. As a system monitoring tool, it records both system and network activity at a detailed level. Since other tools in this implementation also records such activities, Sysdig is considered as a *good to have* tool. Thus, the lack of Sysdig's presence does not affect any functional requirement. In some digital forensics or incident response scenarios, the implementation of Sysdig might be considered relevant, as verification of malware behaviour by several tools are required in order to conduct a valid hypothesis.

Volatility implementation

Memory analysis is out of scope for this thesis, and thus no tool for conducting analysis of memory images has been implemented. In an incident response or digital forensics scenario, memory analysis becomes highly relevant. A majority of malware involves processes running in memory without leaving traces on the disk, such as meterpreter shells. Implementing Volatility to conduct memory analysis might support the findings from the static- and dynamic analysis processes in terms of hypothesis conduction in these scenarios.

Analysis reporting directories

When conducting an analysis with Limon, a directory named after the analysed sample is created, containing the results and output files. If an analysis of the same sample is conducted, the previous directory and all containing files will be removed and replaced with the new analysis results. This becomes an issue in cases where the analyst wishes to produce results from both Ltrace and Strace. Since this has to be performed in two separate operations, the results from the second analysis will overwrite the results from the first one. This leaves the analyst with either Ltrace or Strace report, based on which one was conducted prior to the other.

In order to solve this issue, an if-statement might be added to *dynan.py*, checking whether the Strace or Ltrace output file exists within the directory already. If so, it might be copied to another directory (e.g. */tmp/*) before the analysis is conducted. Upon ended analysis, the file can be copied back to the newly created results directory, and the analyst thus possess both Strace and Ltrace output.

Updating analysis tools

Since Limon has not been updated since 2016, the version of the analysis tools mentioned in the documentation [16]. This involves that some installation instructions are obstacle, as some tools have been updated along the way individually. The BitBucket repository developed in this thesis automatically installs newer ver-

sions of the necessary tools needed for Limon function properly. The user might clone this repository to automatically install the prerequisites needed. Please be advised that this repository will not be maintained, and the user needs to manually update the individual analysis tools if preferred. However, since the installation has been scripted, the implementation should work regardless as long as the respective tool-versions are not taken down by their developers.

7.4 The importance of combined analysis

The main focus of this thesis evolves around dynamically analysing ELF-files during execution to observe run-time behaviour and identify possible malware indicators. Although dynamic analysis is considered the main focus area, it is also important to stress the importance of combining different analysis methods and comparing the results. This is necessary in order to conduct an accurate hypothesis regarding what the potential ELF-malware sample undertakes. While dynamic analysis might be one of the most accurate analysis methods, considering the fact that the malware is actually executed, static- and memory analysis provides useful information to confirm these observations. This is especially important for malware analysis in Linux systems, as this is still a relatively new concept [81].

7.5 Protecting systems against ELF-malware infection

The ELF-file format is most used within Linux clients and servers. Thus these devices are naturally the most vulnerable systems in terms of ELF-malware. To avoid having a Linux system infected by ELF-malware, one must firstly secure the system for initial vectors of compromise. In Windows systems, malware is often delivered through Phishing emails as an attachment. This type of malware delivery is not as important when it comes to Linux system, because of the lack of GUIs. Therefore, other initial attack vectors should be considered when mitigating the risk regarding ELF-malware. The most important initial infection methods include the following [81]:

- Known vulnerabilities or zero-days on publicly faced components.
- Default software credentials.
- Breached or compromised credentials.
- Supply chain attacks or trusted third party relationship abuse.

Chapter 8

Conclusion

The following chapter will give the reader an overview of the thesis project. Furthermore, the chapter shortly describes achievements, how these achievements were reached, followed by an overall learning outcome.

8.1 Project assessment

The project task started out as an open project description with the possibility to make adjustments before it was finalised. Project client Lasse Øverlier at NTNU wanted us to create a solution capable of conducting automated dynamic analysis of ELF-files, producing relevant output which might help identify potential indicators of malware. The requirements regarding which output that was relevant were clearly specified. Apart from this, the task was very flexible in terms of what we as a group consider relevant for solving the problem.
The thesis project started out by investigating the ELF-file in detail. We learned about it's structure and area of usage through various articles, Youtube videos and *Capture the Flag* challenges. Although technical knowledge regarding how the ELF-file is structured is considered out of scope for this thesis, we felt that this was a nice to know approach of some importance when creating the solution.
Requirement specification were carried out, and we early decided to angle this thesis against analysis of sophisticated malware samples, as malware in Linux systems are still in an early stage. We learned about some best practices in dynamic malware analysis in general, and decided to create a sandbox isolated from both the host system and the internet. Thus the hunt began for finding existing technologies which would help us along the way. Several existing frameworks were visited, some were also tested. After some research we stumbled across Limon, which turned out to be one of the few sandbox solutions capable of conducting malware analysis of ELF-files.
After designing the systems infrastructure, the technical implementation was carried out. Being discontinued since 2016, implementing Limon proved to be a tedious and cumbersome process. Thus, the whole process was scripted and uploaded to our own BitBucket repository, which is publicly available for anyone to use. To

test the implementation, project client Lasse Øverlier handed us a handful of ELF-malware samples. A functionality test of 100 samples was conducted, providing useful results which were further discussed at the end of the report.

8.2 Learning outcome and evaluation

During the thesis project period we have gained knowledge regarding several topics within information security, especially malware analysis and Linux binaries. We have learned about how malicious files threaten the Linux landscape, and which classifiers are important in order to determine the possibility of malware in an ELF-binary. Furthermore, we have gained more experience in debugging suspicious programs using different tools, which we consider relevant for our future work life.

When looking at the outcome in retrospect, we have several things in mind regarding what we could have done better. For instance, we could have arranged more continuous meetings with the project client during the project period. The group working sessions should have been arranged more frequently. Working together as a group turned out to be challenging during the COVID-19 pandemic, combined with other factors such as different working hours. Despite these facts, the Kanban methodology has worked great for this type of project affected by a lot of individual work. The decision to choose Limon as the analysis solution should have happened earlier, in order to successfully implement more of the features mentioned in chapter 7.3

8.3 Results

Throughout this thesis we have developed a solution to conduct both static- and dynamic analysis in an isolated sandbox environment. The solution is capable of collecting information on how an ELF-file behaves during execution, in terms of system calls, I/O, library calls, and network activities. We have also discussed common observations in ELF-malware. The time frame was unfortunately a little bit too short to implement all of the features requested by the project client. However, some features do arguably cover some of the missing requests.

We have certainly learned a lot through this thesis, both in terms of technical knowledge and self-knowledge. In total, we are satisfied about the achieved results in regards to the problem statement. Lastly, we hope that the outcome of this thesis might contribute towards important malware research in an area that is still considered young in the cyber threat landscape.

Bibliography

[1] (2021). 'Assembly language,' [Online]. Available: https://techterms.com/definition/assembly_language (visited on 16/05/2021).

[2] (2021). 'Command and control [cc] server,' [Online]. Available: https://www.trendmicro.com/vinfo/us/security/definition/command-and-control-server (visited on 09/05/2021).

[3] (2020). 'Cuckoo sandbox book,' [Online]. Available: https://cuckoo.sh/docs/ (visited on 09/05/2021).

[4] (2021). 'Definition of 'debugging',' [Online]. Available: https://economictimes.indiatimes.com/definition/debugging (visited on 08/05/2021).

[5] B. Lutkevich. (2021). 'Embedded system,' [Online]. Available: https://internetofthingsagenda.techtarget.com/definition/embedded-system (visited on 08/05/2021).

[6] 'What is an exploit?,' [Online]. Available: https://www.cisco.com/c/en/us/products/security/advanced-malware-protection/what-is-exploit.html (visited on 20/05/2021).

[7] Cisco, 'Firewall,' [Online]. Available: https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-firewall.html (visited on 08/05/2021).

[8] M. Kassner. (2011). 'Fuzzy hashing helps researchers spot morphing malware,' [Online]. Available: https://www.techrepublic.com/blog/it-security/fuzzy-hashing-helps-researchers-spot-morphing-malware/ (visited on 07/04/2021).

[9] R. Lanigan. (2016). 'Guest operating system,' [Online]. Available: https://searchservervirtualization.techtarget.com/definition/guest-OS (visited on 09/05/2021).

[10] (2021). 'Hash,' [Online]. Available: https://techterms.com/definition/hash (visited on 09/05/2021).

[11] S. Kenlon, 'How hexdump works,' 2019-08-12. [Online]. Available: https://linux-audit.com/elf-binaries-on-linux-understanding-and-analysis/ (visited on 20/05/2021).

[12] N. Martin. (2016). 'Host operating system,' [Online]. Available: https://searchvmware.techtarget.com/definition/host-operating-system (visited on 09/05/2021).

[13] V. Inc. (2021). 'Hypervisor,' [Online]. Available: https://www.vmware.com/topics/glossary/content/hypervisor (visited on 08/05/2021).

[14] N. Lord, 'What are indicators of compromise?,' 2020. [Online]. Available: https://digitalguardian.com/blog/what-are-indicators-compromise (visited on 09/05/2021).

[15] M. Kerrisk. (2021). 'Ldd - list dynamic dependencies,' [Online]. Available: https://man7.org/linux/man-pages/man1/ldd.1.html (visited on 02/05/2021).

[16] M. K. A. (2015). 'Setting up limon sandbox for analyzing linux malwares,' [Online]. Available: http://malware-unplugged.blogspot.com/2015/11/setting-up-limon-sandbox-for-analyzing.html (visited on 10/05/2021).

[17] J. Cespedes. (2019). 'Ltrace lystem call debugger tool,' [Online]. Available: https://man7.org/linux/man-pages/man1/ltrace.1.html (visited on 04/04/2021).

[18] (2020). 'Lubuntu - about,' [Online]. Available: https://lubuntu.net/about/ (visited on 09/05/2021).

[19] 'What is malware?,' [Online]. Available: https://www.cisco.com/c/en/us/products/security/advanced-malware-protection/what-is-malware.html (visited on 09/05/2021).

[20] Volatility. (2021). 'Volatility workbench,' [Online]. Available: https://www.osforensics.com/tools/volatility-workbench.html (visited on 08/05/2021).

[21] (2016). 'Run hyper-v in a virtual machine with nested virtualization,' [Online]. Available: https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/user-guide/nested-virtualization (visited on 09/05/2021).

[22] (2021). 'What is open source?' [Online]. Available: https://opensource.com/resources/what-open-source (visited on 08/05/2021).

[23] (2021). 'Understanding openstack,' [Online]. Available: https://www.redhat.com/en/topics/openstack/ (visited on 16/05/2021).

[24] T. Appleby. (2019). 'Analysing packed malware,' [Online]. Available: https://resources.infosecinstitute.com/topic/analyzing-packed-malware/ (visited on 16/05/2021).

[25] A. Clark. (2021). 'Pillow - python imaging library,' [Online]. Available: https://pillow.readthedocs.io/en/stable/ (visited on 04/04/2021).

[26] C. I. S. AGENCY. (2021). 'Ransomware,' [Online]. Available: https://www.cisa.gov/ransomware (visited on 09/05/2021).

[27] F. S. Foundation. (2009). 'Readelf man page,' [Online]. Available: https://linux.die.net/man/1/readelf (visited on 02/05/2021).

[28] Z. S. Corp. (2020). 'Remnux: A linux toolkit for malware analysis,' [Online]. Available: https://docs.remnux.org/ (visited on 20/02/2021).

[29] L. Rosencrance, 'Sandbox (software testing and security),' December 2018. (visited on 20/03/2021).

[30] M. Courtemanche. (2012). 'Vmware snapshot,' [Online]. Available: https://searchvmware.techtarget.com/definition/VMware-snapshot/ (visited on 16/05/2021).

[31] J. Kornblum. (2018). 'Ssdeep - fuzzy hashing program,' [Online]. Available: https://ssdeep-project.github.io/ssdeep/index.html (visited on 07/04/2021).

[32] (2021). 'Ssh (secure shell),' [Online]. Available: https://www.ssh.com/academy/ssh (visited on 09/05/2021).

[33] (2021). 'Static application security testing,' [Online]. Available: https://www.synopsys.com/glossary/what-is-sast.html (visited on 09/05/2021).

[34] (2021). 'Strace system call debugger tool,' [Online]. Available: https://linux.die.net/man/1/strace (visited on 02/05/2021).

[35] T. O. Group. (2018). 'Strings utility,' [Online]. Available: https://pubs.opengroup.org/onlinepubs/9699919799/utilities/strings.html (visited on 02/05/2021).

[36] (2021). 'About sysdig,' [Online]. Available: https://sysdig.com/about/ (visited on 08/05/2021).

[37] T. T. Group. (2021). 'Tcpdump/libpcap public repository,' [Online]. Available: https://www.tcpdump.org/index.html#documentation (visited on 20/02/2021).

[38] N. I. of Standards and T. S. P. 800-150. (2016). 'Threat actor,' [Online]. Available: https://doi.org/10.6028/NIST.SP.800-150 (visited on 09/05/2021).

[39] C. Edu. (2020). 'Threat intelligence defined,' [Online]. Available: https://www.forcepoint.com/cyber-edu/threat-intelligence (visited on 09/05/2021).

[40] tshark. (). 'Tshark - dump and analyze network traffic,' [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html (visited on 20/05/2021).

[41] A. Grace. (2020). 'What is a computer virus?' [Online]. Available: https://us.norton.com/internetsecurity-malware-what-is-a-computer-virus.html (visited on 23/07/2021).

[42] Virustotal. (). 'Virustotal - how it works,' [Online]. Available: https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works (visited on 08/05/2021).

[43] (2016). 'Controlling vmware virtual machines from the command line with vmrun,' [Online]. Available: https://www.virtuatopia.com/index.php?title=Controlling_VMware_Virtual_Machines_from_the_Command_Line_with_vmrun (visited on 16/05/2021).

[44] (2019). 'Vmware,' [Online]. Available: https://searchvmware.techtarget.com/definition/VMware (visited on 16/05/2021).

[45] 'About wireshark,' [Online]. Available: https://www.wireshark.org/ (visited on 20/05/2021).

[46] VirusTotal. (2021). 'Yara - the pattern matching swiss nife for malware researchers,' [Online]. Available: https://yara.readthedocs.io/en/stable/ (visited on 10/03/2021).

[47] C. Cimpanu. (2020). 'Linux version of ransomexx ransomware discovered,' [Online]. Available: https://www.zdnet.com/article/linux-version-of-ransomexx-ransomware-discovered/ (visited on 13/03/2021).

[48] S. Gatlan. (2021). 'Chinese state hackers target linux systems with new malware,' [Online]. Available: https://www.bleepingcomputer.com/news/security/chinese-state-hackers-target-linux-systems-with-new-malware/ (visited on 13/03/2021).

[49] F. International, 'Malware is a massive risk and it's everyone's problem!,' 2020-01-30. [Online]. Available: https://linux-audit.com/elf-binaries-on-linux-understanding-and-analysis/ (visited on 10/03/2021).

[50] 'What is a trojan? - definition and explanation,' [Online]. Available: https://www.kaspersky.com/resource-center/threats/trojans (visited on 22/04/2021).

[51] D. Rafter, 'What is a rootkit? and how to stop them,' [Online]. Available: https://us.norton.com/internetsecurity-malware-what-is-a-rootkit-and-how-to-stop-them.html (visited on 22/04/2021).

[52] 'What is scareware?,' [Online]. Available: https://www.kaspersky.com/resource-center/definitions/scareware (visited on 22/04/2021).

[53] 'What is spyware? and how to remove it,' [Online]. Available: https://us.norton.com/internetsecurity-how-to-catch-spyware-before-it-snags-you (visited on 22/04/2021).

[54] F. Hofmann, 'Understanding the elf file format?,' [Online]. Available: https://us.norton.com/internetsecurity-malware-what-is-a-computer-worm.html (visited on 22/04/2021).

[55] M. Sikorski and A. Honig, 'Practical malware analysis,' 2012. [Online]. Available: http://venom630.free.fr/pdf/Practical_Malware_Analysis.pdf (visited on 15/02/2021).

[56] F. Hofmann, 'Understanding the elf file format?,' [Online]. Available: `https://linuxhint.com/understanding_elf_file_format` (visited on 19/02/2021).

[57] https://refspecs.linuxfoundation.org/, 'Elf header,' [Online]. Available: `https://refspecs.linuxfoundation.org/elf/gabi4+/ch4.eheader.html` (visited on 02/03/2021).

[58] M. Boelent, 'He 101 of elf files on linux: Understanding and analysis,' 2019-05-15. [Online]. Available: `https://linux-audit.com/elf-binaries-on-linux-understanding-and-analysis/` (visited on 21/02/2021).

[59] 'Unix system v,' 12.04.2021. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=UNIX_System_V&oldid=1017325059` (visited on 15/03/2021).

[60] S. Microsystems, 'Cddl header start,' [Online]. Available: `https://opensource.apple.com/source/dtrace/dtrace-90/sys/elf.h` (visited on 15/03/2021).

[61] 'Executable and linkable format,' 2020-09-29. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Executable_and_Linkable_Format&oldid=1017977721` (visited on 27/03/2021).

[62] H. Arora, 'Elf virus, part i,' 2012-01-x. [Online]. Available: `https://johnvidler.co.uk/linux-journal/LJ/213/11185.html` (visited on 12/02/2021).

[63] Yobot, 'Silvio cesare,' 2021-01-15. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Silvio_Cesare&oldid=1000477430` (visited on 21/02/2021).

[64] N. Lord, 'What are memory forensics? a definition of memory forensics),' 2020-09-29. [Online]. Available: `https://searchsecurity.techtarget.com/definition/sandbox` (visited on 27/02/2021).

[65] (2021). 'Virtual machine migration (vm migration),' [Online]. Available: `https://www.techopedia.com/definition/15033/virtual-machine-migration-vm-migration` (visited on 20/05/2021).

[66] T. Inc. (2021). 'Obfuscation,' [Online]. Available: `https://www.techopedia.com/definition/16375/obfuscation` (visited on 13/05/2021).

[67] M. K. A. (2015). 'Limon github repository,' [Online]. Available: `https://github.com/monnappa22/Limon` (visited on 18/03/2021).

[68] M. K. A. (2015). 'Automating linux malware analysis using limon sandbox,' [Online]. Available: `https://www.blackhat.com/eu-15/briefings.html#automating-linux-malware-analysis-using-limon-sandbox` (visited on 10/05/2021).

[69] T. H. M. Eckert. (2020). 'Inetsim information page,' [Online]. Available: `https://www.inetsim.org/about.html` (visited on 29/03/2021).

[70] C. P. (2019). 'System call definition,' [Online]. Available: `https://techterms.com/definition/system_call` (visited on 02/05/2021).

[71] (2018). 'Irma: Incident response & malware analysis,' [Online]. Available: https://irma.readthedocs.io/en/latest/ (visited on 15/02/2021).

[72] 'R2pipe official website,' [Online]. Available: https://rada.re/n/r2pipe.html.

[73] Gantt.com. (). 'What is a gantt chart?' [Online]. Available: https://www.gantt.com/ (visited on 05/03/2021).

[74] E. T. Vestad. (2021). 'Thesis bitbucket repository,' [Online]. Available: https://bitbucket.org/espur/dynamisk-elf-analyse/src/master/.

[75] 'Skyhigh documentation,' [Online]. Available: https://www.ntnu.no/wiki/display/skyhigh (visited on 09/02/2021).

[76] 'Oracle vm virtualbox,' [Online]. Available: https://www.virtualbox.org/ (visited on 09/02/2021).

[77] D. Tucakov. (2019). 'Installing desktop gui on an ubuntu server,' [Online]. Available: https://phoenixnap.com/kb/how-to-install-a-gui-on-ubuntu (visited on 22/02/2021).

[78] 'Ubuntu 18.04 lts vmware image,' [Online]. Available: http://www.trendsigma.net/vmware/ubuntu1804t.html (visited on 09/02/2021).

[79] 'Yara-rules repository,' [Online]. Available: https://github.com/Yara-Rules/rules (visited on 10/03/2021).

[80] 'Virusshare malware samples,' [Online]. Available: https://virusshare.com/ (visited on 06/04/2021).

[81] A. Mechtinger, 'Elf malware analysis 101: Linux threats no longer an afterthought,' 2020-06-16. [Online]. Available: https://www.intezer.com/blog/malware-analysis/elf-malware-analysis-101-linux-threats-no-longer-an-afterthought/ (visited on 24/02/2021).

[82] Rapid7, 'About post-exploitation,' [Online]. Available: https://docs.rapid7.com/metasploit/about-post-exploitation/ (visited on 19/05/2021).

[83] 'Cobalt strike official website,' [Online]. Available: https://www.cobaltstrike.com/.

[84] (2021). 'Mimikatz official github repository,' [Online]. Available: https://github.com/gentilkiwi/mimikatz/wiki (visited on 19/05/2021).

[85] A. Viescas. (). 'Linux segmentation faults,' [Online]. Available: https://smallbusiness.chron.com/segmentation-fault-linux-27699.html (visited on 08/05/2021).

[86] (2010). 'What is a segmentation fault on linux?' [Online]. Available: https://stackoverflow.com/questions/3200526/what-is-a-segmentation-fault-on-linux (visited on 08/05/2021).

[87] malware.news. (2020). 'Elf malware analysis 101 part 2: Initial analysis,' [Online]. Available: https://malware.news/t/elf-malware-analysis-101-part-2-initial-analysis/42520 (visited on 14/05/2021).

[88] (2019). 'Crontab in linux with examples,' [Online]. Available: https://www.geeksforgeeks.org/crontab-in-linux-with-examples/ (visited on 08/05/2021).

[89] 'Die - detect it easy,' [Online]. Available: https://horsicq.github.io/.

[90] (2021). 'Die engine repository,' [Online]. Available: https://github.com/horsicq/DIE-engine/releases.

[91] 'Radare2 official website,' [Online]. Available: https://rada.re/n/radare2.html.

[92] 'The tor project,' [Online]. Available: https://www.torproject.org/.

Appendix A

Project plan

Department of Information Security and Communication Technology
DCSG2900 - Bachelor i digital infrastruktur og cybersikkerhet

Automatisert dynamisk analyse av ELF-filer

Skrevet av:
Abdulfatah Abdi-Salah
Espen Taftø Vestad
Amar Licina

Dato: Februar 1, 2021

Innhold

Liste av figurer ii

Liste av tabeller ii

1 Informasjon 1

2 Innledning 1

2.1 Bakgrunn og orientering . . . 1

2.2 Effektmål . . . 1

2.3 Resultatmål . . . 2

3 Omfang 2

3.1 Fagomeråde . . . 2

3.2 Oppgavebeskrivelse . . . 2

3.3 Problemstilling . . . 2

3.4 Avgrensning . . . 2

4 Prosjektorganisering 3

4.1 Ansvar og roller . . . 3

4.2 Rammer . . . 4

4.2.1 Tidsforløp: . . . 4

4.2.2 Rutiner: . . . 4

4.2.3 Språk: . . . 4

4.2.4 Verktøy for system og utvikling: . . . 4

4.2.5 Andre verktøy for gjennomføring: . . . 4

4.2.6 Økonomi: . . . 4

4.2.7 Levering: . . . 4

4.3 Grupperegler og rutiner . . . 5

5 Planlegging, oppfølging og rapportering 5

5.1 Hovedinndeling og utviklingsmodell . . . 5

5.2 Plan for statusmøter . . . 6

6 Organisering av kvalitetssikring 6

6.1 Dokumentasjon, standardbruk og konfigurasjonsstyring . . . 6

7 Plan for gjennomføring 7

7.1 Risikoanalyse 7

8 Gjennomføring **9**

8.1 Milepæler (Milestones) 9

8.2 Hovedaktiviteter 9

Bibliography **10**

Appendix **11**

Liste av figurer

1 Matrise for risikokalkulering. 7

2 Gantt-skjema (Gantt chart) 11

3 Timeliste 12

Liste av tabeller

1 Risikoanalyse 8

2 Milepæler 9

1 Informasjon

Title: Dynamic analysis of ELF-filer	**Gitt dato:** 11. Januar 2021 **Innleveringsdato** 20. mai 2021
Gruppedeltakere Abdulfatah Abdi-Salah abdulfaa@stud.ntnu.no Amar Licina amarl@stud.ntnu.no Espen Taftø Vestad espenves@stud.ntnu.no	**Veileder:** Ernst Gunnar Gran ernst.g.gran@ntnu.no
Oppdragsgiver: NTNU	**Kontaktperson hos oppdragsgiver:** Lasse Øverlier lasse.overlier@ntnu.no

2 Innledning

2.1 Bakgrunn og orientering

Dynamisk analyse av skadevare er spesielt aktuelt nå om dagen, da trusselaktører gjerne benytter mer avanserte metoder for å utvikle skadevare. Obfuskering, varierende oppførsel under kjøring og sletting av spor begrenser nyttigheten vi får av statiske analyser. Dynamisk-skadevare analyse er en tidkrevende prosess som krever gode forkunnskaper med disassembly av programvare og kjennskap til potensielle trussel-indikatorer under eksekvering. Automatisering av slike analyser kan bidra til å effektivisere denne prosessen. Det finnes allerede flere tilgjengelige systemer som tilbyr dynamisk-skadevare analyse for ulike filtyper.

Det finnes derimot lite informasjon rundt automatisering av Skadevare-analyse for ELF-filer, ELF står for for Executable and Linking Format ”ELF files on Linux” n.d. Skadevare for Linux systemer (blant annet løspengevirus), har også økt i relevans den siste tiden. På bakgrunn av dette, har vi fått i oppgave av Lasse Øverlier ved NTNU i Gjøvik å utvikle en løsning som utfører automatisert dynamisk analyse av ELF-binærfiler i et trygt miljø. Løsningen skal komme frem til en metode for å analysere ELF-binærfiler, slik at det blir enklere å kartlegge om filer av denne typen er legitime eller skadelige.

2.2 Effektmål

Effektmål skal være en langsiktig virkning for virksomheten, også potensielt ønsket endring fra dagens situasjon. Gjerne også utrykket i form av verdiskapning eller gevinst. ”Lindeman” ”Alt om effektmål i prosjekt: Definisjon og eksempler” ”2018-09-20”

- Effektivisere analysering av ELF-filer for sikkerhetsanalytikere.
- Utvikle metodikk og konsept som på sikt kan gi sikkerhetsanalytikere en pekepinn på om aktuelle ELF-fil det gjelder inneholder skadevare.

2.3 Resultatmål

Resultatmål er knyttet til løsning av vårt prosjekt, og hva det skal frembringe innenfor vår prosjektperiode. "Lindeman" "Alt om effektmål i prosjekt: Definisjon og eksempler" "2018-09-20"

- Forstå hvordan ELF-filer fungerer og hvordan deres headere er bygd opp.
- Uthente informasjon om hva ELF-filen foretar seg under eksekvering.
- Implementere en automatisert metode for dynamisk analyse av ELF-filer.
- Utforske hvilke sandkasse-teknologier som kan benyttes for dynamisk ELF-analyse og hvordan dette kan utføres i et trygt miljø.
- Drøfte nyttige parametre i dynamisk ELF-analyse for klassifisering av ELF-skadevare.

3 Omfang

3.1 Fagomeråde

Oppgavens fagområde ligger innenfor informasjonssikkerhet i programvare. Løsningen vil i stor grad omhandle områder som reverse engineering, scripting, tekniske ELF-spesifikasjoner og indikatorer.

3.2 Oppgavebeskrivelse

Oppgaven vil utforske hvordan man kan gjennomføre dynamisk analyse av Linux-binærfiler (ELF-filer) i et trygt sandkasse-miljø. Det skal utvikles et miljø som tar i bruk en eller flere metoder for å utføre dynamisk analyse og få nyttig output på hva binærfilen foretar seg under eksekvering. Metodene som brukes kan være nye, eksisterende, kommersielle eller egenutvikledet. Hensikten er å gi analytikeren en indikasjon på om ELF-filen det gjelder kan være ondsinnet.

3.3 Problemstilling

Basert på oppgavebeskrivelsen fra oppdragsgiver og gruppens planlegging av prosjektarbeidet, har vi kommet frem til følgende problemstilling:

Målet med prosjektet er å etablere et sikkert analysemiljø som på en automatisert måte kan analysere ELF-filer dynamisk. Løsningen skal gjennomføre en eller flere metoder for dette, og gi relevant output for hva ELF-filen foretar seg under eksekvering. Gruppen skal også utforske hvilke parametre som kan være nyttig i identifisering av ELF-malware.

3.4 Avgrensning

- Metodene som brukes i analyseringen kan være både nye, eldre og egenutviklede metoder.
- Dersom gruppen finner kommersielt tilgjengelige metoder, kan funksjonaliteten til disse beskrives fremfor å implementeres i miljøet.
- Det er krav om at løsningen skal gi følgende output om den analyserte filen: bibliotekskall, nettverksaksess, diskaksess, og logging av kode som kjøres. Gruppen står fritt til å implementere funksjonalitet som gir output av andre nyttige parametere hvis tid, men disse kan også drøftes i prosjektet fremfor å implementeres i miljøet.

- Prosjektet har hovedfokus på dynamiske analysemetoder, ikke statiske. Noen statiske outputer kan likevel være aktuelle.
- Løsningen vil kun fokusere på filtyper som kan klassifiseres som ELF-filer (Executable Linkable Fileformat). Vi vil ikke fokusere på andre typer binærfiler, som f.eks .exe. Dersom det mot all formodning viser seg at utviklingen av analysemiljøet for ELF-filer blir for komplekst og ugjennomførbart, kan prosjektet flytte hovedfokus mot MacOSX- eller Android-binærfiler i stedet. Dette kan også være aktuelt dersom gruppen får tid til overs.
- Løsningen skal ikke klassifisere om programvaren/filen(e) som analyseres er malware eller ikke. Brukeren skal kun få informasjon om hva binærfilen foretar seg, slik at han/hun får en indikasjon på om programvaren er skadelig eller ei. Visuelle metoder i analyseringsresultatet som grafer, farger, tall-score og entropy kan benyttes for å gi brukeren en indikasjon på dette. Funksjonalitet er hovedfokus og trumfer visualisering, og visualisering forutsetter derfor at vi får tid til dette.

4 Prosjektorganisering

4.1 Ansvar og roller

Gruppen har tatt en felles diskusjon på hvilke roller som kan være hensiktsmessige for prosjektperioden. Vi har dermed kommet frem til at følgende roller skal tildeles gruppemedlemmene for prosjektperioden:

- **Gruppeleder:** Fungerer som et ledd som kan fastslå en avgjørelse dersom det er uenigheter innad i gruppen som ikke kan løses ved en flertallsavgjørelse. Gruppeleder har ikke noe annet spesifikt hovedansvar utover dette, og gruppemedlemmene er selv ansvarlige for at arbeid blir utført.
- **Nestgruppeleder:** Fungerer som et ledd som kan ta over rollen som leder dersom noe skulle oppstå med Gruppeleder.
- **Kontaktperson:** Fungerer som et primært kontaktledd mot oppdragsgiver. Dette betyr ikke at andre gruppemedlemmer ikke kan kontakte oppdragsgiver.
- **Timefører:** Ansvarlig for føring av arbeidstimer i eget timeføringsark. Hvert gruppemedlem er selv ansvarlig for å rapportene inn sin timesbruk i Discord, slik at timefører kan slå disse sammen i timesarket.
- **Sekretær:** Ansvarlig for å ta notater/referat fra møter med veileder og oppdragsgiver. Notatene skal deles med resten av gruppen i intern Discord. I tillegg har sekretær ansvar for å sende møtereferat til veileder etter hvert ukentlige veiledermøte.
- **Fasilitator:** Fungerer som en lenke som vil hjelpe gruppen til å samarbeide bedre ved å hjelpe gruppene med arbeidsflyt og arbeidsdynamikk for å nå gruppens mål. Fasilitatoren vil observere gruppen når de arbeider, under møter eller diskusjoner. Fasilitatoren vil gi nøytral observasjon for gruppen å diskutere. Vanligvis skal ikke fasilitator være med på gruppe arbeid, men ettersom mye av arbeidet kommer til å være virtuelt på grunn av COVID-19, så er det greit å ha denne rollen da vi kan ha en god gruppe dynamik og arbeidsflyt.

Vi har valgt å tildele følgende roller til følgende gruppemedlemmer:

- **Gruppeleder, kontaktpunkt og timefører:** Espen Taftø Vestad
- **Sekretær:** Abdifatah Abdi-Salah
- **Sekretær og Nestgruppeleder:** Amar Licina

To gruppemedlemmer har begge fått sekretærrollen. Dette er for å ha kvalitetssikring av møtereferatene, slik at eventuell viktig info ikke forsvinner. Sekretærene sammenslår sine referater etter møtene før de sendes videre og postes i interne ressurser.

4.2 Rammer

4.2.1 Tidsforløp:

Tidsforløpet for prosjektperioden er definert i detalj i Gantt-skjemaet nedenfor (Figur 2). Vi vil også definere frister som er viktige å overholde under "milepæler" nedenfor.

4.2.2 Rutiner:

Da alle gruppemedlemmene jobber ved siden av universitetet (derav noen jobber turnus), vil vi ikke kunne opprettholde faste møtetider hver uke. En felles kalenderløsning vil benyttes for å samkjøre felles møtetider og arbeidsseanser. Når det gjelder gjøremål og progresjon, vil vi fortløpende sette ukentlige mål iht valgt utviklingsmodell som må overholdes av gruppen. Grunnet situasjonen med pandemien, vil møter ta sted både fysisk og virtuelt på Discord. Møte med veileder vil opprinnelig ta sted hver torsdag kl 09:00, men kan sløyfes eller flyttes ved behov. Møte med oppdragsgiver vil skje etter behov. Foreløpig vil det ikke stiftes faste møtetidspunkt med oppdragsgiver, men dette kan endre seg underveis.

4.2.3 Språk:

Forprosjektet skrives på norsk, mens selve hovedprosjektet skal skrives på engelsk.

4.2.4 Verktøy for system og utvikling:

Filtypene som skal analyseres er "Linux binærfiler", altså ELF-filer. Det vil derfor være naturlig at eksekvering av filen det gjelder skjer i et Linux-miljø. Det er ikke satt noen spesifikke krav til system eller programmeringsspråk for å utvikle løsningen. I utgangspunktet, vil vi benytte plattformen Cuckoo *"Cuckoo Sandbox Book"* "2020" som sandkasse. Det er også mulig at andre sandkasser blir benyttet underveis.

4.2.5 Andre verktøy for gjennomføring:

Vi benytter Latex i Overleaf når rapporten skrives, og Microsoft teams for å kommunisere med oppdragsgiver og veileder. Google calendar vil brukes som kalenderløsning. For intern kommunikasjon og fildeling, vil gruppen benytte Discord.

4.2.6 Økonomi:

Foreløpig ser det ut til at vi ikke kommer til å støte på utgifter eller har spesielle behov for økonomisk støtte. Vi vil understreke at dersom komersielle behov er den eneste utveien, vil dette forskes på og drøftes, men ikke utvikles/implementeres.

4.2.7 Levering:

Levering av bachelor rapporten vil skje i NTNUs digitale eksamenssystem innen 20.05.2021 i form av PDF. Alle vedlegg etc. skal også leveres her.

4.3 Gruppereglar og rutiner

fullført. Planen gir gruppen en kronologisk rekkefølge på hva som burde fullføres til hvilken tid, slik at vi oppnår målene vi har satt. Noen prosjektfaser kan også jobbes med parallelt. *"Gantt and Kanban: a killer combination product development"* 2020

Kanban delen gjør slikt at alle i gruppen kan se hva som trengs å gjøres ("to do"), progresjon (progress) og hva som er fullført ("done"). I vårt tilfelle benytter vi Trello som Kanban board. Kanban boardet vil bestå av følgende felter: "To do", "In progress", "testing", "review", "done". Oppgavene ligger opprinnelig i backlogen "To do", og frem til andre felter ut i fra oppgavens progresjon. Når en oppgave er fullført flyttes den til "Review" delen. Her gjennomgår gruppen oppgaven for å vurdere hvorvidt den er fullført. Etter at gruppen har sett over oppgaven, flyttes den til "Done" dersom den vurderes som ferdigstilt, eller tilbake til "In progress og testing" dersom den krever endringer. Amanda "How to Mix Kanban Gantt Project Management"

5.2 Plan for statusmøter

Vi har besluttet å arrangere to faste statusmøter i uken internt i gruppen, i tillegg til at vi har fast møte med veileder 1 gang i uka:

- Søndag kl 09:00 - 11:00 (Internt ukesmøte)
- Torsdag kl 09:00 - 10: 00 (Møte med veileder)
- Torsdag kl 10:00 - 11:00 (Statusmøte)

Søndagsmøtene er besluttet å være et møte som definerer oppgaveprioriteringer for den kommende uken i henhold til kanban boardet. Torsdager er det først statusmøte med veileder, etterfulgt av et kort internt statusmøte i gruppen. I sistnevnte vil vi gå gjennom status for ukas oppgaver for å estimere hvordan vi ligger an for ukas mål. Dette er også en mulighet for å gjennomgå feedback fra veileder.

Møter med oppdragsgiver tas etter behov, men ikke for sjeldent. Vi har fortløpende kommunikasjon med oppdragsgiver på teams, og har blitt enige i at avtaler kan opprettes når en eller begge av partene ser at dette er hensiktsmessig.

6 Organisering av kvalitetssikring

6.1 Dokumentasjon, standardbruk og konfigurasjonsstyring

Overleaf tilknyttet våre NTNU-kontoer vil bli benyttet til å skrive bacheloroppgaven. Oppgaven skal skrives i Latex, og NTNU's bacheloroppgavemal skal benyttes.

Vi planlegger å bruke et git repository for kildekoden. ***Commit***-meldinger skal være tydelige og presise i henhold til endringer som har blitt gjort. All ulik funksjonalitet skal utvikles i egen branch, og merges ikke med master branch før den er fullført. Ingen skal pushe kildekode direkte til master branch. Dokumentasjon og krav for kode og scripts legges til i readme-filen. Git-løsningen vi planlegger å benytte er Bitbucket, da NTNU har avtaleløsninger med dem. Bitbucket vil hjelpe oss å opprettholde prinsippene av CIA-trekanten. Walkowski" n.d.

Vi benytter Skyhigh openstack. Dette er en multi-tenant virtualiseringsplattform der enhver NTNU-student kan få tilgang til virtualisering ressurser. I vårt tilfelle har vi fått allokert:

- 16 instances
- 16 CPU'er
- 32GB RAM

- 10(50GB)Volumes

Basert på research vi har gjort på forhånd, vil vi sannsynligvis benytte Python3 og Bash for å utvikle løsningen. Dette kan endre seg underveis i prosjektperioden. I utgangspunktet, vil vi benytte plattformen Cuckoo som sandkasse. Det er også mulig at andre sanbox miljøer blir benyttet underveis.

Discord vil bli benyttet til fil- og informasjonsdeling internt i gruppen. Her vil det også være en "check-in"-chat som skal benyttes på starten av hver arbeidsøkt (ref. "grupperegler og rutiner"). Under en check-in, skal gruppemedlemmene poste følgende info:

1. Hva gjorde du sist gang du jobbet?
2. Hva gjør du i dag?
3. Andre ting som er nyttige å vite.

Dette gjøres for å opprettholde loggføring av arbeidsøkter, og for å holde oversikt over hva som jobbes på fra dag til dag.

Microsoft Teams benyttes for å opprettholde kontakt med veileder og oppdragsgiver. Google Calendar bentytes som felles kalenderløsning for gruppemedlemmene for å holde orden på ledige dager. I tillegg vil gruppen benytte Google Sheets for timesføring.

7 Plan for gjennomføring

Vi benytter Gantt-skjema som man kan se på figur 3 for å holde oversikt over hvilke faser prosjektet er inndelt i. Slik får vi en langsiktig oversikt over når ulike faser av prosjektperioden bør være ferdigstilte. Diagrammet vil også hjelpe oss med å holde en kontinuerlig arbeidsflyt.

7.1 Risikoanalyse

Under er en tabell for våre identifiserte risikoer, samt risikonivå før og etter eventuelle tiltak. Risikoene er tilknyttet gruppens fullføring av bachelorprosjektet, og kan bestå av følgende kategorier:

- **T:** Teknologiske
- **F:** Forretningsmessige
- **G:** Gruppemessige

For å kalkulere risikonivå (sannsynlighet x konsekvens) for de identifiserte risikoene, har vi brukt følgende mal for risikokalkulering: som man kan se på figur 1

Konsekvens	1	2	3	4	5
Veldig alvorlig	5	10	15	20	25
Alvorlig	4	8	12	16	20
Moderat	3	6	9	12	15
Liten	2	4	6	8	10
Ubetydelig	1	2	3	4	5
	Veldig lav	Lav	Moderat	Høy	Veldig høy
	Sannsynlighet				

Risiko	Beskrivelse	Tidligere matrise	Tiltak	Ny matrise
Sykdomsfravær (G)	Et gruppemedlem blir utsatt for sykdom og blir fraværende i en kort periode, noe som kan føre til at ukentlige mål ikke blir nådd innen ukens slutt.	2:4	Gruppen omprioriterer og omfordeler oppgaver på en hensiktsmessig måte under sykdomsfraværet. Gruppemedlemmet som rammes av sykdom kan fortsatt ha noen arbeidsoppgaver dersom det er gjennomførbart, eller ta igjen noe arbeid når sykdom opphever.	2:3
COVID-19 (G)	Et eller flere gruppemedlemmer blir smittet av COVID-19 og blir utsatt for et lengre sykdomsfravær.	1:5	Gruppen kontakter universitet og informerer om forholdet. Oppgaver omprioriteres og omfordeles på lik linje med vanlig sykdomsfravær, men med lavere terskel dersom sykdomsforløpet er illebefinnende. Gruppen bør begrense fysisk kontakt med for mange eksterne personer for å unngå spredning.	1:4
Høy oppgavekompleksitet (F)	Kompleksiteten på oppgaven er for høy i henhold til tidsfristen og truer kvaliteten på det ferdige produktet.	2:3	Innsnevre scopet på oppgaven og prioritere kun de viktigste gjøremålene. Eventuelt resterende gjøremål kan drøftes i rapporten.	1:2
Eksterne livssituasjoner (G)	Et gruppemedlem opplever en vanskelig familie- eller livssituasjon (f.eks samlivsbrudd, død o.l.) og har behov for permittering.	2:5	Kontakte veileder / emneansvarlige / universitetet for å finne en løsning ut i fra det enkelte tilfellet.	2:4
Usikret sandbox (T)	Et miljø som ikke er skikkelig sikret kan forårsake skade på konfidensialitet, integritet eller tilgjengelighet på våre egne fysiske systemer.	3:5	Bruke dokumentasjon og best praksis til enhver tid hvor det foregår kjøring av faktisk malware. Blokkere nettverksaksess og benytte snapshots i analysemiljøer.	2:4
Mangel på eksisterende teknologi (T)	Det finnes allerede lite cutting-edge teknologi på problemet vi skal løse, som skaper faglige og tekniske utfordringer for fullføring.	3:3	Finne alternative teknologiløsninger som kan brukes eller improviseres. Eventuelt utvikle egne verktøy dersom dette er utførbart.	3:2
Admin permissions på Discord (G)	Et eller flere gruppemedlemmer kan slette viktig info ved uhell eller av ondsinnede intensjoner, da det er mulig å slette andre sine meldinger.	1:3	Bruke loggføring på andre steder, eller gjennomgå muntlig slik at slike meldinger er husket. Det lagres også på en ekstern harddisk som backup dersom noe skulle oppstå med Discord.	1:1

Tabell. 1: Risikoanalyse

8 Gjennomføring

8.1 Milepæler (Milestones)

Under er en liste med milepæler som vi skal oppnå med ulike frister. Dette er viktig slik at vi jobber mot konkrete mål og frister. Dette er bare de overordnede og viktigste fristene. Mer detaljerte frister er å finne i fremdriftsplanen/Gantt skjemaet.

Milepæl	Frist / Dato
Innlevering forprosjektrapport	31.januar
Implementasjon av analyseverktøy	06. mars
Løsning ferdig utviklet og fungerende, rapporten røfflig ferdig og klar til gjennomgang fra veileder.	10. april
Innlevering av bachelor rapporten	20. mai
Presentasjon av bacheloroppgaven	Uke 22

Tabell. 2: Milepæler

8.2 Hovedaktiviteter

Hovedaktivitetene er til for å definere klare oppgaverammer for å kvalitetssikre den ferdigstilte oppgaven. Aktivitetene er viktige for å sikre riktige beslutninger og vurderinger i henhold til oppgaven. Det er i utgangspunktet ønskelig at aktivitetene gjøres gruppemessig. På den måten kommer alle meninger frem, som bidrar til ytterligere kvalitetssikring av oppgaven.

1. Estimere behov, trusselbilde, problemområde og bakgrunn.
For at hele gruppen skal forstå oppgavebeskrivelsen, oppgavens bakgrunn, og videre arbeidsflyt, er det viktig å definere behovet som oppgaven utlyser. Her er det viktig å stille kritiske spørsmål til oss selv. Hva er problemområde og hvem kan vi løse dette for? I tillegg er det viktig å se på andre relevante faktorer for oppgaven, som trusselbildet i det digitale rom.

2. Definere scope, rammer, grupperegler.
Når gruppen har oppnådd felles forståelse for gruppen, defineres oppgavens scope, avgrensninger, regler og rammer. Dette for å forsikre tydelighet i oppgavens omfang gjennom prosjektperioden.

3. Utforske ulike løsninger, konsept og teknologier som kan løse eller bidra positivt til problemstillingen.
Etter etablering av rammer og omfang, er det klart for å forske videre på hvilke løsninger, konsept og teknologier som kan være relevante for gjennomføring. Dette for å finne ting å forholde oss til på forhånd av utviklingen.

4. Videreutvikle relevante konsept, løsninger og teknologier.
Videreutvikling av våre funn i forrige punkt for å finne svar på problemstillingen. Utvikling vil skje innenfor rammer og omfang tidligere definert.

5. Avslutte utvikling, drøfte resultater og besvare problemet.
Utviklingen avsluttes og resultatene fra fasen drøftes. Dette for å forsørge kvalitetssikring av våre funn og hva de bidrar til.

6. Resultatevaluering og presentasjon.
Våre funn og resultater vurderes og blir klargjort for presentering.

Bibliography

"Cuckoo Sandbox Book" ("2020"). URL: https://cuckoo.sh/docs/ (visited on 9th May 2021).

"ELF files on Linux", "The 101 of (n.d.). *"Michael Boelen"*. URL: https://linux-audit.com/elf-binaries-on-linux-understanding-and-analysis/.

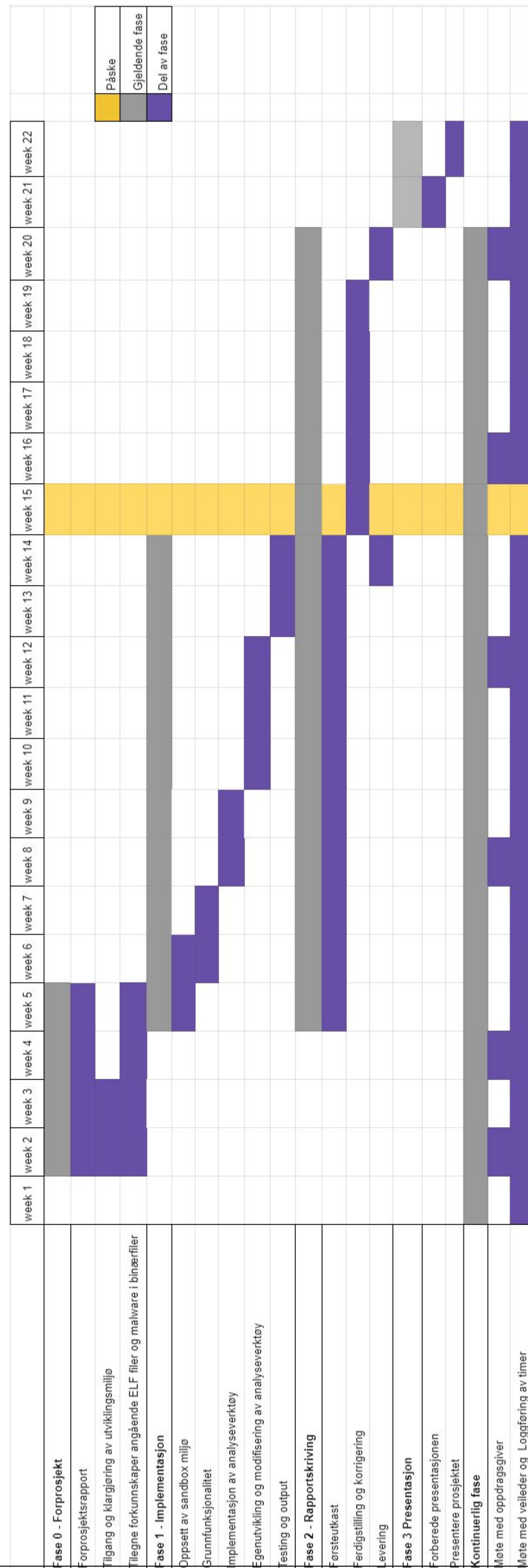
"Gantt and Kanban: a killer combination product development" (2020). URL: https://medium.com/@playbookhq/gantt-and-kanban-a-killer-combination-product-development-75c650a301d4 (visited on 21st Jan. 2020).

"Lindeman", "Kristine ("Alt om effektmål i prosjekt: Definisjon og eksempler" "2018-09-20"). URL: https://www.prosjektbloggen.no/alt-om-effektmal-i-prosjekt-definisjon-og-eksempler.

Amanda ("How to Mix Kanban Gantt Project Management"). *2020.* URL: https://www.ganttic.com/blog/kanban-gantt-project-management (visited on 9th May 2021).

Walkowski", "Debbie (n.d.). *"What Is the CIA Triad?"* URL: https://www.f5.com/labs/articles/education/what-is-the-cia-triad.

Appendix



Figur. 2: Gantt-skjema (Gantt chart)

				41,5	Timer totalt	33	Timer totalt	32,5	Timer total			
Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdi	Detaljer	Ukestimer		Totale timer så langt
Jan	1	05.01.20	Tirs									
		06.01.20	Ons									107
		07.01.20	Tors	2	Møte med ernst og gruppemøte	2	Møte med ernst og gruppemøte	2	Møte med ernst og gruppemøte			
		08.01.20	Fre									Timer gjenværende
		09.01.20	Lør									1603
		10.01.20	Søn							6		
	2	11.01.20	Man	2	Møte med Lasse og gruppemøte	2	Møte med Lasse og gruppemøte	2	Møte med Lasse og gruppemøte			
		12.01.20	Tirs									
		13.01.20	Ons									
		14.01.20	Tors	2	Kickstart/lynkurs og spørretime	2	Kickstart/lynkurs og spørretime	2	Kickstart/lynkurs og spørretime			
		15.01.20	Fre									
		16.01.20	Lør									
		17.01.20	Søn	4	Signering av prosjektavtale, jobbing på forprosjekt	4	Signering av prosjektavtale, jobbing på forprosjekt	4	Signering av prosjektavtale, jobbing på forprosjekt	24		
	3	18.01.20	Man									
		19.01.20	Tirs	2	Jobbing på forprosjekt	2	Jobbing på forprosjekt	2	Jobbing på forprosjekt			
		20.01.20	Ons	4	Jobbing på forprosjekt	4	Jobbing på forprosjekt	4	Jobbing på forprosjekt			
		21.01.20	Tors	5	Veileder møte med Ernst, gruppemøte og forprosjekt	5	Veileder møte med Ernst, gruppemøte og forprosjekt	5	Veileder møte med Ernst, gruppemøte og forprosjekt			
		22.01.20	Fre	2	Jobbing på forprosjekt (Gantt, finpuss etc.)	2	Jobbing på forprosjekt (Gantt, finpuss etc.)	2	Jobbing på forprosjekt (Gantt, finpuss etc.)			
		23.01.20	Lør									
		24.01.20	Søn							39		
	4	25.01.20	Man									
		26.01.20	Tirs	5	Riskanalyse, hovedaktiviteter, milestojnes, konfig	5	Riskanalyse, hovedaktiviteter, milestojnes, konfig	5	Riskanalyse, hovedaktiviteter, milestojnes, konfig			
		27.01.20	Ons					2,5	Jobbing på forprosjekt			
		28.01.20	Tors	3,5	Forprosjekt i latex, internt møte og møte med ernst	2	Internt møte og møte med ernst	2	Internt møte og møte med ernst			
		29.01.20	Fre	3	Formatering i latex							
		30.01.20	Lør	3	Finpuss i latex, grupperoller, utdrag sendt til lasse							
		31.01.20	Søn	4	Review av rapport, formatering, og noen endringer	3	Gantt chart og kilder			38		

Figur. 3: Timeliste

Appendix B

Project agreement

Vår dato Vår referanse

Norges teknisk-naturvitenskapelige universitet

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

NTNU Gjøvik

_______________ (oppdragsgiver), og

Espen Taftø Vestad
Amar Licina
Abdulfatah Abdi-Salah (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 11.01.21 til 20.05.21.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon, reiser og nødvendig overnatting på steder langt fra NTNU i Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle beståtte bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv NTNU Open.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): Ernst Gunnar Gran

Oppdragsgivers kontaktperson (navn): Lasse Øverlier

Student(er) (signatur): Espen Tattø Vestad dato 17.01.21

Abdulfatah Abdisalah dato 17.01.21

Amar Ticina dato 17.01.21

______________________ dato ________

Oppdragsgiver (signatur): Lasse Øverlier dato 18.01.21

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): ______________________ dato ________

Appendix C

Meeting schedule

Møte 07.01.2020:
- Ernst skal veilede indirekte, ikke si direkte hva vi skal gjøre. VI er prosjektledere
- Ha viktige roller i grupper, ta det seriøst og dette skal nevnes i rapporten seriøs kontrakt
- Kontakte lasse om 1 side oppgave fordi det eneste vi har er tittelen (send kopi til Ernst)
- (prøv å bli så mye som ferdig før påsken, da kan ernst se igjennom iløpet av ferien) Inkluder alt som er verdt å nevne i prosjektet.

Møte 09.01.2020:
Forslag til endringer i oppgavebeskrivelse
- Forutsetninger
- Sette et scope
- Liste med krav/målsetninger/forventninger
- Hva er konkret fokus på oppgaven?
- Hvor åpent er det med fritt bruk av forskjellig verktøy/språk osv
- Litt mer tydeligere/punktvis hva mål, forventing, krav osv er. Ser litt ut som et utkast atm

Første møte med Lasse 11.01.2020:
Hva går oppgaven ut på/hva forventes/hva skal vi gjøre:
- Implementere automatiserte metoder for dynamisk malware analyse av ELF-filer (Linux Executables) i sandbox aka trygt miljø.
- Logging av kildekoden kan være aktuelt
- Finne metoder som kan antyde malware i dynamisk kjøring. Vi skal altså implementere metodikk som antyder malware, ikke som nødvendigvis IDENTIFISERER malware.
- Bruk av sandbox teknologi (Cuckoo?)
- Prøve å få ut info om det nevnte (nettverk aksess, funksjonskall api-all etc) for logging av koden er målet for oppgaven.
- Få opp sandbox for trygg kjøring og uthenting
- MacOS/Android på langsiktig basis hvis man får tid eller dersom hovedoppgaven skulle «gå galt».
- Drøfting av andre nyttige/hjelpsomme/viktige metoder (som ikke nødvendigvis trengs å implementeres av oss) er også relevant.

Møte 14.01.2021:
- Overblikk over oppgaveteksten og hvorvidt denne var utfyllende nok.
- Prat rundt hva som skal leveres i før-perioden 1 feb.
- Prat angående tilganger til SkyHigh. Ble enige om å kontakte Lars Erik angående dette (Detaljer på https://www.ntnu.no/wiki/display/skyhigh/).

Møte 21.01.2021:

- Poenget med intruduksjonen, for frem contexten altså problemstillingen.
- En setning eller to om man skal ha avgrensninger, Relatert til tidsaspektet. Man kan ikke velge alt.
- Litt viktig: Den problemstilling definerer prosjektet, og det hvis vi endrer problemstillingen så har vi endret prosjektet.
- Entydig problemstilling: Ikke gi en problemstilling som peker alle veier mer vi ute etter. Hva er det vi egentlig ser etter? Vi skal finne ut en måte / en betydning. Bruk tid på problemstillingen.
- Konfigurasjonssystemet: Administrere dokumentet, hvilket verktøy som skal brukes, Ernst er usikker i hva som legges i det her, og spør om vi ser på andre rapporter hva de har skrevet. Det er ikke noe fasit på hvordan dette skal se ut: Ernst mener "CVS eller GIT" om han skulle tenke spontant.
- Hvilke planning modell: Ingen som slår oss i huet om vi velger en metode som er relevant.< Ernst mener ikke det er galt.
- Innholdet i rapporten er mye mer viktigere enn modell vi bruker.
- Nye plan for gjennomføring, alle i gruppa bør notere ting underveis.

Møte 28.01.2021:

- Referer til figurer i teksten og gi dem navn f.eks figur 1 osv.
- Kravspesifikasjon bør skrives opp som en plan, forklar hvilke man gjorde og hvilke ting man vurderte ikke var nødvendig/mulig.
- Referere eventuelle kilder for figurer.
- Nøye på "hjelper leseren for å forstå, som vi forstår".
- Ikke nødvendig med forklaring bak tallene i risikomatrisen, men blir det i hovedrapporten (hvis det blir nødvendig med risikoanalyse i hovedrapporten).
- Bruk setning eller to for å forstå hvordan risk matrisen.
- "Naturlig" innledende øvelsen, introdusere rapporten. Her kaller vi tabellen, den viser sannsynlighet x konsekvens.
- Send Lasse forprosjektet. - flere roller for gruppa, ikke "delt ansvar"
- Se på andre prosjekter som ideer - Problemstilling: vanskelig å si noe om hvor ambisiøs den er. Ikke den perfekte automatiserte metoden, men en metode
- Spørre Lasse om ambisjonsnivået / realismen i dette, om dette er det han tenker seg.
- Lett å skrive om hva man gjorde og hvorfor, men også interessant å skrive om hva man vurderte å gjøre som f eks ikke virket og hvorfor dette ikke virket. "Vi vurderte også denne metoden her, men fant ut at....".

Møte 11.02.2021:
- Fiks forprosjektet: Husk å bruk det som vedlegg, men ikke bruk flere dager på å rette den.

Møte 18.02.21:
- Skriv om hvorfor vi ikke kan bruke cuckoo.
- Spør lars-erik om det er mulig å pushe opp malware til openstack.
- Snakk med Lasse om hva mener om "limon" , mulig alternative løsninger

Møte 25.02.2021:
- Snakk med lars erik om løsning for limon i openstack.

Møte 26.02.2021:
- OpenStack og nestet virtualisering kan være vanskelig.
- Limon scriptet er gammelt, og verktøy som depender på det kan ha utdaterte versjoner.
- Abdi henter muligens fysisk PC hvis virtualisering blir vanskelig.
- Lese på Jon Everett sin masteroppgave for læringens del.
- Lese litt opp på Vmrun og hva det tilbyr.

Møte 11.03.2021:
- Kilder / referanse.
- Ta med underkapitler selv om vi bare leverer 3 første kapitlene.

Møte 18.03.2021:
- Statusoppdatering
- Mandag 29/ 3. ha ting klart
- Hva er neste steg, er det noe rom for automatisering
- Yara regler
- Hva programvaren gjør. hva den er ute etter, hva skal man bruke resultatet til
- Gjennomgang praksisdelen
- gitt av kjekke Espen

Møte 08.04.2021:
- Svar her mer rettet til hva oppgaven spør etter, enn å forklare ting nøye som ikke er relevant.
- Rapporten er svaret
- Vektorgrafikk (figurer)
- Metodikken
- Skriver og utformer som gir mening for oss. Ernst sine kommentarer er der for veiledning, etter hans mening
- Unngå hyperlinker, legg inn på kilder. Dytt den inn i bibliography
- Se om Lasse har mulighet til å se på rapporten

Møte 15.04.2021:

- Få tilbakemelding fra noen andre enn veileder som ikke har lest teksten.

Møte 22.04.2021:

- Velge hvordan vi skal håndtere kilder (fotnoter eller alt i bibliografi)
- Sørge for at limon kan kjøre 32b malware samples på 64b system

Møte 29.04.2021:

- Finne forskjell mellom resultat fra malware test. Vise antall suksess/feil i prosent.(Bare forslag fra Ernst).
- Levere ferdig rapport til Ernst den 9 mai.

Møte 06.05.2021:

- Så lenge dynamisk analyse er hovedfokus så kan man bruke statisk analyse resultat i rapporten.
- Å nevne ubrukte funksjonaliteter er greit, men ikke skriv noe om ubrukte funksjonaliteter.

Møte 15.05.2021:

- Skriv mer om andre sandbox løsninger, 0,5
- 1 side om hvorfor man bruker/ikke bruker en løsning
- *Skrive om relevant parametere for malware analyse for ELF-filer*
- Mange av filene avslutter med seg-fault, dette kan være forårsaket av exploiten malwaren vil utnytte er patched
- Finn andel stripped med script? - Få frem at vi har et git-repository
- Pass på ordlegging av hva man ikke rakk å få med på prosjektet
- Pass på sidetall, ikke passer 80-90 sider - Innledende setning for kapitler og delkapitler - Fiks forfatternavn i bibliografi

Appendix D

Working hours

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
Januar	1	05.01.21	Tirs							
		06.01.21	Ons							
		07.01.21	Tors	2	Møte med ernst og gruppemøte	2	Møte med ernst og gruppemøte	2	Møte med ernst og gruppemøte	
		08.01.21	Fre							
		09.01.21	Lør							
		10.01.21	Søn							6

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
Januar	2	11.01.21	Man	2	Møte med Lasse og gruppemøte	2	Møte med Lasse og gruppemøte	2	Møte med Lasse og gruppemøte	
		12.01.21	Tirs							
		13.01.21	Ons	2	Kickstart/lynkurs og spørretime	2	Kickstart/lynkurs og spørretime	2	Kickstart/lynkurs og spørretime	
		14.01.21	Tors							
		15.01.21	Fre							
		16.01.21	Lør							
		17.01.21	Søn	7	Signering av prosjektavtale, jobbing på forprosjekt	7	Signering av prosjektavtale, jobbing på forprosjekt	7	Signering av prosjektavtale, jobbing på forprosjekt	33

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
Januar	3	18.01.21	Man							
		19.01.21	Tirs	5	Jobbing på forprosjekt	5	Jobbing på forprosjekt	5	Jobbing på forprosjekt	
		20.01.21	Ons	7	Jobbing på forprosjekt	7	Jobbing på forprosjekt	7	Jobbing på forprosjekt	
		21.01.21	Tors	8	Veileder møte med Ernst, gruppemøte og forprosjekt	8	Veileder møte med Ernst, gruppemøte og forprosjekt	8	Veileder møte med Ernst, gruppemøte og forprosjekt	
		22.01.21	Fre	5	Jobbing på forprosjekt (Gantt, finpuss etc.)	5	Jobbing på forprosjekt (Gantt, finpuss etc.)	5	Jobbing på forprosjekt (Gantt, finpuss etc.)	75
		23.01.21	Lør							
		24.01.21	Søn							

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
Januar	4	25.01.21	Man							
		26.01.21	Tirs	8	Riskanalyse, hovedaktiviteter, milestones, konfig	8	Riskanalyse, hovedaktiviteter, milestones, konfig	8	Riskanalyse, hovedaktiviteter, milestones, konfig	
		27.01.21	Ons							
		28.01.21	Tors	5,5	Forprosjekt, Intern møte & møte med Ernst	5	Internt møte og møte med ernst	5,5	Jobbing på forprosjekt	
		29.01.21	Fre	6	Formatering i latex			5	Internt møte og møte med ernst	
		30.01.21	Lør	6	Finpuss i latex, grupperoller, utdrag sendt til lasse					
		31.01.21	Søn	8	Review av rapport, formatering, levering	6	Gantt chart og kilder	6	kilder, tabell og figurliste, og informasjons beskrivelse	77

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
February	5	01.02.21	Man							
		02.02.21	Tirs							
		03.02.21	Ons							
		04.02.21	Tors	3,5	Møte m Ernst og jobbing	3,5	Møte m Ernst og jobbing	3,5	Møte m Ernst og jobbing	
		05.02.21	Fre							
		06.02.21	Lør	3	Oppsett nettverk, subnett, ruter og FW rules i openstack					
		07.02.21	Søn	5,5	Planlegging for uka + mekking av ssh i openstack	5,5	Planlegging for uka + mekking av ssh i openstack	5,5	Planlegging for uka + mekking av ssh i openstack	30

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
February	6	08.02.21	Man							
		09.02.21	Tirs	8	Gjennomgang av kommentarer forprosjekt, oppsett av vbox	8	Gjennomgang av kommentarer forprosjekt, oppsett av vbox	8	Gjennomgang av kommentarer forprosjekt, oppsett av vbox	
		10.02.21	Ons					5	Startet på introduksjon	
		11.02.21	Tors	10	videre oppsett av cuckoo, scriptet noe av installasjonen	8	jobbet med retting i forprosjekt, startet på introduksjonen	6	Møte med Ernst og jobbet med intro	
		12.02.21	Fre							
		13.02.21	Lør			6	timer rettet forprosjektet og lagt til oppdateringer			
		14.02.21	Søn	8	oppsett av dependencies for Limon, ukesmøte					67

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
February	7	15.02.21	Man	4	fortsettelse på Limon setup (ssdeep, sysdig og inetsim)					
		16.02.21	Tirs	6	Limon oppsett (måtte starte på nytt fordi jeg oufet)					
		17.02.21	Ons	6	Satt opp VMware på egen maskin for å fortsette Limon lokalt	5	Fikset forprosjektet.Sett på inspo i tidligere bachlor prosjekter			
		18.02.21	Tors	7	møte med ernst, oppsett av Limon og Remnux i VMware	5	Møte med Ernst og jobbing videre på intro	5	scripting med radare2 i python + møte med Ernst	
		19.02.21	Fre	5	Gjorde ferdig remnux oppsett og startet på analyse-VM oppsett	5	lagd et forslag på struktur på rapporten	9	scripting av radare2 i python	
		20.02.21	Lør	7	oppsett av analysemaskin i VMware	5	jobbed med introduction og funn på oppsett			
		21.02.21	Søn	2	ukentlig møte	2	ukentlig møte	2	ukentlig møte	75

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
February	8	22.02.21	Man	8	oppsett av remote desktop i open stack			5	Scripting radare2	
		23.02.21	Tirs							
		24.02.21	Ons	8	fullførte gui i openstack, mer vmoppsett, kontaktet lasse og lars-erik					
		25.02.21	Tors	5	møte med Ernst, og med gruppen, leting etter alternative løsninger på limon	4	møte med Ernst, og med gruppen, leting etter alternative løsninger på limon			
		26.02.21	Fre	7	Møte med Lasse, og gruppen, så litt mer på alternative løsninger	7	Møte med Lasse, og gruppen, så litt mer på alternative løsninger	7	Møte med Lasse, og gruppen, så litt mer på alternative løsninger	
		27.02.21	Lør							
		28.02.21	Søn							51

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
Mars	9	01.03.21	Man	7	Utvidet rapportstruktur, addet kapitler til overleaf, testet software					
		02.03.21	Tirs					6	Jobbet mer med på introduksjonen	
		03.03.21	Ons							
		04.03.21	Tors	5	Møte med Ernst og jobbing	5	Møte med Ernst og jobbing	5	Møte med Ernst og jobbing	
		05.03.21	Fre							
		06.03.21	Lør							
		07.03.21	Søn	5	Møte meg gruppen ELF-beskrivelse	5	Møte meg gruppen ELF-beskrivelse	5	Møte meg gruppen ELF-beskrivelse	43

‘’

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
Mars	10	08.03.21	Man	8	Konfigurering av Limon	7	ELF-beskrivelse			
		09.03.21	Tirs	8	Limon og nettverkskonfig. Frodes forelesning.					
		10.03.21	Ons	7	Satte opp nettverk og INetSim					
		11.03.21	Tors	6	Gjennomgang Limon, møte med Ernst, finpuss på script	6	Gjennomgang Limon, møte med Ernst	7	Gjennomgang Limon, møte med Ernst	
		12.03.21	Fre					8	"Fulførte"/satte under review avgrensninger, effektmål, resultatmål, målgruppe, og ryddet opp i strukturen på intro	
		13.03.21	Lør	7	skrev på intro, div review, testet sample			7	"Fulførte"/satte under review motivation	
		14.03.21	Søn	8	møte med gruppa, testet sample	6	møte med gruppen og ELF beskrivelse	8	møte med gruppen	93

Måned	**Uke**	**Dato**	**Ukedag**	Espen	**Detaljer**	Amar	**Detaljer**	Abdulfatah	**Detaljer**	**Ukestimer**
Mars	11	15.03.21	Man	5	Konfigurering av Limon	5	ELF-beskrivelse			
		16.03.21	Tirs	7	Limon og nettverkskonfig. Frodes forelesning.					
		17.03.21	Ons	5	Satte opp nettverk og INetSim	3	Gjennomgang Limon, møte med Ernst	5	jobbet med requirements (Use case/high-level/funksjonalitet)	
		18.03.21	Tors	4	Gjennomgang Limon, møte med Ernst, finpuss på script			2	Gjennomgang Limon, møte med Ernst	
		19.03.21	Fre					2	"Fulførte"/satte under review motivation	
		20.03.21	Lør	6	skrev på intro, div review, testet sample					
		21.03.21	Søn	6	møte med gruppa, testet sample	1	møte med gruppen	1	møte med gruppen	52

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
Mars	12	22.03.21	Man	5	implementation, sendte oppsummering til lasse på teams	2		2	leste gjennom tidligere bachelor oppgaver og begynte på operasjonelle krav	
		23.03.21	Tirs			1	malware i ELF-filer			
		24.03.21	Ons	1	leste litt på rapporten, svarte Lasse	8	Div skriving og finpuss på teori om ELF	1	Leste gjennom andre rapporter og forklaringer om "External requirements"	
		25.03.21	Tors	7	Fortsatte på implementation			4	Fulførte/"satte under review" sekvensdiagram, functional, operational, external requirements	
		26.03.21	Fre	7	fortsatte på implementation, addet nye ord til glossary			6	Fulførte/"satte under review" arkitektur design og netverksdesign	
		27.03.21	Lør			3	retting generelt i 3x, Malware i elf filer			
		28.03.21	Søn	9	rettet introduction, fullførte implementation	3	retting generelt i 3x, Malware i elf filer			59

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
Mars/April	13	29.03.21	Man							
		30.03.21	Tirs							
		31.03.21	Ons							
		01.04.21	Tors							
		02.04.21	Fre							
		03.04.21	Lør	5	implementerte ltrace i limon					
		04.04.21	Søn	7	møte, så videre på sysdig, jobbet med design	4	Møte, retting	4, retting		20

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
April	14	05.04.21	Man	6	Skrev videre på design					
		06.04.21	Tirs							
		07.03.21	Ons	6	overføring av nye samples, svart lasse på teams, skrev videre på design			3	Skrev ferdig previous knowledge, virtualization, startet på analysis	
		08.04.21	Tors	5	møte, begynte på retting i implementati on	3	Sett over komentarer rettet og deltok på møte	3	Møte, rettet opp kommentarene på requirements og design	
		09.04.21	Fre							
		10.04.21	Lør							
		11.04.21	Søn	1	møte	1	møte			28

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
April	15	12.04.21	Man					2	la til innledende avsnitt der det manglet på design og requirements + retting av sequence diagram og intro	
		13.04.21	Tirs	5	rettet videre på implementati on					
		14.03.21	Ons	7	rettet og skrev om i implementati on	4	retting	3	Retting av intro (mine deler)	
		15.04.21	Tors	5	møte, jobbing med test av malware samples	5	retting			
		16.04.21	Fre							
		17.04.21	Lør	5	prøvde å teste samples					
		18.04.21	Søn	3	møte og review	3	møte og review	3	møte og review	45

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
April	16	19.04.21	Man							
		20.04.21	Tirs	6	reconfiga sandboxen	6	lagd en forenklet topologi av instratrukturen til ELF, rettet 3x	3	Start på analyse bit + omskriving av avsnitt på "implemented functionality"	
		21.03.21	Ons							
		22.04.21	Tors	6	møte, fikset error for 32 bit executables	6	lagd en forenklet topologi av instratrukturen til ELF, rettet 3x	2	møte + La til descriptions under glossary	
		23.04.21	Fre							
		24.04.21	Lør							
		25.04.21	Søn	5	Finpuss, skrev om sandbox, previous knowledge etc.	5	Finpuss, skrev om sandbox, previous knowledge etc.	4	Finpuss, skrev om sandbox, previous knowledge etc.	43

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
April/Mai	17	26.04.21	Man	5	sendte samples til Lasse og så på de-obfuskering av kode					
		27.04.21	Tirs	5	gikk gjennom malware samples 1 - 33 og noterte ned de med nyttige resultat	5	Finpuss, skrev om sandbox, previous knowledge etc.			
		28.03.21	Ons	4	gjorde endringer på yara-deteksjon, rettet på implementation					
		29.04.21	Tors	7	så gjennom resultater, møte, retting på design	6	sett på malware samples	6	så gjennom testresultat fra malware samples, finpusset tekst	
		30.04.21	Fre			6	Analysert filer			
		01.05.21	Lør							
		02.05.21	Søn	6	skrev på application design					50

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
Mai	18	03.05.21	Man							
		04.05.21	Tirs							
		05.05.21	Ons	4	Finpusset/retting	4	Finpusset/retting	4	analyse + glossary	
		06.05.21	Tors	5	Retting	3	Retting	3	Retting	
		07.05.21	Fre	3	begynte på performance	7	Fnpussing, skrevet analyseringsdelen			
		08.05.21	Lør	7	skrev videre på performance, gikk gjennom alle samples og lagde statistikk på antall feilede / suksessfulle	7	Finpusset, skrivet analyseringsdelen	7	glossary og analysis	
		09.05.21	Søn	10	retting og kildehenvisning, skriving på performance	7	Finpusset, skrivet analyseringsdelen	7	Analysis, retting, glossary + kilder	78

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
Mai	19	10.05.21	Man							
		11.05.21	Tirs			5	Finpussing og ordne kilder på plass, samtidig rette forprosjektet	5	Analysis, retting, glossary + kilder)	
		12.05.21	Ons							
		13.05.21	Tors	7	discussion	4	Retting	4	Analysis, retting, glossary + kilder)	
		14.05.21	Fre	8	Useful malware indicators in elf-files (symbols, segments, run-time) og further work (forslag til additions til limon som kan forbedre implementasjonen, listings etc	4	Diskutert med Babdi, hvordan ELF header kan ha section header, program header men samtidig ha den også på file data, forstå sammenhengen, og hvordan toplogien stemmer til forklaringen, har også oppdatert visse subsections. derfor jeg lagde en enda subsection som forklarer hvorfor elf header inneholder field (program header og section header	5	Analysis, retting, glossary + kilder	
		15.05.21	Lør							
		16.05.21	Søn	2	Discussion / conclusion / retting					44

Måned	Uke	Dato	Ukedag	Espen	Detaljer	Amar	Detaljer	Abdulfatah	Detaljer	Ukestimer
Mai	20	17.05.21	Man	2	Discussion / conclusion / retting					
		18.05.21	Tirs	10	Discussion / conclusion / retting	4	Rettet på format i kilder	4	Fikset feil i Kilder	
		19.05.21	Ons	8	Gjennomgang av dokumentet, Apendix, retting av forprosjekt	8	Gjennomgang av dokumentet, Apendix, retting av forprosjekt	8	Gjennomgang av dokumentet, Apendix, retting av forprosjekt	
		20.05.21	Tors	12	Gjennomgang av dokumentet, Apendix, retting av forprosjekt	12	Gjennomgang av dokumentet, Apendix, retting av forprosjekt	12	Gjennomgang av dokumentet, Apendix, retting av forprosjekt	
										80

Appendix E

Developed installation scripts

README

How do I get set up?

- Clone the repository

- chmod the “limon_setup.sh” file

- Run “limon_setup.sh” from /home/”user”/ directory
  - “limon_setup.sh” will automatically invoke “analysis_host_setup.sh” and configure the host machine

- Open the VMware GUI

- (Optional) Disable 3D acceleration for better performance

- Go to Edit -> “Virtual Network Editor”
  - Choose “Add Network”
    - Choose “host-only” and give it a fitting name.
    - Click add.
  - Make sure the DHCP and “Host virtual adapter” boxes are ticked.
  - Enter appropriate subnet IP with netmask 255.255.255.0
  - Press save.

- Right click on the virtual machine and choose “Settings” -> “Network adapter”.

- Click “custom” and choose the VMnet you just created.

- Make sure the boxes “connected” and “connect at power on” are checked.

- Make sure that the Host serves is the default gateway and preferred DNS for the guest. On the guest OS:

- Run the command: “route add default gw {IP-ADDRESS} {INTERFACE-NAME}” where IP-address is the Host OS IP and interface-name is the name of the guest network interface (this can be checked by running “ip addr”).

- Verify that the Host serves as GW for guest by running the command: /sbin/route -n

- To set preferred DNS:

  - Open settings in ubuntu and choose network.

  - Click the “cog-wheel” on your connection

  - Select IPv4

  - Disable the “Automatic” toggler under DNS. Enter the Host machine IP. Secondary and thirdary DNS resolvers might also be set if wished.

  - Press save.

  - Updated DNS resolvers might be verified with the command: “systemd-resolve –status | grep ‘DNS Servers’ -A2” (note that previous DNS resolvers might be cached by the system or applications).

- Make sure that the analysis machine has the same static IP-address at all times. To achieve this, we need to modify vmware’s dhcp configuration.

- Open /etc/vmware/vmnet/dhcpd/dhcpd.conf

- You will see an entry for which includes the static IP and MAC-address of the Host.

- Add a new entry right below.

  - Set “hardware ethernet” entry to the MAC-address of the guest.

  - Set “fixed-address” to the static IP of choice.

- Save the configuration

- Restart VMware’s DHCP service:

  - net stop vmnetdhcp

  - net start vmnetdhcp

- Create a snapshot of the guest machine using the Vmware Workstation snapshot manager.

- Update the variables in “conf.py” with the appropriate values.

Usage

- Usage: limon.py [Options] {file} [args]
- Options:

  - -h, –help show this help message and exit

  - -t TIMEOUT, –timeout=TIMEOUT timeout in seconds, default is 60 seconds

  - -p, –perl perl script (.pl)

  - -P, –python python script (.py)

  - -z, –php php script

  - -s, –shell shell script

  - -b, –bash BASH script

  - -C, –ufctrace unfiltered call trace(full trace)

  - -x, –printhexdump print hex dump in call trace (both filtered and unfiltered call trace)

- -l, –libtrace trace library calls
- -L, –libstrace trace BOTH library and system calls

Code listing E.1: Limon installation script for configuring the host machine.

```
#!/bin/bash

# Init
sudo apt-get update -y && sudo apt-get upgrade -y

# Desktop GUI
#sudo apt install -y tasksel
#sudo tasksel install -y lubuntu-core
#sudo service lightdm start
#sudo passwd ubuntu

# Dependencies
sudo apt-get -y install gcc make linux-headers-$(uname -r) dkms
sudo apt install unzip
sudo apt-get -y install automake libtool make gcc pkg-config
sudo apt install -y python-pip

# VMware install
wget http://folk.ntnu.no/espenves/VMware-Workstation-Full-16.0.0-16894299.x86_64.bundle
sudo ./VMware-Workstation-Full-16.0.0-16894299.x86_64.bundle --console
wget http://www.trendsigma.net/vmware/_dl/ubuntu1804t.zip
unzip ubuntu1804t.zip

# Yara
wget https://github.com/VirusTotal/yara/archive/v4.0.5.zip
unzip v4.0.5.zip
cd yara-4.0.5
./bootstrap.sh
./configure
make
sudo make install
make check
cd ..
pip install yara-python
git clone https://github.com/Yara-Rules/rules.git
mv rules/ yara_rules


# Ssdeep
sudo apt-get -y install ssdeep

# Sysdig
curl -s https://s3.amazonaws.com/download.draios.com/stable/install-sysdig | sudo bash


# Volatility
#wget https://github.com/gdabah/distorm/archive/v3.5.zip
#unzip v3.5.zip
#cd distorm-3.5
#sudo python setup.py install
#cd ..
#wget https://ftp.dlitz.net/pub/dlitz/crypto/pycrypto/pycrypto-2.6.1.tar.gz
#tar -zxvf pycrypto-2.6.1.tar.gz
#cd pycrypto-2.6.1/
#chmod +x setup.py
#sudo python setup.py build
#sudo python setup.py install
#cd ..
#pip install Pillow
#pip install openpyxl
#pip install ujson
#wget http://downloads.volatilityfoundation.org/releases/2.4/volatility-2.4.tar.gz
#tar -zxvf volatility-2.4.tar.gz
#cd volatility-2.4/
#sudo chmod +x vol.py
#cd ..

# Misc
mkdir /home/ubuntu/linux_reports
rm pycrypto-2.6.1.tar.gz
rm ubuntu1804t.zip
rm v3.5.zip
rm volatility-2.4.tar.gz
rm v4.0.5.zip

# Remnux
wget https://REMnux.org/remnux-cli
mv remnux-cli remnux
chmod +x remnux
sudo mv remnux /usr/local/bin
sudo remnux install --mode=addon

# Start VMware machine, transfer and execute setup scripts
```

```
vmrun start /home/ubuntu/Ubuntu1804/Ubuntu.vmx
echo "Sleeping for 30 sec after reboot....."
sleep 30
sudo vmrun -gu user -gp password CopyFileFromHostToGuest /home/ubuntu/Ubuntu1804/Ubuntu.vmx
/home/ubuntu/analysis_host_setup.sh /home/user/analysis_host_setup.sh
sudo vmrun -gu user -gp password CopyFileFromHostToGuest /home/ubuntu/Ubuntu1804/Ubuntu.vmx
/home/ubuntu/run_sysdig.sh /home/user/run_sysdig.sh
sudo vmrun -gu user -gp password CopyFileFromHostToGuest /home/ubuntu/Ubuntu1804/Ubuntu.vmx
/home/ubuntu/run_sysdig_full.sh /home/user/run_sysdig_full.sh
sudo vmrun -gu user -gp password runProgramInGuest /home/ubuntu/Ubuntu1804/Ubuntu.vmx /bin/bash
/home/user/analysis_host_setup.sh

# Create snapshot of analysis environment
vmrun -gu user -gp password reset /home/ubuntu/Ubuntu1804/Ubuntu.vmx soft
echo "Sleeping for 20 sec after reboot....."
sleep 20
vmrun -gu user -gp password snapshot /home/ubuntu/Ubuntu1804/Ubuntu.vmx cleansnapshot

sudo reboot
```

Code listing E.2: Limon installation script for configuring the guest machine.

```
#!/bin/bash

# Dependencies
echo "password" | sudo -S apt-get update -y && sudo apt-get upgrade -y
echo "password" | sudo -S apt install -y curl
echo "password" | sudo -S apt install -y git
echo "password" | sudo -S apt install libelf-dev
echo "password" | sudo -S apt-get install build-essential
echo "password" | sudo -S apt install net-tools

# Sysdig
#curl -s https://s3.amazonaws.com/download.draios.com/stable/install-sysdig | sudo bash

# Strace
#echo "password" | sudo -S apt-get remove -y strace
#wget https://strace.io/files/5.11/strace-5.11.tar.xz
#tar -xf strace-5.11.tar.xz
#cd strace-5.11
#./configure --disable-mpers
#make
#make install
#cd
#rm strace-5.11.tar.xz
echo "password" | sudo -S apt-get install -y strace

# PHP
echo "password" | sudo -S apt-get install -y php-cli

# Packages for 32bit exec on 64bit system
echo "password" | sudo -S dpkg --add-architecture i386
echo "password" | sudo -S apt-get update
echo "password" | sudo -S apt-get -y install libc6:i386 libncurses5:i386 libstdc++6:i386
echo "password" | sudo -S apt-get -y install multiarch-support
echo "password" | sudo -S apt-get -y install gcc-multilib
echo "password" | sudo -S apt-get -y install lib32z1 libc6-i386 lib32stdc++6 lib32gcc1 lib32nc
echo "password" | sudo -S apt-get -y install lib32ncurses5

# Directories for analysis
mkdir malware_analysis
mkdir logdir
PATH=$PATH:~/malware_analysis
touch path.txt
echo -n 'PATH="' >> path.txt
echo -n $PATH >> path.txt
echo '"' >> path.txt
# Runs a sudo dummy command in order to make the next command run automatically
echo "password" | sudo -S touch /tmp/temp.txt
cat path.txt | sudo tee /etc/environment
rm path.txt

# Volatility
#git clone https://github.com/volatilityfoundation/volatility.git
#cd volatility/tools/linux/ && make
#cd ../../../

# Clear history
history -c && history  w
```

Appendix F

Thesis project repository

Please consider the following thesis repository for retrieving the developed implementation of Limon.

https://bitbucket.org/espur/dynamisk-elf-analyse/src/master/ [74]

Appendix G

Malware test samples

VirusShare_0034ebc8a85edbb507dd550952a2cb92
VirusShare_00744ba3546a01e8c2a3cb3711c3ca85
VirusShare_0086eced29d57421ec8778f1f3084915
VirusShare_0093fdcb12b6fb836495b7cd53d19ddb
VirusShare_009b3cc8dc9d3d16dc4c363bb573089d
VirusShare_00b522fe648eea24c3eb90b3d814ed72
VirusShare_00d051be54a70826b8d20645900c6c1a
VirusShare_00d0e365b421e81cd7205195199cd0a0
VirusShare_00f7adbe9895699b07a114e383787c74
VirusShare_0100a791cf0924db4d3e890d02e32b60
VirusShare_013acf2ce0515dc1d297d9ab764be847
VirusShare_014575e602cb2c1622c33a413ce2e009
VirusShare_015a2e7e8810fda0ece9dbf407c3e5ef
VirusShare_016550948bfe5f621d8109b30efced41
VirusShare_0167b8b07c83a6375ff467d16f0436a7
VirusShare_0181a556734500536c51159a2ac287fc
VirusShare_018ff2c8b70dc7534eed16c846c756ec
VirusShare_01c5791ee05d656b8c24067bd6bfb70f
VirusShare_01c5f86372a4f31e72675f8be9b4e6c7
VirusShare_01c9f9c35fa06e31662b9a607d8796c8
VirusShare_01e3196969078a89b853ada2b3c9eee6
VirusShare_02033432a69a38770c945a42efbd3b6b
VirusShare_023c291c905f8056e16a62dcc401b1e8
VirusShare_023c5d646b65b2127fa24b5b416e8317
VirusShare_02b5194ca19e6ba87c735e487ca65a2a
VirusShare_02b53285786714766d98bfd8946cde31
VirusShare_02c6a5ec6d06cb003c333a40966fb00d
VirusShare_02eb5393ce1a84b800c5f5d26833e02f
VirusShare_02ee0e4019e18fe3018180565aa822cc
VirusShare_03016755d810a713160f2e86a02735f5
VirusShare_0350e96784538cd11180047abab23dc3
VirusShare_035540dde404c9349963d86b009ddf6c
VirusShare_0367668dbd00eda7a3a81ea3748ad362
VirusShare_03766d86184f1bffc6535b9ef622d7c0
VirusShare_03ab73361885d188d31874dc0164068d
VirusShare_03c0e7e69211f93b9a6b7a8179e38430
VirusShare_03c730b6568a188cce18b826d79df2e4
VirusShare_03cd2f8a76ab98128b42c93a4f8f6d4a
VirusShare_03f1b84ef3c69e8a53be28efed8d326c
VirusShare_03f734c696b0280fdf2f77a5135ebe6a
VirusShare_03fb5f808f57736a6d15cb0a7f11c10f
VirusShare_0400ea2706c4eb5110cd9ab3106e4936
VirusShare_041e4fe0f8dcde038421bd560d2a4a13
VirusShare_041f12f539b38cee6e16bb24701766c3
VirusShare_04253544bdd1f485367cc14f3c00c73c
VirusShare_0440a5a6db8da7782b2d07ab86a7a645
VirusShare_0441dd25bb3b5914f09d1d44063f5389

VirusShare_044a41d1ce9a2f5a7de9559155456f88
VirusShare_0453ae2cac43ee1da908ca414c3e31e4
VirusShare_049874bf0b678b7824c61f9be244ed09
VirusShare_04a9e8116754afb5817ecda18e132fec
VirusShare_04ab137aca2e4ec0981c2bac34ed6126
VirusShare_04bbb03caa394e33718640611eefb869
VirusShare_0686a7459152174f821c8c635cfbda8a
VirusShare_0b855d8d6a3c3ac8d5fd6931570e02ae
VirusShare_10f5beac257a92665866cdc99550b7bb
VirusShare_11c489ddea858030b23f7ac184994439
VirusShare_1c2c9f1af3cd38775278dfc5712692e6
VirusShare_2154c7cdd8b1f44fda317a6ffbd776bf
VirusShare_27e700802c9b01ba5164d0974dd5b7b8
VirusShare_2ad28d994083eb88d56eded361d7e381
VirusShare_31c55141129151ee4728a40613b93eca
VirusShare_39d46a0cd60393e5571b720c915db30d
VirusShare_4087376ef72170f248eb2f0665a26796
VirusShare_4b1e9e8ccf91998393509290d436ede3
VirusShare_4b35f8b0a8dbc25ff6b5433ab48a4f30
VirusShare_52852ac955ba03e4ebb012c55550dca3
VirusShare_559169cd8167dcbaaf065d6a122a289d
VirusShare_58af33baf68feb637b59a20ba4ea0c03
VirusShare_664378d10f610552d17e97cc06ade139
VirusShare_6d5aaa20ea45eb247e483cc9c0519f63
VirusShare_705df7bc13a3fc1bbfc79735455fda68
VirusShare_7af5fe43c89670af8f866d599d1fd3e9
VirusShare_7eb2f5306e6f3acc764a22f5e0874766
VirusShare_81a1f8fe807d4631d548cd6a6d639116
VirusShare_81ea379c237724249c137fc83ef21e9a
VirusShare_8273c7634804f1bf345719f364ef33ce
VirusShare_83aa145f8b12365ca7ce37f0b03bf745
VirusShare_85e2b597de3a53124667d1a5f7863f97
VirusShare_8691a8e2228e160e2d43ec5ef03caf80
VirusShare_898dde6afb3142e607528359b0935e9e
VirusShare_8f194847387186899cc8d9f9ca903e07
VirusShare_942ea0c4cb729d4878eb5b8998981228
VirusShare_94c55f2a600446d6c698f42a4a7e3462
VirusShare_9bd1094c7ad96a5ba4de82173ec8c271
VirusShare_9fdf780ae2d88afb6b147944c3bf16a3
VirusShare_a0ed9cca11e77ed54bc9dc65c1d1f03b
VirusShare_b4088daeb311c24d8f9a20b5ec223bc9
VirusShare_b87e73448b181044996767c3cbca7e8e
VirusShare_bc5cef903c912af3ea8ac94ae77601e0
VirusShare_c0caa5138b067575dc19666370d650dd
VirusShare_c48dc5887d019b44efdef87e6fccdbd3
VirusShare_c76156aef8707a2071294a8d5b9cd974
VirusShare_cc29a224e327412e0db7f3ce5c4f4e00
VirusShare_d0b9d58f3a454ad6df2e4d055858c1e5

VirusShare_d21fb7ed52ba13294240354c1f528d2f
VirusShare_e7bc30e118b776f29541af936061f7de
VirusShare_ea0a4bafec4aad026b6549a83ef701de
VirusShare_f2b00b27e6e8d10d3c27525ecd9af120
VirusShare_f99c1d6cd8874aabedd0129cf592f5ed

Automated dynamic malware analysis of ELF-files

NTNU

Kunnskap for en bedre verden