AALBORG UNIVERSITY

Dynamic Malware Analysis through a Custom Network Topology

supervised by Marios Anagnostopoulos and Jens Myrup Pedersen

Adil Khurshid, Omar Nabil Hawwash

Department of Electronic Systems, CYBER6, 2022-06

Master's Thesis

STUDENT REPORT

AALBORG UNIVERSITY

Preface

Adil Khurshid
Adil Khurshid
akhurs20@student.aau.dk

Omar H.
Omar Nabil Hawwash
ohawwa20@student.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Department of Electronic Systems
Fredrik Bajers Vej 7B
DK-9220 Aalborg
https://www.es.aau.dk/

Title:
Dynamic Malware Analysis through a Custom Network Topology

Theme:
Master thesis

Project Period:
Fall semester 2021 - Spring semester 2022

Project Group:
Group 908

Participant(s):
Adil Khurshid
Omar Nabil Hawwash

Supervisor(s):
Marios Anagnostopoulos
Jens Myrup Pedersen

Copies: 0

Page Numbers: 106

Date of Completion:
May 30, 2022

Abstract:

The project aims to create a virtual platform for the dynamic analysis of malware samples through sandboxing. For this purpose, a virtual network topology is created with EVE-NG, a network emulation application, while within the network a sandbox machine is installed. Furthermore, a number of virtual machines with different levels of hardening with anti-evasion techniques have been set up and are dynamically infected with malware. In addition, honeypots running several services, such as FTP and several web services, have been installed. This topology is configurable, meaning that the network architecture and the virtual machines and services can be modified. This setup allows the researchers to monitor the behavior of the malware and capture its network activity in a controlled environment. The preliminary results show that when a malware infects a machine with a higher level of hardening, it has more active behavior and triggers more detection signatures.

agreement with the author.

Reading Guide

This chapter aims to reach a common understanding of various phrases the group often will utilize throughout this report. Concretely a definition of key phrases used can be found below:

- `"VM"` → Virtual Machine
- `"Sandbox"` → A containerized environment, often used for malware analysis in a technical context
- `"Cuckoo"` → The Cuckoo sandboxing platform
- `"GUI"` → A graphical user interface
- `"CAPE"` → The CAPE sandboxing platform
- `"SHADE"` → The SHADE sandboxing platform
- `"EVE-NG"` → The network emulation platform used for the foundation of this project called EVE-NG
- `"VMWare"` → A virtualization platform suite, which houses VMWare Player and VMWare WorkStation, with the latter having been used for this project
- `"Hardened"` → The Windows 7 virtual machine in which system artefacts have been removed to better hide the fact that it indeed is a virtual machine. This machine has network access, but no internet access.
- `"Weak"` → The Windows 7 machine in which system artefacts have been left untouched. This machine has network access, but no internet access.
- `"Blind"` → The Windows 7 machine in which system artefacts have been left untouched. This machine has no network access nor internet access, hence the name being "Blind".
- `"Re-image/re-imaging"` → The process of restoring the Windows machines to a previously captured state, using an image. This is used to restore Windows to a usable and uninfected state after running a malware sample.
- `"FOG"` → The imaging server used to re-image the virtual machines used for this project
- `"OVF"` → Open Virtualization Format

- `"VMDK"` → The disk files for virtual machines created using VirtualBox, an open source virtualization platform
- `"QEMU"` → A type 1 hypervisor virtualization platform
- `"NAT"` → Network Address Translation - a network type
- `"QCOW2"` → The file format for hard disks used by QEMU/KVM
- `"Crawlers"` → This is a term commonly used in the cyber security community to refer to software traversing through IP addresses or websites, whether for legitimate or nefarious purposes. Google's web crawlers which run through websites to check their legitimacy and metadata are one example. IP crawlers that traverse through IP addresses on the World Wide Web to check for vulnerabilities, are another, such as Shodan.

Contents

Chapter 1

Introduction

1.1 Motivation

Cyber security is an ever-growing field, and has seen rapid growth since the Internet was first introduced in 1995. Back then, 16 million people used the internet [1] - now that same number has risen to an astounding 5,168 million [2]. And as the user base and online assets continue to increase, rewards for adversaries become more easily obtainable. This project aspires and aims to help cyber security professionals and defendants to further strengthen the security of assets and prevent, as well as inform of, potential threats and intrusion attempts.

To help combat cyber security adversaries and further study the behavior of malware, so-called "sandboxes" have been introduced. Sandboxes allow for malware analysts and cyber security researchers to simulate the behavior of a real user system, and infect it with malware. Through this, behavioral analysis can be conducted, in order to see how the malware would behave, which techniques a malware makes use of, and this can be used for forensic research or as part of a longer fortification process of, say, a system, against a certain type of malware [3].

Unfortunately, however, malware authors have become increasingly aware of the existence and patterns of these sandboxes, and thus deploy certain sandbox evasion techniques to avoid the malware running in artificial environments. This is achieved by checking certain system information and artefacts, the usage patterns on the system, installed applications, and so forth. That is the point, we find ourselves in: there is a constant battle between malware adversaries and cyber security professionals as they continue to analyze each other's arsenal and change their own accordingly.

This master thesis aims to delve into malware analysis through sandboxing. Specifically, the project aims to investigate the available techniques for malware analysis that aim to create resilient infrastructures for sandboxing environments. Based on these findings, it will be developed a platform for sandboxing, which will facilitate the study of malware's behavior both local, on the devices under examination, and on the network level. In turn, the collected evidence will contribute to intricate knowledge in analysis, detection and enabling of malware and its behavior, with the hope that this thesis will support the continued combat against cyber security threats in the modern age.

This has led to the identification of several issues, and poses the following research questions, (see section 1.2), which we will attempt to answer throughout the course of this project.

1.2 Scope

We have defined this project's scope has been defined, prior to the project's initiation. This has been done to help narrow down the choice of tools for the duration of the project, as well as help us decide on which methodologies to make use of.

The project's scope is as follows:

- Malware analysis
- Network traffic analysis
- Sandboxing of files and applications
- Sandboxing anti-evasion techniques

The following items are considered out of scope for this project:

- Memory analysis
- 64-bit systems
- Systems running macOS or cell phone operating systems

Anti-debugging techniques has been deemed within the project's scope, although not at a memory level nor at a very low level of abstraction.

The scope is elaborated as:

- Which techniques do malware commonly use to circumvent sandboxing-based analysis environments, and how can this be mitigated?
- How can network traffic of a malware's propagation and execution be collected for further analysis in a scalable manner?
- How can behavioral analysis on malware be done in a controlled environment while not hindering the malware's full execution?

Problem statement:

How can a platform for sandboxing be made resilient enough to allow the study of malware's behavior, providing evidence for further fortification of systems and help future research on the topic?

1.3 Contributions

The following points are considered to be the project's contributions, from a technical standpoint:

- An implementation of a virtualized network topology in Eve-NG for malware analysis
- Usage of FOG with Cuckoo in Eve-NG
- All instances and nodes in the topology are configurable and scalable, meaning that nodes can be removed or added to the topology easily and quickly
- Multiple test machines running the same operating system, but with different setups and different images in FOG, allowing for behavioral analysis on malware based on its execution environment in a scalable and easily maintainable manner
- Two honeypots are set up on the local network, in order to see whether malware probing certain protocols served by the honeypots will be triggered
- A third instance of the Dionaea honeypot is set up facing the public internet, set to acquire binaries that can be analyzed in our scalable sandboxing environment

Chapter 2

Preliminary Analysis

2.1 Introduction

Having defined the scope, the problem statement and the research questions to be answered throughout the course of this project in chapter 1, this chapter will deal with the preliminary analysis. This includes outlining the background of the project, malware as a concept and at a more technical level, the different levels of analysis that will be relevant for this project, as well as sandboxing and hypervisors in general.

2.2 Malware explained

2.2.1 Background

When speaking about malware, it is important to consider the different types of malware that exists, and how they spread. This forms part of the preliminary analysis for delving deeper into the initial parts of the project.

Interestingly enough, as cyberspace has grown over the years, a rise in the amount of malware is evident. In 2012, 99.17 million malicious programs were recorded, whereas that number has risen exponentially to an astounding 1292.90 million programs as of November 9, 2021 [4]. This calls for more attention from the side of the cyber security specialists, defendants and researchers. However, given the transparency in the methods used by the people within cyber security looking to improve defense against malware as part of the educational process, the assailants can tweak their arsenal accordingly. This, in turn, makes it increasingly harder to further fortify already existing systems to an extent of certain security, which also is part of the reason why the expression “you are never 100 percent secure” has become such a big part of the cyber security community [5].

2.2.2 Types of malware

Arguably the biggest hurdle or challenge when dealing with malware and viruses is the fact that so many different types exist [6]. Following, some are outlined in further detail [6].

- Ransomware - a type of malware that encrypts data to hold it for ransom
- Adware - a type of malware that generates income by showing ads to the users on the infected machines
- Keylogger - malware that logs keystrokes on the infected machine and sends them back to the malware developers, can be used for compromization of user accounts
- Trojan - malware that disguises itself within a seemingly legitimate application
- Botnet - malware that enslaves a machine into part of a botnet, where the command-and-control center / botmaster can execute remote commands [7]. These can be used for mining or other nefarious purposes.

The category a malware falls into, depends on the classification and distinction between the types of malware, which at times can be a subjective assessment based on the program’s behavior or pattern of movement. The most tricky part of this is that some malware can be placed into multiple categories as, for instance, could serve ads (Adware) while collecting keystrokes from the user’s input device (Keylogger). Another type of malware can be a program that retrieves certain info and leaks it to the internet as part of making a statement; this has been seen with regards to hacktivism and skilled civilians fighting against their governments.

2.2.3 Real-life examples

In this section, specific real-life examples will be brought to light, to help further the understanding of the potential consequences of attacks. This section will also highlight the methods adversaries have successfully utilized in the past.

The attack on SolarWinds

An example of a well-crafted malware is the malware behind the Sunburst attack, also known as the SolarWinds. One of the things to note about this specific attack is that the malware is hard to put into a single box. In analytical terms, it falls under the "rootkit" umbrella, since it allowed the adversaries a backdoor into the infected devices once they had infected the supply chain, thus distributing malicious firmware they intended to be infected [8]. They, however, had been monitoring e-mails from the NTIA and Treasury staff for months as the attack was going on [9], which falls under Spyware. As far as the code goes, a quick run-down can be found below, as one of the group members investigated the source code for the malware once the attack had been dealt with by the United States authorities and code had become publicly available.

The malicious code was injected into a DLL (dynamic library) file that would be run through Sunburst' standard procedure [8]. It would be initialized through other functions as to not raise suspicion. The code was written in a way that made it blend in with the rest of the platform code - no obfuscation was done, as the adversaries wanted their code to blend in as well as possible with the already existing code.

The code was extremely dormant and patient, clearly crafted by someone skilled and someone who had done a lot of reconnaissance (see: cyber kill chain [10]) beforehand. Parts of the code consisted partly of *if statements* without *else clauses*; the code would check if hashes matched certain files and then proceed. If the files did not match the sought after hash, the code would simply do nothing, and thus avoid detection.

The WannaCry ransomware

The WannaCry ransomware is one of the more well-known examples of malware within the cyber security community. It spread as a worm, and it was made by the hacker group called The Shadow Brokers, who a year prior to the attack had found a weakness in Microsoft Windows and made a hack known as EternalBlue available to the public.

Two months prior to the attack, Microsoft released a patch to protect users against these systems, but since some companies and users had not updated their operating systems at that time, they were left exposed to this attack — and that helped it spread through EternalBlue [11], a Windows Server-Message-Block vulnerability exploiter created by the NSA [12].

WannaCry essentially would encrypt users' data and request a certain amount of money to be paid through Bitcoin, which was a popular cryptocurrency at the time. The user data would remain encrypted until the amount of money was paid, the amount initially being 300 U.S. Dollars worth of Bitcoin, then rising to 600 U.S. Dollars if the user had not paid the requested price after a certain amount of time.

F-Secure claim that some users got their data back after the attack, although the decryption code has, by others, been claimed to be faulty and thus not able to associate the victim's computers with the payments. This would mean that users would pay, but not get their data back [13]. In any case, the WannaCry attack served as an example for cyber security professionals, general users, and companies, as malware continues to increase in volume, as previously mentioned. Interestingly enough, North Korea were blamed for the attack by several countries, including the United States of America [14].

2.3 Static and dynamic analysis

With malware having been explained in section 2.2, the question it poses would be how malware is analyzed. When talking about malware analysis, there are several methodologies and techniques to do so. The ones mentioned in this chapter will be *static and dynamic analysis*, and *automated and manual analysis*. This section will deal with static and dynamic analysis, while the next section will deal with automatic and manual analysis.

Static analysis, in brief, is the inspection of code without executing the code. There are tools that allow analysts to automate this process to some extent, although it also can be done manually. Static analysis can be done manually, by simply inspecting the code, its method calls, variables, libraries it asks for, and so on. However, this can also be achieved with the help of tools, both open-source and closed-source for personal or enterprise use [15].

Dynamic analysis is inspecting the code on run-time. Here, system or API calls can be analyzed, network traffic and enquiries the software makes can be put under scrutiny and this can be used to deduce a verdict: say, whether a program is malicious or intended for nefarious purposes [16].

Dynamic analysis can be done manually, using tools like WireShark to monitor network traffic while the program is running, or a debugger [17]. Automated tools for automated dynamic analysis include sandboxing, which will be explained in section 2.5, but also websites such as VirusTotal.com where a binary file, URL or IP address can be submitted by the user and consequently automatically scanned for malicious patterns or behavior, returning a full report of the scan and its contents. Honeypots are also tools that can be used for malware behavior, although honeypots are the first step of malware analysis: catching the malware. Honeypots are decoy systems that present themselves as real systems to lure attackers into attacking them [18]. The resulting malware binaries or system changes are then used by analysts to further analyze the malware which, depending on the type of malware, can be done in different ways as explained in this chapter.

2.4 Automatic and manual

Having spoken about static and dynamic analysis techniques, two types of analysis should also be emphasized: automatic and manual.

In brief, *automatic malware analysis* is the usage of tools to analyze malware with minimal user input. Sandboxing is an example of this [19].

Manual malware analysis, on the other hand, is where previous experiences or knowledge by the human analysts come into place. This includes techniques such as reverse engineering, where the user attempts to recreate the malware in question to gain a better understanding of its inner workings, by inspecting the source code manually. This process is also known as reverse engineering [20].

Another manual technique is *debugging*, where code is run in chunks; this is a technique through which the analyst would be able to insert so-called breakpoints, at which the code will stop running. This is done through the use of debuggers, which allow testing or examination of code or another program [17]. This method allows for more intricate insight into the inner workings of malware with regards to input and output, albeit requiring a significant amount of manual labor compared to some automated solutions.

Given the project's scope and problem statement set forth by us, automatic dynamic analysis was the chosen methodology for this project. Subsequently, sandboxing was chosen as the baseline, since it allows us to reach its objective mentioned in the problem statement, and can, if done in a proper manner, allow for customization to the extent of satisfying our needs completely.

The section below will therefore discuss an automatic dynamic analysis methodology known as sandboxing, the different tools thereof, and which tools the group ultimately chose to use.

2.5 Overview of sandboxing and its hypervisors

Having now looked into different types of viruses and their methods of propagation, it should become increasingly clear why the cyber defense continues to strengthen its arsenal, and why focus on defending assets continues to grow.

However, there are multiple ways of doing this: the arguably most popular method is called *sandboxing*, as mentioned in chapter 1. Sandboxing essentially allows for a virtualized, contained environment, in which malware can be run and its behavior analyzed. This has proven to be an excellent strategy [21] for analyzing how an attack exactly is conducted, or how a malware propagated, as well as making analysis on certain types of malware to better fortify existing systems and infrastructures.

However, before diving into the different sandboxing tools readily available, it is vital to gain a brief understanding of hypervisors, of which there are two types.

2.5.1 Hypervisors

When talking about hypervisors, there is a distinction between type 1 and type 2 hypervisors. Type 2 hypervisors will be discussed first.

Type 2 hypervisors distinguish themselves from type 1 hypervisors by deploying software that acts as a middle-man between the host operating system and the guest machine's operating system. This is depicted in figure 2.1 [22]. If the user of a hypervisor would like to virtualize a Windows 10 machine, the resources on the host computer would be provided to the guest in proxy [23]. For instance, the graphics card would not be directly provided to the guest machine; instead, it would request a portion of the graphics card through the Virtual Machine Monitor, retaining the physical graphics card in possession of the host's OS. The same goes for CPU cores, RAM allocation and PCI/PCIe devices that are emulated on the guest machine as if it was a physical one [23]. Examples of these hypervisors include VMWare [24] and Oracle VirtualBox [25], with both having free editions for non-commercial use.

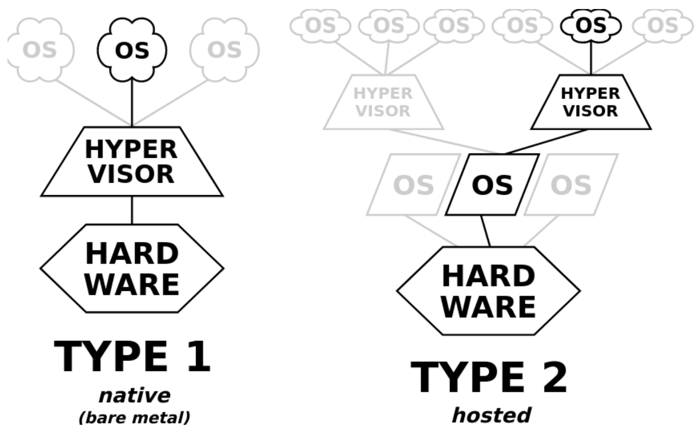


Figure 2.1: The difference between type 1 and type 2 hypervisors, visualized [22].

Type 1 hypervisors on the other hand, are also referred to as bare-bone or bare-metal hypervisors [23]. In the case of virtualizing a Windows 10 machine with, say, a single graphics card unit, as well as a single CPU, it would work the following way:

- For the GPU, for a graphically intense setup using Citrix' hypervisor, for instance, it would be allocated to the virtual machine. This is called a GPU pass-through [26]. However, contrary to the case of type 2 hypervisors, the graphics card unit would not be emulated, but the physical card itself is assigned to the virtual machine. If the host machine only has one graphics card unit, the guest machine would receive a so-called pass-through of said graphics card, and in usual cases only release it back to the host machine upon shutdown of the virtual machine. The host machine would however, based on our experience, still be accessible albeit headlessly through SSH or similar remote access protocols.
- For the CPU, virtualization works similarly in both cases, although more efficiently in the case of type 1 hypervisors [23] since the hypervisor communicates between the hardware and guest directly — see fig. 2.1 — rather than through the host operating system which otherwise could pose some virtualization limitations.
- As far as PCI/PCIe devices go, the process will differ from hypervisor to hypervisor. However, generally speaking, type 1 hypervisors give virtually

unhindered access to the devices whereas virtualization software for type 2 hypervisors may pose obstacles with certain hardware [27]

Both types have drawbacks and advantages, and the use case scenario defines whether type 1 or type 2 hypervisors are needed. For sandboxing, the virtualized system will depend on the needs. For this project, a Windows 7 machine running separately will be needed virtualized; this is needed so traffic can be monitored to and from the machine in isolation so the most accurate results can be acquired and conveyed in this report. In terms of virtualizing this machine, type 1 hypervisors, given our experience, require a lot of overhead, whereas type 2 hypervisors for the most part are plug-and-play. Given our needs and time constraints, a type 2 hypervisor will be used for deployment of the Windows 7 machine. Namely, VMWare Player will be used for this purpose.

With that in mind, sandboxing tools shall be discussed below before the group decides on which to use for this project.

2.5.2 Sandboxing tools

There are different types of sandboxing tools, both cloud-based and ones that can be run locally. Different tools provide different capabilities and certain limitations: for instance some locally-run sandboxing tools are free to use, whereas cloud-based ones would require a certain fee [28], perhaps even a recurring one, as to be compensated for the price for upkeep, maintenance of the space and consumption of resources in the cloud, and so on.

Below is a brief list of the sandboxing tools that have been considered for this project:

- SHADE Sandbox (local)
- Cuckoo (local)
- CAPE (local and cloud)

SHADE Sandbox is a sandboxing tool that caters towards an average user, who cares about security, more-so than focusing on experiments. It works for pre-existing applications on the host machine and allows for said applications to run in a containerized environment by locally virtualizing applications and locking internet files and potential malware [29].

It allows for creation of reports with the findings from running a certain application provided by the user, and is deemed "a lightweight protection layer" with its ease of access being a big selling point. SHADE also has enterprise support against what it views as general company-wide threats, and even offers compatibility with other vendors and product customization [30].

Cuckoo Sandbox is an extremely versatile sandboxing platform, and despite its latest version, at the time of writing being version 2.0.7, being from back in 2019 [31], it still holds up well, given our previous experiences with the platform. Cuckoo operates locally on the user's computer, but could also be deployed in a system on the cloud [32]. One of Cuckoo's trades is allowing different types of input to run in an encapsulated environment. Cuckoo can handle executable and binary files, documents, PDF files, e-mails, and URLs [33]; with URLs, it even opens up the browser automatically in the virtual machine, and attempts to actively instrument it [34].

Cuckoo also allows freedom in terms of setting up a virtual machine in the desired environment for the user. The options preferred by Cuckoo is VirtualBox, although it also allows physical machines; through some virtualization, it would be possible to create a virtual machine and have it behave as a physical machine, thus using Cuckoo that way. It also allows for other virtualization environments, such as VMWare, and claims to support most options on the market natively [35].

Cuckoo also automatically generates reports while analyzing and running the malware on the virtual machine provided by the user. This allows for easier understanding of the malware's ongoings and behavior, and theoretically would function well on a larger scale as a middle-man for malware behavior analysis and evasion technique analysis. It is able to analyze network traffic even when encrypted via SSL, and supports rerouting or dropping certain traffic packets and network traffic rules natively. It is also able to analyze memory usage of malware through either YARA [36] or Volatility [37], if configured to do so [31].

Compared to SHADE, Cuckoo works with files rather than applications, thus giving more options to its users. For instance, an executable can be run within Cuckoo, and, usually, within minutes, Cuckoo will create a report of all the recorded activity for the file [31]. If scaled, this theoretically would allow to be a baseline for trial-error probing with artefacts with efforts which hide the fact that it indeed is a sandbox, and seeing how those changes affects the malicious binaries. This, in essence, is the idea of the project from a more technical standpoint, and resulted in Cuckoo being the chosen tool for sandboxing.

CAPE is an independently made open source extension of Cuckoo, published by

Kevin O'Reilly [38]. It aims to add automated malware unpacking and extraction of configuration details to a Cuckoo platform, and a community version publicly and freely available exists. The community version is only an online version of the platform with limitations, as opposed to the so-called premium version which operates on a locally installed hypervisor; CAPE itself suggests using KVM for guest machine virtualization.

It mainly works through its debugger, which was built on the notion that instruction-level control was necessary for achieving its goal. The debugger aims to be efficient by maximizing use of processor hardware and minimizing the use of Windows' default debugging interfaces. It operates at a granular level, with hardware breakpoints set using API calls or YARA signatures. It uses the latter to detect unpacked payloads. [39]

CAPE essentially operates at the memory level, making it quite unique in how it achieves its goal. CAPE also includes tools to help combat anti-evasion malware; this type of malware will not run if it detects that it is in a virtualized environment. CAPE comes ready with tools that combat this dynamically, and takes advantage of malware techniques to be able to capture unpacked payloads [39]. The aforementioned tools all have advantage as well as drawbacks when the group selected their tool to go forward with and base the platform on.

Ultimately, in order to have as much control and freedom as possible, Cuckoo was chosen. SHADE works with applications rather than files, and while certain malware does hide itself in programs, it certainly would narrow the scope of the project. CAPE offered a good starting point from a theoretical point in terms of helping the group shape up the project from a technical standpoint, and will be used for comparison in chapter 6.

The idea, with which the group will go forward, is that the platform for this project is based on Cuckoo. CAPE will serve as a kind of benchmark for dynamic malware analysis and anti-evasion techniques, and will be compared to our platform once fully deployed. It also will be used for inspiration in terms of finding the right tools to add on top of Cuckoo.

2.6 Summary

This chapter gave an insight into malware as a concept, the different types of malware and provided two of the most notorious examples of malware, briefly mentioning the aftermath thereof. The chapter then delved deeper into the different types of malware analysis, and the techniques considered by us. Then, techniques to combat malware were discussed, and the options we would have going forward were presented. Ultimately, we deemed Cuckoo the most fitting sandboxing platform for the baseline of this project. With that in mind, the system and toolbox to be built for this project are presented in the coming chapters.

Chapter 3

Dynamic Analysis Evasion Techniques

3.1 Introduction

Having established malware, sandboxing and the different types of analysis that can be used to analyze malware, this chapter will deal with dynamic analysis evasion techniques. This is a focal point for the project, since malware alters its behavior based on the environment it finds itself in. In order to better understand how to use that knowledge to the defender's advantage, it is vital to understand which techniques there have been used over time. This chapter aims to clarify that, as well as outline the techniques in detail.

3.2 Related work

Malware is a topic that has been covered extensively in many reports and surveys. When discussing malware, *debugging* is a topic often mentioned.

In this regard, Sikorski and Honig [17] discuss the hands-on aspect of dissecting malware and working with debuggers. Moreover, Tyler Shields [40] discusses anti-debugging from a developer's point of view, in order to provide more insight into anti-debugging techniques which can be employed by malware looking to avoid detection.

Sandboxing is based around the idea of testing and observing malware or malicious files in a contained, safe environment. Of newer surveys, namely in 2019, [19] by Afianian et al. can be mentioned, and its findings will be a grand inspiration for the duration of this project.

Yoshioka et al. [41] discuss the impact of injecting decoys with regards to sandbox evasion. Additionally, Blackthorne et al. [42] discuss how malware behaves while in an emulated environment and techniques malware uses to combat such environments. Moreover, [43] presents and compares the usage of anti-debugging and anti-VM techniques across different malware families.

In terms of major inspiration behind the motive of this project, [44] by Muhovic can be mentioned. The research done in that project inspired us to extend on parts of it, while using other parts of said thesis as an inspiration and for building a foundation of knowledge ahead of this thesis.

3.3 Advantages and drawbacks

The complexity of malware nowadays means that some go beyond normal source code, but instead obfuscate it [19] to plant confusion on the analysts' end. This essentially renders that aspect of manual analysis useless. Although dynamic analysis relies on the malware executing its payload on run-time, it does have some advantages over static manual analysis that have been deemed worthwhile for the scope of this project. For instance, source code, such as the one used in SolarWinds section 2.2, could lay dormant, and thus dynamic manual analysis of the code itself would not suffice in laying the landscape for the program's functionality, in actuality. However, manual analysis, when compared to automatic analysis, could be more time-consuming depending on the task at hand.

These are just some of the challenges presented by the two types of evasion techniques, and thus both will be explained and discussed in this chapter.

3.4 Dynamic manual analysis evasion techniques

Starting off with dynamic manual analysis evasion techniques, these techniques would mostly target debuggers. In a sample size of 6,222 malware samples, 40 percent of malware run in the presence of a debugger albeit not in a virtualized environment would exhibit less malicious behavior [45]. Malware combating dynamic manual analysis will mostly deploy anti-debugger or anti-reverse-engineering techniques; the former can be done by using the debuggers' own techniques, e.g., trap flags or single stepping, against them, while the latter can be done through obfuscation of the code or by introducing excessive complexity [45].

Following, some techniques used for evasion of dynamic manual analysis will be explained. It is worth noting that anti-debugging techniques and anti-VM techniques are not necessarily more prevalent in targeted malware compared to generic malware, which could lead to unexpected results [43].

To start off with, fingerprinting techniques, mentioned below, will be explained in brief detail:

- Probing for system artefacts
- Search for breakpoints
- Analyzing Process Environment Block

3.4.1 Probing for system artefacts

One of the common methods for fingerprinting is probing for system artefacts, since virtualization software is known to leave traces that expose the fact that the system at hand is virtualized. In fact, there are multiple examples of malware fingerprinting virtualized systems [46], checking for the *isDebuggerPresent* flag among other flags. Other malware would simply look for a flag for a specific type of debugger, namely the process file, and see if it was running. Examples of this are [40] and [47]. In the example of [48], the malware would be more sophisticated; when running a virtualized environment, programs' parent processes would belong to, say, a debugger, instead of the Windows Explorer process (*explorer.exe*. Malware would check if the parent process matched the name of a known debugger as a fingerprinting technique, and act accordingly.

3.4.2 Search for breakpoints

When debugging, debuggers will at times set breakpoints, so code can be analyzed a couple of lines at a time. These breakpoints can be done in two ways: the hardware way or the software way. In the former's case, the breakpoint address could be saved into the CPU's register [19], and this can be read by malware [49] operating at a low abstraction level, such as the direct memory of the machine if running with escalated privileges. In terms of software breakpoints, some debuggers write a special INT3 instruction operation code called *0xCC* [19], which would expose it to malware iterating through flags in the system [50] [51]. Breakpoint searching was in 2019 the second-most 'popular' anti-debugging technique used by malware [19] [48].

3.4.3 Analyzing the Process Environment Block

Another rather low-level anti-debugging technique is analyzing the Process Environment Block, colloquially dubbed the PEB, being a per-process data structure in the user system. The PEB will in debugged systems have the flag 'BeingDebugged' [52], which malware can read to determine whether it is running in a debugged system, and thus alter its behavior.

3.5 Dynamic automatic analysis evasion techniques

For automated analysis environments, malware also use fingerprinting techniques, although for sandboxes and honeypots rather than debuggers. There are multiple parts of a system, which can be fingerprinted. The whitepaper [19] classifies these into several types.

The categories for fingerprinting, which will be dealt with in this section, are:

- Hardware-based fingerprinting
- Application- and position-based fingerprinting
- Network-setup-based fingerprinting

3.5.1 Hardware-based fingerprinting

It makes use of the emulated devices plugged in to the system, and depending on the virtual machine at hand, this can be quite easy to fingerprint [53]. For popular virtual machine hypervisors, artefacts pertaining to said manufacturer will by default exist in the system, such as "*VMToolsHook.dll*" [19] and devices such as the *VMXNET* driver [54] in the case of VMWare.

3.5.2 Application- and position-based fingerprinting

This fingerprinting technique is similar to that of system artefact probing for debuggers. Since applications tend to leave behind traces in, say, the registry keys or simply on the disc, malware can detect this [55]. For sandboxes altering data residing in the lower abstraction level such as the memory or kernel, the hooks linking control between the host and guest operating systems can be found and abused by malware [56].

3.5.3 Network-setup-based fingerprinting

It is done based on probing from the malware's side. Malware has been proven to probe the network by checking for fixed IP addresses usually used by certain sandboxing platforms [41]. These IP addresses can at times be achieved through reconnaissance work carried off before this part of an attack. Other giveaways from the network setup's side when malware is investigating could be the lack of an internet connection [42] or unrealistically fast speeds. The latter is usually the case for host to guest connections or host-only connections within hypervisors.

3.5.4 Other techniques used to evade sandboxes

As previously mentioned in section 2.2, certain malware operates in a dormant way. In a more technical sense, malware authors with that in mind are aware of the fact that sandboxes allocate a certain amount of time for analysis per sample [19]. Among the techniques used are stalling, in which malware halts its execution until after the analysis phase is over, thus evading the sandbox analysis and still infecting its target [57]. Other techniques include triggers based on certain facts, such as the system time and date [58], the title of a certain app appearing in a window, or other keyword triggers that will activate the malware in question [59].

3.6 Summary

This chapter began by outlining the work related to this project, and define part of the literature that would be used for the duration of it. The chapter then went on to discuss advantages and drawbacks for both types of analysis, namely manual and automatic, before delving deep into dynamic manual analysis and automatic analysis evasion techniques. Lastly, the chapter outlined other techniques that historically have been used to evade sandboxes. Having decided on Cuckoo, as mentioned in chapter 2, the next chapters will deal with defining the system used for this project as well as implementing Cuckoo and the relevant tools.

Chapter 4

The Tools, Requirements and Process

4.1 Introduction

This chapter will lay out the main points of the thought process behind the system specifications, as well as the design thereof. The chapter will begin by outlining the tools deemed useful for the project, the usage of said tools and what they aim to achieve with regards to the scope and vision of the project. How they relate to each other will also be mentioned in this chapter, and this chapter is considered the first part of the project's platform implementation. The technical aspects of the implementation have been explained more in depth in appendix B.

4.2 Motives behind the selected tools

For the project, as previously mentioned, virtualization is the key, see section 2.2. For purposes of preference, it was decided to use VMWare WorkStation Pro 16 and Eve-NG, which will host and handle the emulated corporate network tools and systems that pertain to the topology.

The guest machines, which Cuckoo will run malware samples on, will all be running genuine 32-bit versions of Windows 7 Ultimate. The Ultimate edition was chosen as it allows for granular control with regards to permissions and superuser control [60]. This is done to ensure that no differences in architecture or installation can be a cause behind differing results when testing malware.

Before delving deeper into the system's design and specifications, it is essential to understand what Eve-NG is, its function as well as the other tools made use of in this project.

EVE-NG is an emulated virtual environment for IT professionals with different backgrounds in the field of IT such as networking, security, DevOps, and NetDevOps. It allows security and network professionals to demonstrate a virtual proof of concept [61] with regards to, amongst other things, network setups and infrastructure. It provides a platform for "images" of other operating systems to both inter-communicate and be run and controlled through it, and has a plethora of compatible systems and software: from Cisco and Juniper products to Linux and Windows images.

Two versions of Eve-NG currently exist: the free-to-use 'Community' edition, and the premium option, i.e., the 'Professional' edition. Eve-NG can be installed either as a separate instance on a virtual machine, or bare-bone on a machine in the cloud, and thus allows for some flexibility with regards to its application. Despite the limitations within this choice, we opted for the Community edition of Eve-NG.

Eve-NG was also compared to other network emulation tools, such as GNS3 and VIRL from Cisco. First and foremost, GNS3 and VIRL have limitations and dependencies that are not present in VMWare, and VMWare would therefore in theory have less hurdles related to its setup process within the emulated corporate network we aimed to build. Moreover, there were differences in how the platforms operate [62], and we were already well acquainted with EVE-NG.

We have previous experience with EVE-NG compared to the two other aforementioned tools, meaning there would be less of a learning curve with the former. Moreover, EVE-NG, in a virtualized environment, cannot run on VirtualBox [63]. Thus, VMWare seemed a more logical decision.

Docker was considered an option when looking into alternatives to VMWare from a virtualization standpoint. In fact, one of our formal acquaintances had been working in parallel with a Docker-based solution for sandboxing, and had yielded successful results. Furthermore, using EVE-NG with Docker would require the Professional version of EVE-NG. This alongside multiple other factors meant that we discarded Docker for virtualizing the Cuckoo host and guest machines.

QEMU, a type 1 hypervisor [64], was opted for, alongside VMWare. This was due to VMWare's ease of use, as well as QEMU's native integration with Eve-NG. QEMU has been specifically chosen seeing as Eve-NG was the chosen topology emulator, and Eve-NG's community edition only allowed for QEMU images to be used. This eliminated the idea of using Docker, which was a Professional feature, and seeing as the QEMU-img utility allowed easy conversion from VMWare image files to QEMU. Another option would have been to use nested virtualization, if we wanted to use VMWare or VirtualBox, in the machine hosting Eve-NG. However, this meant less granular control in terms of artefacts as well as performance drop issues, and QEMU was favored in this regard.

We would create the machines in VMWare originally, then convert these to QEMU images, import them into the topology and run them using QEMU instances linked to EVE-NG. This is the case for the Weak and Blind Windows 7 instances. The Hardened machine was created strictly in QEMU.

Firstly, we do not want a headless setup for the sandboxing-relevant machines, as granular control is easier on a scalable setup if the GUI for those machines is present. Second, EVE-NG functions uses QEMU images for its machines; using Docker would have meant finding a workaround to that, with VMWare instead readily available. Most of the documentation for EVE-NG and Cuckoo is based on more traditional virtualization software such as VMWare and VirtualBox, and we did not feel a need to create further hurdles with regards to the setup.

One of the main goals of this thesis is to make a product that is resilient and scales well; this can be achieved through Docker too, but due to VMWare's popularity in terms of potential troubleshooting forums meant that we opted for a combination of VMWare and QEMU.

With that in mind, the next step in the process would be to build a resilient malware analysis platform, as mentioned in 2. We, with regards to the platform, decided to use the Cuckoo sandboxing platform. However, seeing as Cuckoo expects some kind of virtualization and, in our case, should be implemented in the network infrastructure, we needed a cloning or imaging server. This would be needed for storing the images of the Cuckoo host and guest machines. For this purpose, the FOG imaging server was used [65].

FOG is a free open source imaging server, known as the FOG Project. It helps in taking imaging of different operating systems. The FOG project binds together various open source tools, such as a PHP-based web interface for managing images and host machines. It does not boot the machine from the hard disk itself, but through the WOL (Wake-On-LAN) protocol using PXE (Pre-boot Execution Environment), with the help of TFTP (Trivial File Transfer Protocol) [65]. The boot process and use of PXE will be covered in chapter 5.

The FOG project was also compared to other open source cloning or imaging servers for instance CloneZilla, and Deepfreeze. However, we opted for FOG, since it is the preferred and tested server for Cuckoo, according to Cuckoo's own documentation [66].

Furthermore, the QEMU disk image utility was used to convert the ***.vmdk** disk to a ***.qcow2** disk. The QEMU disk image utility is a utility allowing for creation, management and conversion of image file formats, offline and in a QEMU-supported format. Since Eve-NG uses QEMU-based images, the utility was needed in order to provide the platform with images of an appropriate file format. [67].

Additionally, WinSCP, an open source file transfer tool was made use of. WinSCP is commonly used for transferring files or documents from a local machine to a remote machine; in this case, it allowed us to transfer files from the physical host machine to the virtual machine running Eve-NG. WinSCP supports different protocols, including the Secure File Transfer Protocol (SFTP) [68], which was used in this case.

Given that the envisaged setup is one with a network topology, a central point of communication is needed. For this, we have opted for a layer-3 switch through a Cisco IOS image. This has been chosen due to our previous positive experiences and knowledge of the inner workings of Cisco routers.

Another tool that will be made use of is Dionaea, a low-interaction honeypot that captures binary files of malware targeting certain protocols enabled in Dionaea [69] [70]. This tool will be deployed both locally in our network topology, and on a machine hosted on the cloud by DigitalOcean, a cloud machine hosting service [71].

The reason behind this, is that we wish to investigate whether malware that triggers protocols hosted by Dionaea will be captured by the Dionaea machine in the topology. Certain malware has been chosen with this in mind. This machine will be put in the same network segment, also known as VLANs, as the Windows machines. Whether this machine captures malware binaries from infected machines in the topology will be elaborated on and investigated in the upcoming chapters.

The machine hosted on the cloud will be facing public internet, contrary to the machine in our internal topology. The idea behind this is that the public-facing machine should capture binaries that trigger certain protocols in Dionaea, and these will then be tested in our topology to see whether this yields the same result if the Dionaea machine is not facing the Internet. The WannaCry ransomware is known to target the SMB protocol [11], and this is one that is expected to target the internal Dionaea machine.

We will also make use of WireShark, in order to monitor network traffic while the malware is running. Cuckoo, however, as part of its reporting, also creates a network traffic dump in form of a *.PCAP file, readable in WireShark. We will look into whether these two line up to determine the usability of said *.PCAP file. The information provided by said network dump will also be available in the reports, which can be found in appendix D.

4.3 The requirements

For Eve-NG to run properly, many resources are required. For now, Eve-NG is set-up on a computer. Eve-NG is set up on VMWare Workstation Pro 16, henceforth referred to as VMWare, as a virtual machine, where it has been allocated 32 gigabytes of RAM, 12 CPU cores, as well as 3 virtual hard drives with the sizes 50 gigabytes, 100 gigabytes and 300 gigabytes, respectively.

VMWare itself is running on a Microsoft Windows 10-based machine with 64 gigabytes of RAM and 3 SATA-hard drives with 500 gigabytes each, alongside a 1 terabyte solid state drive.

In summary, the following tools will be used for this thesis:

- VMWare WorkStation Pro 16 and QEMU
- The EVE-NG network emulator
- WinSCP for transferring the image files to EVE-NG
- Linux-distro, Ubuntu, on the Cuckoo and FOG host machine
- The Cuckoo sandboxing platform
- FOG imaging server
- Windows 7 on the Cuckoo guest machines
- A CISCO Layer-3 switch image, set up on EVE-NG
- YARA
- TCPDump

The following tools will be used for artefact modification of the Hardened machine, which will be explained in chapter 5.

- Pafish (Paranoid Fish, virtual environment detection tool [72])

- Al-Khaser v0.81 (deploys techniques used by malware to determine whether it is in a sandboxing or monitored environment) [73]
- SEMS (Anti-Sandbox and Anti-Virtual Machine Tool) [74]

We will elaborate on how the last five tools will be used in the context of this project, and what they achieve, in chapter 5. For more technical details as far as their set up goes, see appendix B.

4.3.1 Virtual network connection types

Before explaining the process, it is essential to understand the virtual connection types, which a VMWare Workstation uses in order to communicate the nodes with each other and with the internet, depending on the end user requirements. For these virtual connections the basic principle for communication is the same whether they are virtual or physical. There are three network connection modes when setting up a virtual NIC (Network Interface Card) for a virtual machine [75].

Bridged connection or VMnet0 In the bridged connection mode, a VM connects to the physical network directly through the host's physical NIC. The host NIC is the bridge to all the virtual machines, similarly to the host machine and other physical machines on the network. Virtual machines obtain their IP address from the DHCP server on the physical network. When connected using the bridged connection mode, a virtual machine appears as just another node on the physical network. Thus, all the computers, virtual or physical all get their IP addresses allocated from the host's NIC. Therefore, the virtual machines on the network are accessible and directly connected to each other [76].

NAT (Network Address Translation) or VMnet8 The next connection mode that will be explained is Network Address Translation, colloquially referred to as NAT. In this mode, the virtual machine relies on the host to work as a NAT device. With NAT mode, a virtual DHCP server is responsible for assigning the IP addresses to the virtual machines, creating a sub-net separated from the physical network. Other physical machines get their IP address from the physical DHCP server, which acts as an external network. The host NIC acts as a NAT device between these two networks, it translates the IP address from the virtual machines to the IP address of the host, to reach the internet from the virtual machine. It also listens to the returning traffic for the virtual machine. It is mainly useful in a scenario, where the client virtual machine would access the internet [77].

Host-only Connection or VMnet1 The last and third connection type is a host-only connection. In this mode, the virtual machines are assigned IP addresses by the virtual DHCP server. The virtual machines can communicate with each other on the network but cannot communicate beyond or outside of the network. This connection is useful when wanting to set up an isolated virtual private network for doing analysis of malware, or experimenting with something that should not spread across the network [78].

4.3.2 The chosen network connection type

Initially, we decided to use the host-only connection as it provides an isolated private network for malware analysis, However it would not allow the EVE-NG virtual machine to communicate with the internet, and also the node within the topology on the EVE-NG virtual machine.

Therefore, the *NAT* connection type was preferred as it provides a separate sub-net from the host network and enables the virtual machine to communicate with the internet for installing dependencies.

The sub-net used for NAT connection mode is *192.168.45.0/24*, and the static IP address used to access the web interface of the EVE-NG is *192.168.45.128* and the gateway address is *192.168.45.1/24*.

The inter-connectivity between the nodes on the internal network used for testing, as depicted in 4.1, is handled and created through a sub-net on EVE-NG itself.

4.4 The process

We installed the community edition of the EVE-NG operating system, which comes pre-packaged with extensions and features catered to integration in VMWare [79]. Once this was imported and installed on VMWare, we proceeded to import images of the systems needed to build the emulated corporate network, as can be seen in figure 4.1.

The network, hence the figure, will contain a Cuckoo sandboxing host machine as well as a guest machine; the host machine will be running Ubuntu, whereas the guest machines will be a virtual machines running Windows 7. The reason why these are run separately and the Windows 7 machines are not virtualized within Cuckoo is that this setup provides separation of network traffic in the logs. This allows a clearer read-out when we analyze potentially malicious traffic among nodes on the emulated corporate network.

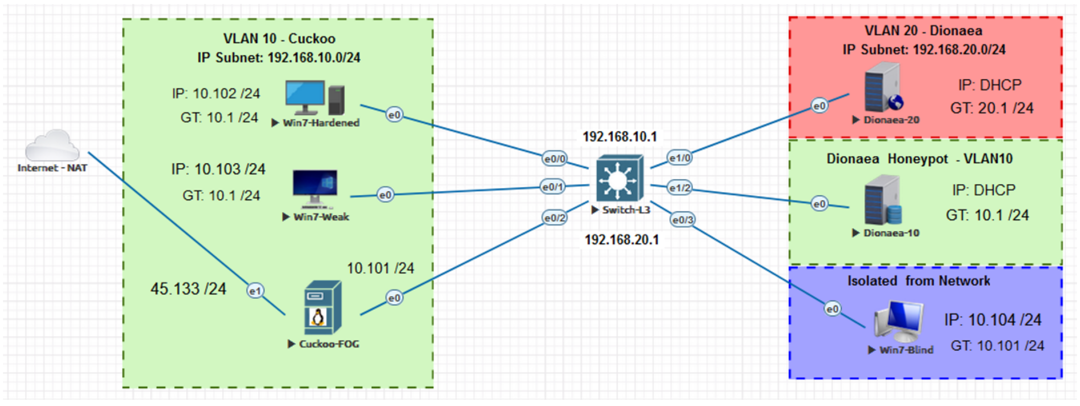


Figure 4.1: The network topology for the project.

It also is worth noting that Cuckoo and FOG run on the same Ubuntu-based machine, as denoted in fig. 4.1. FOG is used to restore the Windows 7 machines on which malware is run and tested, to a clean slate after each analysis is conducted. EVE-NG is complemented by its Windows integration pack, allowing for the topology to function fully with the Windows guest machines.

The technical side of preparing the images for EVE-NG, functions like so, using VMWare, QEMU-img and WinSCP. The following example regards the Windows 7 machine, although the process was repeated for the Ubuntu machine running Cuckoo and FOG. For more details, see appendix B:

- The image is acquired in the form of an *.ISO file
- The ISO image is installed on a virtual machine on VMWare
- Once the image is installed, a *.VMDK (virtual hard disk) file is retrieved through VMWare
- This file is then converted to the *.QCOW2 format using the QEMU-img utility.
- WinSCP is used to upload the newly converted files to Eve-NG's platform.
- The machine running said image will now be manageable through Eve-NG

The conversion and uploading processes will be described in further detail in appendix B.

4.4.1 The network infrastructure

The network infrastructure has been built with two VLANs configured: VLAN 10 and VLAN 20. This allows for network segregation and mitigates the potential risk of malware spreading across devices on the network; this way they, for the most part, would only be able to spread to devices on the same VLAN. They would only be able to cut across VLANs if the layer 3 switch is configured to do so. In this architecture, the switch is not configured to allow that, seeing as we believe an easier solution would be to simply allocate the machines in question the same VLAN.

The switch is in place to handle DHCP configuration for the subnets; it sets IP addresses with the following ranges excluded from the DHCP assignment. This was done to leave a buffer for more static devices, i.e. the layer-3 switch and the Ubuntu machine running Cuckoo and FOG. DHCP options number 66 and 67 have been set manually by us in order to allow for PXE or Wake-On-LAN booting. This allows Eve-NG to function as a manager of sorts, making nodes manageable through the platform itself.

- 192.168.10.100 to 192.168.10.110 for nodes on VLAN 10
- 192.168.20.1 to 192.168.20.101 for nodes on VLAN 20

WireShark [80] is used to capture the network traffic packets and keep an eye on the traffic going back and forth between the different network nodes. This will also be used to analyze network behavior for the virtual machine that will be used for testing and analyzing the malware binaries.

4.5 Summary

This chapter covered the groundwork for the upcoming Architecture chapter. In detail, the chapter presented tools deemed useful in helping us achieve our goals set forth with this thesis. The tools' place in the project, why they took precedence, and which role they would assume was then explained in brief detail. Lastly, the chapter covered converting the operating system images to a format usable by EVE-NG, as well as the technical aspect of the network infrastructure that pertains to the project.

Chapter 5

Architecture

5.1 Introduction

This chapter aims to further explain how the concepts and tools explained in chapter 4 were introduced from a technical standpoint to the project. The chapter will begin by laying the foundation for the implementation itself, describing the tools needed in a concise way. Then, it will delve deeper in how exactly each tool was implemented. In this chapter, the sandboxing environment including the relevant tools are set up, with the entire process explained in detail. The chapter also attempts to demonstrate how the problem statement in chapter 1 has been initially formulated from a technical standpoint.

5.2 Setting up

To start with the chapter of architecture and implementation, initially, a foundation on which the malware analysis platform could be built was needed. Dynamic malware analysis through sandboxing required virtualization to some extent, in which the malware could run as if it was deployed in a real environment, as described in section 2.5.

The previous chapter, which describes a brief overview of system requirements, tools and why they were selected, acts partially as the foundation for the rest of this chapter. This chapter digs deeper into the technical implementation of these tools. In the following subsection, the technical setup of the following tools will be described, with their context briefly reiterated to ensure a more coherent reading experience:

- VMWare Workstation Pro 16
- EVE-NG community Edition
- Cuckoo Sandbox
- FOG Imaging Server

5.2.1 Setting up VMWare Workstation Pro 16

Setting up VMWare Workstation Pro 16 Windows 10 is fairly straightforward. The installation guide can be found in [24].

5.2.2 The requirements for EVE-NG

EVE-NG comes in OVF (The Open Virtualization Format), which is an open standard for packaging and distributing virtual appliances and ISO files. It can be deployed or installed on different virtualization platforms and software, and the chosen platform to install EVE-NG on was VMWare.

Furthermore, the recommended hardware requirements for EVE-NG are as follows [63]:

- CPU: Intel CPU supporting Intel(R) VT-x/EPT virtualization
- Operating System: Windows 10 or a Linux desktop distribution
- VMWare Workstation, version 12.5 or later

The full recommended requirement sheet can be seen in table 5.1.

CPU	8/1 (Number of processors/Number of cores per processor) Enabled Intel VT-x/EPT virtualization engine
RAM	24 GB or more
HDD Space	200 GB or more
Network	VMware NAT or Bridged network adapter

Table 5.1: Requirements for the virtual machine setup running EVE-NG

5.3 The tools and their role

With the above requirements, the PC will be able to run a small to medium topology, though the performance and quality of the topology depends on the number of nodes deployed. Images for the nodes in the topology are also needed. EVE-NG provides some of the images, although the project requires custom images in such a way that they are configured fully, with regards to the problem statement at hand. These images are obtained from different sources and are elaborated on in detail in appendix B.

VMWare is used to initially set up the virtual machines. Seeing as it offers VMWare Tools and thus a more fluent user experience when setting up the machines, it ensures a more time-efficient setup of the topology's machines.

Cuckoo is used as the sandboxing platform itself, handling the malware behavior analysis side of the process. Cuckoo communicates with the machines, runs the binaries in question, then generates a report based on the malware's behavior on runtime. Cuckoo then communicates with FOG to ensure proper reboot of the machine.

As part of installing Cuckoo, various tools have been downloaded and installed, for instance:

- **YARA** for Malware identification and classifications
- **Volatility** for forensic analysis on memory dumps
- **MongoDB** used by cuckoo for database management
- **TCPDump**, the network packet analysis module used for the network dump files referenced by the reports generated by Cuckoo [81].

Cuckoo, when it analyzes malware, maps them up to signatures provided and maintained by its community [66]. It analyzes the malware's behavior and the generated report provides information as to how many signatures matched the behavior shown by the malware during the sample run [31].

YARA complements Cuckoo very well [36], by having rules that are based on a continuously expanding large database of malware samples' signatures and behavioral patterns [82] that can be *matched* based on a binary file's behavior. This has been deemed very important to the project, as there may be malware that Cuckoo signatures will match but YARA rules will fail to recognize, and vice versa. This is something we have noticed in some sample runs conducted post-installation.

Furthermore, Volatility, which is Cuckoo's memory dump module, has been discarded for this project, as Cuckoo does not have memory dump support for physical machines [37]. This decision, and its subsequent drawbacks, will be addressed in chapter 7.

FOG itself handles the imaging of the virtual machines. FOG allows to capture a live image of the Windows machines with the desired configurations, and then revert the machines to the captured states after the binaries are run. This happens through Cuckoo, which tells FOG when the malware analysis is over. Cuckoo will request the machine to reboot, and once rebooted, FOG will handle the re-installation of the pre-captured image through the PXE network boot described previously in chapter 4.

Dionaea has been set up, as mentioned in chapter 4, in order to allow us to:

- a) Acquire more recent binary files
- b) Test these malware files in the internal topology
- c) For the local Dionaea instance: see whether the other machines will probe the machine and fall into the honeypot trap
- d) For the public facing machine: see a difference in Dionaea's efficiency when it is in an internal network with no internet access compared to it being publicly facing the Internet thereby making it visible to crawlers that traverse the web for vulnerable machines

Dionaea has been hosted on two machines running Ubuntu 18.04 in exactly the same manner both locally and on the cloud instance hosted on DigitalOcean's servers. Two machines host Dionaea, one in VLAN 10, with the other machines, and one in VLAN 20.

5.4 The Topology

In order for the project to be representative of malware's behaviors in different environments, we have opted to install multiple machines onto the topology.

The topology can be seen below, in fig. 5.1 with each node signifying a machine.

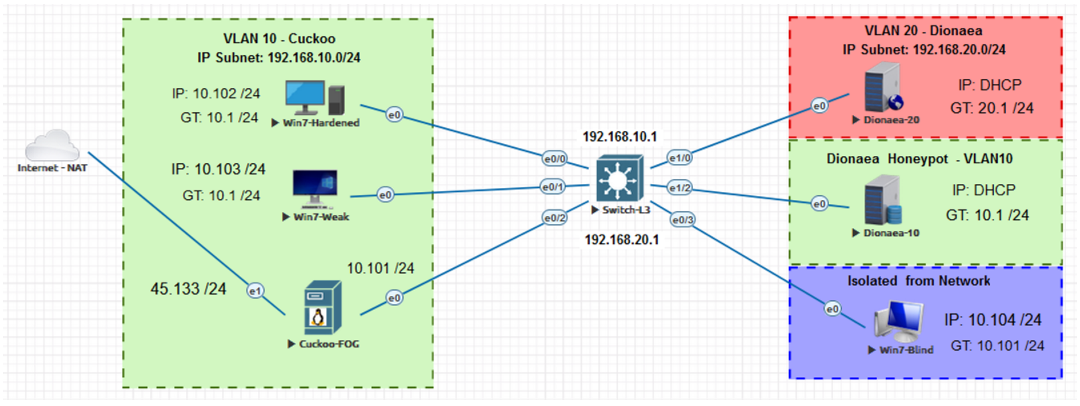


Figure 5.1: The network topology for the project.

- In the aforementioned figure, the green color, seen on the left and right, represents VLAN10, which is named Cuckoo
- The red box, shown on the right, represents VLAN20, named Dionaea.
- These two VLANs are then connected by the Cisco switch, which acts as the default gateway for both the VLANs. This can be seen in the center of the topology.
- The blue area, on the right, in the topology signifies nodes isolated from the rest of the network, in this case Win7-Blind
- The Cuckoo-FOG machine is a Linux-based machine running Ubuntu 18.04, and hosting EVE-NG and the FOG server. This machine is also connected to the Internet-NAT interface, in order to provide internet access, as we needed it for the installation of different dependencies.
- The three Windows machines which are referred to as physical machines are all running a 32-bit genuine copy of Windows 7 Ultimate.
- Windows 7-Hardened is a machine that has had system artefacts removed to masquerade its status as a virtual machine in an analytical environment. How this is specifically implemented can be seen in appendix B. This machine has access to the internal network, but not to the Internet.
- Windows 7-Weak is a machine that has not had system artifacts removed. This machine has access to the internal network, but not to the Internet.
- Windows 7-Blind is an identical copy of Windows 7-Weak. However, this machine does not have access to the internal network nor to the Internet.

- The last machine in the same VLAN, VLAN10, is an Ubuntu 18.04-based machine running Dionaea.
- In a separate VLAN, a Dionaea instance has been placed.

This has been done to ensure scalability in the project, seeing as machines easily can be added to a different VLAN and thereby segregated. This is the case for a Dionaea instance identical to the one in VLAN10, in VLAN20. This has been added merely to test whether malware would cross VLANs if not configured to do so on the layer-3-switch. If desired, it would also be possible to enable VLAN crossing on the L3 switch itself, although we have deemed this redundant, as previously mentioned. Instead, we opted for including all machines, barring the one Dionaea instance, in the same VLAN.

5.4.1 The physical machines

This section describes the configuration of the three Windows 7 VMs as depicted in fig. 4.1. In our case, we will configure them in an emulated corporate network infrastructure, and we will also use the term physical machines [83] to refer to these three machines going forward.

The following changes are made to all of the three machines. These changes are are required by the Cuckoo host machine in order to communicate with them [84].

- **Python** : Allows the agent to run and collect data from the physical machine
- **Pillow (Python Imaging Library)** : This adds capability to the python to take snapshots, and is used to take snapshots from the physical machine desktop.
- **Cuckoo Agent - older version** : Allows the VM to send data back and forth to the Cuckoo host machine. We used the older version of the Cuckoo agent, because only this version works with physical machine. The agent file is a Python script, which should be executed upon startup. This ensures that the Cuckoo host machine can communicate with the guest.
- **Enable auto-logon** (Allows for Cuckoo the agent to start upon reboot)
- **Enable Remote RPC** (Allows for Cuckoo to reboot the sandbox using RPC).

Physical machine: Win7-Weak

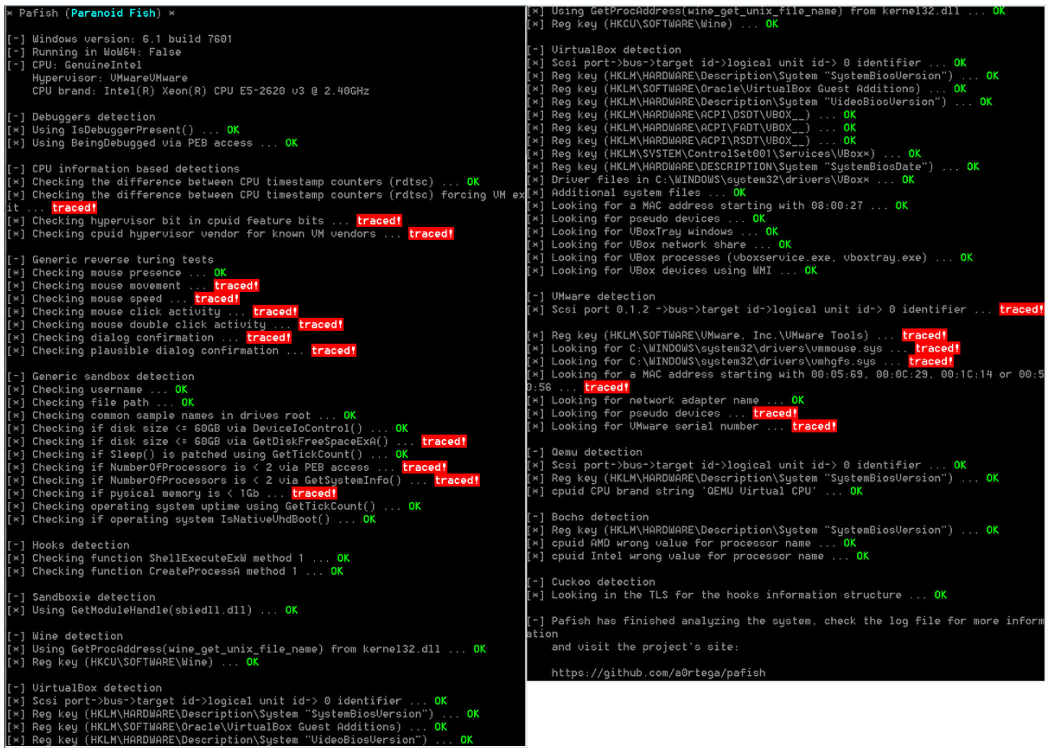
The Win7-Weak machine is left with the default aforementioned configuration, the default gateway, and the DNS are set to the switch IP instead of the host machine IP address, allowing it to send traffic to the network.

Physical machine: Win7-Blind

The Win7-Blind machine has also the aforementioned configuration. However, to isolate it from the network the following configuration was added to the switch in order to make it communicate only with the host machine, without having any interaction with the network. This is achieved through the VLAN Access Control List and VLAN access map by configuring them in the switch's global configuration mode. This will make Win7-blind to see only the Cuckoo host machine and not other machines on the network [85]. Additionally, the default gateway and the DNS for the machine are set to the host machine IP.

Pafish is a tool that detects the virtual machine or malware analysis environments with different techniques that malware usually uses to see them. Pafish can be downloaded from here [72]. The process of installing them on the hardened machine and removing the artefacts from the machine is straightforward [44]. Additionally, we could not remove all the artefacts from the machine since every tool looked for different artefacts for a virtual machine, and some of them were not possible to be mitigated.

However, fig. 5.2 will depict the results of Pafish before and after the mitigation of the found artefacts.

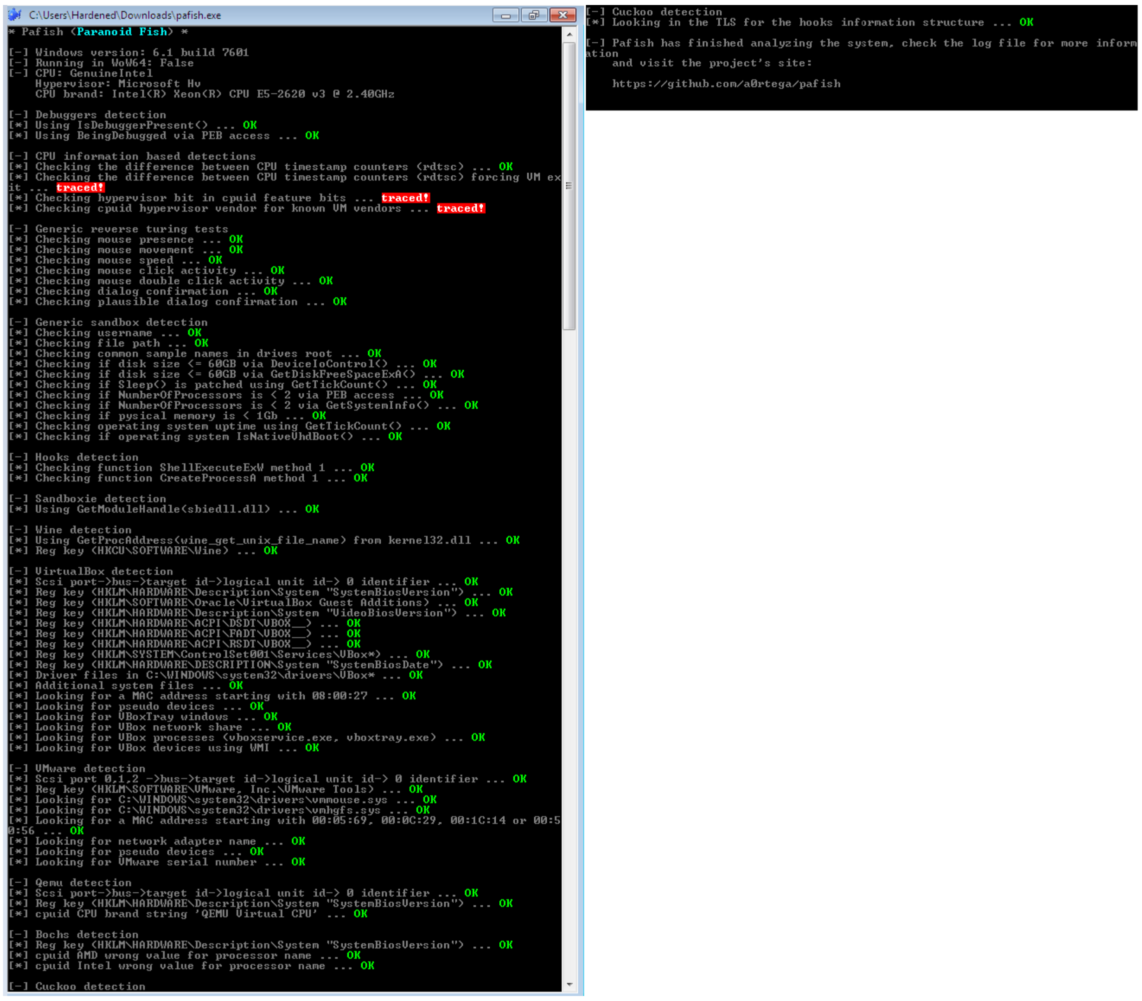


```
* Pafish (Paranoid Fish) *

[-] Windows version: 6.1 build 7601
[-] Running in WoW64: False
[-] CPU: GenuineIntel
    Hypervisor: VMwareVMware
    CPU brand: Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK
[*] Using BeingDebugged via PEB access ... OK

[-] CPU information based detections
[*] Checking the difference between CPU timestamp counters (rdtsc) ... OK
[*] Checking the difference between CPU timestamp counters (rdtsc) forcing VM ex
it ... traced!
[*] Checking hypervisor bit in cpuid feature bits ... traced!
[*] Checking cpuid hypervisor vendor for known VM vendors ... traced!

[-] Generic reverse turing tests
[*] Checking mouse presence ... OK
[*] Checking mouse movement ... traced!
[*] Checking mouse speed ... traced!
[*] Checking mouse click activity ... traced!
[*] Checking mouse double click activity ... traced!
[*] Checking dialog confirmation ... traced!
[*] Checking plausible dialog confirmation ... traced!

[-] Generic sandbox detection
[*] Checking username ... OK
[*] Checking file path ... OK
[*] Checking common sample names in drives root ... OK
[*] Checking if disk size <= 60GB via DeviceIoControl() ... OK
[*] Checking if disk size <= 60GB via GetDiskFreeSpaceExA() ... traced!
[*] Checking if Sleep() is patched using GetTickCount() ... OK
[*] Checking if NumberOfProcessors is < 2 via PEB access ... traced!
[*] Checking if NumberOfProcessors is < 2 via GetSystemInfo() ... traced!
[*] Checking if pysical memory is < 1Gb ... traced!
[*] Checking operating system uptime using GetTickCount() ... OK
[*] Checking if operating system IsNativeVhdBoot() ... OK

[-] Hooks detection
[*] Checking function ShellExecuteExW method 1 ... OK
[*] Checking function CreateProcessA method 1 ... OK

[-] Sandboxie detection
[*] Using GetModuleHandle(sbiedll.dll) ... OK

[-] Wine detection
[*] Using GetProcAddress(wine_get_unix_file_name) from kernel32.dll ... OK
[*] Reg key (HKCU\SOFTWARE\Wine) ... OK

[-] VirtualBox detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] Reg key (HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions) ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "VideoBiosVersion") ... OK
```

```
[*] Using GetProcAddress(wine_get_unix_file_name) from kernel32.dll ... OK
[*] Reg key (HKCU\SOFTWARE\Wine) ... OK

[-] VirtualBox detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] Reg key (HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions) ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "VideoBiosVersion") ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\DSDT\VBOX__) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\FADT\VBOX__) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\RSDT\VBOX__) ... OK
[*] Reg key (HKLM\SYSTEM\ControlSet001\Services\VBox*) ... OK
[*] Reg key (HKLM\HARDWARE\DESCRIPTION\System "SystemBiosDate") ... OK
[*] Driver files in C:\WINDOWS\system32\drivers\VBox* ... OK
[*] Additional system files ... OK
[*] Looking for a MAC address starting with 08:00:27 ... OK
[*] Looking for pseudo devices ... OK
[*] Looking for VBoxTray windows ... OK
[*] Looking for VBox network share ... OK
[*] Looking for VBox processes (vboxservice.exe, vboxtray.exe) ... OK
[*] Looking for VBox devices using WMI ... OK

[-] VMware detection
[*] Scsi port 0,1,2 ->bus->target id->logical unit id-> 0 identifier ... traced!

[*] Reg key (HKLM\SOFTWARE\VMware, Inc.\VMware Tools) ... traced!
[*] Looking for C:\WINDOWS\system32\drivers\vmmouse.sys ... traced!
[*] Looking for C:\WINDOWS\system32\drivers\vmhgfs.sys ... traced!
[*] Looking for a MAC address starting with 00:05:69, 00:0C:29, 00:1C:14 or 00:5
0:56 ... traced!
[*] Looking for network adapter name ... OK
[*] Looking for pseudo devices ... traced!
[*] Looking for VMware serial number ... traced!

[-] Qemu detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] cpuid CPU brand string 'QEMU Virtual CPU' ... OK

[-] Bochs detection
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] cpuid AMD wrong value for processor name ... OK
[*] cpuid Intel wrong value for processor name ... OK

[-] Cuckoo detection
[*] Looking in the TLS for the hooks information structure ... OK

[-] Pafish has finished analyzing the system, check the log file for more inform
ation
    and visit the project's site:

    https://github.com/a0rtega/pafish
```

Figure 5.2: Pafish detecting artefacts before mitigation

In the aforementioned figure, we can see that Pafish has detected all the artefacts on hardened machine using different techniques described in chapter 3. fig. 5.3 will now show the results after mitigating these artefacts.



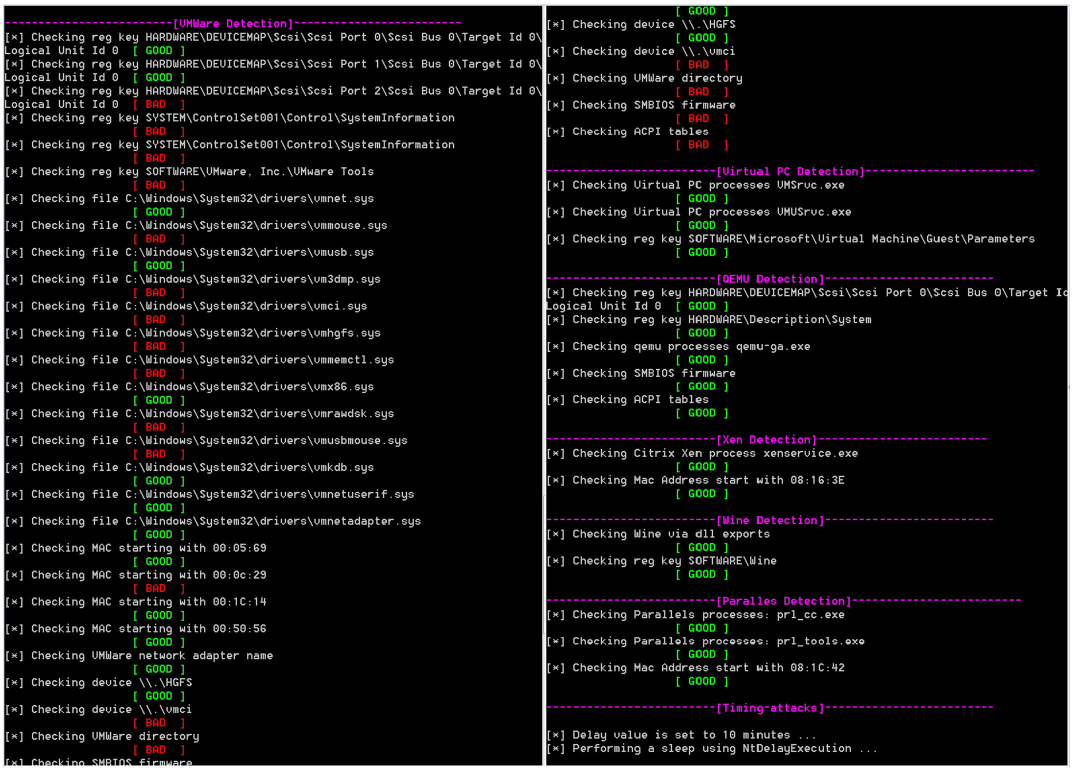
```
* Pafish (Paranoid Fish) *

[-] Windows version: 6.1 build 7601
[-] Running in WoW64: False
[-] CPU: GenuineIntel
    Hypervisor: Microsoft Hv
    CPU brand: Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK
[*] Using BeingDebugged via PEB access ... OK

[-] CPU information based detections
[*] Checking the difference between CPU timestamp counters (rdtsc) ... OK
[*] Checking the difference between CPU timestamp counters (rdtsc) forcing VM ex
it ... traced!
[*] Checking hypervisor bit in cpuid feature bits ... traced!
[*] Checking cpuid hypervisor vendor for known VM vendors ... traced!

[-] Generic reverse turing tests
[*] Checking mouse presence ... OK
[*] Checking mouse movement ... OK
[*] Checking mouse speed ... OK
[*] Checking mouse click activity ... OK
[*] Checking mouse double click activity ... OK
[*] Checking dialog confirmation ... OK
[*] Checking plausible dialog confirmation ... OK

[-] Generic sandbox detection
[*] Checking username ... OK
[*] Checking file path ... OK
[*] Checking common sample names in drives root ... OK
[*] Checking if disk size <= 60GB via DeviceIoControl() ... OK
[*] Checking if disk size <= 60GB via GetDiskFreeSpaceExA() ... OK
[*] Checking if Sleep() is patched using GetTickCount() ... OK
[*] Checking if NumberOfProcessors is < 2 via PEB access ... OK
[*] Checking if NumberOfProcessors is < 2 via GetSystemInfo() ... OK
[*] Checking if pysical memory is < 1Gb ... OK
[*] Checking operating system uptime using GetTickCount() ... OK
[*] Checking if operating system IsNativeVhdBoot() ... OK

[-] Hooks detection
[*] Checking function ShellExecuteExW method 1 ... OK
[*] Checking function CreateProcessA method 1 ... OK

[-] Sandboxie detection
[*] Using GetModuleHandle(sbiedll.dll) ... OK

[-] Wine detection
[*] Using GetProcAddress(wine_get_unix_file_name) from kernel32.dll ... OK
[*] Reg key (HKCU\SOFTWARE\Wine) ... OK

[-] VirtualBox detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] Reg key (HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions) ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "VideoBiosVersion") ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\DSDT\VBOX__) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\FADT\VBOX__) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\RSDT\VBOX__) ... OK
[*] Reg key (HKLM\SYSTEM\ControlSet001\Services\VBox*) ... OK
[*] Reg key (HKLM\HARDWARE\DESCRIPTION\System "SystemBiosDate") ... OK
[*] Driver files in C:\WINDOWS\system32\drivers\VBox* ... OK
[*] Additional system files ... OK
[*] Looking for a MAC address starting with 08:00:27 ... OK
[*] Looking for pseudo devices ... OK
[*] Looking for VBoxTray windows ... OK
[*] Looking for VBox network share ... OK
[*] Looking for VBox processes (vboxservice.exe, vboxtray.exe) ... OK
[*] Looking for VBox devices using WMI ... OK

[-] VMware detection
[*] Scsi port 0,1,2 ->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\SOFTWARE\VMware, Inc.\VMware Tools) ... OK
[*] Looking for C:\WINDOWS\system32\drivers\vmmouse.sys ... OK
[*] Looking for C:\WINDOWS\system32\drivers\vmhgfs.sys ... OK
[*] Looking for a MAC address starting with 00:05:69, 00:0C:29, 00:1C:14 or 00:5
0:56 ... OK
[*] Looking for network adapter name ... OK
[*] Looking for pseudo devices ... OK
[*] Looking for VMware serial number ... OK

[-] Qemu detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] cpuid CPU brand string 'QEMU Virtual CPU' ... OK

[-] Bochs detection
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] cpuid AMD wrong value for processor name ... OK
[*] cpuid Intel wrong value for processor name ... OK

[-] Cuckoo detection
```

```
[-] Cuckoo detection
[*] Looking in the TLS for the hooks information structure ... OK

[-] Pafish has finished analyzing the system, check the log file for more inform
ation
    and visit the project's site:

    https://github.com/a0rtega/pafish
```

Figure 5.3: Pafish detecting artefacts after mitigation

Al-Khaser v0.81 is another tool and a proof of concept, which probes the system in order to see if the system can be flagged as a sandboxing environment or not. The source code and the artifacts it looks for can be found on [73].

This tool uses several malware techniques, which are used by malware found in the wild to detect whether a malware is running in the virtual environment [73], such as anti-debugging, emulation and sandbox detection, and anti-virtualization. However, among all the detection types, the ones deemed noteworthy for this project are anti-VM detection and anti-sandboxing detection. Since this tool produces a very large report and thus would not be feasible to include in this report, we will show the results relevant to the project. Furthermore, a decision was made to not remove all the artefacts Al-Khaser flagged on the machine; for instance removing the fingerprinting of CPU voltage was discarded, with this among others having been deemed out of scope.

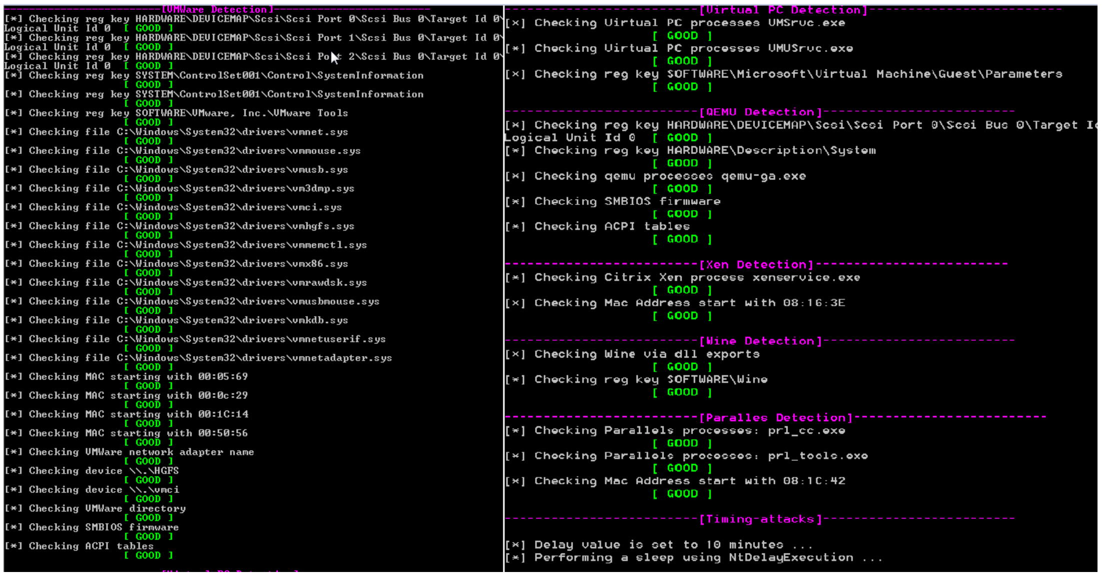


```
------------------------[VMWare Detection]------------------------
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\
Logical Unit Id 0  [ GOOD ]
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 1\Scsi Bus 0\Target Id 0\
Logical Unit Id 0  [ GOOD ]
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 2\Scsi Bus 0\Target Id 0\
Logical Unit Id 0  [ BAD  ]
[*] Checking reg key SYSTEM\ControlSet001\Control\SystemInformation
                   [ BAD  ]
[*] Checking reg key SYSTEM\ControlSet001\Control\SystemInformation
                   [ BAD  ]
[*] Checking reg key SOFTWARE\VMware, Inc.\VMware Tools
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmnet.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmmouse.sys
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmusb.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vm3dmp.sys
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmci.sys
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmhgfs.sys
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmmemctl.sys
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmx86.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmrawdsk.sys
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmusbmouse.sys
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmkdb.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmnetuserif.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmnetadapter.sys
                   [ GOOD ]
[*] Checking MAC starting with 00:05:69
                   [ GOOD ]
[*] Checking MAC starting with 00:0c:29
                   [ BAD  ]
[*] Checking MAC starting with 00:1C:14
                   [ GOOD ]
[*] Checking MAC starting with 00:50:56
                   [ GOOD ]
[*] Checking VMWare network adapter name
                   [ GOOD ]
[*] Checking device \\.\HGFS
                   [ GOOD ]
[*] Checking device \\.\vmci
                   [ BAD  ]
[*] Checking VMWare directory
                   [ BAD  ]
[*] Checking SMBIOS firmware
```

```
                   [ GOOD ]
[*] Checking device \\.\HGFS
                   [ GOOD ]
[*] Checking device \\.\vmci
                   [ BAD  ]
[*] Checking VMWare directory
                   [ BAD  ]
[*] Checking SMBIOS firmware
                   [ BAD  ]
[*] Checking ACPI tables
                   [ BAD  ]

------------------------[Virtual PC Detection]------------------------
[*] Checking Virtual PC processes VMSrvc.exe
                   [ GOOD ]
[*] Checking Virtual PC processes VMUSrvc.exe
                   [ GOOD ]
[*] Checking reg key SOFTWARE\Microsoft\Virtual Machine\Guest\Parameters
                   [ GOOD ]

------------------------[QEMU Detection]------------------------
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id
Logical Unit Id 0  [ GOOD ]
[*] Checking reg key HARDWARE\Description\System
                   [ GOOD ]
[*] Checking qemu processes qemu-ga.exe
                   [ GOOD ]
[*] Checking SMBIOS firmware
                   [ GOOD ]
[*] Checking ACPI tables
                   [ GOOD ]

------------------------[Xen Detection]------------------------
[*] Checking Citrix Xen process xenservice.exe
                   [ GOOD ]
[*] Checking Mac Address start with 08:16:3E
                   [ GOOD ]

------------------------[Wine Detection]------------------------
[*] Checking Wine via dll exports
                   [ GOOD ]
[*] Checking reg key SOFTWARE\Wine
                   [ GOOD ]

------------------------[Paralles Detection]------------------------
[*] Checking Parallels processes: prl_cc.exe
                   [ GOOD ]
[*] Checking Parallels processes: prl_tools.exe
                   [ GOOD ]
[*] Checking Mac Address start with 08:1C:42
                   [ GOOD ]

------------------------[Timing-attacks]------------------------

[*] Delay value is set to 10 minutes ...
[*] Performing a sleep using NtDelayExecution ...
```

Figure 5.4: Al-Khaser detecting artefacts before mitigation

The artefacts deemed relevant were analyzed by us and subsequently removed. The difference shown in fig. 5.4 and fig. 5.5 is the artefacts removed. The ones retained have been deemed out of scope.



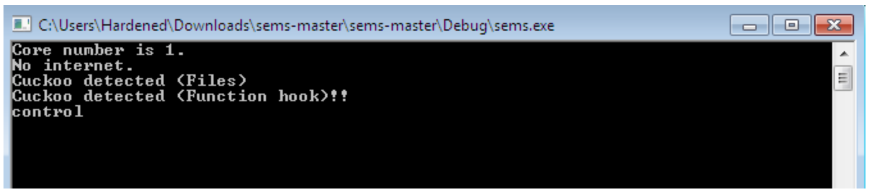
```
-------------------------[VMWare Detection]-------------------------
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\
Logical Unit Id 0  [ GOOD ]
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 1\Scsi Bus 0\Target Id 0\
Logical Unit Id 0  [ GOOD ]
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 2\Scsi Bus 0\Target Id 0\
Logical Unit Id 0  [ GOOD ]
[*] Checking reg key SYSTEM\ControlSet001\Control\SystemInformation
                   [ GOOD ]
[*] Checking reg key SYSTEM\ControlSet001\Control\SystemInformation
                   [ GOOD ]
[*] Checking reg key SOFTWARE\VMware, Inc.\VMware Tools
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmnet.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmmouse.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmusb.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vm3dmp.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmci.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmhgfs.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmmemctl.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmx86.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmrawdsk.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmusbmouse.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmkdb.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmnetuserif.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmnetadapter.sys
                   [ GOOD ]
[*] Checking MAC starting with 00:05:69
                   [ GOOD ]
[*] Checking MAC starting with 00:0c:29
                   [ GOOD ]
[*] Checking MAC starting with 00:1C:14
                   [ GOOD ]
[*] Checking MAC starting with 00:50:56
                   [ GOOD ]
[*] Checking VMWare network adapter name
                   [ GOOD ]
[*] Checking device \\.\HGFS
                   [ GOOD ]
[*] Checking device \\.\vmci
                   [ GOOD ]
[*] Checking VMWare directory
                   [ GOOD ]
[*] Checking SMBIOS firmware
                   [ GOOD ]
[*] Checking ACPI tables
                   [ GOOD ]
```

```
-------------------------[Virtual PC Detection]-------------------------
[*] Checking Virtual PC processes VMSrvc.exe
                   [ GOOD ]
[*] Checking Virtual PC processes VMUSrvc.exe
                   [ GOOD ]
[*] Checking reg key SOFTWARE\Microsoft\Virtual Machine\Guest\Parameters
                   [ GOOD ]

-------------------------[QEMU Detection]-------------------------
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target I
Logical Unit Id 0  [ GOOD ]
[*] Checking reg key HARDWARE\Description\System
                   [ GOOD ]
[*] Checking qemu processes qemu-ga.exe
                   [ GOOD ]
[*] Checking SMBIOS firmware
                   [ GOOD ]
[*] Checking ACPI tables
                   [ GOOD ]

-------------------------[Xen Detection]-------------------------
[*] Checking Citrix Xen process xenservice.exe
                   [ GOOD ]
[*] Checking Mac Address start with 08:16:3E
                   [ GOOD ]

-------------------------[Wine Detection]-------------------------
[*] Checking Wine via dll exports
                   [ GOOD ]
[*] Checking reg key SOFTWARE\Wine
                   [ GOOD ]

-------------------------[Paralles Detection]-------------------------
[*] Checking Parallels processes: prl_cc.exe
                   [ GOOD ]
[*] Checking Parallels processes: prl_tools.exe
                   [ GOOD ]
[*] Checking Mac Address start with 08:1C:42
                   [ GOOD ]

-------------------------[Timing-attacks]-------------------------

[*] Delay value is set to 10 minutes ...
[*] Performing a sleep using NtDelayExecution ...
```

Figure 5.5: Al-Khaser detecting artefacts after mitigation

SEMS: Anti-Sandbox and Anti-Virtual Machine Tool

In order to evade analysis, modern malware authors supply anti-analysis techniques to the malwares. SEMS is a tool which, much like the other aforementioned tools, uses common malware techniques to detect if it is running in a monitored environment [74]. In our project, since we have already mitigated most of the artifacts, this tool is used to confirm and verify that those changes took effect. The following, in fig. 5.6, are the results of the tool, having only detected that the Cuckoo agent is running. However, it is not able to detect any sort of virtual machine environment otherwise.



```
C:\Users\Hardened\Downloads\sems-master\sems-master\Debug\sems.exe
Core number is 1.
No internet.
Cuckoo detected (Files)
Cuckoo detected (Function hook)!!
control
```

Figure 5.6: SEMS detecting artefacts after mitigation

Simulation of internet access is a concept that can help provide more interactive malware analysis [86]. It is less automatic, but allows to e.g. send an executable file to a malware expecting an executable file, and observe a potential change in behavior due to this [86]. This can be achieved through INetSim, but has been deemed out of scope for the project. This decision will be elaborated on in chapter 7.

5.5 Summary

The tools explained in the previous chapter were explained in technical detail in this chapter, with regards to their setup and how they co-exist and cooperate. The sandboxing environment as well as the machines needed to fulfill the scope and problem statement were too described in detail.

Chapter 6

System Testing and Malware Analysis

6.1 Introduction

This chapter will cover the testing part of the project. Having established the tools used as well as the topology for this project, the next step is to analyze the behavior of malware in this containerized environment. This phase of the project will be discussed in detail here, although technical details can be found in appendix B.

6.2 Finding and sorting malware

When deciding on which malware to use to make the testing part appropriately representative, we decided on 20 malware binaries from different malware families or groups. Ultimately, we opted for 20 different malware binaries, with a full list provided below. Four of them will be botnets, three will be trojan horses, and four will be a mixture of different malware families, with worms, cryptolockers and similar. Additionally, we set up a Dionaea machine, as previously mentioned, to capture malware binaries when attacked. Three of these have also been analyzed. The full list of binaries can be found in appendix C.

Botnet1	Conficker7
Botnet2	IRCBot5
Botnet3	Cutwail4
Botnet4	Zeus6
Trojan2	Tapaoux
Trojan4	Sality
Trojan5	Hupigon.exe
Various2	KRLocker/CrimClient
Various3	SkyWiper-A.Flame
Various8	Variant Kazi
Various10	Carberp
Dionaea1	WannaCry instance
Dionaea2	Coin miner worm
Dionaea3	WannaCry instance

The binaries are run once on each of the three respective machines, and their behavior is then analyzed by Cuckoo. This is then interpreted by us using a report extractor program to extract the most important details from the JSON report generated by Cuckoo. The report extractor has been inspired by one of our colleagues, who built the baseline of the extractor. We have since changed the code and developed the program's functionality, allowing it to extract more data, including information regarding YARA rules. The program's source code can be found in the link to the GitHub repository in appendix D.

The behavioral details are then compared. This is done by analyzing the amount of Cuckoo signatures matching for each run of a malware sample, and the amount of YARA signatures matched for the same run. The analysis will be in comparing whether there is a difference when malware is run on the hardened machine, the weak machine, or the blind machine, or whether it, in some cases, makes no difference. The analysis will also attempt to determine whether other nodes on the network, such as the Dionaea machines, will be influenced by the sample runs.

6.3 The hypothesis

The hypothesis that this analysis will attempt to investigate is as follows:

Malware that checks for whether it is in a virtualized environment should not execute its payload as equally if it registers itself as being part of a virtualized environment. Malware that aims to propagate on a local network should not

attempt to execute its payload if it receives information of it not being on a network, and its impact on said machine should therefore be minimal or non-existent.

In technical terms, the hypothesis states that malware checking for whether it is in a virtualized environment should trigger on the Hardened machine more functionalities than on the Weak machine, seeing as the Hardened machine has had certain artefacts and flags removed from its configuration. Thus, malware that is dependent on whether it is being run in a sandbox environment, should not run. On the other side, malware that does not check for sandbox artifacts should run equally on both the Hardened and the Weak machine. Malware that aims to propagate across the local network should not run on the blind machine, seeing as it has no access to any other devices on the network.

Moreover, it is deemed of interest to note whether any malware binaries do try to enumerate devices on the local network at all.

6.4 The results

This section will present the results, starting with the botnets [87], then the trojans and the malware in the 'various' category, and ending with several binaries caught by our Dionaea honeypot deployed, publicly facing the internet.

On tables where there are differences between the runs of the same malware sample across the three machines, the presence of a difference will be signified by either yellow (no difference) or green (difference present). If the table has neither of these two colors, it is because the runs yielded no difference on any of the three machines, regardless of the malware sample run.

Table 6.1: The Botnets

Chosen Malware	Machine	Cuckoo Signatures Matched	YARA rules Matched
Botnet1	Hardened	18	21
Botnet1	Weak	14	21
Botnet1	Blind	2	21
Botnet2	Hardened	2	19
Botnet2	Weak	2	19
Botnet2	Blind	2	19
Botnet3	Hardened	3	9
Botnet3	Weak	3	9
Botnet3	Blind	3	9
Botnet4	Hardened	18	16
Botnet4	Weak	16	16
Botnet4	Blind	16	16

Table 6.2: Trojans and 'various'

Chosen malware	Machine	Cuckoo Signatures Matched	YARA Rules Matched
Trojan2	Hardened	23	22
Trojan2	Weak	23	22
Trojan2	Blind	23	22
Trojan4	Hardened	1	8
Trojan4	Weak	1	8
Trojan4	Blind	1	8
Various2	Hardened	2	19
Various2	Weak	2	19
Various2	Blind	2	19
Various3	Hardened	1	25
Various3	Weak	1	25
Various3	Blind	1	25
Various8	Hardened	14	25
Various8	Weak	14	25
Various8	Blind	14	25
Various10	Hardened	18	17
Various10	Weak	18	17
Various10	Blind	18	17

Table 6.3: Malware caught by our Dionaea honeypot

Chosen malware	Machine	Cuckoo Signatures Matched	YARA Rules Matched
Dionaea1	Hardened	3	0
Dionaea1	Weak	3	0
Dionaea1	Blind	3	0
Dionaea2	Hardened	4	18
Dionaea2	Weak	4	18
Dionaea2	Blind	4	18
Dionaea3	Hardened	3	0
Dionaea3	Weak	3	0
Dionaea3	Blind	3	0

6.5 Analyzing the results

For the malware samples at hand, we expected to see results indicative of the assumption that a machine with removed artefacts would trigger more aspects of a malicious binary file. However, with most of the samples yielding similar results across the three machines, this assumption seems to not hold ground, at least for these samples and this sample size.

All the samples yielded the same YARA rules triggered, independent of the machine the sample was run on. If a binary matched 21 YARA rules on the Hardened machine, it would also match 21 on both the Weak and the Blind machine. This also was the case for botnets, which, to us, was quite surprising.

For the first botnet sample, namely Conficker7, a difference was noted in the Cuckoo signatures matched by the sample runs. The Hardened machine triggered the highest amount of Cuckoo signatures, 18, which is what we expected. The Weak machine triggered 14, while the Blind machine only triggered 2 signatures. The signatures that were triggered by the Hardened one compared to the Weak one were the following 4:

- Attempts to modify Explorer settings to prevent file extensions from being displayed
- Attempts to modify Explorer settings to prevent hidden files from being displayed
- Attempts to modify Explorer settings to hide desktop icons

- Executed a process and injected code into it, probably while unpacking

Zeus6 was also one of the few samples that yielded different results on the Hardened machine compared to its counterparts. Here, the signatures "Creates hidden or system file" and "Checks the presence of IDE (Integrated Drive Electronics) drives in the registry, possibly for anti-virtualization" were triggered, indicating that the hardening of the machine artefacts indeed did make a difference, as expected in the hypothesis, seeing as a system file was created alongside the check for IDE drives.

The remaining botnet samples showed no differences in neither Cuckoo signatures or YARA rules matched across the different machines for their respective samples.

For the Trojans and Various categories, the latter was the case across all sample runs conducted by us.

This was also the case for all the malware samples caught by our Dionaea honeypot. We suspect that this is due to the samples potentially being fairly new, seeing as the honeypot was set up in early May, which would mean less widespread binaries, both in terms of YARA rules and Cuckoo signatures. This assumption is backed up by 2 of the 3 samples yielding zero YARA rule matches and only three signatures, which were the same regardless of the machine they were run on.

A difference also worth exploring is the difference, or lack thereof, in network traffic dump files. The PCAP files have been studied closely by us, both by monitoring the malware runs manually through WireShark and by observing the PCAP files reported by Cuckoo. These were a 1-to-1 match, and there is no notable difference between the two PCAP files in terms of monitoring of network traffic.

The Dionaea machines had a noticeable difference. The local machine did not trap any malware binaries, despite WannaCry being a malware targeting the SMB protocol, as previously mentioned. However, the Dionaea deployed on the cloud captured three malware binaries, all of which have been tested in the internal topology. These, too, did not yield any results in the locally deployed instance of the honeypot. This was confirmed by us, seeing as the three malware in the very end of the table shown in section 6.4 all triggered protocols in Dionaea on the public machine. These were tested on all three Windows machines and nothing was captured on the local Dionaea instance in VLAN 10. We can, moreover, confirm that no malware crossed VLANs over to the Dionaea instance on VLAN 20, either.

Two of the three binaries caught by Dionaea have through analysis proven to be instances of the WannaCry malware, targeting the SMB protocol amongst other things, as previously mentioned. The third binary is a coin miner, which is a type of malware that uses resources for financial purposes on the adversaries' end. Interestingly enough, neither of the three binaries made any contact with the outside world through the public network. This has been confirmed both through the analysis conducted on the report and PCAP file provided by Cuckoo on the cloud instance, as well as when the three files were submitted to VirusTotal, which yielded identical results to the ones we found through its cloud instance of Dionaea.

We believe that the lack of capturing on the local machine could be due to either malware spreading through crawlers and only then targeting Dionaea, seeing as the setups were identical on both the local and the cloud-hosted instance of Dionaea. Another theory we have is that the malware, due to not enumerating the local network, as found out by us through research conducted on the network traffic files, does not see the Dionaea machine. This could be due to these malware binaries not prioritizing local network spreading, but rather encrypting machine files, in the case of WannaCry, or simply spreading on the public Internet rather than locally, in the case of the coin miner worm. We believe that a larger malware sample size could provide a more conclusive result, but the group deems this theory satisfactory and well within the scope of the project.

We believe that the lack of difference in the signatures and rules triggered across the different machines can be due to two factors. One factor being that malware in recent times has been focusing more on anti-debugging rather than anti-virtual-machine techniques, as previously mentioned in chapter 2. This theory can be backed up by the reminder of the Cuckoo agent being present on all three machines, regardless of artifacts, and Cuckoo and its sub-processes could potentially trigger flags on a low level, which would then be caught by the anti-debugging aspects of malware. Regardless of whether other artefacts have been removed, Cuckoo and its sub-processes would prevail, seeing as these are independent of the machine itself, but relate to the agent on the client/guest machine.

Moreover, as mentioned in chapter 5, Cuckoo does not natively support memory dump usage with regards to what it considers physical machines. This could, from our understanding, lead to Cuckoo signatures that otherwise would have been triggered due to work in memory, not be triggered at all, and thus swings the findings. Alternatives to Cuckoo, which for future work could help alleviate this issue, will be mentioned in the next chapter, but using Cuckoo as part of the setup has potentially influenced the results.

The findings are by us deemed to be representative and conclusive within the scope of the project, given the aforementioned theories and ideas. Moreover, the hypothesis is considered by us to be well-founded and backed by the findings above. In a more elaborate manner, we will break down the hypothesis' soundness in the following section.

6.6 Confirming the hypothesis

The hypothesis shall, to ease the explanation and assessment, be split into two parts:

A) Malware that checks for whether it is in a virtualized environment should not execute its payload as equally if it registers itself as being part of a virtualized environment.

B) Malware that aims to propagate on a local network should not attempt to execute its payload if it receives information of it not being on a network, and its impact on said machine should therefore be minimal or non-existent

Part A is considered fulfilled. As evident by the highlighted botnet samples and backed up by the statements in the previous section, a notable difference can be seen in whether the malware binary registers itself as part of a virtualized environment, and alters its behavior based on this.

Part B is considered fulfilled. In the Conficker7 sample, the signatures vary from 18 on the Hardened machine, 16 on the Weak to 2 on the Blind machine, with the latter having no network access and no internet access. The lack of network access would be part of what hindered the execution of the botnet sample. A similar trend can be seen in Zeus6, although not as evident. Once again, the lack of memory dump usage by Cuckoo for physical machines could potentially have changed the outcome of these reports, results, and thus the findings. However, given the circumstances, this part of the hypothesis too is considered fulfilled.

Thus, the hypothesis is, by us, considered confirmed, based on the aforementioned findings and presented theories and results.

6.7 Summary

This chapter covered the testing aspect of the project. The malware samples and their results were presented visually, as well as respectively analyzed in deep detail. The hypothesis was then reiterated and split into two parts, both of which we consider fulfilled given the circumstances and the scope and vision set forth for the project and thesis.

Chapter 7

Project outlook

The project, being based on the prospect of a long thesis, has been a very challenging one. The extended time frame has allowed us to experiment with different types of potential solutions, before settling for the one deemed most suitable for the project's goals and scope.

The following challenges will be addressed:

- Getting FOG to work, lack of documentation
- Integration of FOG and Cuckoo into the network topology
- Certain things being more time consuming than first expected due to the project's uniqueness
- Inconsistencies in the VM's behavior
- Cuckoo not allowing memory dumps for physical machines
- Re-imaging being a very time-consuming process
- Cuckoo's inconsistency in which signatures it caught in certain runs
- Acquiring the binary files for Al-Khaser

7.1 Getting FOG to work, integrating it and Cuckoo into the network topology

One of the challenges that presented itself during the course of the project, and one that had quite grave effects with regards to the deadline, was making FOG function in our desired topology.

FOG is a fairly new project, with its first official release on GitHub dating back to December of 2016 [88]. What we believe this is partly due to, is the lack of documentation, especially when it comes to integrating it with other tools, such as Cuckoo. The documentation presented by the people behind FOG focuses on FOG as a concept and how it can be installed [65].

This proved to be helpful in setting up FOG on its own, although its integration with EVE-NG and Cuckoo proved to be a challenge that took much longer than initially expected. Cuckoo's documentation merely mentions FOG as an option for physical machines, but does not elaborate on this. With the Cuckoo community only having tested a certain version of FOG with a certain version of Cuckoo that has been confirmed as working, this proved to be a challenge due to certain features and outdated dependencies for both of FOG and Cuckoo.

Moreover, once the issues with dependencies were resolved, we had to find a way to ensure that the Cuckoo machine, which now also housed FOG, could properly communicate with the other nodes on the EVE-NG-managed network. This step also took longer than expected, seeing as EVE-NG's Community edition, despite it offering a lot, still is quite limited in some areas, and harder to set-up than if we had acquired the Professional edition. In hindsight, this could have helped avoid certain bottlenecks with regards to the automation aspect of the project when conducting malware sample runs. We would also definitely advise entities making use of this project and setup in the future to utilize the Premium version of EVE-NG.

With regards to EVE-NG, we, as previously mentioned, partly chose it as the network emulation platform due to previous positive experience with it. This proved to be a good choice, as it meant we needed to set aside less time to familiarize itself with the inner workings of EVE-NG.

7.2 Certain things being more time consuming than initially expected

However, due to the project's unique combination of EVE-NG, Cuckoo running what it believes to be physical machines, and FOG, challenges prevailed. For instance, the initial idea was to simply have one Windows machine in the topology. This proved to be rather easy, seeing as the machine should not have any limitations with regards to its communication with the rest of the network.

Later in the process, however, the project's supervision team suggested the introduction of multiple machines with changes made to each, as to distinguish them in the topology and potentially present differing results for each of the malware samples. This in turn provided challenges with regards to limiting some of the access the devices should have, as well as introducing VLAN's and ensuring they would work properly. We had to familiarize ourselves with VLAN access lists and other VLAN concepts, which proved to be quite time consuming, seeing as the setup for such aspects of a network differ from device to device and OS version to OS version.

Figuring out which gateway to use for the Blind machine also proved to be a challenge, as the two options, namely the FOG machine and the layer 3 switch, both would work, but in different ways, before we ultimately opted for the more *logical* choice: the host machine, to avoid any potential contact with the switch.

7.3 Inconsistencies in the VM's behavior and matched Cuckoo signatures

Once the machines were set up and confirmed to work in the topology as we expected, another problem arose: inconsistencies in the machines' behavior. On certain runs, the Python agent for Cuckoo would crash on the client, prompting a restart of the client file, or even Python as a whole on said client machine.

This would happen on what seems to be random occasions, with no apparent explanation behind it, seeing as no difference was made by us with regards to how the machines were booted up or how programs were loaded. We believe part of this could be caused by Cuckoo's communication with physical machines, with that part of Cuckoo still being in its infancy as evident by the lack of documentation on Cuckoo's end [83].

Contrary to Cuckoo's work with VirtualBox, which the team behind Cuckoo suggests as the primary manager for virtual machines that work with Cuckoo, memory dumps through Volatility are not possible using Cuckoo with what it believes is physical machines. This is something we believes has had an impact on the inconsistencies also shown in some of the malware reports generated by Cuckoo for certain malware binaries.

The reason why this is deemed an inconsistency rather than an error on our end is that a binary file could have 2 matched Cuckoo signatures in one run on the Hardened machine, then 17 on the immediately following run on the same machine with no settings altered.

7.4 Cuckoo not allowing memory dumps for physical machines

We believe this is due to Cuckoo's lack of memory dump support for physical machines, as well as it being a non-'native' way of working with Cuckoo, seeing as it uses FOG to manage the machine's state and images. We believe we could have used VirtualBox had the project objectives been different, although this would not have allowed for EVE-NG to be as good of a management tool as it has been for us, despite its shortcomings in the Community edition. Moreover, we believe that with scalability being one of our main objectives, the setup with FOG, Cuckoo and EVE-NG was the right choice.

7.5 Re-imaging being very time consuming

Another issue we encountered throughout the course of the project was the re-imaging process. This became a problem once all the tools had been set up properly and the intercommunication for the machines was set up accordingly. When Cuckoo runs a malware binary on a machine, the report is not fully generated before the machine has been re-imaged. This takes, according to our timing, around 10-20 minutes per machine, depending on the machine and the sample run, again due to inconsistencies in FOG and the virtual machines themselves, for reasons seemingly outside our control, and outside the scope of this project, seeing as this is meant to be a proof of concept.

This, unfortunately, makes the automation aspect of the project setup, thus far, less good than what it could be. However, we believe that, with the right amount of time dedicated into researching potential causes or alternatives to this, this part too could be time efficient for when machines are running samples.

The workaround we have used, for now, when wanting to run a malware samples across the three machines, is initiating a sample on machine A, waiting for the re-imaging to start, then running it on machine B, and so forth. This has proven to be the most time efficient way to run a malware sample across all three machines in this current setup, and we have deemed this a satisfactory solution.

7.6 Acquiring the binary files for Al-Khaser

The last issue that will be addressed in this section, was the gathering of the binary files for Al-Khaser, a tool used to check how hardened the Hardened machine was, as mentioned in chapter 5. Due to Google's Safe Browsing heuristics being triggered by the binary files hosted on the GitHub repository [73], the files had been removed some time ago by the author known as LordNoteworthy.

Acquiring the binary files needed to run the software took roughly two weeks, as we initially contacted LordNoteworthy on the e-mail address they had provided online, before settling for looking through online archives on the WayBackMachine and forums mentioning Al-Khaser. Most only mentioned the tool or the tool's objectives, with the very few links to the binaries being links back to the GitHub repository. After intensive searching, we found a re-upload by a forum user on the GitHub repository, uploaded to MEGA, a file-sharing platform, and these binaries were used for that part of the project.

7.7 The lack of simulated network traffic via INetSim

Simulating network traffic in general is something we have deemed a relevant topic to this project. Instead of simulating network traffic to make malware believe it is on a real corporate network, or using INetSim to communicate with the malware as a spoofed command-and-control center, nodes were added to the network to make malware believe it was in a production environment.

If we wanted to add a node for a simulated DNS server, such as what could be achieved using INetSim, we believe it would have been best to do so on a machine different from the one hosting Cuckoo and FOG. In our topology, this could have been the machine hosting Dionaea.

However, this would mean that traffic would go towards the Dionaea machine deliberately, and this would defeat the purpose of Dionaea's inclusion in this project: to see whether malware targeting its protocols naturally would be drawn to it or not. Dionaea has been included to see whether malware targeting a protocol, e.g. SMB, would go towards a machine on the local network serving a service on the SMB protocol. Asking the malware to contact that same machine in order to use it as a DNS server would mean deliberately showing it that machine, rather than it probing the network on its own, which in turn would give a false indicator of the malware's behavior, thus making the project's results less representative.

The reason why INetSim, or a similar service, would have to be hosted on the same machine as Dionaea in our topology is due to resource restrictions, seeing as we could not have resources provided by the university on a physical machine, which was the initial idea when forming this project with the supervising team.

Chapter 8

Conclusion

The project has overall presented many challenges and setbacks due to its uniqueness. The project's main goal and problem statement was the following:

How can a platform for sandboxing be made resilient enough to allow studying malware's behavior, providing evidence for further fortification of systems and help future studies on the topic?

The problem statement posed three questions, relating to common malware techniques, anti-sandbox techniques and how behavioral analysis on malware can be conducted without hindering malware executing their full payload. Following the project, gathering of malware and analysis thereof, the questions as well as the problem statement are considered fulfilled.

For the first question about the techniques malware makes use of to combat sandboxing-based defenses, the answer is checking system artefacts and flags, whether on the system level or in the memory level. Since sandboxing and debugging are actions that trigger certain flags or leave behind certain artefacts that expose virtualization to crawlers, these are some of the ways in which malware analyzes its environment.

Moreover, checks such as the amount of RAM a machine has or the hard disk size, as well as vendor information, are used to see whether they are 'realistic' when compared to what physical machines usually would have.

For instance, a machine with less than 60 GB of hard drive space running Windows 7 could for certain malware be indicative of a virtualized environment, as we saw in one of the malware sample runs. This issue can, bluntly put, be circumvented by removing certain artefacts left behind by virtualization software, and making machines look as realistic as possible in terms of hardware specifications.

However, working in memory and removing certain flags could also prove to be important, depending on the amount of hardening desired for a certain machine. This has, however, been deemed out of scope for this project, although it had been considered an option throughout the project.

For the second question, the answer has been achieved through the usage of Wire-Shark and the TCPDump module on Cuckoo. Cuckoo itself has helped make this project scalable when paired with Eve-NG and QEMU machines, and has made the project easily manageable on scale, since adding and removing nodes to different VLANs is easily achievable. As for analyzing the traffic, TCPDump allows Cuckoo to provide us with network traffic analysis as part of the report files generated for that specific sample run on each of the machines used for analysis. Eve-NG allows for live capturing of network traffic through two clicks on the layer-3 switch, and this means that both traffic captures can be compared, if need be. However, Cuckoo automatically creates a network dump file in a PCAP format as part of the sample run and includes this in the report, making this part of the project very scalable. This question is therefore deemed fulfilled.

This leads into the third and final question about how behavioral analysis on malware can be conducted while not hindering the malware's full execution. This question, too, is deemed satisfied. Part of the answer lies in the previous paragraphs of this chapter. The other part lies in ensuring that the sandboxing environment is one that can conduct analysis on even the most devastating of malware, such as Petya or WannaCry, which encrypt entire hard drive partitions.

This has been achieved through Cuckoo. The other part of the question, can, as previously mentioned, be answered through hardening of machines by removing certain artefacts. Analyzing memory dumps, as mentioned in chapter 7 could also be done in future work relating to this project, as it would give more insight into how the malware operates on a deeper level.

With those questions addressed, one prevails: **the problem statement**.

The problem statement is considered fulfilled. The fact that three machines with different environments surrounding them, and different setups in terms of artefacts, have been setup for this project, has helped the project's cause greatly.

Removing certain system artefacts has in some cases proven to drastically alter the behavior of certain malware, especially network-reliant ones, where it in others has proven to not make a difference.

This, as previously mentioned, is suspected to be due to anti-debugging techniques being more common than anti-VM techniques, meaning that some of the removed artefacts would not impact the debugging flags' status in memory, as us operating on the memory level was deemed out of scope, see section 1.2.

Despite this, we believe it has made a platform resilient enough to allow studying of malware's behavior and providing evidence that can benefit future studies as well as further fortification of systems.

8.1 Future directions

As far as future directions of the project goes, we believe the project can be developed on using other tools as well. Docker is an example of a virtualization tool that is meant to be very scalable [89], but using this with Eve-NG would require the Professional version of the software. If Docker is an opportunity wanting to be explored by researchers working on this project in the future, another option could be to drop using Eve-NG altogether, and focusing efforts on GNS3 or other network emulator management tools with free support for other virtualization software.

Acquiring the Professional version, which has more advantages such as native Docker container support, custom image templates and the ability to configure startup flags for multiple nodes at once, could be an interesting choice, as some of the features, especially bulk configuration, were missed by us during the course of this project.

Future work based on this project could include using more Dionaea modules to see which protocols malware would target on the local network.

In terms of memory analysis, attempting nested virtualization, thus allowing for usage of Volatility, seeing as the test machines would not be viewed as physical machines by Cuckoo, could also be an option worth exploring, if the hosting computer has more resources allocated than was the case for us.

This could then be used to analyze whether the reports generated by Cuckoo differ from a setup similar to ours, if Volatility is used by Cuckoo, despite this being through nested virtualization.

Finally, INetSim is another aspect that will be touched upon, seeing as it was discarded, see chapter 7. INetSim could be interesting for projects with a more interactive aspect to their analysis, and we believe it would definitely add a new layer to seeing whether a small amount of anti-anti-detection techniques deployed, paired with interaction with the malware, would be enough to make it execute as if it was in a *real* environment, compared to using a lot of anti-anti-detection techniques and no manual interaction with the malware.

We feel this project, either way, can act as a baseline for interesting projects exploring malware and their behavior in the field of cyber security.

We deem all initially presented questions, the hypothesis, as well as the problem statement, addressed and fulfilled to a satisfactory extent.

8.2 Contributions

This project, moreover, contributes with the following to the field of cyber security:

- A network-emulator based sandboxing platform, which, to the group's knowledge is the first one in the industry at this level
- A full thesis addressing any and all issues that arose throughout the research, rather than a privately conducted on-site experiment
- Issues addressed in a manner that hopefully will benefit future studies on the subject
- An open-source code foundation for the parts of this project that readily can be used by others, such as a report extractor script, Cisco's switch configuration and bash scripts for ease of installation of some of the tools used.

Bibliography

[1] Internet World Stats. *Internet Growth Statistics 1995 to 2021 - the Global Village Online*. 2021. URL: https://www.internetworldstats.com/emarketing.htm (visited on 10/24/2021).

[2] Internet World Stats. *World Internet Users Statistics and 2021 World Population Stats*. 2021. URL: https://www.internetworldstats.com/stats.htm (visited on 10/24/2021).

[3] Chris Greamo and Anup Ghosh. "Sandboxing and Virtualization: Modern Tools for Combating Malware". In: *IEEE Security Privacy* 9.2 (2011), pp. 79–82. DOI: 10.1109/MSP.2011.36.

[4] AVTest. *Malware Statistics & Trends Report*. 2021. URL: https://www.av-test.org/en/statistics/malware/ (visited on 11/10/2021).

[5] Shuman Ghosemajumder Harvard Business Review. *You Can't Secure 100% of Your Data 100% of the Time*. 2017. URL: https://hbr.org/2017/12/you-cant-secure-100-of-your-data-100-of-the-time (visited on 02/23/2022).

[6] CrowdStrike. *11 Types of Malware*. 2021. URL: https://www.crowdstrike.com/cybersecurity-101/malware/types-of-malware/ (visited on 11/10/2021).

[7] Marios Anagnostopoulos and John André Seem. "Another Step in the Ladder of DNS-Based Covert Channels: Hiding Ill-Disposed Information in DNSKEY RRs". In: *Information* 10.9 (2019), p. 284.

[8] Ashok Sharma Dev.to. *What does SolarWinds Hack tell us about Backdoor Attacks?* 2021. URL: https://dev.to/ashok83/what-does-solarwinds-hack-tell-us-about-backdoor-attacks-41jn (visited on 02/23/2022).

[9] Twosense.ai. *Russian Hackers target US Treasury, NTIA and more in Huge Cyber Espionage Campaign against the US*. 2020. URL: https://www.twosense.ai/blog/2020/12/14/russian-hackers-target-us-treasury-ntia-and-more-in-huge-cyber-espionage-campaign-against-the-us (visited on 02/23/2022).

[10] Lockheed Martin Corp. *The Cyber kill chain, Lockheed Martin*. 2015. URL: https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/Seven_Ways_to_Apply_the_Cyber_Kill_Chain_with_a_Threat_Intelligence_Platform.pdf (visited on 02/23/2022).

[11] KasperSky. *Ransomware WannaCry: All you need to know*. 2021. URL: https://www.kaspersky.com/resource-center/threats/ransomware-wannacry (visited on 11/15/2021).

[12] Carly Burdova for Avast. *EternalBlue Exploit | MS17-010 Explained | Avast*. 2020. URL: https://www.avast.com/c-eternalblue (visited on 05/10/2022).

[13] BBC News. *Cyber-attack: US and UK blame North Korea for WannaCry*. 2017. URL: https://www.bbc.com/news/world-us-canada-42407488 (visited on 11/15/2021).

[14] The United States of America's Department of Justice. *North Korean Regime-Backed Programmer Charged With Conspiracy to Conduct Multiple Cyber Attacks and Intrusions*. 2018. URL: https://www.justice.gov/opa/pr/north-korean-regime-backed-programmer-charged-conspiracy-conduct-multiple-cyber-attacks-and (visited on 02/23/2022).

[15] Liku Zelleke for Comparitech. *7 Best Static Code Analysis Tools*. 2021. URL: https://www.comparitech.com/net-admin/best-static-code-analysis-tools/ (visited on 03/07/2022).

[16] Radu S Pirscoveanu et al. "Analysis of malware behavior: Type classification using machine learning". In: *2015 International conference on cyber situational awareness, data analytics and assessment (CyberSA)*. IEEE. 2015, pp. 1–7.

[17] *Practical Malware Analysis: The Hands-on Guide to Dissecting Malicious Software*. No Starch Press, 2012.

[18] Imperva. *What is a Honeypot*. 2019. URL: https://www.imperva.com/learn/application-security/honeypot-honeynet/ (visited on 03/13/2022).

[19] Amir Afianian et al. "Malware Dynamic Analysis Evasion Techniques: A Survey". In: *ACM Computing Surveys*. Vol. 52. 2019, Article 126.

[20] NC State University. *Ethics in computing: Reverse engineering*. 2019. URL: https://docs.vmware.com/en/VMware-Workstation-Pro/16.0/com.vmware.ws.using.doc/GUID-89311E3D-CCA9-4ECC-AF5C-C52BE6A89A95.html#GUID-89311E3D-CCA9-4ECC-AF5C-C52BE6A89A95 (visited on 03/07/2022).

[21] Fortinet. *Why Do You Need Sandboxing For Protection?* 2014. URL: https://www.fortinet.com/content/dam/fortinet/assets/solution-guides/Why-Do-You-Need-Sandboxing.pdf (visited on 03/15/2022).

[22] Wikipedia. *Hyperviseur - Hypervisor - Wikipedia*. 2022. URL: https://en.wikipedia.org/wiki/Hypervisor#/media/File:Hyperviseur.svg (visited on 03/15/2022).

[23] Colin Steele Brien Posey Anil Desai. *Type 2 hypervisor (hosted hypervisor)*. 2021. URL: https://searchservervirtualization.techtarget.com/definition/hosted-hypervisor-Type-2-hypervisor (visited on 03/02/2022).

[24] VMWare. *Downloading and installing VMware Workstation (2057907)*. 2021. URL: https://kb.vmware.com/s/article/2057907/ (visited on 03/01/2022).

[25] Oracle. *Oracle VM VirtualBox*. 2022. URL: https://www.virtualbox.org/ (visited on 03/15/2022).

[26] Citrix. *Type 2 hypervisor (hosted hypervisor)*. 2021. URL: https://docs.citrix.com/en-us/citrix-hypervisor/graphics.html (visited on 03/21/2022).

[27] Greg Shields. *Q. Is VMware Workstation a type 1 or type 2 hypervisor?* 2010. URL: https://www.itprotoday.com/server-virtualization/q-vmware-workstation-type-1-or-type-2-hypervisor (visited on 03/21/2022).

[28] OpenSystems. *Cloud Sandbox | Cloud Based Sandbox | Open Systems*. 2022. URL: https://www.open-systems.com/sase/cloud-sandbox/ (visited on 03/01/2022).

[29] SHADE. *Download alternative for Windows sandbox software*. 2021. URL: https://www.shadesandbox.com (visited on 03/01/2022).

[30] SHADE. *Enterprise*. 2021. URL: https://www.shadesandbox.com/enterprise (visited on 03/01/2022).

[31] Cuckoo Sandbox. *Cuckoo Sandbox - Automated malware analysis*. 2019. URL: https://https://cuckoosandbox.org (visited on 03/01/2021).

[32] Cuckoo Sandbox. *Cuckoo Sandbox*. 2022. URL: https://cuckoo.cert.ee (visited on 03/02/2022).

[33] Neil Fox. *Varonis - Cuckoo Sandbox Overview*. 2021. URL: https://www.varonis.com/blog/cuckoo-sandbox/ (visited on 03/03/2022).

[34] Cuckoo. *FAQ - Cuckoo Sandbox v2.0.7 book*. 2021. URL: https://cuckoo.readthedocs.io/en/latest/faq/#can-i-analyze-urls-with-cuckoo/ (visited on 03/03/2022).

[35] Cuckoo. *Requirements - Cuckoo*. 2021. URL: https://cuckoo.sh/docs/installation/host/requirements.html#virtualization-software (visited on 03/02/2022).

[36] YARA. *Cuckoo module - yara 4.1.0 documentation*. 2021. URL: https://yara.readthedocs.io/en/stable/modules/cuckoo.html/ (visited on 03/02/2022).

[37] Cuckoo. *Configuration - Cuckoo*. 2021. URL: https://cuckoo.readthedocs.io/en/latest/installation/host/configuration/ (visited on 03/02/2022).

[38] Kevin O'Reilly. *GitHub - kevoreilly/CAPEv2: Malware Configuration and Payload Extraction*. 2021. URL: https://github.com/kevoreilly/CAPEv2/ (visited on 03/03/2022).

[39] Kevin O'Reilly. *CAPE Sandbox*. 2021. URL: https://capesandbox.com (visited on 03/03/2022).

[40] Tyler Shields. *Anti-debugging — A developer's view*. Veracode Inc., USA, 2010.

[41] Katsunari Yoshioka et al. "Your sandbox is blinded: Impact of decoy injection to public malware analysis systems". In: *Journal of Information Processing* 19 (2011).

[42] Jeremy Blackthorne et al. "AVLeak: Fingerprinting antivirus emulators through black-box testing". In: *Proceedings of the 10th USENIX conference on Offensive Technologies, USENIX Association* (2016).

[43] Ping Chen et al. "Advanced or Not? A Comparative Study of the Use of Anti-debugging and Anti-VM Techniques in Generic and Targeted Malware". In: *SEC 2016: ICT Systems Security and Privacy Protection* (2016).

[44] Tarik Muhovic. *Behavioural Analysis of Malware Using Custom Sandbox Environments*. Aalborg University, 2020.

[45] Xu Chen et al. "Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware". In: *IEEE International Conference on Dependable Systems and Networks with FTCS and DCC* (2008).

[46] JoeSandbox. *Automated Malware Analysis Report for shcndhss.exe*. 2018. URL: https://www.joesandbox.com/analysis/50204/0/html (visited on 03/11/2022).

[47] Joshua Tully. *An Anti-Reverse Engineering Guide*. 2008. URL: https://www.codeproject.com/Articles/30815/An-Anti-Reverse-Engineering-Guide (visited on 03/11/2022).

[48] Rodrigo Rubira Branco, Gabriel Negreira Barbosa, and Pedro Drimel Neto. "Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies". In: *Black Hat* (2012).

[49] Peter Ferrie. *The "Ultimate" Anti-Debugging Reference*. 2011. URL: https://anti-reversing.com/Downloads/Anti-Reversing/The_Ultimate_Anti-Reversing_Reference.pdf (visited on 03/11/2022).

[50] McAfee. *The W9x.CIH virus — via [19]*. 2000. URL: https://home.mcafee.com/virusinfo/virusprofile.aspx?key=10300 (visited on 03/11/2022).

[51] McAfee. *The W32.Mydoom.M@mm virus*. 2007. URL: https://www.symantec.com/security-center/writeup/2004-072615-3527-99 (visited on 03/11/2022).

[52] Microsoft. *PEB (winternl.h*. 2021. URL: https://docs.microsoft.com/en-us/windows/win32/api/winternl/ns-winternl-peb (visited on 03/11/2022).

[53] Anish for MalwareCrypt. *Reptile Malware - Behavioral Analysis*. 2012. URL: http://malwarecrypt.blogspot.com/2012/01/reptile-malware-behavioral-analysis.html/ (visited on 03/13/2022).

[54] VMWare. *Choosing a network adapter for your virtual machine (1001805)*. 2021. URL: https://kb.vmware.com/s/article/1001805/ (visited on 03/13/2022).

[55] Microsoft. *Worm:Win32/Rbot.ST*. 2017. URL: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Worm:Win32/Rbot.ST (visited on 03/13/2022).

[56] Sophos. *W32/Agobot-OT*. 2015. URL: https://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/W32%5C%20Agobot-OT/detailed-analysis.aspx (visited on 03/13/2022).

[57] Christopher Kruegel. *Evasive malware exposed and deconstructed*. RSA Conference, 2015.

[58] Anton Cherepanov eSet. *WIN32/INDUSTROYER - A new threat for industrial control systems*. 2017. URL: https://www.welivesecurity.com/wp-content/uploads/2017/06/Win32_Industroyer.pdf (visited on 03/16/2022).

[59] Paul Roberts. *Mydoom Sets Speed Records*. 2004. URL: https://www.pcworld.com/article/114461/article.html (visited on 03/16/2022).

[60] Austin for groovyTips. *Windows 7 Detailed Version Comparison [groovyTips]*. 2018. URL: https://www.groovypost.com/howto/geek-stuff/windows-7-detailed-comparison/ (visited on 03/01/2022).

[61] EVE-NG. *EVE-NG*. 2021. URL: https://www.eve-ng.net (visited on 11/09/2021).

[62] ipwithease.com. *Introduction to Eve-NG, GNS3 and VIRL*. 2020. URL: https://ipwithease.com/gns3-vs-eve-ng-vs-virl/ (visited on 02/24/2021).

[63] EVE-NG. *Community Cookbook*. 2021. URL: https://www.eve-ng.net/index.php/documentation/community-cookbook/ (visited on 02/25/2022).

[64] QEMU. *QEMU*. 2022. URL: https://www.qemu.org/ (visited on 04/01/2022).

[65] FOGProject. *FOG Wiki*. 2020. URL: https://wiki.fogproject.org/wiki/index.php?title=Introduction#What_is_FOG (visited on 02/24/2021).

[66] Cuckoo Sandbox. *Introduction to Eve-NG, GNS3 and VIRL*. 2020. URL: https://cuckoo.readthedocs.io/en/latest/installation/guest_physical/saving/#fog (visited on 02/24/2021).

[67] QEMU. *QEMU image utility*. 2020. URL: https://qemu.readthedocs.io/en/latest/tools/qemu-img.htm (visited on 02/24/2021).

[68] WinSCP corp. *Intro to WinSCP*. 2020. URL: https://winscp.net/eng/docs/introduction (visited on 02/24/2021).

[69] Dionaea Honeypot. *dionaea honeypot documentation*. 2021. URL: https://dionaea.readthedocs.io/en/latest/installation.html (visited on 04/02/2022).

[70] Kroland.no. *Dionaea - Setting up a Honeypot environment (Part 2)*. 2021. URL: https://kroland.no/2019/10/14/dionaea-setting-up-a-honeypot-environment-part-2/ (visited on 04/02/2022).

[71] DigitalOcean. *DigitalOcean*. 2022. URL: https://www.digitalocean.com/ (visited on 11/16/2021).

[72] Alberto Ortega. *GitHub repository*. 2015. URL: https://github.com/a0rtega/pafish (visited on 03/14/2022).

[73] LordNoteWorthy. *Github Repo*. 2021. URL: https://github.com/LordNoteworthy/al-khaser (visited on 04/11/2022).

[74] AlicanAkyol AlicanAkyol. *Anti-Sandbox and Anti-Virtual Machine Tool*. 2021. URL: https://github.com/AlicanAkyol/sems (visited on 04/07/2022).

[75] VMWare. *Understanding Common Networking Configurations*. 2019. URL: https://docs.vmware.com/en/VMware-Workstation-Pro/16.0/com.vmware.ws.using.doc/GUID-D9B0A52D-38A2-45D7-A9EB-987ACE77F93C.html (visited on 03/01/2022).

[76] VMWare. *Configuring Bridged Networking*. 2019. URL: https://docs.vmware.com/en/VMware-Workstation-Pro/16.0/com.vmware.ws.using.doc/GUID-BAFA66C3-81F0-4FCA-84C4-D9F7D258A60A.html#GUID-BAFA66C3-81F0-4FCA-84C4-D9F7D258A60A (visited on 03/01/2022).

[77] VMWare. *Configuring Network Address Translation*. 2022. URL: https://ethics.csc.ncsu.edu/intellectual/reverse/study.php (visited on 03/07/2022).

[78] VMWare. *Configuring Host-Only Networking*. 2019. URL: https://docs.vmware.com/en/VMware-Workstation-Pro/16.0/com.vmware.ws.using.doc/GUID-93BDF7F1-D2E4-42CE-80EA-4E305337D2FC.html#GUID-93BDF7F1-D2E4-42CE-80EA-4E305337D2FC (visited on 03/01/2022).

[79] EVE-NG. *Download*. 2021. URL: https://www.eve-ng.net/index.php/download/ (visited on 02/24/2022).

[80] WireShark. *Wireshark - Go Deep*. 2021. URL: https://www.wireshark.org/ (visited on 11/16/2021).

[81] Cuckoo. *Requirements - Cuckoo*. 2021. URL: https://cuckoo.sh/docs/installation/host/requirements.html (visited on 03/02/2022).

[82] YARA. *Yara-Rules/rules: Repository of yara rules*. 2022. URL: https://github.com/Yara-Rules/rules (visited on 12/02/2021).

[83] cuckoo sandbox. *Preparing the Guest (Physical Machine)*. 2020. URL: `https://cuckoo.readthedocs.io/en/latest/installation/guest_physical/` (visited on 03/14/2021).

[84] Cuckoo Sandbox. *Network Configuration — Cuckoo Sandbox v2.0.7 Book*. 2020. URL: `https://cuckoo.readthedocs.io/en/latest/installation/guest_physical/network/` (visited on 02/24/2021).

[85] Cisco Community. *How do I isolate one workstation on a LAN*. 2007. URL: `https://community.cisco.com/t5/switching/how-do-i-isolate-one-workstation-on-a-lan/td-p/892503` (visited on 03/07/2022).

[86] Erik Hjelmvik for NetreSec. *Installing a Fake Internet with INetSim and PolarProxy*. 2019. URL: `https://www.netresec.com/?page=Blog&month=2019-12&post=Installing-a-Fake-Internet-with-INetSim-and-PolarProxy` (visited on 05/02/2022).

[87] Marios Anagnostopoulos, Georgios Kambourakis, and Stefanos Gritzalis. "New facets of mobile botnet: architecture and evaluation". In: *International Journal of Information Security* 15.5 (2016), pp. 455–473.

[88] FOGProject. *Releases - FogProject/fogproject*. 2016. URL: `https://github.com/FOGProject/fogproject/releases` (visited on 04/02/2022).

[89] Turbo. *Turbo and Docker*. 2021. URL: `https://app.turbo.net/docs/about/turbo-and-docker#layering` (visited on 11/16/2021).

[90] EVE-NG site. *what is difference between native and htm management*. 2020. URL: `https://www.eve-ng.net/index.php/documentation/howtos-video/use-html5-and-native-console/` (visited on 03/02/2021).

[91] EVE-NG. *EVE-NG Supported images*. 2021. URL: `https://www.eve-ng.net/index.php/documentation/supported-images/` (visited on 03/03/2022).

[92] Qemu. *Qemu defininition*. 2021. URL: `https://wiki.qemu.org/Main_Page` (visited on 03/03/2022).

[93] iteasypass. *Dynamips / Dynagen Tutorial*. 2021. URL: `https://www.iteasypass.com/dynamips.htm` (visited on 03/03/2022).

[94] iteasypass. *Recommended IOL image versions:* 2021. URL: `https://www.eve-ng.net/index.php/documentation/howtos/howto-add-cisco-iol-ios-on-linux/` (visited on 03/03/2022).

[95] EVE-NG. *Ready to go images for EVE-NG*. 2021. URL: `https://www.eve-ng.net/index.php/documentation/howtos/howto-create-own-linux-host-image/` (visited on 03/07/2022).

[96] proliferoustech. *Setting up the successor to UnetLab: EVE-NG!* 2017. URL: `https://www.proliferoustech.com/blogs/20170411-setting-up-the-successor-to-unetlab-eve-ng/` (visited on 03/07/2022).

[97] EVE-NG. *Versions this guide is based on*. 2021. URL: https://www.eve-ng.net/index.php/documentation/howtos/howto-add-vm-ware-esxi/ (visited on 03/07/2022).

[98] EVE-NG. *This table shows correct foldername for QEMU images used under EVE*. 2021. URL: https://www.eve-ng.net/index.php/documentation/qemu-image-namings/ (visited on 03/03/2022).

[99] EVE-NG. *Download Links and info for EVE-NG*. 2021. URL: https://www.eve-ng.net/index.php/download/ (visited on 03/08/2022).

[100] Cisco. *Using the command line interface*. 2021. URL: https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3850/software/release/3se/consolidated_guide/b_consolidated_3850_3se_cg_chapter_01.html (visited on 03/10/2022).

[101] Pivitgolbal. *Overview of the DHCP Server*. 2021. URL: https://info.pivitglobal.com/resources/cisco-ios-dhcp-server-configuration (visited on 03/10/2022).

[102] Ciscopress. *Verifying and Troubleshooting DHCP Configuration*. 2010. URL: https://www.ciscopress.com/articles/article.asp?p=1574301&seqNum=6 (visited on 03/10/2022).

[103] Cisco Inc. *Configuring VLANs*. 2021. URL: https://www.cisco.com/c/en/us/td/docs/routers/ir910/software/release/1_0/configuration/guide/ir910scg/swvlan.pdf (visited on 03/11/2022).

[104] Cisco Inc. *Configuring InterVLANs Routing*. 2021. URL: https://www.cisco.com/c/en/us/support/docs/lan-switching/inter-vlan-routing/41860-howto-L3-intervlanrouting.html (visited on 03/11/2022).

[105] Utopianknight.com. *Cuckoo Installation on Ubuntu 20*. 2020. URL: https://utopianknight.com/malware/cuckoo-installation-on-ubuntu-20/ (visited on 03/14/2022).

[106] Cuckoo Sandbox. *Cuckoo Installation*. 2020. URL: https://cuckoo.readthedocs.io/en/latest/installation/host/installation/# (visited on 03/14/2022).

[107] cuckoo sandbox. *Cuckoo working directory*. 2020. URL: https://cuckoo.readthedocs.io/en/latest/installation/host/cwd/ (visited on 03/14/2021).

[108] Cuckoo Sandbox. *Code of the physical.py file on GitHub*. 2020. URL: https://github.com/cuckoosandbox/cuckoo/blob/master/cuckoo/machinery/physical.py (visited on 03/14/2021).

[109] FOG Project. *FOG 1.4.4 Officially Released*. 2017. URL: https://news.fogproject.org/fog-1-4-4-officially-released/ (visited on 03/14/2021).

[110] Cuckoo Sandbox. *Code of the physical.py file on GitHub*. 2020. URL: `https://github.com/cuckoosandbox/cuckoo/blob/master/cuckoo/machinery/physical.py` (visited on 03/14/2021).

[111] Cisco community. *DHCP option 67 on a 3750*. 2014. URL: `https://community.cisco.com/t5/switching/dhcp-option-67-on-a-3750/td-p/2486406` (visited on 10/24/2021).

[112] FOG Project. *Capture an image*. 2020. URL: `https://docs.fogproject.org/en/latest/getting_started/capture_an_image.html` (visited on 11/15/2021).

[113] open-suse. *Running Virtual Machines with qemu-kvm*. 2020. URL: `http://open-suse.ru/opensuse-doc/cha.qemu.running.html` (visited on 11/15/2021).

[114] Brian Linkletter. *Build a custom Linux Router image for UNetLab and EVE-NG network emulators*. 2017. URL: `https://www.brianlinkletter.com/2017/03/build-custom-linux-router-image-unetlab-eve-ng-network-emulators/` (visited on 11/15/2021).

[115] DinoTools. *Home of the dionaea honeypot*. 2021. URL: `https://github.com/DinoTools/dionaea` (visited on 03/02/2022).

Appendix A

Thesis contract

Project Title: Analyzing malware through Sandboxing
Starting: 1 September 2021
Deadline: medio/ultimo May 2022
ECTS: 50
If long master's thesis please indicate courses: Advanced Topics in Cyber Security – Privacy Engineering (elective).

Supervisors: Marios Anagnostopoulos and Jens Myrup Pedersen

Project description: This master thesis aims to delve into malware analysis through sandboxing. Specifically, we will investigate the available techniques for malware analysis that aim to create resilient infrastructures for sandboxing environments. Based on these findings, we will implement a platform for sandboxing, which will facilitate the study of the malware's behavior and the capturing of network traffic and other evidence. In turn, this evidence will contribute to the creation of a comprehensive data set for training and evaluation of ML techniques for the detection of malware.

Plan for thesis supervision and lab work: Weekly meetings with Marios have been agreed on. We, Omar and Adil, will work on virtually a daily basis on-site (in the study hall for Cyber Security & ICTE on 3rd floor at Frederikskaj) unless otherwise agreed on in private. Jens and other interested cyber security professionals working on a similar project using AAU's resources will from time to time take part in some of the meetings.

Approved by the supervisors and the Study Board's Head of Studies.

A.1 The workflow and communication

The group, consisting of Adil Khurshid and Omar Nabil Hawwash, hereinafter referred to as the project group, has agreed to meeting for 6-hour workdays every day at 10:00 AM local time in Copenhagen, and work on-site at Aalborg University, henceforth referred to as 'the university', in the Cyber Security group rooms facilitated by the university. On days with course classes, the group will meet briefly before the class starts, for roughly two hours.

Any obstacles that would mean no meetings on a certain day will be communicated beforehand amongst the group, in writing or verbally. Fridays are considered days off, and weekends are days on which either no work is done, or days considered for homework.

A thesis contract has been formed with supervisors Marios Anagnostopoulos and Jens Myrup Pedersen, with Peyman Pahlavani helping from time to time. The idea is that some of the documentation for the project work is converted to scientific articles to help further solidify the project and research put forth with regards to this thesis.

Meetings with the supervisors take place on a weekly basis unless otherwise agreed on. Meetings are, usually, done on-site, unless obstacles mean that they will be held online, in which case this is also agreed on in writing.

For project planning and resource sharing, a spreadsheet and a shared Google Drive as well as a Discord server are used. For planning of the project, a Google Spreadsheet and Trello are made use of, and the thesis source code shall be available on a GitHub repository shared among the project stakeholders (consisting of the project group and the project supervisors).

Appendix B

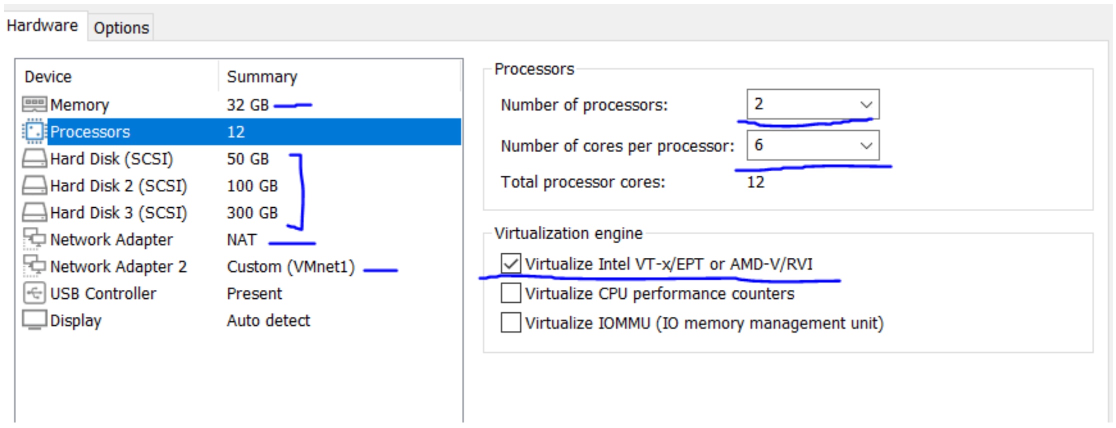
Technical setup

B.1 EVE-NG VM Setup and Settings

The virtual machine hosting EVE-NG is an Ubuntu-flavored Linux distribution, hence the installation process being very similar to installing a normal Ubuntu virtual machine on VMWare. It is quite simple and rather straightforward to do.

- First, the group downloaded the ISO file provided on the EVE-NG website [79]
- Then, a new virtual machine was created and set-up on VMWare, given the requirements shown in figure B.1, and the ones provided by EVE-NG in their cookbook [63].

The configuration that the group ended up with can be seen in figure B.1.



Device	Summary
Memory	32 GB
Processors	12
Hard Disk (SCSI)	50 GB
Hard Disk 2 (SCSI)	100 GB
Hard Disk 3 (SCSI)	300 GB
Network Adapter	NAT
Network Adapter 2	Custom (VMnet1)
USB Controller	Present
Display	Auto detect

Processors

Number of processors:	2
Number of cores per processor:	6
Total processor cores:	12

Virtualization engine

- [✓] Virtualize Intel VT-x/EPT or AMD-V/RVI
- [ ] Virtualize CPU performance counters
- [ ] Virtualize IOMMU (IO memory management unit)

Figure B.1: EVE-NG Virtual Machine Configuration

As can be seen in figure B.1, the machine has been allocated 32 gigabytes of RAM, a total of 2 processors with 12 cores, and a total of 350 GB of hard drive space. Consequently, the machine has been configured to use `NAT` as well as the `Host-only` interface.

The details about two virtual network interfaces, `NAT` and `Host-only`, can be found in section 4.3. Initially, the group decided to use the host-only connection as it provides an isolated private network for malware analysis, However it would not allow the EVE-NG virtual machine to communicate with the internet, and also the node within the topology on the EVE-NG virtual machine.

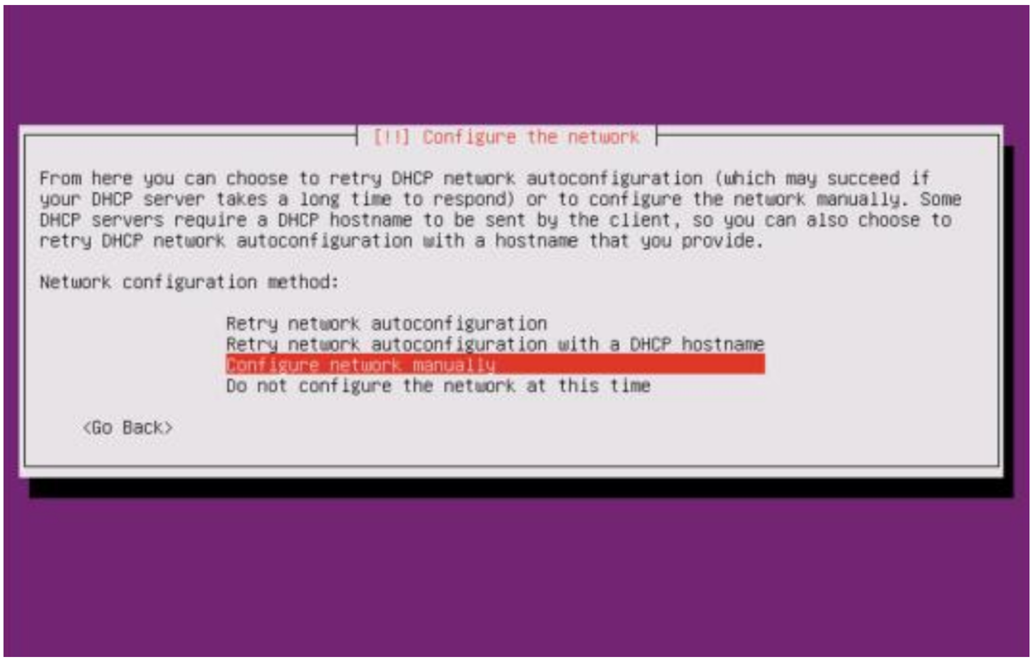
Therefore `NAT` connection mode was preferred as it provides a separate sub-net from the host network and enables the virtual machine to communicate with the internet for installing dependencies.

The sub-net used for NAT connection mode is `192.168.45.0/24`, and the static IP address used to access the web interface of the EVE-NG is `192.168.45.128` and the gateway address is `192.168.45.1/24`.

The inter-connectivity between the nodes on the internal network used for testing, as depicted in 4.1, is handled and created through a sub-net on EVE-NG itself, and is thus not part of the virtual machine's configuration on VMWare.

Once the configuration for the VM is set up, EVE-NG is then installed through the ISO file downloaded from EVE-NG's website. The important configuration, while installing the EVE-NG on VMware Workstation is configuring the network or sub-net `192.168.45.0/24` and the static IP address `192.168.45.128/24` that we want to use to access the web interface of the EVE-NG machine. In this section, the installation will not be described in detail, although it, if desired, can be found in [63]. However, The network configuration in EVE-NG installation is described below. We can see the configuration options provided by the EVE-NG installation set up in fig. B.2.

- In the network configuration window we select configure network manually instead of DHCP
- The desirable EVE-NG Web Interface IP is entered i.e. `192.168.45.128/24`
- The gateway for the sub-net is configured as `192.168.45.1/24`
- Other configuration for the network can be configured according to the requirements, though the aforementioned ones were the only ones needed for this stage of the project.



```
[!!] Configure the network

From here you can choose to retry DHCP network autoconfiguration (which may succeed if
your DHCP server takes a long time to respond) or to configure the network manually. Some
DHCP servers require a DHCP hostname to be sent by the client, so you can also choose to
retry DHCP network autoconfiguration with a hostname that you provide.

Network configuration method:

                         Retry network autoconfiguration
                         Retry network autoconfiguration with a DHCP hostname
                         Configure network manually
                         Do not configure the network at this time

    <Go Back>
```

Figure B.2: Network Configuration for EVE-NG virtual machine

Consequently, following the network configuration and installation of EVE-NG, the group is greeted with a message from the EVE-NG VM, confirming the web interface address:

```
http://192.168.45.128/
```

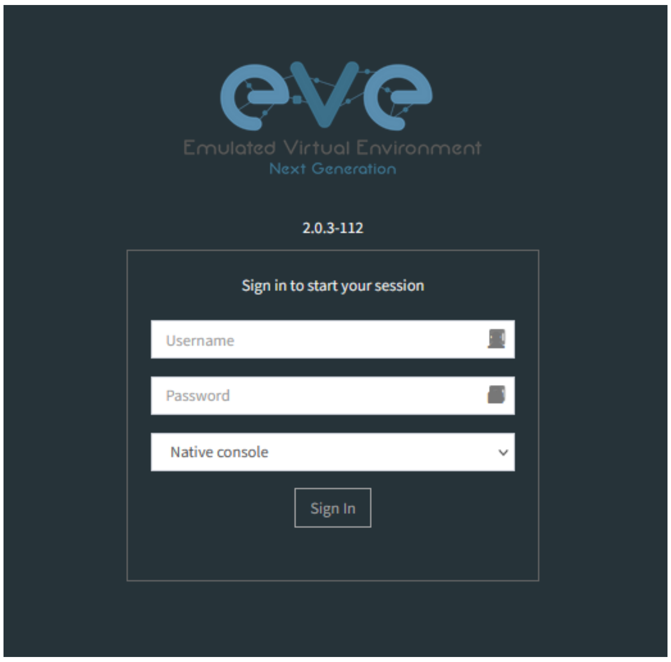


Figure B.3: EVE-NG login screen

The default username for the virtual machine is **root** and the password **eve**. In order to access the EVE-NG web interface, the username **admin** and password **eve** are used.

Visiting the aforementioned IP address prompts the group for a username and password see fig. B.3, in addition to a drop-down menu with two options: an HTML5 console and the native console. There is a slight difference between the two consoles regarding their functionality [90]. Since the project goals can be achieved using the native console, it will be used as the main console for the project when dealing with EVE-NG.

Once EVE-NG is running and accessible, the next step would be to upload the desired images to EVE-NG. These images are an integral part of setting up the network topology and infrastructure that will be used for malware analysis.

B.2 Import and usage of disk images in EVE-NG

EVE-NG is an Ubuntu Server-based Virtual Machine (VM) that supports pre-configured multiple hypervisors on one virtual machine, and essentially accepts three types of images, see below [91].

- QEMU images to emulate real world devices running Windows, Linux, Mikrotik, Cisco, and many others [92].
- Dynamips to run virtual Cisco devices. These are Cisco router emulated images used to emulate routers as if they were physical devices [93].
- Cisco IOL (Internet Operating System on Linux) images can also be configured to run on EVE-NG. These images are usually in **.bin** format, and are Cisco proprietary images, therefore IOL can only be used by Cisco employees or by authorized consumers [94].

Initially, to configure the topology in fig. 4.1 the group needed to have a switch or a router for the topology to communicate. Other open source images like Linux router template and pfsense can be used as the communicating device for the nodes in the topology.

However the image the group ended up using is a Cisco IOL image of a Cisco switch. As mentioned previously, IOL images from Cisco are only to be used by authorized customers; one of the group members already has been granted access to the image from the member's Cisco certification. The image is trusted and lightweight, compared to other open source router images available.

In addition to the aforementioned images, the remaining images, i.e. Ubuntu, Windows 7, Windows Server, Kali Linux and an Ubuntu LTS 18.04 image had to be acquired separately. Ubuntu and Window machine were acquired from their official websites and then converted to Qcow2 as mentioned in section 4.4. Furthermore, EVE-NG also provide some ready to go images which can be used for different purposes for instance Kali linux, and VirtualPC [95].

B.2.1 Uploading the disk images

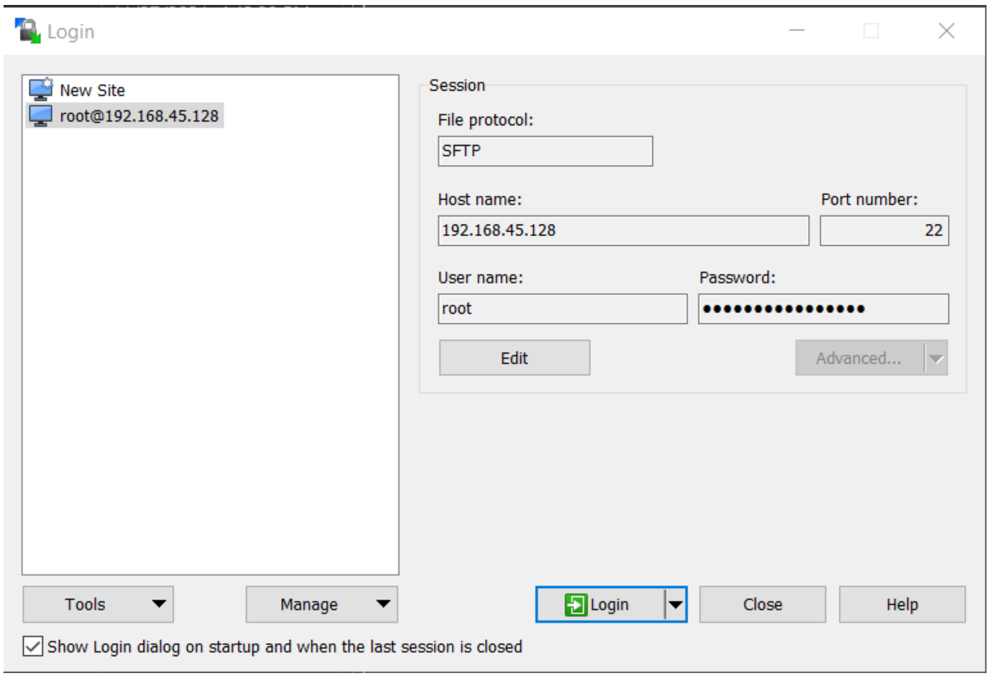
Now that we have one IOU image for the Cisco switch and disk files **.vmdk** for the Ubuntu, Windows virtual machine, after installing it on the VMware, we will upload them to the EVE-NG, and then we will convert them to the **.qcow2** format. Additionally, the list of the images that EVE-NG supports can be found her [91].

The directories where the images will be uploaded are as follow [96]:

```
# Dynamips images goes here
opt/unetlab/addons/dynamips
# IOL or IOU images will be uploaded here
/opt/unetlab/addons/iol/bin
# QEMU images upload here
```

```
/opt/unetlab/addons/qemu
```

Furthermore, the tool which is used by the group for uploading the images is WinScp, however any other SFTP connection tool, such as FileZilla. can be used. The setup of WinSCP can be seen in fig. B.4



Session
File protocol: SFTP
Host name: 192.168.45.128
Port number: 22
User name: root
Password: ●●●●●●●●●●●●●●●●
☑ Show Login dialog on startup and when the last session is closed

Figure B.4: WinSCP Setup for uploading images

B.2.2 Convert the VMware disk image to a QEMU disk image

This section will now describe, how the group converted the VMware disk image **.vmdk** of Cuckoo-FOG machine (UbuntuServer.vmdk) to (hda.qcow2, and hdb.qcow2) with the help of the QEMU imaging utility. The steps described as follows are also repeated to acquire **.qcow2** image disk for Windows virtual machines[97].

- SSH to EVE-NG virtual machine with root access from PuTTY, and create a directory with any name, for instance abc.

```
root@eve-ng:~#mkdir abc
root@eve-ng:~#cd abc
```

- Upload the `.vmdk` image to the EVE-NG `root/abc` directory using WinSCP.
- Convert the `.vmdk` hard disk file to the `.qcow2` format.

```
/opt/qemu/bin/qemu-img convert -f vmdk -O qcow2 UbuntuServer.vmdk hda
    .qcow2
```

- Now, create another hard disk, as this will be used as a storage unit by our VM.

```
/opt/qemu/bin/qemu-img create -f qcow2 hdb.qcow2 100G
```

- While working with QEMU disk images, it should be ensured that a directory with the image name in the image specific directory exists, see naming convention [98].

```
mkdir /opt/unetlab/addons/qemu/esxi-Cuckoo
```

- Move the created files, i.e. hda.qcow2 and hdb.qcow2 to the recently created directory.

```
mv hda.qcow2 hdb.qcow2 /opt/unetlab/addons/qemu/esxi-Cuckoo/
```

- Delete the directory **abc**, and fix the permissions accordingly, per the documentation.

```
/opt/unetlab/wrappers/unl\_wrapper -a fixpermissions
```

The attributes of the created disk image hda.qcow2 can be seen below.

```
root@eve-ng:~# cd /opt/unetlab/addons/qemu/esxi-Cuckoo/
root@eve-ng:/opt/unetlab/addons/qemu/esxi-Cuckoo# qemu-img info hda.qcow2
image: hda.qcow2
file format: qcow2
virtual size: 100G (107374182400 bytes)
disk size: 34G
cluster_size: 65536
Format specific information:
    compat: 1.1
    lazy refcounts: false
    refcount bits: 16
    corrupt: false
```

B.3 Building and configuring the topology

At this point, the starting blocks for building the desired topology in EVE-NG can be laid out. However, before diving deep into building the topology, and configuring the end-devices, a Window client-side pack, provided by EVE-NG, will be needed. This will install the necessary tools on the host machine in order to communicate with end devices of the topology.

EVE-NG also provides a client-side pack for Linux, though the host machine for this project is a Windows 10 machine. Therefore, the Windows client-side pack, which includes tools such as UltraVNC, Putty, and Wireshark, will be installed. [99]. Once it is installed, the group then pointed the browser of the host machine to the IP address of the EVE-NG and log in with the default credentials **admin: eve**, and the native console.

Once logged in to EVE-NG, navigate to Add new lab under File Manager to create the lab. The name and description thereof can be added, and then swiftly saved. see fig. B.5.

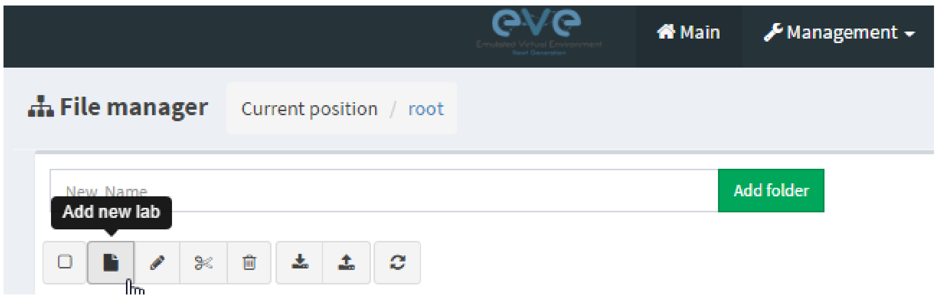


Figure B.5: Add new lab in EVE-NG

Building topology from the images that are uploaded to EVE-NG in the aforementioned appendix B.2 is relatively straightforward. However, for illustrative purposes, the process of adding an object or image for building the topology shall be described below. The same steps can be repeated to add other disk images as well. Firstly, a node containing the Cisco IOL switch image will be added.

- After saving the lab, right click in the empty grid and chose add a new node tab.

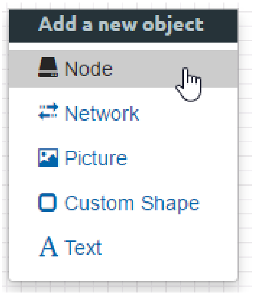
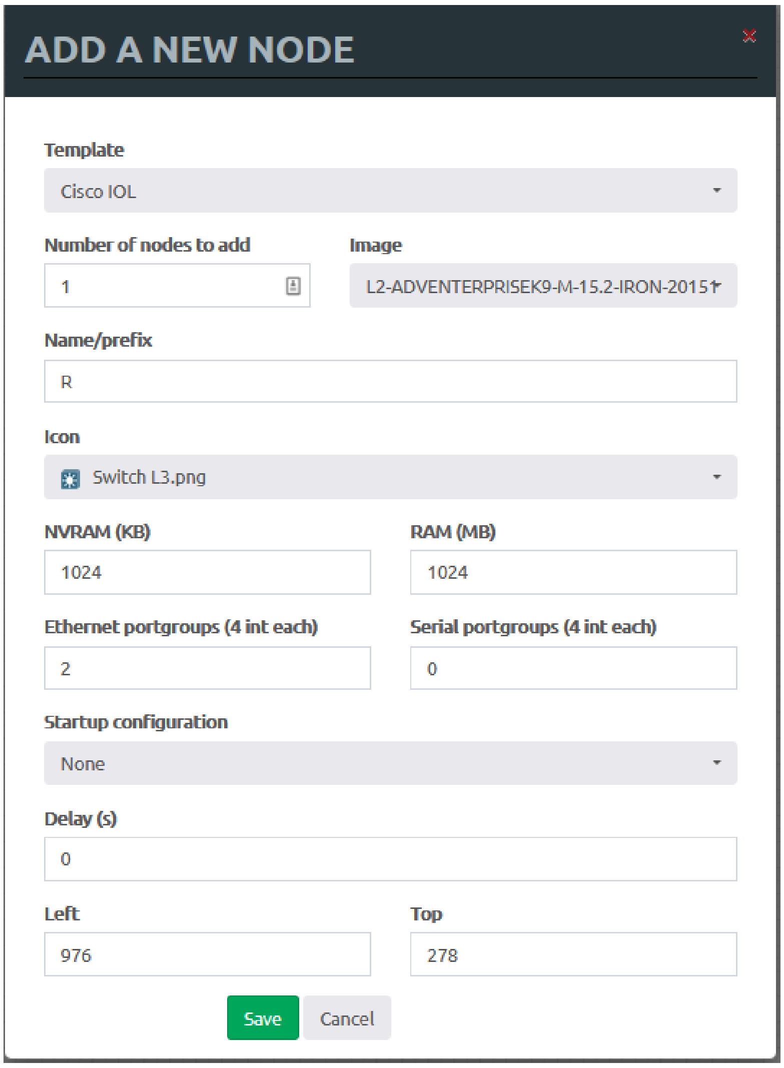


Figure B.6: Add new node to the topology

- By selecting the node, we will see a long drop-down list of the templates provided by the EVE-NG, which requires images from different vendors. We can however use the templates, with our uploaded images. Here we select Cisco IOL from the drop-down list, rename it and select the icon that we want to reflect it with in the topology.
- Furthermore, we can see in fig. B.7, that we have added two Ethernet port-groups of 4 interfaces each to the switch. This means we can add a total of 8 end devices to the switch.



ADD A NEW NODE

Template: Cisco IOL

Number of nodes to add: 1

Image: L2-ADVENTERPRISEK9-M-15.2-IRON-2015

Name/prefix: R

Icon: Switch L3.png

NVRAM (KB): 1024

RAM (MB): 1024

Ethernet portgroups (4 int each): 2

Serial portgroups (4 int each): 0

Startup configuration: None

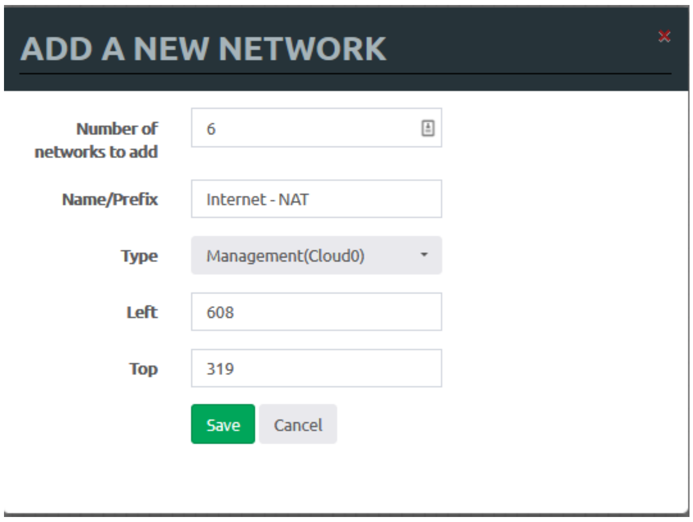
Delay (s): 0

Left: 976

Top: 278

Figure B.7: Adding switch node to the topology

- Similarly, we added the rest of the end devices in the topology and connected them to the switch on different interfaces as show in fig. 4.1.
- subsequently, we also want to add a network interface in the topology, which will connect to the cuckoo host machine, and will be used to provide internet connectivity to the cuckoo host machine for installing and configuring the cuckoo sandbox, its respective dependencies, and the FOG imaging server respectively. This will take an IP address from the **192.168.45.0/24** NAT vNIC of the VMware. As shown in fig. B.8


ADD A NEW NETWORK

Number of networks to add	6
Name/Prefix	Internet - NAT
Type	Management(Cloud0)
Left	608
Top	319

Figure B.8: Internet connectivity to cuckoo host machine

At this point, the topology is ready for configuration, initially, we will start configuring the switch, which will work as the default gateway, and DHCP server for both VLAN10, and VLAN20 as described in section 4.4.1. the group started the switch by right click on the device and then click start, after starting the switch we were able to get a telnet connection from the switch with SecureCRT application, any other tool like Putty can also be used for this purpose.

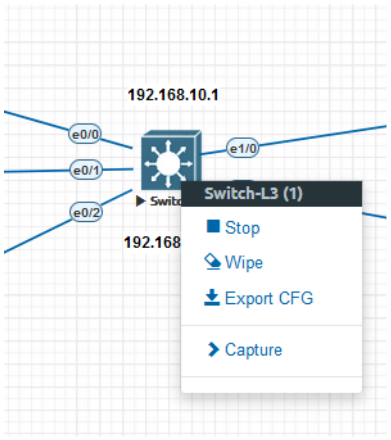
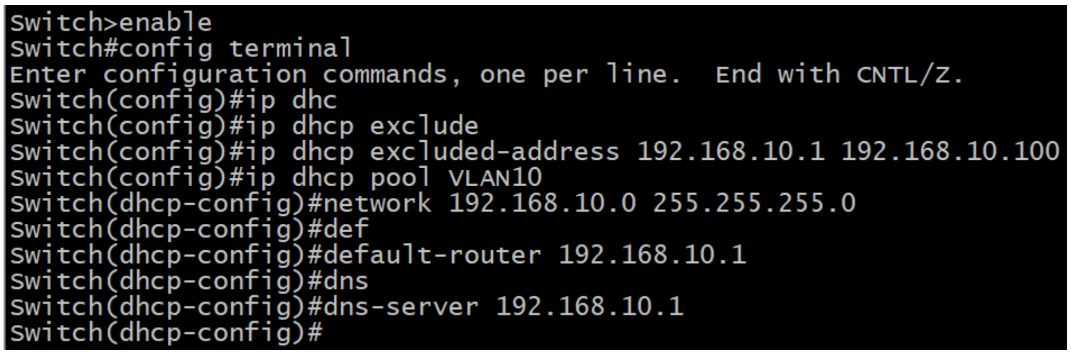


Figure B.9: Getting telnet connection from the switch

After getting the telnet connection we need to know some of the Cisco commands, the Cisco user interface is divided into many different modes for instance User EXEC, Privileged EXEC, and Global configuration. Therefore the commands depend on the mode in which the user is [100].

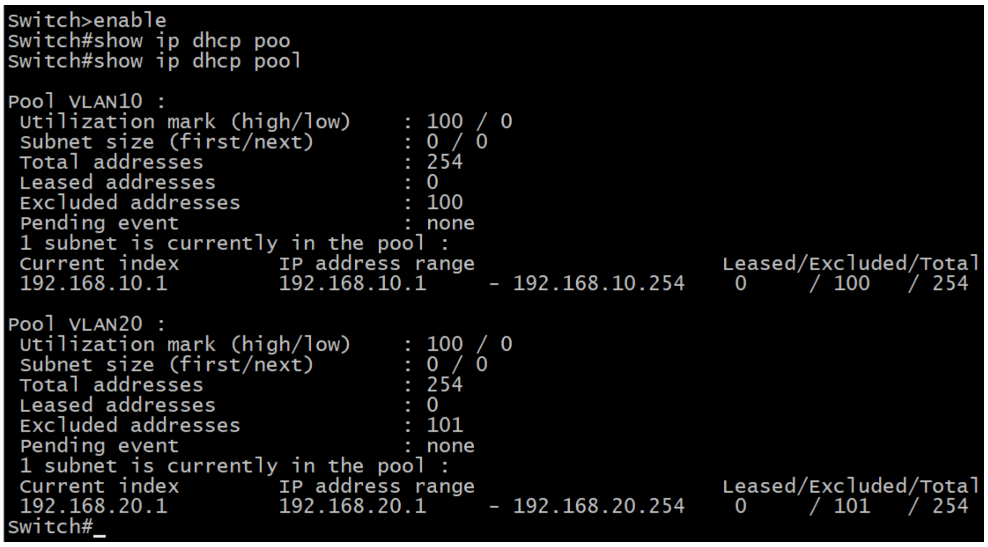
Once we are familiar with different modes, the following fig. B.10 will show the configuration of the DHCP on the switch for VLAN10 and will also depict excluded addresses, that we want to use for static purposes in our topology[101].



```
Switch>enable
Switch#config terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Switch(config)#ip dhc
Switch(config)#ip dhcp exclude
Switch(config)#ip dhcp excluded-address 192.168.10.1 192.168.10.100
Switch(config)#ip dhcp pool VLAN10
Switch(dhcp-config)#network 192.168.10.0 255.255.255.0
Switch(dhcp-config)#def
Switch(dhcp-config)#default-router 192.168.10.1
Switch(dhcp-config)#dns
Switch(dhcp-config)#dns-server 192.168.10.1
Switch(dhcp-config)#
```

Figure B.10: Configuration of DHCP

Similarly, we also configured the DHCP for VLAN20. On top of that, the group also segregated the network of the switch by creating two VLANS, VLAN10, and VLAN20, following fig. B.11 depicts the **show** command results of the VLANS, and DHCP configuration [102].

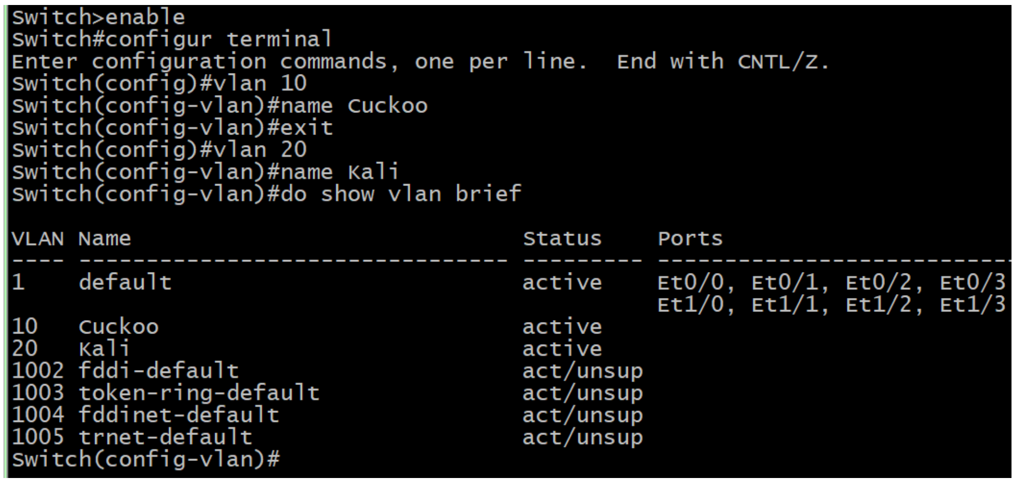


```
Switch>enable
Switch#show ip dhcp poo
Switch#show ip dhcp pool

Pool VLAN10 :
 Utilization mark (high/low)    : 100 / 0
 Subnet size (first/next)       : 0 / 0
 Total addresses                : 254
 Leased addresses               : 0
 Excluded addresses             : 100
 Pending event                  : none
 1 subnet is currently in the pool :
 Current index        IP address range                    Leased/Excluded/Total
 192.168.10.1         192.168.10.1     - 192.168.10.254    0     / 100   / 254

Pool VLAN20 :
 Utilization mark (high/low)    : 100 / 0
 Subnet size (first/next)       : 0 / 0
 Total addresses                : 254
 Leased addresses               : 0
 Excluded addresses             : 101
 Pending event                  : none
 1 subnet is currently in the pool :
 Current index        IP address range                    Leased/Excluded/Total
 192.168.20.1         192.168.20.1     - 192.168.20.254    0     / 101   / 254
Switch#_
```

Figure B.11: Show command results for DHCP configuration

We are not allowing the two VLANs to communicate with each other, therefore we did not configure the interVLAN routing between the VLANs. VLAN10 was named Cuckoo and VLAN20 was named Kali. fig. B.12 shows how to create

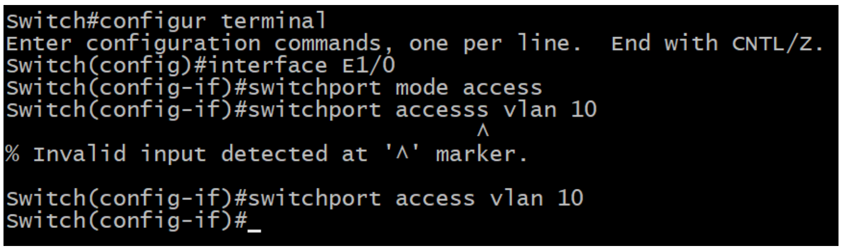
VLANs on the switch.



```
Switch>enable
Switch#configur terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Switch(config)#vlan 10
Switch(config-vlan)#name Cuckoo
Switch(config-vlan)#exit
Switch(config)#vlan 20
Switch(config-vlan)#name Kali
Switch(config-vlan)#do show vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- --------------------------
1    default                          active    Et0/0, Et0/1, Et0/2, Et0/3
                                                Et1/0, Et1/1, Et1/2, Et1/3
10   Cuckoo                           active
20   Kali                             active
1002 fddi-default                     act/unsup
1003 token-ring-default               act/unsup
1004 fddinet-default                  act/unsup
1005 trnet-default                    act/unsup
Switch(config-vlan)#
```

Figure B.12: VLAN configuration on the switch

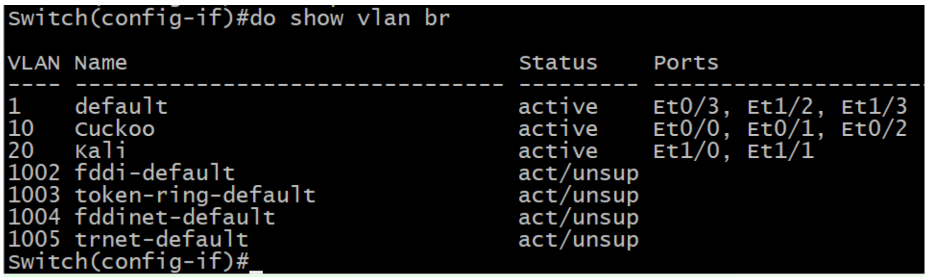
In the aforementioned diagram, we can see that all the interfaces are part of the default vlan1.To make the end device part of the specific VLAN, each interface port connected to the end-devices were configured to access the required VLAN respectively. fig. B.13 shows how to configure a port as an access port in VLAN 10 [103].



```
Switch#configur terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Switch(config)#interface E1/0
Switch(config-if)#switchport mode access
Switch(config-if)#switchport accesss vlan 10
                                   ^
% Invalid input detected at '^' marker.

Switch(config-if)#switchport access vlan 10
Switch(config-if)#_
```

Figure B.13: Configure a port as an access port in VLAN 10

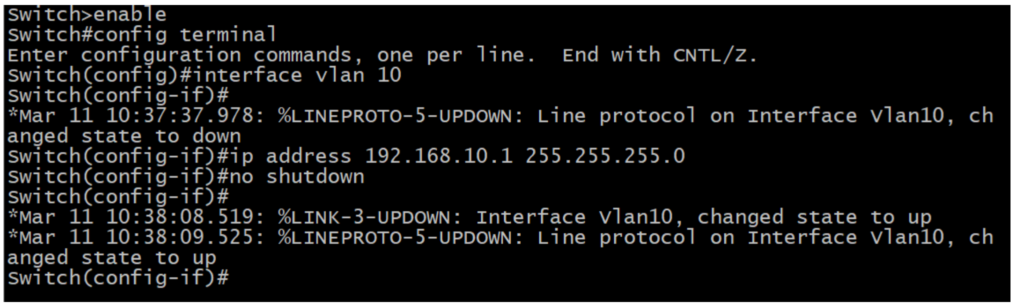
Similarly, all the ports in the switch connected to different endpoints in the topology were configured in their respective VLAN. Below we can see different ports within their respective VLANS, for instance, Ethernet port 0/0 is connected to Windows7-Hardened machine, Ethernet port 0/1 is connected to Windows-Weak machine, and Ethernet port 0/2 is connected to Cuckoo-Fog machine, and they are all part of the VLAN10 as show in fig. 4.1.



```
Switch(config-if)#do show vlan br

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------
1    default                          active    Et0/3, Et1/2, Et1/3
10   Cuckoo                           active    Et0/0, Et0/1, Et0/2
20   Kali                             active    Et1/0, Et1/1
1002 fddi-default                     act/unsup
1003 token-ring-default               act/unsup
1004 fddinet-default                  act/unsup
1005 trnet-default                    act/unsup
Switch(config-if)#
```

Figure B.14: Switch Port access to their respective VLANS

The last think to configure on the switch at this point is to configure the VLANs interface for an IP address **192.168.10.1/24** on VLAN10, and **192.168.20.1/24** on VLAN20, which will work as the default gateway for the VLANs respectively[104]. fig. B.15 shows the IP address configuration on VLAN10 switch interface.



```
Switch>enable
Switch#config terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Switch(config)#interface vlan 10
Switch(config-if)#
*Mar 11 10:37:37.978: %LINEPROTO-5-UPDOWN: Line protocol on Interface Vlan10, ch
anged state to down
Switch(config-if)#ip address 192.168.10.1 255.255.255.0
Switch(config-if)#no shutdown
Switch(config-if)#
*Mar 11 10:38:08.519: %LINK-3-UPDOWN: Interface Vlan10, changed state to up
*Mar 11 10:38:09.525: %LINEPROTO-5-UPDOWN: Line protocol on Interface Vlan10, ch
anged state to up
Switch(config-if)#
```

Figure B.15: Configuration of the Switch VLAN Interface

At this stage, the configuration that the group wanted to do on the switch is now done, all the configuration on the switch can be checked and confirmed with the below command in the privilege EXEC mode of the switch.

```
Switch#show running-config
```

Finally, we need to save the running configuration on the switch, to the startup configuration. We can do that with one of the following commands.

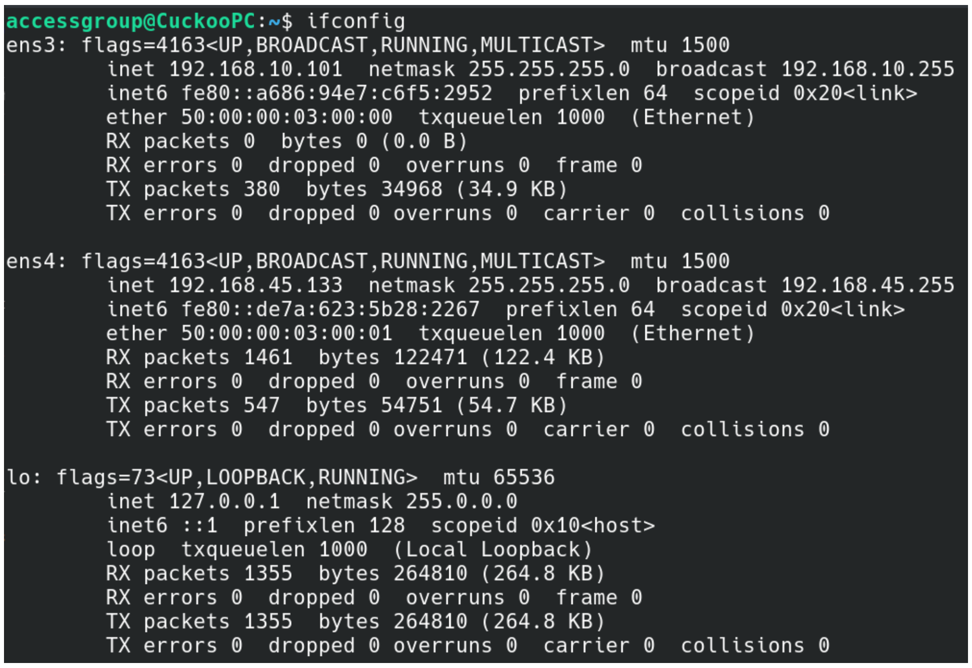
```
Switch#copy running-config startup-config
OR
Switch#write
```

B.4 Cuckoo Sandbox Setup

This part of the report will now explain the process of installing, and configuring the Cuckoo sandbox on the virtual machine reflected as Cuckoo-FOG in fig. 4.1, which is an Ubuntu based VM. However, before diving deeper into the installation and configuration of the Cuckoo sandbox, we first need to understand the network configuration of the VM.

In fig. B.16 we can see that this VM has three interfaces:

- **ens3** with IP address: 192.168.10.101/24, which is used to connect the VM to the internal subnet of VLAN10, and will be configured to speak to the guest VM running Windows 7, with the help of FOG imaging server.
- **ens4** with IP address: 192.168.45.133/24, which is connected to the cloud, and to the external NAT vNIC of the VMware machine. This will be used by the machine to gain access to the internet for installation and configuration purposes.
- **loopback** with IP address: 127.0.0.1/8, which is used to access the web interface of the Cuckoo sandbox.



```
accessgroup@CuckooPC:~$ ifconfig
ens3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.10.101  netmask 255.255.255.0  broadcast 192.168.10.255
        inet6 fe80::a686:94e7:c6f5:2952  prefixlen 64  scopeid 0x20<link>
        ether 50:00:00:03:00:00  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 380  bytes 34968 (34.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens4: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.45.133  netmask 255.255.255.0  broadcast 192.168.45.255
        inet6 fe80::de7a:623:5b28:2267  prefixlen 64  scopeid 0x20<link>
        ether 50:00:00:03:00:01  txqueuelen 1000  (Ethernet)
        RX packets 1461  bytes 122471 (122.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 547  bytes 54751 (54.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 1355  bytes 264810 (264.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1355  bytes 264810 (264.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure B.16: Interfaces of Cuckoo-FOG virtual machine

To start off with installing Cuckoo sandbox, the group has been through extensive configuration of the Ubuntu VM in order to get cuckoo installed properly. As part of Cuckoo, various tools have been downloaded and installed, for instance :

- **Volatility** for Forensic analysis on memory dumps
- **YARA** for Malware identification and classifications
- **MongoDB** for Database Management
- **TCPDump** for Network packet analyser

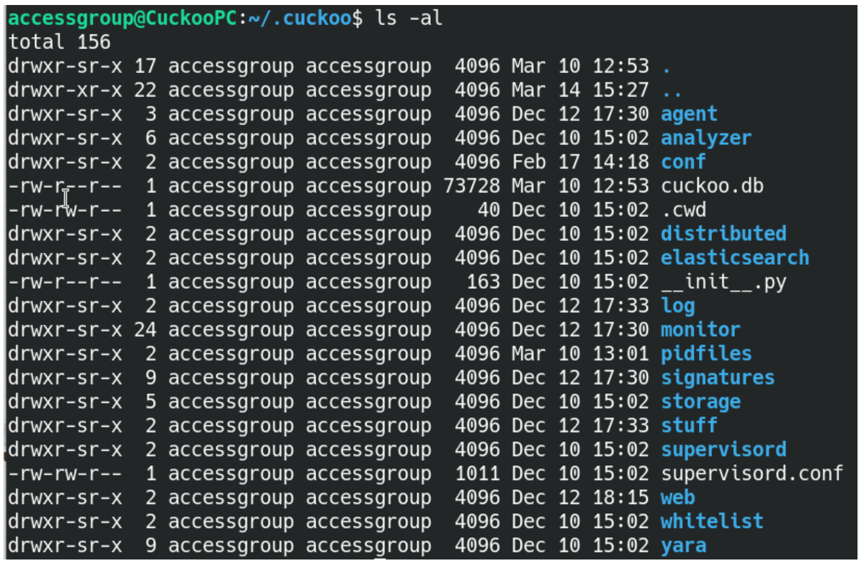
On top of that a cuckoo user was also created, which we did not used to log in to, though is used by the cuckoo sandbox. The list of the installed dependencies for Cuckoo can be found in the proof of concept code repository appendix D. For installing these dependencies, the group got inspiration from [105].

After the installation of dependencies and tools that were needed for Cuckoo, the group installed the sandbox itself, which can be installed in different ways [106].

```
#Checks for the requirements
pip install -U pip setuptools
#install the cuckoo sandbox
sudo -H pip install -U cuckoo
#create default directories
cuckoo
```

This installation will create different directories, in the path of **Cuckoo Working Directory(CWD)**.

In our case the CWD is on the default path, which is `/home/cuckoo/.cuckoo`. The directories are used to store the results of the Cuckoo, configurable components, and generated data by the analysis. Similarly the agent directory, which consists of `agent.py` and `agent.sh` will be used to speak with the cuckoo guest machine. The files, and directories includes the following [107], described in fig. B.17.



```
accessgroup@CuckooPC:~/.cuckoo$ ls -al
total 156
drwxr-sr-x 17 accessgroup accessgroup  4096 Mar 10 12:53 .
drwxr-xr-x 22 accessgroup accessgroup  4096 Mar 14 15:27 ..
drwxr-sr-x  3 accessgroup accessgroup  4096 Dec 12 17:30 agent
drwxr-sr-x  6 accessgroup accessgroup  4096 Dec 10 15:02 analyzer
drwxr-sr-x  2 accessgroup accessgroup  4096 Feb 17 14:18 conf
-rw-r--r--  1 accessgroup accessgroup 73728 Mar 10 12:53 cuckoo.db
-rw-rw-r--  1 accessgroup accessgroup    40 Dec 10 15:02 .cwd
drwxr-sr-x  2 accessgroup accessgroup  4096 Dec 10 15:02 distributed
drwxr-sr-x  2 accessgroup accessgroup  4096 Dec 10 15:02 elasticsearch
-rw-r--r--  1 accessgroup accessgroup   163 Dec 10 15:02 __init__.py
drwxr-sr-x  2 accessgroup accessgroup  4096 Dec 12 17:33 log
drwxr-sr-x 24 accessgroup accessgroup  4096 Dec 12 17:30 monitor
drwxr-sr-x  2 accessgroup accessgroup  4096 Mar 10 13:01 pidfiles
drwxr-sr-x  9 accessgroup accessgroup  4096 Dec 12 17:30 signatures
drwxr-sr-x  5 accessgroup accessgroup  4096 Dec 10 15:02 storage
drwxr-sr-x  2 accessgroup accessgroup  4096 Dec 12 17:33 stuff
drwxr-sr-x  2 accessgroup accessgroup  4096 Dec 10 15:02 supervisord
-rw-rw-r--  1 accessgroup accessgroup  1011 Dec 10 15:02 supervisord.conf
drwxr-sr-x  2 accessgroup accessgroup  4096 Dec 12 18:15 web
drwxr-sr-x  2 accessgroup accessgroup  4096 Dec 10 15:02 whitelist
drwxr-sr-x  9 accessgroup accessgroup  4096 Dec 10 15:02 yara
```

Figure B.17: Directories created by cuckoo in the CWD

All the configuration files, that we will be configure to make cuckoo work properly are in the directory `$CWD/conf` i.e `$CWD/conf/cuckoo.conf`, `$CWD/conf/physical.conf` and so on. we configured these files as follows:

```
sudo nano .cuckoo/conf/cuckoo.conf
version_check = yes
#This will check physical.conf
machinery = physical
memory_dump = yes
```

```
sudo nano .cuckoo/conf/physical.conf
[physical]
machines = Win7-Hardened,Win7-Weak,Win7-blind
#Credentials to access the guest machines
user = Hardened
password = Test123
#Default network interface.
interface = ens3
[fog]
hostname = 192.168.10.101
username = fog
password = password
[Win7-Hardened] # Windows 7 Hardened
label = Win7-Hardened
ip = 192.168.10.102
[Win7-Weak] # Windows 7 Weak
label = Win7-Weak
ip = 192.168.10.103
[Win7-blind] # Windows 7 Blind
```

```
label = Win7-blind
ip = 192.168.10.104
```

```
sudo nano .cuckoo/conf/memory.conf
#Guest Machine
guest_profile = Win7SP1x86_64
delete_memdump = yes
```

```
sudo nano .cuckoo/conf/processing.conf
[memory]
enabled = yes
```

```
sudo nano .cuckoo/conf/reporting.conf
[singlefile]
# Enable creation of report.html and/or report.pdf?
enabled = yes
# Enable creation of report.html?
html = yes
# Enable creation of report.pdf?
pdf = no
```

B.4.1 Setting up the FOG project

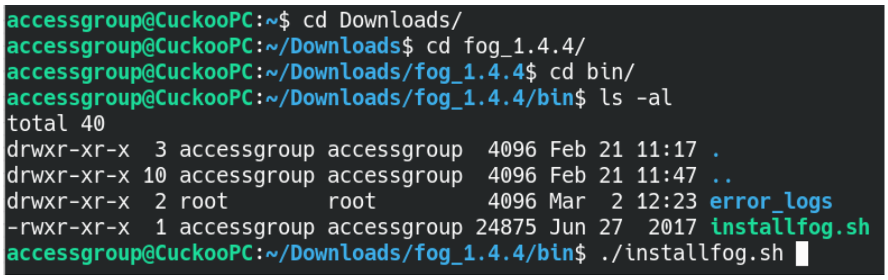
This section narrates the installation and settings of the FOG Imaging server, also known as FOG project as described in section 4.2. FOG imaging server or FOG project is installed on the same Ubuntu machine, on which the group installed the cuckoo sandbox.

Furthermore, the purpose of installing FOG is to have a way for cuckoo guest machine to returned back to a clean state after the analysis. Initially, the group installed and configured FOG project, v1.5.9. However, Cuckoo only supports some specific version i.e., 1.3.4 and 1.4.4 [108]. To install and configure FOG project the following steps were taken.

- Download the FOG project v1.4.4 or 1.3.4 with `wget` from FOG project website [109]
- Now change to Downloads directory and unzip the file.

```
accessgroup@CuckooPC:~#cd Downloads/
accessgroup@CuckooPC:~#tar -zsvf fog_1.4.4.tar.gz
```

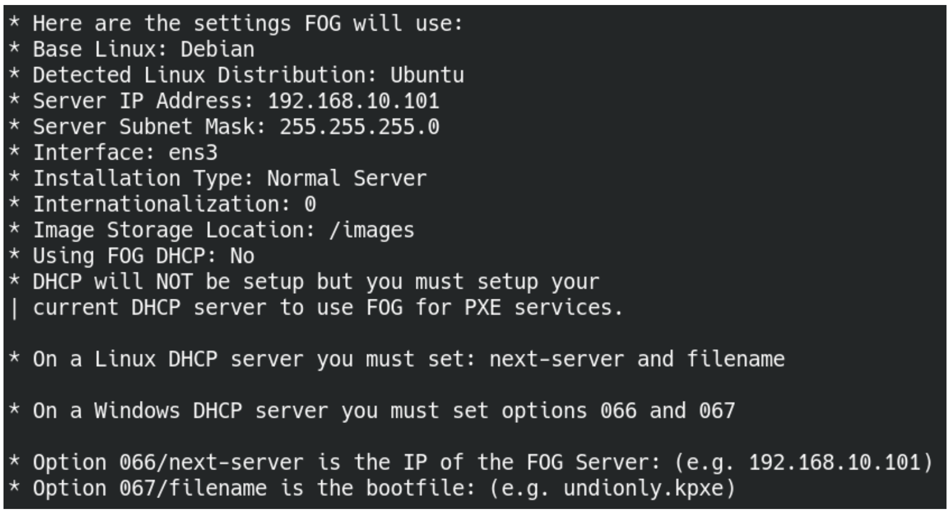
- Now change to the unzip directory and then to `bin` directory within `fog_1.4.4`, and run the file `installfog.sh`, as shown below in fig. B.18.



```
accessgroup@CuckooPC:~$ cd Downloads/
accessgroup@CuckooPC:~/Downloads$ cd fog_1.4.4/
accessgroup@CuckooPC:~/Downloads/fog_1.4.4$ cd bin/
accessgroup@CuckooPC:~/Downloads/fog_1.4.4/bin$ ls -al
total 40
drwxr-xr-x  3 accessgroup accessgroup  4096 Feb 21 11:17 .
drwxr-xr-x 10 accessgroup accessgroup  4096 Feb 21 11:47 ..
drwxr-xr-x  2 root        root         4096 Mar  2 12:23 error_logs
-rwxr-xr-x  1 accessgroup accessgroup 24875 Jun 27  2017 installfog.sh
accessgroup@CuckooPC:~/Downloads/fog_1.4.4/bin$ ./installfog.sh
```

Figure B.18: Installing FOG Project

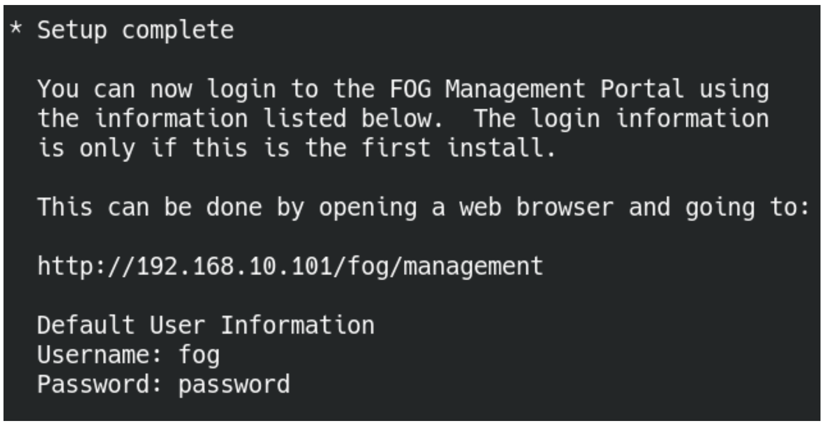
- The FOG project installation will now ask for some settings, for instance, server IP address for FOG server,which is `192.168.10.101/24`, the Interface that we wanted to use is `ens3`, the setting can be configured depending upon the requirements, and setup that one would like to use [110]. In our case, the settings can be seen in fig. B.19.



```
* Here are the settings FOG will use:
* Base Linux: Debian
* Detected Linux Distribution: Ubuntu
* Server IP Address: 192.168.10.101
* Server Subnet Mask: 255.255.255.0
* Interface: ens3
* Installation Type: Normal Server
* Internationalization: 0
* Image Storage Location: /images
* Using FOG DHCP: No
* DHCP will NOT be setup but you must setup your
| current DHCP server to use FOG for PXE services.

* On a Linux DHCP server you must set: next-server and filename

* On a Windows DHCP server you must set options 066 and 067

* Option 066/next-server is the IP of the FOG Server: (e.g. 192.168.10.101)
* Option 067/filename is the bootfile: (e.g. undionly.kpxe)
```

Figure B.19: settings of the FOG imaging server

- With the settings of the FOG, we can also see a message about setting up of the DHCP options, i.e. option 066, and option 067 need to be configured on the DHCP server. The DHCP server in our case is Cisco switch, which will be configured for these options later on.
- Once the installation is completed. FOG sever can be access through the following IP. see fig. B.20



```
* Setup complete

  You can now login to the FOG Management Portal using
  the information listed below.  The login information
  is only if this is the first install.

  This can be done by opening a web browser and going to:

  http://192.168.10.101/fog/management

  Default User Information
  Username: fog
  Password: password
```

Figure B.20: FOG management portal information

- At this point, the FOG imaging server is installed, and ready to take an image from the physical machines. Hence we will now setup our physical machines, and image them with the help of PXE network boot.

B.4.2 The physical machines

This section describes the configuration of the Windows 7 VMs as depicts in fig. 4.1, These machines are . Since we will configure this machine as a cuckoo guest machine, and as a separate entity on the network. This is therefore refer to as a physical machine. Which can be deployed in a real network. In our case, we will configure it in an emulated corporate network infrastructure, and we will also use the term physical machines [83].

Each machine is named in accordance with the artefacts they provide to the malware, and access to the network within the topology.

- **Win7-Weak** machine without anti-evasion techniques, plus network access
- **Win7-Blind** machine without anti-evasion techniques plus without network access
- **Win7-Hardened** machine with anti-evasion techniques, plus network access

The following changes are made to all of the three machine, Which are required by the cuckoo host machine to communicate with them.

- **Python** : Allows the agent to run and collect data from the physical machine

- **Pillow (Python Imaging Library)** : This adds capability to the python to take snapshots, and is used to take snapshots from the physical machine desktop.
- **Cuckoo Agent - older version** : Allows the VM to send data back and forth to the Cuckoo host machine. We used the older version of cuckoo agent, because only this version works with physical machine. The agent file is a Python script, which should be executed upon startup. This ensures that the Cuckoo host machine can communicate with the guest. The file is placed in the following directory, allowing it to launch when the user account **Hardened** is logged in to.

```
C:\Users\$USERNAME$\AppData\Roaming\Microsoft\Windows\Start Menu\
    Programs\Startup
```

- **Enable auto-logon** (Allows for the agent to start upon reboot)
- **Enable Remote RPC** (Allows for Cuckoo to reboot the sandbox using RPC). This can be achieved by running the following commands in cmd as admin.

```
reg add "hklm\software\Microsoft\Windows NT\CurrentVersion\WinLogon"
    /v DefaultUserName /d Hardened /t REG_SZ /f
reg add "hklm\software\Microsoft\Windows NT\CurrentVersion\WinLogon"
    /v DefaultPassword /d Test123 /t REG_SZ /f
reg add "hklm\software\Microsoft\Windows NT\CurrentVersion\WinLogon"
    /v AutoAdminLogon /d 1 /t REG_SZ /f
reg add "hklm\system\CurrentControlSet\Control\TerminalServer" /v
    AllowRemoteRPC /d 0x01 /t REG_DWORD /f
reg add "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion
    \Policies\System" /v LocalAccountTokenFilterPolicy /d 0x01 /t
    REG_DWORD /f

```

Physical machine: Win7-Weak

The win7-Weak machine is left with the default aforementioned configuration, the default gateway, and the DNS are set to the switch IP instead of the host machine IP address, allowing it to send traffic to the network.

```
IP Address: 192.168.10.103
Subnet Mask: 255.255.255.0
Default Gateway: 192.168.10.1
Primary DNS: 192.168.10.1
```

Physical machine: Win7-Blind

The win7-blind machine has also the aforementioned configuration. However, to isolate it from the network the following configuration was added to the switch in order to make it communicate only with the host machine, without having any interaction with the network. This is achieved through the VLAN Access Control list and VLAN access map by configuring them in the switch's global configuration mode. This will make Win7-blind to see only cuckoo host machine and not other machine on the network [85].

```
ip access-list extended blind-cuckoo
permit ip host 192.168.10.104 host 192.168.10.101
exit
ip access-list extended blind-vlan
permit ip host 192.168.10.104 192.168.10.0 0.0.0.255
exit
vlan access-map vacl1 10
match ip address blind-cuckoo
action forward
exit
vlan access-map vacl1 20
match ip address blind-vlan
action drop
exit
vlan access-map vacl1 30
action forward
exit
vlan filter vacl1 vlan-list 10
exit
```

Code B.1: VLAN Access Control list

Additionally, the default gateway and the DNS for the machine are set to the host machine IP. which can be seen as follows.

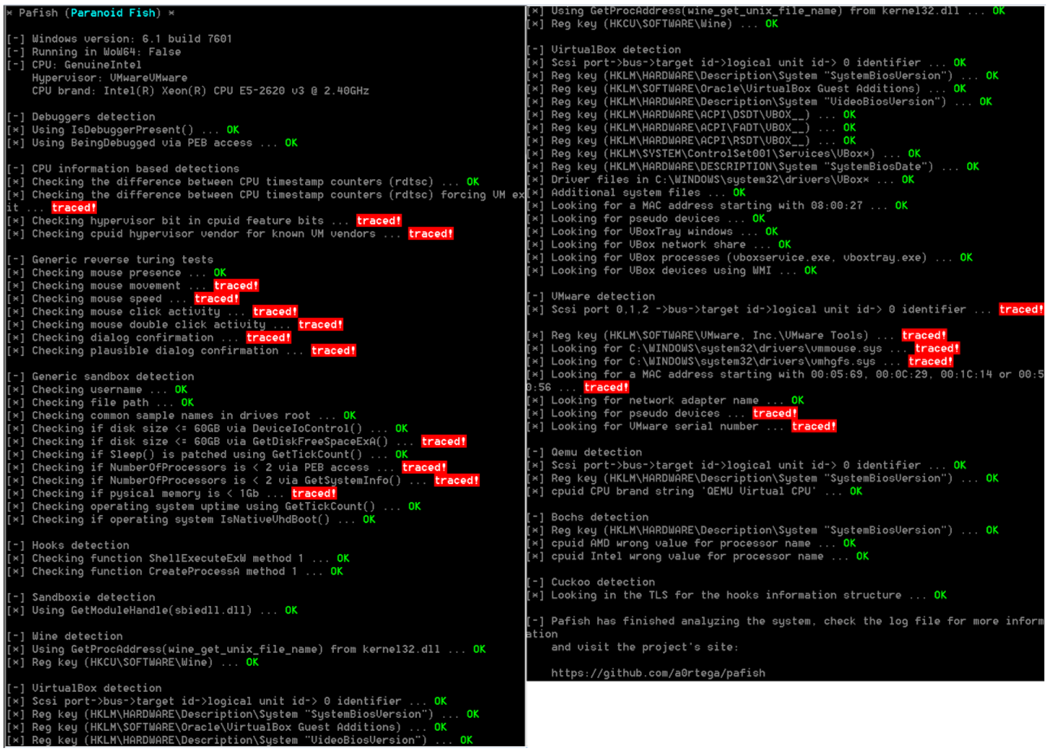
```
IP Address: 192.168.10.104
Subnet Mask: 255.255.255.0
Default Gateway: 192.168.10.101
Primary DNS: 192.168.10.101
```

Physical machine: Win7-Hardened

With the default configuration, the group also implemented some anti-evasion tools on this machine to make it difficult for the malware to detect if it is running in a virtual environment. The tools used by the group are as follows:

Pafish Pafish is a tool that detects the virtual machine or malware analysis environments with different techniques that malware usually uses to see them. Pafish can be downloaded from here [72]. The process of installing them on the hardened machine and then rectifying the artefacts from the machine is straightforward [44]. Additionally, the group could not remove all the artefacts from the machine since every tool looked for different artefacts for a virtual machine, and some of them were not possible to be mitigated.

However, following we will show the results of Pafish before see fig. B.21 and after the mitigation of the artefacts see fig. B.22.

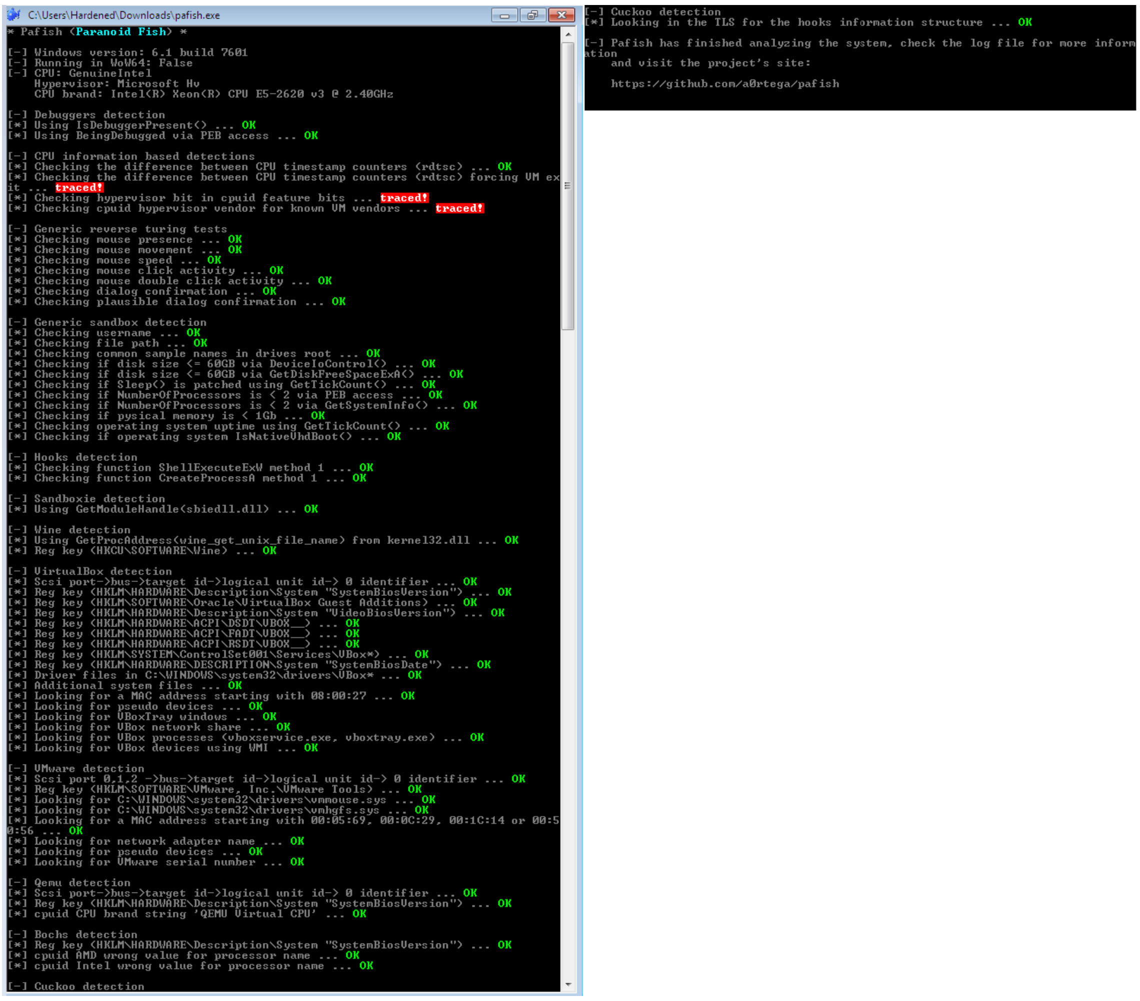


```
* Pafish (Paranoid Fish) *

[-] Windows version: 6.1 build 7601
[-] Running in WoW64: False
[-] CPU: GenuineIntel
    Hypervisor: VMwareVMware
    CPU brand: Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK
[*] Using BeingDebugged via PEB access ... OK

[-] CPU information based detections
[*] Checking the difference between CPU timestamp counters (rdtsc) ... OK
[*] Checking the difference between CPU timestamp counters (rdtsc) forcing VM ex
it ... traced!
[*] Checking hypervisor bit in cpuid feature bits ... traced!
[*] Checking cpuid hypervisor vendor for known VM vendors ... traced!

[-] Generic reverse turing tests
[*] Checking mouse presence ... OK
[*] Checking mouse movement ... traced!
[*] Checking mouse speed ... traced!
[*] Checking mouse click activity ... traced!
[*] Checking mouse double click activity ... traced!
[*] Checking dialog confirmation ... traced!
[*] Checking plausible dialog confirmation ... traced!

[-] Generic sandbox detection
[*] Checking username ... OK
[*] Checking file path ... OK
[*] Checking common sample names in drives root ... OK
[*] Checking if disk size <= 60GB via DeviceIoControl() ... OK
[*] Checking if disk size <= 60GB via GetDiskFreeSpaceExA() ... traced!
[*] Checking if Sleep() is patched using GetTickCount() ... OK
[*] Checking if NumberOfProcessors is < 2 via PEB access ... traced!
[*] Checking if NumberOfProcessors is < 2 via GetSystemInfo() ... traced!
[*] Checking if pysical memory is < 1Gb ... traced!
[*] Checking operating system uptime using GetTickCount() ... OK
[*] Checking if operating system IsNativeVhdBoot() ... OK

[-] Hooks detection
[*] Checking function ShellExecuteExW method 1 ... OK
[*] Checking function CreateProcessA method 1 ... OK

[-] Sandboxie detection
[*] Using GetModuleHandle(sbiedll.dll) ... OK

[-] Wine detection
[*] Using GetProcAddress(wine_get_unix_file_name) from kernel32.dll ... OK
[*] Reg key (HKCU\SOFTWARE\Wine) ... OK

[-] VirtualBox detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] Reg key (HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions) ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "VideoBiosVersion") ... OK
```

```
[*] Using GetProcAddress(wine_get_unix_file_name) from kernel32.dll ... OK
[*] Reg key (HKCU\SOFTWARE\Wine) ... OK

[-] VirtualBox detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] Reg key (HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions) ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "VideoBiosVersion") ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\DSDT\VBOX__) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\FADT\VBOX__) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\RSDT\VBOX__) ... OK
[*] Reg key (HKLM\SYSTEM\ControlSet001\Services\VBox*) ... OK
[*] Reg key (HKLM\HARDWARE\DESCRIPTION\System "SystemBiosDate") ... OK
[*] Driver files in C:\WINDOWS\system32\drivers\VBox* ... OK
[*] Additional system files ... OK
[*] Looking for a MAC address starting with 08:00:27 ... OK
[*] Looking for pseudo devices ... OK
[*] Looking for VBoxTray windows ... OK
[*] Looking for VBox network share ... OK
[*] Looking for VBox processes (vboxservice.exe, vboxtray.exe) ... OK
[*] Looking for VBox devices using WMI ... OK

[-] VMware detection
[*] Scsi port 0,1,2 ->bus->target id->logical unit id-> 0 identifier ... traced!

[*] Reg key (HKLM\SOFTWARE\VMware, Inc.\VMware Tools) ... traced!
[*] Looking for C:\WINDOWS\system32\drivers\vmmouse.sys ... traced!
[*] Looking for C:\WINDOWS\system32\drivers\vmhgfs.sys ... traced!
[*] Looking for a MAC address starting with 00:05:69, 00:0C:29, 00:1C:14 or 00:5
0:56 ... traced!
[*] Looking for network adapter name ... OK
[*] Looking for pseudo devices ... traced!
[*] Looking for VMware serial number ... traced!

[-] Qemu detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] cpuid CPU brand string 'QEMU Virtual CPU' ... OK

[-] Bochs detection
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] cpuid AMD wrong value for processor name ... OK
[*] cpuid Intel wrong value for processor name ... OK

[-] Cuckoo detection
[*] Looking in the TLS for the hooks information structure ... OK

[-] Pafish has finished analyzing the system, check the log file for more inform
ation
    and visit the project's site:

    https://github.com/a0rtega/pafish
```

Figure B.21: Pafish detecting artefacts before mitigation

In the aforementioned figure, we can see that Pafish has detected all the artefacts on hardened machine using different techniques described in chapter 3. The following figure will now show the results after mitigating these artefacts.



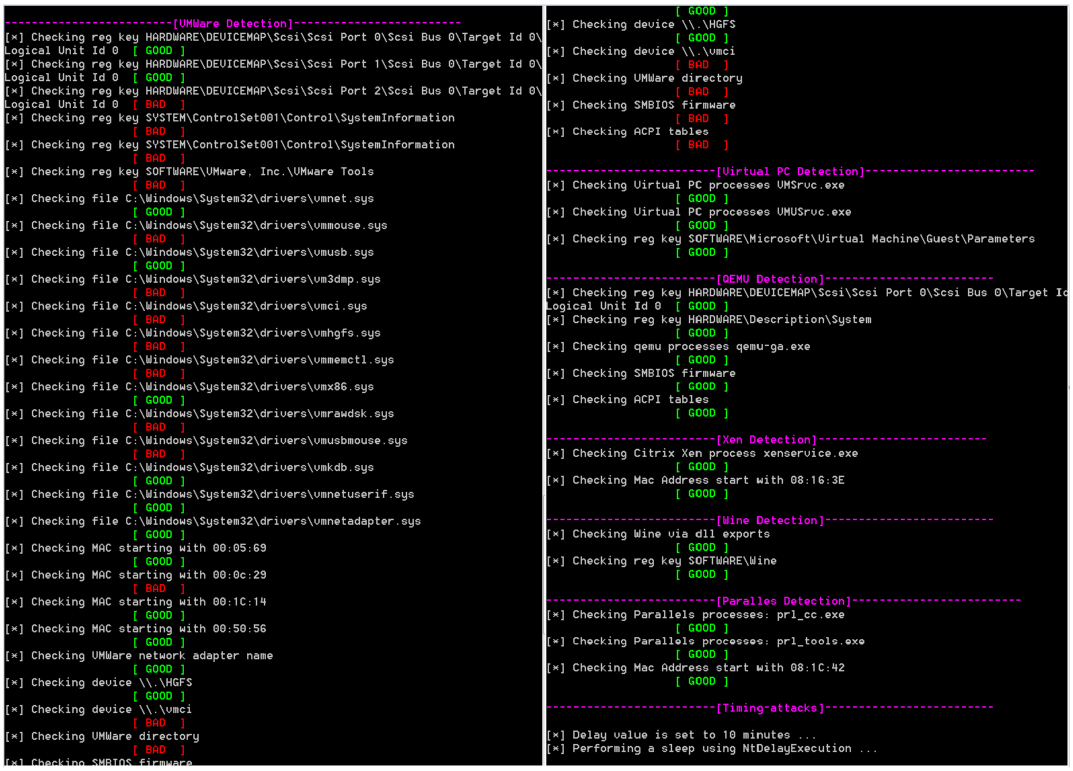
```
* Pafish (Paranoid Fish) *

[-] Windows version: 6.1 build 7601
[-] Running in WoW64: False
[-] CPU: GenuineIntel
    Hypervisor: Microsoft Hv
    CPU brand: Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK
[*] Using BeingDebugged via PEB access ... OK

[-] CPU information based detections
[*] Checking the difference between CPU timestamp counters (rdtsc) ... OK
[*] Checking the difference between CPU timestamp counters (rdtsc) forcing VM ex
it ... traced!
[*] Checking hypervisor bit in cpuid feature bits ... traced!
[*] Checking cpuid hypervisor vendor for known VM vendors ... traced!

[-] Generic reverse turing tests
[*] Checking mouse presence ... OK
[*] Checking mouse movement ... OK
[*] Checking mouse speed ... OK
[*] Checking mouse click activity ... OK
[*] Checking mouse double click activity ... OK
[*] Checking dialog confirmation ... OK
[*] Checking plausible dialog confirmation ... OK

[-] Generic sandbox detection
[*] Checking username ... OK
[*] Checking file path ... OK
[*] Checking common sample names in drives root ... OK
[*] Checking if disk size <= 60GB via DeviceIoControl() ... OK
[*] Checking if disk size <= 60GB via GetDiskFreeSpaceExA() ... OK
[*] Checking if Sleep() is patched using GetTickCount() ... OK
[*] Checking if NumberOfProcessors is < 2 via PEB access ... OK
[*] Checking if NumberOfProcessors is < 2 via GetSystemInfo() ... OK
[*] Checking if pysical memory is < 1Gb ... OK
[*] Checking operating system uptime using GetTickCount() ... OK
[*] Checking if operating system IsNativeVhdBoot() ... OK

[-] Hooks detection
[*] Checking function ShellExecuteExW method 1 ... OK
[*] Checking function CreateProcessA method 1 ... OK

[-] Sandboxie detection
[*] Using GetModuleHandle(sbiedll.dll) ... OK

[-] Wine detection
[*] Using GetProcAddress(wine_get_unix_file_name) from kernel32.dll ... OK
[*] Reg key (HKCU\SOFTWARE\Wine) ... OK

[-] VirtualBox detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] Reg key (HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions) ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "VideoBiosVersion") ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\DSDT\VBOX__) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\FADT\VBOX__) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\RSDT\VBOX__) ... OK
[*] Reg key (HKLM\SYSTEM\ControlSet001\Services\VBox*) ... OK
[*] Reg key (HKLM\HARDWARE\DESCRIPTION\System "SystemBiosDate") ... OK
[*] Driver files in C:\WINDOWS\system32\drivers\VBox* ... OK
[*] Additional system files ... OK
[*] Looking for a MAC address starting with 08:00:27 ... OK
[*] Looking for pseudo devices ... OK
[*] Looking for VBoxTray windows ... OK
[*] Looking for VBox network share ... OK
[*] Looking for VBox processes (vboxservice.exe, vboxtray.exe) ... OK
[*] Looking for VBox devices using WMI ... OK

[-] VMware detection
[*] Scsi port 0,1,2 ->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\SOFTWARE\VMware, Inc.\VMware Tools) ... OK
[*] Looking for C:\WINDOWS\system32\drivers\vmmouse.sys ... OK
[*] Looking for C:\WINDOWS\system32\drivers\vmhgfs.sys ... OK
[*] Looking for a MAC address starting with 00:05:69, 00:0C:29, 00:1C:14 or 00:5
0:56 ... OK
[*] Looking for network adapter name ... OK
[*] Looking for pseudo devices ... OK
[*] Looking for VMware serial number ... OK

[-] Qemu detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] cpuid CPU brand string 'QEMU Virtual CPU' ... OK

[-] Bochs detection
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] cpuid AMD wrong value for processor name ... OK
[*] cpuid Intel wrong value for processor name ... OK

[-] Cuckoo detection
```

```
[-] Cuckoo detection
[*] Looking in the TLS for the hooks information structure ... OK

[-] Pafish has finished analyzing the system, check the log file for more inform
ation
    and visit the project's site:

    https://github.com/a0rtega/pafish
```

Figure B.22: Pafish detecting artefacts after mitigation

Al-Khaser v0.81 is another tool and a proof of concept, which performs different malware tricks in order to see if the system can be flagged as a sandbox environment or not. The source code and the artifacts it looks for, can be found here [73].

This tool uses a bunch of malware techniques, which are used by the malware author in the wild to detect whether a malware is running in the virtual environment, to name some of the techniques that it uses are anti-debugging, emulation, sandbox detection, and Anti-virtualization. However, among all the detection, for our project the detections that we are interested in are the anti-vm detection, and anti sandboxing detection. Since this tools produces a huge report and it would not be feasible to include all the results that it generates. the group will show the result, which are deemed important to the project. Furthermore, it was also not possible to remove all the artifacts, that Alkhaser found on the machine, for instance it was difficult for the group to remove the fingerprinting of CPU voltage.

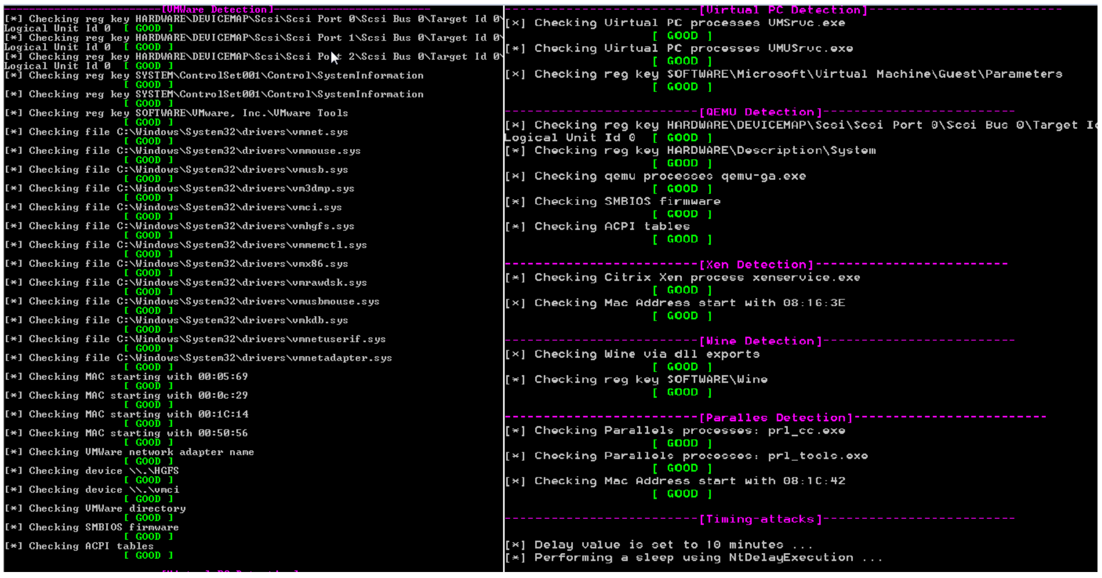


```
-------------------------[UMWare Detection]-------------------------
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\
Logical Unit Id 0  [ GOOD ]
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 1\Scsi Bus 0\Target Id 0\
Logical Unit Id 0  [ GOOD ]
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 2\Scsi Bus 0\Target Id 0\
Logical Unit Id 0  [ BAD  ]
[*] Checking reg key SYSTEM\ControlSet001\Control\SystemInformation
                   [ BAD  ]
[*] Checking reg key SYSTEM\ControlSet001\Control\SystemInformation
                   [ BAD  ]
[*] Checking reg key SOFTWARE\UMware, Inc.\UMware Tools
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmnet.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmmouse.sys
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmusb.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vm3dmp.sys
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmci.sys
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmhgfs.sys
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmmemctl.sys
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmx86.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmrawdsk.sys
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmusbmouse.sys
                   [ BAD  ]
[*] Checking file C:\Windows\System32\drivers\vmkdb.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmnetuserif.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmnetadapter.sys
                   [ GOOD ]
[*] Checking MAC starting with 00:05:69
                   [ GOOD ]
[*] Checking MAC starting with 00:0c:29
                   [ BAD  ]
[*] Checking MAC starting with 00:1C:14
                   [ GOOD ]
[*] Checking MAC starting with 00:50:56
                   [ GOOD ]
[*] Checking UMWare network adapter name
                   [ GOOD ]
[*] Checking device \\.\HGFS
                   [ GOOD ]
[*] Checking device \\.\vmci
                   [ BAD  ]
[*] Checking UMWare directory
                   [ BAD  ]
[*] Checking SMBIOS firmware
```

```
                   [ GOOD ]
[*] Checking device \\.\HGFS
                   [ GOOD ]
[*] Checking device \\.\vmci
                   [ BAD  ]
[*] Checking UMWare directory
                   [ BAD  ]
[*] Checking SMBIOS firmware
                   [ BAD  ]
[*] Checking ACPI tables
                   [ BAD  ]

-------------------------[Virtual PC Detection]-------------------------
[*] Checking Virtual PC processes UMSrvc.exe
                   [ GOOD ]
[*] Checking Virtual PC processes UMUSrvc.exe
                   [ GOOD ]
[*] Checking reg key SOFTWARE\Microsoft\Virtual Machine\Guest\Parameters
                   [ GOOD ]

-------------------------[QEMU Detection]-------------------------
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target I
Logical Unit Id 0  [ GOOD ]
[*] Checking reg key HARDWARE\Description\System
                   [ GOOD ]
[*] Checking qemu processes qemu-ga.exe
                   [ GOOD ]
[*] Checking SMBIOS firmware
                   [ GOOD ]
[*] Checking ACPI tables
                   [ GOOD ]

-------------------------[Xen Detection]-------------------------
[*] Checking Citrix Xen process xenservice.exe
                   [ GOOD ]
[*] Checking Mac Address start with 08:16:3E
                   [ GOOD ]

-------------------------[Wine Detection]-------------------------
[*] Checking Wine via dll exports
                   [ GOOD ]
[*] Checking reg key SOFTWARE\Wine
                   [ GOOD ]

-------------------------[Paralles Detection]-------------------------
[*] Checking Parallels processes: prl_cc.exe
                   [ GOOD ]
[*] Checking Parallels processes: prl_tools.exe
                   [ GOOD ]
[*] Checking Mac Address start with 08:1C:42
                   [ GOOD ]

-------------------------[Timing-attacks]-------------------------

[*] Delay value is set to 10 minutes ...
[*] Performing a sleep using NtDelayExecution ...
```

Figure B.23: AlKhaser detecting artefacts before mitigation

Every single the artifacts that are found by Alkhaser, is then mitigated or removed by the group. Hence following is the results of Alkhaser after mitigation of the artifacts.



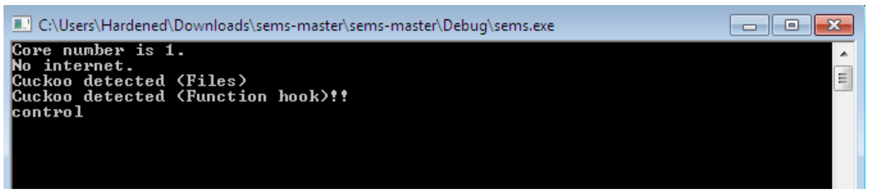
```
--------------------------[VMWare Detection]--------------------------
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\
Logical Unit Id 0  [ GOOD ]
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 1\Scsi Bus 0\Target Id 0\
Logical Unit Id 0  [ GOOD ]
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 2\Scsi Bus 0\Target Id 0\
Logical Unit Id 0  [ GOOD ]
[*] Checking reg key SYSTEM\ControlSet001\Control\SystemInformation
                   [ GOOD ]
[*] Checking reg key SYSTEM\ControlSet001\Control\SystemInformation
                   [ GOOD ]
[*] Checking reg key SOFTWARE\VMware, Inc.\VMware Tools
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmnet.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmmouse.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmusb.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vm3dmp.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmci.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmhgfs.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmmemctl.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmx86.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmrawdsk.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmusbmouse.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmkdb.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmnetuserif.sys
                   [ GOOD ]
[*] Checking file C:\Windows\System32\drivers\vmnetadapter.sys
                   [ GOOD ]
[*] Checking MAC starting with 00:05:69
                   [ GOOD ]
[*] Checking MAC starting with 00:0c:29
                   [ GOOD ]
[*] Checking MAC starting with 00:1C:14
                   [ GOOD ]
[*] Checking MAC starting with 00:50:56
                   [ GOOD ]
[*] Checking VMWare network adapter name
                   [ GOOD ]
[*] Checking device \\.\HGFS
                   [ GOOD ]
[*] Checking device \\.\vmci
                   [ GOOD ]
[*] Checking VMWare directory
                   [ GOOD ]
[*] Checking SMBIOS firmware
                   [ GOOD ]
[*] Checking ACPI tables
                   [ GOOD ]
```

```
-------------------------[Virtual PC Detection]-------------------------
[*] Checking Virtual PC processes VMSrvc.exe
                   [ GOOD ]
[*] Checking Virtual PC processes VMUSrvc.exe
                   [ GOOD ]
[*] Checking reg key SOFTWARE\Microsoft\Virtual Machine\Guest\Parameters
                   [ GOOD ]

-------------------------[QEMU Detection]-------------------------
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target I
Logical Unit Id 0  [ GOOD ]
[*] Checking reg key HARDWARE\Description\System
                   [ GOOD ]
[*] Checking qemu processes qemu-ga.exe
                   [ GOOD ]
[*] Checking SMBIOS firmware
                   [ GOOD ]
[*] Checking ACPI tables
                   [ GOOD ]

-------------------------[Xen Detection]-------------------------
[*] Checking Citrix Xen process xenservice.exe
                   [ GOOD ]
[*] Checking Mac Address start with 08:16:3E
                   [ GOOD ]

-------------------------[Wine Detection]-------------------------
[*] Checking Wine via dll exports
                   [ GOOD ]
[*] Checking reg key SOFTWARE\Wine
                   [ GOOD ]

-------------------------[Paralles Detection]-------------------------
[*] Checking Parallels processes: prl_cc.exe
                   [ GOOD ]
[*] Checking Parallels processes: prl_tools.exe
                   [ GOOD ]
[*] Checking Mac Address start with 08:1C:42
                   [ GOOD ]

-------------------------[Timing-attacks]-------------------------

[*] Delay value is set to 10 minutes ...
[*] Performing a sleep using NtDelayExecution ...
```

Figure B.24: AlKhaser detecting artefacts after mitigation

SEMS: Anti-Sandbox and Anti-Virtual Machine Tool

In order to evade analysis, modern malware authors supply anti-analysis techniques to the malwares. SEMS is a tool which is also like the other above mentioned tools uses these techniques to find out, and detect if it is running in a controlled environment. In our project, since we have already mitigated most of the artifacts, this tool is used to confirm and verify those changes. The tool can be found here [74]. Following are the results of the tool, which can only detect that the agent of cuckoo is running and can detect cuckoo or the sandbox, however, it is not able to detect any sort of virtual machine environment.



```
C:\Users\Hardened\Downloads\sems-master\sems-master\Debug\sems.exe
Core number is 1.
No internet.
Cuckoo detected (Files)
Cuckoo detected (Function hook)!!
control
```

Figure B.25: SEMS detecting artefacts after mitigation

B.5 Capture an image at FOG

With the above mentioned tools and configurations for the physical machines. The physical machines are now ready to be imaged on the FOG imaging server, which will then be used by the cuckoo host to restart these machines to a clean slate after the analysis. The next step is to register the images of the above machines at FOG imaging server.

Furthermore, before registering the image at FOG, we will configure the DHCP option 066, for the IP of the TFTP, and option 067 for the file on the default gateway i.e Cisco Switch, from which the physical machine will boot through network. This is done as follows [111].

```
ip dhcp pool VLAN10
network 192.168.10.0 255.255.255.0
dns-server 192.168.10.1
option 66 ip 192.168.10.101
option 67 ascii undionly.kpxe
default-router 192.168.10.1
```

B.5.1 Registering the machine at FOG

Now that the FOG imaging server, and the physical machines are ready. It is time to capture the images from the physical machines. We can use a single image for all three of the machines, which can be used to start the machine to a clean slate from, after the analysis.

However, in our project we are using all three machines with different configurations and we want to see the impact of malware on each of these machines. Hence we will capture images from all three physical machines. Following we will describe the process of capturing image from one machine, The same process is repeated for the other two as well. To capture the image, First we want to boot the machine from the network, and register it in the FOG [112].

To boot the physical machine from the network, initially the group changed the boot order of the QEMU image, by right clicking on the node and then clicking edit. where we then changed the boot order in QEMU custom options as shown below [113].

```
-machine type=pc,accel=kvm -cpu host,+pcid,+kvm_pv_unhalt,+kvm_pv_eoi,
    hv_spinlocks=0x1fff,hv_vapic,hv_time,hv_reset,hv_vpindex,hv_runtime,
    hv_relaxed,hv_synic,hv_stimer -vga std -usbdevice tablet -boot order=
    nd #bootorder
-drive file=/opt/qemu/share/qemu/virtio-win-drivers.img,index=1,if=floppy,
    readonly
```

As QEMU uses **-boot options** to specify the boot order of the drives, for instance

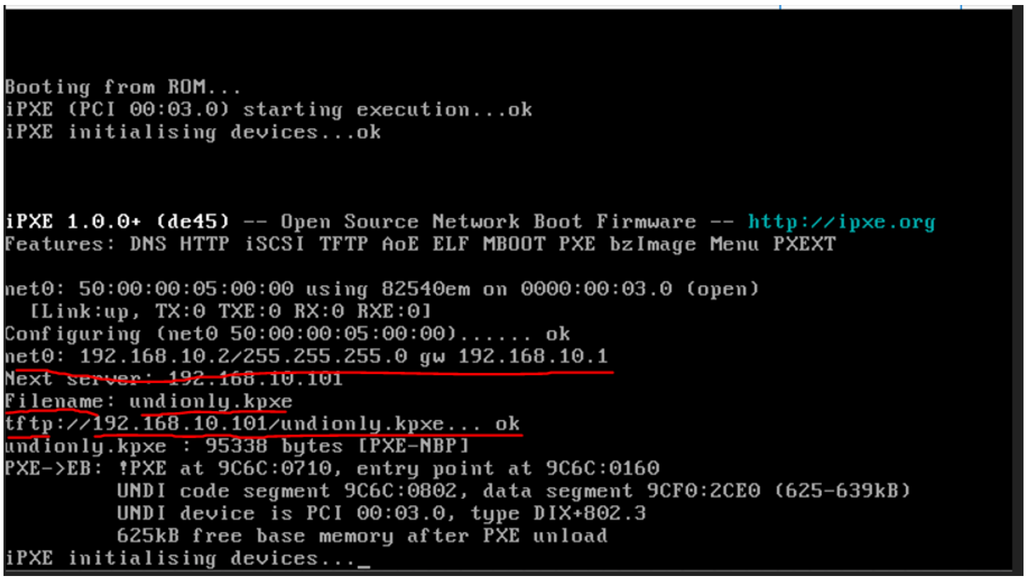
-boot order=ndc first tries to boot from the network, then from the first CD-ROM and finally from the harddisk. Here we changed it to the network and then harddisk. However, as soon as we restarted the machine, it would change the boot order again to the default boot order which was **-boot order=cd**. Hence we wanted to permanently boot the machine t from the network as the first option.

In Eve-NG every node has a template file that will specify the startup parameters, we changed the boot order in the parameters of the windows machines to **-boot order=nc** [114].

```
root@eve-ng:/opt/unetlab/html/templates/intel# sudo nano win.yml
type: qemu
description: Windows
name: Win
cpulimit: 1
icon: Desktop.png
cpu: 1
ram: 4096
ethernet: 1
console: vnc
shutdown: 1
qemu_arch: x86_64
qemu_version: 4.1.0
qemu_options: -machine type=pc,accel=kvm -cpu host,+pcid,+kvm_pv_unhalt,+
    kvm_pv_eoi,hv_spinlocks=0x1fff,hv_vapic,hv_time,hv_reset,hv_vpindex,
    hv_runtime,hv_relaxed,hv_synic,hv_stimer
   -vga std -usbdevice tablet -boot order=nc -drive file=/opt/qemu/share/
    qemu/virtio-win-drivers.img,index=1,if=floppy,readonly
```

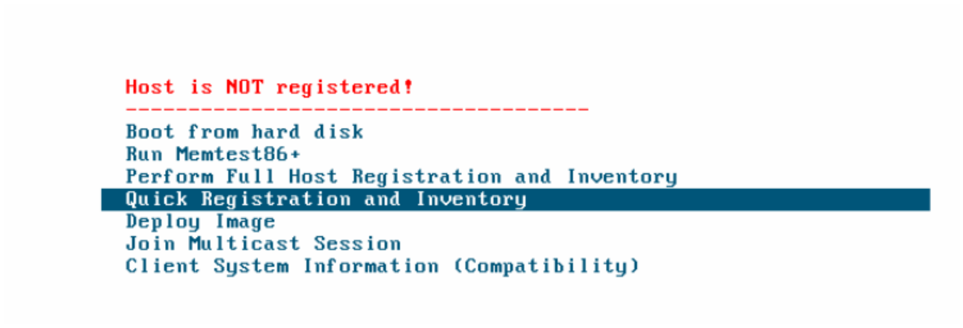
Through this change, the physical machine will now first look for a network boot task, if FOG has a task for it, it will boot from the network otherwise it will start from the harddisk. Now that the machine can start from network its time to turn it on, and register the machine at FOG server.

In fig. B.26 we can see physical machine starting from the network. Once it starts from network we will be greeted with a FOG menu see fig. B.27, though which we can quick register the desire machine at the FOG.



```
Booting from ROM...
iPXE (PCI 00:03.0) starting execution...ok
iPXE initialising devices...ok


iPXE 1.0.0+ (de45) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS HTTP iSCSI TFTP AoE ELF MBOOT PXE bzImage Menu PXEXT

net0: 50:00:00:05:00:00 using 82540em on 0000:00:03.0 (open)
  [Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (net0 50:00:00:05:00:00)...... ok
net0: 192.168.10.2/255.255.255.0 gw 192.168.10.1
Next server: 192.168.10.101
Filename: undionly.kpxe
tftp://192.168.10.101/undionly.kpxe... ok
undionly.kpxe : 95338 bytes [PXE-NBP]
PXE->EB: !PXE at 9C6C:0710, entry point at 9C6C:0160
         UNDI code segment 9C6C:0802, data segment 9CF0:2CE0 (625-639kB)
         UNDI device is PCI 00:03.0, type DIX+802.3
         625kB free base memory after PXE unload
iPXE initialising devices..._
```

Figure B.26: Physical machine booting from Network



```
Host is NOT registered!
-------------------------------------
Boot from hard disk
Run Memtest86+
Perform Full Host Registration and Inventory
Quick Registration and Inventory
Deploy Image
Join Multicast Session
Client System Information (Compatibility)
```

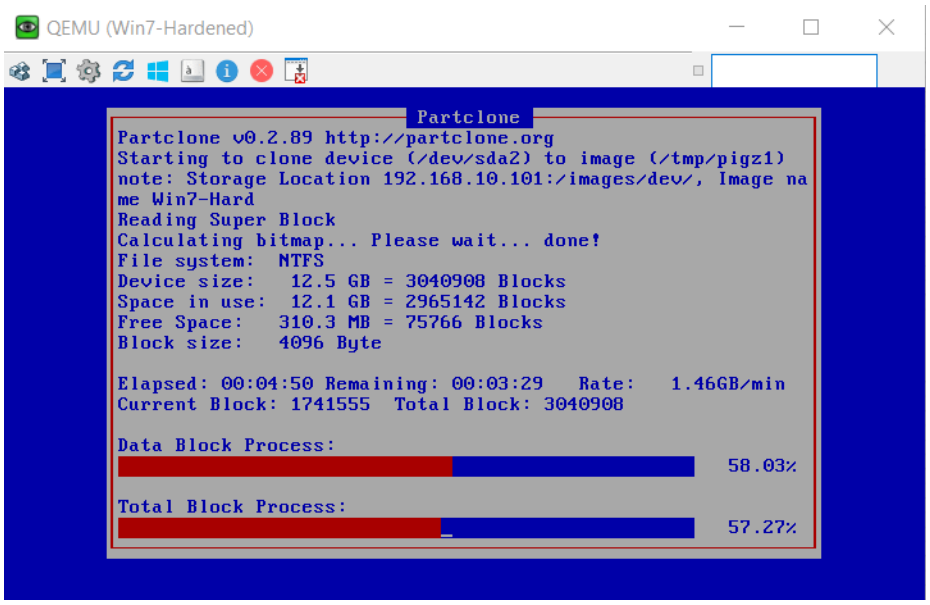
Figure B.27: FOG boot menu

We can see that the physical machine is not registered from the above menu. This tells us that the physical machine is not known by FOG Server. By choosing the quick registration and inventory we can register the physical machine in the FOG. To capture the image, we must first register the machine into the FOG.

Once the machine is registered, we will rename the image to win7-Hardened and assigned it to a host. The process is repeated for all the three Windows 7 machines [112].

Once the image is assigned to a host named Win7-Hardened, Next we will create a new task in the FOG server for this machine to capture an image from it, while it starts from the network.In the following fig. B.28, an image captured from a

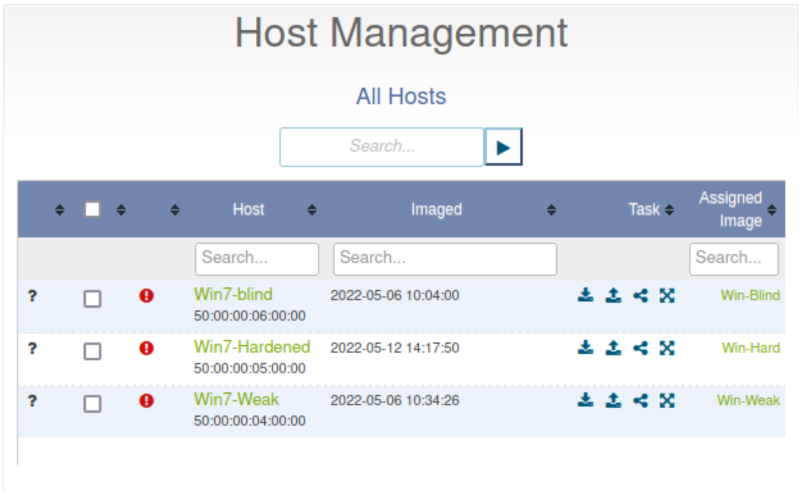
Win7-Hardened machine can be seen.



```
Partclone
Partclone v0.2.89 http://partclone.org
Starting to clone device (/dev/sda2) to image (/tmp/pigz1)
note: Storage Location 192.168.10.101:/images/dev/, Image na
me Win7-Hard
Reading Super Block
Calculating bitmap... Please wait... done!
File system:  NTFS
Device size:   12.5 GB = 3040908 Blocks
Space in use:  12.1 GB = 2965142 Blocks
Free Space:   310.3 MB = 75766 Blocks
Block size:   4096 Byte

Elapsed: 00:04:50 Remaining: 00:03:29   Rate:   1.46GB/min
Current Block: 1741555  Total Block: 3040908

Data Block Process:
                                                   58.03%

Total Block Process:
                                                   57.27%
```

Figure B.28: Capturing image from Win7-Hardened

After Capturing the image from all the three machine on FOG server, we are ready to send malware to each machine by spinning cuckoo and the cuckoo web server. Below fig. B.29 shows three images captured from all three machine in the topology.



Host Management

All Hosts

			Host	Imaged	Task	Assigned Image
?	☐	❗	Win7-blind 50:00:00:06:00:00	2022-05-06 10:04:00		Win-Blind
?	☐	❗	Win7-Hardened 50:00:00:05:00:00	2022-05-12 14:17:50		Win-Hard
?	☐	❗	Win7-Weak 50:00:00:04:00:00	2022-05-06 10:34:26		Win-Weak

Figure B.29: Captured images in FOG from all three machine

At this point, all three machines are ready to be used as cuckoo guests. Next, we will install and configure Dionaea honeypot to see, if it can capture the malware interacting in the local network.

B.6 Dionaea - Setting up a Honeypot

In this section of the report, we will go through the installation and the configuration of a honeypot application named Dionaea. Dionaea is usually used to capture the malware that interacts with its exposed network services. It is recommended to be used on a public IP, however, it can also be used in a local network for testing purposes.

In our project, we ran this honeypot on a node in the EVE-NG topology, to see if the malware that runs in the cuckoo guest machine does interact with the honeypot. For this purpose, we installed Dionaea in both VLANs, i.e VLAN10 and VLAN20. the source code and the protocols, which Dionaea offers can be found at [115].

The requirements for installing Dionaea honeypot are Ubuntu 18.0 LTS, and Python Runtime 3.9 as recommended [69]. The process of installing and converting the Ubuntu machine is described in appendix B.2.2. This installation can also be done by uploading the .ISO file of the Ubuntu to EVE-NG [95].

Once the image is uploaded, this Ubuntu machine which is named as Dionaea-10, is then connected to the Internet by attaching it to the Cloud in fig. 4.1.

Once the machine is up and running, the following steps were taken in order to install Dionaea.

```
Sudo apt update && upgrade -y
git clone https://github.com/DinoTools/dionaea.git
cd dionaea
```

Hereafter the required dependencies were install by.

```
sudo apt-get install \
    build-essential \
    cmake \
    check \
    cython3 \
    libcurl4-openssl-dev \
    libemu-dev \
    libev-dev \
    libglib2.0-dev \
    libloudmouth1-dev \
    libnetfilter-queue-dev \
    libnl-3-dev \
    libpcap-dev \
    libssl-dev \
    libtool \
    libudns-dev \
    python3 \
    python3-dev \
    python3-bson \
    python3-yaml \
    python3-boto3 \
    fonts-liberation
```

Once all the dependencies were installed, we then created a build directory and run cmake to setup the build process.

```
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX:PATH=/opt/dionaea ..
```

Lastly, we run make to build and run make install to install the honeypot.

```
make
sudo make install
```

The honeypot is now installed and can be found in the directory

```
/opt/dionaea
```

Now that honeypot is installed we wills start the honeypot by

```
/opt/dionaea/bin/dionaea
```

At this point, the honeypot is up and running. Now we will move this machine to VLAN10 and will assign an IP to it through DHCP of the Cisco Switch.

Furthermore, we will leave the configured service as default, since we are not checking any specific services, and we want to see if any of the services are contacted by the malware. if any of the services is contacted, we can see it in the Dionaea log file.

Additionally, we wanted to run the Dionaea process to run as a system service in the background as this machine will be stopped and started while doing the malware analysis. For this purpose, We created a file in [70].

```
sudo nano /etc/systemd/system/dionaea.service
```

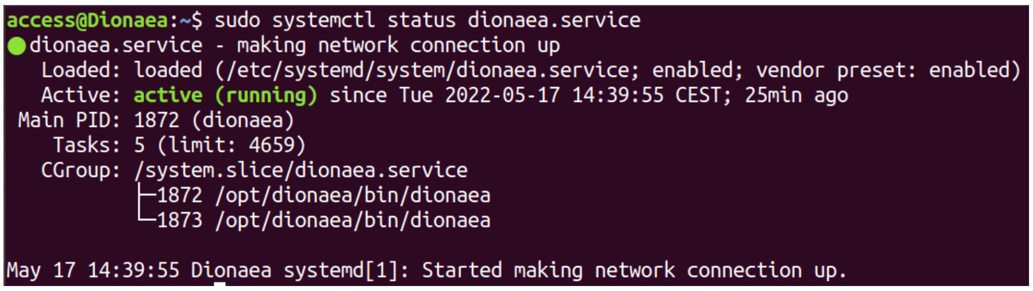
and then we pasted the dionaea binary in the file in order to make it easy to manage the dionaea service.

```
[Unit]
Description = making network connection up
After = network.target
[Service]
ExecStart = /opt/dionaea/bin/dionaea
[Install]
WantedBy = multi-user.target
```

Now we will enable the service by and we can check that the service active and running.

```
sudo systemctl enable dionaea.service #Enable the dionaea service at the
    boot
sudo systemctl start dionaea.service  # Starts the dionaea service
sudo systemctl stop dionaea.service # Stop the dionaea service
sudo systemctl status dionaea.service # To check the status of the service
```

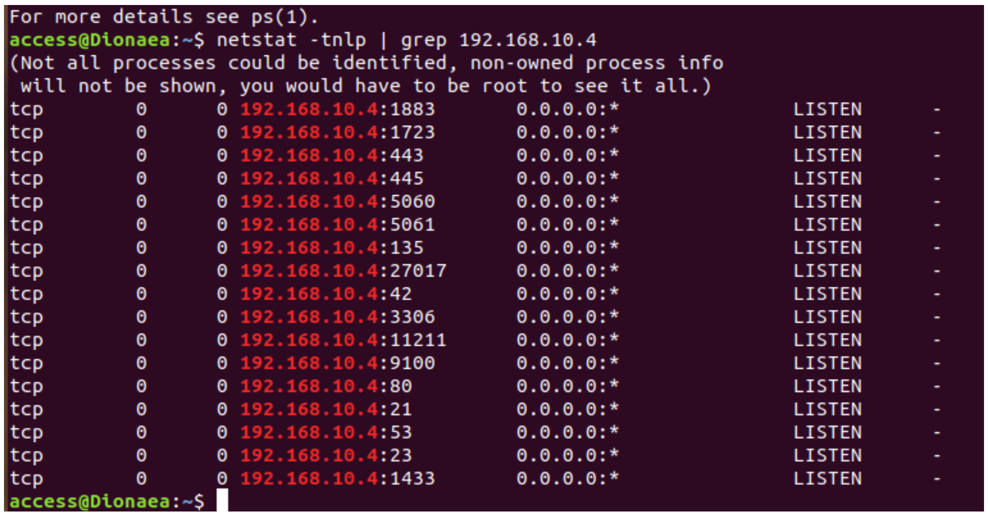
let's now confirm that the service is up and running. see fig. B.30



```
access@Dionaea:~$ sudo systemctl status dionaea.service
● dionaea.service - making network connection up
   Loaded: loaded (/etc/systemd/system/dionaea.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2022-05-17 14:39:55 CEST; 25min ago
 Main PID: 1872 (dionaea)
    Tasks: 5 (limit: 4659)
   CGroup: /system.slice/dionaea.service
           ├─1872 /opt/dionaea/bin/dionaea
           └─1873 /opt/dionaea/bin/dionaea

May 17 14:39:55 Dionaea systemd[1]: Started making network connection up.
```

Figure B.30: Dionaea honeypot service up and running

To check the protocol and service that it is offering. see fig. B.31



```
For more details see ps(1).
access@Dionaea:~$ netstat -tnlp | grep 192.168.10.4
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 192.168.10.4:1883       0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:1723       0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:443        0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:445        0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:5060       0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:5061       0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:135        0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:27017      0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:42         0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:3306       0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:11211      0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:9100       0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:80         0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:21         0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:53         0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:23         0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.10.4:1433       0.0.0.0:*               LISTEN      -
access@Dionaea:~$
```

Figure B.31: Ports on which Dionaea is listening

In the technical setup we have now configured all the nodes in the Topology fig. 4.1. The platform is now ready to be used and the malware can be sent to analysed in the topology.

Appendix C

List of malware binaries

Below, a table can be found containing a list of all the malware binaries we have used during this project. They have been split into colors based on their category, to allow for an easier overview.

Dionaea1 and *Dionaea3* are instances of the WannaCry malware, while *Dionaea2* is a coin miner malware, as can be seen by looking up their hashes on platforms such as VirusTotal.

Table C.1: A list of all malware binaries tested throughout the course of this project

Malware alias	**File name / hash**
Botnet1	Conficker7.exe
Botnet2	IRCBot5.exe
Botnet3	Cutwail4.exe
Botnet4	Zeus6.exe
Botnet5	Zeus1.exe
Trojan1	LoadMoney1.exe
Trojan2	Tapaoux.exe
Trojan3	Sinowal.exe
Trojan4	Sality.exe
Trojan5	Hupigon.exe
Various1	Keylogger-Ardamax.exe
Various2	KRLocker/CrimClient.exe
Various3	SkyWiper-A.Flame - <hash>advnetcfg.ocx
Various4	CryptoLocker 2014 - 1002.exe
Various5	Artemis - install.exe
Various6	njRAT.exe
Various7	PlugX.exe
Various8	Variant Kazi - 21.exe
Various9	ZeroAccess - zeroaccess_porn.avi.exe
Various10	Carberp/AAA.exe
Various11	Win32.InfoStealer.Dexter - win33.exe
Dionaea1	017f63d0be693e53bc5b8edd426cfbd1
Dionaea2	fcb6b0f95853dfda72d5535a424b3a29
Dionaea3	02c5f1515bf42798728fac17bfe1e4c1

Appendix D

Malware reports and source code

The source code portions of this project, any scripts and so on, can be found on the following link, as a GitHub repository: `https://github.com/n3xtd00rpanda/NetworkSandboxScripts`

A link to most of the malware reports and binaries — some have been stripped due to Google Drive's auto-scan feature — can be found here:

`https://mega.nz/folder/wttGlRZa#e3xUPxTCzDW6ZCrYXgH8dQ`