University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Implementation of a Reproducible and Realistic Data Collection System for Dynamic Malware Analysis

Vichhay Ok
Wil SG, Switzerland
Student ID: 17-709-296

Supervisor: Jan von der Assen, Dr. Alberto Huertas
Date of Submission: August 13, 2023

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Zusammenfassung

Diese Arbeit geht dem Bedarf an besserer Software für die dynamische Malware-Analyse nach, indem die bereits existierende SecBox-Plattform, eine ressourcenschonende, containerbasierte Malware-Analyse-Sandbox, verbessert wird. Die Verbesserungen sollen eine akkurate, einheitliche und reproduzierbare Analyse von diversen Malware-Typen ermöglichen. Die Arbeit taucht in die Prinzipien der dynamischen Analyse und in die Grundlagen der Reproduzierbarkeit ein, um ein klares Verständnis des Problems zu schaffen. Die verbesserte SecBox-Plattform beinhaltet einen Befehlsrekorder, um akkurat Befehle zu reproduzieren, sowie einen CSV Generator, um Systemmetriken wie CPU und RAM Nutzung zu verfolgen. Durch die Evaluation von vier Malware-Typen, einschließlich eines selbstgeschriebenen Skripts, zeigte die überarbeitete SecBox-Plattform über verschiedene Sandbox-Instanzen hinweg eine hohe Einheitlichkeit. Dies unterstreicht ihre Nützlichkeit für die reproduzierbare, dynamische Malware-Analyse.

Abstract

This thesis addresses the need for improved tools in dynamic malware analysis by enhancing the existing SecBox platform; a lightweight, container-based malware analysis sandbox. The enhancements aim at ensuring accurate, consistent, and reproducible analysis of diverse malware types. The thesis delves into the principles of dynamic malware analysis and what constitutes reproducibility, enabling an in-depth understanding of the problem space. The enhanced SecBox platform includes a command recorder to meticulously record and replicate commands and a CSV generator to monitor system metrics like CPU and RAM usage. Through evaluations with four types of malware, one of which was a custom script, the revamped SecBox platform demonstrated high consistency across sandbox instances, underscoring its usefulness in reproducible dynamic malware analysis.

Acknowledgments

I would like to express my deepest gratitude to my supervisors, who provided valuable insights, mentorship, and support throughout this journey. Special thanks go to Jan von der Assen, whose exceptional guidance and commitment were instrumental in shaping this thesis.

On a more personal note, I wish to express my heartfelt thanks to my family, whose unwavering support and encouragement have been the bedrock of my academic career. In particular, I remember my late mother, whose dream it was for me to attend university. This achievement is as much hers as it is mine, and I hope it serves as a testament to her enduring influence on my life.

Lastly, my profound gratitude goes to my girlfriend, Eida Keakavoocy Behbahani. Her steadfast support and love were my anchor in the tumultuous sea of academia. She assumed the heaviest burden during my busiest times, caring for our shared world while I devoted myself to this work. Her commitment and understanding have been beyond measure, and for that, I am forever grateful.

Contents

Chapter 1

Introduction

In today's digital age, malware poses a significant global threat, with cyberattacks becoming increasingly severe and frequent. This escalating issue is receiving more attention across numerous studies on malware, including recent surveys such as that by [1]. Given the profound implications for cybersecurity across various sectors, from businesses to government institutions, the growing diversity of malware [2] underscores a definitive need for appropriate tools.

1.1 Motivation

The state-of-the-art survey on dynamic malware analysis by [3] sheds light on some of the significant challenges faced in the field. They highlight that many research papers suffer from inadequate evaluation: benign files are often omitted from datasets, and test sets do not include a realistic ratio of malicious to benign files, making them misaligned with reality. There is also a concerning lack of comparison to prior tools and transparency, with source code frequently not made available.

These issues indicate that current research is often neither realistic nor reproducible, underscoring a critical gap in the field. Without appropriate tooling and methodologies, the research community and industry struggle to effectively combat the growing threat of malware. This work, motivated by an urgent need for improvement, aims to refine the SecBox platform [4] - a collaborative, lightweight malware analysis sandbox using container virtualization - ensuring it provides accurate, consistent, and reproducible dynamic malware analysis.

1.2 Description of Work

In order to enhance the SecBox platform to fill the current needs, the primary objective of this work is to design and implement a set of enhancements that ensure the analysis within the platform is realistic and reproducible.

1.3 Thesis Outline

First, an understanding of the current malware landscape is gained within chapter 2, by taking a look at its evolution and the different types that encompass the prevalent behaviors. Within the same chapter, a closer look is taken at dynamic malware analysis and the concept of sandboxing. Finally, the history of reproducibility is explored and its definition explained. A comparison of existing tools and what properties constitute reproducibility is highlighted within chapter 3. Following that, chapter 4 explains the architecture of the proposed solution and the steps necessary to reproduce an experiment, given the newly added functionalities. Chapter 5 details the actual implementation of the enhancements and discusses the necessary measures adopted during this process. Chapter 6 assesses the effectiveness of the implemented enhancements using various evaluation techniques or metrics, and chapter 7 concludes the thesis, summarizing the findings, highlighting limitations, and suggesting future work.

Chapter 2

Background

This chapter presents an essential background to gain a common understanding of the context and significance of this research. It begins by introducing a comprehensive overview of malware and its different types, its evolution and the impact it has. Subsequently, malware analysis is defined, with a specific focus on dynamic malware analysis. Finally, this chapter delves into the concept of reproducibility and its necessity within the field.

2.1 Malware

As indicated in the survey of [5], malware is short for malicious software. Within the survey, it is highlighted that in addition to the formal name, there exist numerous definitions of its functionality, however most common is that it is a type of computer code, specifically designed to be hostile and used to cause harm or subvert the intended function of a system. Different types of malware exhibit a specific type of behavior and are thus usually classified into certain categories. Classification is made difficult by the multifaceted and adaptable characteristics of malware. However, the following terms, as defined in the paper of [6], have proven to be enduring and encompass the prevalent behaviors of malware: **Virus** is a self-replicating malicious program, reminiscent to the biological one. Developed as an executable, it spreads by copying itself to other host systems. It needs to be transferred through files, including media and network types. Depending on the complexity of its own code, it is capable of modifying the replicated copies. They are used for various purposes, including theft of information and/or money, creation of botnets, or rendering of advertisements [6].

A **Worm** is, similar to the virus, also a self-replicating malicious program, with the difference that it requires no human interaction to spread.mentions that it uses targeted vulnerabilities in the operating system or installed software and consumes a lot of bandwidth and processing resources through continues scanning. In doing so, it can destabilize the host, leading to potential crashes. It is also capable of the same actions as a virus by virtue of code payloads [6].

Trojans are a type of program that present themselves as legitimate software. Upon download and execution, it embeds malicious routines of files on the host system. Furthermore, whilst it is not capable of self-replication, it allows remote access to third-parties. Depending on the payload attached to it, a Trojan may display different ways of harm to the host system [6].

Spyware uses functions in the user's operating system to spy on user activity. They may be accompanied by additional capabilities, such as being able to interfere with network connections to modify security settings on an infected system. Spyware spreads by attaching itself to legitimate software, but can also be introduced through a payload. Tracked user activity include user behavior, keystrokes and internet usage. The tracked data is then sent to third-parties [6].

Adware, short for advertising supported software, is designed to automatically deliver advertisements to the host's system. This can be done via pop-ups, within other legitimate software or other means. These unwanted advertisements are used to generate revenue [6].

Rootkits, as the name implies, are used to gain continuous root access to a host's system and thus have high privileges. They employ different obfuscation techniques and are usually complex, and thus difficult to remove [6].

Bots, derived from the word robots, are designed to perform specific operations. By remotely taking control of a host's system, they are capable of spreading to other host computers and then form a network that is known as a botnet. These botnets are then used to perform the designated actions and are controlled by an attacker or bot master [6].

Ransomware is a program written to infect a host system or network, in order to hold it captive while requesting a ransom. This is usually done by encrypting the user's data and followed by a message being displayed that prompts the victim to pay a certain amount, in order to regain access to their data [6].

Cryptominers, as explained in the paper of [7], employ a technique that is also known as **cryptojacking** wherein a software or script is loaded onto a victim's machine, to stealthily mine cryptocurrency and thereby exhausting their resources. This generates income at the expense of the victim.

Understanding these distinct types of malware and their behaviors is crucial in creating an effective data collection system for malware analysis.

2.1.1 Evolution of Malware

The article of [8] shows how much malware has evolved, starting from its roots. Elk Cloner was the first piece of software written to spread itself. It would spread by cloning itself to new disks introduced to the system, and once triggered, it would display a poem explaining how Elk Cloner was copying itself through the victim's machine and that it may be no easy task to reverse its effects. This virus was attached to an actual game, written for the Apple

IIc's operating system, and served as a fairly harmless practical joke. Great interest was sparked during its time, and thus many other such pranks began showing up on bulletin board systems around the world. It did not take long for people to go beyond practical pranks and start creating malicious software designed to harm a victim's system. Soon after the first computer virus was spread, the first versions of antivirus software began being written as a public product in the latter half of the 1980s. From that moment on, malware has continued to evolve and become more sophisticated, whilst countermeasures have done so as well. Whereas at first "virus" was an overarching term for mischievous software, it later became reserved for malware that had the capability to attack its target machine, but could not replicate on its own. It was by the early 1990s, that malware was learning to obfuscate itself. So-called polymorphic viruses were created that could rewrite themselves whilst keeping their intended functionality. Thus, the so-called cat and mouse game between researchers and malware authors had begun. The continuous evolution and sophistication of malware necessitates the development of equally advanced and adaptable systems for data collection and analysis.

2.2 Dynamic Malware Analysis

As discussed in the state-of-the-art survey by [3], dealing with new and unknown malware, especially given the obfuscation techniques they employ to stay hidden, is no easy feat To determine whether an unknown or new executable is malicious or not, it is common to use an expert analyst, and whilst manual analysis is reliable, there is the issue of scalability. A tool to combat these issues is automated analysis. Herein, a distinction between static and dynamic analysis is made. Whilst static analysis relies on the extraction of information within the code of the sample to be analyzed, dynamic analysis gains information via direct execution and tracing of artifacts. One key aspect is that the security of the system could be compromised if no other measures are taken, as the sample code is loaded into the random access memory (RAM) and executed by the hosting central proccessing unit (CPU). The advantage of dynamic analysis, on the other hand, is that it is immune to the various obfuscation techniques that malware employs, as it does not rely on the analysis of the binary code itself. This is because whether the malware alters itself or not, the artifacts, or effects it causes, remain the same. Thus, a dynamic malware analysis tool is composed of three main components: The malware sample, the hardware and operating system being used, and finally, the analysis tool that is employed. The complexity of dynamic malware analysis and the inherent challenges it presents underscores the need for an effective, reproducible, and realistic data collection system.

2.3 Sandboxing

The survey of [3] makes further mention that, in order to combat the aforementioned security issues of running malware, a so-called guest-host model can be employed, in which malware and analysis are executed on separate operating systems. This can be done by virtue of virtual machines, a hypervisor, or an emulator. All of these options are

ways to ensure a secure environment, i.e. a sandbox. Employment of a sandbox comes with challenges, as malware can be capable of detecting whether it is being run in an analysis framework and then subvert its behavior. To avoid this from happening, the sandbox needs to avoid leaving footprints, such as analysis processes, registry keys and more. Because of these challenges, there is a constant arms race between malware authors and analysis tool developers. An advantage that sandboxes provide, on the other hand, is that there exist a more effective approach for returning the system to a clean state. Whereas bare metal approaches require completely replacing the hardware or reformatting the system, sandboxes can make use of so-called snapshots. Snapshots are a process in which the guest operating system (OS), in its current state, can be saved to a file for later use. They include the RAM and file system of the guest, thus comprehensively representing a machine's state.

2.4 Data Collection in Dynamic Malware Analysis

The process of data collection in dynamic malware analysis is pivotal, instrumental to elucidating the behaviors exhibited by malware. [9] illustrates the use of function call monitoring, which pertains to the interception of calls made by a program to designated functions. This method leverages the power of abstraction provided by functions, which distill implementation details into semantically richer representations. This is achieved through *hooking*, a technique in which a supplementary function, known as the hook function, is invoked concurrently with the intended function. This hook function is accountable for implementing desired analysis functionalities, which may include recording its invocation or closely inspecting input parameters. Interception targets primarily consist of Application Programming Interfaces (APIs) and system calls. APIs are groups of functions presenting a cohesive set of functionalities frequently utilized by applications to execute tasks, while system calls represent the primary mode through which user-mode applications request the operating system to perform tasks.

The survey by [9] also highlights other crucial techniques:

Function Parameter Analysis: This process involves the dynamic tracking of actual values passed when a function is invoked. It aids in correlating individual function calls to provide a deeper understanding of the program's behavior.

Information Flow Tracking: Here, the propagation of *taint-labels*, or specific data, is tracked throughout system execution. This aids in observing how a program manipulates and processes data.

Instruction Trace: This involves collecting and analyzing sequences of machine instructions that have been executed, potentially unveiling insights not immediately evident in higher-level analysis reports.

Autostart Extensibility Points (ASEPs): Monitoring these points is critical as malware often leverage ASEPs to ensure their activation during the system's boot process or when specific applications are initiated.

2.5 Reproducibility

The article of [10] highlights that the term ”reproducibility” is nonstandard and unsettled across sciences. However, the article makes mention that the term was coined by Claerbout, a geophysicist known for his work in the field of signal processing and notably, the development of reproducible research [11]. He originally makes mention that reproducibility is achieved with transparency, by association with a software platform and a set of procedures that permit the reader of a paper to see the entire processing trail from the raw data and code, to figures and tables. The article also mentions that the U.S. National Science Foundation subcommittee on replicability in science defines reproducibility as the ability of a researcher to duplicate the results of a prior study using the same materials as were used by the original investigator. A second researcher should thus be able to build the same analysis files and implement the same statistical analysis, with the same raw data in an attempt to yield the same results. This paper sticks to these definitions, and as further supported by the reproducibility award, given by [12]. Badges are rewarded according to whether results can be reproduced (as closely as possible if hardware differs), artifacts are available, and whether the artifacts are evaluated and reusable. Reproducibility, as defined and practiced in this research, is a key attribute of the proposed data collection system, ensuring consistent and verifiable results across different instances of dynamic malware analysis.

Chapter 3

Related Work

To gain an understanding of whether reproducibility is promoted in current and past research, this chapter offers a high-level look at the context of computer research to identify properties enabling reproducibility, followed by a summary of relevant dynamic malware analysis tools and research.

3.1 Methodology

A semi-systematic literature research containing various combinations and variations of the terms malware, reproducible, analysis, research, and cybersecurity on Google Scholar yielded no papers specifically focusing on reproducibility within the context of dynamic malware analysis. However, in the broader context of computer research, there is a plethora of relevant studies. Thus, a summary of the findings of the broader context is first made.

In order to understand how reproducibility can be promoted and enabled, the higher level findings are used to draw out reproducibility properties, as these properties can be applied to computer research in general. The properties are then used to highlight whether different research within the context of dynamic malware analysis allows for reproducibility. It should be noted that the focus was primarily on highly relevant or recent tools and research. Other tools that could have been included were omitted, considering how similar the missing properties to enable reproducibility were.

3.2 Reproducibility in Computer Research and Its Application in Dynamic Malware Analysis

This section will now first examine what constitutes reproducibility and the associated challenges within the broader context of computer research. Following that, a closer look is taken into different dynamic malware analysis tools. That is, to which extent

reproducibility principles have been integrated into their functionality and whether gaps exist.

3.2.1 Enabling Reproducibility in Computer Research

Within his paper, [13] shows that reproducibility has the potential to serve as a minimum standard for judging scientific claims when full independent replication is not possible. He created a spectrum of reproducibility that starts from publication only, on to full replication. In order to facilitate reproducible research, written code should be published, ideally cleaned-up, along with the corresponding metadata and data sets used. As a further aspect, the role of metadata in reproducible computational research is explored in the paper of [14]. Method details, such as versions and parameters, but also steps along the entire scientific process should be written down, including data collection and selection strategies, and finally, hardware and statistical methods as well that link these elements to publication.

In the case study of [15], a survey, addressed to researchers in academic and private sectors, was made. They used the findings to create guidelines to enable reproducibility. The reproduction process should be highly automated (this could be achieved with an execution script) as this decreases manual intervention, which may introduce variability and inconsistency during analysis. Published code should be provided as source code and/or run within a virtual environment to address security issues. Commercial libraries and other components that are locked behind payments should be avoided, and the software and environment for the reproduction process should stay available. A possible tool that is mentioned to achieve a lot of these points includes Docker, as the Dockerfiles can contain simple creation instructions for images and can be easily shared due to their small size. Furthermore, this ensures controlled and reproducible environments.

To summarize the prior findings, the following points need to be taken into consideration when aiming for reproducibility:

- Source code provided, ideally cleaned up.
- Data set is publicly available or provided.
- Metadata, including but not limited to method details, the steps taken along the entire scientific process, and the hardware as well as statistical methods has to be included.
- The published code should be run within a virtual environment to address security issues.
- Commercial libraries and other components locked behind payments should be avoided.
- The reproduction process should be highly automated as manual intervention is prone to introducing variability and inconsistency.

- The software and environment, as well as datasets used, need to stay available as research papers stay relevant far into the future.

3.2.2 Reproducibility in Dynamic Malware Analysis

This subsection will examine whether the properties and practices identified before, are applied within the context of dynamic malware analysis. A heavier emphasis is put on tools and research included within the surveys done by [9] and [3]. Dynamic malware analysis systems not running on virtual environments will be skipped, as these already run a security risk and are thus an obstacle to reproducibility.

TTAnalyze, which is an extension of the QEMU software, is introduced in the paper of [11]. Malware is dynamically analyzed via execution within an emulated operating system environment, simulating Windows XP, where the actions are monitored. To be more precise, malware samples are loaded into a controlled environment. Afterwards, it then uses a method of hooking system calls to monitor the actions of the malware. A log is generated during execution, which includes API calls made by the malware and their parameters, changes to the files system or registry, and network activity. Theoretically, as the analysis of malware behavior can be automated, the data collection part could be reproducible. However, this is only speculation as the source code is not supplied. The software itself continued work under the name Anubis [16]. Later on, it underwent another name change to **LastLine** and became commercial software.

CWSandbox, presented in the work of [17], executes malware samples in a simulated environment, much like TTAnalyze and, as will be discussed further below, DRAKVUF. It monitors all system calls, thereby automatically generating a detailed report. Because they operate similarly, a comparable basis in the realm of dynamic malware analysis can be drawn. Furthermore like Ether, which will be discussed in the next section, the software functions by proactively monitoring the actions of malware during execution, with the difference that API hooking and DLL injection is used, thus making it more versatile. A log is generated with a wide range of activities, including files system and registry modifications, network activities and API calls and great emphasis is put on the automation of the software. As neither source code, nor data sets used are provided, only a speculation can be made that the data collection part could be reproducible as well, as the analysis process is automated. The software is currently only available commercially, as it later continued work under the name Threat Track and is now known as **Threat Analyzer** [18].

Ether, which is another example of a dynamic analysis software that is unfortunately closed source, is found within the works of [19]. It is designed to overcome some limitations that traditional analysis software presents, mainly how malware can obfuscate itself when discovering that it is being analyzed. The software itself is described in low level detail, such as it leveraging Intel VT [20], but further meta data cannot be gathered. Transparency is achieved by utilizing hardware virtualization extensions. There is no mention of automation within the paper.

DRAKVUF, first presented in the work of [21], uses hardware-assisted virtualization extensions to create an isolated, controlled and consistent environment for malware analysis. It aims to achieve high levels of stealth and is currently still maintained. Furthermore, it is open source and well-documented [22]. The software is capable of monitoring at both kernel and user-space levels. Automation is supported to some extent. DRAKVUF is still maintained and as it is open source, commercial libraries and components aren't in use. Data collection is done by capturing every system call and many types of hardware events and a detailed log is generated that provides rich metadata about the behavior of each malware sample. The main challenge of reproducibility is that it is user reliant to ensure the properties being held.

Cuckoo Sandbox [23] is a widely-used, open source malware analysis tool, that is currently unmaintained, but a full rewrite is in progress. It allows configuration of various aspects of the analysis environment, including virtual machines and network settings. It is also capable of generating its analysis results, which supports sharing and collaboration, and thus in turn reproducibility. Further, a thorough documentation is available online. Analysis includes techniques, such as hooking system calls, tracking of file systems and registry changes, and capture of network traffic. To analyze malware, a suspicious file or a URL is submitted into the tool. The sandbox then sets up an isolated environment, to execute the submitted file in. The previously named techniques are then used to collect a vast amount of data about its behavior. Furthermore, it is also capable of taking screenshots to capture visual activity of the malware. The software is open source, and thus does not use any commercial components. Once set up, the software is capable of processing multiple samples in a row without manual intervention. It also includes detailed logs and thus provides comprehensive metadata. The one hindrance to reproducibility is the same one as with DRAKVUF: User reliability and responsibility.

Finally, it is important to acknowledge the presence and significance of commercial dynamic malware analysis tools, such as SOCRadar's Threat Intelligence [24], any.run [25], and Joe Sandbox [26]. These tools are widely used in the industry and provide crucial services in malware detection and analysis. However, due to their closed-source nature, full assessment of their reproducibility properties remains challenging. This emphasizes the transparency advantage of open-source tools.

3.2.3 Discussion

From the prior findings, a definite lack of focus on the discipline of reproducibility can be gathered, even though the usage of sandboxes already facilitates reproducibility. Virtual environments provide a stable testing ground, where configurations can be set prior to the analysis step. The research papers didn't provide their source code, but supplied the steps to achieve similar code. Two of the research papers presented within the prior section became commercial software. The open source tools proved to be ideal to use for reproducible research, but are user reliant to do so. If the usage of these tools were less user heavy and introduced automation of the reproduction process to lessen manual intervention, the open source tools would already fulfill all necessary properties. The two open source solutions require the user to manually set up and customize the environment.

To fully ensure reproducibility, it is essential that the users document their setup, configuration and analysis procedures in detail, but also that they share their malware samples and raw analysis data whenever possible. This is something that could be automated or facilitated within SecBox with the help of scripts and prompts. A further common obstacle is the usage of a non public malware data set, or missing information on how these data sets are used. Within the Table 3.1, ✓ will refer to a property being fulfilled, × to a property not being fulfilled, and "!" to a property being partially fulfilled or being user reliant.

Table 3.1: Comparison of Reproducibility Properties Across Analysis Tools

	TTAnalyze	CWSandbox	Ether	DRAKVUF	Cuckoo Sandbox	This Work
Source code provided	×	×	×	✓	✓	✓
Data set available or provided	×	×	×	×	×	✓
Metadata	!	!	!	!	!	✓
No commercial libraries/components	×	×	Unknown	✓	✓	✓
Automated Process	unknown	unknown	×	!	!	!
Long-term availability	×	×	×	✓	!	!

Chapter 4

Architecture

This chapter provides an overview of the architecture of the proposed solution. The objective is to provide a high-level overview of the implementation in order to gain a sufficient understanding of the system and potentially reimplement it using different technologies.

4.1 Overview of the System

SecBox [4] is a collaborative, container-based, lightweight dynamic malware analysis sandbox platform. It provides access to a selection of malware samples, allows real-time interaction with the sandbox via terminal interfaces, and provides interactive visualization capabilities. Furthermore, it features a multi-step analysis process that allows users to discern malicious behavior using a baseline instance as a reference. Downstream analysis tasks are supported through the export of system call and network packet data.

4.2 Enhancements for Reproducibility

To enhance SecBox's reproducibility features, this thesis introduces two important features. These enhancements aim to ensure that the system adheres to the principles of reproducibility that are discussed in the prior chapter.

Firstly, a mechanism is implemented that records terminal commands issued to the sandboxes. The automation aspect of reproducibility is addressed by this feature. These recordings can be output into JavaScript Object Notation (JSON) files that can be uploaded into the SecBox environment to precisely replicate the original analysis process, maintaining the same command sequence, timing, and the terminal it corresponds to. This allows other users to repeat analysis and experimentation of malware samples.

Secondly, a Comma-Separated Values (CSV) generator has been implemented to accurately track and record CPU and RAM usage. This implementation was motivated by the need to provide more detailed and accurate metadata for analysis sessions. It replaces

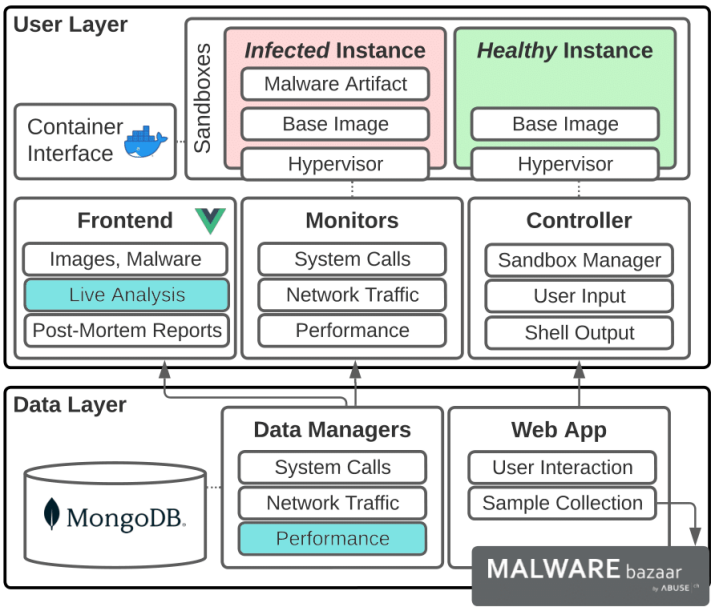

Figure 4.1: The SecBox Architecture, marked in blue are the parts, where changes were implemented [4]

the malfunctioning post-analysis graphs for performance metrics in the original platform, thereby enhancing the reproducibility and comparability of analysis sessions.

4.3 Core Components

This section provides an overview of the key architectural components within the SecBox platform and their functionalities. The system is composed of three main components:

4.3.1 Frontend

The frontend, constructed using Vue 3, primarily facilitates user interaction and data visualization. It comprises several navigable pages to guide users through different phases of the analysis process. These pages include:

Home Page: A welcoming interface for users when they first access the system. It displays some recent reports. From here, users can initiate the analysis process by selecting a malware sample and a Linux image from respective drop-down lists.

Live-Analysis Page: Enables users to monitor and interact with running sandboxes in real-time. Within a component called the live terminal, the recording feature introduced in this work comes into play. The exact sequence of commands issued to the terminals corresponding to both infected and healthy instances of the sandbox are recorded and can be reused to replicate the same analysis in the future. A helper function within the live analysis component had to be implemented as well to enable dynamic and automatic

switching between the combined and split terminal. This feature is crucial for the reproducibility aspect introduced in this thesis, as this allows a user to rerun the exact same commands, at the same time again.

Post-Analysis Page: Allows users to select relevant charts that visualize the results of their analysis.

Report-Page: Provides a platform for users to annotate the selected charts, facilitating the interpretation and understanding of the data.

Report Dashboard: Lists past analyses, thus offering a convenient way for users to revisit previous work.

4.3.2 Backend

The backend component serves as the main conduit for data flow between the client and the host, while also managing user interactions. It communicates via WebSockets with the frontend and host, providing real-time data transfer. Each host monitor has a corresponding data manager in the backend that processes and forwards the data to the clients involved in the analysis process. This work implements a CSV generator within the performance manager that is automatically invoked, when data is handled. The generated CSV files contribute to the overall reproducibility feature of SecBox, as detailed in the "Enhancements for Reproducibility" section. User interaction tasks such as controlling the sandboxes, handling command prompts, and managing reports are also handled by the backend. The MongoDB database is utilized for data storage, specifically storing previous analysis results, raw data, and information on host capabilities. This feature plays a critical role in ensuring reproducibility, as it allows for the consistent storage and retrieval of analysis data.

4.3.3 Host

The host is the core component of the SecBox solution, responsible for malware isolation and data collection. It is built using Python and uses Socket.IO for communication with other components. The host consists of three key sub-components:

Controller: Manages interaction with the sandbox using the Docker software development kit (SDK). The controller is in charge of starting, stopping, and interacting with gVisor containers, which provide added separation between the running applications and the host operating system.

Sandbox: Executes malware in an isolated environment. The malware samples are pulled directly from Malware Bazaar [27].

Monitors: Extract data from the sandbox and transmit it to the respective data managers in the backend. The monitors use different means to extract various types of data such as performance metrics, network metrics, and system call monitoring.

4.3.4 Running a Reproducible Experiment

Once the installation and initial setup process is complete, the user will need to select an operating system and a malware sample. Following this, two terminal windows are presented, to which the user can send any number of commands as desired.

Upon completion of the experiment, the user can generate a record of their inputs by pressing the *GENERATE AND DOWNLOAD SCRIPT* button. This action triggers the download of a JSON file named *commandsScript.json*. This file encapsulates all issued commands, the terminal to which they were sent, and their relative timestamps.

To reproduce the experiment, the same user or a different user can upload the previously generated script using the *UPLOAD AND RUN SCRIPT* button, after duplicating the original setup. This will trigger the re-execution of the commands in the corresponding terminals, respecting the same delays that occurred in the original experiment.

Chapter 5

Implementation

This chapter details the process involved in implementing the two enhancements to the SecBox platform: the recorder and the CSV generator. The enhancements are primarily designed to improve the reproducibility of malware analysis sessions by providing robust automation and precise system resource tracking.

5.1 Porting to ARM

Originally, the plan for this thesis was to deploy SecBox on a Raspberry Pi 3 and subsequently enhance the platform's reproducibility features. This was based on the decision that Internet of Things devices face unique malware analysis challenges and thus have become the focus of the security community in recent years [28]. However, this approach led to several compatibility issues. The bash shell script files (SH files) provided could not be utilized as multiple software components were either incompatible with the Raspberry Pi or required different installation instructions. Moreover, the Raspberry Pi 3's RAM proved insufficient, which led to the decision to switch to a Raspberry Pi 4. Despite this change, further complications arose due to missing syscalls in gVisor within an Advanced RISC Machine (ARM)64 based system [29], for which no workarounds or community efforts were available. Consequently, the final decision was made to switch to a classic Linux system.

5.2 Command Recorder

The command recorder is implemented as a part of the **'LiveTerminal.vue'** component, which is a crucial part of the frontend where user interaction with the sandbox takes place. The main aim of the recorder is to keep track of the commands issued to the different terminal interfaces (combined, clean and infected) along with their relative timestamps.

Issued commands, the terminal type it was sent to, and the timestamp are pushed into an array every time a command is entered in any of the terminal interfaces. The array thus

stores the exact sequence and timing of the commands issued, enhancing the system's ability to replicate the analysis process with high fidelity.

The recorder offers three key functions to aid in reproducibility:

DownloadScript: This function facilitates the download of the recorded commands in a JSON file. The downloaded file can serve as a record of an analysis session, and can be used to reproduce the same session later.

It is composed of two methods, generateScript and downloadScript. The following lines show the implementation of the script generator. Note that the line involving timestamp serves to convert it into a relative one. This means that the amount of milliseconds starting from the first command is tracked for each subsequent one.

Listing 5.1: Method, responsible for the generation of a command script

```
generateScript: function() {
    let commands = []
    let firstCommandTimestamp = this.commands[0].timestamp;

    for (let commandObj of this.commands) {
        commands.push({
        terminal: commandObj.terminal,
        command: commandObj.command,
        timestamp: commandObj.timestamp - firstCommandTimestamp
        });
    }

    let script = JSON.stringify(commands);

    return script;
}
```

In order to make the generated script downloadable, another method called downloadScript is created that leverages an invisible anchor element and also simulates a link click, thus downloading the JSON file. This is shown in the next few lines.

Listing 5.2: Method, responsible for the download of a generated command script

```
downloadScript: function() {
  const scriptContent = this.generateScript();
  const scriptName = 'commandsScript.json';

  let element = document.createElement('a');
  element.setAttribute('href',
  'data:application/json;charset=utf-8,'
  + encodeURIComponent(scriptContent));
  element.setAttribute('download', scriptName);

  element.style.display = 'none';
```

```
    document.body.appendChild(element);

    element.click();

    document.body.removeChild(element);

}
```

UploadScript: This function allows users to upload a previously recorded command script. The command script is parsed and automatically executed in the system.

Listing 5.3: Method, responsible for the upload of a generated command script

```
uploadScript: function() {
const inputElement = document.createElement('input');
inputElement.type = 'file';
inputElement.accept = '.json';
inputElement.onchange = (event) => {
  const file = event.target.files[0];
  const reader = new FileReader();
  reader.onload = () => {
    const commands = JSON.parse(reader.result);

    this.runScript(commands);
  };
  reader.readAsText(file);
};
inputElement.click();
}
```

RunScript: This function takes the parsed commands and replays them in the system, effectively recreating the analysis session. The function includes an internal mechanism to respect the timing between the commands, thus ensuring a faithful reproduction of the original session. It also uses a key called terminal to determine whether the terminal needs to be switched, thus ensuring that the commands are sent to the correct terminal.

Listing 5.4: Method, responsible for the replay of a generated command script

```
runScript: async function(commands) {
for (let i = 0; i < commands.length; i++) {
  const commandObj = commands[i];
  const command = commandObj.command;
  const terminal = commandObj.terminal;
  const delay = i === 0 ? 0 :
  commands[i].timestamp - commands[i-1].timestamp;
  this.$emit('update-combined-cli', terminal === 'combined');

  await new Promise(resolve => setTimeout(resolve, delay));
```

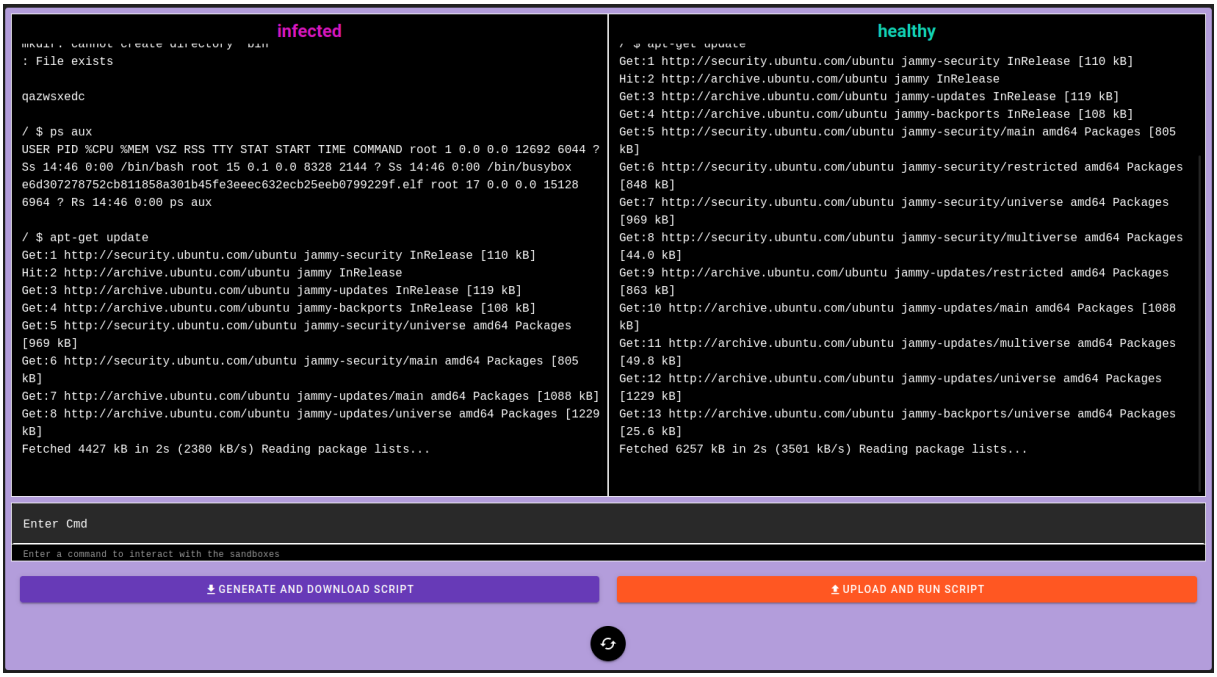

Figure 5.1: The download and upload button that has been added to aid in reproducibility

```
        if (terminal === 'combined') {
          this.cli_text = command;
          this.onEnter();
        } else if (terminal === 'clean') {
          this.cli_text_clean = command;
          this.onEnter();
        } else if (terminal === 'infected') {
          this.cli_text_infected = command;
          this.onEnter();
        }
      }
    }
```

Finally, to make the command recorder accessible to users, two buttons have been added to the user interface, located just below the terminal interfaces: one for downloading the recorded command sequences, and another for uploading.

5.3 CSV Generator

The CSV generator plays a key role in enhancing the reproducibility of malware analysis sessions within SecBox by enabling accurate and reliable tracking of system resource usage. Integrated within the Performance Manager in the backend of the system, it records metrics related to CPU and RAM usage in a systematic and automated way.

The generator operates by extracting data from the system's stats that are obtained via docker. To calculate CPU usage, it divides the total CPU usage by the system CPU usage, factoring in the number of online CPUs. The result is multiplied by 100 to obtain the CPU usage percentage.

For RAM usage, the generator divides the current memory usage by the memory limit. This fraction is then multiplied by 100 to represent the RAM usage as a percentage.

A key feature of the CSV Generator is its automatic invocation. Every time data is sent to the backend, the generator is automatically triggered, ensuring that all relevant resource usage data is captured. The generated CSV files serve as a reliable record of the system's performance during an analysis session. This allows for an accurate reproduction of the resource usage graphs in the post-analysis phase, thereby contributing to the system's reproducibility.

Chapter 6

Evaluation

This chapter discusses the experimental setup and results of the evaluation conducted to test the reproducibility features introduced into the SecBox platform. The evaluation aims to provide empirical evidence supporting the claim that the implemented features, that being the command recorder and CSV generator, enhance the reproducibility of malware analysis within the SecBox platform.

6.1 Experimental Setup

The evaluation was conducted on a system with the following specifications:

- Operating System: Ubuntu 22.04.2 LTS
- Processor: AMD Ryzen 9 5900X 12-Core Processor, 4200 Mhz, 24 logical Processors
- Physical Memory (RAM): 32.0 GB

Changes in system configurations, such as different hardware specifications or internet connection speeds, may impact the performance and reliability of the command execution and, therefore, the reproducibility of the malware analysis.

Each experiment is carried out by initially setting up the sandbox with an Ubuntu Jammy container image, and the specified malware sample. Then a set of commands is run in the different terminals and a JSON file is then generated from the sent commands. Finally, the script is uploaded in 9 additional instances, in order to test for reproducibility. The commands are kept basic, since the only relevant metric is whether the patterns remain the same between each analysis run.

The set of commands run in the different terminals includes basic setup and operations such as setting up the malware and performing 'apt-get update'. For the Monti ransomware, the `cat readme.txt` command was also used to display the ransom instructions.

However, the platform only executes commands after the completion of the prior command, and the script currently does not detect failed commands. Therefore, these tests and results are contingent upon successful command execution. The reference malware omits the 'apt-get update' command.

The data generated during each run, specifically the resource utilization metrics, are recorded by the CSV generator and subsequently used for analysis and comparison. The read/write counts are taken from the post analysis graph.

Additional files sourced from Digital Corpora [30] were added to the home directory in both Docker environments to enhance the realism of the experiment. The downloaded ZIP file was extracted beforehand.

6.2 Data Analysis Methodology

Data from the CSV files is processed using the pandas [31] library, an open-source data manipulation and analysis tool in Python. Graphical representations of the processed data are created using matplotlib [32], a plotting library for Python and its numerical mathematics extension, NumPy [33]. The cosine similarity graph is generated with the help of scikit-learn [34]. The cosine similarity matrix compares the RAM and CPU usage patterns between each sandbox by comparing the corresponding means.

The equation looks as follows:

$$\text{cosine\_similarity}(A, B) = \frac{a_1 \cdot b_1 + a_2 \cdot b_2}{\sqrt{a_1^2 + a_2^2} \times \sqrt{b_1^2 + b_2^2}}$$

Where a_1 and a_2 correspond to the mean CPU and RAM usage of one sandbox instance and b_1 and b_2 to the mean CPU and RAM usage of a second instance.

6.3 Results

The effectiveness of the command recorder and CSV generator in enhancing the reproducibility of the SecBox platform was tested using three different types of malware. Each malware type was analyzed across ten trials to measure the consistency of the system's resource utilization metrics (CPU usage, RAM usage). The first two malware types include the read/write count, whilst the last includes the directory graph. The RAM and CPU usage are scaled from 0 to 1.

6.3.1 Reference Malware

A reference malware with a predictable script is created to ensure accurate results. The script is written as follows:

```
#!/bin/bash

# Create the govdocsdataset directory if it doesn't exist
# and populate with sample files
if [ ! -d "/root/govdocsdataset" ]; then
    mkdir /root/govdocsdataset
    echo "Sample content" > /root/govdocsdataset/file1.txt
    echo "Another sample" > /root/govdocsdataset/file2.txt
fi

# Create a few large files
for i in 1 2 3; do
    dd if=/dev/zero of=/tmp/${i}.img bs=1M count=1024
    sleep 30
done

# Read a bunch of files from /root/govdocsdataset
for file in /root/govdocsdataset/*; do
    cat $file
    sleep 5
done

# Scan a predictable number of hosts.
for i in $(seq 1 5); do
    ping -c 1 192.168.1.$i
    &> /dev/null && echo "192.168.1.$i is up" &
done
```

Running the script within the infected instance led to higher CPU and RAM usage as can be seen in 6.1 and 6.2. This is as expected as the script creates large files and performs reading operations as well. This is reflected in 6.4 as well. The cosine similarity graph, seen in 6.3, shows a high level of similarity as well, which points to the reproducibility of the script being successful. Finally, because all network layer graphs, as shown in 6.5, of the different instances looked exactly the same, the graph has been included as well.

6.3.2 Mirai Analysis

Mirai [35], a malware type typically used in large-scale network attacks, forms the first part of the analysis.

The Figures 6.6 and 6.7 show that the usage is remarkably similar between each sandbox instance. There is less CPU and RAM utilization within the infected instance and also less variation as can be seen with the smaller standard deviation error bars. Read and write operation counts, as seen in Figure 6.9 are higher within the healthy instances, with one notable spike. This could hint at Mirai causing the system to move into a highly passive state. The figure 6.8 shows that the similarity cosine between each sandbox is

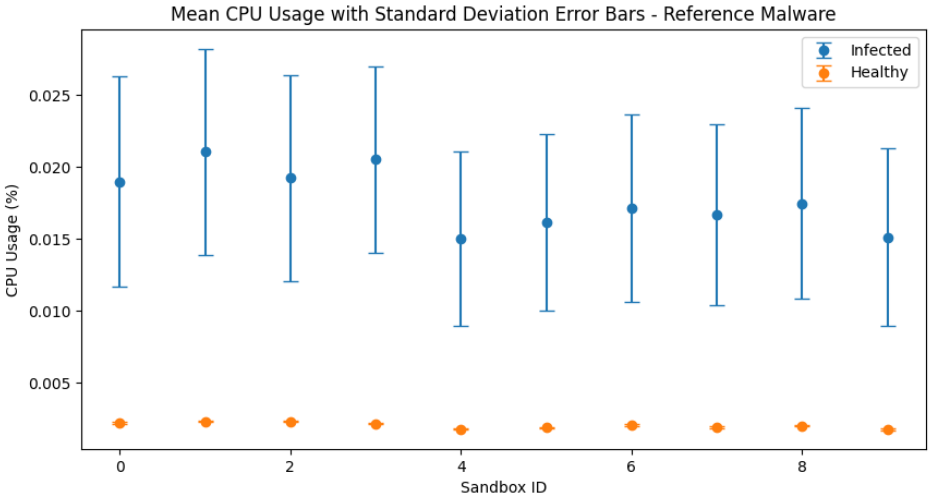

Figure 6.1: CPU Usage Across Multiple Runs With Reference Malware

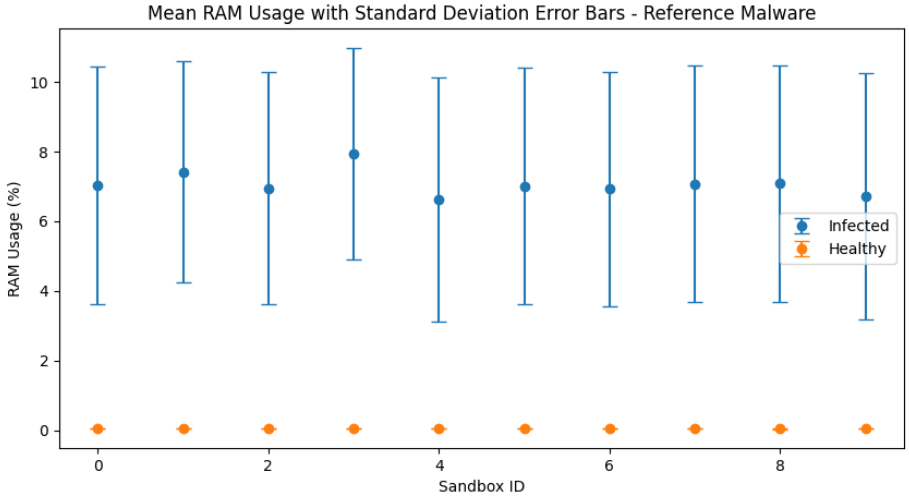

Figure 6.2: RAM Usage Across Multiple Runs With Reference Malware

extremely close to 1.0, indicating that the patterns of CPU and RAM usage are the same between each instance.

6.3.3 Monti Analysis

The ransomware called Monti forms the second part of the analysis.

The graphs in figures 6.10 and 6.11 show an increase in CPU and RAM usage within infected instances, with the RAM usage being significantly higher. There was one spike in CPU usage within one instance. The read/write count in figure 6.13 reflects this as well, with the infected instances having higher amounts. The cosine similarity graph in

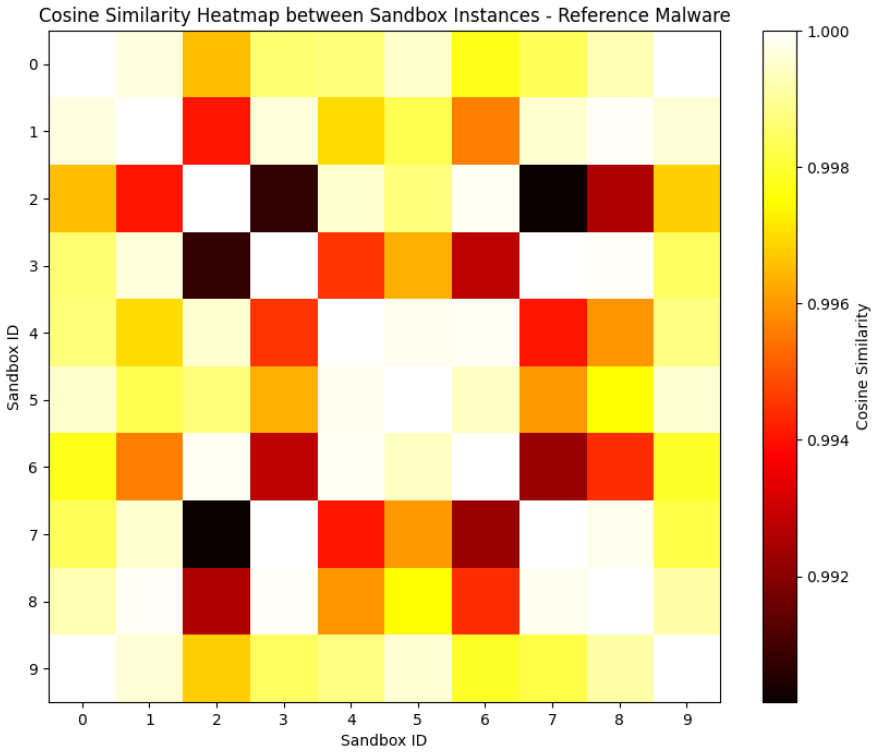

Figure 6.3: Cosine Similarity Across Multiple Runs With Reference Malware

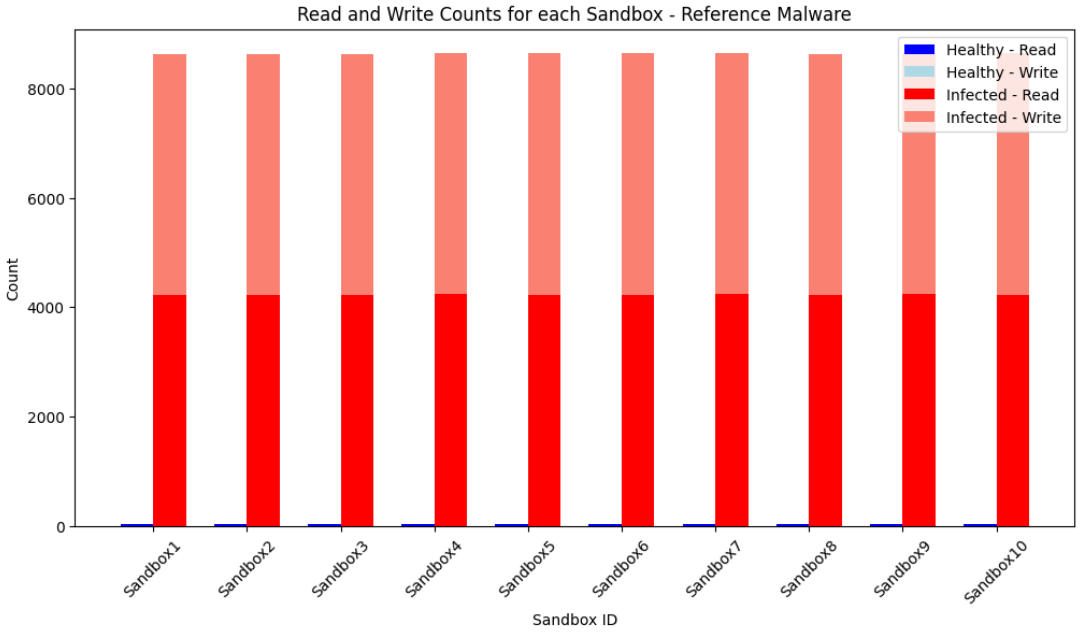

Figure 6.4: Read/Write Counts Across Multiple Runs With Mirai Sample

figure 6.12, just as was the case with the Mirai malware, ranges from around 0.92 to 1.00, thus showing that the instances all show the same patterns.

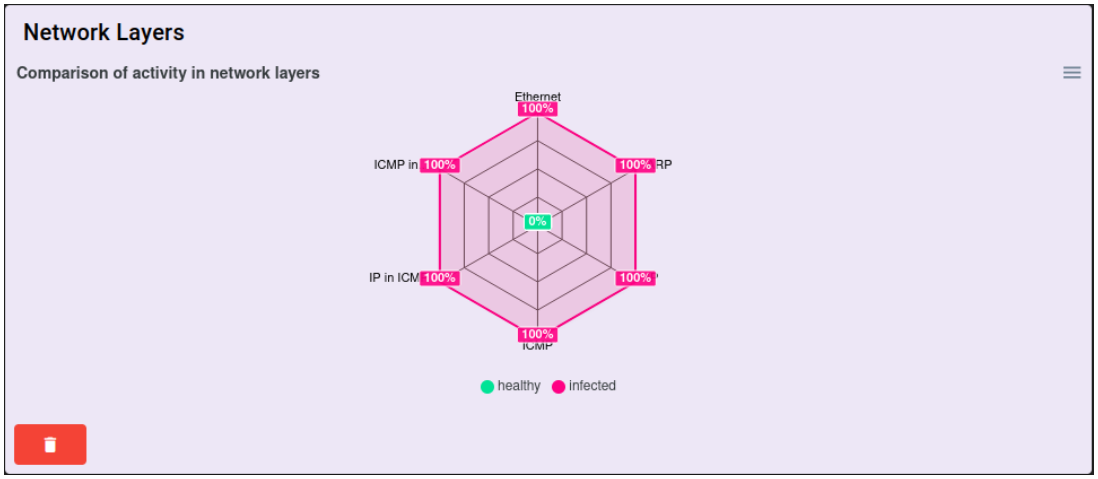

Figure 6.5: Network Layers Of One Instance With Reference Malware

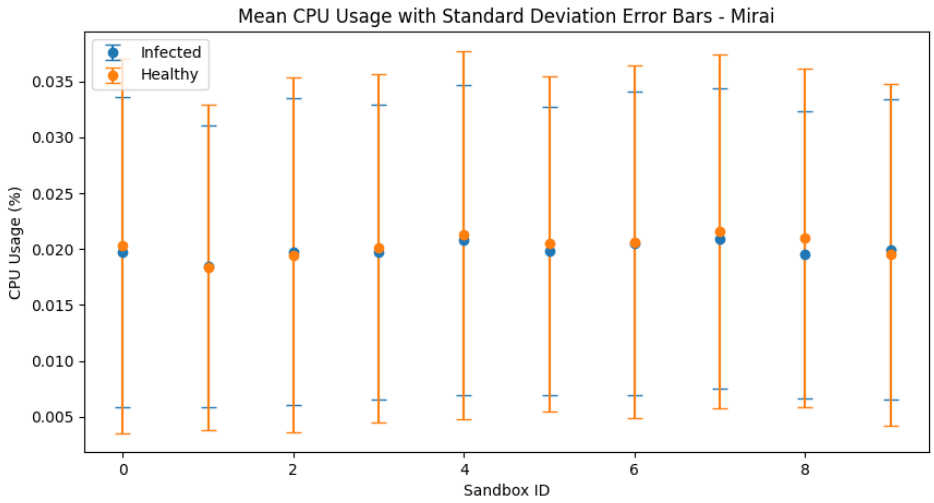

Figure 6.6: CPU Usage Across Multiple Runs With Mirai Sample

Within the SecBox platform itself, a readme.txt file is created within the directory that is encrypted, containing instructions on how to proceed. All of the files within the encrypted directory now have a '.puuuk' file extension. This indicates that the malware operates as expected, and explains how the higher read/write count comes to be.

6.3.4 Coinminer Analysis

In the final analysis, this work utilized the script named Coinminer F from the original paper of the SecBox platform [4]. This decision was made because the script obtained from the platform did not function correctly, occasionally leading to crashes.

Upon execution, there is a noticeable and consistent increase in CPU and RAM activity, as shown in figures 6.14 and 6.15. The cosine similarity matrix in figure 6.16 is extremely close to 1 across the board, pointing to the patterns also being the same between each

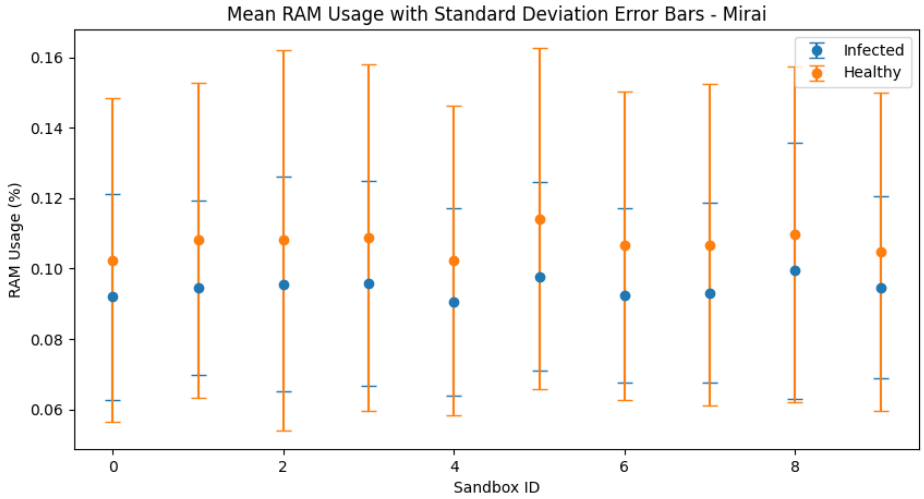

Figure 6.7: RAM Usage Across Multiple Runs With Mirai Sample

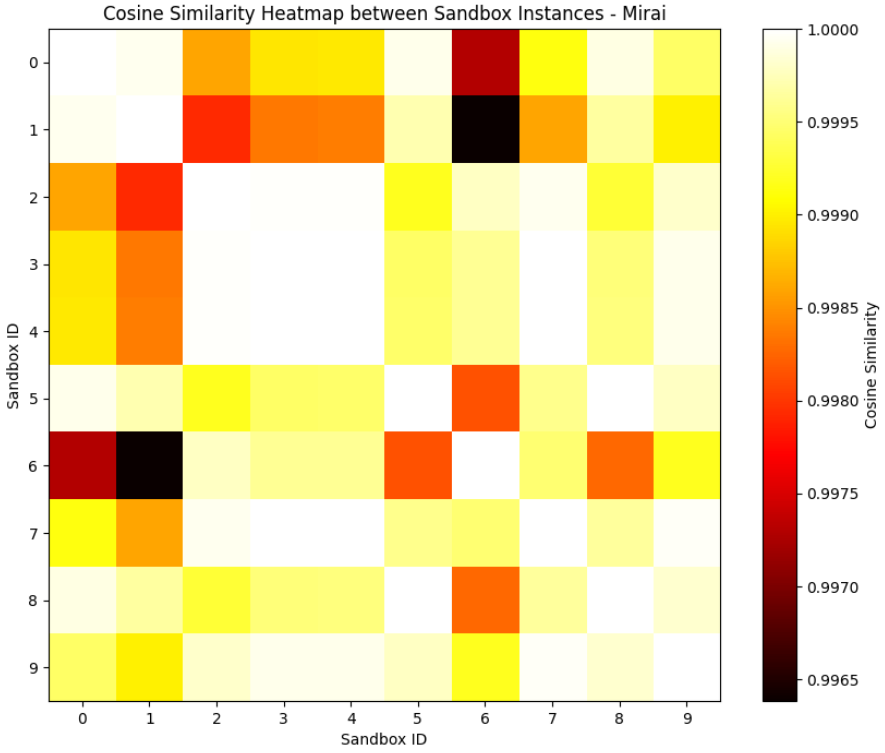

Figure 6.8: Cosine Similarity Across Multiple Runs With Mirai Sample

instance. Lastly, as the post analysis graph of the read/write count wasn't functioning properly, the directory graph was included instead, as the generated sunburst chart of figure 6.17 showed the same result between each instance. Notably, two directories are created, with one being marked for deletion.

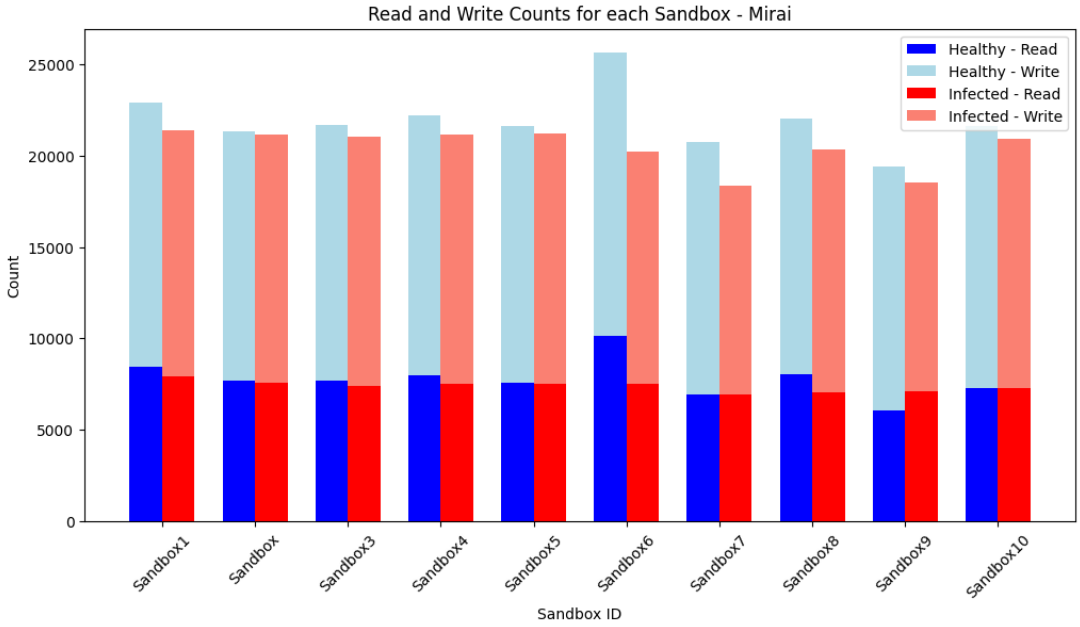

Figure 6.9: Read/Write Counts Across Multiple Runs With Mirai Sample

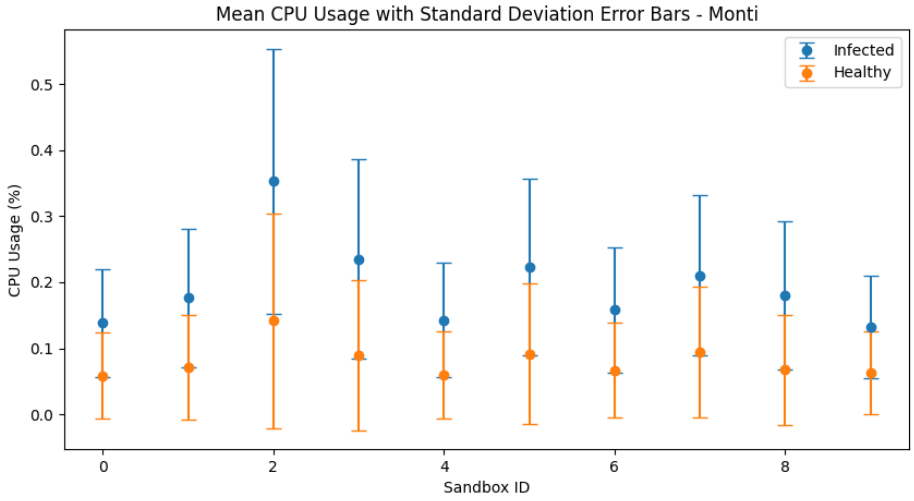

Figure 6.10: CPU Usage Across Multiple Runs With Monti Sample

6.4 Discussion

The goal of this study was to investigate the effects of implementing reproducibility features, namely the command recorder and CSV generator, into the SecBox platform. Across three different malware analyses, the study found a high degree of consistency in system resource utilization metrics such as CPU and RAM usage, read/write operations, and directory graphs across different sandbox instances.

The reference malware analysis showed increased CPU and RAM usage across all infected instances, and higher read/write counts than the healthy instances. The cosine similarity

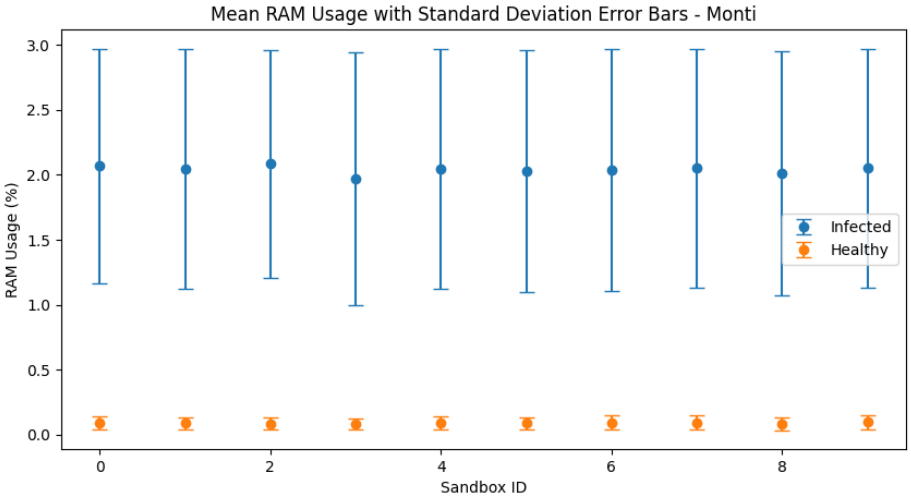

Figure 6.11: RAM Usage Across Multiple Runs With Monti Sample

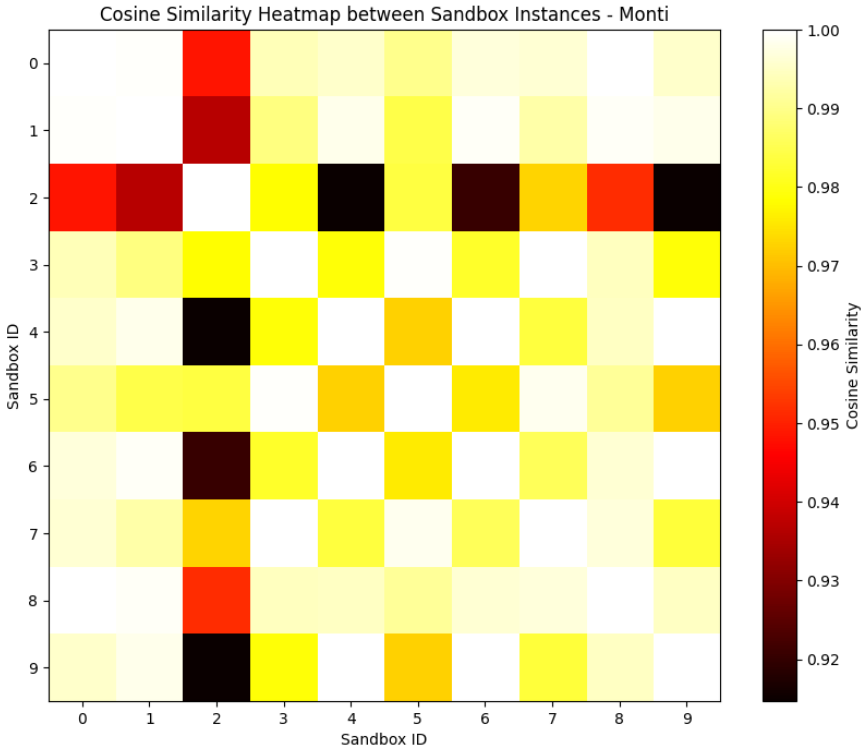

Figure 6.12: Cosine Similarity Across Multiple Runs With Monti Sample

scores are very close to 1.0 and all network layer graphs, showed the same layers being used in the infected instances. This shows a remarkable success in reproducibility.

The Mirai malware analysis showed that the usage patterns were remarkably similar across all instances with a cosine similarity score close to 1. This implies a high degree

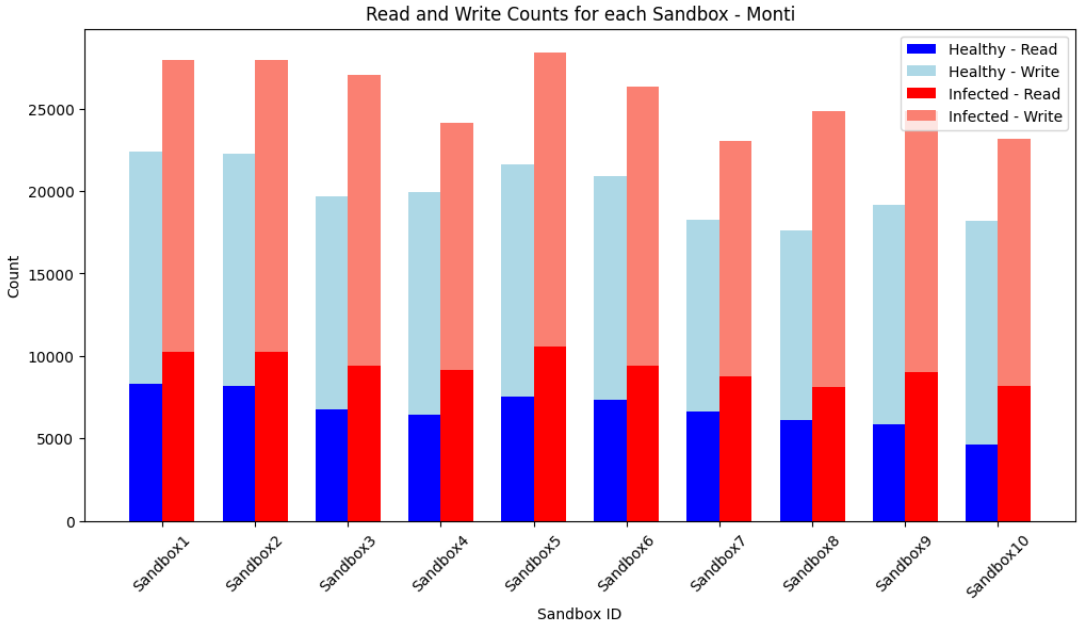

Figure 6.13: Read/Write Counts Across Multiple Runs with Monti Sample

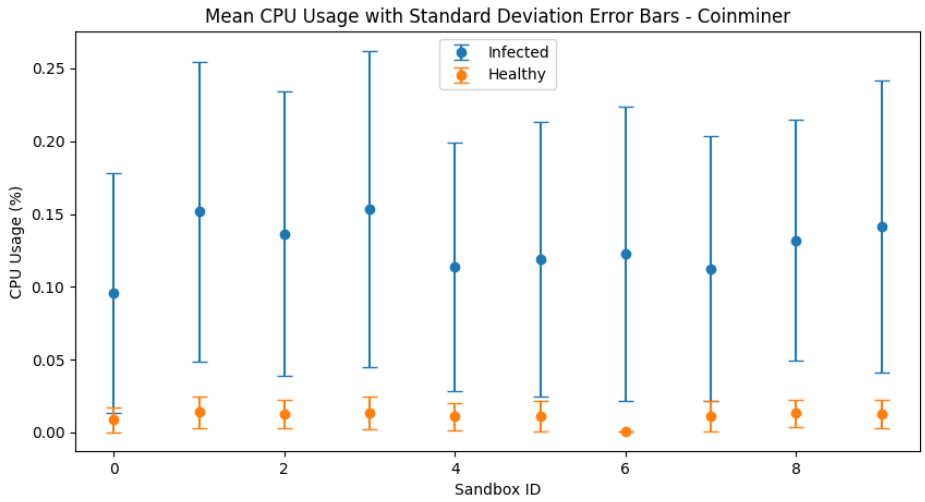

Figure 6.14: CPU Usage Across Multiple Runs With Coinminer Sample

of reproducibility in the system's operation when dealing with this type of malware. One interesting observation is that the infected sandboxes showed a lower CPU and RAM usage.

The Monti ransomware analysis revealed increased CPU and RAM usage in infected instances, and the cosine similarity scores ranged from 0.92 to 1.00, further supporting the claim of high reproducibility. Additionally, the ransomware's operation within the SecBox platform was confirmed as expected, contributing to the higher read/write count observed.

In the Coinminer analysis, consistent increases in CPU and RAM activity were observed,

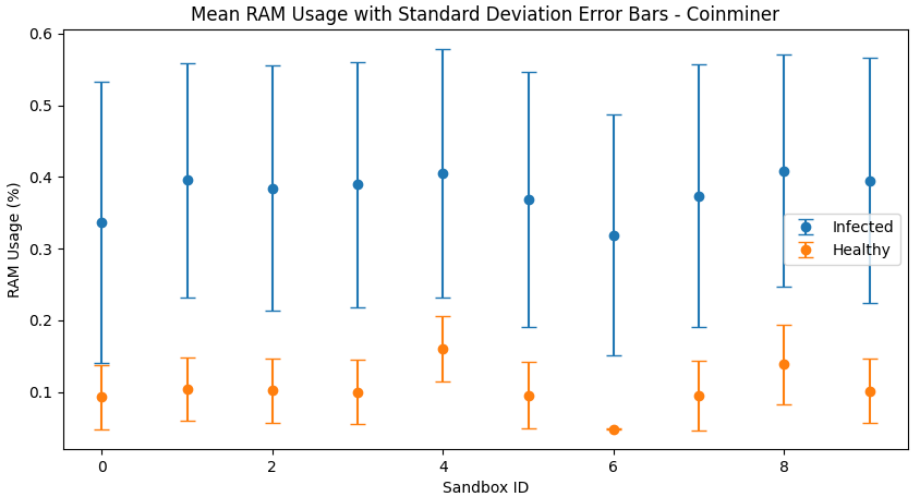

Figure 6.15: RAM Usage Across Multiple Runs With Coinminer Sample

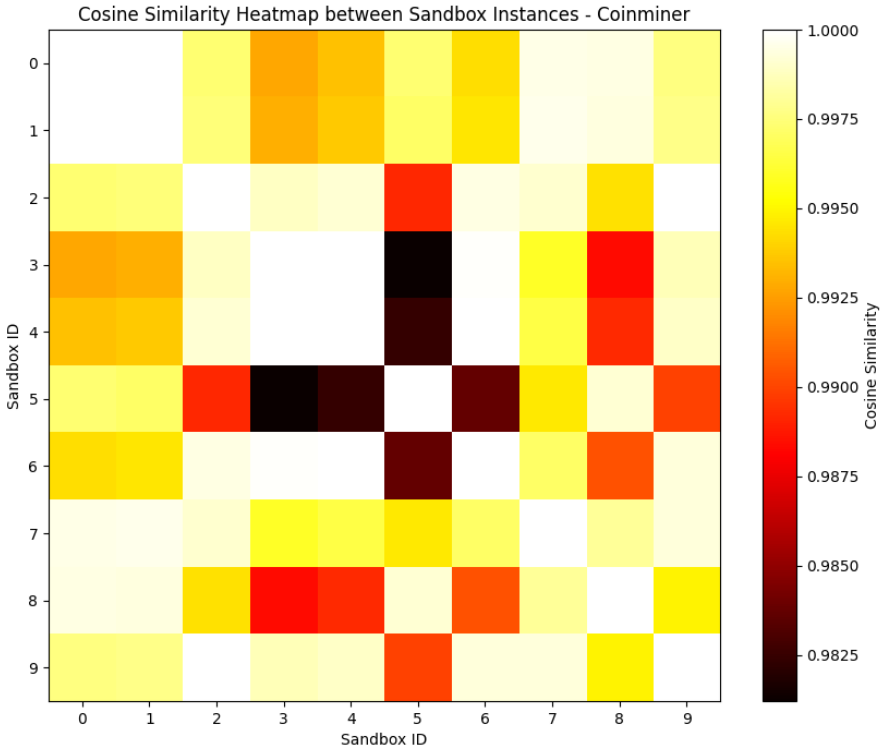

Figure 6.16: Cosine Similarity Across Multiple Runs With Coinminer Sample

and the patterns were also almost identical across different instances, as evidenced by the cosine similarity matrix. Notably, there were issues with the read/write count graph, but the directory graph results provided additional reproducibility evidence.

These results are encouraging and confirm the effectiveness of the implemented features

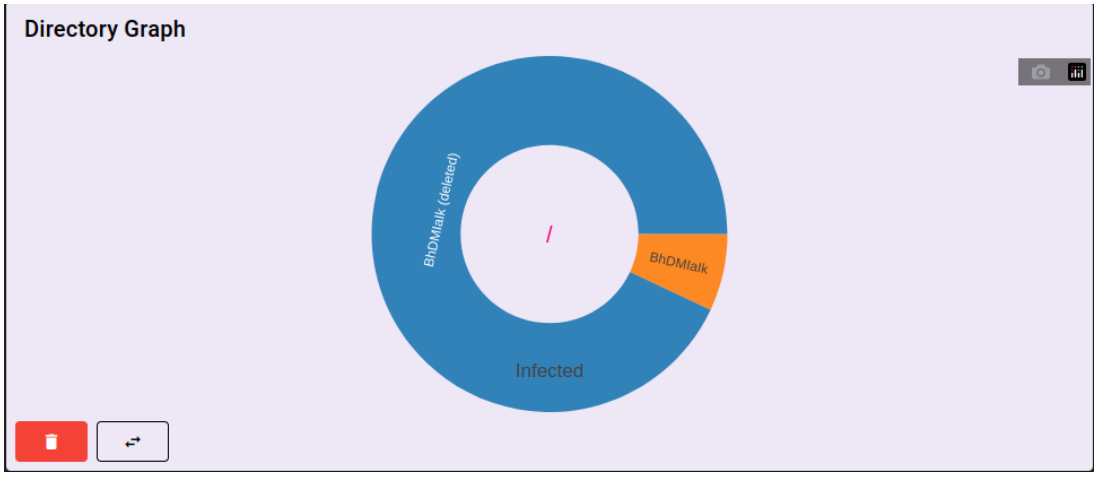

Figure 6.17: Directory Graph Of One Run With Coinminer Sample

in enhancing the reproducibility of malware analysis within the SecBox platform. It is evident that the tool is capable of producing consistent results across multiple instances, a valuable trait for comparative analysis and collaboration within cybersecurity research.

6.5 Limitations

While the experimental results affirm the potential benefits of the command recorder and CSV generator for the reproducibility of malware analysis within the SecBox platform, several limitations within the study must be acknowledged.

Firstly, the hardware and software configurations used in the study were rather specific, potentially reducing the generalizability of the findings. The experimental setup was conducted using a system with Ubuntu 22.04.2 Long Term Support (LTS), AMD Ryzen 9 5900X 12-Core Processor, and 32.0GB RAM, which may not represent the broad range of configurations that the SecBox platform may encounter in real-world usage. Therefore, the reproducibility and performance of the features in other setups remain to be seen.

Secondly, the evaluation involved only three types of actual malware: Mirai, Monti, and Coinminer. Given the diverse and evolving nature of malware, this narrow scope of samples could limit the breadth of the conclusions that can be drawn from the study. It is possible that other types of malware, especially those with more complex or stealthy operation techniques, may produce different results in the reproducibility tests.

Thirdly, the experiments relied heavily on manual operation and observation, which could introduce human errors and inconsistencies into the process. The lack of automation might also limit the scalability of the study, as it may be impractical to manually perform tests with a larger number of malware samples or across a broader range of system configurations.

Lastly, during the Coinminer analysis, the read/write count graph was malfunctioning, a limitation that prevented the full analysis of this particular aspect of system activity. This

malfunction underlines the potential for technical issues that may impact the completeness and reliability of the data collected during the experiments.

Chapter 7

Summary and Conclusions

This chapter serves as a comprehensive summary and conclusion of the thesis, summarizing the primary research objectives, methodologies employed, and key findings.

7.1 Summary

The objective of this work was to design and implement a reproducible and realistic data collection system for dynamic malware analysis by enhancing an existing platform, SecBox [4]. The introduction chapter outlined the need for such an enhancement, given the evolving landscape of cybersecurity threats and the imperative for better analysis tools.

In the background chapter, key principles and concepts relevant to dynamic malware analysis and malware itself were explored, but also what constitutes reproducibility. This provided a theoretical foundation for the enhancements made to the SecBox platform.

The related work chapter highlighted existing methods and tools in dynamic malware analysis, helping to identify gaps and opportunities for enhancement in the SecBox platform. It also served to draw out reproducibility properties.

In the architecture chapter, a detailed overview of the proposed solution's structure was provided. It served to lay out the architectural groundwork for the improvements made in the platform, setting the stage for subsequent discussion.

The implementation chapter then delved into the technicalities of developing these features. The command recorder was designed to record and replicate all commands issued to two identical sandboxes, one of which included the selected malware sample. The CSV generator was implemented to automatically record CPU and RAM usage, providing valuable data for analysis.

In the evaluation chapter, the implemented features were tested to assess their effectiveness. The evaluation was conducted using a specific system setup on four types of malware: A reference malware sample, Mirai, Monti and Coinminer. The results revealed

a high degree of consistency across different sandbox instances for each type of malware, indicating effective reproducibility facilitated by the implemented features.

7.2 Conclusion

The research presented in this thesis highlights the successful design and implementation of enhancements to the SecBox platform, aimed at improving dynamic malware analysis by ensuring reproducibility. The command recorder and CSV generator have proven to be effective tools for this purpose. The ability to replicate all commands issued to a sandbox environment and capture key system parameters like CPU and RAM usage has significant implications for malware analysis. The consistency in results across sandbox instances has validated the potential of the designed system to offer accurate, consistent, and reproducible dynamic malware analysis.

7.3 Implications

The proposed enhancements to SecBox have substantial implications for the field of cybersecurity. By enabling detailed recording of all commands issued in the sandbox environment, the command recorder significantly improves the reproducibility of the malware analysis process, allowing for more accurate comparison and validation of results across different analysis instances.

The CSV generator, on the other hand, facilitates a deeper understanding of malware behavior by automatically monitoring key system parameters such as CPU and RAM usage. Because the recorded data is in CSV format, it offers a streamlined means for analysts to engage in further analysis. The use of CSV format allows for easier data manipulation and statistical analysis, which can provide further insights into the behavior and impacts of various malware types.

These enhancements augment the analytical capabilities of the SecBox platform, and by extension, the capabilities of malware analysts, making this tool more effective in combating evolving cybersecurity threats.

7.4 Limitations and Challenges

The work presented is not without its limitations. The testing was confined to three malware types and a specific system setup. While the results were promising, there's a need to validate the system with a broader variety of malware and diverse system setups.

The implemented recorder relies on manual operation, which is prone to human error and limits scalability. Additionally, the system currently lacks a fail-safe mechanism for dealing with command execution failures. This could potentially lead to significant issues

if later commands rely on the successful execution of prior ones. Designing a mechanism to handle such failures, possibly through the implementation of error detection and rollback functionalities, is crucial to ensure the integrity and consistency of the analysis process.

A significant challenge encountered during this work was the malfunctioning post.analysis process. The platform's in-built capability to generate performance graphs often malfunctions, and the read-write count graphs can become faulty when faced with large and steadily increasing data. This issue currently severely limits the usefulness of the system's data visualization tools, requiring users to find alternative methods for data interpretation and analysis.

Lastly, while the development of the CSV generator improved the ability to record and analyze system resource usage, it currently only supports CPU and RAM usage.

7.5 Future Work

Several areas of potential future work have been identified as a result of this study. These encompass improvements to the current implementation, as well as exploring new features and functionalities.

The first point of future work could involve conducting additional testing across a wider range of malware types and varying system setups. As the system was primarily validated using three types of malware, broader testing would serve to further validate and improve the functionality of the enhancements implemented in the thesis.

Given the current reliance on manual operation, automating the command recording process to a certain extent would be a valuable avenue for future development. By minimizing human intervention, it could improve the system's scalability and reduce the chance of errors. Careful consideration must be given to preserving user flexibility and command control in any automated process.

The issue of command execution failure remains a significant concern. As such, designing a fail-safe mechanism that can handle command execution failures would be a beneficial addition. This might involve the development of error detection and rollback functionalities, or a strategy for rerunning failed commands.

Efforts to correct the malfunctions of the post-analysis process, particularly the system's inherent ability to generate performance graphs, should be a priority. Future work could focus on debugging and refining this feature to ensure its correct operation, particularly with large datasets. Additionally, in-built generation of graphs for comparison between multiple analysis runs could be taken into consideration.

Another potential area of focus is the expansion of the CSV generator's capabilities to record other system resources, such as disk input/output (I/O), network activity, or even graphics processing unit (GPU) usage. This expansion could provide a more comprehensive picture of malware behavior but would necessitate careful handling to avoid inducing significant performance overhead.

Finally, integrating more advanced analytics and visualization tools into the platform could be a beneficial feature, assisting users in interpreting and understanding the collected data in a more intuitive or comprehensive manner. Such tools could offer automated analysis, correlation identification, or trend tracking functionalities, further augmenting the platform's analytical capabilities.

Bibliography

[1] N. Sun, M. Ding, J. Jiang, *et al.*, "Cyber threat intelligence mining for proactive cybersecurity defense: A survey and new perspectives", *IEEE Communications Surveys & Tutorials*, 2023.

[2] L. Caviglione, M. Choraś, I. Corona, *et al.*, "Tight arms race: Overview of current malware threats and trends in their detection", *IEEE Access*, vol. 9, pp. 5371–5396, 2020.

[3] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, "Dynamic malware analysis in the modern era—a state of the art survey", *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–48, 2019.

[4] J. v. d. Assen, A. H. Celdrán, A. Zermin, R. Mogicato, G. Bovet, and B. Stiller, "Secbox: A lightweight container-based sandbox for dynamic malware analysis", in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, 2023, pp. 1–3.

[5] N. Idika and A. P. Mathur, "A survey of malware detection techniques", *Purdue University*, vol. 48, no. 2, pp. 32–46, 2007.

[6] A. P. Namanya, A. Cullen, I. U. Awan, and J. P. Disso, "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.

[7] H. H. Al Hajri, B. M. Al Mughairi, M. I. Hossain, and A. M. Karim, "Crypto jacking a technique to leverage technology to mine crypto currency", *Int. J. Acad. Res. Bus. Social Sci*, vol. 9, no. 3, pp. 1210–1221, 2019.

[8] F. Touchette, "The evolution of malware", *Network Security*, vol. 2016, no. 1, pp. 11–14, 2016.

[9] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools", *ACM computing surveys (CSUR)*, vol. 44, no. 2, pp. 1–42, 2008.

[10] S. N. Goodman, D. Fanelli, and J. P. Ioannidis, "What does research reproducibility mean?", *Science translational medicine*, vol. 8, no. 341, 341ps12–341ps12, 2016.

[11] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic analysis of malicious code", *Journal in Computer Virology*, vol. 2, pp. 67–77, 2006.

[12] A. SIGMOD, *Acm sigmod*, `https://reproducibility.sigmod.org/`, Accessed: 2023-06-13, 2023.

[13] R. D. Peng, "Reproducible research in computational science", *Science*, vol. 334, no. 6060, pp. 1226–1227, 2011.

[14] J. Leipzig, D. Nüst, C. T. Hoyt, K. Ram, and J. Greenberg, "The role of metadata in reproducible computational research", *Patterns*, vol. 2, no. 9, p. 100 322, 2021.

[15] W. Elmenreich, P. Moll, S. Theuermann, and M. Lux, "Making computer science results reproducible-a case study using gradle and docker", *PeerJ Preprints*, vol. 6, e27082v1, 2018.

[16] T. Mandl, U. Bayer, and F. Nentwich, "Anubis analyzing unknown binaries the automatic way", in *Virus bulletin conference*, vol. 1, 2009, p. 02.

[17] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox", *IEEE Security & Privacy*, vol. 5, no. 2, pp. 32–39, 2007.

[18] V. S. Group, *Threat analyzer*, `https://vipre.com/products/threat/threat-analyzer/`, Accessed: 2023-05-11, 2023.

[19] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: Malware analysis via hardware virtualization extensions", in *Proceedings of the 15th ACM conference on Computer and communications security*, 2008, pp. 51–62.

[20] *Intel virtualization technology*, Accessed: 2023-05-08, 2023. [Online]. Available: `https://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html`.

[21] T. K. Lengyel, S. Maresca, B. D. Payne, G. D. Webster, S. Vogl, and A. Kiayias, "Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system", in *Proceedings of the 30th annual computer security applications conference*, 2014, pp. 386–395.

[22] T. K. Lengyel, *Drakvuf*, Accessed: 2023-05-08, 2023. [Online]. Available: `https://github.com/tklengyel/drakvuf/`.

[23] C. S. Team, *Cuckoo sandbox*, Accessed: 2023-05-08, 2019. [Online]. Available: `https://cuckoosandbox.org/`.

[24] *Cyber threat intelligence*, Accessed: 2023-07-28, 2023. [Online]. Available: `https://socradar.io/suites/cyber-threat-intelligence/`.

[25] *Any.run features*, Accessed: 2023-07-28, 2023. [Online]. Available: `https://any.run/features/`.

[26] *Joe sandbox*, Accessed. 2023-07-28, 2023. [Online]. Available: `https://joesecurity.org/joe-sandbox-desktop`.

[27] *Malwarebazaar*, Accessed: 2023-08-13, 2023. [Online]. Available: `https://bazaar.abuse.ch/`.

[28] Q.-D. Ngo, H.-T. Nguyen, V.-H. Le, and D.-H. Nguyen, "A survey of iot malware and detection methods based on static features", *ICT Express*, vol. 6, no. 4, pp. 280–286, 2020.

[29] *Gvisor*, Accessed: 2023-08-06, 2023. [Online]. Available: `https://gvisor.dev/docs/`.

[30] *Digital corpora*, Accessed: 2023-08-07, 2023. [Online]. Available: `https://downloads.digitalcorpora.org/corpora/files/govdocs1/zipfiles/`.

[31] W. McKinney *et al.*, "Data structures for statistical computing in python", in *Proceedings of the 9th Python in Science Conference*, Austin, TX, vol. 445, 2010, pp. 51–56.

[32] J. D. Hunter, "Matplotlib: A 2d graphics environment", *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[33] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with NumPy", *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

[34] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[35] M. Antonakakis, T. April, M. Bailey, *et al.*, "Understanding the mirai botnet", in *26th USENIX security symposium (USENIX Security 17)*, 2017, pp. 1093–1110.

[36] *Enhanced secbox github repository*, Accessed: 2023-08-07, 2023. [Online]. Available: https://github.com/vichhay97/SecBox.

[37] *Secbox github repository*, Accessed: 2023-08-07, 2023. [Online]. Available: https://github.com/SaltySpatula/SecBox.

[38] *Nodejs 16 installation on ubuntu 20.04*, Accessed: 2023-08-07, 2023. [Online]. Available: https://www.stewright.me/2022/01/tutorial-install-nodejs-16-on-ubuntu-20-04/.

[39] *Yarn installation*, Accessed: 2023-08-12, 2023. [Online]. Available: https://www.python.org/downloads/release/python-390/.

[40] *Python 3.9*, Accessed: 2023-08-12, 2023. [Online]. Available: https://www.python.org/downloads/release/python-390/.

[41] *Pip installation*, Accessed: 2023-08-12, 2023. [Online]. Available: https://pip.pypa.io/en/stable/installation/.

Abbreviations

RAM	Random Access Memory
CPU	Central Processing Unit
OS	Operating System
API	Application Programming Interface
ASEPs	Autostart extensibility Points
JSON	JavaScript Object Notation
CSV	Comma-Separated Values
SDK	Software Development Kit
IoT	Internet of Things
ARM	Advanced RISC Machine
SH	Shell
LTS	Long Term Support
I/O	Input/ Output
GPU	Graphics Processing Unit

List of Figures

List of Tables

Appendix A

Installation Guidelines

This chapter provides instructions for setting up an instance of the enhanced SecBox [4]. For a comprehensive understanding of SecBox and its enhancements over the original platform, refer to [36] and [37] respectively.

A.1 Frontend

The frontend requires Node 16.X [38] and Yarn [39]. To install the dependencies, navigate to **SecBox/app** and run **npm install**.

A *.env* file called **app.env** is already supplied and is set up for local deployment. To deploy the frontend locally, run **npm run serve**. For alternative setups **app.env** must be altered with the desired backend IP.

A.2 Backend

The backend requires Python 3.9 [40] and pip [41]. All dependencies required for the backend can be installed in **Secbox/api** with **pip install -r requirements.txt**.

A *.env* file for local deployment called **api.env** is supplied as well and can be altered with an alternative MongoDB address.

The backend can be started with the command

```
python3 webapp_api.py
```

A.3 Host

The host's dependencies can be installed by navigating to **SecBox/host** and then running the command **pip install -r requirements.txt**.

To set up the host on a machine, run:

```
sudo ./setup.sh
sudo ./setup_bazel_gvisor.sh
```

A *.env* file for local deployment called **host.env** is supplied. It must be altered correspondingly, should the backend IP not be local.

The host can then be run with

```
sudo -E python3 host.py
```