AALBORG UNIVERSITY

Network analysis system for self-propagating malware

Supervised by Marios Anagnostopoulos & Peyman Pahlivani

Master's Thesis

Aalborg University
Electronics and IT

Preface

Aalborg University, June 1, 2023

Mohamed Msaad
mmsaad18@student.aau.dk

David Holm Audran
daudra21@student.aau.dk

Electronics and IT
Aalborg University
http://www.aau.dk

Title:
Network analysis system for self-propagating malware

Theme:
Master's Thesis

Project Period:
Spring Semester 2023

Project Group:

Participant(s):
Mohamed Msaad
David Holm Audran

Supervisor(s):
Marios Anagnostopoulos
Peyman Pahlevani

Page Numbers: 102

Date of Completion:
June 1, 2023

Abstract:

Malware continues to pose a threat to computer systems worldwide. Some come equipped with worm capabilities, meaning they can self-propagate from one system to another without human interaction. Moreover, the evolution of malware to being form-changing makes it increasingly difficult for traditional detection techniques to effectively identify and mitigate those threats. Furthermore, existing sandboxing techniques must be improved when studying the network behavior of self-propagating malware. In this work, we present the integration of SPM analysis into the existing CAPEv2 Sandbox and enable automatic SPM analysis and data gathering. Furthermore, we integrate Security Onion and all its available network monitoring and forensic tools to work alongside CAPEv2. Giving more possibilities to the malware analyst. We also provide complete documentation and recommendations to build and enhance the analysis system for physical and virtual testbeds. While offering guidance to the different use cases. Finally, we demonstrate the efficiency of the system with real-world samples.

Contents

vii

Table 1: Acronyms and abbreviations used in the report

Acronym or abbreviation	**Definition**
AUB	Aalborg University Library
SPM	Self Propagating Malware
LAN	Local Area Network
WAN	Wide Area Network
VM	Virtual Machine
PhM	Physical Machine
CSV	Comma-Separated Values
CVE	Common Vulnerabilities and Exposures
DNS	Domain Name System
CnC	Command and Control
SDN	Software Defined Network
ARP	Address Resolution Protocol
DHCP	Dynamic Host Configuration Protocol
MAC	Media Access Control
SMB	Server Message Block
TCR	Tree Connect Request
URI	Uniform Resource Identifier
NBNS	Net-Bios Name Service
UDP	User Datagram Protocol
TCP	Transport Control Protocol
ICMP	Internet Control Message Protocol
NIC	Network Interface Card
ET	Evasion Technique
GW	Gateway
SMB	Server Message Block
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
NFS	Network File System
POC	Proof of Concept
SPM	Self-propagating Malware

Chapter 1

Introduction

The COVID-19 pandemic has changed our societies and shifted most of the work online. This fast transition offered new attack surfaces to the attackers, which led to an increase in cyber-attacks, making the year 2021 the highest in 17 years[54]. The attackers take control over the target machine by installing malicious software and deceiving the victim into clicking or downloading malicious Malware. The latter poses a significant threat to the security of the Internet as in 2021 alone, a Ransomware attack caused $159.4bn loss [18]. Malware comes in different families and is classified based on its purposes, such as Spyware, Adware, and Ransomware. Their purpose can range from data compromise and DDoS attacks to the system's real corruption. Moreover, other Malware comes equipped with the worm capability. A computer worm is a subset of the Trojan malware that can self-propagate from one computer to another without human activation after infecting a system. The first infected machine is referred to as *Patient-0*. A worm typically spreads across a network through the Internet or across the Local Area Network (LAN). Self-Propagating Malware (SPM) are very dangerous and can quickly take over an entire network of n nodes. In a ransomware attack [14] caused by the WannaCry Malware, more than 300 thousands vulnerable devices in 150 countries were infected in a few days. Other cyber-attacks were carried out on the state level, where the *Stuxnet* malware caused substantial damage to the Iranian nuclear program.[15].

Cybersecurity researchers, on the other hand, are fighting back by analyzing Malware. Two analysis methods exist. Static and Dynamic analysis [28, 99]. Static analysis is based on reverse engineering the Malware's binaries to obtain the source code to understand the Malware's structure. Dynamic analysis, also known as Sandbox, is a dynamic analysis system where the Malware runs in an isolated environment and test malware or any other untrusted third-party programs [96]. It offers good isolation and prevents the Malware from escaping. However, many things could be improved by this technology. The attackers know its existence

and build their Malware with evasion techniques. With this, the Malware inspects its runtime environment and detects the existence of the Sandbox [42]. The other drawback is the standalone concept, where the Sandbox runs the Malware in a machine, a Physical-Machine (PhM) or virtual machine (VM). Typically, the Sandbox only spins one machine to run the Malware and study its behavior. Eventually, the LAN contains only one host, so the worm cannot spread to other hosts. Consequently, the worm behavior of the Malware cannot be thoroughly studied.

1.1 Motivation

In 2021, the Cybersecurity and Infrastructure Security Agency (CISA) observed ransomware attacks against 14 of the 16 U.S. critical infrastructure sectors, such as Food and Agriculture, Defense, Energy, Information Technology, and Government facilities [26]. ENISA (European Union Agency for Cybersecurity) also ranks malware and ransomware as two of the eight threats against the EU. Stating that after the COVID-19 pandemic, malware detection is rising again caused by an increase in the total number of malware [41]. In their white paper, SonicWall reported 465,501 never before seen malware variants by their systems in 2022, and the first time their total number of detection exceeded 450,000 [40]. With the average cost of ransomware infection being $812,960, this is a very lucrative market for cybercriminals [102]. Furthermore, the development of Malware-as-a-Service (MaaS) has permitted adversaries without programming skills to launch attacks using complex malware [72]. It makes it easier for low and nonskilled people to integrate into the cybercriminal market while reducing the risk of exposure for the top cyber criminals. Some MaaS vendors ask for a share of the ransom revenue instead of an investment upfront.

Our project is motivated by several factors. Firstly, we seek to provide a platform to help understand the propagation mechanisms used by self-propagating malware in networks. This knowledge can inform the development of improved defense strategies and enable earlier detection of such threats.

Secondly, we aim to address some of the limitations of sandboxing techniques, which rely on isolated environments to test and analyze untrusted software. This approach is often restricted to a single physical or virtual machine, which hinders the study of network behavior in the context of self-propagating malware.

Thirdly, we recognize the shortcomings of traditional signature-based malware detection systems, particularly in the face of polymorphic or other form-changing malware. Such threats alter their binary or instruction sets without modifying their behavior, rendering traditional detection techniques ineffective [32].

Finally, we note the need for dedicated platforms for analyzing the network behavior of self-propagating malware. To the best of our knowledge, no such platform exists.

1.2 Problem Formulation

Malware continues to pose a significant threat to computer systems and networks worldwide. It is crucial to have effective methods for analyzing their behavior to mitigate their potential impact, as new forms are continually being developed. Current analysis methods have limitations, especially concerning self-propagating malware. Therefore, there is a need for a dynamic analysis system that can monitor the behavior of self-propagating malware in real time.

The goal of this Master's thesis is to design and develop a dynamic analysis system to study the network behavior of self-propagating malware. This system should be able to monitor and capture network traffic generated by the malware and analyze its behavior to identify its propagation patterns and potential targets. To construct the project goals and challenges, we address the following research questions:

- *How to integrate the analysis of self-propagating malware into a dynamic analysis tool?*
- *How to build a heterogeneous environment as a testbed for self-propagating- malware?*
- *How to collect evidence from self-propagating malware?*

The proposed system will be evaluated using a range of self-propagating malware samples, and its effectiveness will be measured based on its ability to accurately identify and analyze its behavior. This work will provide a deeper understanding of the behavior of self-propagating malware and hopefully contribute to developing effective methods to detect and mitigate these threats.

1.3 Scope

The scope of the research project will be limited to developing and evaluating a dynamic malware analysis tool that addresses the limitations of existing tools regarding self-propagating malware (SPM). The objectives of the research are: Perform an in-depth analysis of the current literature and related work done in the field. This will help us obtain information critical to the project and inspire us to build our malware analysis environment. Furthermore, we offer thorough documentation about the test environment and the SPM integration measures to enhance the test-environment performance. Attached with step-by-step documentation of the entire system setup and guidance throughout the design. Ensuring an easy replication process and saving time. However, we do not cover the evasive techniques related to the Sandbox technology. But, we discuss the evasion techniques that might arise from our SPM integration.

We do not go in-depth with the malware analysis. Nevertheless, we use it to verify the functionality of the Sandbox. We use our results to confirm the authenticity of the analysis and confirm the Sandbox is well functioning. While going in-depth for the related work!

1.4 Contribution

Our contribution revolves around investigating the current literature on general malware analysis and SPM analysis. Moreover, we provide detailed documentation of the setup and test of the Sandbox and fix all the bugs encountered during the sandbox setup. We integrate SPM analysis into the current Sandbox technology without changing or decreasing the performance of the Sandbox by adding the SPM option as a plugin on demand. I.e., the user can specify SPM or a regular analysis. Additionally, we integrate Security Onion with the Sandbox to enable additional monitoring and forensics tools to the Sandbox.–For instance: Identify network scans, and create real-time alerts that can help raise instant actions in case of an incident.

1.5 Delimitation

In this section, we list the research delimitation for this master thesis. The present research is delimited in the following aspects:

- The research approach: The research employs a methodology comprising a literature review and empirical experiments. The literature review examines existing publications to gain insight into self-propagating malware and its analysis. Additionally, it helps gather information about our test methods and hints at where to find malware traces in our test results. For instance, what communication protocol and ports are used by the malware. Empirical experiments are conducted to validate and test the proposed testbeds.

- The test environment: The research is constrained to two implemented testbeds, one composed of physical machines and the other virtual. These testbeds are configured in a specific way to correspond to the research requirements. While the test environment provides a controlled system for experimentation, it does not encompass all possible hardware, network setups, or software variations.

- Specific tools: The research focuses on specific tools employed for malware analysis and detection. The primary tools utilized include the CAPEv2 sandbox, the FOG project server and client, Security Onion, and virtualization tools such as KVM-QEMU and VirtualBox.

- The type of malware studied: This research focuses solely on self-propagating malware. It aims to investigate its characteristics, behavior, and detection techniques.
- Honeypots usage: We do not use the honeypots to study the malware or to capture traffic. This is due to the redundancy introduced if the honeypot captures malware payload, as it will also be captured by the Sandbox. Moreover, besides enriching the network's heterogeneity, they are used as a physical and visual notifiers of malware connection attempts.
- The sophistication of the malware tested: This research aims to test the integration of SPM analysis within the Sandbox technology. Hence, we use less sophisticated malware with known behavior. We do not test new or advanced malware due to its risk to the AAU network infrastructure and data privacy.

1.6 Outline

The first chapter introduces the project and covers its motivations, problem formulation, scope, delimitations and contributions. In Chapter 2, we provide a detailed description of the different methods used to conduct the research. Chapter 3 highlights the background knowledge we consider is essential for this research. It provides an explanation of malware by going through its definition, different types, and evolution over the years. Malware analysis, detection techniques as well as evasion techniques are also discussed. Furthermore, we mention how malware propagation can be modeled. Chapter 4, provides an in-depth analysis of the current literature in the field. The goal being to find relevant information that we can use for our research. Chapter 5 provides the technical analysis of the project. We go through the used hardware, software and network setup, describing those different pieces and justifying their use. Additionally, we cover the implementation of the project, which is the one of the critical parts. In that section, we describe the installation and configuration of the different parts of the system. Finally, we go through the other critical part, which is the testing and validation of the implemented system. Chapter 6 discusses the research as well as future work, and concludes the report.

Chapter 2

Methodology

This chapter provides a detailed description of the methods we used to conduct our research. It helps us justify and explain the different techniques and the reasoning behind our choices. This also ensures that our research is reliable and reproducible.

2.1 Process Model

We used Process Model as a tool to keep track of the project advancement and the software development process. The project includes the following phases: Install, test, integrate, and test. We decided to follow the Agile model because of its tolerance to adaptive planning and flexibility in the continuous improvement/changes of the final product [110]. The project is divided into four main stages:

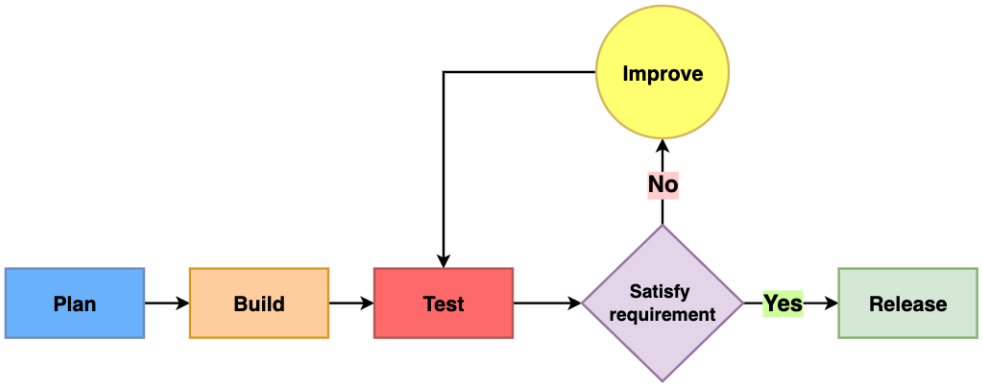


Figure 2.1: Process stages

The project aims to integrate SPM analysis within the Sandbox technology as an extra plugin. The plugin will only be active on demand. This choice ensures

no altering of the Sandbox's original mechanism. Furthermore, the process model consists of four stages: –Plan: Emphasizes highlighting the current sprint's goal. Further, the work is distributed among the group members for the build process. The planning is always related to the project's requirements. Build: Turning the plan into a product to meet the project requirement. Test: The test phase focuses on testing the built product for stability, performance, and meeting the project's requirements. Release: An automated after a successful test that meets all the requirements. Finally, the improvement phase is only needed if the product fails to meet the requirement satisfaction. Hence, it is included as an optional phase.

2.2 Literature search

A literature search helps to establish the context of the research by identifying the current state of knowledge in the field. It provides a theoretical foundation for our project and can help identify literature gaps. The literature review also includes information on the research methodology used by other authors, which allows us to select our own. Finally, it ensures that our research is original, relevant, and contributes to the existing body of knowledge. We consider the following criteria to ensure the study is an eligible source of information for our project:

- Relevance - The paper should be related to the topic or research question. It should address similar issues and problems that the project aims to explore.
- Quality - The paper should be high quality and published in a reputable journal or conference proceedings. We check if the journal is considered predatory using Beall's list [11].
- Recency - The paper should be relatively recent and up-to-date. It should reflect the latest developments and trends in the field. We choose to not include papers older than 2018.
- Clarity - The paper should be clearly written, and the research should be easy to understand and follow.

To find literature for our project's state-of-the-art, we searched for different articles with the following keywords to target specifically related research:

- Computer worms
- Self-propagating malware (SPM)
- SPM analysis.
- Malware analysis surveys

- Dynamic malware analysis
- Malware network analysis
- Bot and botnets analysis

Furthermore, we filter the search, sort, and select material if it contains one of the following pieces of information:

- The test-bed schema or topology for our project's state-of-the-art. We want to inspire ourselves from recent research and use the results and experience gathered to build our platform efficiently and effectively.
- Provides malware hashes: Malware hashes identify specific variants of malware. Collecting them to see how our system performs against known malware with known behavior.
- Provides an analysis strategy so we know where to look. This will give us hints on how to build our own analysis strategy.

2.3 Project requirements

The project focuses on integrating and automating the SPM analysis option within the Sandbox technology. Hence, we derive the project's essential pillars and use the MoSCoW prioritization method to sort and prioritize the features. The four categories of the method are *Must have*, *Should have*, *Could have*, and *Won't have*.

Must Have	- An environment with multiple hosts (n >1) on the LAN. Namely, the infected machine has other targets to scan and will trigger the worm characteristic of the malware. - Be reproducible. I.e., the sandbox can be reproduced if the physical requirements are met. - Collect network evidence from the SPM analysis for investigation
Should Have	- A heterogeneous environment. - Automated forensic tools during analysis
Could Have	- TLS for communication between the Sandbox and the testbed.
Won't Have	Be portable. I.e., the system can be ported from one machine to another like a docker container for instance.

Table 2.1: The project's requirements

We define them as follows: *Must have* are the critical features, and if not achieved, the project/tool is considered a failure. *Should have* are essential for

the project completion but not critical. *Could have* are desirable features but have a negligible impact on the project/tool if left out. They can be included if there is enough time and resources. *Won't have* features are the ones that are not a priority at all for the project and will not be included in the final product.

2.4 Threat model

We use the threat model to explain the Sandbox topology and the entire ecosystem. This helps to understand the system and define the malware's possible scenarios. Additionally, the threat model helps identify the malware's capabilities and limitations inside and outside the Sandbox ecosystem. The Sandbox technology allows the choice between a physical machine as a testbed or a Virtual Machine. Hence, we provide a threat model for both use cases.

2.5 White/Black box testing

CAPEv2 is an open-source sandbox, so we used white-box testing to find information about its structure and different parts. The documentation is well structured on the official website [57] and helps us set up the sandbox. –For instance, the installation and the configuration of the testbeds (physical and VM). WhiteBox testing helped us find the structure of the sandbox as a program and know where to change the sandbox's code to integrate the SPM analysis. Nevertheless, the official documentation must include steps critical to the sandbox's functionality. Hence, we used white-box testing to detect, find, and solve the errors and missing packages and report them in the fully detailed documentation. We also used Black-Box testing to determine the best setup for some project software. For example, we need to find out how the imaging server manages the memory compression of a physical machine. Thus, by using BlackBox testing, we discovered how to map the machine's memory for a faster restoration time.

2.6 Testing and validation

To ensure that the dynamic analysis ecosystem runs without exceptions that might interrupt the analysis and hence give incorrect data, we developed a testing strategy to ensure all the functionalities are running: - Functional Test: We test the system for all the project's requirements. - Multi-Platform Test: We conduct tests on both system types and ensure no critical errors exist. - Robustness Test: We test the system against real-life malware to test its performance and recovery from damage. - Data test: We test if the system captures the required data for the analysis.

Chapter 3

Background

In this chapter, we highlight what we consider as the essential background knowledge we need to have to fulfill our project requirements. We start by exploring what malware is and how it has been classified by researchers. We continue by mentioning different malware analysis techniques and what has been done by malware developers to counter those, known as evasion techniques. We also cover how the malware propagation can be modelled.

3.1 Malware explained

Malicious software, known as malware, is a program that performs unauthorized operations on the environment in which it executes and can sometimes cause damage [108]. They come in different forms (exe, dll, excel sheets) and serve other purposes, such as Spyware, Ransomware, Trojans, etc.

3.1.1 Malware types

Due to this broad definition, malware can serve different purposes. Thus, the researchers started dividing them into sets or families and are as follows:

Virus: Is a self-replicating malware. Usually found as an executable and spreads by copying itself to the target system. However, they are passive malware and need human intervention to spread. This can be done via USBs, media files, or networks. Depending on the level of sophistication, the virus can be polymorphic, meaning that it can modify its replicated version to avoid detection [60, 101]. Moreover, a Virus can have different purposes, from data theft and spying to complete system damage.

Worm: An active self-replicating malware that can spread over the network and infect other machines by exploitation of a vulnerability or multiple ones at different levels. The primary levels are the Operating System (OS) level, with, for

instance, the Eternal-Blue vulnerability in Windows 7 OS [CVE-2017-0144] [30, 91], and the Application level, such as the LOG4J vulnerability[CVE-2021-44228] [31, 51, 44]. The vital concept of a computer worm is its ability to spread without human intervention [20]. Additionally, the worm can serve different purposes as the virus, such as data theft, sabotage, spying, or being a part of a botnet. Moreover, the worm uses the network to spread. Hence it will leave propagation traces that can be studied and used to detect future attacks. As this paper focuses on SPM, more recent research on worm behavior will be detailed in the following chapter.

Trojan Horse: Known as Trojan, malware disguises itself as benign programs. Its mechanism is to embed the malicious program within a legit application. Once executed on a machine, it drops the malicious executable and changes features on the victim's machine to enable its persistence [50, 82]. However, the Trojan can not self-replicate and relies on the OS to activate (it activates on the next boot). Moreover, the Trojan can give different options to the attacker through a backdoor for CnC. This allows the attacker to step in and do malicious activities.

Spyware: The primary purpose of such malware is to spy on the victim's machine looking for valuable data to exfiltrate. However, Spyware can be equipped with other utilities or upgraded throughout the CnC channel. Spyware spreads like Trojans by attaching themselves to legitimate software to camouflage its existence. Usually, the Spyware is user's behavior specific and achieves its goal by listening to the user activities and input on the infected machine [13, 67]. An excellent example of Spyware is the KeyLogger [58].

Adware: A malware mainly designed to generate income for its owner. This is achieved by delivering advertisements and pop-up windows, especially on websites. The Adware infects the server and starts displaying Ads for any visitor. However, some Adware can be equipped with extra features, such as Spyware, and hence can be used for other purposes [104, 83].

Rootkit: A malicious program that deploys stealth techniques to avoid detection in a system. These techniques are complex and advanced. For instance, DLL hijacking and process injection are some methods to keep the Rootkit undetected [115]. The danger of a Rootkit is the ability to take complete control of the system and gain high privileges. Some Rootkits can modify the kernel's data structures via Direct Kernel Object Manipulation (DKOM). Which leads to incorrect user requests and hinders the existence of malicious activities [100, 39, 81]. This makes Rootkits very invasive and hard to remove.

Bots: Deriving its name from Robots, Bots were first created to manage the Internet-Relay-Channel (IRC) chat channels. Which is a text-based communication protocol developed by Jarkko Oikarinen in 1988 [70]. Not all Bots are harmful. Some Bots are used for commercial and legit business, such as Chat-Bots that can welcome a user and answer some questions. For example, the Slackbot. Others are built to do malicious activities and can form a Botnet, a set of interconnected

Bots. Several botnet typologies exist, such as Distributed or centralized [8]. Moreover, Bots are controlled via a CnC channel and can serve different purposes–For instance, running a Spam campaign, DDoS attacks, and others. A good example is the Mirai Botnet used to perform DDoS attacks on legitimate businesses [45, 116]. Botnets are also known for their fast-spreading capabilities and for infecting vulnerable devices. Bots aim to infect as many devices as possible and grow the botnet. The latter can be used for different purposes [19]

Ransomware: A malware that infects the victim's machine and holds it hostage, until the user pays a ransom, usually in the form of a Crypto-Currency transaction. The malware installs itself and starts encrypting the system's files. When the encryption is finished, the user is prompted with a window holding the details of the incident and how to get the data back. *WannaCry* is an example of Ransomware family [7]. Moreover, WannaCry propagates using the same means as a computer worm. These behavior have been witnessed with the two Ransomware versions found in the wild *WannaCryptor* and *Petya* [82].

3.1.2 Malware evolution

The first malware documented ranks back to the floppy disk era, where malware needed human interaction to move from one machine to another. After the birth of the Internet, we witnessed the appearance of new types of malware and an enhancement in their capabilities. Over the years, malware has gone from an experiment/prank to an uncontrollable threat to any machine. In their paper, Namanya et al. [82] present an overview of the world of malware and provide details on its evolution. They mention five malware generations:

- The first generation of malware mainly replicate with the assistance of human activity and through floppy disks, removable media, and CDs.
- Second-generation malware self-replicate without help and share the functional characteristics of the first generation. They propagate through files and media.
- Third Generation utilizes the capabilities of the Internet in their propagation vectors leading to big-impact viruses.
- Fourth Generation is more organization-specific and uses multiple vectors to attack mainly antivirus software or systems due to the commercialization of malware.
- Fifth Generation is characterized by the use of malware in cyber-warfare and the different variations of Malware-as-a-Service, which importance was mentioned in 1.1.

From what is known about malware today, and through this overview, we can clearly see that malware has evolved incrementally, adding new capabilities with each generation, building upon the previous one. For instance, Fifth generation malware can have all the abilities of the previous ones. It can replicate using removable media, as the First; self-replicate and use files as the Second; use the Internet to spread as the Third; and finally use multiple vectors as the Fourth.

3.2 Malware analysis

Malware analysis is the study of malware's behavior. The objective is to understand how malware works and how it can be detected and eliminated. The malware classification is based on malware analysis, composed of two methods: Static and Dynamic analysis [28]. Static analysis revolves around analyzing the source code of a program without running it. It is considered the safest way to analyze malware. However, malware is found as executable programs. Hence, the static analysis is based on reverse engineering the malware's binary to obtain the source code. With this at hand, the researchers can predict the malware behavior, extract the exploit, and build a detection mechanism against it. Some Reverse-Engineering tools available to the community are *Ghidra* [84] and *Radare2* [89]. However, no automated process exists yet, and the whole operation is performed manually. Making it time-consuming and requiring knowledge of low-level programming languages such as Assembly. The other type is dynamic analysis, also known as Sandboxing, where the malware runs in a Sandbox and is monitored during its entire execution lifetime. This makes studying malware faster, as the researchers can obtain the process, memory, and network traces left by the malware and develop detection techniques based on it.

3.3 Malware detection techniques

Malware detection techniques are methods to identify malicious software or code on a system. Sandboxes can extract a lot of information about the malware runtime. From this, detection methods are created, and they can detect the presence of malware by the following:

Behavioral based: The detection techniques are based on the malware's behavior, meaning all the actions it takes during its execution time. For instance, created files, accessed files, API calls, changing parameters in the host machine, and others. All this is collected as a dataset and is used to write appropriate detection [2].

Anomaly based: This relies on Machine-Learning where a model is trained with datasets. The model classifies the program based on anomalies [85, 90].

Signature based: This technique can be static and is based on identifying a se-

quence of bytes in the malware's binary. Or dynamic, based on the malware's inspection during runtime. All the data produced by the malware can be used to build a detection signature. Usually, by identifying a sequence of bytes, payload bytes, dropped file names, and their cryptographic hash values. This technique is very efficient against monomorphic malware as the payload and other produced bytes are constant. However, this fails miserably against polymorphic malware. Polymorphic malware is a sophisticated type and uses different techniques to avoid detection. The malware changes all the produced data during its runtime by using code obfuscation and encryption to replicate. Hence, for all the dropped files, the payload will have high entropy [2]. Since this project's scope is dynamic analysis, we focus more on the issues and short-comes of this field.

3.4 Honeypots

A honeypot is designed to appear vulnerable and lure adversaries into attacking it. The honeypots simulate a vulnerable program by returning fake data to adversaries, such as a vulnerable version of a communication protocol, such as SMB and others. Honeypots can be used in two scenarios; the first one is for defense purposes, where the honeypots reside behind the firewall to deceive any adversary that could reach the LAN into interacting with it. The other goal is for research, where the honeypot is placed in the open with a public IP. Hence, it can be accessed from anywhere. This will attract any adversary, malware, or bots that scan the Internet for future attack targets.

Honeypots can be divided into three categories depending on the interaction they offer to an adversary. Low Interaction Honeypots (LiHo) are the simplest, and they do not offer much except returning fake data to the user. Examples of such honeypots are Dionaea for computers and HosTaGe for mobiles [106]. They are considered very efficient as they can extract valuable data without much simulation overhead. Other honeypot types are Medium Interaction Honeypots (MeHo) and High Interaction Honeypots (HiHo). Moreover, Honeypots are a good defense tool and offer a platform for studying malware behavior and catching exploit payloads [98].

3.5 Malware Propagation Modeling

To study and evaluate malware propagation schemes, researchers adopt different methods to create some formulas that can be applied to predict the propagation behavior of malware. Malware Propagation Modeling (MPM) creates mathematical models to describe malware behavior during its propagation in a network or a system. The models predict how the malware will propagate into new systems,

identify vulnerabilities, and build mitigation strategies. Although it is not in the scope of this project, we briefly introduce some terminology used within the field of malware analysis and will be used in this report. There exist different MPMs, such as:

- *Agent Based Model (ABM)*, which is based on the simulation of different agents in a simulated environment, aims to study these agents' behavior based on established rules. These models have three main characteristics: agents, environments, and rules [17, 16]. This approach can provide more detailed information about how malware spreads in specific scenarios. Nevertheless, it can be more complex, and this complexity introduces an overhead when the number of agents to simulate rises due to the simulation computational cost.

- *Machine learning models*: By teaching Machine Learning (ML) models with data from previous malware infections to predict how new malware will propagate [61, 93]. However, it requires large amounts of data and is often used with other modeling approaches to enhance the correct prediction ratio.

- *Epidemiological models*: These models are built based on the epidemiology principles, which study the spread of diseases in Biology. The models use several parameters, such as infection and recovery rates, to simulate and predict malware propagation through a population of nodes (devices). In the recent year, several models have been introduced. The Simple (Classical) Epidemic Model—SI Model was the first model trained to predict malware propagation. The model is a straightforward adaptation of epidemics models used broadly in Biology. Typically, these models have been used to study virus propagation in a closed, controlled population, such as flu and other viruses [56]. Different models have been proposed to enhance the detection approach as the SI model fails to detect sophisticated spreading techniques, such as SIS, SIR, and SIRS models [64].

3.6 Evasion Techniques

One of the significant drawbacks of dynamic analysis is the evasion techniques (ETs) used by malware. As most of the deployed Sandboxes are open-source, the attackers can fingerprint them and build malware with evasive techniques to avoid being analyzed and revealing their cyber weapons. During its runtime, the malware can check if it exists inside a VM or the existence of a Sandbox by different means [79, 65, 78, 113]. In a survey by Sharma et al., they demonstrate such ETs utilized by Advanced Persistent Threats (APTs). According to CrwodStrike and other intelligence agencies, the APT is defined as follows: APT is a sophisticated and sustained cyberattack in which an intruder establishes an undetected network presence to steal sensitive data over a prolonged period via evasion techniques [3, 4]. Nevertheless, the interesting part of the survey is the summary of all the encountered ETs and anti-analysis techniques demonstrated below.

<table>
<tr><th colspan="4">Evasion Manoeuvres</th></tr>
<tr><th rowspan="2">Stealth Mechanism</th><th rowspan="2">Anti-Analysis Mechanism</th><th colspan="2">Cover Communication</th></tr>
<tr><th>Attribution Evasion</th><th>Network Perimeter Defense Evasion</th></tr>
<tr><td>Process Injection</td><td>Anti-Debugging & Anti RE</td><td>Fast Flux</td><td>Encrypted Communication</td></tr>
<tr><td>DLL Hicjaking</td><td>Anti-Emulation</td><td>Domain Generation Algorithms</td><td>Network Protocol Abuse</td></tr>
<tr><td>Fileless Technique</td><td>Anti-Virtualization</td><td>Abusing Cloud Infrastructure</td><td></td></tr>
<tr><td>Obfuscation</td><td>Anti-Sandbox</td><td></td><td></td></tr>
<tr><td>Polymorphic & Metamorphism</td><td></td><td></td><td></td></tr>
</table>

Table 3.1: Evasion Maneuvers summary

For instance, we can see that malware author can utilize anti-debugging techniques to prevent the malware from being debugged. This is usually used by calling API such as *isDebuggerPresent* on Windows OS. Moreover, we see the Anti-Virtualization and Anti-Sandbox techniques. Nevertheless, the project's scope is limited to ETs since we want the studied SPM to trigger and propagate. The ETs are summarized in two categories:

Timing-based techniques: The malware uses the system's resources to verify its identity. For instance, using the RTSC instruction. Since the Pentium appearance, the RTSC has been used as a 64-bit register on all x86 processors to count the number of cycles since reset [111]. Other methods involve a sleeping period, where the malware sets a future date for the execution. However, since the Sandbox runs only for a limited period (By default, 300 seconds), the malware will not run and cannot be studied.

Artifact-based techniques: The malware detects the presence of a VM or Sandbox based on traces left in the system. For VM detection, the malware can sense the presence of a hypervisor [12, 65]. Or check the file system and the registry. Or Check if a registry key exists [23]. All the disclosed ETs are listed here [42].

Human Artifact based: Another way the malware can check the environment for analysis artifacts is the lack of human behavior in a Sandbox. Usually, a Sandbox runs the malware. Hence, the malware can check for human activities by verifying mouse movement, browsing history, installed applications, and others [59, 5, 63].

Networking Artifact base: A Sandbox can usually allow Internet communication or prohibit it by dropping all outgoing packets. However, a DNS sinkhole such as *InetSim* can reply to all DNS queries with fake IP addresses [52]. Moreover, when

the Internet is allowed, the malware can inspect the global IP of the network and verify if it belongs to a known Sandbox provider or a research facility. Sometimes the malware will not trigger if no Internet is allowed inside the Sandbox [24]. Another way of detecting Sandbox via network artifacts is discussed in [114], where they crafted a program that collects data about its environment and sends it back to a server for processing. They submitted their program to some public Sandbox providers, such as VirusTotal, and gathered intelligence about the environment. The intelligence data includes public IP addresses, internet routes, and others. They demonstrated that an adversary can have the same approach and build evasive malware based on the gathered data.

3.7 Summary

In conclusion, there are many aspects to consider about malware, for instance, the fact that multiple malware families can be self-propagating or that a specific malware program can combine capacities from different malware families. Evasion Techniques (ET) also have to be considered, no matter what type of analysis. However, due to many known evasion techniques, especially those related to virtualization. By looking at the public ET database, we spot categories such as Hardware, Network, and others [42]. Moreover, since our project aims to build a testbed system that will allow analyzing the network behavior of an SPM. We will focus only on the ET the virtualization produces on the network aspect. We definitely recommend eliminating as many artifacts as possible. However, this is fine-tuning the Sandbox and is a secondary step of our project, as we consider building and integrating SPM as the primary goal.

Chapter 4

Related Work

This chapter aims to browse the literature on SPM and their network analysis to obtain valuable information about SPM behavior obtained. Eventually, one of the goals of this project is to build an analysis system for SPM analysis. Meaning other related work where the structure of the analysis system is shared will be valuable information that can guide us throughout the system's development. Furthermore, we will look at the detection techniques built and what dataset they were built upon. Lastly, as mentioned in the delimitation[1.5], we look for SPM candidates tested previously if the exact cryptographic value of the malware is presented (SHA256 or MD5). Otherwise, it will be hard for us to determine the candidates as most of the malware sample databases such as *Virusshare*, *malwarebazar* or *virusbay* do not have any label or tag for SPM. Other databases have the TAG option, such as *tria.ge*. However, the TAGS are arbitrary and can sometimes be misleading.

Akbanov et al. [6] study two Ransomware with worm capability. Both SPMs are different WannaCry versions, and their cryptographic hash value is *db349b97c37d22f5ea1d1841e3c89eb4* and *84c82835a5d21bbcf75a61706d8ab549*. They use two VMs with Windows 7 OS to simulate an environment for propagation. This enables the propagation of the two SPMs as they use the EternalBlue exploit to infect and take control. The VMs are interconnected using Open-vSwitch, designed to enable massive network automation through programmatic extension. The third VM has Ubuntu OS and is the GW for the two other VMs. Additionally, it has a network sniffing tool (Wireshark) and is used to capture the traffic for further analysis.

Moreover, they perform static analysis on the executable files with the Pestudio tool, free software used to examine executable files and supports 32-bit and 64-bit architectures. They reveal that the SPM has different components: The Worm, the Encrypter, and the Decrypter. During the SPM's execution, the worm invokes the *iphlpapi.dll* to get the network configuration of the infected host. Later, the encrypter invokes the *kernel32.dll* and *msvcrt.dll*. Further, the Crypto APIs are used

to generate random symmetric and asymmetric cryptographic keys. However, we are interested in the SPM execution and will focus on the dynamic analysis part. Additionally, they performed dynamic analysis using Sandbox technology with VM-based hosts, as shown above. The test showed that the SPM's worm component invokes the *InernetOpenUrl* function and attempts to establish a connection to www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com. They confirm that this network operation is a kill-switch or an evasion technique. The worm stops running if the domain is active and cannot connect to the requested domain name. It continues running under the process name *mssecsvs 2.0* on the infected machine. After the WannaCry installs itself as a service, the worm component extracts hardcoded resources and place them in the following directory: *C:\Window\taskche.exe*. Next, the encrypter is invoked and checks if at least one of the three mutual exclusion objects (mutexes) exists. If so, the encrypter terminates immediately. Otherwise, the encryption process starts. They do not mention why the encrypter behaves in such a way. However, we think it is to check if the machine is already infected, and the encrypter should not encrypt another time. Furthermore, they mention that the encrypter uses an AES 16-byte symmetric key with the help of *CryptGenRandom*. Lastly, all the AES keys are encrypted with the hardcoded public RSA key so only the malware owner can decrypt them. The SPM completes the infection process and renames all the encrypted files with the extension *.WNCRY*.

After infecting the first machine, referred to as *Patient-0*, the next phase infects other hosts available on the LAN. After the kill-switch connectivity check to the malicious domain name is passed. The worm component is active, retrieving its local IP to identify the subnets via the *GetAdaptersInfo*. It initializes the *mssecsvs 2.0* service, which tries to spread the WannaCry's payload to other hosts in the network through the SMB vulnerability on any vulnerable system [75]. From their analysis of the worm behavior, they noticed that WannaCry creates two separate threads that simultaneously replicate the payload on the LAN and WAN. Next, patient-0 tries to establish connections to all the possible addresses on the subnet (from XX.XX.XX.2 to XX.XX.XX.254). For instance, they see that patient-0 with the IP 192.168.180.130 sent SMB packets to the other VM at 192.168.180.134. Furthermore, they see that patient-0 tries to spread to external networks by spoofing IP addresses and attempting TCP connections on port 445. Another remark is the spoofed IPs used to connect to external networks, namely 192.168.56.20 and 172.16.99.5. These IPs are hardcoded into the worm and can be obtained by extracting strings from the WannaCry binaries. Particularly, WannaCry sends three NetBIOS session-setup packets. Two of the NetBIOS session packets contain the two aforementioned spoofed IP addresses. Additionally, the WannaCry tries to establish external connections to the C&C server using the data available on the *c.wncry* file, which holds the configuration data and a list of endpoint onion addresses in the form of *.onion* next to the compressed Tor installation file. While

communicating with Tor addresses, WannaCry creates a secure HTTPS connection on port 443, using standard Tor ports 9001 and 9050 for network traffic and directory information. All the pinpointed behavior of WannaCry is used as a basis for designing their detection and mitigation mechanism. Regarding their solution to the WannaCry spreading mechanism, they propose an SDN-based detection that relies on DNS traffic inspection with dynamic blacklisting. The solution monitors the network traffic for malicious links and IP addresses initiated during the WannaCry C&C communication. Each malicious domain or IP name is blacklisted, and a new open-flow table is added to the SDN mechanism. This mechanism has been proposed here [21] in a similar work against another version of Ransomware *CryptoWall*.

Finally, they tested their proposed solution and showcased its efficiency against the WannaCry Ransomware. They also claim that it can be used against any SPM that behaves similarly. For instance, the new flow table of the vSwitch will contain the infected machine's IP (192.168.180.130) and the corresponding TCP port numbers 445 and 139. Nevertheless, this basic solution relies on hardcoded domain names and IPs. I.e., If the malware uses a randomly generated domain name as a kill-switch, uses randomly generated spoofed IPs, or uses random ports, this detection method will be absolute. Another drawback is the detection concept. The system will allow the first infection to learn new malicious domain names and create the appropriate control flow to block future requests from the newly infected host. However, it will certainly stop any SPM that follows the same procedure with hardcoded data.

In another similar work against another variant of the WannaCry ransomware by Rouka et al. [97]. They followed the same strategy with the SDN setup and the number of hosts but with the *ExPetr* ransomware. The system setup is similar to the previous work mentioned above. It has the following hosts: Two Windows7 VMs, one the infected machine (*Patient-0*) and the other uninfected with the IPs 10.0.0.3 and 10.0.0.4 accordingly. An Ubuntu VM is acting as Gateway with 10.0.0.2. We will skip the infection process and focus on propagation to avoid repetition since the ransomware variants have the same final purpose [103].

After the host is infected, the worm component of *ExPetr* starts gathering information about the LAN. In the case of the presence of a DHCP server running, the worm extracts the set of existing IP addresses using *DHCPS.dll* and *NETAPI32.dll* libraries. In the case of no DHCP, the worm uses ARP to extract all the available hosts on the network. This is a significant upgrade from the previous ransomware version, where the worm uses TCP SYN scan to extract running hosts and open ports on the LAN. Hence, using a legit way of scanning. Nevertheless, the ARP broadcast packet needs to be sent n times, where n is the number of hosts on the LAN, trying to resolve each host at the time. Only the machine with the corresponding IP address will reply with the corresponding MAC address [55].

After the IPs of all the hosts on the LAN are resolved with ARP, the worm starts a TCP handshake, trying to connect ports 139 and 445. This TCP connection is to establish the SMB Protocol Negotiation and Session Setup. The credentials used to authenticate are retrieved from the Windows Active Directory during infection. However, If no Active Directory exist, they only witnessed the worm using the stolen credentials of the patient-0. Further, patient-0 sends a TCP request to connect to the admin folder of the target host. If this is successful, patient-0 copies some files into the admin folder. These malicious files will be remotely executed and infect the target host. Once the mission is completed, patient-0 performs an SMB Tree Disconnect, and the session is terminated between the two hosts, and the cycle of discovering other hosts continues. However, the *ExPetr* has a slightly different approach towards failed connections on TCP ports 139 and 445 with a running server. They witnessed the *ExPetr* trying to copy the malicious executable to the admin folder of the Ubuntu machine using an HTTP PROPFIND request. According to Microsoft, the HTTP PROPFIND method retrieves properties for a resource identified by request URI [53]. Nevertheless, the Ubuntu servers replied with error code 501 (Method Not Implemented). In conclusion, the *ExPetr* malware shows much sophistication by using a different scan via the ARP method and attempting connections with other protocols (SMB and HTTP). Nevertheless, these payload inspections are limited to this variant in this version. If the strings are different, these procedures will be absolute. Additionally, the strings to compare against are obtained beforehand using reverse engineering and not in real time.

Regarding their solution, they propose similar approaches to stop worm propagation as the previously discussed work. For instance, dropping all TCP packets with destination port 445 or 139 used for the SMB protocol (Port Blocking). Additionally, they propose protocol payload inspection searching for the bitcoin address associated with the *ExPetr* ransomware. The latter is extracted through reverse-engineering the binaries, and each SMB packet on ports 139 445 is inspected for the matching string *"1 Mz715 3HMuxXTuR2R1t78mGSdzaAtNbBWX"*. However, they do not elaborate on the matching strategy if it is *String A == String B* or matching the pattern. Regarding the HTTP protocol, they inspect all the traffic for the keyword *"PROPFIND ladmin$"*. Any packets that contain the malicious payload are dropped in both protocols. Later, they mention that the payload inspection has fewer false positives than the port blocking since it is more specific.

In a similar work by Alotaibi [10], they investigate the behavior of another ransomware named *BadRabbit*. They follow the same approach in the previous two related works [6, 97] by using SDN-based network security to stop the spread of the*BadRabbit*. However, their analysis system structure contains extra hosts and is listed as follows: They used VirtualBox to host the virtual machines (VMs) for dynamic analysis with four VMs created. The first is with REMnux as a GW, which hosts a fake HTTP service. Two Windows 10 systems, one infected with BadRabbit

and one clean, and one Windows 7 VM. They used different clean Windows OS to simulate different scenarios to better study the behavior of the worm in detail as follows:

1. Scenario 1: One uninfected Windows 7 VM with a password that does not exist in the worm's dictionary attack list. The other is a Windows 10 VM with the same credentials as the infected host. This is to see the propagation techniques of the *BadRabbit*.

2. Scenario 2: One Windows7 not patched against the MS17-010 vulnerabilities with an active SMBv1 authentication. The same is true for infected Windows 10 VM. This is to verify if *BadRabbit* did or did not exploit the EternalRomance vulnerability [80].

3. Scenario 3: One Windows 10 VM with credentials not in the worm's dictionary attack list. We failed to understand what they meant by the following sentence

   > "This has been done to monitor ransomware activity in an otherwise well-secured network."

   However, we believe they used this scenario to monitor the *BadRabbit* 's behavior when it cannot crack the credentials for the target host.

4. Scenario 4: One Windows 10 VM with existing credentials in the worm's dictionary attack list. The other host is a Windows 7 with credentials missing from the worm's dictionary attack list. However, they need to mention the purpose of such a scenario.

Regarding the *BadRabbit* propagation phase, the worm conducts a network enumeration to obtain the hosts on the LAN. The first enumeration is to extract the already known hosts by using the *GetExtendedTcpTable* function to retrieve a table containing a list of TCP endpoints available to the application [47]. We consider this operation a stealthy way of retrieving all the hosts known to the machine instead of scanning for them and producing network-traffic noise. However, this does not guarantee to have all the hosts within the current LAN the infected machine is on. The second enumeration is performed using the *GetIpNetTable* function to retrieve the IPv4 to physical address mapping table. From Windows Vista and later, the *GetIpNetTable2* function can retrieve the other hosts' IP addresses [48]. The worm uses *NetServerEnum* to extract all the running servers' IPs on the network. The last enumeration method is by using the following Windows DHCP APIs *DhcpEnumSubnetClients*, *DhcpEnumSubnets*, and *DhcpGetSubnetInfo*. We believe the worm uses them in the following order for stealth, and according to their description on Microsoft documentation:

1. DhcpEnumSubnets: Returns an enumerated list of subnets defined on the DHCP server [34]. This way, the worm checks if any subnets exist without scanning blindly for subnets.

2. DhcpGetSubnetInfo: Returns information on a specific subnet [35]. The worm can see if anything exists on the found subnet.

3. DhcpEnumSubnetClients: Returns an enumerated list of clients with served IP addresses in the specified subnet [33]. This way, the worm will have a list of available hosts without scanning.

If any hosts are found on the same LAN, the worm propagates using similar methods to the *ExPetr* malware. I.e., it uses the infected host's stolen credentials to authenticate to other target hosts. They mention that the malware uses Mimikatz tool [77] to obtain the credentials. Moreover, it uses a similar exploit against the SMB protocol vulnerabilities (EternalRomance). Furthermore, they mention that the *BadRabbit* worm component has some evasion techniques against anti-viruses and has countermeasures against debugging. Lastly, the mitigation proposed against the *BadRabbit* is similar to the previously discussed works. I.e., the detection algorithms target the protocols used by the malware (SMB and HTTP and the used ports 80, 139, and 445). Nevertheless, they suggest the usage of Honeypots to detect the *BadRabbit* as it sends packets on ports 445 and 139 to all the devices on the network. Honeypots are fake system that emulates a vulnerable service to lure attackers into attacking it. Any connection or connection attempts (network scan) will trigger the honeypot and raise a network alarm [107, 106]. Honeypots can be deployed in different places in a network to serve other purposes. If the honeypot is located behind the router, it will be an early detection system. Additionally, it can be exposed to the Internet; in this case, it serves as a data collection tool about the different attacks attempted on it. Similarly, honeypots have been used to collect intelligence about malware attacks and techniques. Sethia et al. have proposed to use *Dionaea* honeypot to capture malware traffic [98].

Some important worm characteristics are mentioned in [62]. They discuss the behavior of evasive worms, where the worm could be sophisticated and act cautiously to avoid detection. They define the different types of worms depending on their behavior. The *Classic Worm* is the least-sophisticated type and does not take any consideration to its environment, bandwidth, or monitoring tools. However, this is the easiest to detect due to its network burst during the enumeration. The *Evasive Worm* is more sophisticated with countermeasures against analysis systems. However, they do not mention examples of the evasive techniques, but we assume that the evasive worm could be equipped with Anti-Analysis or Anti-Sandbox techniques. Similarly, Eder-Neuhauser[38] et al. discuss different classes of worms depending on their level of stealth and sophistication. They name three types and provide the real malware names that fit this classification.

- *Pandemic Malware*: Aggressive malware that follows a topological-scan strategy to find and infect all devices on the network in the shortest time possible. Examples: *Code Red 1 and 2*, *Nimda*, *Slammer* and *Conficker*.
- *Endemic Malware*: Sacrifices speed over stealthiness and operates with a less conspicuous hit-list and permutation-scan strategy. Examples: *Regin*, *Duqu*, and *Flame*.
- *Contagion Malware*: Does not scan the network or initiate connections and appends on legitimate communication flows to avoid detection. Examples: *Gauss*, *Equation* and *AdWind*.

Chernikova et al. propose a cyber resiliency strategy against the WannaCry ransomware [25]. They study the problem of how to build a resilient system and how to configure it against SPMs. Moreover, they develop a model (SIIDR) to map the malware's network behavior to recognize and stop it from spreading. Again, the paper studies the old WannaCry relying on EternalBlue vulnerability and does not provide its cryptographic hash value. They use an anonymized dataset from an industry partner consisting of 3.4 million nodes and 50 million links for the dataset. Moreover, they chose to study only the traffic on ports 80 and 22 based on the Critical Watch Report of 2019[1], where it states that 65% of the vulnerabilities on TCP/UDP ports are associated with three ports: 22, 443 and 80. Regarding their testbed environment, they built a virtual-based analysis system with the following: One VM vulnerable to EternalBlue Windows exploit that was used in the 2017 WannaCry attack and another VM infected with WannaCry. Nevertheless, they do not mention the number of VMs. Furthermore, the model can identify the WannaCry spreading behavior based on two characteristics: The number of threads (ΔT) used to scan the network and the time between each scan (Δt). Concluding with their offered mitigation, which focuses on network segmentation principles such as Node Splitting, Edge Hardening, and Node Hardening.

In a recent work by Almashhadani et.al. [9], they built a testbed composed of two Physical machines (PhM) with Windows7, with one of them being *patient-0* and the other one clean with two VMs under its subnet. A Linux machine with a firewall enabled captures the LAN traffic and saves it to a database. The three machines are connected to the same subnet using a switch connected to a router. The test exhibit is *Locky*, and they provided the cryptographic value. Additionally, they allow internet access to the testbed as their analysis focuses on bidirectional communication analysis of the Ransomware. Regarding their analysis results, they found that *Locky* uses 3.77% HTTP, 7.25% DNS, and 6.05% NBNS. Additionally, they map all the TCP traffic for the *Locky* Ransomware and found out that around

[1]https://www.newnettechnologies.com/study-finds-majority-of-port-vulnerabilitiesare-found-in-three-ports.html.

99% is malicious. They observed numerous TCP reset (RST, ACK) packets in *Locky* 's traffic used to terminate the TCP connections abnormally. Which can be used as a metric for detection.

Regarding botnets, which are also known for their propagation ability especially targeting IoT devices [112]. Gallopeni et.al [46] have built a cluster system to study the behavior of the Mirai botnet closely. The system is a physical single-board computer cluster based on an ASUS-Tinker board (similar to Raspberry Pi) to simulate Mirai-infected hosts. They used a MikroTik router running Router-OS to have more programmatic control over the network. Additionally, they used two separate subnets to see the propagation scheme on two different levels. Their analysis showed that the scanning behavior can be identified by the number of ARP requests on the LAN, which can be detected by setting a threshold. Moreover, the C&C traffic can be identified by Telnet traffic, where the bot will initiate at least one C&C connection. Lastly, the attack commands were identified by the keyword *"detected"* in the payload.

We observed the lack of UDP from all the literature we have been through. Most of the work concentrates on TCP since the studied malware uses TCP for scanning. However, Norwawi et.al. [71] analyze the UDP-scanning behavior of internet worms. UDP is a connectionless protocol, which means it does not require a constant, established connection between the hosts and does not require a connection setup like the TCP handshake (SYN/SYN-ACK). Nevertheless, the UDP can still scan hosts and determine their state depending on the errors returned. The different scan scenarios are as follows: - **Open port**: If the destination host responds with UDP responses, the host is up, and the port is open. - **Open port with filter**: The worm sends a UDP request, but the destination host does not respond. This indicates an open port with a filter that UDP is considered to be open with filter [73]. - **Closed port**: If the destination's port is closed, the host responds with an ICMP Port Unreachable (ICMP type 3 code 3). - **host down**: If the host is down or the IP address is not used, the router will respond with an ICMP Unreachable Host (ICMP type 3 code 1). - **time out**: In case the destination host or the router drops a packet due to a time-out. An ICMP time exceeded is generated. Moreover, they propose the UDP Scanning Worm Detector (UDPSWD), an algorithm that can detect hosts that scan the network using the UDP. The algorithm is based on a counter of UDP connection error packets and compares it to a threshold. Nevertheless, they mention that this will not detect the successful UDP connection that could be made by an infected host. Regarding their setup, they used a Windows 2000 Professional Pack 4 with an Internet connection. They do not mention details about the worm used in the experiment nor justify some mentioned host preparation. However, we consider this work relevant as it tackles the UDP scanning aspect of SPM.

4.1 Summary

From all the discussed work, we have evaluated the testbeds mentioned above and their analysis performance. Moreover, we could see the evolution of the worm's capabilities when it comes to the propagation phase. –For instance, an upgrade in the scanning technique by using the ARP scan and avoiding using a TCP SYN scan, known as the Defacto of the network enumeration phase of every cyber attack. Another remark is the evolution of malware scanning targets and techniques. As discussed earlier, the worm scans the network horizontally, i.e., scans its subnet for other hosts. In addition to a bidirectional vertical scan, by spoofing IPs and scanning from the Gateway level, scan the infected machine's subnet and its client, if any exist. During our literature review, we usually encountered VM-based testbeds. This is because it is an easy, cheap, and scalable solution. Additionally, the malware sample tested uses an Internet-based kill-switch and do not check the environment for evasion techniques. We assume such a conclusion because the malware completes the installation phase, which is the phase where the malware will check for evasion techniques and abort installation to avoid behavioral studies. Furthermore, we encountered studies that mention the term *evasive worm*. To our knowledge, this term describes worms that have evasive techniques to not be detected during the propagation phase. However, we believe a worm can be even more sophisticated and fingerprint the testbed network for Sandbox traces. For instance, the worm can fingerprint how many hosts are on the network and abort if no other hosts exist.

Chapter 5

Technical analysis

To choose from the different possibilities to build an analysis system that can monitor the behavior of SPM, we conducted a search on the available technologies. From the literature review, we encountered the usage of Virtual Machines (VMs) or physical machines as testbeds. Furthermore, we experienced using REMnux, a Linux toolkit for reverse engineering and malware analysis. It provides a valuable collection of free analysis tools. Nevertheless, it is not an open source, and we will have less control over the Sandbox mechanism [94]. Thus, we shifted the focus to only open-source dynamic analysis tools. We find the two most famous when looking at the available open-source dynamic analysis tools (Sandbox). Cuckoo and CAPEv2. CAPEv2 is the updated version of Cuckoo Sandbox and is actively maintained, contrary to Cuckoo, which is discontinued for maintenance and updates [29]. The choice of using the Sandbox instead of building a program from scratch is due to its well documented and rich with other tools that might be needed next to the network traffic monitoring. For instance, memory analysis and process analysis. The other reason is to refrain from recreating the wheel and spending time enhancing it instead.

Each time the Sandbox analysis terminates, the testbed are returned to their initial saved state. For the VMs, the hypervisor manages all these operations and runs the malware analysis on a snapshot. A snapshot is an image of the OS that the hypervisor uses to run the OS without affecting the original state. Nevertheless, the physical testbed can be imaged locally or remotely. The local imaging is performed by using software like DeepFreeze or similar. DeepFreeze freezes the drivers (Storage HDD) and restores them to their original state [43]. If the choice is remote imaging, the FOG-Project server can be used. FOG is a free open-source network computer cloning and management solution. It can image Windows, Linux, and MacOS [92]. The FOG server is considered the most secure as it keeps the testbed image in another location and sends it to the client machine over the network. This is another precaution if the local backup gets corrupted. To raise confidence, we

recommend using both solutions. Moreover, FOG uses Preboot Execution Environment (PXE) protocol to restore the physical testbed to the initial saved state. PXE, an industry-standard crafted by Intel, provides pre-boot services at the device firmware. PXE enables downloading network boot programs to client devices. However, the PXE requires the physical testbed to be connected via Ethernet, as the PXE loads before the OS. This is another limitation because it will need a LAN switch to build a populated network with physical devices that require restoration in case of infection.

Regarding the testbeds, we prioritized the physical testbed to save time fighting against evasive techniques. As discussed earlier, some malware is equipped with evasive techniques that can detect the existence of a VM or the Sandbox mechanism. Hence, it will not trigger malicious code and avoid being analyzed. Nevertheless, we also prepare a VM testbed. This is due to their easy and fast deployment. For instance, the VM's OS installation takes less time than a physical device. Hence, we used VMs to prepare and test the testbed for functionalities. Followed by documenting each step to replicate it on a physical machine. This helped to save a lot of time.

Finally, we present the ideal scenario, where the perfect system is presented as a coupled different entities on the same device. Nevertheless, due to the previously mentioned limitations (physical and software), we ended up with the same system spread over different devices. It is worth noting that this limitation does not present any performance decrease in the system.

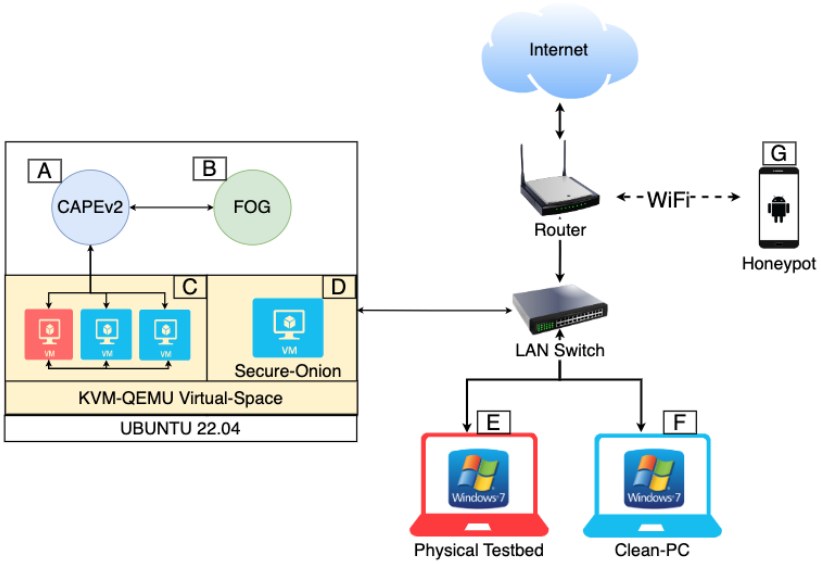


Figure 5.1: Ideal System topology

The components of the system are labeled with capital letters and are as follows:

- **Router**: Used to create an isolated subnet.
- **Switch**: The switch creates a LAN network due to the router's Ethernet port limitation (only four LAN ports).
- **A**: The CAPEv2 Sandbox is installed on a bare-metal host running Ubuntu 22.04.
- **B**: The FOG imaging server. Due to its OS limitation, we moved it to another host.
- **C**: The virtual environment created by the KVM hypervisor. It contains a Virtual testbed (in red) and at least two other VMs acting as target hosts.
- **D**: Secure-Onion is a network monitoring and forensics tool with various free plugins [86].
- **E**: The physical testbed where the malware will be executed. Due to the FOG imaging server requirement, this machine must be connected via Ethernet.
- **F**: Another physical computer acting as a target for the SPM. This machine is imaged using the FOG server and can be restored on demand. I.e., it can be restored only in case of infection. Hence, it does not need to be connected with Ethernet.
- **G**: A Samsung Mobile connected via WiFi and running a mobile honeypot application. This honeypot is used to alert about scans on the LAN, acts as another target on the network, and makes it heterogeneous.

Finally, due to the encountered limitations, we set up the sandbox and its helping components in separate machines to test the proof of concept. Thus, installing all the components on one machine is scheduled as future work. The current setup is as follows:

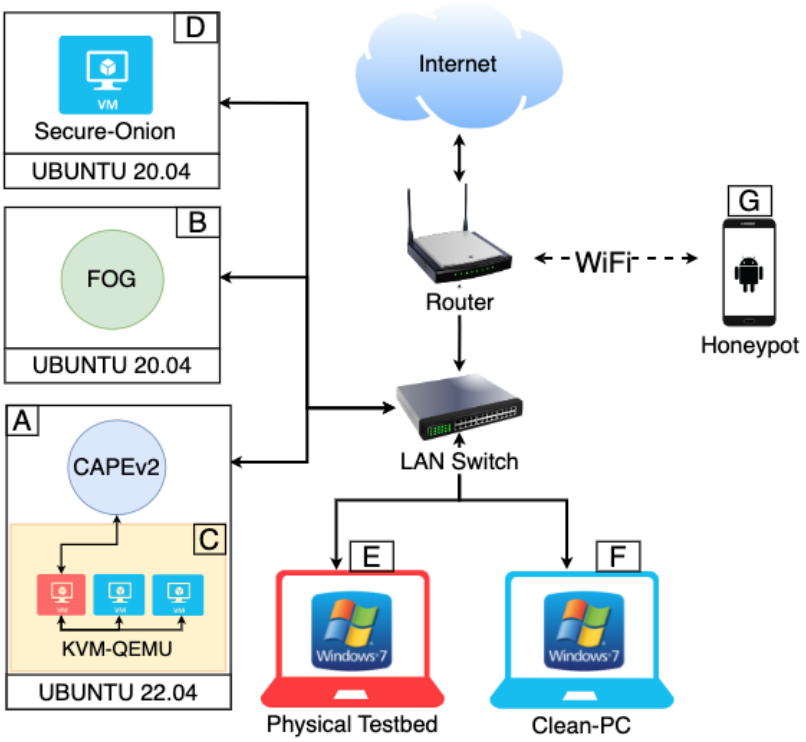


Figure 5.2: Current system topology

5.1 Hardware

After displaying the system's layout and its components, we list the hardware used in the project. The hardware is listed in the table below:

Machine	CPU cores	RAM	Storage
ThinkStation-1	16	48GB	450GB
ThinkStation-2	24	64GB	2TB
ThinkPad	4	8GB	250GB
Laptop 1	4	4GB	465GB
Laptop 2	4	4GB	465GB
Router	xx	xx	xx
LAN switch	xx	xx	xx

The Thinkstation-1 is the first received hardware to build the project. It has only one NIC (Ethernet) and hence a limitation concerning subnetting. I.e., we could

bridge the WiFi card and the Ethernet to create a subnet, where the Thinkstation-1 can act as a router while having an Internet connection available [69]. Other limitations concerning Security Onion are listed in section 5.3.4. Furthermore, the Thinkstation-2 was received a bit late during the project. It has more computational power, RAM, and storage but only has one NIC. Hence, we decided to use it to host Security Onion instead, as we focused on testing the concept. ThinkPad is the laptop on which FOG Server is running, as it has required RAM and storage. Laptop-1 is the physical testbed and acts as Patient-0. Once infected with an SPM, we expect patient-0 to start scanning the LAN for other victims to propagate to. Once the analysis is completed, the computer is restored to the initial saved state via the FOG server. The Laptop-2 serves as a target for the SPM propagation. We only repair it on demand in case of infection. The number of devices to act as targets should be at least 1. The more devices, the better.

The router creates an independent network and avoids being on the same network as other fellow students. It also gives us control over the DHCP server, where we can reserve IP addresses for the different hosts without using static IP on the machines.

Finally, the network switch is an 8-Port 10/100/1000Mbps Desktop Switch. We use it to overcome the 4 LAN ports limitation on the router. Nonetheless, it is plug-and-play and does not provide Port-Mirroring capability [105]. Port mirroring enables sending a copy of network packets sent over one switch port to another called a monitoring port. This is a considerable limitation to our project as we intend to collect the network traffic from the infected machine towards the entire network. While a LAN switch will direct Unicast packets only toward its recipient. Hence, the Sandbox can only sniff packets with its IP and will not see others. Nevertheless, we recommend a switch with the mirror-port capability to solve this issue. For instance, the Catalyst 3750 Switch [27] allows the configuration of a network analysis port, where the traffic can be directed for analysis. Otherwise, we have to collect traffic from each machine.

5.2 Network

The network section is related to our threat model, where we list the malware capabilities and limitations and map the entire system's network for both scenarios. The VM analysis is a more secure approach as the Sandbox has complete control over the VMs. Namely, the Sandbox is the Gateway for the VMs and can allow or drop the traffic generated from the VMs toward the outside internet. Hence, when malware tries to scan other networks, the Sandbox can drop all traffic to different interfaces and allow only scanning on the virtual interface.

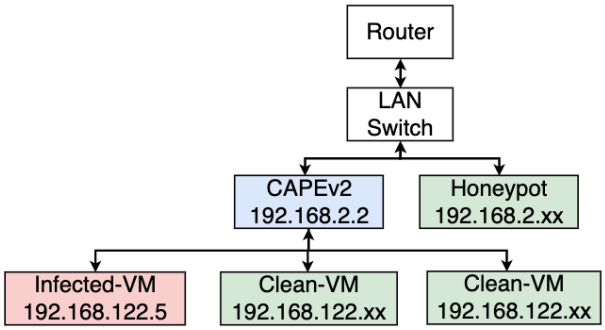


Figure 5.3: Virtual testbed network topology

Concerning the physical testbed, the Sandbox and the physical testbed are both on the same LAN, with the following IPs 192.168.2.2 and 192.168.2.4, respectively. This means the CAPEv2 Sandbox is visible to the malware as another host on the same network, not its Gateway. This is due to the physical limitation we have in our hardware. We only have one NIC and could not have a physical subnet by bridging the two interfaces (WiFi and Ethernet). Thus, we use a router and a switch to create a LAN. Moreover, firewall rules are adopted to minimize the malware's possibilities against the sandbox host machine. I.e., we cannot deny all the traffic between the physical and the Sandbox itself. Nevertheless, we only allow communication on the needed ports. Hence other connection attempts on other ports are dropped. Finally, honeypots are used to alert about any network scanning happening at the sandbox level in both the physical and the virtual setup. Moreover, we do not use the honeypots to catch malware payload as the Sandbox will already capture it. They are used to populate the network with targets for the malware and create a heterogeneous network.

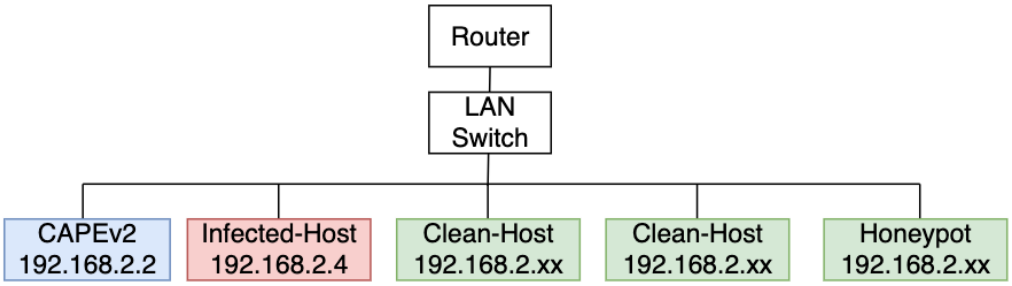


Figure 5.4: Physical testbed network topology

Finally, we assume that the malware can scan in all directions. It can scan the

LAN for other hosts or its DHCP server to check if it has other hosts in its subnet. Lastly, it can scan the Gateway and its LAN for available hosts. The following figure describes the possible malware scanning scenarios:

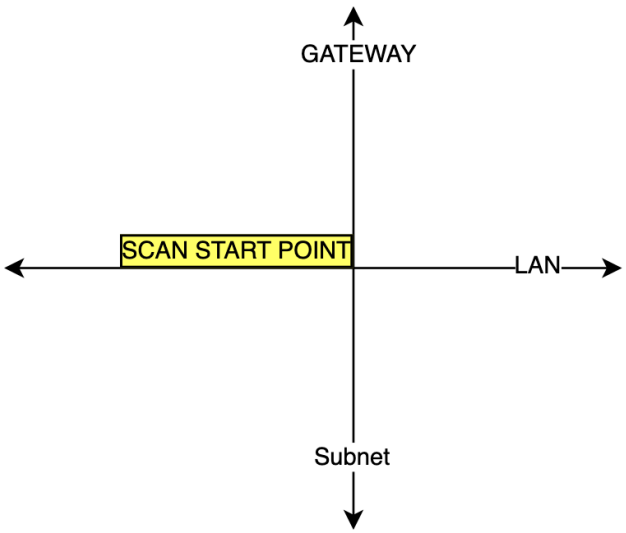


Figure 5.5: Possible scanning scenarios

5.3 Software

This section lists the software used in the project chronologically to follow the same structure as in the Hardware section. It is worth mentioning that some software, such as the Sandbox, is a coupling of different software. Nevertheless, we address it as a single entity for simplification.

5.3.1 CAPEv2

CAPEv2 is an open-source automated malware analysis system, also known as Sandbox. It enables automatic submission, extraction (if zipped), and execution of sample programs (usually malware) inside a contained environment (Virtual or physical). Below is the architecture from the official website. It only displays the Virtual testbed option. However, the same architecture applies to the physical testbed.

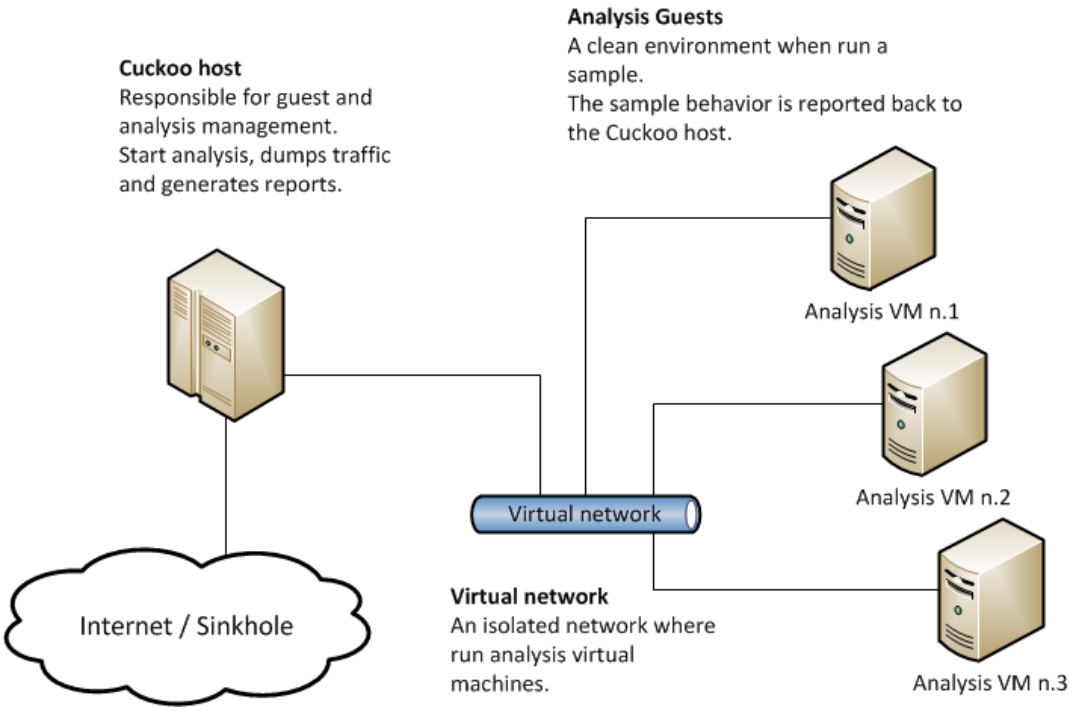


Figure 5.6: CAPEv2 architecture. Source[1]

CAPEv2 is installed on a Ubuntu 22.04 host machine. The OS version choice is recommended by the CAPEv2 requirements [87]. However, Ubuntu 20.04 is also supported. Nevertheless, we faced some issues with this version with the sandbox installation and decided to leave it as future work. Moreover, CAPEv2 enables retrieving the following information from the analysis:

- API-Trace: List of win32 API calls originating from all the processes spawned by the malware.
- File-Trace: List all the opened, created, and deleted files created by the malware during the analysis time.
- Memory-Dump: Provides the memory dump of malware processes or the option of obtaining the testbed machine's full memory dump.
- Screenshots: Visual documentation of the desktop during the malware execution.
- Network-Trace: The option of obtaining the network traffic of the testbed machine. This is considered the most critical aspect of our project's requirement, as we focus on the network behavior of SPMs.

[1]https://capev2.readthedocs.io

Concerning the Virtual testbeds, the CAPEv2 official documentation recommends using KVM-QEMU hypervisor over others as it offers better stealth than other hypervisors such as VirtualBox. The Sandbox comprises four main processes and runs as services on the host OS. Namely, *cape.service* is responsible for the whole Sandbox. It manages the submission and triggers the *cape-processor.service* once the analysis is done for processing. The *cape-rooter.service* is responsible for routing and managing the traffic. As noted in the documentation, the *cape-rooter* is the only service that runs with root privileges. Finally, the *cape-web.service* manages the webserver to display data for the user. Moreover, CAPEv2 uses Postgresql by default as the database to store the analysis data, such as Task-ID, the sample's cryptographic hash value (usually MD5 and SHA256), and others. While MongoDB is used to keep the analysis processed data, such as network traffic files (pcap), memory dump, process dump, and others. Below is a simplified diagram to demonstrate the internal structure of the CAPEv2 sandbox with a virtual testbed as an example.

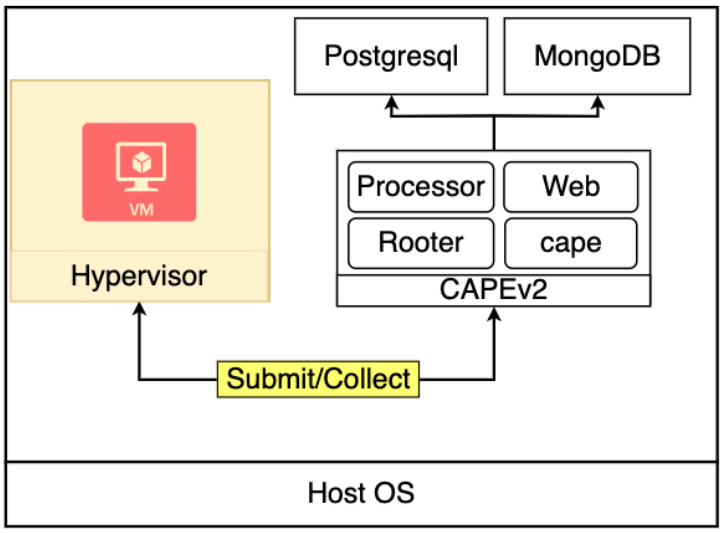


Figure 5.7: CAPEv2 internal structure

Other tools are integrated with CAPEv2 and are used to obtain data, such as *tcpdump*, a program to sniff the network traffic. It is a part of the system's program, so we do not include it in the diagram. Suricata and others are as well used by CAPEv2. Nevertheless, the Sandbox uses it only to process data. Hence, it is considered a plugin and not a core element of the Sandbox.

5.3.2 Testbed OS

Windows 7 is the OS for virtual and physical testbed and the clean target machines. This choice is made upon the need to allow the testbed infection. I.e., we chose an Operating-Systems that is known to be vulnerable to some known malware. For instance, the SMB protocol vulnerability [74]. The earlier related work also inspired the choice, where some Ransomware variants displayed self-spreading abilities. Regarding the virtual testbed, we have three VMs. Patient-0 and another clean Windows 7 VM. Next to a Windows 10 VM without user authentication to ease malware propagation. An additional Ubuntu 22.04 server VM is added to increase the virtual network heterogeneity.

5.3.3 FOG

FOG is a computer cloning and management solution that can capture and deploy disk images over the network. In a dynamic malware analysis environment, it is crucial to be able to restore a system to a clean state. The FOG Project allows us to create clean, fully configured systems images. It is compatible with different versions of all the leading operating systems, namely Windows, Linux, and MacOS, which can fit the diverse needs of malware analysis. FOG is designed to be scalable and is particularly useful when multiple machines are required. Additionally, the management is centralized, simplifying the administration of backup tasks across multiple machines. FOG is considered more secure than other solutions, such as DeepFreeze, as the machine images are kept in a remote location instead of locally, eliminating the risk of the local backup image being corrupted by malware. Furthermore, we can ensure a consistent configuration and state by deploying images tailored to our needs. This consistency eliminates variations that may affect the accuracy of our analysis results. It also ensures the results' reproducibility, allowing other researchers to validate findings and replicate experiments.

One of our first ideas was to have multiple Raspberry Pis acting as target machines, mainly due to the abundance in the university's Lab. However, as Raspberry Pis do not support PXE boot by default, FOG cannot capture and deploy images of those. It is possible to make it work by modifying the firmware of the Raspberry Pi and FOG's source code. However, this is a really time-consuming task and will derive us from the focus of our project. Hence, we decided to use devices supported by FOG out of the box.

5.3.4 Security Onion

We use Security Onion as a monitoring tool. It is a free Linux distribution that combines several open-source security tools, such as Zeek, Suricata, and Stenographer, into a single integrated solution. One of the core components of Security

Onion is the Security Onion Console (SOC), which serves as the primary interface for managing and investigating security events. Security Onion is designed to be scalable and handle large networks, which allows us to add machines at will. Furthermore, Security Onion comes with predefined rules (that can be personalized) and signatures that are used for real-time detection of malicious activities such as network-based infections and CnC communication, but also behavioral analysis making it capable of detecting derivations from normal behavior or suspicious communication patterns. Finally, it incorporates the Elastic Stack, which collects logs from different sources into a centralized, searchable repository. This allows us to perform detailed forensic analysis, review network traffic, and correlate events across other systems, even after the analysis is finished.

Security Onion provides different deployment types: Import, Evaluation, Standalone, and Distributed. The Import mode, which provides the most straightforward architecture, is a single box that runs components to import PCAP files and analyze. It does not have any monitoring capabilities. The following architecture is Evaluation. It has a network interface dedicated to sniffing live traffic and generating logs. It is designed for a quick installation to temporarily test Security Onion. It is not intended for production usage. The Standalone deployment is similar to the Evaluation. However, it will generate more logs and is suitable for testing, labs, and POCs. It is designed for production in low-throughput environments. The last deployment mode is Distributed, which, as the name implies, consists of multiple nodes running sensors. It is the recommended deployment type, providing greater scalability and performance. However, it costs more upfront to set up, as numerous machines across the network are needed. We chose the Standalone deployment mode as it offers many integrated forensic tools within the Sandbox ecosystem, contributing to a more effective and precise analysis.

Security Onion needs One "Management" NIC and one "Monitoring" NIC to function correctly. The Management NIC is responsible for giving an IP address to the Security Onion machine, where other computers in the network can access and manage it. The Monitoring NIC has no IP address and is in promiscuous mode, meaning it sniffs the whole network. However, this is a limitation. The ThinkStation computers we received to build this project have only one NIC. The solution we found is to install Security Onion in a VM where 2 NICs are attached in bridge mode to the physical network, with the Monitoring NIC being in "Promiscuous mode -> Allow all". We also found another limitation by trial and error: This solution only works with VirtualBox and not KVM. We also faced another hardware limitation using Security Onion. Indeed, the router we were given for this project does not support port mirroring. This dramatically limits the ability of Security Onion to do real-time monitoring and automated analysis. However, this does not render Security Onion totally useless, as the user can still manually import a pcap file captured by the Sandbox for further analysis. By doing so, Security Onion will

import the pcap into the console and generate IDS alerts and network metadata, which is more convenient for forensic analysis.

5.3.5 Malware selection

To test our Sandbox against malware to determine its accuracy and robustness, we replicated the same experiment with the two candidates discussed in the section Related Work [4]. Namely, the two Ransomware with the following MD5 cryptographic Hash value. This will give us some data to compare to determine the Sandbox's functionality. Finally, we use Virushare [109] and MalwareBazre [68] as database sources for the malware samples.

Malware	Corresponding MD5
Candidate 1	db349b97c37d22f5ea1d1841e3c89eb4
Candidate 2	84c82835a5d21bbcf75a61706d8ab549

Table 5.1: SPM samples information

5.3.6 Firewall rules

To provide more security to the testbed, we set up firewall rules on three machines in our testbed. Namely, the CAPEv2, the FOG, and the Security Onion machines. The other machines are considered target machines and do not need security, as the goal is to trigger malware. The firewall rules are access control mechanisms that firewalls use to protect from unwanted traffic. In our case, we use them for containment and isolation. The goal is to restrict malware in our specific test environments and prevent it from spreading beyond.

For the FOG machine, we have multiple ports that need to be opened, as it depends on some network protocols, like FTP, NFS, DHCP, and HTTPS.

PORT	Usage
21	FTP - Used by FOG for file transfer and to move and rename image files at the end of image capture.
80/443	HTTP/HTTPS - used by FOG for client-to-server communication.
111	RPCBIND - Used by FOG to help clients to determine the ports on which other services in the system are running.
2049	NFS - Used by FOG for image capture and deployment.
3306	MySQL - The database holding information and metadata about hosts and images.

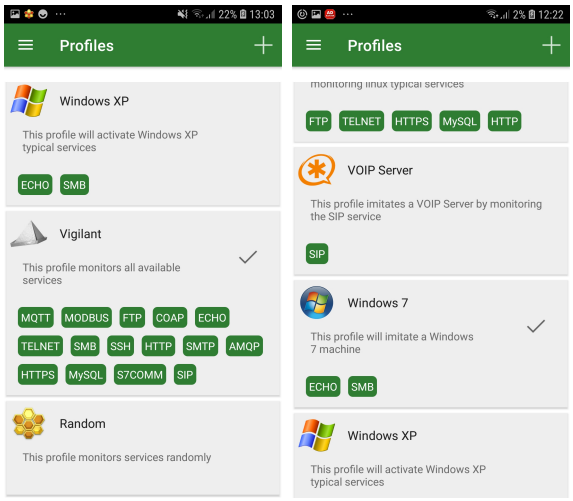
Table 5.2: FOG machine required open Ports

Regarding the CAPEv2 Sandbox, the machine needs to allow communication on port 8000 (HTTP) - for the Agent.py. Port 2042 (HTTP) must also be open for the result server, which receives data from the testbeds. We do not want any incoming connections to the machine for Security Onion. Hence, all incoming connections on all ports are denied.

As it can be seen in figure [D.2], the port *22* is opened on all physical machines. It is used to connect to the devices remotely and perform administration tasks. It is used primarily for debugging purposes. Under analysis, we recommend that port *22* be closed on all machines. However, it is not mandatory as SSH is protected by public key encryption and, in this case, a substantial 12 characters password.

5.3.7 Honeypots

Honeypots To Go, or HosTaGe, is a low-interaction mobile honeypot that aims at porting the honeypot technology into smartphones. It helps detect malicious or compromised WiFi networks by simulating a vulnerable application and attracting adversaries. It can emulate the following protocols as of the latest version: AMQP, COAP, ECHO, FTP, HTTP, HTTPS, MySQL, MQTT, MODBUS, S7COMM, SNMP, SIP, SMB, SSH, SMTP, and TELNET [1]. Moreover, we ensure the mobile's OS where HosTaGe is installed is the latest version. Finally, we use the honeypot to populate the network with heterogeneous devices. Nevertheless, we recommend activating all the protocols supported to enhance the chance of malware-honeypot interaction. This can be done by choosing the service *Vigilant*, or by selecting the Windows 7 profile to simulate a vulnerable Windows7 machine:



(a) Vigilant Profile **(b)** Windows 7 profile

Figure 5.8: HosTaGe suggested profiles

5.4 Implementation

The CAPEv2 Sandbox documentation is well documented in general and detailed around the Sandbox usage. However, some steps and crucial setup can hinder functionality. This section gives a detailed installation guide and troubleshoots some installation conflicts. The CAPEv2 team clearly states that their installation script is not a silver bullet. Moreover, CAPEv2 is maintained in a rolling manner. I.e., there is no release or version. Once you install it, you will pull the latest repository from GitHub [88]. Finally, our contribution will decrease the time needed to install, configure and test CAPEv2 Sandbox to less than one hour.

5.4.1 Milestone 1: Prepare the host

The host OS choice is due to our experience with the Linux platform and due to the continuous security updates support for Ubuntu 22.04. Furthermore, we used Ubuntu 22.04 with minimal installation configuration. This is to ensure that all packages are correctly installed. Both to free memory and avoid package conflicts. Once done, we recommend installing Timeshift as a backup and recovery tool. It will help keep a snapshot of the system and recover if the Sandbox installation fails. Next is to equip the host with some tools and packages required to build the KVM-QEMU hypervisor. The dependencies are collected in one install script[A.1]. Followed by installing the KVM-QEMU hypervisor. The CAPEv2 documentation offers an install script, but we wrote our own to avoid errors. See the appendix for the installation script[A.2]. Once the installation and reboot finish, we recommend taking a snapshot with Timeshift after verifying that the virtual manager works appropriately. Now the host is prepared for the Sandbox installation. To install the Sandbox, we used the script provided by CAPEv2 on github [88]. Nevertheless, we modified it slightly to avoid errors and some installation miss-configurations. For instance, some dependencies needed to install MongoDB are omitted, breaking the installation flow. Leading to a broken Sandbox installation with many errors. Additionally, we add some Postgresql database commands to correct the user's permission. The Sandbox's user needs to be an admin to create, write, and delete tables from the database. The modified script is available on this link[[2]], as it is very long to add to the appendix. After the installation is complete, a reboot is recommended.

5.4.2 Milestone 2: Prepare the testbeds

During our experimentation with CAPEv2 Sandbox, we discovered that only the Windows 7 32bit version works, and we decided to keep the documentation in

[2]https://drive.google.com/file/d/1VnAdzZbOJTnzL7rSAocxbd9WiKyIzFwe/view?usp=sharing

detail. Aiming to reduce the time required to prepare a testbed. Most of the download links from Microsoft do not exist anymore. Hence we used the web archive to find one. We recommend having a physical machine with at least 4 CPU cores and a minimum of 8GB RAM or a virtual machine with those exact requirements, as we experienced good performance with these values. We must first install the base OS - Windows 7 32bit. Second, we must install the software needed to run the Sandbox - .Net Framework, Python3 32bit, CAPEv2 agent, and re2 Python library, among others. Complete installation, configuration steps, and details are in the appendix [B].

5.4.3 Milestone 3: Configure the Sandbox & Troubleshooting

After the testbeds are ready, we start configuring the Sandbox and fix upcoming issues. We first begin to modify CAPEv2 configuration files used to configure the general behavior of the Sandbox and the analysis options. As stated by CAPEv2 in the documentation, the automated installation of CAPEv2 uses third-party dependencies that change frequently and can break the installation. Hence, we must check the installation and CAPE service logs and fix the issues ourselves. While experimenting with the installation script, we encountered some issues with MongoDB, poetry, and Detect It Easy. Details concerning Sandbox configuration and issues troubleshooting are attached in the appendix [C].

5.4.4 Milestone 4: Integrating The SPM analysis support

We must create and configure multiple hosts on the physical and virtual systems to make the environment suitable for malware propagation. The essential part of a good analysis is patient-0, where the malware has memory access. Hence, the VM should be with as few artifacts as possible. For instance, the VM virtual hard drive should be more than 60GB to avoid triggering the storage evasion technique [42]. Nevertheless, the other target hosts are not bound by this restriction since the malware is expected to only do network fingerprinting. We build multiple VMs with the minimum space to support an Operating-System. For Windows 7 and 10 32bit OS, 1 gigahertz (GHz) or faster x86 Processor, 1 GB RAM, and 16 GB available hard disk space [76]. We build the target VMs with 20GB each with 5GB extra for additional software. The Ubuntu server VM has 10GB of storage. Furthermore, using different VMs with different OS makes the virtual network heterogeneous and will help study the malware in a better fashion. I.e., it could expose some hidden malware capability, where the malware tries to spread to different platforms. This is an effort to advise towards heterogeneous networks, as most of the encountered work does provide only a non-heterogeneous environment. Below is a snippet of the KVM GUI showing all the target hosts prefixed with SPM next to the testbed VM *win7x86pro*. We allow the network sharing option in the *spm1_win7x86home* to open

some connections ports that are used by some Ransomware variants to spread, as witnessed in the related work section [97], where the WannaCry Ransomware uses the SMB protocol to carry the attack.

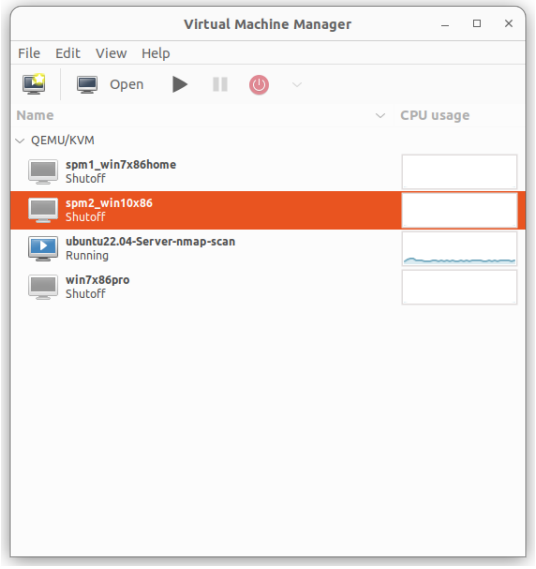


Figure 5.9: VM-Testbed and other target hosts

Concerning the physical system, we use multiple machines, as discussed in section [5.1]. We enable the Windows Home-Group network sharing option on both systems. This will allow the SMB vulnerability known for the two Ransomware samples we choose as candidates. Namely ensuring that the Sandbox can monitor the SMB interaction and we can verify and validate by looking at the network traffic. A full scan of both networks can be found in the appendix [D.1] and [D.2] Furthermore, we conduct network forensics and remove the VM network artifacts. During the SPM integration, we see a technical problem with the MAC Addresses of the VMs. The KVM-QEMU uses the *52:54:00* identifier, and we do not want the malware to be able to fingerprint the hosts using this artifact. Hence, we change the identifier to *DC:53:60*, the MAC Address identifier for Intel. We need to do this for each VM.

```
1 Nmap scan report for Jason-PC (192.168.122.218)
2 Host is up (0.0010s latency).
3 Not shown: 991 filtered ports
4 PORT STATE SERVICE
5 ...
6 MAC Address: 52:54:00:79:1E:A6 (QEMU virtual NIC)
```

Listing 5.1: Nmap scan results of the target VM

Additionally, the malware can fingerprint the default virtual gateway as shown below:

```
1  Nmap scan report for Station0 (192.168.122.1)
2  Host is up (0.00019s latency).
3  Not shown: 995 closed ports
4  ...
5  MAC Address: 52:54:00:15:C5:A2 (QEMU virtual NIC)
```

To remove this network artifact, we need to change the KVM-QEMU MAC address to *F8:D1:11*, which is the identifier of the TP-LINK vendor [66]. We emphasize changing the MAC address of the virtual interface to something similar to a router so it matches the network topology and enhances the heterogeneous network simulation. Regarding the physical testbed, the network is already heterogeneous enough with different OS and devices (Mobile Phones). Furthermore, the file to modify is at */etc/libvirt/qemu/networks/default.xml*. Consequently, the hosts will display the real provider's MAC address as shown from the nmap scan below. The full scan results are included in the appendix[D.1].

```
1  # Qemu-Kvm Bridge (GATEWAY)
2  Nmap scan report for Station0 (192.168.122.1)
3  Host is up (0.000097s latency).
4  Not shown: 994 closed ports
5  PORT STATE SERVICE
6  ...
7  MAC Address: F8:D1:11:15:C5:A3 (Tp-link Technologies)
8  # win7x86pro (Patient-0)
9  Nmap scan report for 192.168.122.5
10 Host is up (0.00017s latency).
11 Not shown: 990 closed ports
12 PORT STATE SERVICE
13 ...
14 MAC Address: DC:53:60:18:97:37 (Intel Corporate)
```

After preparing the network environment to be heterogeneous, we need to modify the CAPEv2 source code to adapt it for the SPM analysis. To do so, we had to investigate the source code to find the script responsible for the submission. Namely in the *submit.py* Python script. This script can be used by the user or via the Web-GUI. Moreover, to modify the implementation without changing the sandbox default behavior, we chose to follow the CAPEv2 developer's approach, and these are the following steps:

Create the *spm.conf* file, a configuration file where we can list the names of the SPM VMs. The file is formatted as follows:

```
[spm]
# list of the spm-machines from [virsh list --all]
machines = spm1_win7x86home,spm2_win10x86
```

Now, we modify the *submit.py* to support an additional argument as follows: We first import the module *Config* and read the *spm.conf* to load the SPM-VMS list.

```
# Import the spm conf file for reading
from lib.cuckoo.common.config import Config
spm_conf = Config("spm")
```

Next, we add the argument *–spm* to the argument parser of the *submit.py*, making it accessible from the command line.

```
# adding the SPM argument >> execute the command [virsh start {VM_NAME}]
parser.add_argument("--spm", action="store_true", default=False, help="Spin other VMs for Self-Propagating-Malware SPM", required=False)
```

The last step is to find the code line where the submission occurs. The following code snippet is added after the check for remote is false. We do a for loop and spin all the SPM-VMs using the KVM-QEMU command.

```
if args.remote:
    ......
else:
    if args.spm:
        vms_list = spm_conf.spm.machines.split(",")
        for i in vms_list:
            print(f"VM name: {i}")
            subprocess.run(['virsh','snapshot-revert','--domain',f"{i}","--snapshotname","snapshot1"])
```

Regarding the safety of the clean VMs, if they get infected by the malware, we run only a snapshot of the VM. This will ensure any memory change is discarded by the hypervisor. Finally, we automate the Sandbox's submission process. We provide the *spm_submit.sh* script to do so. The script will prompt about the file or the folder path, then execute the submission. It is worth mentioning that this SPM automation is only feasible on the VMs as we have complete control. While this is not the case regarding the physical hosts. I.e., we cannot turn on a computer remotely, limiting the automation. Nevertheless, the machines can go to sleep mode to save energy, and FOG can use WAKE-ON-LAN API to bring them back to operational mode.

```
1  echo "Please provide the sample path:"
2  read path
3  echo "Submit To specific Machine? Y:N"
4  read x
5  if [ "$x" = "Y" ]; then
6    echo "Provide machine name"
7    cd /opt/CAPEv2
8    read y
9    poetry run python3 utils/submit.py --spm --enforce-timeout --
         ↪ Machine $y $path
10 else
11   cd /opt/CAPEv2
12   poetry run python3 utils/submit.py --spm --enforce-timeout $path
13 fi
```

5.4.5 Milestone 5: Prepare the backup tool

FOG Project has a client-server architecture. The server is responsible for management and storage, while the client links the physical machine to the server. The server has to be configured and running before the client can register. FOG Server is made to be installed on RedHat-based and Debian-based distributions such as CentOS, Fedora, or Ubuntu. We chose Ubuntu, an OS we have experience with, specifically version 20.04, as recommended by the FOG Project itself. We are ready to start installing the FOG server when our base distribution is installed. The installation steps are listed in the appendix [E.1]. When the server is up and running, we can install the FOG client and prepare physical machines for imaging and recovery by following the steps specified in the appendix [E.2]. When all those steps are performed, we can run malware on our physical machines and restore them to a clean state.

5.4.6 Milestone 6: Prepare the monitoring tool

To set up our monitoring tool, we first need to create a VM in VirtualBox with at least the following requirements: 12GB RAM, 4 CPU cores, 200GB of storage, and 2 NICs. As explained in section 5.3.4, one NIC has to be in promiscuous mode and allow sniffing of all machines, as shown in the appendix [F.1]. That will be the interface we use for monitoring. The other interface does not need that and will be used for management. Both interfaces are in bridge mode. VirtualBox's bridged networking creates a new network interface in software. The host can send data to the guest through that interface and receive data from it. This is very

convenient in our case, as it enables us to have multiple VMs on the same host with different physical IP addresses and can communicate with one another and the host, allowing us to circumvent certain hardware limitations.

By trial and error, we found that the installation must be done on VirtualBox to circumvent the hardware limitations we face. This is because KVM's bridge mode is different from VirtualBox. Security Onion's official documentation also recommends VirtualBox or VMWare as hypervisors of choice.

However, this leads to software limitations. Indeed, as KVM and VirtualBox are two different hypervisors, they can not run simultaneously on one machine. As KVM is used for the virtualized system, the VirtualBox Security Onion VM has to run on another physical system. Hence, the virtual system's real-time monitoring is impossible for Security Onion. Nonetheless, we can still capture the traffic using the CAPEv2 Sandbox and tcpdump, and Security Onion can still be used for forensics.

When the VM settings are done, we can start the installation of Security Onion. That can be done on a CentOS 7 or Ubuntu 20 OS or using the Security Onion ISO directly. The complete installation and configuration steps are in the appendix [F.2].

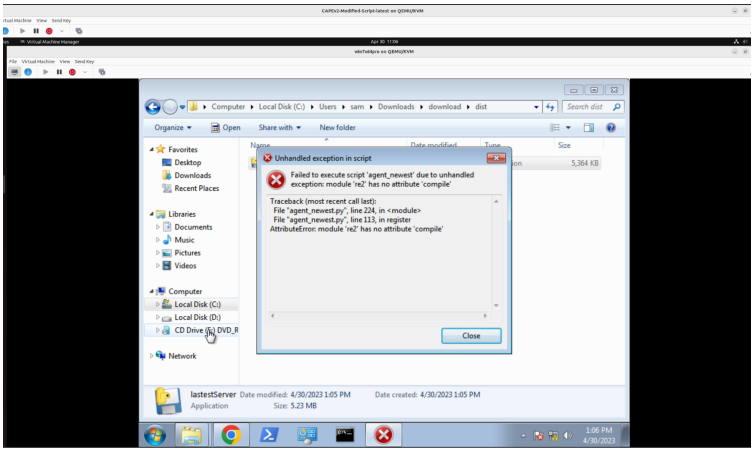
Finally, we recommend backing up all the critical machines' memory to have a backup in case of memory damage due to malware propagation. Namely the CAPEv2 Sandbox machine and the Security-Onion ones. Moreover, we recommend using DeepFreeze or backing up the ready FOG server machine into an isolated hard drive for security purposes. This way, we avoid losing all the backups if the FOG machine is damaged.

5.5 Test and Validation

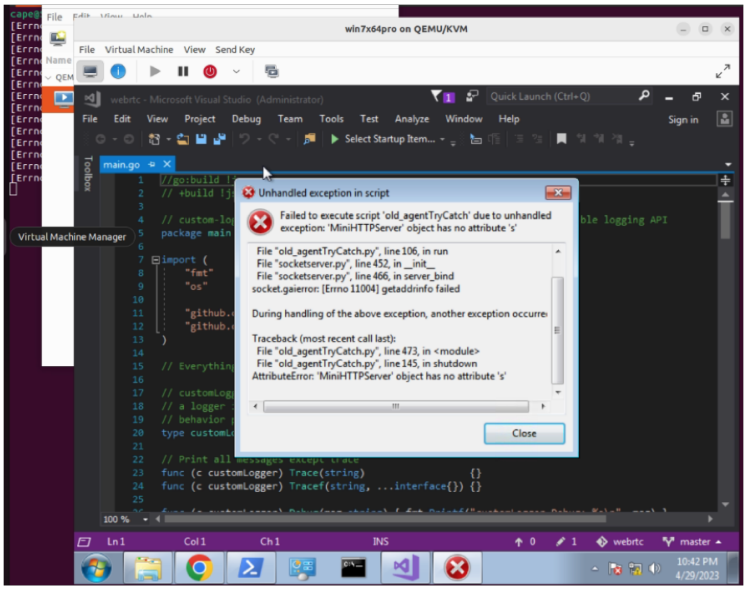
Test and validation are crucial steps in our system implementation. It helps us identify eventual bugs and errors, as well as ensuring that the system is reliable and consistent. To ensure that we walk through all the components test, we follow the chronological order in which they are introduced.

5.5.1 Sandbox Test

The testing steps are listed in chronological order. The first step is to test the Sandbox itself. Firstly, we test that the sample programs are submitted and executed. This will ensure that the Sandbox runs appropriately from start to end and ensure accurate analysis results. Namely, we encountered inaccurate data several times due to an unhandled or missing package in the testbed, interrupting the analysis as shown below.



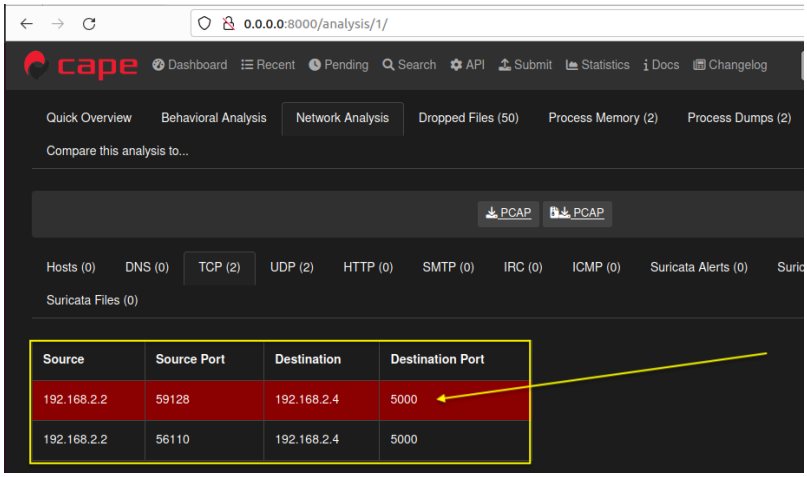
(a) RE2 conflict error



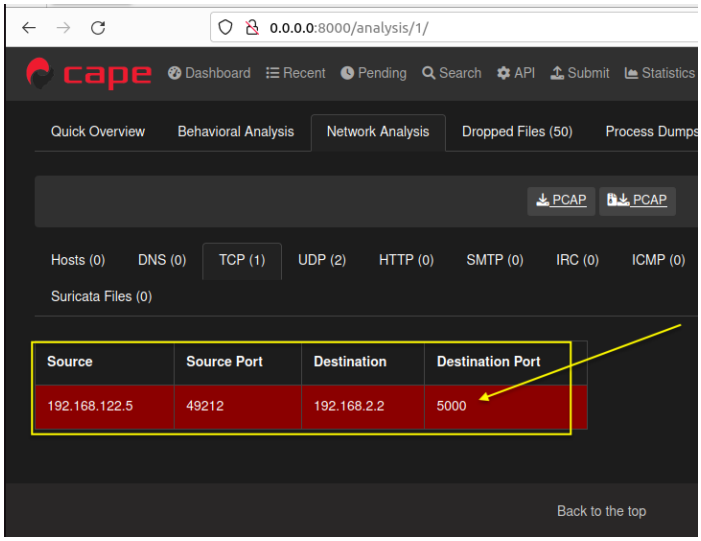
(b) Networking related exceptions

Figure 5.10: VM testbed unhandled exceptions

We built a small application that simulates malware to avoid such a scenario. A client-server application. The server application is submitted to the Sandbox for analysis. While the client is continuously attempting to connect every 2 seconds to the testbed machine. Once the server executes on the testbed machine, it opens port 5000, and a TCP connection is established, simulating the Control and Command (CnC) used by malware. As the server receives a connection, it replies with a defined message. Indicating that the sample was executed on the testbed. Furthermore, we collect the network traffic during the analysis as a dataset.



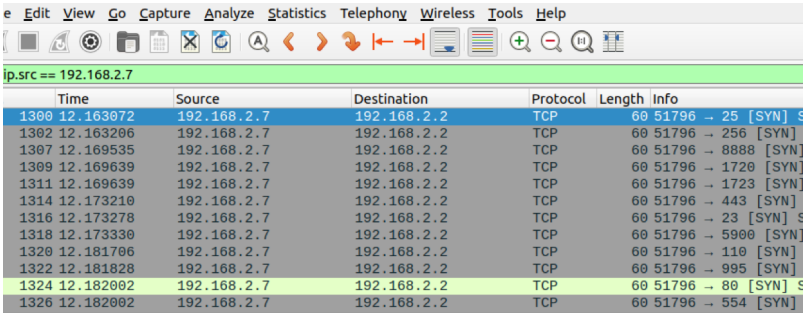
(a) Physical Machine testbed



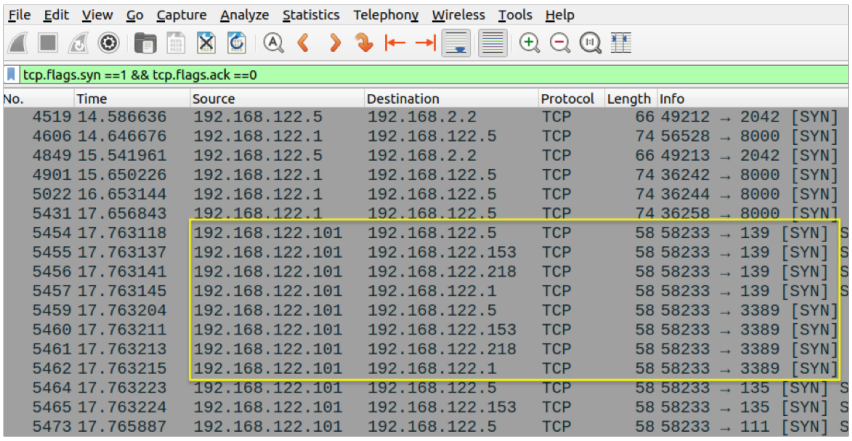
(b) Virtual Machine testbed

Figure 5.11: Server-Client Sandbox execution proof

After ensuring a working Sandbox, the next step is to test for our SPM integration. Before running malware, we decided to make an nmap scan to simulate malware scanning the network. The virtual network consists of the following hosts: The testbed VM with the only static IP 192.168.122.5 and 3 other VMs. All the other hosts are left to use DHCP to obtain an IP from the hypervisor. We use the Ubuntu server at [192.168.122.101] to scan the virtual network while we submit the Sever-Client application for analysis. We verify the network traffic and see that the nmap scan can be seen as shown below. For instance, the scan on port 139. For other functionalities, such as detection and other plugins, we append figures showing the results of the tests G.3. It is worth mentioning that the physical testbed does not display the entire network traffic due to the LAN-Switch limitation and will not display traffic directed to other hosts.



(a) Physical Machine testbed

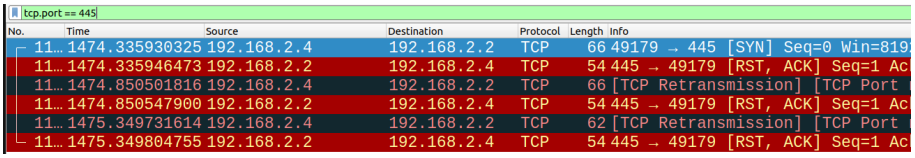


(b) Virtual Machine testbed

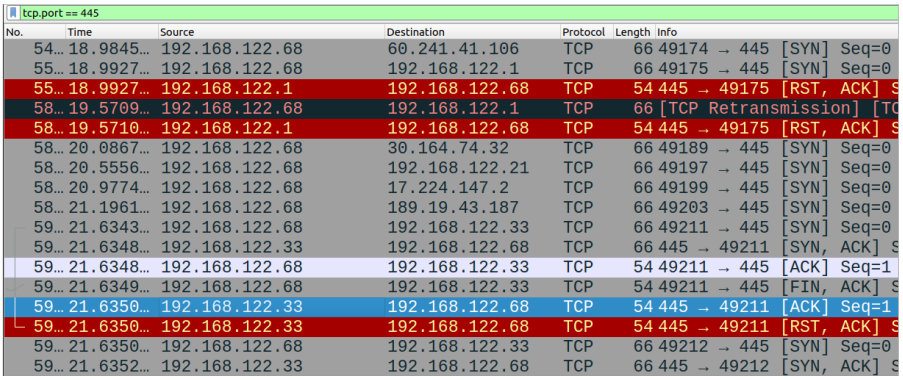
Figure 5.12: Nmap scan network proof

Moreover, we test the Sandbox with the two selected malware. This ensures that

the Sandbox detects the malicious behavior of malware and labels it accordingly. Additionally, we test with the same malware and compare the behavior to determine the results' accuracy and the Sandbox's functionality. As displayed below, we see the malware scanning behavior clearly. Additionally, we see the exact domain name resolution *www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com*, which is a kill-switch designed to check for network connectivity and detect DNS Sinkholes.

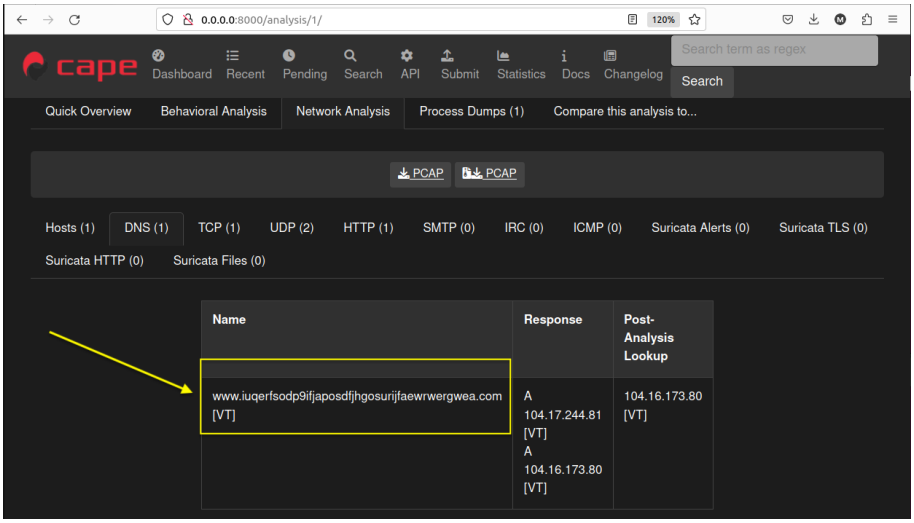


(a) Physical Testbed

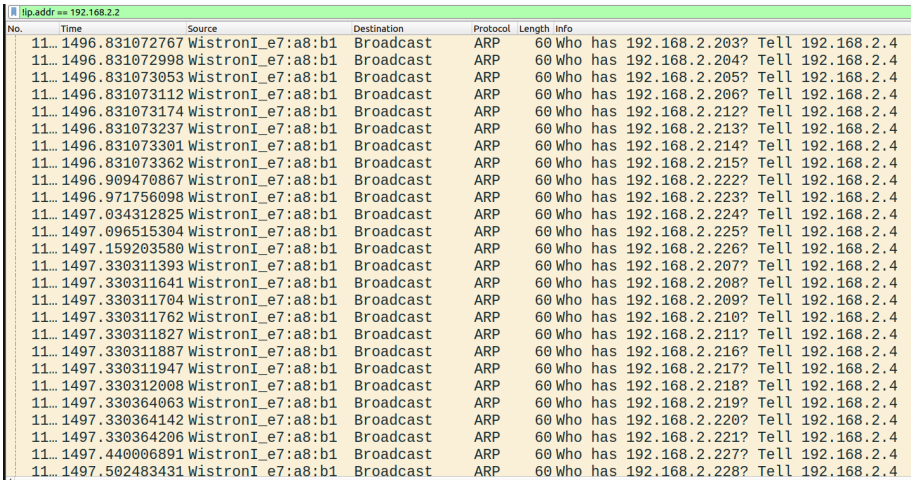


(b) VM testbed

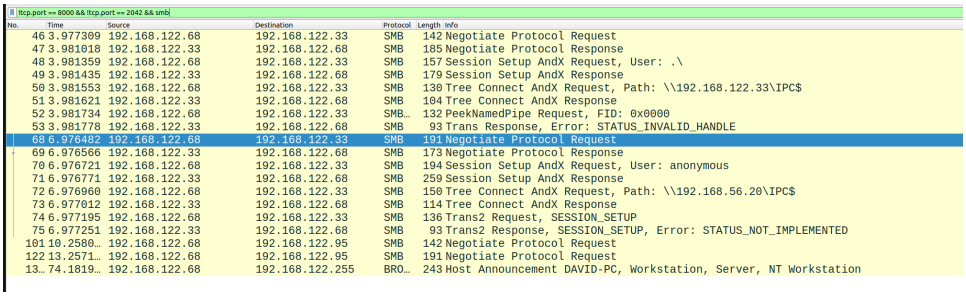
Figure 5.13: Wannacry TCP sweep-scan



(a) DNS kill-switch



(b) ARP scan

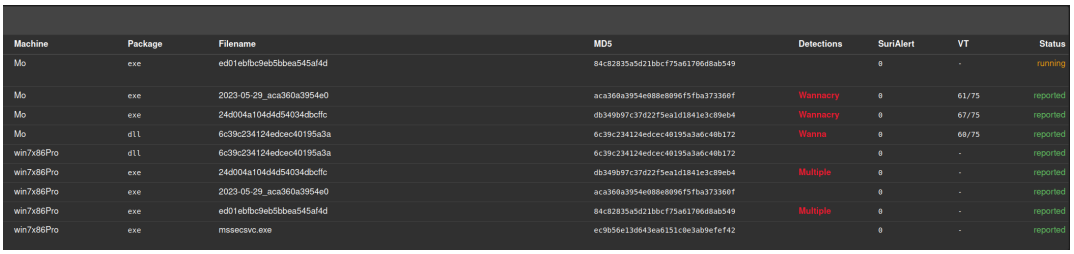


(c) SMB connection attempts

Figure 5.14: Wannacry scanning sophistication

We can observe the exact behavior of the malware on both, as the scanning method is different on the local network. The Worm uses the ARP protocol to scan the network and initiate connections to the available ones using the SMB protocol.

The last test was to verify if the different platform analyses could also be used to identify the evasive behavior of the malware. We did test the two selected malware and other variants of WannaCry on both platforms. We observed a higher VirusTotal score and accurate labeling when using the physical machine. We relate this to the malware's evasive techniques against the VM platform. We recommend adopting the same approach of testing malware on different platforms. Namely, to have different data to compare and to discover if the malware is equipped with evasive techniques.



Machine	Package	Filename	MD5	Detections	SuriAlert	VT	Status
Mo	exe	ed01ebfbc9eb5bbea545af4d	84c82835a5d21bbcf75a61706d8ab549		0	-	running
Mo	exe	2023-05-29_aca360a3954e0	aca360a3954e088e8096f5fba373360f	Wannacry	0	61/75	reported
Mo	exe	24d004a104d4d54034dbcffc	db349b97c37d22f5ea1d1841e3c89eb4	Wannacry	0	67/75	reported
Mo	dll	6c39c234124edcec40195a3a	6c39c234124edcec40195a3a6c40b172	Wanna	0	60/75	reported
win7x86Pro	dll	6c39c234124edcec40195a3a	6c39c234124edcec40195a3a6c40b172		0	-	reported
win7x86Pro	exe	24d004a104d4d54034dbcffc	db349b97c37d22f5ea1d1841e3c89eb4	Multiple	0	-	reported
win7x86Pro	exe	2023-05-29_aca360a3954e0	aca360a3954e088e8096f5fba373360f		0	-	reported
win7x86Pro	exe	ed01ebfbc9eb5bbea545af4d	84c82835a5d21bbcf75a61706d8ab549	Multiple	0	-	reported
win7x86Pro	exe	mssecsvc.exe	ec9b56e13d643ea6151c0e3ab9efef42		0	-	reported

Figure 5.15: Physical Vs VM test results comparison

It is worth mentioning that the physical machine is ideal for evasive techniques as it produces fewer artifacts than the VM. However, the Sandbox does not have complete control over the machine. This can be confirmed when submitting the Wannacry variants on the physical machine. After a few seconds from the malware execution, the CAPEv2 fails to reboot the device via the FOG task. This leads to the machine hanging in the Shutting-Down screen. We believe this is due to the malware disabling something or scheduling a task the system fails to kill. To fix this, we need to force the shutdown of the physical machine to complete the FOG restoration procedure. Followed by a Sandbox reset to allow pending task if any:

```
1    sudo systemctl restart cape.service cape-processor.service cape
       ↪ -rooter.service cape-web.service
```

5.5.2 FOG test

In order to test if FOG is working as intended, we first test the capture and deploy process on a single machine. These can be launched through the FOG Web Interface.

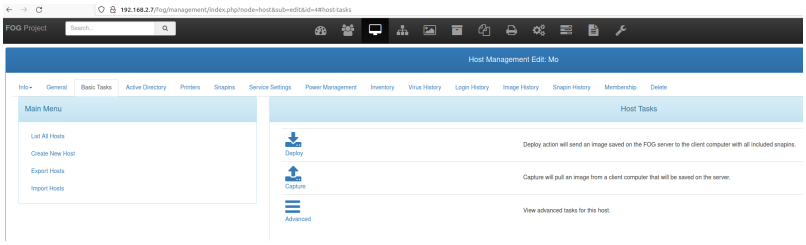


Figure 5.16: Host task scheduler - FOG Server

When those operations are successful, we can move on and test FOG's integration with CAPEv2. We change CAPEv2's configuration to use a physical machine. However, we submitted the server-client application used to test the Sandbox instead of submitting malware. CAPEv2 does the analysis, and when it timeouts, it notifies the FOG server, which commands the target machine to reboot and restore the system to the captured state. We know this step is successful by looking at the target machine in its restoration phase, as shown in figure [5.17], and the CAPEv2 logs. The logs show the interaction between CAPEv2 and FOG.

Listing 5.2: CAPE service logs for the Physical testbed

```
1  # -- Machine reboot logs --
2  May 31 11:42:09 Station0 python3[49427]: DEBUG:modules.machinery.
       ↪ physical:Socket killed from analysis machine due to reboot
3  May 31 11:42:09 Station0 python3[49427]: 2023-05-31 11:42:09,680 [
       ↪ modules.machinery.physical] DEBUG: Socket killed from
       ↪ analysis machine due to reboot
4  # -- Restore operation logs --
5  May 31 11:42:09 Station0 python3[49427]: DEBUG:modules.machinery.
       ↪ physical:Restore operation for Mo still running
6  May 31 11:42:29 Station0 python3[49427]: 2023-05-31 11:42:29,905 [
       ↪ modules.machinery.physical] DEBUG: Restore operation for Mo
       ↪ still running
7  May 31 13:03:50 Station0 python3[49427]: DEBUG:lib.cuckoo.core.
       ↪ resultserver:Task 8: Stopped tracking machine 192.168.2.4
8  May 31 13:03:50 Station0 python3[49427]: INFO:lib.cuckoo.core.
       ↪ scheduler:Disabled route 'internet'
9  # -- Analysis logs --
10 May 31 13:03:50 Station0 python3[49427]: DEBUG:lib.cuckoo.core.
       ↪ scheduler:Task 8: Released database task with status True
11 May 31 13:03:50 Station0 python3[49427]: INFO:lib.cuckoo.core.
       ↪ scheduler:Task 8: analysis procedure completed
```

During our first tests with the FOG backup solution, we noticed that capturing and deploying images took a long time. This was because we captured and deployed the whole C drive of the machine, taking up to 100GB of storage and more than one hour to complete. To shrink the size of the images and speed up the process, we decided to split the hard drive into two different drives, namely C and D. With C drive having enough storage to support Windows 7 installation and all the software for the sandbox. While D has the rest storage. This has reduced the image size to 21GB (after compression) and the time to complete a task to less than 5 minutes, as shown in figure [5.17]. However, this approach will trigger one evasive technique known by the community, where an evasive sample will check the machine's storage size and raise a flag if the storage is less than 60GB [95]. This is a snapshot of the Pafish trace on our physical machine.

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

Listing 5.3: Pafish trace on the physical testbed

However, it will only trigger 1 out of 2, as shown above. The only solution to remove this artifact is to sacrifice the restoration speed and upgrade the C drive to over 60GB.

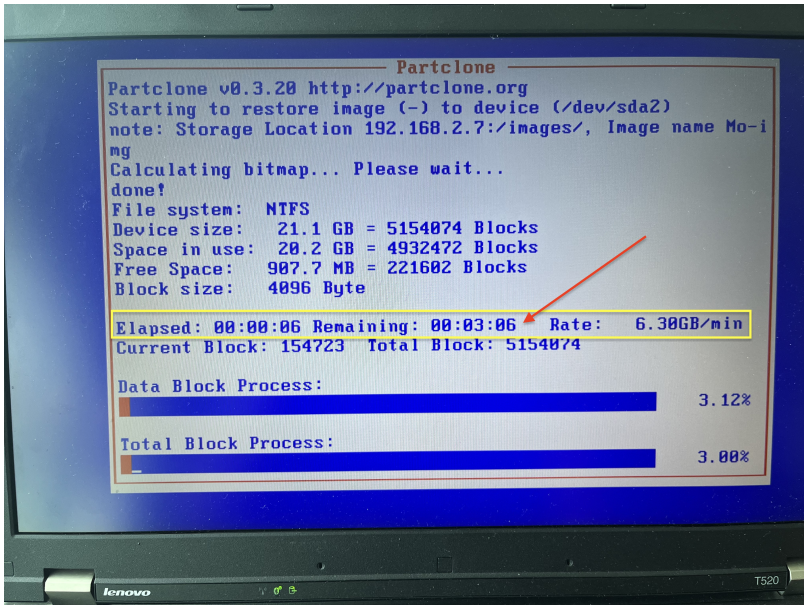


Figure 5.17: Restoring the target machine

5.5.3 Security Onion Test

To test our monitoring tool, we want to ensure it works correctly. To do so, we can check the Security Onion Console and the status command output. The Security Onion Console shows that the expected machines are present according to their IP addresses, as shown in the following image. That means that Security Onion effectively monitors the physical network in real-time.

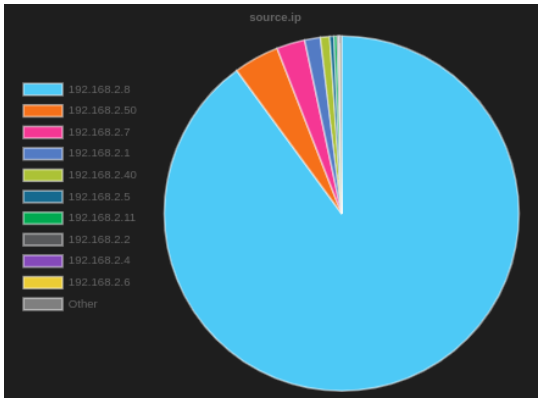


Figure 5.18: Identified IPs from Security Onion

The IP address *192.168.2.8* is the most present as it is the physical machine on which Security Onion is installed, and it acts as a gateway for the Security Onion

VM.

Second, we type the following command in the Security Onion VM.

```
1 sudo so-status
```

Listing 5.4: Command to see the status of our Security Onion machine

That command gives us a positive result, as shown in the appendix [G.1], which means everything works as intended.

The next thing is to test to which extent it is possible to monitor the traffic from the Security Onion VM. To do so, we use a classic nmap scan from a physical device in the network.

```
1 sudo nmap -sS -Pn 192.168.2.0/24
```

Listing 5.5: Nmap scan command

That command will tell nmap to scan the whole network, replicating one of the possible malware behavior. In appendix [G.2], which shows screenshots from the Security Onion Console, we can see the machine with the IP address *192.168.2.2* is scanning multiple ports of the device with IP address *192.168.2.8*. Furthermore, we can see alerts relating to the same network scan. Hence, the scanning behavior is indeed detected in Security Onion.

For the virtual system, as it is impossible to have Security Onion monitor the virtual network due to hardware and software limitations, we use the tcpdump command as follows:

```
1 tcpdump -i [interface] -w [filename].pcap
```

Listing 5.6: Tcpdump command

Tcpdump will capture full-size packets on the specified interface and write them into the specified file. Despite being unable to capture the traffic from the virtual system in real-time, Security Onion can be used after the analysis to analyze pcap files captured on the virtual system.

For the physical system, monitoring the traffic from the whole network is impossible due to hardware limitations, as specified in section [5.3.4]. The only scanning behavior we can monitor is to our specific machine, *192.168.2.8*, as shown in the appendix [G.2].

Finally, we tested Security Onion with real malware samples that we selected. Security Onion successfully flags the malicious behavior and generates expected alerts. The malware is identified as a WannaCry variation, using Eternal Blue vulnerabilities.

	count	rule.name	event.module	event.severity_label
	1	ET EXPLOIT Possible ETERNALBLUE Probe MS17-010 (Generic Flags)	suricata	high
	1	ET EXPLOIT Possible ETERNALBLUE Probe MS17-010 (MSF style)	suricata	high
	2	ET MALWARE Known Sinkhole Response Kryptos Logic	suricata	low
	1	ET MALWARE Possible WannaCry DNS Lookup 1	suricata	high
	2	ET MALWARE W32/WannaCry.Ransomware Killswitch Domain HTTP Request 1	suricata	high

Figure 5.19: Security Onion generated alerts - Real Malware

The user can further investigate those alerts. The information found is the source IP and port, the destination IP and port, the exact packet(s) that triggered the rule, the rule, and the timestamp, among others. As mentioned in section [5.3.4], those logs are stored in the Elastic Stack, which is responsible for keeping all logs and making them searchable. The Security Onion web interface incorporates Kibana, a data visualization tool for the Elastic Stack. Data is searchable and can be exported into a desired format for future use by a machine learning model.

Chapter 6

Discussion & Conclusion

The sandbox technology is very promising, especially with its both testbed possibility. This research studies the different behavior of malware on various platforms and identifies evasive malware. Moreover, it is a complex system and can sometimes be a cumbersome process to set up and tune, or at least we encountered many issues after the installation phase. Nevertheless, the time can be reduced to only an hour, saving the researchers time on fine-tuning instead of debugging the Sandbox. Moreover, we guarantee that our documentation will result in a fully functional Sandbox with the SPM integration for both VM and Physical Sandbox. Furthermore, the Sandbox is still in progress and many new tools have been added, making it an excellent platform for SPM analysis. Nevertheless, we recommend spending more time fine-tuning the Sandbox and eliminating artifacts that could be detected by the malware and hinder the analysis results, precisely any artifact related to the network setup. –For instance, MAC addresses for the VMs, the Virtual Gateway, and the VM hostnames to simulate a subnet with a router. The better simulation, the higher chance of deceiving the malware. We acknowledge that we did not focus on introducing a Linux or another OS testbed. Nevertheless, we do not think it is a crucial requirement and could be planned as future work. Regarding Sandbox's test, we successfully tested its functionality and provided evidence of bug-free runtime. Eventually, this guarantees a successful malware analysis. Additionally, the Sandbox has passed successful malware tests on the physical and virtual testbeds and has recovered entirely from the WannaCry disk damage. Indicating a well-functioning FOG imaging server and ensuring the system's robustness. Moreover, the Sandbox offers the network traffic as a dataset to be saved and processed with the monitoring tool of choice.

Eventually, we successfully implemented a test environment with multiple hosts on the LAN. The test environment is also reproducible and effectively collects network evidence from SPM analysis. Furthermore, it is heterogeneous as it is compromised of various versions of different operating systems. Software

and hardware limitations significantly limit the Security-Onion monitoring tool, rendering its use disputable. However, if the constraints are overcome, it is an excellent addition to the dynamic malware analysis system. Indeed, it enables the system to collect more logs and perform forensics analysis on that out-of-the-box.

Finally, for the SPM analysis system, we believe the AAU LAB should consider purchasing professional network monitoring tools that could be easily integrated. For example, A LAN switch with port mirroring capability would reduce the complexity of the analysis system and the network. Moreover, the Sandbox-SPM integration is feasible and can offer more data than building the analysis system from scratch. Finally, we suggest the enhancement of the Sandbox as future work to minimize the artifacts produced by the Sandbox installation and communication. –For instance, the FOG imaging server and the CAPEv2 Sandbox use non-encrypted communication (HTTP/FTP). While an encrypted communication will obfuscate the payload, and no malware cannot be fingerprint it.

Bibliography

[1] AAU-Net-Sec. *HosTaGe*. 2023. URL: https://aau-network-security.github.io/HosTaGe/.

[2] Adel Abusitta, Miles Q. Li, and Benjamin C.M. Fung. "Malware classification and composition analysis: A survey of recent developments". In: *Journal of Information Security and Applications* 59 (2021), p. 102828. ISSN: 2214-2126. DOI: https://doi.org/10.1016/j.jisa.2021.102828. URL: https://www.sciencedirect.com/science/article/pii/S2214212621000648.

[3] *ADVANCED PERSISTENT THREAT (APT)*. 2023. URL: https://www.crowdstrike.com/cybersecurity-101/advanced-persistent-threat-apt/.

[4] *Advanced Persistent Threats (APTs)*. 2023. URL: https://www.mandiant.com/resources/insights/apt-groups.

[5] Amir Afianian et al. "Malware Dynamic Analysis Evasion Techniques: A Survey". In: *ACM Comput. Surv.* 52.6 (Nov. 2019). ISSN: 0360-0300. DOI: 10.1145/3365001. URL: https://doi.org/10.1145/3365001.

[6] Maxat Akbanov, Vassilios G. Vassilakis, and Michael D. Logothetis. "Ransomware detection and mitigation using software-defined networking: The case of WannaCry". In: *Computers & Electrical Engineering* 76 (2019), pp. 111–121. ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2019.03.012. URL: https://www.sciencedirect.com/science/article/pii/S0045790618323164.

[7] RANDI EITZMAN ALEX BERRY JOSH HOMAN. *WannaCry Malware Profile*. 2021. URL: https://www.mandiant.com/resources/blog/wannacry-malware-profile.

[8] Kamal Alieyan et al. "Botnet and Internet of Things (IoTs): A definition, taxonomy, challenges, and future directions". In: *Research Anthology on Combating Denial-of-Service Attacks*. IGI Global, 2021, pp. 138–150.

[9] Ahmad O. Almashhadani et al. "A Multi-Classifier Network-Based Crypto Ransomware Detection System: A Case Study of Locky Ransomware". In: *IEEE Access* 7 (2019), pp. 47053–47067. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2907485.

[10] Fahad M. Alotaibi and Vassilios G. Vassilakis. "SDN-Based Detection of Self-Propagating Ransomware: The Case of BadRabbit". In: *IEEE Access* 9 (2021), pp. 28039–28058. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3058897.

[11] Originally Jeffrey Beall updated by anonymous. *Beall's list*. 2021. URL: https://beallslist.net/.

[12] *Anti-debugging and anti-VM techniques and anti-emulation [updated 2019]*. URL: https://resources.infosecinstitute.com/topic/anti-debugging-and-anti-vm-techniques-and-anti-emulation/.

[13] Kalyan Anumula and Joseph Raymond. "Adware and Spyware Detection Using Classification and Association". In: *Proceedings of International Conference on Deep Learning, Computing and Intelligence: ICDCI 2021*. Springer. 2022, pp. 355–361.

[14] Mike Azzara. *What is WannaCry Ransomware and How Does It Work?* May 2021. URL: https://www.mimecast.com/blog/all-you-need-to-know-about-wannacry-ransomware/.

[15] Bojana Bakić et al. "10 years since Stuxnet: What have we learned from this mysterious computer software worm?" In: *2021 25th International Conference on Information Technology (IT)*. Feb. 2021, pp. 1–4. DOI: 10.1109/IT51528.2021.9390103.

[16] Farrah Kristel Batista, Angel Martín del Rey, and Araceli Queiruga-Dios. "A New Individual-Based Model to Simulate Malware Propagation in Wireless Sensor Networks". In: *Mathematics* 8.3 (2020). ISSN: 2227-7390. DOI: 10.3390/math8030410. URL: https://www.mdpi.com/2227-7390/8/3/410.

[17] Farrah Kristel Batista, Angel Martín del Rey, and Araceli Queiruga-Dios. "A Review of SEIR-D Agent-Based Model". In: *Distributed Computing and Artificial Intelligence, 16th International Conference, Special Sessions*. Ed. by Enrique Herrera-Viedma et al. Cham: Springer International Publishing, 2020, pp. 133–140. ISBN: 978-3-030-23946-6.

[18] PAUL BISCHOFF. *Ransomware attacks cost the US 159.4bn in downtime alone in 2021*. July 2022. URL: https://www.comparitech.com/blog/information-security/us-ransomware-attacks-cost/.

[19] Adam Borys et al. "An Evaluation of IoT DDoS Cryptojacking Malware and Mirai Botnet". In: *2022 IEEE World AI IoT Congress (AIIoT)*. June 2022, pp. 725–729. DOI: 10.1109/AIIoT54504.2022.9817163.

[20] Azzedine Boukerche and Qi Zhang. "Countermeasures against Worm Spreading: A New Challenge for Vehicular Networks". In: *ACM Comput. Surv.* 52.2 (May 2019). ISSN: 0360-0300. DOI: 10.1145/3284748. URL: https://doi-org.zorac.aub.aau.dk/10.1145/3284748.

[21] Krzysztof Cabaj and Wojciech Mazurczyk. "Using software-defined networking for ransomware mitigation: the case of cryptowall". In: *Ieee Network* 30.6 (2016), pp. 14–20.

[22] CAPEv2. *Requirements*. 2023. URL: https://capev2.readthedocs.io/en/latest/installation/guest/requirements.html#requirements.

[23] Evasion Checkpoint. *Check if particular registry paths exist*. 2021. URL: https://evasions.checkpoint.com/techniques/registry.html\#check-if-particular-registry-paths-exist.

[24] Xu Chen et al. "Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware". In: *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. June 2008, pp. 177–186. DOI: 10.1109/DSN.2008.4630086.

[25] Alesia Chernikova et al. "Cyber Network Resilience Against Self-Propagating Malware Attacks". In: *Computer Security – ESORICS 2022*. Ed. by Vijayalakshmi Atluri et al. Cham: Springer International Publishing, 2022, pp. 531–550. ISBN: 978-3-031-17140-6.

[26] CISA. *2021 Trends Show Increased Globalized Threat of Ransomware*. 2022. URL: https://www.cisa.gov/news-events/cybersecurity-advisories/aa22-040a.

[27] Cisco. *Catalyst 3750 Switch Software Configuration Guide, 12.2(52)SE*. 2023. URL: https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3750/software/release/12-2_52_se/configuration/guide/3750scg/swspan.html.

[28] Ryan Clancy. *A Quick Guide to Reverse Engineering Malware*. 2022. URL: https://www.eccouncil.org/cybersecurity-exchange/ethical-hacking/malware-reverse-engineering/.

[29] cuckoosandbox. *Cuckoo*. 2023. URL: https://github.com/cuckoosandbox/cuckoo#readme.

[30] *CVE-2017-0144 Detail*. 2018. URL: https://nvd.nist.gov/vuln/detail/CVE-2017-0144.

[31] *CVE-2021-44228 Detail*. 2018. URL: https://nvd.nist.gov/vuln/detail/CVE-2021-44228.

[32] Xiyue Deng and Jelena Mirkovic. "Polymorphic Malware Behavior Through Network Trace Analysis". In: *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. 2022, pp. 138–146. DOI: 10.1109/COMSNETS53615.2022.9668396.

[33] *DhcpEnumSubnetClients function (dhcpsapi.h)*. 2021. URL: https://learn.microsoft.com/en-us/windows/win32/api/dhcpsapi/nf-dhcpsapi-dhcpenumsubnetclients.

[34] *DhcpEnumSubnets function (dhcpsapi.h)*. 2021. URL: https://learn.microsoft.com/en-us/windows/win32/api/dhcpsapi/nf-dhcpsapi-dhcpenumsubnets.

[35] *DhcpGetSubnetInfo function (dhcpsapi.h)*. 2021. URL: https://learn.microsoft.com/en-us/windows/win32/api/dhcpsapi/nf-dhcpsapi-dhcpgetsubnetinfo.

[36] doomedraven. *choco.bat*. 2023. URL: https://github.com/kevoreilly/CAPEv2/blob/master/installer/choco.bat.

[37] doomedraven. *disable_win7noise1.bat*. 2023. URL: https://github.com/doomedraven/Tools/blob/master/Windows/disable_win7noise.bat.

[38] Peter Eder-Neuhauser, Tanja Zseby, and Joachim Fabini. "Malware propagation in smart grid networks: metrics, simulation and comparison of three malware types". In: *Journal of Computer Virology and Hacking Techniques* 15 (2019), pp. 109–125. ISSN: 2263-8733. DOI: 10.1007/s11416-018-0325-y.

[39] Christopher C Elisan. *Malware, Rootkits & Botnets A Beginner's Guide*. McGraw Hill Professional, 2012.

[40] ENISA. *2023 SonicWall Cyber Threat Report*. 2023. URL: https://www.sonicwall.com/medialibrary/en/white-paper/2023-cyber-threat-report.pdf.

[41] ENISA. *ENISA Threat Landscape 2022*. 2022. URL: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2022.

[42] *Evasion techniques*. 2022. URL: https://evasions.checkpoint.com/.

[43] Faronics. *Deep Freeze*. 2023. URL: https://www.faronics.com/en-uk/products/deep-freeze.

[44] Sylvia Feng and Muharman Lubis. "Defense-In-Depth Security Strategy in Log4j Vulnerability Analysis". In: *2022 International Conference Advancement in Data Science, E-learning and Information Systems (ICADEIS)*. IEEE. 2022, pp. 01–04.

[45] Getoar Gallopeni et al. "A practical analysis on mirai botnet traffic". In: *2020 IFIP Networking Conference (Networking)*. IEEE. 2020, pp. 667–668.

[46] Getoar Gallopeni et al. "A Practical Analysis on Mirai Botnet Traffic". In: *2020 IFIP Networking Conference (Networking)*. June 2020, pp. 667–668.

[47] *GetExtendedTcpTable function (iphlpapi.h)*. 2021. URL: https://learn.microsoft.com/en-us/windows/win32/api/iphlpapi/nf-iphlpapi-getextendedtcptable.

[48] *GetIpNetTable function (iphlpapi.h)*. 2021. URL: https://learn.microsoft.com/en-us/windows/win32/api/iphlpapi/nf-iphlpapi-getipnettable.

[49] Google. *re2*. 2023. URL: https://github.com/google/re2.

[50] Nana Kwarne Gyamfi and Ebenezer Owusu. "Survey of mobile malware analysis, detection techniques and tool". In: *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE. 2018, pp. 1101–1107.

[51] Raphael Hiesgen et al. "The race to the vulnerable: Measuring the log4j shell incident". In: *arXiv preprint arXiv:2205.02544* (2022).

[52] Erik Hjelmvik. *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.

[53] *HTTP PROPFIND Method*. 2015. URL: https://learn.microsoft.com/en-us/previous-versions/office/developer/exchange-server-2003/aa142960(v=exchg.65).

[54] IBM. *How much does a data breach cost?* 2021. URL: https://www.ibm.com/security/data-breach.

[55] *IP Addressing: ARP Configuration Guide, Cisco IOS Release 15M&T*. 2012. URL: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipaddr_arp/configuration/15-mt/arp-15-mt-book/arp-config-arp.html.

[56] Vasileios Karyotis and M.H.R. Khouzani. "Chapter 3 - Early malware diffusion modeling methodologies". In: *Malware Diffusion Models for Wireless Complex Networks*. Ed. by Vasileios Karyotis and M.H.R. Khouzani. Boston: Morgan Kaufmann, 2016, pp. 39–60. ISBN: 978-0-12-802714-1. DOI: https://doi.org/10.1016/B978-0-12-802714-1.00013-X. URL: https://www.sciencedirect.com/science/article/pii/B978012802714100013X.

[57] kevoreilly. *CAPEv2*. 2023. URL: https://capev2.readthedocs.io/en/latest/.

[58] *KEYLOGGER*. 2023. URL: https://www.malwarebytes.com/keylogger.

[59] Mijoo Kim et al. "A study on behavior-based mobile malware analysis system against evasion techniques". In: *2016 International Conference on Information Networking (ICOIN)*. Jan. 2016, pp. 455–457. DOI: 10.1109/ICOIN.2016.7427158.

[60] Joxean Koret and Elias Bachaalany. *The antivirus hacker's handbook*. John Wiley & Sons, 2015.

[61] Ayush Kumar and Teng Joon Lim. "EDIMA: Early Detection of IoT Malware Network Activity Using Machine Learning Techniques". In: *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. Apr. 2019, pp. 289–294. DOI: 10.1109/WF-IoT.2019.8767194.

[62] Jun Li, Devkishen Sisodia, and Shad Stafford. "On the Detection of Smart, Self-Propagating Internet Worms". In: *IEEE Transactions on Dependable and Secure Computing* (2022), pp. 1–13. ISSN: 1941-0018. DOI: 10.1109/TDSC.2022.3194127.

[63] Songsong Liu et al. "Enhancing malware analysis sandboxes with emulated user behavior". In: *Computers & Security* 115 (2022), p. 102613. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2022.102613. URL: https://www.sciencedirect.com/science/article/pii/S0167404822000128.

[64] Wanping Liu and Shouming Zhong. "Web malware spread modelling and optimal control strategies". In: *Scientific reports* 7.1 (2017), pp. 1–19.

[65] Yehonatan Lusky and Avi Mendelson. "Sandbox Detection Using Hardware Side Channels". In: *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. Apr. 2021, pp. 192–197. DOI: 10.1109/ISQED51717.2021.9424260.

[66] *MAC Address Lookup*. 2023. URL: https://maclookup.app/search/result?mac=F8:D1:11.

[67] KME Narasima Mallikarajunan et al. "Detection of spyware in software using virtual environment". In: *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE. 2019, pp. 1138–1142.

[68] *MalwareBazaar*. 2023. URL: https://bazaar.abuse.ch/.

[69] Rajkumar Maurya. *How to bridge WiFi and Ethernet adapters to share internet on Windows 11 or 10*. 2021. URL: https://www.how2shout.com/how-to/bridge-wifi-to-ethernet-adapter-to-share-internet.html.

[70] Dana Mayor. *RC: History, Origin, and More*. 2022. URL: https://history-computer.com/irc-guide/.

[71] Norita Md Norwawi et al. "Detection Algorithm for Internet Worms Scanning that Used User Datagram Protocol". In: *International Journal of Information and Computer Security* 11 (Jan. 2019), p. 1. DOI: 10.1504/IJICS.2019.10016150.

[72] Per Håkon Meland, Yara Fareed Fahmy Bayoumy, and Guttorm Sindre. "The Ransomware-as-a-Service economy within the darknet". In: *Computers & Security* 92 (2020), p. 101762. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2020.101762. URL: https://www.sciencedirect.com/science/article/pii/S0167404820300468.

[73] James Messer. *Secrets of Network Cartography: A comprehensive guide to nmap*. Professor Messer, 2007.

[74] Microsoft. *Microsoft Security Bulletin MS17-010 - Critical*. 2023. URL: https://learn.microsoft.com/en-us/security-updates/SecurityBulletins/2017/ms17-010.

[75] Microsoft. "Microsoft Security Bulletin MS17-010—Critical". In: (2017).

[76] Microsoft. *Windows 7 system requirements*. 2020. URL: https://support.microsoft.com/en-us/windows/windows-7-system-requirements-df0900f2-3513-a851-13e7-0d50bc24e15f.

[77] *mimikatz*. 2018. URL: https://github.com/ParrotSec/mimikatz/blob/master/README.md.

[78] Najmeh Miramirkhani et al. "Spotless Sandboxes: Evading Malware Analysis Systems Using Wear-and-Tear Artifacts". In: *2017 IEEE Symposium on Security and Privacy (SP)*. May 2017, pp. 1009–1024. DOI: 10.1109/SP.2017.42.

[79] Samrah Mirza et al. "A Malware Evasion Technique for Auditing Android Anti-Malware Solutions". In: *2021 IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. Oct. 2021, pp. 125–130. DOI: 10.1109/WETICE53228.2021.00034.

[80] *MS17-010 EternalRomance/EternalSynergy/EternalChampion SMB Remote Windows Code Execution*. 2018. URL: https://www.rapid7.com/db/modules/exploit/windows/smb/ms17_010_psexec/.

[81] Mifraz Murthaja et al. "An automated tool for memory forensics". In: *2019 International Conference on Advancements in Computing (ICAC)*. IEEE. 2019, pp. 1–6.

[82] Anitta Patience Namanya et al. "The World of Malware: An Overview". In: *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*. Aug. 2018, pp. 420–427. DOI: 10.1109/FiCloud.2018.00067.

[83] Anitta Patience Namanya et al. "The world of malware: An overview". In: *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE. 2018, pp. 420–427.

[84] NationalSecurityAgency. *Ghidra Software Reverse Engineering Framework*. 2023. URL: https://github.com/NationalSecurityAgency/ghidra.

[85] Sabila Newaz, Hasan Md Imran, and Xingya Liu. "Detection of Malware Using Deep Learning". In: *2021 IEEE 4th International Conference on Computing, Power and Communication Technologies (GUCON)*. Sept. 2021, pp. 1–4. DOI: 10.1109/GUCON50781.2021.9573991.

[86] Security Onion. *Security Onion*. 2023. URL: https://docs.securityonion.net/en/2.3/about.html.

[87] Kevin O'Reilly. *CAPEv2*. 2023. URL: https://github.com/kevoreilly/CAPEv2/.

[88] Kevin O'Reilly. *To install CAPE*. 2020. URL: https://capev2.readthedocs.io/en/latest/installation/host/installation.html#to-install-cape.

[89] radare org. *Radare2: Libre Reversing Framework for Unix Geeks*. 2023. URL: https://github.com/radareorg/radare2.

[90] Nagababu Pachhala, S. Jothilakshmi, and Bhanu Prakash Battula. "A Comprehensive Survey on Identification of Malware Types and Malware Classification Using Machine Learning Techniques". In: *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)*. Oct. 2021, pp. 1207–1214. DOI: 10.1109/ICOSEC51865.2021.9591763.

[91] Gonzalo De La Torre Parra et al. "Detecting Internet of Things attacks using distributed deep learning". In: *Journal of Network and Computer Applications* 163 (2020), p. 102662.

[92] FOG Project. *FOG Project*. 2023. URL: https://fogproject.org/.

[93] Vinayakumar R. et al. "Ransomware Triage Using Deep Learning: Twitter as a Case Study". In: *2019 Cybersecurity and Cyberforensics Conference (CCC)*. May 2019, pp. 67–73. DOI: 10.1109/CCC.2019.000-7.

[94] REMnux. *REMnux: A Linux Toolkit for Malware Analysis*. 2023. URL: https://docs.remnux.org/.

[95] Check Point Research. *Evasions: Generic OS queries*. 2023. URL: https://evasions.checkpoint.com/techniques/generic-os-queries.html#check-if-hard-disk.

[96] kevoreilly Revision. *Sandboxing*. 2022. URL: https://capev2.readthedocs.io/en/latest/introduction/sandboxing.html.

[97] Elpida Rouka, Celyn Birkinshaw, and Vassilios G. Vassilakis. "SDN-based Malware Detection and Mitigation: The Case of ExPetr Ransomware". In: *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*. Feb. 2020, pp. 150–155. DOI: 10.1109/ICIoT48696.2020.9089514.

[98] Vasu Sethia and A Jeyasekar. "Malware Capturing and Analysis using Dionaea Honeypot". In: *2019 International Carnahan Conference on Security Technology (ICCST)*. Oct. 2019, pp. 1–4. DOI: 10.1109/CCST.2019.8888409.

[99] S. Sibi Chakkaravarthy, D. Sangeetha, and V. Vaidehi. "A Survey on malware analysis and mitigation techniques". In: *Computer Science Review* 32 (2019), pp. 1–23. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2019.01.002. URL: https://www.sciencedirect.com/science/article/pii/S1574013718301114.

[100] Rami Sihwail, Khairuddin Omar, and KA Zainol Ariffin. "A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis". In: *Int. J. Adv. Sci. Eng. Inf. Technol* 8.4-2 (2018), pp. 1662–1671.

[101] Michael Sikorski and Andrew Honig. *Practical malware analysis: the hands-on guide to dissecting malicious software*. no starch press, 2012.

[102] Sophos. *The State of Ransomware 2022*. 2022. URL: https://assets.sophos.com/X24WTUEQ/at/4zpw59pnkpxxnhfhgj9bxgj9/sophos-state-of-ransomware-2022-wp.pdf.

[103] Aditya Tandon and Anand Nayyar. "A Comprehensive Survey on Ransomware Attack: A Growing Havoc Cyberthreat". In: *Data Management, Analytics and Innovation*. Ed. by Valentina Emilia Balas, Neha Sharma, and Amlan Chakrabarti. Singapore: Springer Singapore, 2019, pp. 403–420. ISBN: 978-981-13-1274-8.

[104] F. Thomas. *Adware: The Only Book You'll Ever Need.* 2015.

[105] TP-LINK. *Installation Guide*. 2023. URL: https://static.tp-link.com/upload/manual/2023/202304/20230411/7106509649_TL-SG108(UN)_IG.pdf.

[106] Emmanouil Vasilomanolakis et al. "HosTaGe: A Mobile Honeypot for Collaborative Defense". In: *Proceedings of the 7th International Conference on Security of Information and Networks*. SIN '14. New York, NY, USA: Association for Computing Machinery, 2014, 330–333. ISBN: 9781450330336. URL: https://doi.org/10.1145/2659651.2659663.

[107] Emmanouil Vasilomanolakis et al. "This Network is Infected: HosTaGe - a Low-Interaction Honeypot for Mobile Devices". In: *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*. SPSM '13. New York, NY, USA: Association for Computing Machinery, 2013, 43–48. ISBN: 9781450324915. DOI: 10.1145/2516760.2516763. URL: https://doi.org/10.1145/2516760.2516763.

[108] P Vinod et al. "Survey on malware detection methods". In: *Proceedings of the 3rd Hackers' Workshop on computer and internet security (IITKHACK'09)*. 2009, pp. 74–79.

[109] *VirusShare.com - Because Sharing is Caring*. 2023. URL: https://virusshare.com/.

[110] Visual-Paradigm. *Adaptive vs Predictive Planning: When Agile? When Waterfall?* 2021. URL: https://tinyurl.com/4wsj2sfp.

[111] Wikipedia. *Time Stamp Counter*. 2021. URL: https://en.wikipedia.org/wiki/Time_Stamp_Counter.

[112] Hui Xia et al. “Modeling and Analysis Botnet Propagation in Social Internet of Things”. In: *IEEE Internet of Things Journal* 7.8 (Aug. 2020), pp. 7470–7481. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.2984662.

[113] Akira Yokoyama et al. “SandPrint: Fingerprinting Malware Sandboxes to Provide Intelligence for Sandbox Evasion”. In: *Research in Attacks, Intrusions, and Defenses*. Ed. by Fabian Monrose et al. Cham: Springer International Publishing, 2016, pp. 165–187. ISBN: 978-3-319-45719-2.

[114] Katsunari Yoshioka et al. “Vulnerability in Public Malware Sandbox Analysis Systems”. In: *2010 10th IEEE/IPSJ International Symposium on Applications and the Internet*. July 2010, pp. 265–268. DOI: 10.1109/SAINT.2010.16.

[115] Ilsun You and Kangbin Yim. “Malware Obfuscation Techniques: A Brief Survey”. In: *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*. Nov. 2010, pp. 297–300. DOI: 10.1109/BWCCA.2010.85.

[116] Xiaolu Zhang et al. “Iot botnet forensics: A comprehensive digital forensic case study on mirai botnet servers”. In: *Forensic Science International: Digital Investigation* 32 (2020), p. 300926.

Appendix A

Host Setup

```
1  #!/bin/bash
2  USER=$(whoami)
3  #update repo
4  sudo apt update -y && apt-get update -y
5  sudo apt install -y net-tools git curl wget
6  # install python and pip
7  sudo apt install python3-pip -y || return
8  python3 -m pip install testresources libmagic
9  # Install these deps, they are needed to build most of the
       ↪ packages needed in cape install script
10 sudo apt-get -y install build-essential cmake ninja-build python3-
       ↪ dev cython3 pybind11-dev libre2-dev libjansson-dev libcurl4-
       ↪ gnutls-dev libgeoip-dev libxml2 libxslt1-dev libvirt-dev
11
12 # Then install python-libvirt. Better do it manually by getting
       ↪ the source
13 wget https://files.pythonhosted.org/packages/77/64/066990
       ↪ abfec62e6976e5e74a6fc6e8403fcc8b4f6d266deb85a31994f34f/
       ↪ libvirt-python-8.8.0.tar.gz
14 tar -zxvf libvirt-python-8.8.0.tar.gz
15 cd libvirt-python-8.8.0
16 sudo make && sudo make install
17 sudo reboot now
```

Listing A.1: Dependencies installation script

```
1  #!/bin/bash
2  USER=$(whoami)
```

```
3  echo $USER
4  sudo apt update && sudo apt upgrade -y
5  sudo apt install qemu-kvm libvirt-daemon-system libvirt-clients
       ↪ bridge-utils -y
6  sudo usermod -a -G kvm ${USER}
7  sudo usermod -a -G libvirt ${USER}
8  sudo systemctl enable --now libvirtd
9  sudo apt install virt-manager -y
10 sudo reboot now
```

Listing A.2: KVM-QEMU hypervisor installation script

Appendix B

Testbed setup

Program	Description
.NET FRAMEWORK	Microsoft .NET FRAMEWORK >= 4.8 is required by the CAPEv2 sandbox.
Python 3	Required by the CAPEv2 sandbox. We recommend to use the version 3.8.10 32bit. Install, add it to path and disable path length if needed.
Git	Needed to fetch repositories from GitHub by Python-Package manager (pip).
CMake	An open-source and cross-platform family of tools for building, testing and packaging software. Needed to compile and build some C libraries wrapped in Python.
MinGW	Previously known as mingw32, is an open-source software development environment to create applications for Microsoft Windows. It includes a port of the GNU Compiler Collection (gcc and g++). Required by some Python packages.
Visual Studio. 2017	Required to build and compile some python packages such as re and pyre2. From the installer, the user needs to pick C++ development tools for installation and select the following for installation elements: VCC 2017 version 15.9 v16.16 latest. C++ profiling tools. Windows sdk 10 10.0.177630 and Visual C++ tools for CMake. A system reboot is required.
Chocolatey	An open source project that provides a package manager for Windows, and can be installed via Powershell [1].

Table B.1: Windows 7 32bit testbed dependencies

After the Windows 7 installation terminates, the following steps are similar to the physical machine setup. We use the Windows 7 Professional Service Pack1, and it needs some updates to be able to support Python 3. Namely *windows6.1-kb4457144x86*. After the update is installed, the user can proceed with installing all the following programs that are needed for the Sandbox as showed in the table above.

Once finished, the user needs to install the two scripts provided by CAPEv2. The *choco.bat*, which installs some Virtual machine core dependency [36]. Followed by the *disable_win7noise1.bat*, which is a script for disabling some Windows network programs that produce network noise [37]. After rebooting the machine, the VM is ready for the sandbox communication setup. The CAPEv2 uses the *agent.py* Python script to communicate with the VM and transfer all the analysis files and binaries. Additionally, it transfers all the analysis data back to the Sandbox resultserver. However, the agent.py script uses a library that is not mentioned in the official documentation, and it is not a native Python library. They only mention Pillow, which is a python library used to take screenshots [22]. From the agent.py, we see the import of the *re2* package as shown below, which will throw the ModuleNotFoundError exception:

```
1    try:
2        import re2 as re
3    except ImportError:
4        import re
```

Listing B.1: Agent.py re2 library import

RE2 is a regular expression library written in C++ and can be wrapped and imported by Python [49]. We have encountered some difficulties compiling it on Windows 7 due to some compiler conflict. Hence, we decided to build it from source using. Nevertheless, we offer the compiled files to be simply downloaded and placed in *C:/Program Files/* to save time[2]. After putting the RE2 in the location, install the python library *pyre2*. Now the VM testbed should be ready for the last step. Adding the agent.py as a Windows task is well documented in the official documentation. However, we recommend using *pyinstaller* or *nuitka* to compile the agent.py into an executable for more stealth.

```
1    # can add other flags --nooutput --nowindow for more stealth
2    pyinstaller --onefile --name [exe-name].exe agent.py
```

Listing B.2: Agent.py to executable command

[2]https://drive.google.com/drive/folders/1JRVA9ZgOHZbFmubOFQVfjSM-dd2oJ4A2?usp=sharing

After placing the agent executable as a task, reboot to verify if it runs on the OS startup by suing a simple HTTP-GET request:

```
# can add other flags --nooutput --nowindow for more stealth
curl http://[testbed-ip]:8000
```

Listing B.3: Agent.py verification

Finally, we take a snapshot with KVM and exit. Regarding the physical testbed, we take an image of the system using the FOG server.

Appendix C

Sandbox configuration & Troubleshooting

Before starting, we need to make sure that the */opt/CAPEv2/* directory is owned by the Sandbox user (cape user). This was the case for us as some of the Python package installation inherits the root scope, and hence will be only accessible by root user. This can be noticed from the installation script warning logs. Nevertheless, we simply recursively change the ownership with:

```
1     sudo chown -R cape:cape /op/CAPEv2/
```

```
1 # Make sure this file exists. If not create a symbolic link of /
      ↪ usr/bin/tcpdump and change the group of tcpdump to pcap
      ↪ group
2 tcpdump = /usr/sbin/tcpdump
3 # change this to your virtual-bridge interface name. In our case
      ↪ virbr0
4 interface = virbr0
```

Listing C.1: auxiliary.conf setup

The next file is *cuckoo.conf*. This is where the result server and the Postgresql database credentials are defined. The result-server is where the sandbox is listening for the analysis results. In our case 192.168.2.2. The Postgresql database is used to store the analysis related data. Such as submission-task ID, testbeds names and information and others. Without it, the cape.service will throw a connection refused exception and stop running. To fix this, modify the connection string in cuckoo.conf:

```
1 [database]
2 # For production we strongly suggest go with PostgreSQL
```

```
3 #connection = postgresql://[username]:[password]@localhost:5432/[
      ↪ db-name]
4 connection = postgresql://cape:cs3.zer0.d4y@localhost:5432/cape
```

Listing C.2: cuckoo.conf setup

Now, we need to change a hardcoded database name in the CAPEv2 source code. The file is at */opt/CAPEv2/lib/cuckoo/core/database.py*

```
1 # Find the Line number 540 and change the value to cape.db
2 db_file = os.path.join(CUCKOO_ROOT, "db", "cuckoo.db")
```

The last step is to check if the public schema in the Postgresql database belongs to the CAPEv2 user:

```
1 # start postgresql shell
2 psql
3 # check the ownership of the shcemas:
4 \dn
5 # change the ownership to the cape user.
6 ALTER SCHEMA public OWNER TO cape;
7 # quit to save
8 \q
```

If we decide to use the virtual testbed, we need to configure the *kvm.conf* and the *qemu.conf*. This is where information about the VMs is kept.

```
1  [kvm]
2  # Obtain the list of your VMs from the GUI or via terminal [virsh
       ↪ list --all]
3  machines = win7x32pro
4  # The interface on which the VM is spinned. In our case virbr0
5  interface = virbr0
6  [win7x32pro]
7  # Specify the label name of the current machine as specified in
       ↪ your
8  # libvirt configuration.
9  label = Win7x86
10 # Specify the OS platform uof the VM-testbed
11 platform = windows
12 # Specify the IP address of the current virtual machine.
13 ip = 192.168.122.5
14 # Set the machine architecture
15 arch = x86
```

Listing C.3: kvm.conf setup

```
[qemu]
# Path to one qemu binary (assumes the other ones are there as
    ↪ well)
path = /usr/bin/qemu-system-x86_64
# Specify a comma-separated list of available machines
machines = win7x32pro
# Specify the network interface for network traffic with tcpdump.
interface = virbr0
[win7x32pro]
label = win7x86
# image path
image = /var/lib/libvirt/images/win7x32pro.qcow2
# saved snapshot name
snapshot = snapshot1
#vm arch (x86/x64/arm/arm64/sh4/sparc/sparc64/powerpc/powerpc64/
    ↪ mips/mipsel)
arch = x86
# [windows/darwin/linux].
platform = Windows
# Specify the IP address of the VM.
ip = 192.168.122.5
```

Listing C.4: qemu.conf setup

The *routing.conf* is responsible for the network routing between the CAPEv2 and the VMs. The following two values need to be changed

```
# If you want to allow the internet, else chose none
route = internet
# Network interface that allows a VM to connect to the entire
    ↪ internet:
internet = eno1
```

Listing C.5: routing.conf setup

If a physical machine is used as a testbed, we need to modify the *physical.conf* file with the following information as shown below:

```
[physical]
# The name of the user on the target physical machine
machines = Mo
```

```
# the physical interface of the host where CAPEv2 is installed.
interface = eno1
[fog]
# Provide the ip of the machine where fog runs. In our case the VM
    ↪ 's IP
hostname = 192.168.2.7
# Find it in the menu: Settings/Fog Settings/ API Setting. Make
    ↪ sure to enable it.
apikey = XXXX
# Find it under the menu: Users/List All users/fog-username/API.
    ↪ Make sure to enable it.
user_apikey = XXXX
[Mo]
# Specify the label name of the current machine as specified in
    ↪ your
# physical machine configuration.
label = Thinkpad-AAU1
# Specify the OS of the physical machine
platform = windows
# Specify the IP address of the physical machine
ip = 192.168.2.5
# CAPEv2 server's IP. THE THINKSTATION1 IP in our case.
resultserver_ip = 192.168.2.2
resultserver_port = 2042
# The physical machine's architecture
arch = x86
```

Listing C.6: physical.conf setup

The last is *reporting.conf*, where the processed data is stored. This conf file is critical to CAPEv2 and without MongoDB, the user cannot display the analysis results, and will see some exception logs on the sandbox Web-GUI.

```
[mongodb]
enabled = yes
host = 127.0.0.1
port = 27017
db = cape
# Set those values if you are using mongodb authentication
username = cape
password = cs3.zer0.d4y
authsource = cape
```

Listing C.7: processing.conf setup

One of the errors we face is the MongoDB. We need to create the sandbox user and change some values inside the MongoDB server to allow create/delete/drop and other privileged operations. To achieve this, we use the following Mongo-Shell commands:

```
# connect to the admin db. By default, there is not auth needed!
mongosh --port 27017
# now you are in. Use admin DB
test> use admin
switched to db admin
# Check if user cape or cuckoo exists?
admin> show users
[]
# check for DB cape or cuckoo
admin> show databases
admin 40.00 KiB
config 12.00 KiB
local 40.00 KiB

# No such user. Create one with admin to any db
#Make sure you are connected to db admin (use admin)
admin> db.createUser({user: "cape", pwd: "cs3.zer0.d4y", roles:[{
    ↪ role: "userAdminAnyDatabase", db: "admin"}]})

# Now create a db with name cape
> use cape
# create a collection otherwise the database creattion does not
    ↪ take effect!
cape> db.createCollection("analysis")

# Just to be safe, create a user for DB cape
cape> db.createUser({user: "cape", pwd: "cs3.zer0.d4y", roles:[{
    ↪ role: "userAdmin", db: "cape"}]})
# grant some rules (just to be on the safe side!). UserAdmin
    ↪ should inherit read and write by default!
cape> db.grantRolesToUser("cape",["userAdmin",{ role: "readWrite",
    ↪  db: "cape"}])
cape> db.grantRolesToUser("cape",["readWrite",{ role: "userAdmin",
    ↪  db: "cape"}])
```

```
# That is all. quit to save
quit
```

Listing C.8: Fix MongoDB issues

There are hardcoded data again for mongoDB in their source code. The Find the following Python script at */opt/CAPEv2/dev_utils/mongodb.py* and change it:

Listing C.9: Fix MongoDB hardcoded data

```
# Line 11. The MongoDb database name > change to cape
mdb = repconf.mongodb.get("db", "cuckoo")
 def connect_to_mongo()
    host=repconf.mongodb.get("host", "127.0.0.1"),
    port=repconf.mongodb.get("port", 27017),
    username=repconf.mongodb.get("username"),
    password=repconf.mongodb.get("password"),
    # change this auth-source to match the cape user. > ("
        ↪ authsource", "cape")
    authSource=repconf.mongodb.get("authsource", "cuckoo"),
    connect=True,
```

Concerning poetry, which is the tool used for dependency management and packaging. We detected points to different directory than the one where CAPEv2 gets installed, namely */home/cape/* instead of */opt/CAPEv2/*. To solve this issue, we change the *cache-dir* and the *virtualens.path* to point at the desired directory:

```
# activate the enviroment
source .cache/pypoetry/virtualenvs/capev2-t2x27zRb-py3.10/bin/
    ↪ activate
# change value cache-dir and virtualenvs.path
poetry config cache-dir "/opt/CAPEv2/.cache/pypoetry"
poetry config virtualenvs.path "/opt/CAPEv2/.cache/pypoetry/
    ↪ virtualenvs"
# Now start installing the requirements from the requirements.txt
poetry run python3 -m pip install --upgrade pip
poetry run python3 -m pip install -r requirements.txt
# if you get errors such as "no such module for [missing module]",
    ↪  run
poetry run python3 -m pip install [missing-module]
```

Listing C.10: Poetry conflict solution

Detect It Easy is a program used by CAPEv2 to detect the sample program's extension (.exe, .dll, .bin and others). The installation fails installing it as some depen-

dencies are not included in the script and are not installed by default on the OS. To solve this, we provide a manual installation script.

Listing C.11: DIEC install script

```
#!/bin/bash
# install some dependencies
echo "---[Installing dependencies]---"
sudo apt-get update -y && sudo apt-get install -y libqt5opengl5
    ↪ libqt5scripttools5 libqt5script5 libqt5sql5
cd /tmp && wget https://github.com/horsicq/DIE-engine/releases/
    ↪ download/3.07/die_3.07_Ubuntu_22.04_amd64.deb
# install diec
sudo dpkg -i die_3.07_Ubuntu_22.04_amd64.deb
```

A reboot is recommended and then the sandbox is ready for testing.

Appendix D

SPM integration

```
1  Completed SYN Stealth Scan at 13:25, 16.86s elapsed (4000 total
       ↪ ports)
2  # GATEWAY
3  Nmap scan report for Station0 (192.168.122.1)
4  Host is up (0.000097s latency).
5  Not shown: 994 closed ports
6  PORT STATE SERVICE
7  22/tcp open ssh
8  53/tcp open domain
9  111/tcp open rpcbind
10 8000/tcp open http-alt
11 MAC Address: F8:D1:11:15:C5:A3 (Tp-link Technologies)
12
13 # win7x86pro (Patient-0)
14 Nmap scan report for 192.168.122.5
15 Host is up (0.00021s latency).
16 Not shown: 990 closed ports
17 PORT STATE SERVICE
18 135/tcp open msrpc
19 139/tcp open netbios-ssn
20 445/tcp open microsoft-ds
21 8000/tcp open http-alt
22 49152/tcp open unknown
23 49153/tcp open unknown
24 49154/tcp open unknown
25 49155/tcp open unknown
26 49156/tcp open unknown
27 49157/tcp open unknown
```

```
28 MAC Address: DC:53:60:18:97:37 (Intel Corporate)
29
30 # SPM1 win7x86home
31 Nmap scan report for Jason-PC (192.168.122.32)
32 Host is up (0.00026s latency).
33 Not shown: 993 filtered ports
34 PORT STATE SERVICE
35 21/tcp open ftp
36 22/tcp open ssh
37 135/tcp open msrpc
38 139/tcp open netbios-ssn
39 445/tcp open microsoft-ds
40 2869/tcp open icslap
41 5357/tcp open wsdapi
42 MAC Address: DC:53:60:79:1E:A6 (Intel Corporate)
43 # SPM2 win10x86
44 Nmap scan report for DESKTOP-674A24R (192.168.122.95)
45 Host is up (0.00026s latency).
46 Not shown: 997 filtered ports
47 PORT STATE SERVICE
48 21/tcp open ftp
49 22/tcp open ssh
50 135/tcp open msrpc
51 139/tcp open netbios-ssn
52 445/tcp open microsoft-ds
53 MAC Address: DC:53:60:92:1E:A6 (Intel Corporate)
54
55 # Ubuntu server
56 Initiating SYN Stealth Scan at 13:25
57 Scanning sspnserver (192.168.122.101) [1000 ports]
58 Discovered open port 22/tcp on 192.168.122.101
59 Completed SYN Stealth Scan at 13:25, 0.04s elapsed (1000 total
       ↪ ports)
60 Nmap scan report for sspnserver (192.168.122.101)
61 Host is up (0.0000040s latency).
62 Not shown: 999 closed ports
63 PORT STATE SERVICE
64 22/tcp open ssh
65
66 Read data files from: /usr/bin/../share/nmap
67 Nmap done: 256 IP addresses (5 hosts up) scanned in 18.69 seconds
```

```
68          Raw packets sent: 7594 (326.040KB) | Rcvd: 4037 (165.624
               ↪ KB)
```

Listing D.1: Virtual-Network full Nmap scan results

```
1  Starting Nmap 7.80 ( https://nmap.org ) at 2023-05-31 22:34 UTC
2  Nmap scan report for _gateway (192.168.2.1)
3  Host is up (0.0021s latency).
4  Not shown: 997 closed ports
5  PORT STATE SERVICE
6  80/tcp open http
7  1900/tcp open upnp
8  49152/tcp open unknown
9  MAC Address: F8:D1:11:72:A1:8A (Tp-link Technologies)
10 #Physical-Testbed.
11 Nmap scan report for 192.168.2.4
12 Host is up (0.00077s latency).
13 Not shown: 988 closed ports
14 PORT STATE SERVICE
15 135/tcp open msrpc
16 139/tcp open netbios-ssn
17 445/tcp open microsoft-ds
18 7070/tcp open realserver
19 8000/tcp open http-alt
20 49152/tcp open unknown
21 49153/tcp open unknown
22 49154/tcp open unknown
23 49156/tcp open unknown
24 49157/tcp open unknown
25 49158/tcp open unknown
26 49159/tcp open unknown
27 MAC Address: F0:DE:F1:E7:A8:B1 (Wistron Infocomm (Zhongshan))
28 #FOG server
29 Nmap scan report for 192.168.2.7
30 Host is up (0.00095s latency).
31 Not shown: 992 closed ports
32 PORT STATE SERVICE
33 21/tcp open ftp
34 22/tcp open ssh
35 80/tcp open http
36 111/tcp open rpcbind
37 443/tcp open https
```

```
2049/tcp open nfs
3306/tcp open mysql
MAC Address: 54:EE:75:DC:AD:04 (Wistron InfoComm(Kunshan)Co.)
#Sec-Onion Physical
Nmap scan report for 192.168.2.8
Host is up (0.00060s latency).
Not shown: 999 filtered ports
PORT STATE SERVICE
22/tcp open ssh
MAC Address: 04:7C:16:21:CA:BA (Unknown)
#Clean-target 1
Nmap scan report for 192.168.2.10
Host is up (0.0080s latency).
Not shown: 996 closed ports
PORT STATE SERVICE
135/tcp open msrpc
139/tcp open netbios-ssn
445/tcp open microsoft-ds
5357/tcp open wsdapi
MAC Address: 8C:70:5A:21:83:60 (Intel Corporate)
#Sec-onion
Nmap scan report for 192.168.2.50
Host is up (0.00090s latency).
Not shown: 997 filtered ports
PORT STATE SERVICE
22/tcp open ssh
80/tcp open http
443/tcp open https
MAC Address: 8C:17:5A:44:A5:5E (Unknown)
#Sandbox host
Nmap scan report for Station0 (192.168.2.2)
Host is up (0.0000030s latency).
Not shown: 995 closed ports
PORT STATE SERVICE
22/tcp open ssh
111/tcp open rpcbind
2042/tcp open isis
8000/tcp open http-alt
# Samsung Mobile (HosTaGe Honeypot)
Nmap scan report for 192.168.2.110
Host is up (0.040s latency).
```

```
79 Not shown: 990 closed ports
80 PORT STATE SERVICE
81 7/tcp open echo
82 21/tcp open ftp
83 22/tcp open ssh
84 23/tcp open telnet
85 25/tcp open smtp
86 80/tcp open http
87 443/tcp open https
88 1025/tcp open NFS-or-IIS
89 3306/tcp open mysql
90 5060/tcp open sip
91 MAC Address: AC:5F:3E:75:F5:98 (Samsung Electro-mechanics(thailand
       ↪ ))
92
93 Nmap done: 256 IP addresses (7 hosts up) scanned in 20.55 seconds
```

Listing D.2: Physical-Network full Nmap scan results

Appendix E

FOG documentation

E.1 Installing and Configuring the FOG Server

The first step is to install git and clone the FOG repository. After this, we have to enter the folder and launch the installer as root user. The installer asks the user a few questions such as which distribution of Linux it is going to be installed on, or which interface it will run on. Some choices are based on each specific setup. In our case, here is the summary of the answers:

```
* Base Linux: Debian
* Detected Linux Distribution: Ubuntu
* Interface: eno1
* Server IP Address: 192.168.2.7
* Server Subnet Mask: 255.255.255.0
* Hostname: damo-StandardUbuntu
* Installation Type: Normal Server
* Internationalization: No
* Image Storage Location: /images
* Using FOG DHCP: Yes
* DHCP router Address: 192.168.2.1
* Send OS Name, OS Version, and FOG Version: Yes
```

Figure E.1: Summary FOG Server configuration

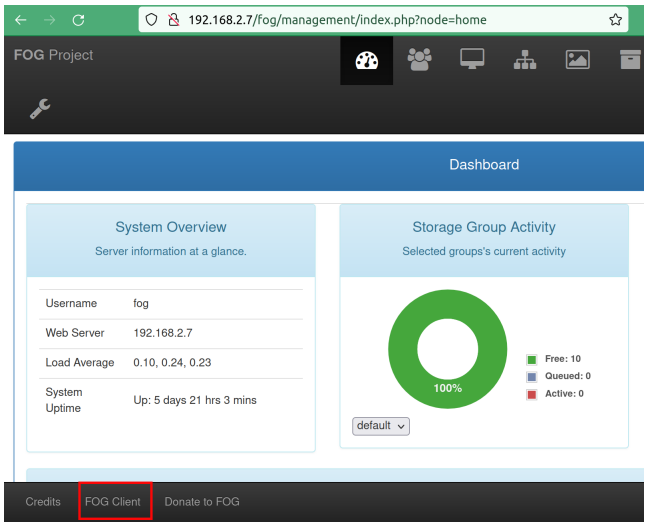
The complete list of our answers for the setup can be found in table E.1. After the installation, we recommend to reboot the computer. That done, the FOG Server can be accessed using the provided IP address, in our case 192.168.2.7/fog. Now that our backup server is set, we can move on and configure the clients we want to backup.

Questions	Answers
What version of Linux would you like to run the installation for? 1) Redhat Based Linux 2) Debian Based Linux 3) Arch Linux	2
Starting Debian based Installation What type of installation would you like to do? [N/s (Normal/Storage)]	N
We found the following interfaces on your system: * enp0s31f6 - 192.168.2.7/24 Would you like to change the default network interface from enp0s31f6? If you are not sure, select No. [y/N]	N
Would you like DHCP to handle DNS? [Y/n] What DNS address should DHCP allow?	Y, 127.0.0.53
Would you like to use the FOG server for DHCP service? [y/N]	y
This version of FOG has internationalization support, would you like to install the additional language packs? [y/N]	N
Would you like to enable secure HTTPS on your FOG server? [y/N]	N
Which hostname would you like to use? Currently is: damo-StandardUbuntu Would you like to change it? If you are not sure, select No. [y/N]	N
FOG would like to collect some data: We would like to collect the following information: 1. OS Name (CentOS, RedHat, Debian, etc....) 2. OS Version (8.0.2004, 7.2.1409, 9, etc....) 3. FOG Version (1.5.9, 1.6, etc....) What is this information used for? We would like to simply track the common types of OS being used, along with the OS Version, and the various versions of FOG being used. Are you ok with sending this information? [Y/n]	Y

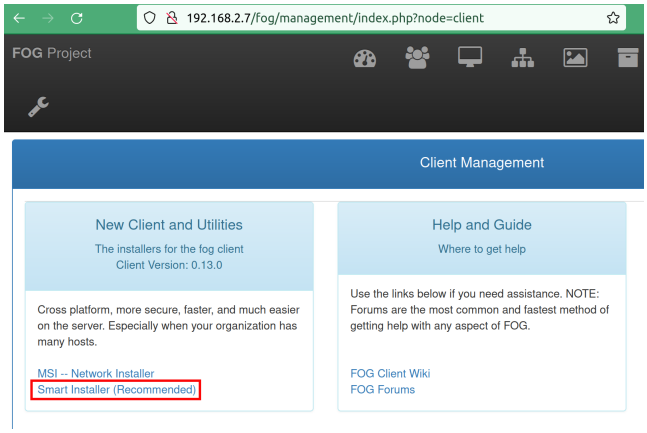
Table E.1: FOG Server Installer - Questions and answers

E.2 Installing and Configuring the FOG Client

From the client machine, we have to download the SmartInstaller.exe, which is present on the FOG server. To do so, we access the following URL in our browser: 192.168.2.7/fog/management/index.php?node=client
or manually go to the "Client Management" page of the FOG server by clicking "Client" on the bottom of the main management page, as described in figure E.2.



(a) Accessing client installers from FOG homepage



(b) Downloading the SmartInstaller

Figure E.2: Getting the SmartInstaller from FOG Server

When the installer is downloaded, we have to launch it. Exe files are executable files that can be run on Windows operating systems. Hence, if we want to run the installer on a Linux system, we first have to download a program that is able to

launch it. FOG Project recommends using mono. That can be achieved by typing the following commands.

```
1  sudo apt install gnupg ca-certificates
2  sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --
       ↪ recv-keys 3FA7E0328081BFF6A14DA29AA6A19B38D3D831EF
3  echo "deb https://download.mono-project.com/repo/ubuntu stable-
       ↪ focal main" | sudo tee /etc/apt/sources.list.d/mono-
       ↪ official-stable.list
4  sudo apt update
5  sudo apt-get install xprintidle -y
6  sudo apt install mono-complete -y
```

Listing E.1: Installing mono on Linux clients

The installer has to be launched as root user. The only thing to specify is the address of the FOG server.

When that is done, we have to register the machine in the FOG server. To do so, we have to set the machine we want to register to boot using the network. The machine should automatically find the FOG server, if not, we just have to specify the IP address of the FOG server. After a few seconds, we get presented with a FOG menu, where we choose the "Perform Full Host Registration and Inventory" option, as shown in the following image.

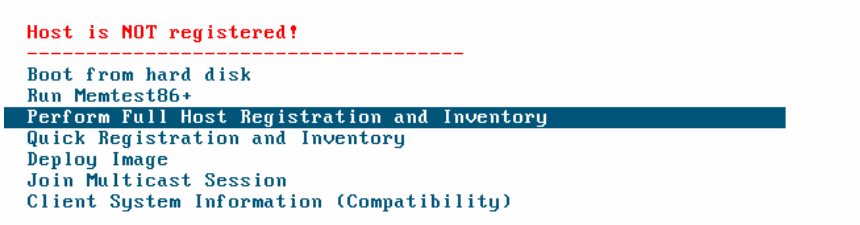


Figure E.3: FOG menu after network boot

The only thing we have to specify in this step is the hostname of the machine and the username of the user on it.

The next step is to register the image we want to create in the FOG Web User interface. We click on the images tab on the top of the FOG page and then "Create a new image". In this "New image" submenu, we can specify different things for our image such as the name, the OS, the storage location and the compression rate (shown in figure E.4). We finally click on "Add" to create the image.

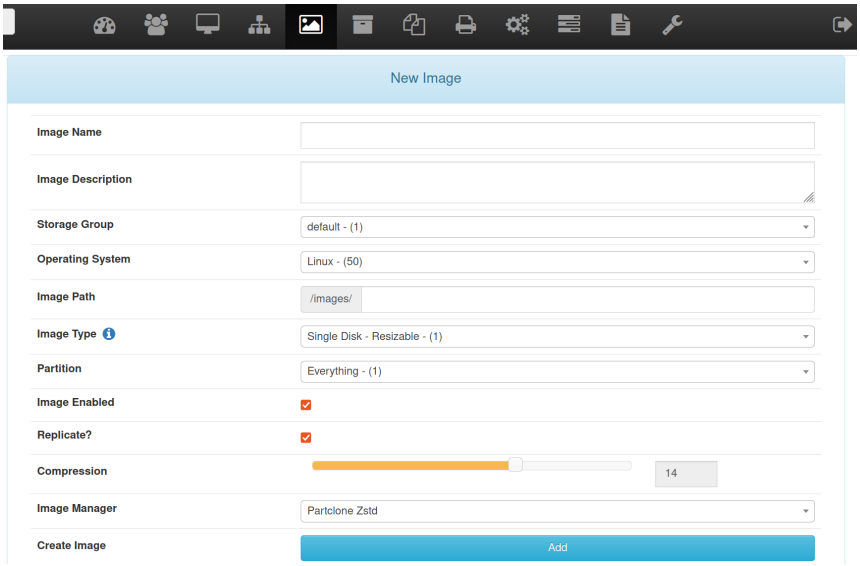


Figure E.4: Creating an image on the FOG Server

The next step is to associate our host with the image we just created. To do so, we go to the "Host Management" page and click on "List All Hosts" and finally the name of the host we registered earlier. On the host image scroll menu, we simply select the image we created in the previous step and click "Update" to apply changes.

The final step is to launch a capture task to backup our host. From the Host Management page of our host, we go to "Basic tasks" and then "Capture", where we can choose when to schedule the task (shown in figure E.5).

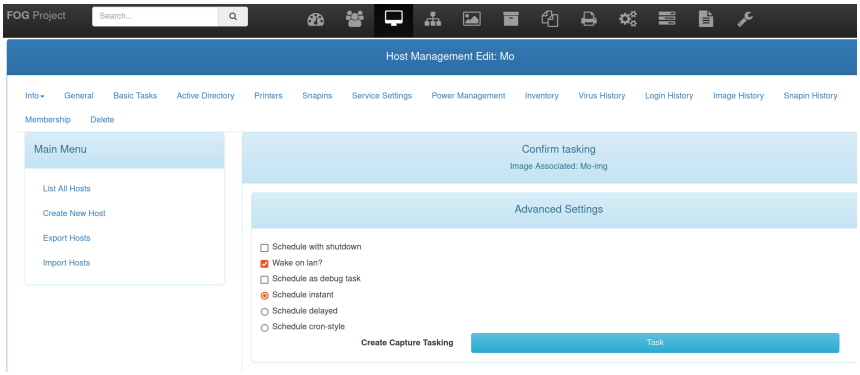


Figure E.5: Creating a capture task on the FOG Server

Appendix F

Security Onion documentation

F.1 Security Onion - figures

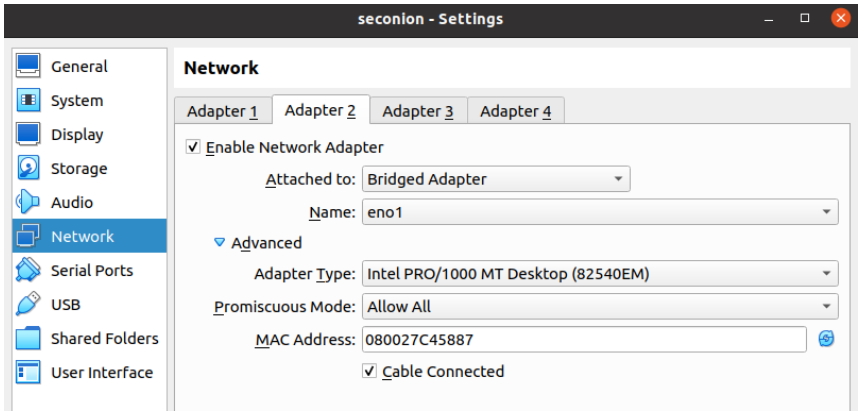


Figure F.1: NIC setup on Security Onion VM for monitoring

F.2 Security Onion installation and configuration

The first step of the installation is to choose which type of installation we want, as listed in section 5.3.4. In our case, we chose the Standalone type which is running on only one machine and is production ready. The standalone type is the most suited for us as we have a limited number of devices to use.

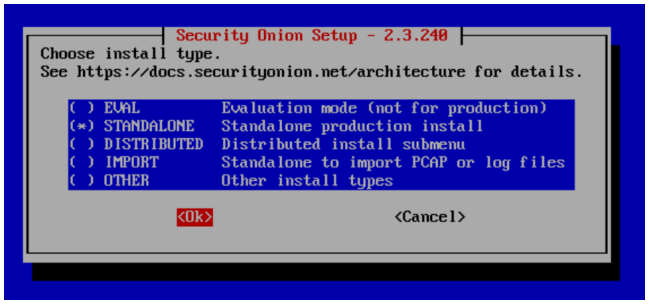


Figure F.2: Installation type

The next step is to choose the hostname we want to set for the machine.

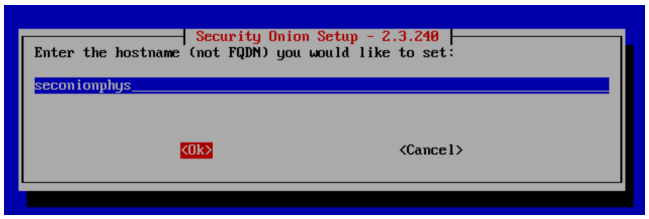
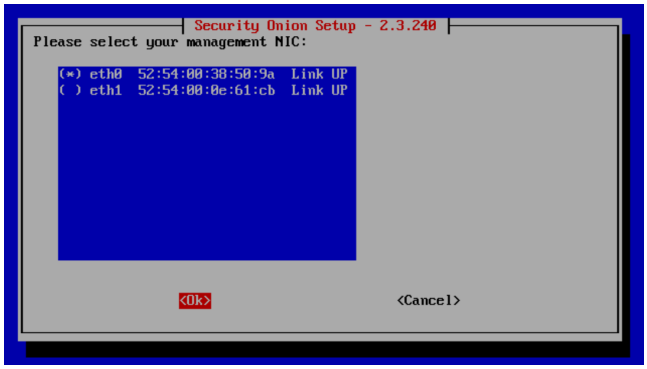
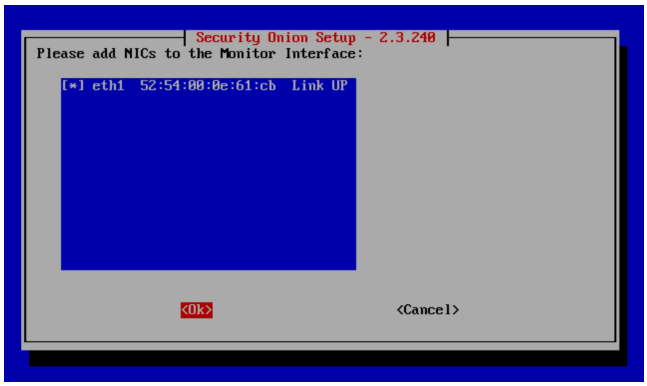


Figure F.3: Hostname

Next, we have to set which interface will be used for management and monitoring. In this step, we want to be careful in selecting which interface is which as only one of those is set to promiscuous mode and thus will be able to monitor traffic.



(a) Management NIC



(b) Monitor NIC

Figure F.4: NICs settings

Next, we can chose if we want the IP of the machine to be static or DHCP. For simplicity, we recommend to use static.

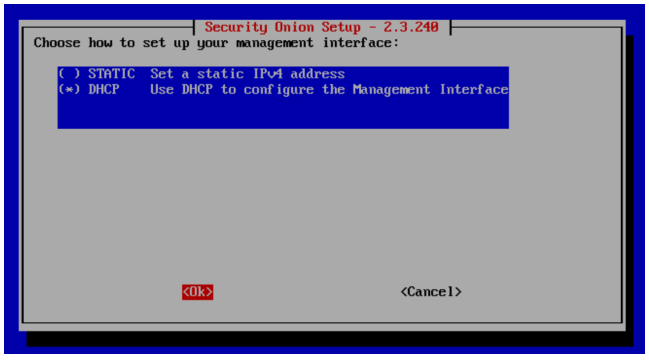


Figure F.5: IP resolution

The next step is to chose if we want to personalize the configuration of the machine. In our case, we go with the basic settings.

The following step is to create an email address to access the Security Onion Console, along with a password, which is up to the user.

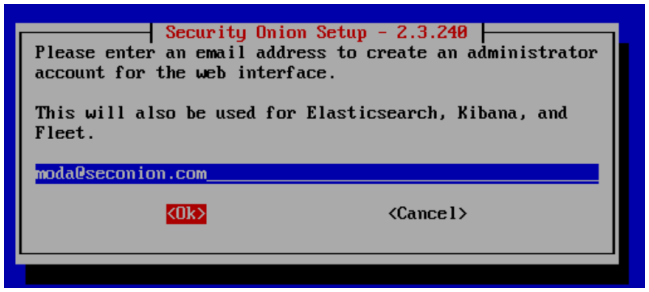


Figure F.6: Email

Following that, the user can choose how he wants network metadata and IDS alerts to be generated. That can be using Suricata only or in combination with Zeek.

Next, we chose which ruleset we want to set for the IDS. Here we have the choice between three rulesets: Emerging Threats Open, Emerging Threats Pro and Snort subscriber. We are forced to chose Emerging Threats Open as both other rulesets need a paid license to be activated.

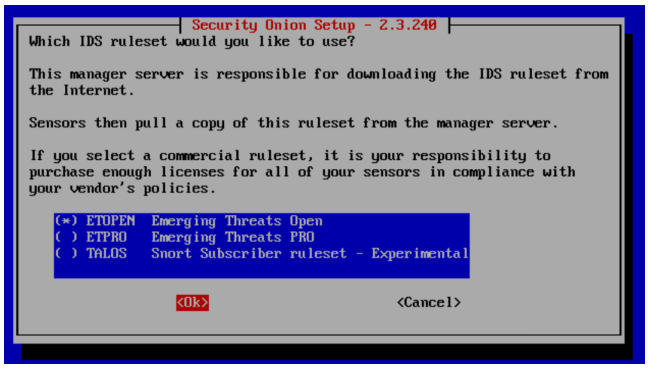


Figure F.7: Ruleset

Next we can chose which optional services we want to activate. We select all of them as we want Security Onion to collect as much information as possible.

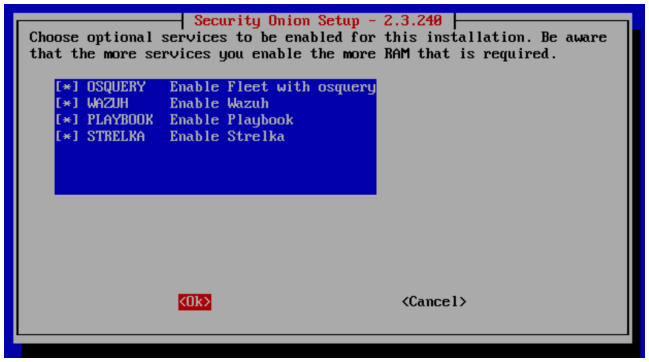


Figure F.8: Optional services

The following step is to choose how to access the web interface. The choice is between using the IP address of the machine or the hostname. As the hostname can be confused we another machine in the network if not configured correctly, we recommend using the IP address.

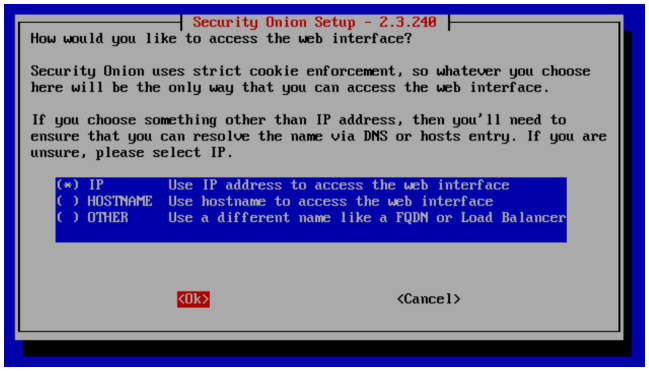


Figure F.9: Web interface

In the next steps, we can select how much Zeek and Suricata processes we want to be running on the machine. We decided to go with the default, which is two in our case.

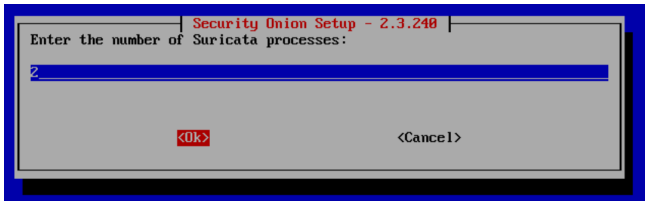


Figure F.10: Suricata processes

When asked about the Docker IP range, the NTP servers and the configuration, we keep the default settings.

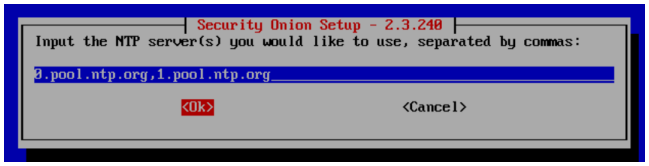


Figure F.11: NTP

Finally, we have to set an IP address or IP range we want to allow access to the Security Onion Console. In our case, it is 192.168.2.0/24

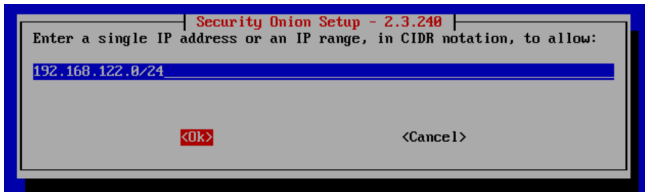
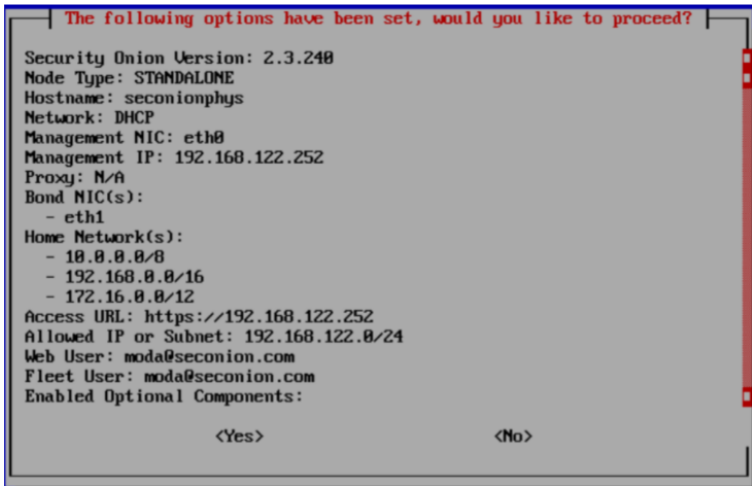


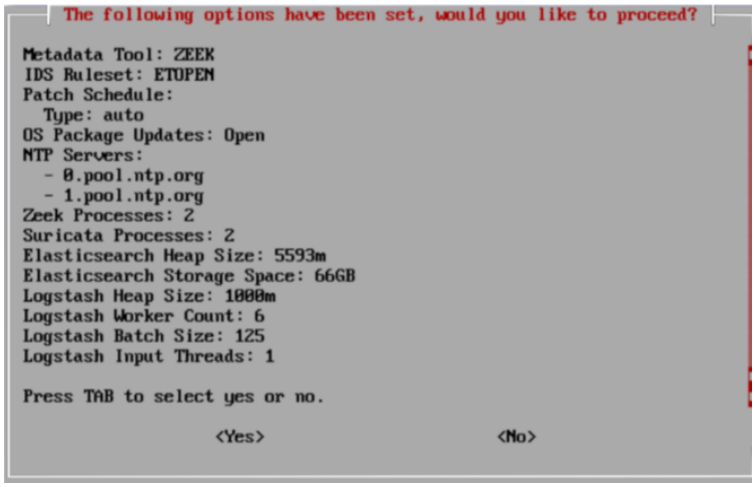
Figure F.12: So-allow

Once those steps and the following installation are completed, Security Onion is fully setup and ready to use.

The figures that show the summary of the configuration can be found in the Appendix F.13.



(a) Configuration Summary 1/2



(b) Configuration Summary 2/2

Figure F.13: Summary of Security Onion configuration

Appendix G

Test and validation

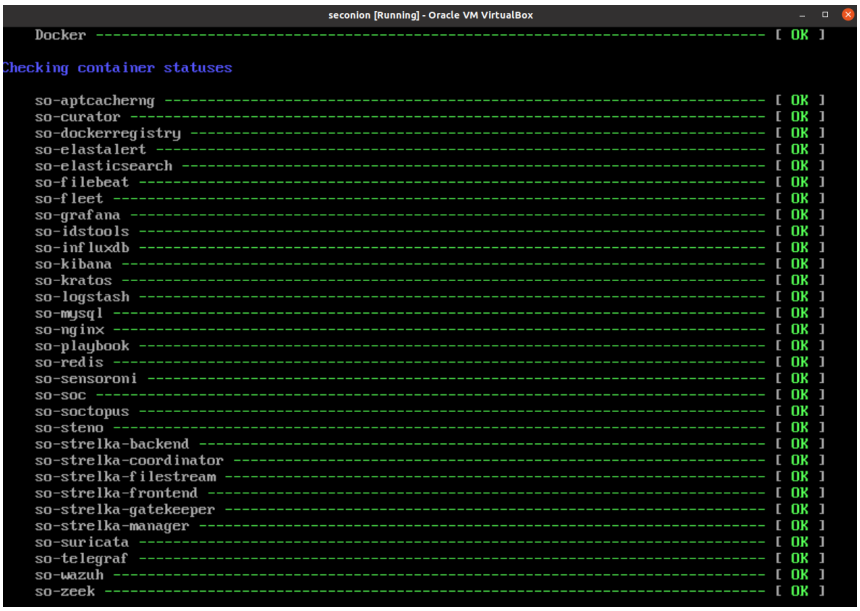
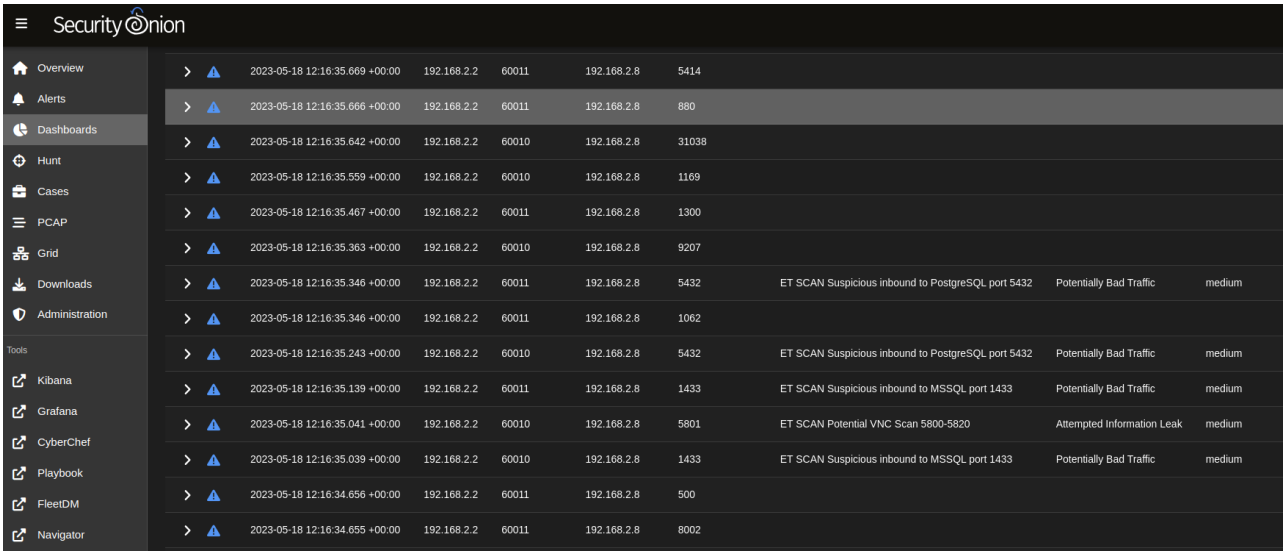
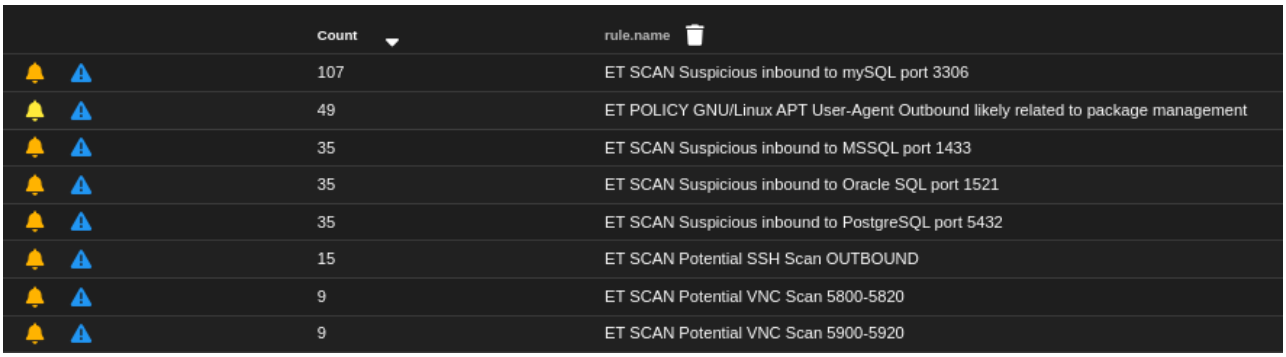


Figure G.1: Output of so-status command

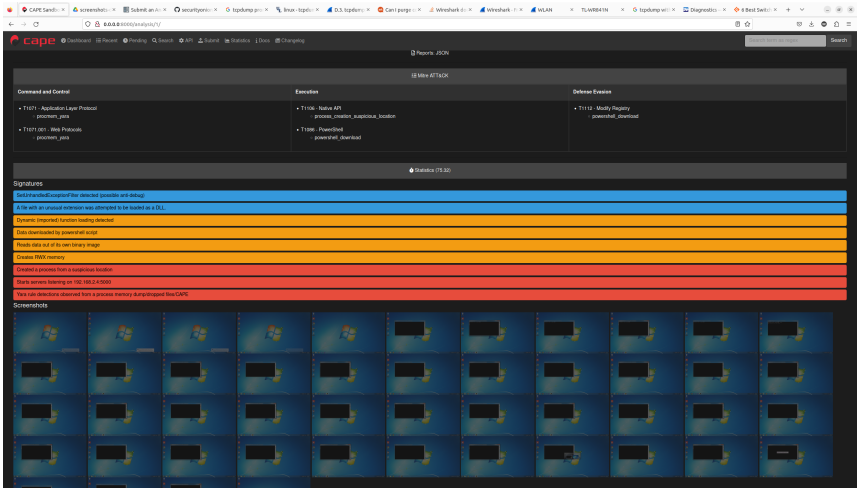


(a) PCAP

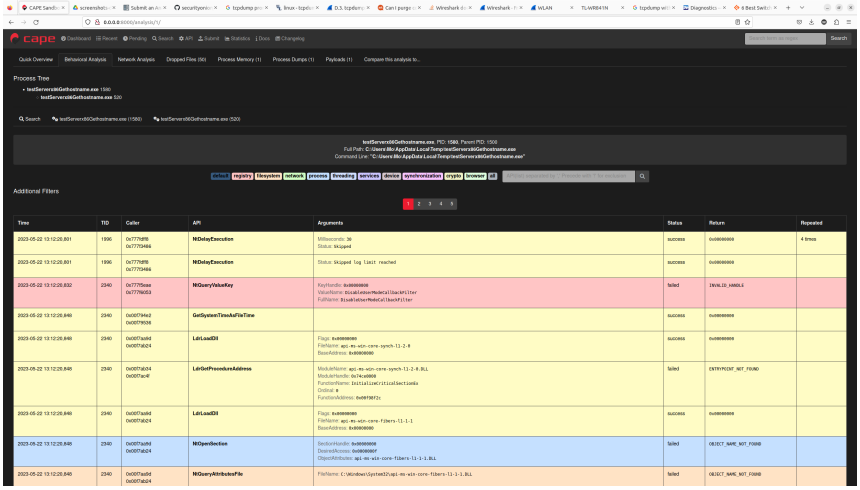


(b) Alerts

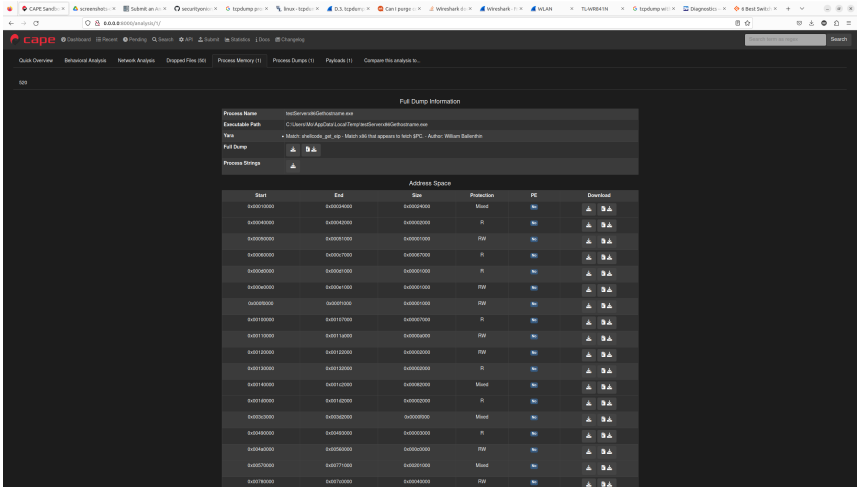
Figure G.2: Nmap scan test - Security Onion



(a) MITRE ATT&CK and screenshots



(b) Behavioral analysis



(c) Memory-Dump analysis

Figure G.3: Sandbox Plugins tests