



UNIVERSITY OF PADOVA

DEPARTMENT OF MATHEMATICS "TULLIO LEVI-CIVITA"

MASTER THESIS IN CYBERSECURITY

COMPARATIVE ANALYSIS OF MALWARE BEHAVIOR IN HARDWARE AND VIRTUAL SANDBOX

SUPERVISOR

PROF. ALESSANDRO BRIGHENTE
UNIVERSITY OF PADOVA

MASTER CANDIDATE

MUHAMMAD ASAD JAHANGIR JAFFAR

CO-SUPERVISOR

PROF. SEBASTIAN BIEDERMANN
TECHNISCHE HOCHSCHULE WÜRZBURG-SCHWEINFURT

STUDENT ID

2005940

ACADEMIC YEAR

2022-2023

“DEDICATION OR QUOTE”

”NOT ALL THOSE WHO WANDER ARE LOST.” J. R. R. TOLKEIN

Abstract

Malicious software, or malware, continues to be a pervasive threat to computer systems and networks worldwide. As malware constantly evolves and becomes more sophisticated, it is crucial to develop effective methods for its detection and analysis. Sandboxing technology has emerged as a valuable tool in the field of cybersecurity, allowing researchers to safely execute and observe malware behavior in controlled environments. In this context, a hardware sandbox for malware analysis, defined as an isolated environment with dedicated hardware and a separate operating system, is employed to safely execute and study potentially malicious software. This controlled space ensures the containment of threats through features such as network isolation, snapshot capabilities, and resource monitoring. Analysts utilize the hardware sandbox to observe and understand malware behavior, generate detailed reports, and develop effective countermeasures, safeguarding the integrity of the host system. This thesis presents a comprehensive investigation of the behavior of malware samples when executed in both hardware and virtual sandboxes. The primary objective is to assess the effectiveness of hardware sandboxing in capturing and analyzing malware behaviors compared to traditional virtual sandboxes. The research methodology involves the execution of various malware samples in both hardware and virtual sandboxes, followed by the analysis of key parameters, including memory changes, file system logs, and network traffic. By comparing the results obtained from the two sandboxing approaches, this study aims to provide insights into the advantages and limitations of each method. Furthermore, the research delves into the potential evasion techniques employed by malware to bypass detection in either sandboxing environment. Identifying such evasion strategies is vital for enhancing the overall security posture and developing more robust defense mechanisms against evolving malware threats. The findings of this research contribute to the field of cybersecurity by shedding light on the strengths and weaknesses of hardware and virtual sandboxes for malware analysis. Ultimately, this work serves as a valuable resource for security practitioners and researchers seeking to improve malware detection and analysis techniques in the ever-evolving landscape of cybersecurity threats.

Contents

| | |
|---|-----------|
| ABSTRACT | v |
| LIST OF FIGURES | ix |
| LIST OF TABLES | xi |
| LISTING OF ACRONYMS | xiii |
| 1 INTRODUCTION | 1 |
| 1.1 Introduction to Sandbox Technology | 1 |
| 1.2 Background And Research Problem | 2 |
| 1.3 Novel solution | 3 |
| 1.4 Organization of the Thesis | 3 |
| 2 LITERATURE REVIEW | 5 |
| 2.1 Virtual Sandbox | 5 |
| 2.1.1 Virtual Sandbox Issues | 6 |
| 2.1.2 Sandbox detection | 7 |
| 2.2 Dedicated Hardware for Guest Systems | 9 |
| 2.2.1 BareBox | 10 |
| 2.2.2 Hardware Supported Virtual Machines | 11 |
| 2.3 System Virtual Machines | 11 |
| 3 SYSTEM AND THREAT MODELS | 13 |
| 3.1 Understanding Modern Malware Threats | 13 |
| 3.1.1 Malware executable code | 15 |
| 3.1.2 Characteristics of Modern Malware | 16 |
| 3.1.3 Malware Delivery | 18 |
| 3.2 Virtual Sandbox Threats and Vulnerabilities | 21 |
| 3.3 Threat Models | 23 |
| 4 METHODOLOGY | 25 |
| 4.1 Hardware Components | 25 |
| 4.1.1 LeetDMA | 26 |
| 4.1.2 KVM SWITCH | 29 |
| 4.1.3 network-attached storage (NAS) | 30 |

| | | |
|-------|--|----|
| 4.2 | Operating System Reset | 31 |
| 4.2.1 | Data Encryption with RollBack Rx | 32 |
| 4.2.2 | DATA STORAGE | 37 |
| 4.3 | Hardware Sandbox | 40 |
| 5 | DATABASE AND RESULTS | 43 |
| 5.1 | Malware Database | 43 |
| 5.1.1 | Malware's SHA-256 | 45 |
| 5.2 | Analysis of Memory Dumps | 46 |
| 5.2.1 | Plist and Psscan | 46 |
| 5.2.2 | Filescan | 49 |
| 5.2.3 | malfind | 53 |
| 5.2.4 | Dlllist | 54 |
| 5.2.5 | Driverirp | 56 |
| 5.2.6 | Windows Privileges | 59 |
| 5.2.7 | Cmdline | 62 |
| 5.2.8 | handles | 64 |
| 5.3 | network traffic | 66 |
| 5.3.1 | Malware's traffic database | 66 |
| 5.3.2 | Malware Traffic Analysis | 67 |
| 5.4 | Log Files | 76 |
| 5.4.1 | Log's of Hardware sandbox | 76 |
| 5.4.2 | LOGS OF VIRTUAL SANDBOX | 81 |
| 6 | CONCLUSION | 83 |
| 6.1 | Limitation | 84 |
| | REFERENCES | 85 |
| | ACKNOWLEDGMENTS | 91 |

Listing of figures

| | | |
|------|--|----|
| 1.1 | Sandbox working principle | 2 |
| 3.1 | The Evolution Of Cyber Threats [1]. | 14 |
| 3.2 | Hidden Code Extraction [2]. | 16 |
| 3.3 | Active vs Passive Attack [3]. | 17 |
| 3.4 | Shortcut execution leading to host infection [4]. | 19 |
| 3.5 | Shortcut-triggered HTML file download [4]. | 20 |
| 3.6 | Base64-encoded content within the shortcut [4]. | 21 |
| 4.1 | LeetDMA Device [5]. | 26 |
| 4.2 | Artix-7 35T Block Diagram [6]. | 29 |
| 4.3 | Dual KVM Switch. | 30 |
| 4.4 | Network Attached Storage [7]. | 31 |
| 4.5 | File system in the absence of RollBack Rx | 33 |
| 4.6 | File system using RollBack Rx | 34 |
| 4.7 | RollBack Rx encryption. | 36 |
| 4.8 | Different Types of Storage [8]. | 37 |
| 4.9 | Hardware Sandbox. | 40 |
| 4.10 | Working of Sandbox. | 42 |
| 5.1 | Malware samples shared in last 30 days [9]. | 44 |
| 5.2 | Malware families and their tags | 45 |
| 5.3 | Psscan and Pslist before execution. | 47 |
| 5.4 | Hardware sandbox changes after execution. | 47 |
| 5.5 | Virtual sandbox chnages after execution. | 48 |
| 5.6 | Hb and Hd comparison. | 51 |
| 5.7 | Va files section of the algorithm. | 52 |
| 5.8 | Code injection by the malware sample. | 53 |
| 5.9 | Hardware offset difference before and after malware execution. | 56 |
| 5.10 | Virtual offset difference before and after malware execution. | 58 |
| 5.11 | System privileges granted to the malware. | 59 |
| 5.12 | hardware and virtual sandbox cmdline. | 62 |
| 5.13 | Malware interaction with critical system process. | 64 |
| 5.14 | Malware interaction for virtual. | 65 |
| 5.15 | Ip's handshakes | 67 |

| | | |
|------|--|----|
| 5.16 | Suspicious Handshake. | 67 |
| 5.17 | Infamous Chisel graph. | 68 |
| 5.18 | 192.168.0.2 refer and communicating files. | 69 |
| 5.19 | One of the nodes from the communicating files. | 70 |
| 5.20 | One of the nodes from the refer files. | 70 |
| 5.21 | Ip 40.125.122.151 graph. | 71 |
| 5.22 | Nodes of refer and communicating files. | 72 |
| 5.23 | Handshake in virtual sandbox. | 73 |
| 5.24 | Ip 199.201.110.204 communicating file node. | 74 |
| 5.25 | Communicating file node details. | 75 |
| 5.26 | Malware sample process log's. | 76 |
| 5.27 | Process threads | 77 |
| 5.28 | Consistent Malicious Operations Across Different Environments. | 82 |

Listing of tables

| | | |
|-----|---|----|
| 2.1 | Malware's techniques for detecting virtual environments. | 8 |
| 3.1 | Modern Malware Approach's. | 23 |
| 4.1 | Artix-7 Features [10]. | 28 |
| 5.1 | Results from the Algorithm. | 50 |
| 5.2 | Hardware and Virtual sandbox configurations with the network. | 67 |

Listing of acronyms

| | |
|----------------------|--|
| DOS | Disk Operating System |
| FPGAs | Field Programmable Gate Arrays |
| TAN | Transaction Authentication Number |
| JPCERT/CC ... | Japan Computer Emergency Response Team Coordination Center |
| CC | Command and Control server |
| CAB | Cabinet |
| DLLs | Dynamic Link Library |
| HDD | Hard Disk Drive |
| TTPs | Tactics, Techniques and Procedures |
| DMA | Direct Memory Access |
| PCI | Peripheral Component Interconnect |
| ASICs | Application Specific Integrated Cir- cuits |
| KVM | Keyboard, Video and Mouse |
| NAS | Network-Attached Storage |
| DAS | Direct-Attached Storage |
| AES | Advanced Encryption Standard |
| DES | Data Encryption Standard |
| RAID | Redundant Array of Independent Disks |
| CPUs | Central Processing Units |
| Vad | Virtual Address Descriptor |
| Hb | Before malware execution in hardware sandbox |

| | |
|--------------------|---|
| Vb | Before malware execution in virtual sandbox |
| Ha | After malware execution in hardware sandbox |
| Va | After malware execution in virtual sandbox |
| Hd | Differences between Hb and Ha |
| Vd | Difference between Vb and Va |
| VMM | Virtual Machine Monitor |
| IDTR | Interrupt Descriptor Table Register |
| LDTR | Local Descriptor Table Register |
| VMI | Virtual Machine Introspection |
| NIC | Network Interface Card |
| IP | Internet Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| ICMP | Internet Control Message Protocol |

1

Introduction

1.1 INTRODUCTION TO SANDBOX TECHNOLOGY

Cybersecurity plays a crucial role in safeguarding systems against cyber threats and hostile attacks, serving as a protective shield for countries worldwide. A key component of cybersecurity is the use of sandboxes, which are isolated environments simulating end-user operating conditions. These sandboxes allow the secure execution of suspicious programs, minimizing the risk to the host device and network. By remaining external to the system, sandboxes prevent system failures and curb the spread of software vulnerabilities between systems.

In the era of information exchange via the internet, ensuring data security is paramount. With more than 65-70 percent of transactions occurring online, cybersecurity becomes increasingly vital in today's environment. The advent of emerging technologies emphasizes the importance of secure protocols in contemporary settings. Cybersecurity applications extend across various domains, including cloud and mobile computing, net banking, e-commerce, and more [11].

Addressing the pervasive threat of cybercrime, which adversely impacts the economy, society, and personal information, requires a comprehensive and secure approach to digital transactions. A sandbox serves as a testing environment, allowing users to run programs or execute

files without affecting the application, system, or platform they are operating on. Widely used by software engineers and cybersecurity specialists, sandboxes are instrumental in preventing unauthorized access and protecting system resources, contributing significantly to organizational security. This study aims to evaluate the utilization of sandboxes in the realm of cybersecurity, exploring their benefits and advantages in enhancing digital security measures.

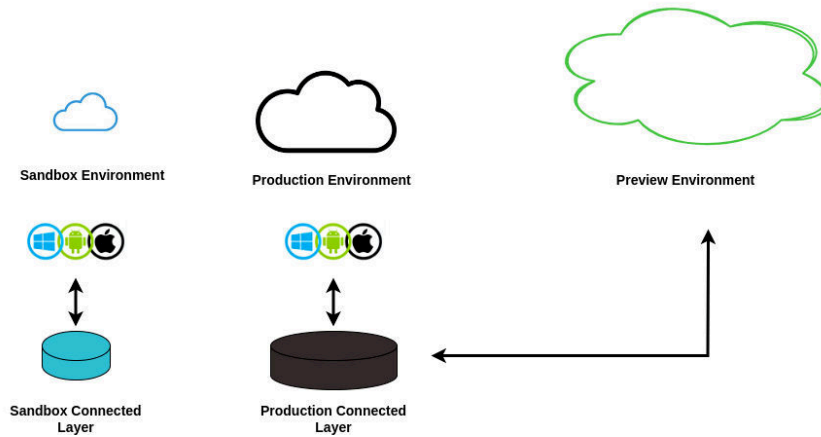


Figure 1.1: Sandbox working principle

As in the figure 1.1, We have a sandbox environment where we test new applications or files before sending them to production and preview environments. A sandbox is a controlled and isolated space that mimics the production environment without posing any risk to critical systems. In this secure environment, we can assess the behavior and performance of applications or files, identify potential security vulnerabilities, and ensure they meet our standards before deploying them to live production or preview systems. This practice adds an extra layer of security by allowing us to catch issues early, minimizing the impact on operational environments and enhancing the overall reliability of our systems.

1.2 BACKGROUND AND RESEARCH PROBLEM

Cybersecurity is an ever-evolving discipline, continually facing evolving threats. The objective of this thesis is to enhance malware analysis, recognizing the aging nature of current virtualization-based solutions in the context of rapidly evolving malware. A perpetual competition exists between security analysts and malicious actors, necessitating constant innovation to detect novel penetration techniques. While virtual sandboxes prove useful in identifying suspicious activities, our thesis advocates for the utilization of dedicated hardware for this purpose. Building

upon past research, our work contributes to the exploration of more robust and effective strategies in cybersecurity.

1.3 NOVEL SOLUTION

The proposed solution in this thesis for addressing modern malware involves the utilization of a dedicated hardware equipped with its distinct operating system. This approach aims to reveal the authentic behavior of malware, allowing it to manifest in its intended manner. By running malware in this specialized environment, security analysts can glean deeper insights into its functionalities and characteristics. While this concept is not entirely novel in the research industry, it serves as an additional perspective to existing work in the field. The fundamental idea is to provide an alternative means to enhance malware analysis, leveraging dedicated hardware resources for a more accurate depiction of malware behavior, ultimately contributing to advancements in cybersecurity practices.

1.4 ORGANIZATION OF THE THESIS

In this chapter, a concise overview of sandboxes and their background has been provided, aiming to familiarize the reader with the thesis's objectives. Subsequent chapters will delve into the following topics:

- In Chapter 2, the literature review we undertake a comprehensive exploration of the research pertinent to the thesis. The central focus of our discussion revolves around the imperative need for dedicated hardware in the context of sandboxes and the inherent vulnerabilities that virtual sandboxes face in the realm of modern malware. Despite the relatively limited body of work dedicated to hardware-based sandboxes, we aspire to intricately examine and expound upon the conceptual framework and significance underlying this approach.
- In Chapter 3, we explore system and thread models, focusing on the challenges posed by modern malware. We delve into the tactics employed by malware to detect virtualization, its techniques for self-unpacking, and the evolutionary trends observed in malware behavior over the years.
- In Chapter 4, our methodology focuses on the development of a hardware-based sandbox, aiming to authentically replicate real-world conditions for a more accurate analysis of malware behavior. Unlike traditional virtual sandboxes, our approach involves using

an actual system and carefully selected hardware components to mirror the characteristics of a genuine operating environment. This strategy allows us to closely emulate the conditions in which malware operates, enhancing our ability to understand and detect evolving threats. Through this innovative approach, we contribute to advancing cybersecurity practices and strengthening defense mechanisms.

- In Chapter 5, we delve into the database and results section, presenting the experimental data and outcomes. The analysis includes a comprehensive comparison between our hardware-based sandbox and a virtual sandbox. We discuss the insights gained from the experiments and highlight the differences observed in performance and effectiveness. This chapter provides a detailed examination of the obtained results, shedding light on the strengths and limitations of our approach in contrast to conventional virtual sandbox solutions.
- In Chapter 6, the limitations of the work are discussed followed by providing future research directions and concluding the thesis.

2

Literature Review

In this chapter, we undertake a comprehensive exploration of the research pertinent to the thesis. The central focus of our discussion revolves around the imperative need for dedicated hardware in the context of sandboxes and the inherent vulnerabilities that virtual sandboxes face in the realm of modern malware. Despite the relatively limited body of work dedicated to hardware-based sandboxes, we aspire to intricately examine and expound upon the conceptual framework and significance underlying this approach.

2.1 VIRTUAL SANDBOX

Sandboxes play a crucial role in environments where the use of software components is necessary but current verification or trustworthiness is lacking. These isolated components are often perceived as potential sources of malicious activity or vulnerability. Prominent examples of sandboxed components include browser engines like Google Chrome and Internet Explorer, productivity software such as Microsoft Word and Adobe Reader, and even operating system kernels like Windows 8. Virtual machines, which operate software on simulated hardware independently from the host system, are commonly employed in malware analysis, containing potentially harmful computations. Mobile ecosystems, like Android, also utilize sandboxes to restrict the impact of malicious applications on devices. In the complex landscape of software systems, sandboxes offer a form of salvation by isolating and containing potentially problematic components. Rather than attempting to thoroughly scrutinize intricate software

systems, sandboxes provide a more manageable alternative. Common characteristics of mainstream sandboxes include reliance on easily composed coarse-grained operating system features, transparency to users of sandboxed components, and limited utilization of extensive research in software security sandboxing. Over the past few decades, researchers have dedicated efforts to constructing diverse sandboxes capable of containing a wide array of computations, ranging from full-fledged desktop applications to various types of applications, such as third-party libraries in Java programs or ads on websites. These sandboxes serve multifaceted purposes, from preventing memory corruption exploits to enforcing control and data-flow integrity, and introducing diversity to counteract existing monocultures [12].

2.1.1 VIRTUAL SANDBOX ISSUES

In the realm of cybersecurity, the escalating sophistication of malware, ranging from viruses and worms to bots, necessitates a proactive approach involving vigilant observation, in-depth analysis of their mechanisms, and the swift identification of potential issues. Isolated sandboxes have proven to be invaluable environments for conducting such endeavors, offering inherent resilience against external attacks and infections. Technological advancements, particularly in operating system (OS) and hardware virtualizations, have streamlined the creation of isolated sandboxes. Given the disruptive nature of malware, which often requires frequent rebuilding of analyzing environments, the widespread adoption of virtualization technologies becomes evident. The ease of rebuilding environments introduced to virtualization technology enhances the efficiency of the malware analysis process [13]. Shifting focus to the mobile landscape, Android has solidified its position as a dominant operating system for mobile platforms, boasting significant market share. Its security measures, including application sandboxing and a permission framework, aim to regulate system resource access and manage communication between applications. However, recent incidents have exposed vulnerabilities in Android's security framework, leading to privilege escalation attacks such as confused deputy and collusion attacks. Confused deputy attacks involve a malicious application exploiting the vulnerabilities of a privileged but confused application. On the other hand, collusion attacks see malicious applications cooperating to combine their permissions, enabling actions beyond their individual privileges. Despite efforts to address these issues through security extensions like Kirin, TaintDroid, Saint, QUIRE, and IPC Inspection, none fully tackle both confused deputy and collusion attacks. Many solutions exhibit shortcomings, such as incompatibility with legacy applications or inefficiency. This underscores the need for a comprehensive, system-centric

solution to effectively address Android’s sandboxing issues [14].

2.1.2 SANDBOX DETECTION

Virtualization/Sandbox Evasion is a tactic employed by adversaries in their defense evasion strategy to identify and steer clear of virtualization and analysis environments, like those found in malware analysis sandboxes. When the malware recognizes the presence of a virtual machine or sandbox environment, it either disconnects from the victim or refrains from executing malicious actions, such as downloading additional payloads [15]. Adversaries employ various techniques for Virtualization/Sandbox Evasion, such as examining the presence of security monitoring tools like Sysinternals and Wireshark. They may also inspect system artifacts linked to analysis or virtualization. Additionally, adversaries might observe genuine user activity to assess whether the system is in an analysis environment. Some employ sleep timers or loops in the malware code to evade detection when operating within a temporary sandbox [16]. Table 2.1 presents different malware types and their respective techniques for detecting virtual environments.

Cloud computing is emerging as a prominent computing model, with virtualization playing a crucial role in its implementation. Virtualization involves the creation and operation of multiple virtual machines (VMs) or guest operating systems on a single physical machine, facilitated by a Virtual Machine Monitor (VMM) or Hypervisor. The Hypervisor abstracts physical machine resources, including CPU, memory, I/O, and NIC, among various virtual machines. This resource sharing presents security challenges for cloud service providers, as it amplifies the risks associated with unknown malware and sophisticated rootkits that may compromise critical kernel data structures. Traditional in-host anti-malware solutions prove insufficient in ensuring the security of guest operating systems, particularly within a virtualized environment [17]. VMI, or Virtual Machine Introspection, has the capability to collect the status details of active VMs while operating within the Virtual Machine Monitor. It can extract valuable information, including the process list and kernel driver modules, by examining the visible raw data in the live virtual machine’s memory. This process is referred to as bridging the semantic gap [18].

| ID | Name | Description |
|--------|-------------|---|
| So331 | Agent Tesla | Agent Tesla has the ability to perform anti-sandboxing and anti-virtualization checks [19]. |
| So534 | Bazar | Bazar can attempt to overload sandbox analysis by sending 1550 calls to printf [20]. |
| So268 | Bisonal | Bisonal can check to determine if the compromised system is running on VMware [21]. |
| So1070 | Black Basta | Black Basta can make a random number of calls to the kernel32.beep function to hinder log analysis. |
| So1039 | Bumblebee | Bumblebee has the ability to perform anti-virtualization checks [22]. |
| So484 | Carberp | Carberp has removed various hooks before installing the trojan or bootkit to evade sandbox analysis or other analysis software. |
| So023 | CHOPSTICK | CHOPSTICK includes runtime checks to identify an analysis environment and prevent execution on it [23]. |
| Go012 | Darkhotel | Darkhotel malware has employed just-in-time decryption of strings to evade sandbox detection [24]. |
| So554 | Egregor | Egregor has used multiple anti-analysis and anti-sandbox techniques to prevent automated analysis by sandboxes. |
| So666 | Gesemium | Gesemium can use junk code to generate random activity to obscure malware behavior [25]. |
| So499 | Hancitor | Hancitor has used a macro to check that an ActiveDocument shape object in the lure message is present. If this object is not found, the macro will exit without downloading additional payloads [26]. |
| So1020 | Kevin | Kevin can sleep for a time interval between C2 communication attempts [27]. |
| So455 | Metamorfo | Metamorfo has embedded a "vmdetect.exe" executable to identify virtual machines at the beginning of execution [28]. |
| So147 | Pteranodon | Pteranodon has the ability to use anti-detection functions to identify sandbox environments [29]. |

Table 2.1: Malware's techniques for detecting virtual environments.

2.2 DEDICATED HARDWARE FOR GUEST SYSTEMS

The widespread use of virtualized setups is common in malware analysis. Virtual environments offer crucial features for this purpose, such as the ability to isolate and swiftly restore the system to a known state after analysis. In the first quarter of 2011 alone, McAfee Labs identified over six million unique malware samples. The continuous surge in new malware highlights the need for high-performance analysis frameworks capable of examining numerous malware samples within a specified timeframe. Virtualization-based solutions have been a logical choice. Unfortunately, a new type of malware, known as VM-aware malware, has emerged. This malware can detect virtualized or emulated environments and adapt its behavior to evade analysis.

The methods employed by malware to detect virtual machines and emulated environments have been extensively documented in the table 2.1. These techniques take advantage of certain artifacts, either software or hardware-related, introduced by the virtualization or emulation layer situated between the operating system in operation and the underlying hardware. Some of these detection techniques focus on recognizing specific configurations or I/O ports within the guest system, while others rely on imprecise system emulation or advanced time-based detection strategies [30]. For instance, a system running within a VMware guest OS can examine the names of available virtual devices, access the magic I/O port (0x5658, 'VX'), or read the values of LDTR or IDTR registers, which differ from the known values on a bare-metal system [31]. Notably, all these detection techniques can be executed in user mode.

In simpler terms, malware has ways of figuring out if it's running in a virtual machine or an emulated environment. This is done by looking at specific things, like the names of virtual devices or certain I/O ports, which are like communication channels between different parts of the system. The malware can also check certain registers to see if they match what's expected in a normal, non-virtual environment. These techniques are like tricks that the malware uses to detect where it's running, and they can all be done without needing special permissions. It is crucial to establish a distinct hardware environment for the host system to function as a sandbox for analyzing modern malware. Two techniques for achieving dedicated hardware for the host systems, which were proposed in the past are as follow:

1. BareBox.
2. Hardware supported virtual machine.

2.2.1 BAREBOX

BareBox, which introduces a methodology of an innovative technique for system restoration without the need for reboots. BareBox executes and monitors malware in its native environment. Upon completing an analysis run, the running operating system (OS) is promptly restored to its previously-saved clean state on-the-fly, taking only a few seconds. This restoration process not only retrieves the underlying disk state but also the volatile state of the entire operating system, including running processes, memory cache, and filesystem cache. Our restoration system comprises an overlay-based volatile mirror disk, a symmetric physical memory partition, and a custom-written operating system named Meta-OS [32].

Meta-OS facilitates the restoration of the entire physical memory of the running operating system once a malware analysis run concludes. Due to circular dependencies between physical memory and CPU context, performing a complete OS restore from within the OS being restored is practically impossible. To address this challenge, Meta-OS provides out-of-OS execution control. The operation of other devices and peripherals is closely dependent on the state of the main physical memory, requiring careful handling of device states for a safe OS restoration. BareBox have few challenges, which are as follows [32]:

- **Virtualization:** While the bare-metal analysis system aims to emulate a realistic environment, executing unknown malicious code on actual hardware entails inherent risks, including potential system-level corruption, particularly through malicious BIOS updates. Achieving complete isolation of BIOS and peripheral devices, like the hard disk, is more challenging compared to virtualized systems.
- **Stealthiness:** Bare-metal analysis systems provide superior stealthiness against VM-aware malware compared to VM/emulation-based systems. However, achieving complete transparency in analysis is theoretically impossible if the malware operates at the same privilege level as the analyzer, requiring practical approaches like stealth techniques or a black box method with some trade-offs in contextual information.
- **System Restore:** In the absence of virtualization, there are restricted choices for swift and effective system restoration. Many existing solutions primarily focus on restoring the persistent state of the operating system by recovering the hard disk, necessitating a system reboot for the restoration process to conclude. However, this reboot is predominantly constrained by I/O operations, resulting in extended downtimes between consecutive analysis runs. Additionally, restoring BIOS and other device firmwares poses another set of challenges.

2.2.2 HARDWARE SUPPORTED VIRTUAL MACHINES

The architecture of a hardware-supported virtualization/sandboxing system extends a conventional computer architecture with a virtualization facility. This hardware sandbox is created by deploying multiple computers within the virtualization facility, each running a guest operating system or a natively executed application. To address the dynamic demands for varying sandbox configurations, including processor cores, memory, and devices, the authors advocate the use of reconfigurable logic, specifically FPGAs. Two paradigms are defined: the instantiation paradigm, utilizing reconfigurable logic to create entire computers; and the virtualization/sandboxing paradigm, where the host operating system manages all dedicated resources, forming a virtualization/sandboxing system. The discussion delves into architectural implications, focusing on the processor, memory, and devices [33].

- **Number of Guests:** The quantity of guest systems is typically unrestricted. Restrictions come from the available space in the reconfigurable logic and the space needed to instantiate a guest machine.
- **Processor Issues:** A processor serves as the core processing component of a computer. Implementing the guest processor in reconfigurable logic enables distinct central processing elements for the host and guest operating systems and applications. This separation prevents conflicts and establishes a physical barrier between them, a feature not attainable in a typical virtual machine or sandboxing system.
- **Memory Issues:** Regarding the memory requirements for guest machines, several inquiries and challenges emerge [34]:
 1. What is the required memory capacity for the guest system?
 2. Can the reconfigurable area supply the necessary amount of memory?
 3. How is the enforcement of the main idea of virtualization/sandboxing paradigm (where all resources are managed by the host operating system) applied to the guest's memory?

2.3 SYSTEM VIRTUAL MACHINES

A system VM environment can concurrently accommodate multiple system images, each operating its own operating system and corresponding applications. Each operating system over-

sees a set of virtualized hardware resources, encompassing processors, storage, and peripheral devices essential for input and output (I/O) operations. In a system VM environment, the host platform's actual resources are shared among guest system VMs, facilitated by a software layer called the virtual machine monitor (VMM). The VMM oversees the allocation and access to the hardware resources of the host platform, providing the illusion to guest operating systems that they own and allocate resources. These virtual resources may or may not correspond to physical resources, and the VMM decides how access is granted to virtual machines. System virtual machines find various applications, both historically and in current and future scenarios [35].

System virtual machines serve various purposes, with applications ranging from historical contexts to current and anticipated future needs. Some specific examples of these versatile applications are discussed below:

- **Multiple single-application virtual machines:** This concept builds upon the idea of having multiple individual virtual machines, each dedicated to running a single application. By running each application in its own virtual machine, the overall system becomes more robust. Unlike a conventional system where issues with one application could crash the entire machine, in a virtual machine setup, problems with one application are less likely to impact others running on different virtual machines. This is particularly beneficial in scenarios where a new application could potentially harm the entire system due to bugs or viruses.
- **Multiple secure environments:** A system virtual machine (VM) creates a secure sandbox by isolating one system environment from others, offering a level of security beyond what a single operating system can provide. For instance, if a user is hesitant to move applications to a web server without assurance that their resources and activities won't be accessed or monitored by other users on the same server, a virtual machine can be employed. This virtual environment ensures isolation, making it practically impossible for one user to observe or alter another user's data and activities.
- **Mixed-OS environments:** A single hardware platform can simultaneously run two distinct operating systems. This allows a user to utilize office productivity tools on one operating system and, for example, choose another operating system for application development. By installing two virtual machines on a single hardware platform, each dedicated to a specific operating system, users can seamlessly switch between their preferred environments.

3

System and Threat Models

Systematic code reuse has been an elusive goal for software engineering practice since the term was first coined: it represents the use of existing software, or software knowledge, to build new software, following the reusability principles software engineering scholars have extensively evidenced that more recent iterations of software often build upon the foundations of prior versions of established software. This principle applies uniformly, even to malicious software, which frequently evolves to evade detection by anti-malware solutions.

3.1 UNDERSTANDING MODERN MALWARE THREATS

The evolution of malware, spanning its origins as playful pranks or inadvertent experiments to its contemporary utilization for commercial purposes, constitutes a compelling narrative marked by significant milestones and interrelated developments.

Numerous instances of malware generation have been meticulously documented within controlled laboratory environments. However, these occurrences remained restricted within the controlled confines of laboratory settings, isolated from broader digital ecosystems. A pivotal juncture in the malware's evolutionary journey unfolded in 1981 with the emergence of a computer virus that extricated itself from its originative boundaries, coinciding with the nascent era of personal computing.

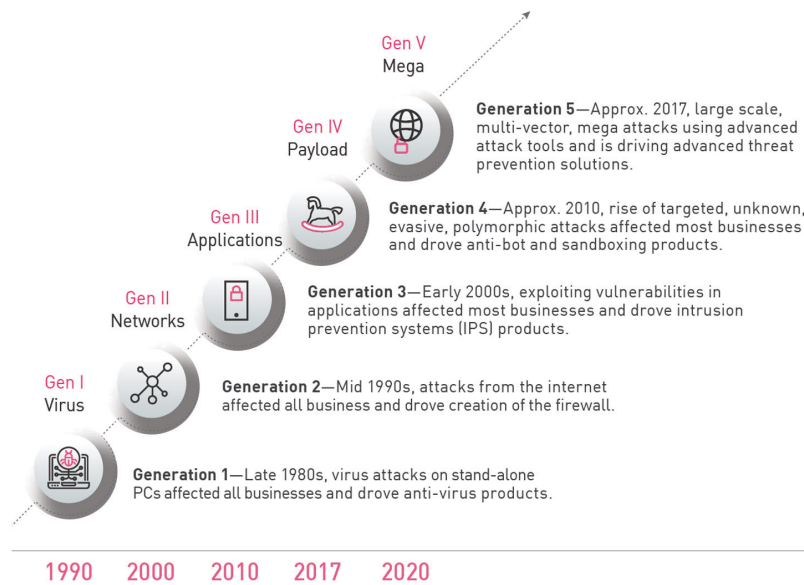


Figure 3.1: The Evolution Of Cyber Threats [1].

In 1986, another milestone was reached as the first Microsoft PC virus surfaced in the wild, closely mirroring the characteristics of its predecessor by predominantly causing disruption rather than substantial damage. Notably, this virus ushered in a new phase of malware concealment, effectively eluding detection by hiding within storage media.

Further complexity in the malware narrative emerged in 1990 with the appearance of a self-propagating and self-replicating program, known as the Morris worm, released onto the burgeoning internet landscape. This period also marked the introduction of the term "malware," signifying malicious software, a widely accepted umbrella term encompassing all software with detrimental intent within computational systems. The subsequent decades witnessed a transformative evolution in malware, characterized by a two-fold dynamic progression: an escalation in both the intricacy and the sheer volume of malware instances (see Figure 3.1). This interconnected series of events and developments underscores the intriguing evolution of malware within the digital age. An investigation into the historical evolution of malware is essential for a comprehensive comprehension of these malicious software entities. This endeavor enables a deeper understanding of their development and evolution over the years, ultimately contributing to enhanced knowledge and insights into the nature of malware.

- The first generation (DOS Viruses) of malware mainly replicate with the assistance of

human activity.

- Second generation malware self-replicate without help and share the functionality characteristics of the first generation. They propagate through files and media.
- Third Generation utilise the capabilities of the internet in their propagation vectors leading to big impact viruses.
- Fourth Generation are more organisation specific and use multiple vectors to attack mainly anti-virus software or systems due to the commercialisation of malware.
- Fifth Generation is characterised by the use of malware in cyberwarfare and the now popular malware as a service [1].

Each jump in generation is characterised by increase in complexity of the malware and more propagation vectors. Tricks of the older generation of malware are always seen to be re-utilised in newer generations of malware and complexities discovered over the years always seem to follow the evolving trends in technology [36].

3.1.1 MALWARE EXECUTABLE CODE

In contemporary malware, like viruses, trojans, and worms, a common technique for hiding their true intentions is to create and run program code while the program is already running. When these types of malware run, they change or reveal a hidden piece of program code that was kept secret when the program was originally created. The way they change this code can be quite simple, like mixing in some data with the code (for example, using XOR encryption), or more complex. Regardless of how complex it is, the result is that when someone tries to analyze the program, it won't look like regular code, making it harder to figure out what the program is actually doing. As an example, encrypted viruses are a type of malware that consistently uses the same method to reveal their hidden code when they run, even in different versions.

In the above figure 3.2, Starting with a malware instance, we begin by performing static analysis over it to acquire a model of what its execution would look like if it did not generate and execute code at runtime; this is depicted in Step 1. The statically derived model and the malware instance are then fed into the dynamic analysis component where the malware is executed in a sterile, isolated environment. The malware's execution is paused after each instruction and its execution context is compared with the static code model, as shown in Step

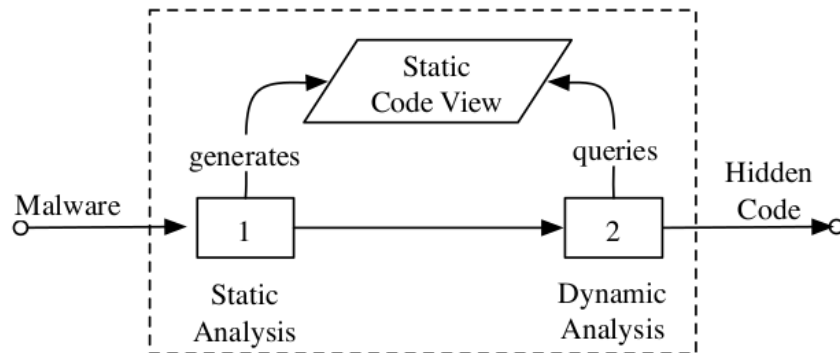


Figure 3.2: Hidden Code Extraction [2].

2. When the first instruction of a sequence not found in the static model is detected, representations of that unknown instruction sequence are written out and the malware’s execution is halted [2]. However, this approach doesn’t aim to identify if a program is malware. Instead, it enhances current malware research and detection methods by using a program’s unpacked code to conduct a more thorough and faster analysis. For a better understanding, it’s advisable to analyze malware behavior within a virtual sandbox environment.

3.1.2 CHARACTERISTICS OF MODERN MALWARE

Fundamentally, there are two types of threats to any network and computer systems, Active Threats and Passive Threats. Passive threats are more often than not, a byproduct or subset of successful execution of an active threat and can be easily dealt with, once our computer system or network is free of or immune to active threats [37].

- Active banking malware is designed to steal account credentials by removing the two-factor authentication system. A popular approach involves Transaction Authentication Number (TAN) theft. TAN is used by online banking services as a form of single use one-time passwords to authorize financial transactions. When the bank receives a request from the user (either via mobile or desktop), it generates the TAN and sends it via SMS to the bank customer’s device. This process is intercepted by the banking Trojan malware that extracts the TAN and sends it back to the bank to gaining access to bank account to complete one time the illegal banking transaction (e.g. funds withdrawal). The users awaiting for the TAN typically think that their request is not delivered and therefore request another TAN number.
- Passive banking malware. In contrast to the active banking malware, the passive malware is designed to monitor the use of mobile banking apps. This type of banking Trojan

disguises itself as legitimate apps (i.e. Google Play Store apps) and once installed, it will run as a service in the background to monitor events on the host device. This enables it to capture incoming SMS, monitor installed apps, and communicate with a remote server. The malware then searches for the existence of any targeted banking apps on the victim's mobile. If any results found, it will remove and download a malicious version to replace the original apps. This malicious version displays a fake user interface asking for user to input their credential information. The attackers then can sniff the banking credentials for illegal banking transaction. They can also capture other useful data that generate revenue for them (e.g. credit card number), see figure 3.3 [3].

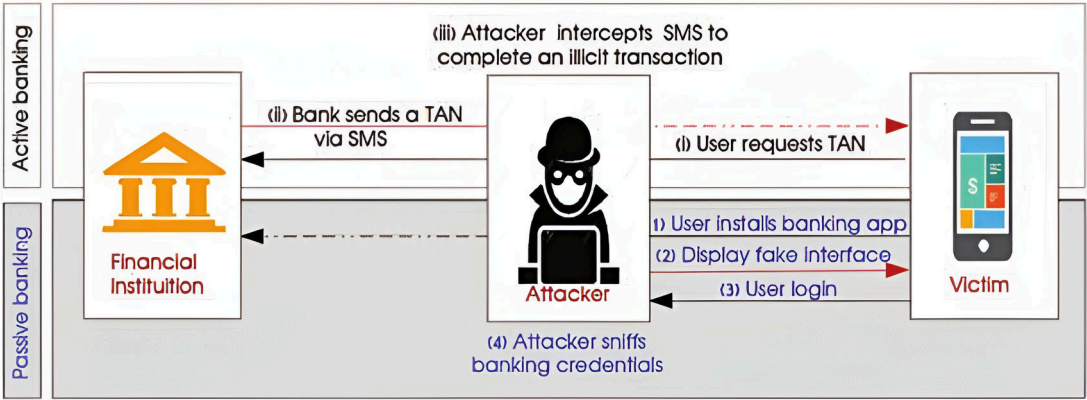


Figure 3.3: Active vs Passive Attack [3].

From observing infection patterns of famous malware like Wnna- Cry, we can deduce that the number of newly infected machines is exponentially and directly proportional to the number of already infected machines. Therefore, it is safe to characterize the nature of actively evolving threats and their working can be defined as: After a victim host computer identifies and infects a vulnerable computing system on any given network, this newly victimized computer system will automatically and autonomously scan all available networks, so as to identify and infect other vulnerable computer systems [38]. It's important to mention that most current detection methods assume that infected computers are constantly scanning the network to spread themselves as fast as possible. However, malicious programmers have started to create attack patterns that intend to disrupt and mislead the existing active threat detection systems. Specifically, 'polymorphic blending', 'camouflaging' and 'obfuscating' are the newer attack patterns being increasingly used by a recently discovered set of active threats, which elude detection by hibernating (i.e., by stopping duplication and propagation) for a pre-programmed or dynamically determined period of time [37]. Furthermore, these modern active threats, in addition

to the patterns mentioned earlier, also share a characteristic of self-replicating and spreading, much like traditional active attacks. In other words, they can quickly attack and infect numerous vulnerable computer systems. Nonetheless, they are very different from traditional threats in the manner where they blend, camouflage or obfuscate any noticeable patterns in the currently affected computer systems for a pre-programmed or dynamically instantiated unit of time. The camouflaging technique involves manipulating the signatures of infected files. Such a manipulation of infected files' signatures prohibits existing detection schemes from tracking any unwanted activity (e.g., duplication and propagation). This is extremely dangerous as patterns of exponential infection often go undetected until it's too late to stop the infection [39].

3.1.3 MALWARE DELIVERY

Malware has changed over the years, as discussed in section 3.1, each version used different methods to spread or infect computers, for example macro viruses intended for Microsoft Word must be in Word documents to be effective, and so anti-virus programs typically do not scan incoming executable for those viruses—but they do scan any incoming Microsoft Word files for them. This creates a “gap” in protection. If, for example, a macro virus were embedded in an executable file in such a way that the executable file would ignore it when executed, but a second program could locate that virus and load it into an existing Microsoft Word document in such a way that the virus would be triggered when the file were opened, the anti-virus programs would not detect the macro virus' entry onto the system [40].

An attacker must achieve two goals to infect any user's computer system through web browser.

- Attacker must find an approach to connect with the user or victim.
- Attacker need to install malicious code on the victim's computer.

Both these goals will be achieved quickly and without the concern of the user which depends on attacker's tactics. There is another way to infect a user's system with malware, attacker just simply ask user to visit a website which contains a malicious code while user visit this site his system will automatically get infected by malware. These days' attackers are focusing on different delivery mechanisms, and usually send malware infected post over social networking sites, such as Facebook. Other attackers decide to target websites that potential victimized people will visit on their own. To achieve this, an attacker compromises the targeted site and inserts a little

piece of HTML code that associates back to their server. This malicious code can be loaded from anywhere, including a totally distinctive website. Every time a user visits a compromised website in this manner, the attacker’s malicious code has a chance to infect user’s system with malware [41]. Sometimes, the author of the malicious file uses a packer software to first parse portable executable (PE) internal structures. Then, it reorganizes PE headers, sections, import tables, and export tables into new structures. During packing, a packer software sometimes encrypts the code and resource sections using the compression and encryption libraries [42]. The PE files are packed in a manner that significantly hinders both reverse engineering attempts and the ability of antivirus programs to discern the malicious nature of the PE file. Since April 2019, Japan Computer Emergency Response Team Coordination Center (JPCERT/CC) has been monitoring a series of attacks involving the distribution of targeted emails to Japanese organizations. The objective of these emails is to persuade recipients to download a harmful shortcut file. These emails include a link to a shortcut file hosted on a cloud service. When this shortcut file is executed, it triggers a downloader. The chart below illustrates the sequence of events that occur from the moment the shortcut file is executed to when the downloader successfully infects a host.

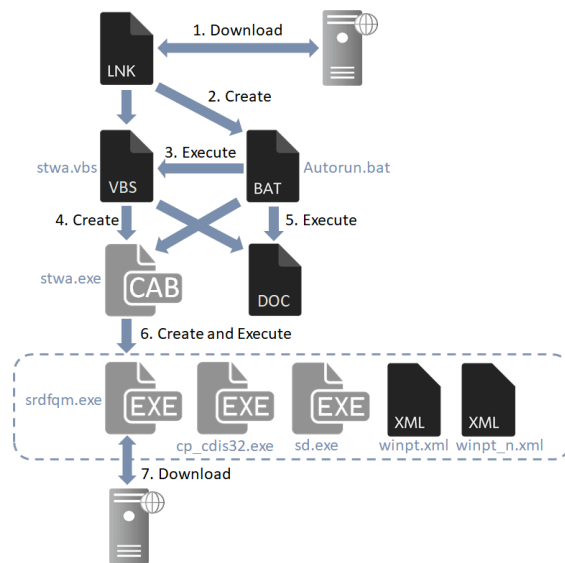


Figure 3.4: Shortcut execution leading to host infection [4].

The shortcut files examined by JPCERT/CC have been found to include the following code. This code initiates the download of an HTML file (as depicted in Figure 3.4), which incorporates VBScript. Subsequently, this VBScript is responsible for the creation and execution of

both a VBS file (stwa.vbs) and a BAT file (Autorun.bat).

```
1 <html>
2 <head>
3 <title>
4 </title>
5 <script language=vbscript>
6   on error resume next
7   Set WshShell = CreateObject("Wscript.Shell")
8   Set WshSysEnv = WshShell.Environment("Process")
9   temppath = WshSysEnv.Item("TEMP")
10  tempvbs = temppath + "\"+stwa.vbs"
11  tempbat = temppath + "\"+Autorun.bat"
12  Set fso = CreateObject("Scripting.FileSystemObject")
13  Set vbsfile = fso.CreateTextFile(tempvbs, True)
14  vbsfile.writeline "Sub NN(SF,DF,TF,SP,n)"
15  vbsfile.writeline "Dim fso"
16  vbsfile.writeline "Set fso=CreateObject(""Scripting.FileSystemObject"")"
17  vbsfile.writeline "Dim tmp"
18  vbsfile.writeline "Set tmp=fso.GetSpecialFolder(2)"
19  vbsfile.writeline "Dim BS"
20  vbsfile.writeline "Dim OS"
21  vbsfile.writeline "Set BS=CreateObject(""ADODB.Stream"")"
22  vbsfile.writeline "BS.Type=1"
23  vbsfile.writeline "BS.Open"
24  vbsfile.writeline "BS.LoadFromFile SF"
25  vbsfile.writeline "Set OS=CreateObject(""ADODB.Stream"")"
26  vbsfile.writeline "OS.Type=1"
27  vbsfile.writeline "OS.Open"
28  vbsfile.writeline "BS.Position=SP"
29  vbsfile.writeline "BS.CopyTo OS,n"
30  vbsfile.writeline "OS.SaveToFile tmp&"\"&TF,2"
31  vbsfile.writeline "Dim qwer"
32  vbsfile.writeline "Dim asdf"
33  vbsfile.writeline "Dim zxcv"
34  vbsfile.writeline "Dim uiop"
35  vbsfile.writeline "Dim hjkl"
36  vbsfile.writeline "Dim vbnm"
37  vbsfile.writeline "Set qwer=CreateObject(""Scripting.FileSystemObject"")"
38  vbsfile.writeline "Set asdf=qwer.GetFiles(tmp&"\"&TF)"
```

Figure 3.5: Shortcut-triggered HTML file download [4].

When stwa.vbs is run, it deciphers the data encoded in Base64 within the shortcut file (as seen in Figure 3.6). It then stores this decoded data as a Windows executable file (stwa.exe) and a phony Word document that appears on the screen.

| | | |
|----------|---|---------------------------------|
| 000008e0 | 53 00 79 00 73 00 74 00 65 00 6d 00 33 00 32 00 | S . y . s . t . e . m . 3 . 2 . |
| 000008f0 | 28 00 43 00 3a 00 5c 00 57 00 69 00 6e 00 64 00 | (. C . . . ¥ . W . i . n . d . |
| 00000900 | 6f 00 77 00 73 00 29 00 00 00 00 00 00 00 00 | o . w . s .) |
| 00000910 | 65 00 00 00 31 53 50 53 a6 6a 63 28 3d 95 d2 11 | e . . . 1SPS . jc (= . . . |
| 00000920 | b5 d6 00 c0 4f d9 18 d0 49 00 00 00 1e 00 00 00 | 0 . . . I |
| 00000930 | 00 1f 00 00 00 1c 00 00 00 43 00 3a 00 5c 00 57 | C . . . ¥ . W |
| 00000940 | 00 69 00 6e 00 64 00 6f 00 77 00 73 00 5c 00 53 | . i . n . d . o . w . s . ¥ . S |
| 00000950 | 00 79 00 73 00 74 00 65 00 6d 00 33 00 32 00 5c | . y . s . t . e . m . 3 . 2 . ¥ |
| 00000960 | 00 63 00 6d 00 64 00 2e 00 65 00 78 00 65 00 00 | . c . m . d . . . e . x . e . . |
| 00000970 | 00 00 00 00 00 00 00 00 00 60 00 00 00 03 00 00 | |
| 00000980 | a0 58 00 00 00 00 00 00 00 77 69 6e 2d 6a 31 6d | . X win - jim |
| 00000990 | 33 6e 37 62 66 72 62 6c 00 8c 11 29 c3 01 31 b3 | 3n7bfrbl 1 . |
| 000009a0 | 40 85 8e f7 3b 31 c1 74 bb 18 b5 31 1f 66 d4 e5 | @ . . . : 1 . t . . . 1 . f . . |
| 000009b0 | 11 8f 2f 00 0c 29 81 1b f9 8c 11 29 c3 01 31 b3 | . . /) 1 . |
| 000009c0 | 40 85 8e f7 3b 31 c1 74 bb 18 b5 31 1f 66 d4 e5 | @ . . . : 1 . t . . . 1 . f . . |
| 000009d0 | 11 8f 2f 00 0c 29 81 1b f9 00 00 00 00 54 56 71 | . . /) TVq |
| 000009e0 | 51 41 41 4d 41 41 41 41 45 41 41 41 41 2f 2f 38 | QAAMAAAAEAAAA//8 |
| 000009f0 | 41 41 4c 67 41 41 41 41 41 41 41 41 41 51 41 41 | AALgAAAAAAAAAQAA |
| 00000a00 | 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 | AAAAAAAAAAAAAAAAAA |
| * | | |
| 00000a20 | 41 41 41 41 41 0a 41 41 41 41 41 41 41 41 38 41 | AAAAA . AAAAAAAAA8A |
| 00000a30 | 41 41 41 41 34 66 75 67 34 41 74 41 6e 4e 49 62 | AAAA4fug4AtAnIb |
| 00000a40 | 67 42 54 4d 30 68 56 47 68 70 63 79 42 77 63 6d | gBTM0hVghpcyBwcm |
| 00000a50 | 39 6e 63 6d 46 74 49 47 4e 68 62 6d 35 76 64 43 | 9ncmFtIGNhbm5vdC |
| 00000a60 | 42 69 5a 53 42 79 64 57 34 67 61 57 34 67 0a 52 | BiZSBydW4gaW4g . R |
| 00000a70 | 45 39 54 49 47 31 76 5a 47 55 75 44 51 30 4b 4a | E9TIGivZGUuDQOKJ |
| 00000a80 | 41 41 41 41 41 41 41 41 41 42 6d 5a 65 30 5a 49 | AAAAAAAAABmZeOZI |
| 00000a90 | 67 53 44 53 69 49 45 67 30 6f 69 42 49 4e 4b 4b | gSDSiIEg0oiBINKK |
| 00000aa0 | 33 77 57 53 69 55 45 67 30 6f 72 66 41 64 4b 4d | 3wWSiUEg0orfAdKM |
| 00000ab0 | 67 53 44 53 69 74 38 0a 45 45 6f 74 42 49 4e 4b | gSDSit8 . EEotBINK |
| 00000ac0 | 49 67 53 43 53 76 51 45 67 30 6f 72 66 41 42 4b | IgSCSvQEg0orfABK |
| 00000ad0 | 66 41 53 44 53 67 58 43 2f 55 6f 6a 42 49 4e 4b | fASDSgXC/UojBINK |
| 00000ae0 | 4b 33 77 58 53 69 4d 45 67 30 6f 72 66 42 4a 4b | K3wXSiMEg0orfBJK |
| 00000af0 | 49 77 53 44 53 6c 4a 70 59 32 67 69 42 49 4e 4b | IwSDSiJpY2giBINK |

Figure 3.6: Base64-encoded content within the shortcut [4].

The stwa.exe file uses a self-extract format (CAB) and generates a group of files when it's launched. Among these files, srdfqm.exe serves as the downloader responsible for carrying out the primary tasks, including communication. This malware primarily functions as a download agent. Upon execution, it initiates specific communication procedures with a Command and Control (C&C) server. Subsequently, it retrieves a file from the C&C server and stores it on the local device. The nature of the malware determines the subsequent course of action, which may involve carrying out a particular type of attack [4].

3.2 VIRTUAL SANDBOX THREATS AND VULNERABILITIES

Several years ago, manually handling the analysis of hundreds of thousands of malware samples became unmanageable. To address this challenge, sandboxes have been employed to automate the process of examining malware samples. This approach enables the collection of data on how the malware behaves dynamically. Malware creators are becoming more clever to achieve their

goal of infecting and taking control of computer systems. Many industries use virtual sandboxes to protect their systems and workflow, but this hasn't stopped malware creators. They know that their harmful programs are tested in a safe virtual environment before they can harm a real system. This awareness has pushed attackers to find smart ways to trick the testing environment into not noticing their harmful activities. This ongoing cat-and-mouse game between cybersecurity professionals and malware creators has prompted a continuous evolution in the tactics employed by the latter. Malware authors have developed techniques that can detect when they are running in a virtual environment, allowing them to lay low and avoid arousing suspicion. Attackers employ multiple techniques to identify the existence of a virtual sandbox. To overcome the protective virtual environment, the attacker needs a deep understanding of the underlying sandbox structure. Although not many malware use anti-sandbox techniques to evade detection, some of them do. The traditional anti-sandbox techniques include detection of virtualization, running processes, detection of debuggers, detection of hooked functions, injected DLLs, etc. Most of these checks can be easily flagged as malicious. Some advanced techniques are also known; detecting whether sleep functions are emulated, detection of network connectivity, mouse movement, etc. But the traditional virtualization detection techniques can be detected, and the malware can be blocked. An attacker designs their malware with the aim of tricking the virtual sandbox. During the design process, the attacker considers specific factors and parameters, as outlined in Table 3.1 below, that the malware should be able to identify.

| Technique's | Approach's |
|-------------------------|---|
| Virtualization | Hiding virtualization from a malicious process is challenging, and even with tools that aim to conceal it, new detection methods can emerge by exploiting common mistakes. |
| C&C Servers | Anti-sandbox techniques, such as IP blacklists, protect C&C servers. AV-Tracker is one example, but it has limitations when the C&C server's IP is revealed on new sandboxes. |
| Validator Style Malware | APTs employ validator-style malware, which assesses the environment and deploys advanced malware only on validated, non-sandboxed targets. |
| Windows ID | Windows product ID – is it a known sandbox product ID? Or a faked one including alphabetic letters? |
| HDD | Hard Disk Type, layout – is HDD less than 20 GBytes? |
| Hardware Layout | Hardware layout (processor, memory, motherboard, BIOS, network cards) – is it running with 256 Mbyte of memory? Is this a Qemu? Is the MAC address known for Virtualbox? |
| Windows Settings | Windows settings (installation date, version, current time) – e.g. is the current time on the OS years behind the real current time? |
| Screen Resolution | Screen resolution – is it 640x480? Or 800x600? |
| Username's | Username, computer name, domain – in a targeted attack, attackers might know the Windows domain name, and only allow running if the domain is detected. |
| Networks | Available network shares – no network shares in a corporate environment? |

Table 3.1: Modern Malware Approach's.

3.3 THREAT MODELS

As we have discussed previously, virtual sandboxes have inherent limitations and vulnerabilities when it comes to analyzing and detecting malware. These limitations have become increasingly evident as attackers continue to evolve and employ more sophisticated methods to infiltrate systems.

In modern times, malware has become adept at assessing specific system parameters, enabling it to differentiate between a simulated sandbox environment and an actual, operational

system. It conceals its true nature during sandbox analysis, only revealing its malicious intent when executed within a genuine system. This poses a substantial challenge for security researchers aiming to comprehensively understand the behavior of such malware and develop effective countermeasures to protect systems from future attacks.

To address this dilemma, researchers have explored the concept of hardware-based sandboxes as a potential solution. A hardware-based sandbox differs from its virtual counterpart in that it utilizes physical hardware resources, such as dedicated processors and memory, to isolate and execute potentially malicious code. The primary idea behind this approach is that when malware is executed within a hardware-based sandbox, it is more likely to exhibit its genuine behavior. This is in stark contrast to virtual sandboxes, where the malware often remains dormant or employs evasion techniques, making it difficult to uncover its true nature. Hardware-based sandboxes offer several advantages in the realm of malware analysis. Firstly, the inherent hardware isolation makes it considerably more challenging for malware to detect that it is running within a controlled environment. This minimizes the risk of malware altering its behavior to evade detection, as it commonly does in virtual environments. Secondly, hardware-based sandboxes provide researchers with a more accurate view of how the malware interacts with system resources and how it attempts to exploit vulnerabilities. This insight is invaluable for developing robust security measures.

Furthermore, by studying the behavior of modern malware within a hardware-based sandbox, researchers can gain a deeper understanding of its tactics, techniques, and procedures (TTPs). This knowledge can then be used to develop proactive security measures that not only detect known malware but also anticipate and thwart new and emerging threats.

4

Methodology

The core rationale behind the development of a hardware-based sandbox is to delve into the true nature of malware and comprehensively analyze its behavior in an environment that mirrors its intended target. It is well-known that malicious actors tailor their malware to exploit vulnerabilities and security gaps in real systems, rather than virtual sandboxes. Consequently, our hardware-based sandbox is meticulously constructed to replicate the characteristics and attributes of a real operating environment. In this chapter, we will provide an in-depth account of our approach, which involves the utilization of an actual system alongside a carefully chosen set of hardware components to form the backbone of our sandbox infrastructure. This method enables us to closely mimic the conditions under which malware typically operates in the real world. By creating a more faithful representation of genuine systems, we aim to gain a more precise understanding of how malware behaves and to facilitate advanced threat analysis and detection. This innovative approach serves as a crucial step towards enhancing cybersecurity practices and safeguarding against evolving threats.

4.1 HARDWARE COMPONENTS

Before we dive into the details of our hardware sandbox's architecture, it's essential to take a closer look at the various components we've used. Each of these components has a significant role to play, and it's important to understand how they work and what functions they perform within the sandbox. This understanding will provide a solid foundation for comprehending

the overall structure and operation of our hardware sandbox.

4.1.1 LEETDMA

LeetDMA is a hardware device created by Engima-X, designed to work seamlessly with the PCILEECH tool developed by Ulf Frisk, an IT security analyst from Sweden. Figure 4.1 provides an illustration of the LeetDMA board for reference.

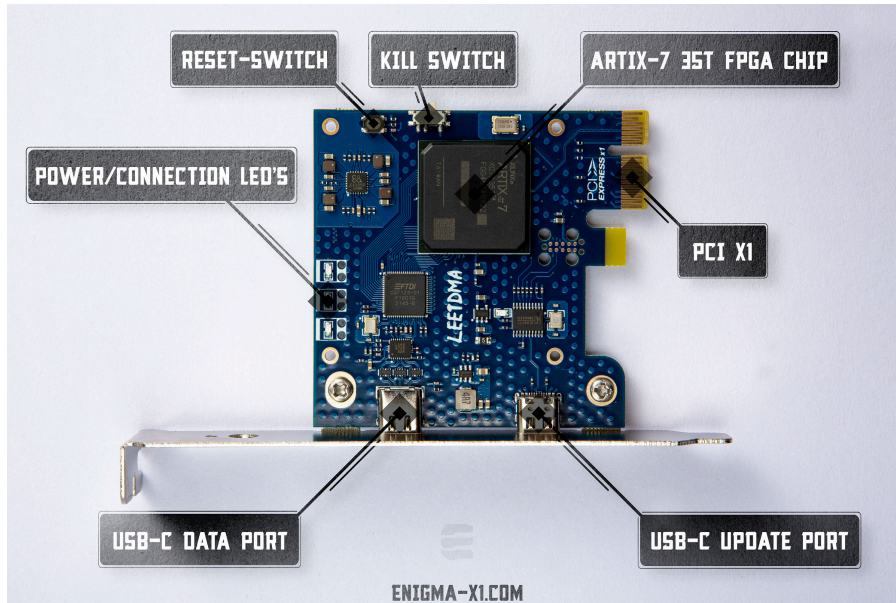


Figure 4.1: LeetDMA Device [5].

- **PCI XI** The LeetDMA device is equipped with a PCIe x1 interface that connects to the motherboard's PCIe x1 slot. This connection allows LeetDMA to gain access to the computer's physical memory. It appears as a standard PCIe device to the computer, leading the system to lower its security privileges. Once the device is properly connected to our sandbox, a red LED indicator illuminates, indicating that the device is successfully connected and ready for operation.
- **USB-C DATA PORT** The USB data port is an essential component of the hardware sandbox, facilitating the connection between the sandbox and the host machine through a USB cable. When the guest and host are correctly linked, the process of transferring the physical memory from the guest machine to the host machine becomes a straightforward task.
- **USB-C UPDATE PORT** The USB-C update port serves as a means for developers or users to program the device at a later time when necessary. It provides a way to make

adjustments or updates to the device's programming in the future.

- **KILL SWITCH** The kill switch in the LeetDMA device is a safety feature used to quickly and forcefully shut down the device when needed. It serves as a security measure to prevent unintended or unauthorized actions and is crucial in ensuring the device's safe and controlled operation.
- **RESET-SWITCH** The reset switch in the LeetDMA device is used to restore the device to its default or initial state. It can be helpful in situations where the device encounters issues or malfunctions, allowing it to be reset to a known configuration for troubleshooting or regular operation. The reset switch provides a convenient way to address operational challenges and maintain the device's reliability and functionality.
- **POWER PORT** The "power port" in LeetDMA is typically used to provide electrical power to the device. It serves as the connection point for the power source, often an external power adapter, to supply the necessary electrical energy to operate the LeetDMA device. The power port ensures that the device receives a stable and consistent power supply, which is essential for its proper functionality and performance.
- **ARTIX-7 FPGA CHIP** Field Programmable Gate Arrays (FPGAs) are special computer chips made up of a grid of tiny customizable building blocks and pathways that connect them. What makes FPGAs unique is that you can change how they work even after they've been made. This sets them apart from Application Specific Integrated Circuits (ASICs), which are created to do specific jobs and can't be changed once they're made. The Artix-7 FPGA within the LeetDMA device serves as a versatile and programmable component that plays a pivotal role in memory access and data manipulation. It enables tasks such as data extraction, memory control, buffering, and the implementation of custom functionality. This programmable hardware adds flexibility and adaptability to the LeetDMA device, allowing it to effectively interface with and analyze the memory of target systems for various purposes, making it a valuable asset for memory-related operations[10]

| | FEATURES |
|----------------|--|
| Artix-7 | Advanced high-performance FPGA logic based on real 6-input lookup table (LUT) technology configurable as distributed memory. |
| | 36 Kb dual-port block RAM with built-in FIFO logic for on-chip data buffering. |
| | High-performance SelectIO™ technology with support for DDR3 interfaces up to 1866 Mb/s. |
| | High-speed serial connectivity with built-in multi-gigabit transceivers from 600 Mb/s to max. rates of 6.6 Gb/s up to 28.05 Gb/s, offering a special low- power mode, optimized for chip-to-chip interfaces. |
| | A user configurable analog interface (XADC), incorporating dual 12-bit 1MSPS analog-to-digital converters with on-chip thermal and supply sensors. |
| | DSP slices with 25 x 18 multiplier, 48-bit accumulator, and pre-adder for high-performance filtering, including optimized symmetric coefficient filtering. |
| | Powerful clock management tiles (CMT), combining phase-locked loop (PLL) and mixed-mode clock manager (MMCM) block for high precision and low jitter. |
| | Quickly deploy embedded processing with MicroBlaze processor |
| | Integrated block for PCI Express® (PCIe), for up to x8 Gen3 Endpoint and Root Port designs. |
| | Wide variety of configuration options, including support for commodity memories, 256-bit AES encryption with HMAC/SHA-256 authentication, and built-in SEU detection and correction. |
| | Low-cost, wire-bond, bare-die flip-chip, and high signal integrity flipchip packaging offering easy migration between family members in the same package. All packages available in Pb-free and selected. |
| | Designed for high performance and lowest power with 28 nm, HKMG,HPL process, 1.0V core voltage process technology and 0.9V core voltage option for even lower power. |

Table 4.1: Artix-7 Features [10].

Table 4.1 above highlights the noteworthy features of the Artix-7 series. Meanwhile, Figure 4.2 below illustrates the schematic representation of the Artix-7 series chip.

Block Diagram

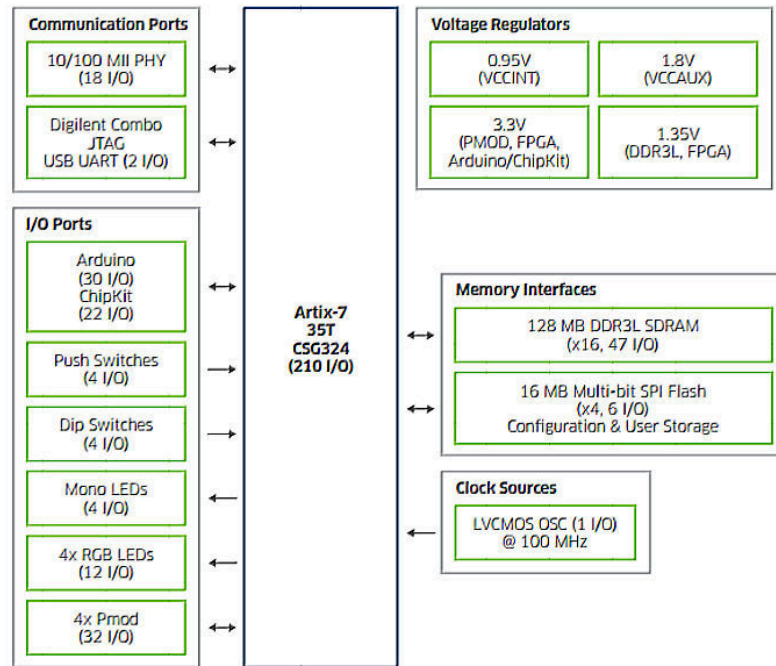


Figure 4.2: Artix-7 35T Block Diagram [6].

4.1.2 KVM SWITCH

A KVM (Keyboard, Video, and Mouse) switch is a device that allows users to control multiple computers with a single keyboard, monitor, and mouse. It works by enabling you to switch between different connected computers, effectively giving you control over multiple machines without the need for separate input devices for each.

To connect two monitors with a KVM switch, you would typically have the KVM switch connected to both computers and each computer connected to a monitor. The KVM switch enables you to toggle control between these computers, allowing you to use one keyboard and mouse to operate both machines. This simplifies the process of working with multiple computers.

We implemented a KVM switch between the host and the guest (sandbox) machine. The KVM switch allowed us to connect two monitors while providing control over both machines

with a single mouse and keyboard. This streamlined our workflow, making it more efficient. In Figure 4.3 below, you can see the KVM switch connected to two monitors.

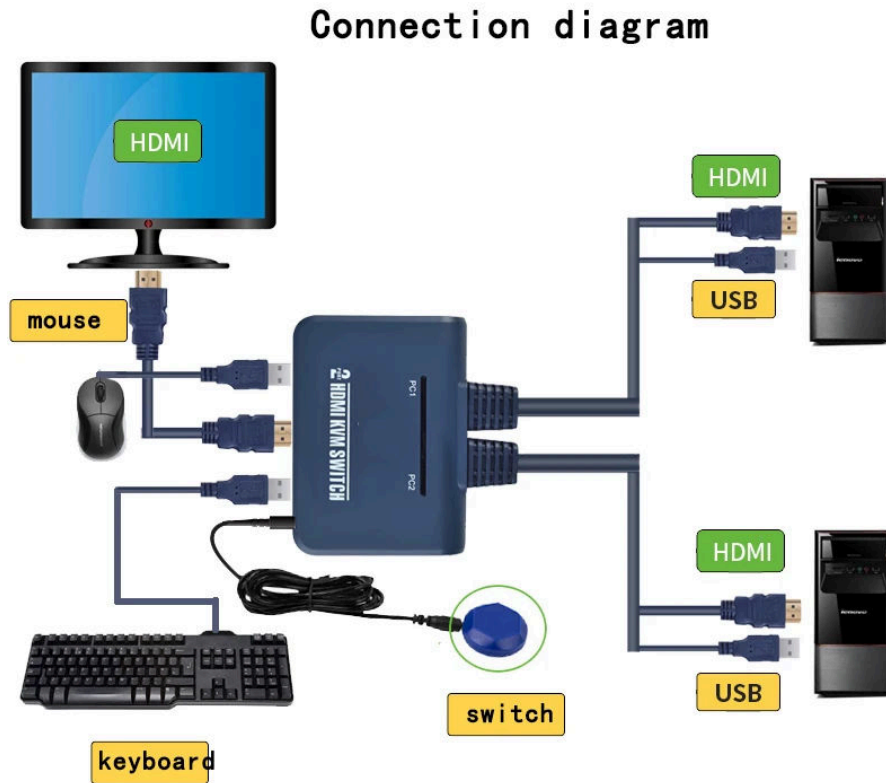


Figure 4.3: Dual KVM Switch.

4.1.3 NETWORK-ATTACHED STORAGE (NAS)

A Network-Attached Storage (NAS) device is like a special kind of storage box that's part of your network, not directly connected to a single computer. It's like a smart storage box because it has its own computer brain and system to run software. This brain helps make sure that files stored in it can be easily shared with the right people.

The great thing about a NAS device is that it's like a magic box that can be opened by lots of people, different computers, phones, and even from far away if it's set up correctly. It's a handy way to store and share files on your network. There are mainly two ways to add extra storage

to your computer. First, you can directly connect a hard drive or SSD to your computer using USB or other cables, and it's usually exclusive to that computer. This is called "direct-attached storage" (DAS). Second, you can connect storage devices to your home or business network, either through wired (ethernet) or wireless (Wi-Fi) connections. These network-connected storage devices are known as "network-attached storage" (NAS). NAS systems can be accessed by multiple users on the same network, and sometimes even from the internet.

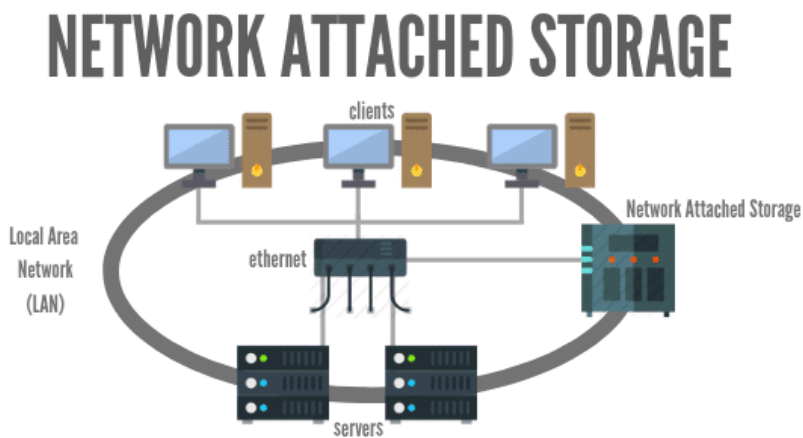


Figure 4.4: Network Attached Storage [7].

As depicted in Figure 3.5, an illustration reveals the interconnection of a NAS (Network-Attached Storage) device with three client machines. This configuration enables the trio of clients to engage in file sharing and data exchange. Notably, should the administrator opt to expand the network to include additional clients, such an integration can be seamlessly executed. Notably, the salient characteristic of this arrangement lies in the absence of physical interconnections among the clients, with all data interactions facilitated through the intermediary NAS device.

4.2 OPERATING SYSTEM RESET

It is imperative to highlight that a fundamental characteristic of a sandbox environment is the requirement for a pristine operating system for the analysis of each new malware instance. In the case of virtual sandboxes, this need is readily met by simply shutting down the system; upon its subsequent activation, a fresh operating system is made available for the analysis of a different malware specimen. In our hardware-based sandbox implementation, we have addressed

this necessity by integrating the software solution "Reboot Restore Rx" provided by Horizon DataSys. This software ensures the restoration of a clean operating environment after each analysis session, facilitating the seamless transition to a new malware analysis task.

4.2.1 DATA ENCRYPTION WITH ROLLBACK RX

The software architecture employed in Reboot Restore Rx Pro/ RollBack Rx Pro relies on the mapping of hard drive sectors. This sector-oriented safeguard is inherently encrypted, rendering it concealed within the file system. Consequently, it remains imperceptible to potential threats such as viruses and malware. Data encryption, a well-known practice among PC users, serves the purpose of safeguarding computer data against unauthorized access. Various techniques exist to achieve this goal, but they all share a fundamental concept: transforming data into a different, coded format, accessible only to those possessing a secret key, often referred to as a decryption key or password. RollBack Rx, a software akin to Windows time machine, offers instant data recovery for computers. While on the surface it may not seem related to data encryption, in practice, RollBack Rx provides a distinctive and effective method for protecting data from unauthorized entry. This paper elucidates how RollBack Rx accomplishes data encryption through its unique, straightforward, and efficient approach—a fortuitous outcome of its instant data recovery design.

STANDARD DATA ENCRYPTION

Before delving into RollBack Rx's method of data encryption, it's helpful to briefly review the standard approach to data encryption. Data encryption's primary objective is to safeguard the confidentiality of digital data when stored on computer systems. To grasp the various techniques employed in data encryption, it's essential to have a fundamental understanding of how data is both stored and accessed on computer systems.

Computer data storage is a multifaceted process, with Windows-based computers entrusting much of this responsibility to the operating system's file system manager. Simplifying this intricate process, it can be divided into two core steps: first, locating the stored data (handled by the indexing system), and second, reading or writing the stored data (the actual data). Standard data encryption primarily focuses on the second step, which involves the data itself. It utilizes encryption algorithms such as DES and AES to take the data and subject it to a series of intricate operations, resulting in a fixed-length ciphertext. This encryption process is highly

effective but can be time-consuming, often requiring hours to encrypt or decrypt an entire drive's worth of data.

ROLLBACK RX ENCRYPTION

RollBack Rx places its data encryption emphasis on the initial step of data storage, which involves the indexing system. While RollBack Rx primarily serves as data protection software and not originally intended as data encryption software, the data encryption feature emerged as an unforeseen outcome of its data protection design.

In a Windows system lacking RollBack Rx, the IO file manager's perspective on data storage simplifies to a binary representation: used space is depicted as black, and free space is depicted as white, as shown in figure 4.5 below.

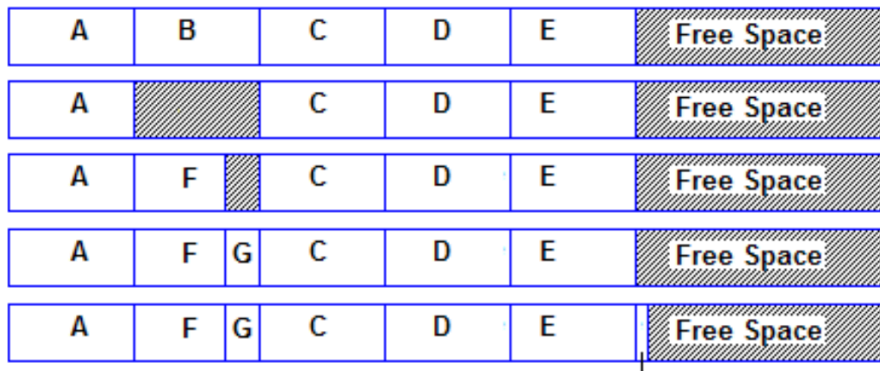


Figure 4.5: File system in the absence of RollBack Rx

On a computer with RollBack Rx, the way data is managed is still like a basic "black and white" picture. But in this case, it's just a small part of the whole colorful picture. Below figure 4.6 shows the imaginary view of file systems with RollBack snapshots.

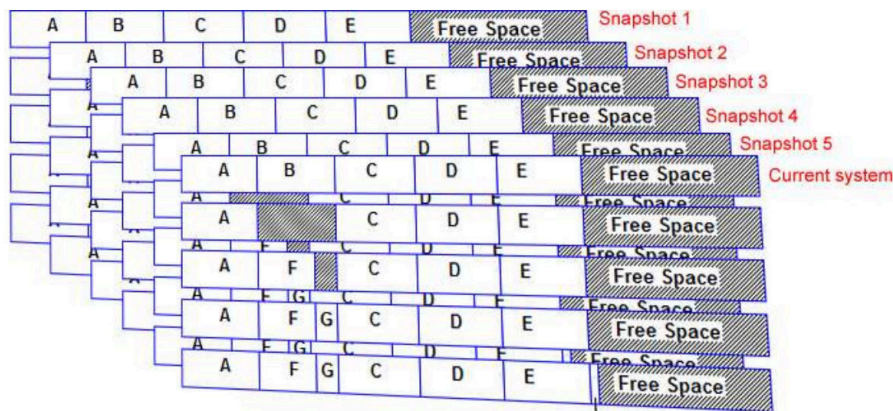


Figure 4.6: File system using RollBack Rx

Using RollBack Rx, we've observed that within a single snapshot, we can't see data from other snapshots. This happens because the current snapshot's file system doesn't have information about files in other snapshots. This approach aligns with RollBack Rx's data protection philosophy, which aims to shield data from potential corruption by keeping it separate.

This same philosophy extends to data encryption: the most effective way to prevent unauthorized access to data is by keeping it hidden from unauthorized users. This forms the basis of data encryption within RollBack Rx.

The question arises: how do we actually achieve this invisibility of data to unauthorized access?

Data stored in RollBack Rx snapshots can only be seen when you're inside that specific snapshot. To access the data, you need to use the right key to enter that particular snapshot. This access or opening of a RollBack Rx snapshot is managed by RollBack Rx's pre-operating system subsystem. It reads information from hidden snapshot tables, which are not visible as regular files in the file system. Moreover, the contents of these snapshot tables are encrypted using AES encryption. To ensure that only authorized users can load a snapshot table, the process of loading and decrypting these snapshot tables is protected by a pre-operating system password. This password system ensures that access to the protected data remains completely secure. For a cautious user, concerns may linger about the security of their actual data. They might wonder how RollBack Rx prevents unauthorized access to their data if someone not allowed were to some-

how get their hands on it. Here's where the brilliance of RollBack Rx's design becomes evident. RollBack Rx addresses this concern by essentially doing nothing different from its regular operation for instant data protection and recovery. The noteworthy aspect of RollBack Rx is that it doesn't back up or restore the actual data, which could be a time-consuming process. The way RollBack Rx reads and writes protected data can be likened to an "encryption" technique. Data encryption typically involves taking plain data, using an encryption algorithm and key to transform it into ciphertext, which can only be viewed in its original form when decrypted with the correct key. In the case of RollBack Rx, it applies a similar concept to protect your data. RollBack Rx safeguards data stored on a PC using a combination of a subsystem and a kernel system. These components work together to create a sector map and a virtual shield, making it so the operating system is not privy to data movements at the sector level.

Each sector that has a corresponding entry on the sector map is marked as "Used." In contrast, sectors without data entries in the sector map are labeled as "Free." When RollBack Rx is installed, the kernel system designates the hard drive, safeguarding the "Used" sectors under this virtual shield. This classification operates independently of the operating system.

Any alterations made after RollBack Rx's installation fall into the "Snapshot Used" sector category. These sectors are an extension of the "Used" ones, with linked pointers connecting them. Importantly, the operating system remains unaware of this two-stage link pointer that exists between the data, bridging the gap between the "Used" and "Snapshot Used" sectors.

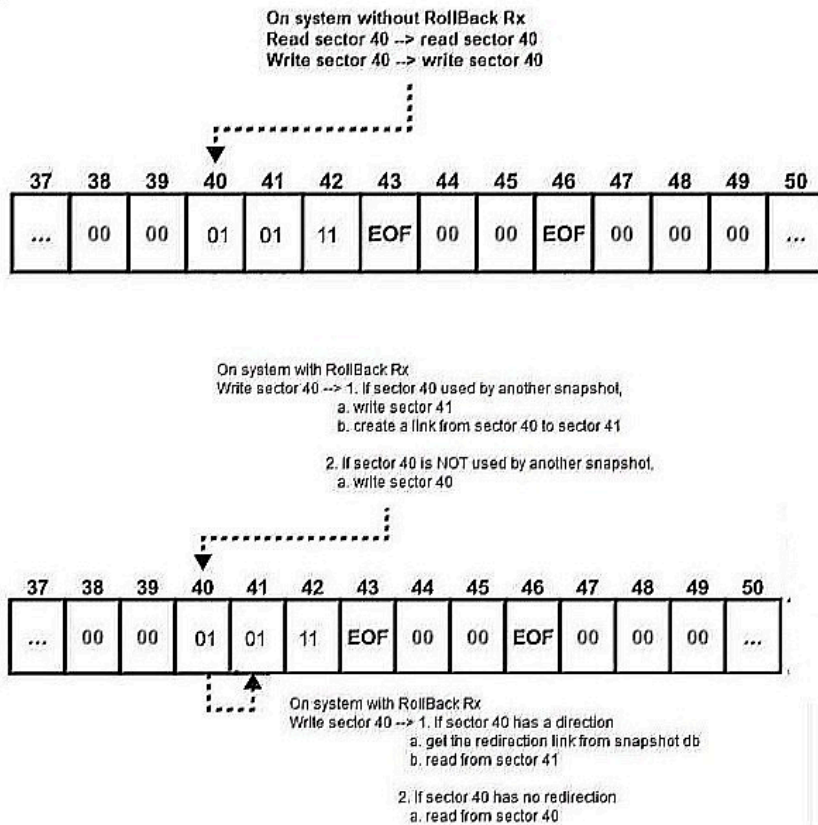


Figure 4.7: RollBack Rx encryption.

As shown in Figure 4.7, within a system using RollBack Rx, the contents of sector 40 are not encrypted. However, unless you access RollBack Rx’s snapshot tables and re-interpret the sector links, there’s no way to discern that the actual content of sector 40 is mirrored in sector 41. The data in sector 40 becomes unintentionally disguised through a series of redirections.

This aspect is where the brilliance of RollBack Rx’s design shines. In contrast to standard encryption software that transforms data into a different form, RollBack Rx dissects the data within a file into smaller components and connects them through redirection and indexing within the snapshot tables. Consequently, the data becomes effectively ”disguised” into another form without undergoing a traditional encryption process[43].

The map of the snapshot tables, which is essential for reconstructing all the links forming the data stored in snapshots (essentially, the data’s disguise), is encrypted. It is also conveniently

protected

4.2.2 DATA STORAGE

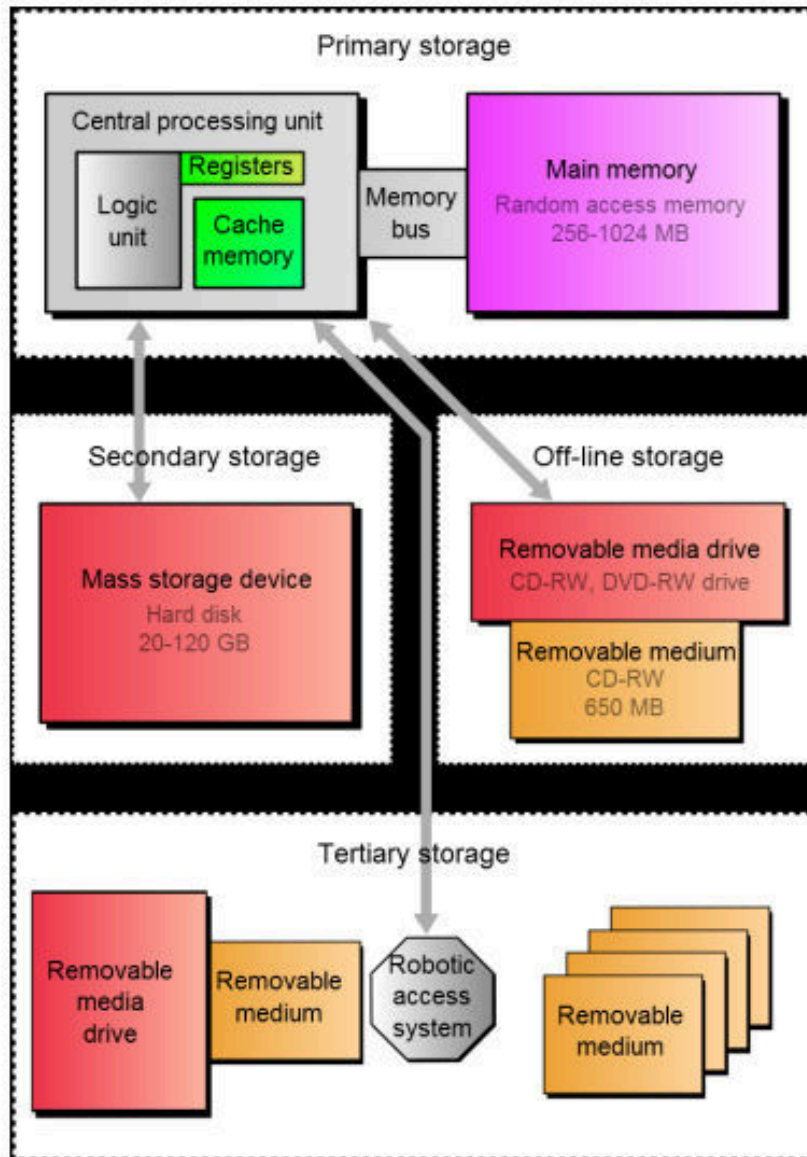


Figure 4.8: Different Types of Storage [8].

Above figure 4.8 represents the different types of storage's in the computer world. NAS is considered to be in the "Secondary storage" which also includes all types of hard disks. here are various types of secondary data storage options accessible today. The most widespread one is the magnetic disk, which you can find in almost every PC and server. This kind of disk employs a magnetic coating to record data as tiny magnetic particles. However, the challenge with this storage is that it's a mechanical device, and although it may function well for a long time, there are no guarantees. To safeguard against potential data loss due to a magnetic disk failure, there's a solution known as RAID. Different RAID types are accessible, with the most common ones for home use being RAID 0 and RAID 1. Network-attached storage (NAS) is a type of hard disk storage, but it differs in that it's like an external server with its dedicated hardware and network connection. Typically, a NAS is set up to function primarily as a file server, equipped with tools for overseeing network shares, users, and their permissions [44].

IS IT POSSIBLE FOR RANSOMWARE TO COMPROMISE A NAS DRIVE?

The ZDNet ransomware attack stands out as one of the most notable cyberattacks in recent history, extending its reach to impact even local government offices in the United States. This malicious ransomware attack did not discriminate, targeting a wide array of devices including corporate supercomputers, personal computers, mobile phones, and even supposedly secure NAS devices. Consequently, the concise response to the question "can ransomware infect NAS" is affirmative.

Once one device within the network becomes infected, the zdnet ransomware has the capacity to spread its infection to other interconnected devices. The malware's modus operandi involves encrypting the data on affected computers, and the only means to regain access to this data is by paying a specified "ransom" fee. For those who are unable to meet this financial demand, the consequence is data loss and the potential compromise of other sensitive information.

CAN NAS DRIVES FALL VICTIM TO THE NEW RANSOMWARE?

It's not sufficient to just comprehend whether ransomware can infiltrate NAS. We must also recognize the potential consequences of a ransomware attack on our computer systems. Beyond the risk of losing crucial personal and sensitive data stored on our devices, the zdnet ransomware could open doors to threats from terrorist organizations and other entities seeking

to disrupt our communities. Data holds immense power, and those who gain access to our sensitive information can potentially wield it against us.

In scenarios where ransomware infiltrates NAS, users of NAS devices are typically provided with instructions on how to recover their lost data. Making the ransom payment may seem like a path to decrypt their computer files, but there's no guarantee that files will be fully restored even after payment. The delays in data recovery for both private and public institutions could significantly impact the services they offer to their customers, patrons, and constituents [45].

In our project, we utilize a NAS device primarily for file sharing rather than storing critical data. Moreover, all data sharing occurs within a secure environment. Even in the event of data loss due to a possible attack or network-related issue, the impact on our operations would be negligible.

4.3 HARDWARE SANDBOX

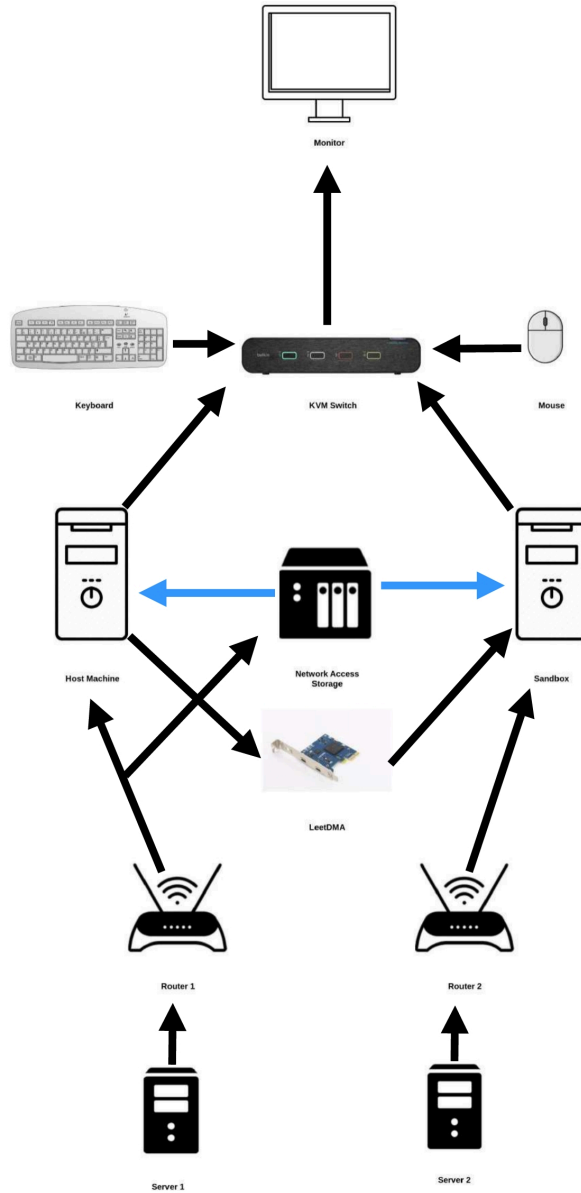


Figure 4.9: Hardware Sandbox.

Figure 4.9 illustrates the hardware-based sandbox setup, comprising two central processing units (CPUs): one dedicated to our host machine and the other to the hardware-based sandbox. Notably, our sandbox maintains no physical connection with the host machine. The hardware

sandbox features distinct processing and memory units reserved exclusively for malware analysis. It operates on a segregated network, distinct from our host and NAS device, which share a common network. This separation is integral to isolate the sandbox from the broader network within the lab.

In addition, we incorporate LeetDMA hardware, establishing a connection with the sandbox motherboard via a PCIe x1 interface. This hardware component is further linked to our host machine through a USB-C data port. Both the host and sandbox are interconnected via a KVM switch, and a single mouse and keyboard are also integrated into the KVM switch setup. The KVM switch, in turn, connects to the monitor to facilitate visualization.

Our choice of the Windows 10 operating system for the sandbox is motivated by the prevalence of malware targeting the Windows OS, a common platform in various real-world scenarios, such as hospitals, offices, and stock exchanges. In contrast, we utilize Ubuntu for the host machine.

Consider the scenario involving individuals Luca and Rosa, who operate within an office environment. If either of them receives a suspicious email containing a link or executable file, they transmit this file or link to the sandbox operator, depicted in Figure 4.10 below. The sandbox operator's role is to execute the provided executable file within the sandbox environment to assess its behavior. Following the execution, the sandbox operator scrutinizes network traffic, inspects system log files, and conducts a physical memory dump. For the physical memory dump process, we employ the pcileech tool, designed to be compatible with the LeetDMA device, as elaborated earlier, and integrated within the sandbox. This configuration allows us to extract the system's memory after executing the malware. Upon conducting a comprehensive analysis, if the sandbox operator deems the provided executable or link non-malicious, they issue a green signal, permitting Luca and Rosa to proceed with opening or executing the file on their respective host systems.

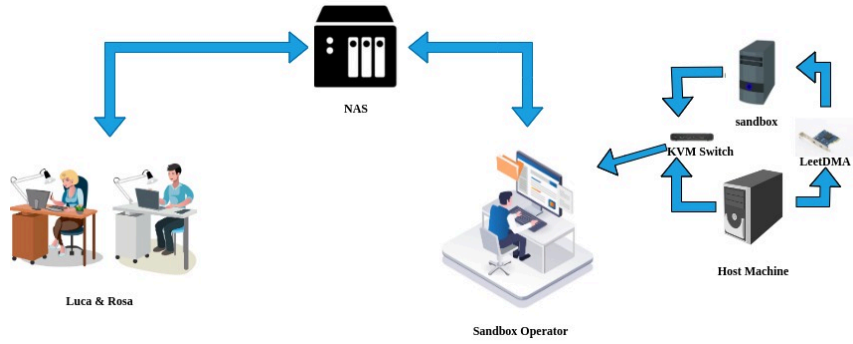


Figure 4.10: Working of Sandbox.

5

Database and Results

5.1 MALWARE DATABASE

In order to assess the functionality of our sandbox environment, it is imperative to carry out tests involving the execution of malware and observe its behavior. To do so, we must first acquire contemporary datasets of malicious software. For our experimental purposes, we have accessed a selection of malware samples from MalwareBazaar, a platform where IT security researchers continually update their findings of new malware samples. Researchers can submit these samples to the community after confirming their malicious nature. This practice ensures that the malware executed in our experiments is indeed confirmed as such.

It is worth noting that all the malware uploaded to MalwareBazaar by IT security researchers are verified instances of malicious software. Furthermore, the malware submitted to MalwareBazaar is typically novel or newly discovered by these researchers. MalwareBazaar permits both commercial and non-commercial utilization of their data, making it a highly suitable resource for our malware analysis endeavors.

To locate recently identified malware samples on MalwareBazaar, one can establish an alert as follows:

- **Tag:** In the context of malware analysis, a "tag" is a descriptive label or identifier assigned to a specific piece of malware or a related attribute. Tags help researchers and analysts

categorize and organize malware samples based on their characteristics or behavior.

- **Signature:** A "signature" is a specific pattern, code, or unique characteristic that is used to identify and detect malware. Signatures are like fingerprints for malware; they are created by security experts to recognize and block known malicious code.
- **YARA Rule:** YARA is a tool used in malware analysis to define and identify patterns or characteristics within files or data. A "YARA rule" is a set of criteria or conditions specified in the YARA language that helps analysts search for and classify malware based on those criteria.
- **ClamAV Signature:** ClamAV is an open-source antivirus software. A "ClamAV signature" is a specific pattern or rule used by ClamAV to detect and quarantine known malware. These signatures are regularly updated to stay current with emerging threats.
- **Vendor Detection:** In the realm of cybersecurity, "vendor detection" refers to the ability of security software or solutions provided by various cybersecurity companies (vendors) to identify and flag potential threats or malware. Each vendor may have its own unique detection methods and databases for recognizing malicious software.

The figure provided below illustrates the daily count of distinct malware samples shared on MalwareBazaar over a span of 30 days.

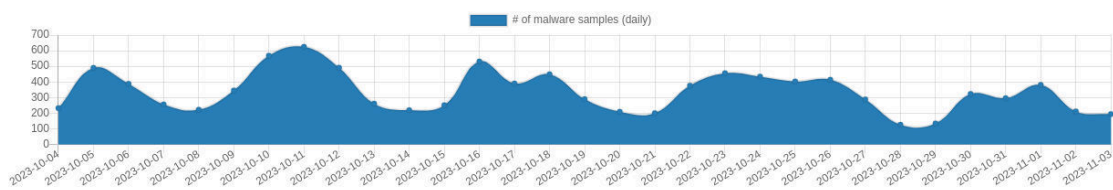


Figure 5.1: Malware samples shared in last 30 days [9].

Malware on MalwareBazaar is categorized based on its behavior by IT security researchers, enabling us to identify and understand the behavioral characteristics of different malware types. The following figure presents a pie chart representing the distribution of malware families and associated tags.

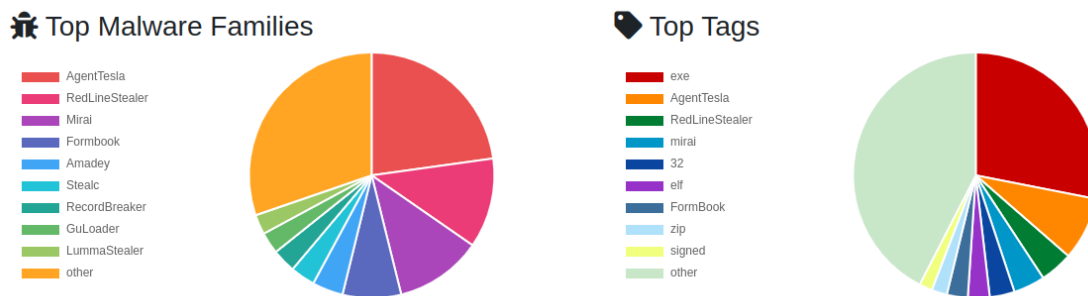


Figure 5.2: Malware families and their tags

5.1.1 MALWARE'S SHA-256

For our project, we utilized a total of 8 malware samples, all of which were sourced from MalwareBazaar. It's worth noting that malware samples can occasionally be removed from MalwareBazaar, but you can still verify and obtain additional information about these samples by checking them on VirusTotal. Below are the SHA-256 hashes for these samples, and for more comprehensive details, you can refer to VirusTotal.

- 1dec94b96of40e3a95e4aaa81f5783c221e4792d429701701052a03686aeb2cd
- 5c55ecc6f12aa60aco2540ebb2af7b778od148d76ao7ad27bfadod4f3bc1ao67
- 9fde5aer1eb2789887b7513a95obd2fb41f5b44d6eco756ao81ccfa7d9b4d63fd
- 94e2aab4ao36f8788be69af92be5568812541993f735732ecofodoodec17126b
- 3o7fo12f74f209e4b837140o455ee296585a6d6aod4c2195df2186ecafoacd79
- bec11b766972a7bb426o3a2e6f1452cc5a769779586b1712oef4299dbo6fboo
- e5615a6c9o478a371o4od8f7d4721e183b5efccoboe6ce64b7ab4ee1do4bfobe
- f58fea7363o83d8ad73358f871c6483ab17d8o4ce34eb6558c62b335oad368df

Each of these samples was individually executed within a controlled sandbox environment. Subsequently, the network traffic generated was captured using Wireshark and subjected to analysis through VirusTotal and BrimSecurity tools. Moreover, the physical memory was extracted through the utilization of LeetDMA hardware, a compatible solution with the PCILeech tool [46]. Lastly, an analysis of the sandbox's log files was conducted to complete the comprehensive examination.

5.2 ANALYSIS OF MEMORY DUMPS

We initially acquired a pristine memory dump from a clean, malware-free system, serving as a baseline reference. This reference dump will be employed for comparison with memory dumps from infected systems, facilitating the rapid identification of alterations induced by the malware. Additionally, we will assess the changes introduced by the same malware in a controlled virtual sandbox environment. During the analysis, we will examine the following Volatility tool plugins to detect system modifications introduced by malware:

- **Pslist:** Lists running processes, allowing you to identify any malicious processes created by the malware.
- **Psscan:** Scans for terminated or hidden processes, which may include remnants of malware activity.
- **Filescan:** Lists open files, which can reveal files accessed or modified by the malware.
- **Malfind:** Scans for injected code and suspicious memory regions, making it useful for detecting process injection by malware.
- **Dlllist:** Lists loaded DLLs, helping you identify any malicious or injected DLLs.
- **Driverirp:** Focuses on driver-related information, which can be important for identifying driver-based malware.
- **Windows.privileges:** Lists processes with privileges.
- **Windows.cmdline:** It is used to extract and display the command line arguments associated with processes in a memory dump of a Windows system.
- **Handles:** Provides information about file and registry handles, helping you identify suspicious file or registry access.

5.2.1 PSLIST AND PSSCAN

First, we will examine the initial or clean system process lists using `pslist` and `psscan`. This will provide us with a baseline of the system's normal state. These baselines will help us detect any alterations or the execution of malicious software within the system. We will also inspect

the hash of any malware once it has been executed. The figure 5.3 illustrates the results of pscan and pslist before any malware was executed.

Figure 5.3: Pscan and Pslist before execution.

Upon executing the malware identified by the hash `1dec94b96f40e3a95e4aa81f5783c221e4792d429701701052a03686aeb2cd`, it becomes evident that the malware has been successfully executed within the system.

Figure 5.4: Hardware sandbox changes after execution.

| | | | | | | | | | | | | | | | | | |
|---|-----|---------|---------|---|------|----------------------------|-----|------|------|----------------|----------------|---|---|---|------|----------------------------|-----|
| 1 | cmd | cmd.exe | cmd.exe | 1 | True | 2023-05-22 19:37:56.000000 | N/A | 1892 | 4036 | 1dec94b960f40e | 0xb6856d8b8880 | 1 | - | 1 | True | 2023-05-22 19:37:56.000000 | N/A |
| 1 | cmd | cmd.exe | cmd.exe | 2 | True | 2023-05-22 19:37:56.000000 | N/A | 3792 | 1892 | v6742880.exe | 0xb6856d8b8880 | 1 | - | 1 | True | 2023-05-22 19:37:57.000000 | N/A |
| 1 | cmd | cmd.exe | cmd.exe | 3 | True | 2023-05-22 19:37:56.000000 | N/A | 1592 | 3792 | v1974809.exe | 0xb6856d7f1340 | 1 | - | 1 | True | 2023-05-22 19:37:57.000000 | N/A |
| 1 | cmd | cmd.exe | cmd.exe | 4 | True | 2023-05-22 19:37:56.000000 | N/A | 1100 | 4036 | 1dec94b960f40e | 0xb6856cf4d340 | 1 | - | 1 | True | 2023-05-22 19:38:06.000000 | N/A |
| 1 | cmd | cmd.exe | cmd.exe | 5 | True | 2023-05-22 19:37:56.000000 | N/A | 5688 | 1100 | v6742880.exe | 0xb6856db04080 | 1 | - | 1 | True | 2023-05-22 19:38:06.000000 | N/A |
| 1 | cmd | cmd.exe | cmd.exe | 6 | True | 2023-05-22 19:38:06.000000 | N/A | 5604 | 5688 | v1974809.exe | 0xb6856d8b8880 | 1 | - | 1 | True | 2023-05-22 19:38:06.000000 | N/A |
| 1 | cmd | cmd.exe | cmd.exe | 7 | True | 2023-05-22 19:38:06.000000 | N/A | 5900 | 5604 | b4574510.exe | 0xb6856c7eb300 | 4 | - | 1 | True | 2023-05-22 19:38:06.000000 | N/A |
| 1 | cmd | cmd.exe | cmd.exe | 8 | True | 2023-05-22 19:38:06.000000 | N/A | 4400 | 1592 | b4574510.exe | 0xb6856d89e340 | 4 | - | 1 | True | 2023-05-22 19:38:06.000000 | N/A |

Figure 5.5: Virtual sandbox changes after execution.

In the figure 5.4, It's worth noting that following the execution of the malware, two additional executables, namely v642880.exe and v1974809.exe, emerged in the system. These executables were not present in our system prior to the malware's execution. The emergence of v642880.exe and v1974809.exe following malware execution likely indicates that the initial malware was designed to drop or download additional files as part of its malicious activity, potentially for persistence, expansion, or further malicious actions. This behavior is common in malware to achieve specific objectives.

In Figure 5.5, we observe a consistent pattern with the executed executables, v642880.exe and v6742880.exe. However, it's noteworthy that the malware executed a distinct executable, v6742880.exe, in the virtual environment instead of v642880.exe. This variation in behavior can be attributed to environment detection, dynamic adaptation, and potential configuration disparities between the hardware and virtual setups, reflecting common tactics employed by malware authors to obfuscate analysis and cater to specific environmental conditions. Some factors also involved are:

- **Environment Sensing:** Some malware is designed to detect the specific environment it is running in, such as a physical hardware system versus a virtual machine. Malware authors may build in mechanisms to identify virtualized environments to avoid analysis, as analysts often use virtual machines to safely investigate malware. The malware may behave differently based on this detection.
- **Dynamic Behavior:** Malware may exhibit dynamic behavior, adapting its actions based on its environment. It might have been programmed to choose different executable names, or the environment-specific payloads may have varied filenames.
- **Payload Delivery:** The malware in the two environments may have fetched different payloads from remote servers or dropped different files onto the system. The choice of payload can depend on the environment or may be randomly determined by the malware itself.

- **Configuration Differences:** The hardware and virtual environments might have subtle differences in their configuration, such as file paths or system properties, which could lead to variations in the behavior of the malware.
- **Anti-Analysis Techniques:** Malware authors often employ anti-analysis techniques to make it more challenging for security researchers to analyze their code. Running different executables in different environments can be one such technique.

5.2.2 FILESCAN

Algorithm 5.1 Set Intersection Pseudocode

Input:

Hb - Set of elements for *Hb*

Ha - Set of elements for *Ha*

Vb - Set of elements for *Vb*

Va - Set of elements for *Va*

// Calculate *Hd* by finding the intersection of *Hb* and *Ha*

$Hd \leftarrow Hb \cap Ha$

// Calculate *Vd* by finding the intersection of *Vb* and *Va*

$Vd \leftarrow Vb \cap Va$

// Calculate *Final_output* by finding the intersection of *Vd* and *Hd*

$Final_output \leftarrow Vd \cap Hd$

Output:

Final_output - Set of elements that are common to both *Vd* and *Hd*

In our algorithm, "Hb" represents the files we collected before the malware infection, "Ha" represents the files we collected after the malware infection, "Vb" represents the set of files or elements before the virtual memory dump, and "Va" represents the files we collected after running the malware in a virtual environment. We compared these sets of files to identify the differences.

After applying the algorithm mentioned earlier, when we look at Figure 5.6, we will observe the following results.

| Name | Type | Size | Values |
|---|------|------|-----------------------------|
| Before ₁ (H _b) | set | 3109 | \Windows\System32\.... |
| Before ₂ (V _b) | set | 1850 | \Users\asad\.... |
| After ₁ (H _a) | set | 2502 | \ProgramData\Mozilla-.... |
| After ₂ (V _a) | set | 2246 | \Windows\System32\.... |
| Difference ₁ (H _d) | set | 309 | \Users\asad\AppData\.... |
| Difference ₂ (V _d) | set | 788 | \Windows\Microsoft.NET\.... |
| Difference combined | set | 16 | \Windows\System32\drive.... |

Table 5.1: Results from the Algorithm.

Upon reviewing "difference₁ (H_d)," it becomes apparent that the algorithm yielded a total of 309 files. This indicates that there is a disparity in the files between the pre-malware execution and post-malware execution states within the hardware sandbox, with a variance of 309 files.

| | | |
|-----|-----|---|
| str | 32 | \\Windows\\System32\\storageiml.dll |
| str | 29 | \\Windows\\System32\\vmltoml.dll |
| str | 29 | \\Windows\\System32\\winhttp.dll |
| str | 49 | \\Windows\\SoftwareDistribution\\ReportingEvents.log |
| str | 28 | \\Windows\\System32\\C_1251,HL5 |
| str | 37 | \\Windows\\System32\\drivers\\USBXHCI.SYS |
| str | 92 | \\Users\\asadj\\AppData\\Local\\Microsoft\\Edge\\User Data\\Default\\Code Cache ... |
| str | 36 | \\Windows\\System32\\LicenseManager.dll |
| str | 75 | \\Users\\asadj\\AppData\\Local\\Google\\Chrome\\User Data\\Default\\DownCache\\d ... |
| str | 103 | \\Users\\asadj\\AppData\\Local\\Packages\\38833FF26BA1D.UnigramPreview_g9c9v ... |
| str | 102 | \\Program Files\\WindowsApps\\Microsoft.SkypeApp_15.97.3204.0_x64__kzf8qx ... |
| str | 39 | \\Windows\\System32\\spool\\drivers\\x64\\PCC |
| str | 30 | \\Windows\\System32\\bthprops.cpl |
| str | 79 | \\Users\\asadj\\AppData\\Local\\Microsoft\\OneDrive\\21.220.1024.0005\\QEt5Qm1M ... |
| str | 84 | \\Users\\asadj\\Desktop\\1dec94b960f40e3a95e4aaa81f5783c221e4792d4297017010 ... |
| str | 37 | \\Windows\\System32\\drivers\\USBXHCI.SYS |
| str | 101 | \\ProgramData\\Microsoft\\Windows Defender\\Scans\\mpcache-DC74DC1470C654DD ... |
| str | 90 | \\Users\\asadj\\AppData\\Local\\Microsoft\\Edge\\User Data\\Default\\Sessions\\Ap ... |
| str | 49 | \\Windows\\System32\\winevt\\Logs\\HardwareEvents.evtx |
| str | 91 | \\Windows\\System32\\config\\systemprofile\\AppData\\LocalLow\\Microsoft\\Cryp ... |
| str | 10 | \\TaskIndex |
| str | 78 | \\Users\\asadj\\AppData\\Local\\Microsoft\\Edge\\User Data\\Default\\Extension R ... |
| str | 85 | \\Windows\\System32\\config\\TxR\\{53b39e3d-18c4-11ea-a811-000d3aa4692b}.Tx ... |
| str | 102 | \\Program Files\\WindowsApps\\Microsoft.VCLibs.140.00_14.0.30704.0_x64__8 ... |
| str | 37 | \\Winsock2\\CatalogChangeListener-274-0 |
| str | 88 | \\Windows\\SystemApps\\Microsoft.Windows.SecHealthUI_cw5n1h2txyewy\\pris\\r ... |
| str | 89 | \\Users\\asadj\\AppData\\Local\\Microsoft\\OneDrive\\23.091.0430.0001\\OneDrive ... |
| str | 133 | \\ProgramData\\Microsoft\\Windows\\AppRepository\\Packages\\Microsoft.UI.Xam ... |

Figure 5.6: Hb and Hd comparison.

Upon examining the disparities between Hb and Hd, we can discern the alterations introduced to the file system by the malware. In Figure 5.6, we can observe a portion of these changes. For instance, after the execution of the malware with the hash "1dec94b960f40e3a95e4-aaa81f5783c221e4792d4297017010" we witness a sequence of subsequent file executions, including the appearance of seemingly random Chinese characters in "str 8." Additionally, at "str 40," we encounter the file "\LOCAL\mojo.572.2064.97017010" among others. As previously noted, these alterations amount to a total of 309 files modified by the malware.

```
str 84  \Users\asad\Desktop\1dec94b960f40e3a95e4aaa81f5783c221e4792d4297017010 ...
str 37  \Windows\System32\drivers\USBXHCI.SYS
str 101 \ProgramData\Microsoft\Windows Defender\Scans\mpcache-DC74DC1470C654DD ...
str 90  \Users\asad\AppData\Local\Microsoft\Edge\User Data\Default\Sessions\Ap ...
str 49  \Windows\System32\winevt\Logs\HardwareEvents.evtx
str 91  \Windows\System32\config\systemprofile\AppData\LocalLow\Microsoft\Cryp ...
str 10  \TaskIndex
str 78  \Users\asad\AppData\Local\Microsoft\Edge\User Data\Default\Extension R ...
str 85  \Windows\System32\config\TxR\{53b39e3d-18c4-11ea-a811-000d3aa4692b}.Tx ...
str 102 \Program Files\WindowsApps\Microsoft.VCLibs.140.00_14.0.30704.0_x64_8 ...
str 37  \Winsock2\CatalogChangeListener-274-0
str 88  \Windows\SystemApps\Microsoft.Windows.SecHealthUI_cw5n1h2txyewy\pris\r ...
str 89  \Users\asad\AppData\Local\Microsoft\OneDrive\23.091.0430.0001\OneDrive ...
str 133 \ProgramData\Microsoft\Windows\AppRepository\Packages\Microsoft.UI.Xam ...
```

Figure 5.7: Va files section of the algorithm.

In Figure 5.7, "Va" denotes the section containing virtual sandbox files after the malware's execution. As observed in table 5.1, this section initially had 1850 files, but post-malware execution, it increased to 2246. When we compare the files in "Va" with those in "Vb," representing the virtual files section before malware execution, we identify "Vd," denoting the second set of differences. Notably, "Vd" reveals 788 discrepancies, indicating that the malware introduced 788 file changes that were not present in "Vb."

Interestingly, when we examine the overlap between "Vd" and "Hd," which represents the final output of our algorithm, we find only 16 files in common. Several explanations could account for this observation. It's possible that the malware adapts its behavior depending on whether it's in a hardware sandbox or a virtual sandbox. Alternatively, the malware might detect the environment and adjust its actions accordingly. Another possibility is that in the virtual environment, the malware generates seemingly benign files, potentially employing obfuscation techniques to mislead researchers. However, when executed in a real system, it might initiate harmful files meant to compromise the system's security.

5.2.3 MALFIND

```

59 3512 b4574510.exe 0x7fe00000 0x7fe4ffff VadS PAGE_EXECUTE_READWRITE 2 1 Disabled
60 ec ff ff 04 00 00 00 .....
61 01 00 00 00 02 0e 03 .....
62 1c 00 00 00 68 01 d9 07 ....h...
63 16 00 00 00 34 27 fb 69 ....4'.i
64 00 10 fa 09 e0 b2 0b 6a .....l..j
65 cc 1f fa 09 f0 53 09 00 ....l.S...
66 00 00 00 00 40 80 e0 7f ....@...
67 00 00 e0 7f 80 80 e0 7f .....
e0 7f 00 80 e0 7f 80 80 e0 7f
68 4940 b4574510.exe 0x7fe00000 0x7fe4ffff VadS PAGE_EXECUTE_READWRITE 2 1 Disabled
69 ec ff ff 04 00 00 00 .....
70 01 00 00 00 02 0e 03 .....
71 1c 00 00 00 68 01 d9 07 ....h...
72 16 00 00 00 34 27 fb 69 ....4'.i
73 00 10 fa 09 e0 b2 0b 6a .....l..j
74 cc 1f fa 09 f0 53 09 00 ....l.S...
75 00 00 00 00 40 80 e0 7f ....@...
76 00 00 e0 7f 80 80 e0 7f .....
e0 7f 00 80 e0 7f 80 80 e0 7f
-----
PID Process Start VPN End VPN Tag Protection CommitCharge PrivateMemory File output Hexdump Disasm
2388 shdServ.exe 0x530000 0x530fff VadS PAGE_EXECUTE_READWRITE 1 1 Disabled
00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 .....
00 00 53 00 00 00 00 00 ..S.....
00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 .....
5900 b4574510.exe 0x780000 0x78ffff VadS PAGE_EXECUTE_READWRITE 1 1 Disabled
20 f6 f3 31 36 20 00 01 ...16...
ee ff ee ff 02 00 00 00 .....
a4 00 78 00 a4 00 78 00 ..X...X.
00 00 78 00 00 00 78 00 ..X...X.
0f 00 00 00 a0 04 78 00 .....X.
00 f0 78 0e 00 00 00 .....X...
01 00 00 00 00 00 00 00 .....
f0 0f 78 00 f0 0f 78 00 ..X...X.
2d f6 f3 31 36 20 00 01 ee ff ee ff 02 00 00 a4 00 78 00 a4 00 78 00 00 78 00 00 78 00 0f 00 00 a8 04 78 00 00 01 00 00 00 00
00 00 f0 0f 78 00 f0 0f 78 00

```

Figure 5.8: Code injection by the malware sample.

In the figure 5.8, the image on the top represents the hardware sandbox, while the one on the bottom corresponds to the virtual sandbox. In both images, we can observe the presence of b4574510.exe, indicating a process that has been subjected to code injection. This injection is associated with the execution of 1dec94b96f40e3a95e4aaa81f5783c221e4792d42970170105-2a03686aeb2cd.exe. The existence of the b4574510.exe process in the memory dump strongly implies that the malware has effectively injected its code into the active system.

ANALYSIS AND COMPARISON

- **Process Name:** The presence of the b4574510.exe process in both memory dumps indicates that the same process is injected into both the hardware and virtual machines. This process is likely the result of the execution of the malware sample with the specified hash.
- **Virtual Address Descriptor (Vad):** In both datasets, the Vad descriptor is identical and marked as VadS. This suggests that the same memory region or section is involved in both instances. This consistency is a strong indicator of similarity in memory structures.
- **Protection:** The memory region is marked with PAGE_EXECUTE_READWRITE protection in both cases. This protection level allows both execution and modification of memory contents. Such permissions are often associated with code injection and are a significant indicator of potential malicious activity.

- **Commit Charge and Private Memory:** While there are slight differences in the values for Commit Charge and Private Memory between the hardware and virtual machine datasets, these discrepancies may not be of substantial significance in this analysis. They could be attributed to variations in system configurations.
- **Disabled Status:** In both cases, the b4574510.exe process is marked as "Disabled," indicating that it is not currently active or running. The status suggests that this process is not in use at the time of the memory dumps.

The findings reveal that the same b4574510.exe process, associated with the malware sample's hash, is present in both the hardware and virtual machine memory dumps. The shared characteristics, including the identical Vad descriptor and PAGE_EXECUTE_READWRITE protection, are compelling evidence of a common memory structure or injected code.

The PAGE_EXECUTE_READWRITE protection and the process's "Disabled" status raise concerns about potential code injection and unauthorized memory modifications. It is crucial to conduct further in-depth analysis to determine the specific behavior and purpose of this injected process, as well as to assess any potential threats it may pose to the system.

5.2.4 DLLLIST

Following the execution of the malware sample, 1dec94b96of40e3a95e4aaa81f5783c221e4792-d429701701052a03686aeb2cd.exe, in both virtual and hardware sandboxes, and subsequent analysis of the dlllists, the following results were obtained.

HARDWARE SANDBOX (EXECUTION ON 2023-05-15)

- **1dec94b96of40e.exe:** Executes the malware sample located in a user's Downloads folder. Loads system DLLs, including wow64.dll, wow64win.dll, and wow64cpu.dll.
- **v6742880.exe:** Executes the malware sample located in a temporary folder (IXP003.TMP). Loads wow64.dll, wow64win.dll, and wow64cpu.dll.
- **v1974809.exe:** Executes the malware sample located in a temporary folder (IXP004.TMP). Loads wow64.dll, wow64win.dll, and wow64cpu.dll.
- **b4574510.exe:** Executes the malware sample located in a temporary folder (IXP005.TMP). Loads wow64.dll, wow64win.dll, and wow64cpu.dll.

VIRTUAL SANDBOX (EXECUTION ON 2023-05-22)

- **rdec94b96of40e.exe:** Loads wow64cpu.dll, ntdll.dll, wow64.dll, and wow64win.dll. Executes the malware sample located in a temporary folder.
- **v6742880.exe:** Executes the malware sample located in a temporary folder (IXP003.TMP). Loads ntdll.dll, wow64.dll, wow64win.dll, and wow64cpu.dll.
- **vi974809.exe:** Executes the malware sample located in a temporary folder (IXP004.TMP). Loads ntdll.dll, wow64.dll, wow64win.dll, and wow64cpu.dll.
- **b4574510.exe:** Executes the malware sample located in a temporary folder (IXP005.TMP). Loads ntdll.dll, wow64.dll, wow64win.dll, and wow64cpu.dll.

RED FLAGS: In both the hardware and virtual sandboxes, the malware demonstrates a consistent pattern of behavior that raises red flags and poses a potential threat to the system's security. This behavior includes the execution of the malware sample and, more critically, the loading of essential system DLLs.

What's particularly concerning is the consistent loading of specific system DLLs, namely wow64.dll, wow64win.dll, and wow64cpu.dll. These DLLs serve as fundamental components in the way applications interact with the Windows operating system. When a piece of malware manipulates or loads these DLLs, it can potentially gain unauthorized access to system functions and resources.

Such actions can lead to a range of threats, including but not limited to:

- **Privilege Escalation:** Malicious software can exploit vulnerabilities within these DLLs to elevate its privileges within the system, gaining more control and access to sensitive data.
- **System Manipulation:** By tampering with these critical system components, malware can manipulate or disrupt system operations, leading to instability and poor performance.
- **Data Theft:** The malware may exploit the loaded DLLs to access, exfiltrate, or manipulate user data, potentially leading to data breaches or identity theft.

- **Remote Control:** Loading these DLLs might allow the malware to establish a backdoor or remote control over the infected system, giving cybercriminals unauthorized access to the machine.

5.2.5 DRIVERIRP

| Offset-Known | Exception | Driver | Name | Service Key | Alternative Name |
|----------------|-----------|--------|------|-----------------|------------------|
| 0xa684c87c0c80 | True | RAW | | \FileSystem\RAW | |
| 0xa684c8952ce0 | False | | N/A | N/A | |

| Offset-Known | Exception | Driver | Name | Service Key | Alternative Name |
|----------------|-----------|--------|------|-----------------|------------------|
| 0xb30645085280 | False | | N/A | N/A | |
| 0xb306454058c8 | False | N/A | N/A | N/A | |
| 0xb30645435918 | False | N/A | N/A | N/A | |
| 0xb306459cda70 | True | RAW | | \FileSystem\RAW | |
| 0xb30645f550c0 | False | | N/A | N/A | |
| 0xf806713b5a6a | False | N/A | N/A | | |

Figure 5.9: Hardware offset difference before and after malware execution.

HARDWARE: In our research, we examined the computer’s memory using ‘windows.driver module.Driver-Module.’ We did this before and after running malware on a protected system, and the results are visible in Figure 5.9. The part that’s highlighted in the figure represents what the malware did in the computer’s memory.

Our analysis revealed that there were noticeable differences in the list of system drivers before and after the malware event. It’s worth noting that malware often focuses on these drivers because they are essential for the computer’s operation. Malware may try to change them to gain unauthorized access or control over the system.

The data we collected showed that after the malware was executed, some drivers were either added or changed. This is a noteworthy observation because the new drivers didn’t have any built-in safeguards, and the existing drivers lost their protective measures. This alteration in

the driver landscape may raise concerns, suggesting the possibility of the malware making modifications or adding its own drivers.

This difference becomes even more apparent when we compare the number of drivers present before the malware (only two) to the number after the malware (more than two, as highlighted), also the fact that we couldn't find the offsets '0xb306454058c8,' '0xb30645435918,' and '0xf806713b5a6a' in the 'DriverRip' module of Volatility, especially in a memory dump taken after malware execution, raises concerns. It suggests that the malware may have influenced or altered the system's drivers, which can impact the behavior of the system.

While the 'DriverModule' output indicates the presence of certain drivers with offset values, the absence of these offsets in the 'DriverRip' results may signify that the malware manipulated the system drivers or their associated information, making them undetectable or invisible to certain analysis tools like 'DriverRip.'

This scenario is consistent with the idea that malware often seeks to modify drivers to evade detection and operate stealthily. The absence of these offsets in 'DriverRip' could be a deliberate attempt by the malware to hide its presence and actions.

Here are a few possible explanations for this situation:

- **Malware Modification:** The malware may have loaded a driver or modified an existing one during its execution, causing the offset 0xb306454058c8 to appear in the `DriverModule` output.
- **Rootkit Activity:** Some types of malware, especially rootkits, are designed to hide their presence and tampering with the system. They might manipulate the memory analysis tools, making it challenging to detect their activities.

| Offset | Known Exception | Driver Name | Service Key | Alternative Name |
|----------------|-----------------|-------------|-------------|------------------|
| 0x800654393a58 | False | N/A | N/A | N/A |
| 0xb60564710cf0 | True | RAW | | \FileSystem\RAW |

| Offset | Known Exception | Driver Name | Service Key | Alternative Name |
|----------------|-----------------|-------------|-------------|------------------|
| 0xae0d84514cf0 | True | RAW | | \FileSystem\RAW |
| 0xae0d84d17acf | False | N/A | N/A | N/A |

Figure 5.10: Virtual offset difference before and after malware execution.

VIRTUAL: When we compare Figure 5.9 and Figure 5.10, we notice differences in the offset values. These differences can be explained by the nature of the environment where the memory dumps were taken. In virtualized settings, memory dumps might not capture the same level of detail or exhibit the same offset variations as in physical hardware.

It's important to understand that the way malware behaves can be influenced by the specific environment it operates in. Different types of malware may exhibit varying behaviors, and how they interact with the system's drivers can be influenced by whether the system is running on physical hardware or within a virtualized environment.

Furthermore, we also observed a similar behavior with the offset '0x800654393a58' as it was not found in the list of offsets in the 'DriverRip' module. In general, when we compare the memory dumps, it appears that the malware did not introduce more offsets in the 'DriverModule' of Volatility in the virtual environment compared to the hardware-based environment.

5.2.6 WINDOWS PRIVILEGES

| | | | | |
|-----------|----------------|----|---|--|
| 5955 768 | 1dec94b960f40e | 2 | SeCreateTokenPrivilege | Create a token object |
| 5956 768 | 1dec94b960f40e | 3 | SeAssignPrimaryTokenPrivilege | Replace a process-level token |
| 5957 768 | 1dec94b960f40e | 4 | SeLockMemoryPrivilege | Lock pages in memory |
| 5958 768 | 1dec94b960f40e | 5 | SeIncreaseQuotaPrivilege | Increase quotas |
| 5959 768 | 1dec94b960f40e | 6 | SeMachineAccountPrivilege | Add workstations to the domain |
| 5960 768 | 1dec94b960f40e | 7 | SeTcbPrivilege | Act as part of the operating system |
| 5961 768 | 1dec94b960f40e | 8 | SeSecurityPrivilege | Manage auditing and security log |
| 5962 768 | 1dec94b960f40e | 9 | SeTakeOwnershipPrivilege | Take ownership of files/objects |
| 5963 768 | 1dec94b960f40e | 10 | SeLoadDriverPrivilege | Load and unload device drivers |
| 5964 768 | 1dec94b960f40e | 11 | SeSystemProfilePrivilege | Profile system performance |
| 5965 768 | 1dec94b960f40e | 12 | SeSystemTimePrivilege | Change the system time |
| 5966 768 | 1dec94b960f40e | 13 | SeProfileSingleProcessPrivilege | Profile a single process |
| 5967 768 | 1dec94b960f40e | 14 | SeIncreaseBasePriorityPrivilege | Increase scheduling priority |
| 5968 768 | 1dec94b960f40e | 15 | SeCreatePagefilePrivilege | Create a pagefile |
| 5969 768 | 1dec94b960f40e | 16 | SeCreatePermanentPrivilege | Create permanent shared objects |
| 5970 768 | 1dec94b960f40e | 17 | SeBackupPrivilege | Backup files and directories |
| 5971 768 | 1dec94b960f40e | 18 | SeRestorePrivilege | Restore files and directories |
| 5972 768 | 1dec94b960f40e | 19 | SeShutdownPrivilege | Present Shut down the system |
| 5973 768 | 1dec94b960f40e | 20 | SeDebugPrivilege | Debug programs |
| 5974 768 | 1dec94b960f40e | 21 | SeAuditPrivilege | Generate security audits |
| 5975 768 | 1dec94b960f40e | 22 | SeSystemEnvironmentPrivilege | Edit firmware environment values |
| 5976 768 | 1dec94b960f40e | 23 | SeChangeNotifyPrivilege | Present,Enabled,Default Receive notifications of changes to files or directories |
| 5977 768 | 1dec94b960f40e | 24 | SeRemoteShutdownPrivilege | Force shutdown from a remote system |
| 5978 768 | 1dec94b960f40e | 25 | SeUndockPrivilege | Present Remove computer from docking station |
| 5979 768 | 1dec94b960f40e | 26 | SeSyncAgentPrivilege | Present Synchronizing directory service data |
| 5980 768 | 1dec94b960f40e | 27 | SeEnableDelegationPrivilege | Enable user accounts to be trusted for delegation |
| 5981 768 | 1dec94b960f40e | 28 | SeManageVolumePrivilege | Manage the files on a volume |
| 5982 768 | 1dec94b960f40e | 29 | SeImpersonatePrivilege | Impersonate a client after authentication |
| 5983 768 | 1dec94b960f40e | 30 | SeCreateGlobalPrivilege | Default Create global objects |
| 5984 768 | 1dec94b960f40e | 31 | SeTrustedCredManAccessPrivilege | Access Credential Manager as a trusted caller |
| 5985 768 | 1dec94b960f40e | 32 | SeRelabelPrivilege | Modify the mandatory integrity level of an object |
| 5986 768 | 1dec94b960f40e | 33 | SeIncreaseWorkingSetPrivilege | Present Allocate more memory for user applications |
| 5987 768 | 1dec94b960f40e | 34 | SeTimeZonePrivilege | Present Adjust the time zone of the computer's internal clock |
| 5988 768 | 1dec94b960f40e | 35 | SeCreateSymbolicLinkPrivilege | Required to create a symbolic link |
| 5989 768 | 1dec94b960f40e | 36 | SeDelegateSessionUserImpersonatePrivilege | Obtain an impersonation token for another user in the same session. |
| 5990 4276 | v6742880.exe | 2 | SeCreateTokenPrivilege | Create a token object |
| 5991 4276 | v6742880.exe | 3 | SeAssignPrimaryTokenPrivilege | Replace a process-level token |
| 5992 4276 | v6742880.exe | 4 | SeLockMemoryPrivilege | Lock pages in memory |
| 5993 4276 | v6742880.exe | 5 | SeIncreaseQuotaPrivilege | Increase quotas |
| 5994 4276 | v6742880.exe | 6 | SeMachineAccountPrivilege | Add workstations to the domain |
| 5995 4276 | v6742880.exe | 7 | SeTcbPrivilege | Act as part of the operating system |
| 5996 4276 | v6742880.exe | 8 | SeSecurityPrivilege | Manage auditing and security log |
| 5997 4276 | v6742880.exe | 9 | SeTakeOwnershipPrivilege | Take ownership of files/objects |
| 5998 4276 | v6742880.exe | 10 | SeLoadDriverPrivilege | Load and unload device drivers |
| 5999 4276 | v6742880.exe | 11 | SeSystemProfilePrivilege | Profile system performance |
| 6000 4276 | v6742880.exe | 12 | SeSystemTimePrivilege | Change the system time |
| 6006 3322 | v4374510.exe | 10 | SeTakeOwnershipPrivilege | Take ownership of files/objects |
| 6069 3512 | b4574510.exe | 11 | SeSystemProfilePrivilege | Profile system performance |
| 6070 3512 | b4574510.exe | 12 | SeSystemTimePrivilege | Change the system time |
| 6071 3512 | b4574510.exe | 13 | SeProfileSingleProcessPrivilege | Profile a single process |
| 6072 3512 | b4574510.exe | 14 | SeIncreaseBasePriorityPrivilege | Increase scheduling priority |
| 6073 3512 | b4574510.exe | 15 | SeCreatePagefilePrivilege | Create a pagefile |
| 6074 3512 | b4574510.exe | 16 | SeCreatePermanentPrivilege | Create permanent shared objects |
| 6075 3512 | b4574510.exe | 17 | SeBackupPrivilege | Backup files and directories |
| 6076 3512 | b4574510.exe | 18 | SeRestorePrivilege | Restore files and directories |
| 6077 3512 | b4574510.exe | 19 | SeShutdownPrivilege | Present Shut down the system |
| 6078 3512 | b4574510.exe | 20 | SeDebugPrivilege | Debug programs |
| 6079 3512 | b4574510.exe | 21 | SeAuditPrivilege | Generate security audits |
| 6080 3512 | b4574510.exe | 22 | SeSystemEnvironmentPrivilege | Edit firmware environment values |
| 6081 3512 | b4574510.exe | 23 | SeChangeNotifyPrivilege | Present,Enabled,Default Receive notifications of changes to files or directories |
| 6082 3512 | b4574510.exe | 24 | SeRemoteShutdownPrivilege | Force shutdown from a remote system |
| 6083 3512 | b4574510.exe | 25 | SeUndockPrivilege | Present Remove computer from docking station |
| 6084 3512 | b4574510.exe | 26 | SeSyncAgentPrivilege | Present Synchronizing directory service data |
| 6085 3512 | b4574510.exe | 27 | SeEnableDelegationPrivilege | Enable user accounts to be trusted for delegation |
| 6086 3512 | b4574510.exe | 28 | SeManageVolumePrivilege | Manage the files on a volume |
| 6087 3512 | b4574510.exe | 29 | SeImpersonatePrivilege | Impersonate a client after authentication |
| 6088 3512 | b4574510.exe | 30 | SeCreateGlobalPrivilege | Default Create global objects |
| 6089 3512 | b4574510.exe | 31 | SeTrustedCredManAccessPrivilege | Access Credential Manager as a trusted caller |
| 6090 3512 | b4574510.exe | 32 | SeRelabelPrivilege | Modify the mandatory integrity level of an object |
| 6091 3512 | b4574510.exe | 33 | SeIncreaseWorkingSetPrivilege | Present Allocate more memory for user applications |
| 6092 3512 | b4574510.exe | 34 | SeTimeZonePrivilege | Present Adjust the time zone of the computer's internal clock |
| 6093 3512 | b4574510.exe | 35 | SeCreateSymbolicLinkPrivilege | Required to create a symbolic link |
| 6094 3512 | b4574510.exe | 36 | SeDelegateSessionUserImpersonatePrivilege | Obtain an impersonation token for another user in the same session. |
| 6095 4940 | b4574510.exe | 2 | SeCreateTokenPrivilege | Create a token object |
| 6096 4940 | b4574510.exe | 3 | SeAssignPrimaryTokenPrivilege | Replace a process-level token |
| 6097 4940 | b4574510.exe | 4 | SeLockMemoryPrivilege | Lock pages in memory |
| 6098 4940 | b4574510.exe | 5 | SeIncreaseQuotaPrivilege | Present Increase quotas |
| 6099 4940 | b4574510.exe | 6 | SeMachineAccountPrivilege | Add workstations to the domain |
| 6100 4940 | b4574510.exe | 7 | SeTcbPrivilege | Act as part of the operating system |
| 6101 4940 | b4574510.exe | 8 | SeSecurityPrivilege | Present Manage auditing and security log |
| 6102 4940 | b4574510.exe | 9 | SeTakeOwnershipPrivilege | Present Take ownership of files/objects |
| 6103 4940 | b4574510.exe | 10 | SeLoadDriverPrivilege | Present Load and unload device drivers |
| 6104 4940 | b4574510.exe | 11 | SeSystemProfilePrivilege | Present Profile system performance |
| 6105 4940 | b4574510.exe | 12 | SeSystemTimePrivilege | Present Change the system time |
| 6106 4940 | b4574510.exe | 13 | SeProfileSingleProcessPrivilege | Present Profile a single process |
| 6107 4940 | b4574510.exe | 14 | SeIncreaseBasePriorityPrivilege | Present Increase scheduling priority |
| 6108 4940 | b4574510.exe | 15 | SeCreatePagefilePrivilege | Present Create a pagefile |
| 6109 4940 | b4574510.exe | 16 | SeCreatePermanentPrivilege | Present Create permanent shared objects |
| 6110 4940 | b4574510.exe | 17 | SeBackupPrivilege | Present Backup files and directories |
| 6111 4940 | b4574510.exe | 18 | SeRestorePrivilege | Present Restore files and directories |
| 6112 4940 | b4574510.exe | 19 | SeShutdownPrivilege | Present Shut down the system |
| 6113 4940 | b4574510.exe | 20 | SeDebugPrivilege | Present Debug programs |
| 6114 4940 | b4574510.exe | 21 | SeAuditPrivilege | Generate security audits |

Figure 5.11: System privileges granted to the malware.

In the presented data (Figure 5.11), it is evident that the malware application has sought numerous privileges. When the malware was executed, it initiated requests for specific privileges from the system, aiming to acquire enhanced capabilities and perform malicious actions. To comprehensively understand the implications of these privileges, we need to analyze their intended purposes and the potential harm that could arise if the malware were to be granted or able to request these privileges.

1. **SeLockMemoryPrivilege:**

- Allows a process to lock pages in memory.
- Potential risk: This privilege could be used to prevent the operating system from swapping the process's memory to disk, making it harder for the system to manage memory usage.

2. **SeIncreaseQuotaPrivilege:**

- Allows a process to increase its resource quotas.
- Potential risk: This privilege could be abused to consume additional system resources beyond the normal limits.

3. **SeMachineAccountPrivilege:**

- Allows a process to add workstations to the domain.
- Potential risk: If misused, this privilege could potentially allow the malware to join the system to a domain, gaining unauthorized access.

4. **SeTcbPrivilege (Act as part of the operating system):**

- Allows a process to act as part of the operating system.
- Potential risk: This privilege is highly sensitive and dangerous. It grants the process significant control over the system, and if exploited, it could lead to system compromise.

5. **SeSecurityPrivilege (Manage auditing and security log):**

- Allows a process to manage auditing and security logs.
- Potential risk: An attacker can manipulate the logs to hide their activities, making it difficult to detect and investigate malicious actions.

6. **SeTakeOwnershipPrivilege:**

- Allows a process to take ownership of files and objects.
- Potential risk: This privilege can be used to manipulate file and object ownership, making it harder to track changes and investigate unauthorized access.

7. **SeLoadDriverPrivilege:**

- Allows a process to load and unload device drivers.
- Potential risk: Loading unauthorized or malicious drivers can lead to system instability or compromise.

8. **SeShutdownPrivilege:**

- Allows a process to shut down the system.
- Potential risk: Unauthorized shutdowns can disrupt system operations and potentially lead to data loss.

9. **SeDebugPrivilege:**

- Allows a process to debug other programs.
- Potential risk: Debugging privileges can be misused to analyze, modify, or control other processes, potentially enabling malware to evade detection and perform malicious actions.

The processes named v6742880.exe, b4574510.exe, v1974809.exe, and 1dec94b96of40e.exe have been endowed with certain privileges. This is a cause for concern because these privileges bestow the processes with the potential to inflict harm upon the system.

However, the malware exhibits a consistent pattern of requesting the same privileges in the virtual sandbox as observed in the hardware environment. This uniformity suggests that the malware’s approach to exploiting system permissions remains consistent across different execution contexts. This behavior may be attributed to the malware’s static characteristics, its ability to operate independently of the specific environment, or its unchanging focus on specific malicious objectives associated with the requested privileges. It’s worth noting that this consistency could be a deliberate tactic employed by malware developers to avoid detection by minimizing conspicuous variations between environments. While the malware’s actions related to privileges remain similar, it’s crucial to recognize that its overall behavior may encompass additional evasion and obfuscation techniques that go beyond the analysis of privileges alone.

5.2.7 CMDLINE

```

1892 1dec94b960f40e "C:\Users\asad\Desktop\1dec94b960f40e3a95e4aaa81f5783c221e4792d429701701052a03686aeb2cd\1dec94b960f40e3a95e4aaa81f5783c221e4792d429701701052a03686aeb2cd.exe"
3792 v6742880.exe C:\Users\asad\AppData\Local\Temp\IXP000.TMP\v6742880.exe
1592 v1974809.exe C:\Users\asad\AppData\Local\Temp\IXP001.TMP\v1974809.exe
1100 1dec94b960f40e "C:\Users\asad\Desktop\1dec94b960f40e3a95e4aaa81f5783c221e4792d429701701052a03686aeb2cd\1dec94b960f40e3a95e4aaa81f5783c221e4792d429701701052a03686aeb2cd.exe"
5688 v6742880.exe C:\Users\asad\AppData\Local\Temp\IXP003.TMP\v6742880.exe
5804 v1974809.exe C:\Users\asad\AppData\Local\Temp\IXP004.TMP\v1974809.exe
5900 b4574510.exe C:\Users\asad\AppData\Local\Temp\IXP005.TMP\b4574510.exe
1400 b4574510.exe C:\Users\asad\AppData\Local\Temp\IXP002.TMP\b4574510.exe

9368 1dec94b960f40e Required memory at 0x5d1bf8 is inaccessible (swapped)
3108 v6742880.exe Required memory at 0x111cb8 is inaccessible (swapped)
7336 v1974809.exe Required memory at 0x1b1cb8 is inaccessible (swapped)
768 1dec94b960f40e Required memory at 0x151c28 is inaccessible (swapped)
4276 v6742880.exe Required memory at 0x1b1cd8 is inaccessible (swapped)
9024 v1974809.exe Required memory at 0x5a1cd8 is inaccessible (swapped)
3512 b4574510.exe Required memory at 0x5c1cd8 is inaccessible (swapped)
4940 b4574510.exe Required memory at 0x5d1cb8 is inaccessible (swapped)
5376 1dec94b960f40e Required memory at 0x541bf8 is inaccessible (swapped)
2632 v6742880.exe Required memory at 0x531cb8 is inaccessible (swapped)
4456 v1974809.exe Required memory at 0x5e1cb8 is inaccessible (swapped)

```

Figure 5.12: hardware and virtual sandbox cmdline.

In Figure 5.12 presented above, we are able to observe a potential invasion technique employed by the malware sample. By examining and contrasting the behaviors in both hardware and virtual sandboxes, the distinctions become apparent, as outlined below:

In the virtual sandbox memory dump, several notable findings were observed:

- Multiple Temporary Executables:** The presence of multiple executable files (v6742880.exe, v1974809.exe, and b4574510.exe) within user-specific directories, specifically under C:\Users\asad, raises concerns. Malware often disguises itself as legitimate files in these locations to evade detection.
- User Interaction:** The location of these executables hints at potential user-initiated actions, suggesting that the malware may have been triggered by user interaction. This is a concerning red flag as it could indicate a social engineering component, such as email attachments or malicious downloads.

In the hardware sandbox memory dump, the focus was on identifying "Required memory at" entries indicating inaccessible (swapped) memory regions. The following observations were made:

1. **Memory Corruption:** The presence of multiple "Required memory at" entries strongly suggests that the malware engaged in memory manipulation and manipulation tactics to evade analysis. Memory corruption is a common technique used by malware to make the analysis more challenging.

COMPARATIVE ANALYSIS: The key differences between the two sandbox environments lie in their behavior. The virtual sandbox showed signs of user interaction with potentially malicious files, while the hardware sandbox exposed memory manipulation tactics by the malware. The discrepancies are vital and need to be addressed:

- In the virtual sandbox, the malware's initial execution was successful, and it posed a risk to user systems. The executables located in user directories could potentially exfiltrate sensitive information, posing a threat to data privacy and system security.
- In the hardware sandbox, the malware's behavior is more evasive. It actively manipulates memory, causing memory corruption, which indicates an intent to disrupt analysis and conceal its activities.

RED FLAGS:

1. **Execution in User Directories:** The presence of executables in user directories is a significant concern as it suggests user interaction with potentially malicious files.
2. **Inaccessible Memory Regions:** The "Required memory at" entries in the hardware sandbox memory dump indicate memory corruption and active evasion techniques employed by the malware.

5.2.8 HANDLES

| | | | | | | | |
|-----|-----------|----------------|-------|-----------|----------|----------------|----------|
| 564 | csrss.exe | 0xb306501a3080 | 0x8c4 | Process | 0x1fffff | 1dec94b960f40e | Pid 9368 |
| 564 | csrss.exe | 0xb30650356e10 | 0x8c8 | ALPC Port | 0x1f0001 | | |
| 564 | csrss.exe | 0xb3064aa31d20 | 0x8cc | ALPC Port | 0x1f0001 | | |
| 564 | csrss.exe | 0xb3064f7c3080 | 0x8d0 | Process | 0x1fffff | v1974809.exe | Pid 4456 |
| 564 | csrss.exe | 0xb3064e4572c0 | 0x8d4 | ALPC Port | 0x1f0001 | | |
| 564 | csrss.exe | 0xb3064f631080 | 0x8d8 | Thread | 0x1fffff | Tid 9272 | Pid 9368 |
| 564 | csrss.exe | 0xa28c9f32f2b0 | 0x8e0 | Section | 0xf0007 | | |
| 564 | csrss.exe | 0xa28c9765a4b0 | 0x8e4 | Section | 0xf0007 | | |

Figure 5.13: Malware interaction with critical system process.

In figure 5.13 above we can see that in hardware based sandbox, "564 csrss.exe 0xb3064f631080 0x8d8 Thread 0x1fffff Tid 9272 Pid 9368" indicates that "csrss.exe" is holding a handle (0x8d8) that references a thread (Tid 9272) in the process with PID 9368 (the malware). This suggests that the "csrss.exe" process is interacting with or managing a specific thread within the malware process. This interaction between "csrss.exe" and a malware thread is highly suspicious and a red flag. "csrss.exe" should not normally be involved with the management of threads within a malware process.

RED FLAGS

1. **Data theft or exfiltration:** "csrss.exe" may be accessing or manipulating sensitive data within the malware process, leading to data theft or exfiltration.
2. **Manipulation of malware behavior:** "csrss.exe" could potentially control or influence the actions of the malware process, redirecting it to perform malicious activities.
3. **Privilege escalation:** If "csrss.exe" is communicating with the malware process, it may attempt to elevate its privileges, giving it more control over the system.
4. **Obfuscation or evasion:** The malware might use "csrss.exe" as a means to obfuscate its presence, making it harder for security tools to detect and analyze the malicious activity.
5. **Payload delivery:** The interactions could be part of a multi-stage attack, with "csrss.exe" delivering additional payloads or instructions to the malware process.
6. **Execution of malicious code:** There's a possibility that "csrss.exe" is executing code or commands within the malware process, potentially leading to further harmful actions.

| | | | | | | |
|------|----------------|----------------|-------|-----------------|----------|--|
| 1892 | 1dec94b960f40e | 0xb6056e218670 | 0x18c | EtwRegistration | 0x804 | |
| 1892 | 1dec94b960f40e | 0xb6056e19f860 | 0x190 | Event | 0x1f0003 | |
| 1892 | 1dec94b960f40e | 0xb6056e19fde0 | 0x194 | Event | 0x1f0003 | |
| 1892 | 1dec94b960f40e | 0x800659cc0bc0 | 0x198 | Key | 0x20019 | MACHINE\SYSTEM\CONTROLSET001\CONTROL\NETWORKPROVIDER\HWORDER |
| 1892 | 1dec94b960f40e | 0x800659cbf790 | 0x19c | Key | 0x20019 | MACHINE\SYSTEM\CONTROLSET001\CONTROL\NETWORKPROVIDER\PROVIDERORDER |
| 1892 | 1dec94b960f40e | 0xb6056e19fae0 | 0x1a0 | Semaphore | 0x100003 | |
| 1892 | 1dec94b960f40e | 0xb6056e19f760 | 0x1a4 | Semaphore | 0x100003 | |
| 1892 | 1dec94b960f40e | 0xb6056e21a970 | 0x1a8 | EtwRegistration | 0x804 | |
| 1892 | 1dec94b960f40e | 0xb6056e70b90 | 0x1ac | File | 0x120089 | \Device\DeviceApt\CMapi |
| 1892 | 1dec94b960f40e | 0xb6056e219e10 | 0x1b0 | EtwRegistration | 0x804 | |
| 1892 | 1dec94b960f40e | 0xb6056e2190f0 | 0x1b4 | EtwRegistration | 0x804 | |
| 1892 | 1dec94b960f40e | 0xb6056e21a190 | 0x1b8 | EtwRegistration | 0x804 | |
| 1892 | 1dec94b960f40e | 0xb6056e19f3e0 | 0x1bc | Semaphore | 0x100003 | |
| 1892 | 1dec94b960f40e | 0xb6056e19fd60 | 0x1c0 | Semaphore | 0x100003 | |
| 1892 | 1dec94b960f40e | 0xb6056e19f560 | 0x1c4 | Event | 0x1f0003 | |
| 1892 | 1dec94b960f40e | 0xb6056e6f740 | 0x1c8 | File | 0x100003 | \Device\KsecDD |
| 1892 | 1dec94b960f40e | 0xb6056e219c50 | 0x1cc | EtwRegistration | 0x804 | |
| 1892 | 1dec94b960f40e | 0xb6056e2191d0 | 0x1d0 | EtwRegistration | 0x804 | |
| 1892 | 1dec94b960f40e | 0xb6056e2198d0 | 0x1d4 | EtwRegistration | 0x804 | |
| 1892 | 1dec94b960f40e | 0xb6056e21a350 | 0x1d8 | EtwRegistration | 0x804 | |

Figure 5.14: Malware interaction for virtual.

In Figure 5.14, representing the virtual environment, a similar pattern of malware interaction with critical system processes is observed, paralleling the behavior depicted in the physical environment. However, the virtual environment provides additional insights, as detailed in the figure.

The process ID 1892, which is associated with a malware sample, shows various handles and their respective types. Each handle type indicates different interactions or resources accessed by the process. Here's what you can infer from the information, along with potential damage that could be done:

1. **EtwRegistration handles:** Two EtwRegistration handles are observed. These are related to Event Tracing for Windows (ETW) registration, a component used for monitoring and logging events in Windows. Potential damage: The malware could manipulate or interfere with event tracing, which might allow it to cover its tracks and avoid detection.
2. **Event handles:** Three Event handles are present, indicating the use of events for synchronization and inter-process communication. Potential damage: Events can be used to coordinate actions between processes. The malware might use this for launching attacks or managing malicious activities.
3. **Key handles:** Two Key handles are found, which are associated with specific registry keys in the "MACHINE\SYSTEM\CONTROLSET001" path. These keys could be accessed for various purposes. Potential damage: Unauthorized access to registry keys can lead to changes in system settings, potentially causing instability, and enabling further malware persistence or privilege escalation.
4. **Semaphore handles:** Four Semaphore handles are listed, suggesting the use of semaphores for synchronization and coordination between processes. Potential damage: Semaphores

can be used for synchronization. The malware might utilize these for timing or coordination of malicious actions.

5. **File handles:** Two File handles are seen. One is related to ”\Device\DeviceApi\CMApi,” and the other is related to ”\Device\KsecDD.” These handles could indicate interactions with these device drivers or files. Potential damage: File handles can be used for reading, writing, or manipulating files and drivers. The malware might attempt to modify critical system files or steal sensitive information.
6. **Mutant handles:** Two Mutant handles are found, which are used for synchronization and coordination. Mutants are often used to ensure exclusive access to resources. Potential damage: Malware could use mutants to control access to specific resources or ensure only one instance of the malware runs, making it harder to detect and remove.

The presence of these various handles indicates that the malware process is interacting with different system resources, events, and synchronization mechanisms. These interactions could be used for malicious purposes such as evasion, privilege escalation, communication, or resource manipulation. The potential damage includes compromising system stability, enabling persistence, launching attacks, and interfering with system functions.

5.3 NETWORK TRAFFIC

We obtained four pcap files, each divided into two sets—one from a virtual sandbox and the other from a hardware sandbox. Utilizing a malware traffic analysis website, we extracted data. To analyze the traffic and identify potential issues, we employed Wireshark and Brim Security Zui.

5.3.1 MALWARE’S TRAFFIC DATABASE

As previously noted, we employed four malware samples for analysis, namely:

- Brazil-Targeted malware infection from email.
- Google AD —> fake libre office page.
- Italian template word docs push ursnif.
- Rigeek and redline.

5.3.2 MALWARE TRAFFIC ANALYSIS

| Sandbox | IPv4 Address | Subnet Musk | Default Gateway |
|----------|--------------|-----------------|-----------------|
| Hardware | 192.168.0.3 | 255.255.255.128 | 192.168.0.1 |
| Virtual | 192.168.0.4 | 255.255.255.128 | 192.168.0.1 |

Table 5.2: Hardware and Virtual sandbox configurations with the network.

BRAZIL-TARGETED MALWARE INFECTION FROM EMAIL.

HARDWARE SANDBOX: Following the execution of malware, a connection is observed between IP address 192.168.0.3 and IP address 192.168.0.2, as captured in Wireshark, as depicted in the figure below.

| | | | | | |
|---|----------|-------------|-------------|-----|--|
| 1 | 0.000000 | 192.168.0.3 | 192.168.0.2 | TCP | 66 56254 → 7680 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1 |
| 2 | 0.000091 | 192.168.0.2 | 192.168.0.3 | TCP | 66 7680 → 56254 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1 |

Figure 5.15: Ip's handshakes

As we analyze the pcap file of the respective malware sample, these are the findings observed in our hardware sandbox. Initially, we initiated Wireshark and subsequently executed the malware sample within the system, capturing approximately 3736 packets. This dataset is deemed sufficient for uncovering the malware's interactions within the network traffic.

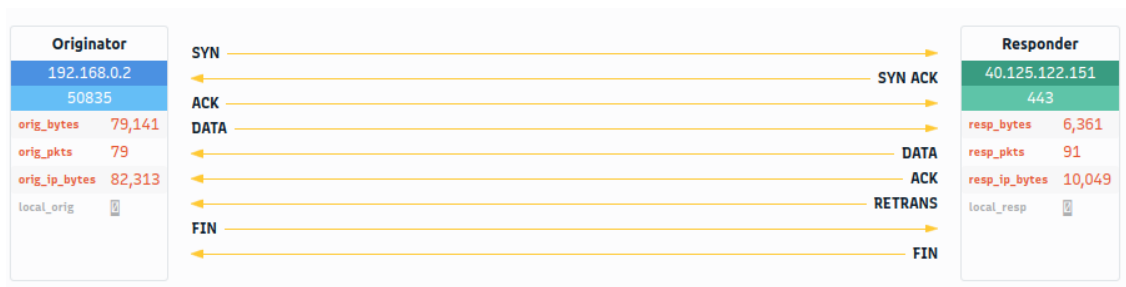


Figure 5.16: Suspicious Handshake.

Observing the depicted Figure 5.15 reveals an unusual handshake occurring between two IP addresses. Upon conducting a more in-depth examination of these two IP addresses (192.168.0.2 and 40.125.122.151), it becomes evident that they have already been flagged by previous security researchers. Ip 192.168.0.2 is related to a malware who is trying to make connections with

other domains and ip's. We initiated Wireshark, ran the malware sample, and captured network packets for analysis. Upon examination, we observed a connection between IP addresses 192.168.0.2 and 40.125.122.151, raising suspicions. Figure 5.16 below illustrates the network activity graph for IP address 192.168.0.2, depicting its interactions with both communicating and referred files.

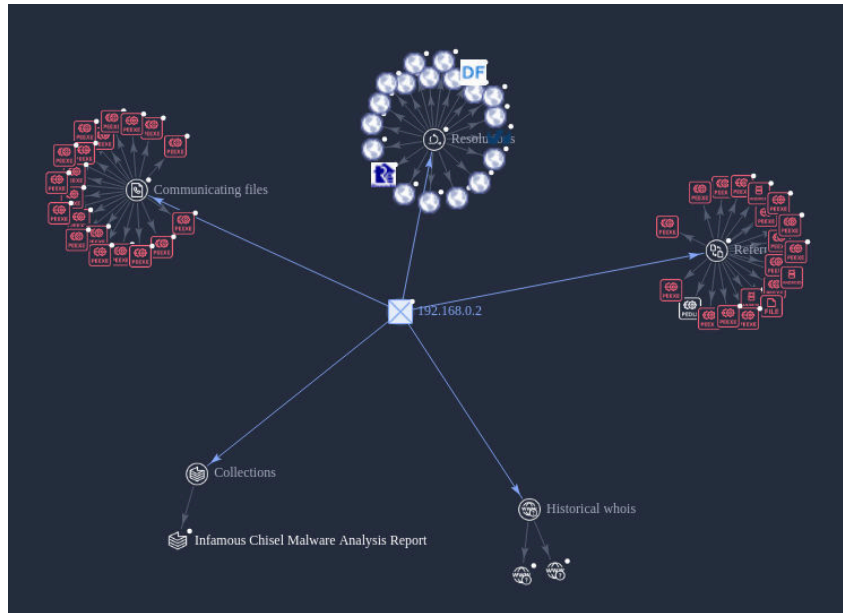


Figure 5.17: Infamous Chisel graph.

Infamous Chisel, a comprehensive set of components designed for persistent access to compromised Android devices via the Tor network, periodically collects and exfiltrates victim information. This includes system details, commercial applications, and apps specific to the Ukrainian military.

The malware conducts routine scans on the device, specifically targeting files with predefined extensions. It also features functionality for scanning the local network to gather data on active hosts, open ports, and banners.

Remote access is achieved by configuring and executing Tor with a hidden service, directing to a modified Dropbear binary for SSH connectivity.

Furthermore, the malware, associated with the IP address 192.168.0.2, expands its capabilities to encompass network monitoring, traffic collection, SSH access, network scanning, and SCP file transfer. Notably, the IP address 192.168.0.2 is a known element of the Infamous Chisel malware infrastructure.

The IP address 192.168.0.2 is linked to malicious activity. Figure 5.16 displays its connections with reference and communicating files, which are categorized in the figure 5.17 below for further examination.





| Children Relations | |  | Children Relations | |  |
|--------------------|------|---|--------------------|-----|---|
| Collections | 281 |  | Bundled files | 185 |  |
| Contacted domains | 166 |  | Collections | 3 |  |
| Contacted ips | 521 |  | Contacted domains | 3 |  |
| Contacted urls | 187 |  | Contacted ips | 20 |  |
| Dropped files | 2840 |  | Dropped files | 71 |  |
| Execution parents | 10 |  | Execution parents | 3 |  |

Figure 5.18: 192.168.0.2 refer and communicating files.

In Figure 5.17, the left side represents communicating files, while the right side represents reference files associated with the IP address 192.168.0.2. Upon closer examination of a node from the communicating files, a noteworthy instance emerges. A Win32 executable named "file.exe," sized at 133.87 kB, is identified. This file, initially observed on October 11, 2020, has undergone one submission and is flagged as malicious by security tools such as DeepInstinct, Cybereason, AVG, BitDefenderTheta, and Fortinet. These collective detections strongly suggest the involvement of the IP address 192.168.0.2 in previous malicious activities. As shown in the figure 5.18 below.

| | |
|-------------|---------------------|
| Type | Win32 EXE |
| Size | 133.87 kB |
| First Seen | 2020-10-11 12:53:11 |
| Last Seen | 2020-10-11 12:53:11 |
| Submissions | 1 |
| File Name | file.exe |

Detections

| | |
|------------------|---------------------------------|
| DeepInstinct | MALICIOUS |
| Cybereason | malicious.10d215 |
| AVG | Win32:TrojanX-gen [Trj] |
| BitDefenderTheta | Gen:NN.ZexaF.36792.iGX@aWE D1Fo |
| Fortinet | W32/Themida.0752!tr |

Figure 5.19: One of the nodes from the communicating files.

| | |
|-------------|---------------------|
| Type | Win32 EXE |
| Size | 877.00 kB |
| First Seen | 2008-02-04 19:33:13 |
| Last Seen | 2021-12-09 21:11:02 |
| Submissions | 12 |
| File Name | LOD.EXE |

Detections

| | |
|--------------|------------------------------|
| Cybereason | malicious.fc4c3c |
| MaxSecure | Trojan.Malware.300983.susgen |
| Yandex | Trojan.Hooker!TbSub9OIVh0 |
| DeepInstinct | MALICIOUS |
| Kingsoft | malware.kb.a.1000 |

Figure 5.20: One of the nodes from the refer files.

In Figure 5.19, a noteworthy entry in the "Refer Files" section is highlighted. The focus is on a Win32 executable named "LOD.EXE," with a size of 877.00 kB. This file was initially detected on February 4, 2008, and its most recent appearance was on December 9, 2021, with a total of 12 submissions. The file has raised concerns as it has been flagged as malicious by several cybersecurity tools, including Cybereason, MaxSecure, Yandex, DeepInstinct, and Kingsoft.

Cybereason identifies it as "malicious.fc4c3c," MaxSecure labels it as "Trojan.Malware.300983.susgen," Yandex categorizes it as "Trojan.Hooker!Tbsub9OIVho," and DeepInstinct marks it as "MALICIOUS." This collective identification highlights potential security risks associated with "LOD.EXE,".

The presence of the Win32 executable "file.exe" and "LOD.EXE" associated with the IP address 192.168.0.2 raises red flags due to their consistent identification as malicious by multiple cybersecurity tools. The unified detection by tools such as DeepInstinct, Cybereason, AVG, BitDefenderTheta, Fortinet, MaxSecure, Yandex, and Kingsoft indicates a high likelihood of malicious activities linked to this IP address. The historical involvement of "file.exe" and the persistent threat posed by "LOD.EXE" underscore the need for immediate attention and mitigation efforts to address the potential security risks introduced by 192.168.0.2.

Examining Figure 5.15 reveals a network handshake between IP addresses 40.125.122.151 and 192.168.0.2, prompting further attention to this connection.

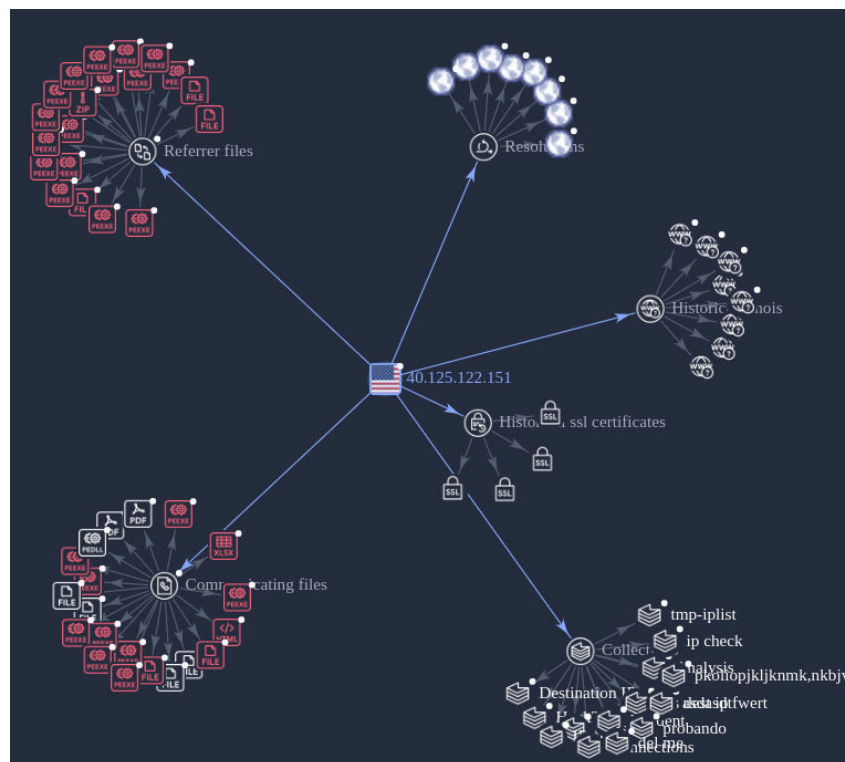


Figure 5.21: Ip 40.125.122.151 graph.

By examining a node from the refer and communicating files of IP address 40.125.122.151, depicted in Figure 5.20, we observe recurring malicious behavior, as detailed below.

Upon examining a node from the refer files associated with IP address 40.125.122.151, a suspicious JavaScript file named "baa29ee86782076df01c2a7e0df4d189dbc27a3da7a6ee10c0cf5-94e5cfa8f5.js" is identified. This file, first observed on April 16, 2023, has undergone one submission and is flagged as malicious by security tools such as MAX, ALYac, Google, GData, and ZoneAlarm. Simultaneously, a corresponding node from the communicating files reveals an HTML file named "payload2.html" with a size of 557.00 B, first observed on June 1, 2022. This file has also undergone one submission and is detected as malicious by Fortinet, ZoneAlarm, Sophos, Kaspersky, and ESET-NOD32. The collective detection of these files underscores a concerning association with malicious activities related to IP address 40.125.122.151, signifying a red flag, we can see it in figure 5.20 below.

| | | | |
|-------------|--|-------------|---------------------|
| Type | JavaScript | Type | HTML |
| Size | 56.04 kB | Size | 557.00 B |
| First Seen | 2023-04-16 20:10:37 | First Seen | 2022-06-01 11:10:12 |
| Last Seen | 2023-04-16 20:10:37 | Last Seen | 2022-06-01 11:10:12 |
| Submissions | 1 | Submissions | 1 |
| File Name | baa29ee86782076df01c2a7e0df4d189dbc27a3da7a6ee10c0cf594e5cfa8f5.js | File Name | payload2.html |

| Detections | |
|------------|----------------------------------|
| MAX | malware (ai score=84) |
| ALYac | Heur.BZC.PZQ.Boxter.909.3F7C81D1 |
| Google | Detected |
| GData | Heur.BZC.PZQ.Boxter.909.3F7C81D1 |
| ZoneAlarm | HEUR:Trojan.PowerShell.Generic |

| Detections | |
|------------|--------------------------------|
| Fortinet | HTML/CVE_2022_30190.Altr |
| ZoneAlarm | HEUR:Exploit.Script.Generic |
| Sophos | Troj/DocDI-AGDX |
| Kaspersky | HEUR:Exploit.Script.Generic |
| ESET-NOD32 | Win32/Exploit.CVE-2022-30190.A |

Figure 5.22: Nodes of refer and communicating files.

VIRTUAL SANDBOX: In comparing the virtual and hardware sandboxes during the execution of the same malware, a notable distinction emerges. In the virtual sandbox, the IP address 192.168.0.4 engages directly with IP 40.125.122.151. Conversely, in the hardware-based sandbox, a two-step process occurs: first, a handshake is established with 192.168.0.2, followed by a subsequent handshake with 40.125.122.151. As shown in figure 5.23 below.



Figure 5.23: Handshake in virtual sandbox.

The observed difference in network behavior between the virtual sandbox and hardware sandbox, specifically regarding the sequence of handshakes with different IPs (192.168.0.2 and 40.125.122.151), may be attributed to several factors related to the environment and the malware’s behavior:

1. **Environment Configuration:** Virtual environments often have different network configurations, including network interfaces, settings, and virtualization-specific parameters. The malware may respond differently based on the network environment it detects.
2. **Malware Evasion Tactics:** Malware is known to employ evasion tactics to avoid detection. By introducing an additional handshake with a local IP (192.168.0.2), the malware may be attempting to confuse or bypass security measures. This could be a deliberate strategy to make the analysis more challenging.
3. **Dynamic Behavior:** Malware can exhibit dynamic behavior based on its analysis of the environment. It might perform certain actions conditionally, adapting its behavior to the specifics of the system it is running on. This adaptability could lead to variations in the observed network interactions.
4. **Network Segmentation:** The hardware sandbox might have a more complex network setup, including network segmentation or additional security layers. The malware might be responding to this by adjusting its communication strategy.
5. **Stealth and Persistence:** Making an initial connection to a local IP (192.168.0.2) before reaching out to the final destination (40.125.122.151) could be a technique to establish persistence, evade detection, or blend into the local network environment.

In Figure 5.24 and 5.25, a node from the communicating files associated with IP address 199.201.110.204 reveals the presence of a potentially malicious file. The file, identified as an Office Open XML Document named "CT6034_Assignment_2_-.docx," has a size of 1003.10

kB. This document was first observed on March 29, 2023, with a single submission. Cybersecurity tools, including Antiy-AVL and ESET-NOD32, have flagged it as a potential threat, categorizing it as a Trojan or a document with VBA (Visual Basic for Applications) code.

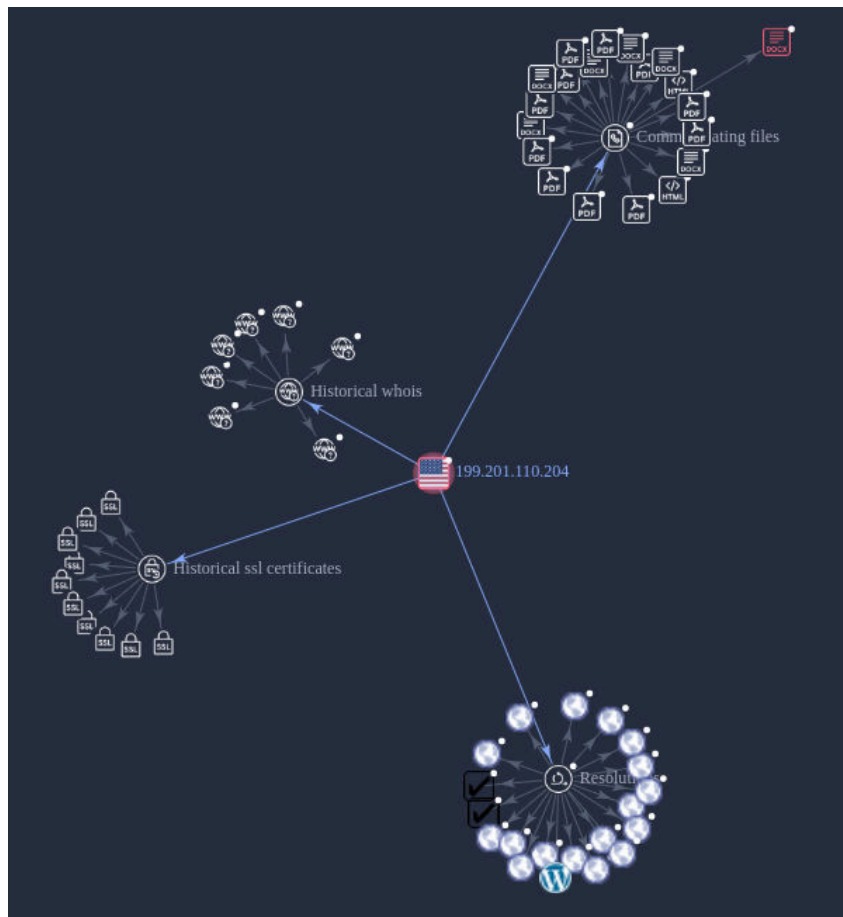


Figure 5.24: Ip 199.201.110.204 communicating file node.

| | |
|-------------|--------------------------------|
| Type | Office Open XML Document |
| Size | 1003.10 kB |
| First Seen | 2023-03-29 18:35:06 |
| Last Seen | 2023-03-29 18:35:06 |
| Submissions | 1 |
| File Name | CT6034_Assignment_2 _-.docx |

Detections

| | |
|------------|---------------------------------|
| Antiy-AVL | Trojan[Phishing]/MSOffice.Agent |
| ESET-NOD32 | VBA/Subdoc.B |
| Bkav | Undetected |
| Lionic | Undetected |
| Elastic | Undetected |

Figure 5.25: Communicating file node details.

After executing the same malware on both hardware and virtual sandboxes, we observed a notable discrepancy in the network traffic when monitoring with Wireshark. Specifically, the IP address 199.201.110.204, identified as malicious by various security researchers, was absent from the virtual sandbox traffic. This deviation raises concerns about potential evasion or altered behavior based on the sandbox environment. The absence of the IP address 199.201.110.204 in the virtual sandbox traffic may be attributed to the malware's ability to recognize the virtualized environment and alter its behavior accordingly. Malware often employs evasion techniques to detect virtualization or sandboxing, allowing it to behave differently or remain dormant to avoid detection and analysis. In this case, the malware might have checks in place to identify characteristics of a virtual environment, leading to the decision not to communicate with the IP address in question. This adaptive behavior is a common tactic employed by malware to evade detection and hinder analysis efforts in virtualized settings.

***NOTE:** The variations in traffic are substantial, and we cannot address all of them comprehensively.*

5.4 LOG FILES

We captured PML files using Microsoft's Procmon tool while running various processes. As part of our analysis, we executed a malware sample identified by the hash: 1dec94b96of40e3a95-e4aaa81f5783c221e4792d429701701052a03686aeb2c. Our aim is to observe the changes made by the malware and understand its behavior in both our hardware and virtual sandboxes. Alongside memory dumps and network traffic analyses, we utilized Process Monitor to gain insights into the malware's actions and interactions with the system.

5.4.1 LOG'S OF HARDWARE SANDBOX

In the figure 5.26 below, we can see that upon executing the malware sample in our dedicated hardware sandbox, we observed that the malware, identified by the hash 1dec94b96of40e3a95-e4aaa81f5783c221e4792d429701701052a03686aeb2cd.exe, was successfully launched and is now prepared for analysis. Notably, upon the initiation of the process, a thread was also generated.

| Time | Process Name | PID | Operation | Path | Result | Detail |
|-----------|------------------|------|-------------------|--|------------------|------------------------|
| 11:45:... | 1dec94b960f40... | 9368 | Process Start | | SUCCESS | Parent PID: 6432... |
| 11:45:... | 1dec94b960f40... | 9368 | Thread Create | | SUCCESS | Thread ID: 9272 |
| 11:45:... | 1dec94b960f40... | 9368 | QueryNormalize... | C:\Users\asad\Downloads\1dec94b96... | BUFFER OVERFL... | |
| 11:45:... | 1dec94b960f40... | 9368 | Load Image | C:\Users\asad\Downloads\1dec94b96... | SUCCESS | Image Base: 0xc0e0... |
| 11:45:... | 1dec94b960f40... | 9368 | QueryNormalize... | C:\Windows\System32\ntdll.dll | SUCCESS | |
| 11:45:... | 1dec94b960f40... | 9368 | Load Image | C:\Windows\System32\ntdll.dll | SUCCESS | Image Base: 0x7fe... |
| 11:45:... | 1dec94b960f40... | 9368 | QueryNormalize... | C:\Windows\SysWOW64\ntdll.dll | SUCCESS | |
| 11:45:... | 1dec94b960f40... | 9368 | Load Image | C:\Windows\SysWOW64\ntdll.dll | SUCCESS | Image Base: 0x775... |
| 11:45:... | 1dec94b960f40... | 9368 | CreateFile | C:\Windows\Prefetch\1DEC94B960F4... | NAME NOT FOUND | Desired Access: Q... |
| 11:45:... | 1dec94b960f40... | 9368 | RegOpenKey | HKLM\System\CurrentControlSet\Contr... | REPARSE | Desired Access: Q... |
| 11:45:... | 1dec94b960f40... | 9368 | RegOpenKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS | Desired Access: Q... |
| 11:45:... | 1dec94b960f40... | 9368 | RegQueryValue | HKLM\System\CurrentControlSet\Contr... | NAME NOT FOUND | Length: 80 |
| 11:45:... | 1dec94b960f40... | 9368 | RegCloseKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS | |
| 11:45:... | 1dec94b960f40... | 9368 | RegOpenKey | HKLM\SYSTEM\CurrentControlSet\Con... | REPARSE | Desired Access: Q... |
| 11:45:... | 1dec94b960f40... | 9368 | RegOpenKey | HKLM\SYSTEM\CurrentControlSet\Con... | NAME NOT FOUND | Desired Access: Q... |
| 11:45:... | 1dec94b960f40... | 9368 | RegOpenKey | HKLM\SYSTEM\CurrentControlSet\Con... | REPARSE | Desired Access: Q... |
| 11:45:... | 1dec94b960f40... | 9368 | RegOpenKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS | Desired Access: Q... |
| 11:45:... | 1dec94b960f40... | 9368 | RegQueryValue | HKLM\System\CurrentControlSet\Contr... | NAME NOT FOUND | Length: 24 |
| 11:45:... | 1dec94b960f40... | 9368 | RegCloseKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS | |
| 11:45:... | 1dec94b960f40... | 9368 | CreateFile | C:\Windows\System32\wow64.dll | SUCCESS | Desired Access: E... |
| 11:45:... | 1dec94b960f40... | 9368 | QueryNormalize... | C:\Windows\System32\wow64.dll | SUCCESS | |
| 11:45:... | 1dec94b960f40... | 9368 | Load Image | C:\Windows\System32\wow64.dll | SUCCESS | Image Base: 0x7fe... |
| 11:45:... | 1dec94b960f40... | 9368 | ReadFile | C:\Windows\System32\wow64.dll | SUCCESS | Offset: 242,176, Le... |
| 11:45:... | 1dec94b960f40... | 9368 | QueryNormalize... | C:\Windows\System32\wow64win.dll | SUCCESS | |
| 11:45:... | 1dec94b960f40... | 9368 | Load Image | C:\Windows\System32\wow64win.dll | SUCCESS | Image Base: 0x7fe... |
| 11:45:... | 1dec94b960f40... | 9368 | ReadFile | C:\Windows\System32\wow64win.dll | SUCCESS | Offset: 449,024, Le... |
| 11:45:... | 1dec94b960f40... | 9368 | ReadFile | C:\Windows\System32\wow64win.dll | SUCCESS | Offset: 323,584, Le... |
| 11:45:... | 1dec94b960f40... | 9368 | CreateFile | C:\Windows\System32\wow64log.dll | NAME NOT FOUND | Desired Access: R... |
| 11:45:... | 1dec94b960f40... | 9368 | ReadFile | C:\Windows\System32\wow64.dll | SUCCESS | Offset: 337,408, Le... |
| 11:45:... | 1dec94b960f40... | 9368 | CreateFile | C:\Windows\System32\wow64.dll | SUCCESS | Desired Access: R... |
| 11:45:... | 1dec94b960f40... | 9368 | QueryNameInfo... | C:\Windows | SUCCESS | Name: \Windows |
| 11:45:... | 1dec94b960f40... | 9368 | CloseFile | C:\Windows | SUCCESS | |
| 11:45:... | 1dec94b960f40... | 9368 | RegOpenKey | HKLM\Software\Microsoft\Wow64\86 | SUCCESS | Desired Access: R... |
| 11:45:... | 1dec94b960f40... | 9368 | RegQueryValue | HKLM\SOFTWARE\Microsoft\Wow64\... | NAME NOT FOUND | Length: 520 |
| 11:45:... | 1dec94b960f40... | 9368 | RegQueryValue | HKLM\SOFTWARE\Microsoft\Wow64\... | SUCCESS | Type: REG_SZ, Le... |
| 11:45:... | 1dec94b960f40... | 9368 | RegCloseKey | HKLM\SOFTWARE\Microsoft\Wow64\... | SUCCESS | |
| 11:45:... | 1dec94b960f40... | 9368 | QueryNormalize... | C:\Windows\System32\wow64cpu.dll | SUCCESS | |
| 11:45:... | 1dec94b960f40... | 9368 | Load Image | C:\Windows\System32\wow64cpu.dll | SUCCESS | Image Base: 0x775... |
| 11:45:... | 1dec94b960f40... | 9368 | ReadFile | C:\Windows\System32\wow64cpu.dll | SUCCESS | Offset: 6,144, Leng... |
| 11:45:... | 1dec94b960f40... | 9368 | ReadFile | C:\Windows\System32\wow64cpu.dll | SUCCESS | Offset: 9,216, Leng... |
| 11:45:... | 1dec94b960f40... | 9368 | RegOpenKey | HKLM\System\CurrentControlSet\Contr... | REPARSE | Desired Access: Q... |
| 11:45:... | 1dec94b960f40... | 9368 | RegOpenKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS | Desired Access: Q... |
| 11:45:... | 1dec94b960f40... | 9368 | RegSetInfoKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS | KeySetInformation... |
| 11:45:... | 1dec94b960f40... | 9368 | RegQueryValue | HKLM\System\CurrentControlSet\Contr... | NAME NOT FOUND | Length: 80 |

Figure 5.26: Malware sample process log's.

PROCESS THREAD: Upon executing the provided malware sample within our controlled environment, a stack trace analysis was conducted. The stack as shown in figure 5.27, includes entries related to the kernel (ntoskrnl.exe), indicating interactions at the system level. Notably, several entries marked as <unknown> raise concerns, as they suggest intentional obfuscation to conceal specific operations. User space addresses lacking module information further complicate the identification of the malware’s origin and purpose. The absence of clear module details and the potential use of anti-analysis techniques highlight the sophistication of the malware.

| Frame | Module | Location | Address | Path |
|-------|-------------|------------------------|-------------------|---------------------------------|
| K 0 | ntoskml.exe | ntoskml.exe + 0x642776 | 0xffff80664e42776 | C:\Windows\system32\ntoskml.exe |
| K 1 | ntoskml.exe | ntoskml.exe + 0x63c346 | 0xffff80664e3c346 | C:\Windows\system32\ntoskml.exe |
| K 2 | ntoskml.exe | ntoskml.exe + 0x672440 | 0xffff80664e72440 | C:\Windows\system32\ntoskml.exe |
| K 3 | ntoskml.exe | ntoskml.exe + 0x40a9b8 | 0xffff80664c0a9b8 | C:\Windows\system32\ntoskml.exe |
| U 4 | <unknown> | 0x7fe7baee8f4 | 0x7fe7baee8f4 | |
| U 5 | <unknown> | 0x7fe796c8e73 | 0x7fe796c8e73 | |
| U 6 | <unknown> | 0x7fe796c63c3 | 0x7fe796c63c3 | |
| U 7 | <unknown> | 0x7fe7a95db20 | 0x7fe7a95db20 | |
| U 8 | <unknown> | 0x7fe598b526b | 0x7fe598b526b | |
| U 9 | <unknown> | 0x7fe598b697e | 0x7fe598b697e | |
| U 10 | <unknown> | 0x7fe7a2796e3 | 0x7fe7a2796e3 | |
| U 11 | <unknown> | 0x7fe7a20276b | 0x7fe7a20276b | |
| U 12 | <unknown> | 0x7fe7a25a278 | 0x7fe7a25a278 | |
| U 13 | <unknown> | 0x7fe7a23a416 | 0x7fe7a23a416 | |
| U 14 | <unknown> | 0x7fe7a23a046 | 0x7fe7a23a046 | |
| U 15 | <unknown> | 0x7fe7a2486cf | 0x7fe7a2486cf | |
| U 16 | <unknown> | 0x7fe7a247c88 | 0x7fe7a247c88 | |
| U 17 | <unknown> | 0x7fe7a247271 | 0x7fe7a247271 | |
| U 18 | <unknown> | 0x7fe7a246cde | 0x7fe7a246cde | |
| U 19 | <unknown> | 0x7fe7a24b3b2 | 0x7fe7a24b3b2 | |
| U 20 | <unknown> | 0x7fe7ba70330 | 0x7fe7ba70330 | |
| U 21 | <unknown> | 0x7fe7baa2f76 | 0x7fe7baa2f76 | |
| U 22 | <unknown> | 0x7fe7a957034 | 0x7fe7a957034 | |
| U 23 | <unknown> | 0x7fe7baa26a1 | 0x7fe7baa26a1 | |

Figure 5.27: Process threads

QUERY NORMALIZED FILES: The analysis of the QueryNormalizedFileNameInformation-File event in the malware sample reveals a concerning "BUFFER OVERFLOW." The associated stack trace provides insights into the execution path and potential exploitation of system drivers. Notably, the stack involves the FLTMGR.SYS driver, suggesting interaction with the file system filter manager. Furthermore, the presence of the Shieldf.sys driver, which is not a standard Windows driver, raises suspicions about its involvement in the execution flow. The repetition of calls to the kernel (ntoskrnl.exe) and the file system filter manager may indicate an attempt to manipulate or compromise system-level functionalities. The presence of an <unknown> module in the stack adds an additional layer of obscurity, possibly employed as an evasion technique.

LOAD IMAGE: The examination of the malware sample reveals suspicious behavior marked by the successful execution of a "load image" operation. The involvement of critical kernel components, such as `ntoskrnl.exe`, signifies an attempt by the malware to interact with core system functions. The presence of an <unknown> module within the stack trace suggests deliberate obfuscation, raising concerns about the legitimacy and transparency of the loaded image. This obscured module could potentially be part of an evasion strategy employed by the malware to subvert analysis and detection mechanisms. The success of the operation highlights the incorporation of external code or data into the process, indicating a potential avenue for malicious activities. The overall behavior poses a significant threat as it allows the malware to clandestinely introduce and execute arbitrary content within the system, potentially leading to further compromise or exploitation. Immediate action is recommended, including reporting these findings to the security team for prompt mitigation measures against potential security risks. Further, another load image by the malware shows. In the recent "load image" operation, the malware sample extended its activities to involve the critical system component `ntdll.dll`. The presence of `ntdll.dll` in the stack trace suggests that the malware is dynamically linking with this essential Windows Dynamic Link Library for specific functions. `ntdll.dll` plays a crucial role in providing low-level access to the Windows kernel and various system services. The direct association of the malware with `ntdll.dll` during a "load image" operation raises red flags. This interaction could indicate an attempt by the malware to leverage advanced system-level functions, potentially for malicious purposes. The specific actions performed by the malware through `ntdll.dll` remain undisclosed, making it challenging to discern the exact nature of its activities. Given the criticality of `ntdll.dll` in system operations, the malware's involvement with this library poses a significant risk. The potential exploitation of system-level functionalities through `ntdll.dll` could lead to unauthorized access, system manipulation, or the introduction of further malicious elements.

FILE CREATION: The analysis of the malware sample reveals a concerning progression in its activities, particularly in a subsequent "create file" operation where the result indicates "name not found." The associated stack trace sheds light on the sequence of events: The operation begins with interactions involving the file system filter manager (`FLTMGR.SYS`), emphasizing the malware's engagement with file-related functions. Notably, the stack involves the non-standard driver `Shieldf.sys`, underscoring its continued participation in the malware's execution flow. The subsequent calls to `ntoskrnl.exe` and `ntdll.dll` indicate the malware's attempt to manipulate core system functionalities. The involvement of these critical system components

during a "create file" operation raises significant concerns. The result of "name not found" implies an attempt by the malware to access or create a file that does not exist, suggesting potential reconnaissance or data manipulation. The use of the non-standard driver Shieldf.sys in conjunction with system-level components intensifies the suspicion surrounding the malware's behavior. The complexity of the stack trace and the dynamic interactions with essential system elements make it challenging to discern the exact nature of the threat. This activity poses a severe risk, as it indicates the malware's intent to manipulate file-related operations at a fundamental level. The potential consequences include unauthorized access, data tampering, or the introduction of malicious files.

FILE READING: Following the "load image" operation, the malware sample engages in a "read file" operation, achieving success with I/O flags indicating non-cached, paging I/O, and synchronous paging I/O. The associated stack trace provides insights into the sequence of events:

1. **FLTMGR.SYS Involvement:** The stack initiates with interactions in FLTMGR.SYS, suggesting the malware's engagement with file system filter manager functions. This could indicate attempts to manipulate or read specific files within the system.
2. **ntoskrnl.exe and Shieldf.sys Interaction:** Subsequent involvement of ntoskrnl.exe and Shieldf.sys in the stack implies the malware's influence on core kernel and system functions. The combination suggests a multifaceted approach, potentially for evasive maneuvers or deeper system penetration.
3. **ntdll.dll Dynamic Linking:** The stack traces into ntdll.dll, showcasing the malware's reliance on advanced system-level functions. Dynamic linking with ntdll.dll is often indicative of sophisticated malware seeking to leverage powerful system capabilities.

RED FLAGS AND OBSERVATIONS:

- **Non-Cached and Paging I/O:** The utilization of non-cached, paging I/O, and synchronous paging I/O flags suggests a strategic approach to file reading. These flags may be employed to ensure data consistency or facilitate covert operations by bypassing standard file caching mechanisms.
- **Infiltration of Kernel and System Functions:** The interaction with ntoskrnl.exe and Shieldf.sys indicates the malware's intrusion into crucial system components. This type of behavior is concerning as it signifies potential attempts to subvert security measures and gain deeper control over the system.

- **Dynamic Linking Complexity:** The reliance on dynamic linking with ntdll.dll showcases a level of sophistication in the malware's design. This complexity may contribute to its ability to adapt to different system environments and evade detection.

The malware's combination of file system manipulation, kernel interaction, and dynamic linking raises concerns about its potential for advanced evasion and control capabilities. The strategic use of I/O flags further indicates a deliberate approach to file reading, possibly for data exfiltration or the execution of specific malicious payloads.

REGISTRIES: The malware sample, identified as "1dec94b96of40e3a95e4aaa81f5783c221e4-792d429701701052a03686aeb2cd.exe," exhibits concerning behavior by interacting with the system registry. The operation in question is a "regopenkey" within the registry class, with a result of "reparse" and desired access specified as "query value." The stack trace reveals the following sequence of events:

1. **ntoskrnl.exe Involvement:** The stack begins with interactions in the Windows kernel, specifically ntoskrnl.exe. This suggests that the malware is manipulating core kernel functions to access or modify the registry.
2. **ntdll.dll Interaction:** The stack includes calls to ntdll.dll, a crucial Windows Dynamic Link Library. This dynamic linking indicates the malware's use of advanced system-level functions, possibly for malicious purposes.
3. **Registry Operation Details:** The "regopenkey" operation signifies an attempt by the malware to open a specific registry key, a critical component of the Windows system configuration.
4. **Result: Reparse and Success:** The result being "reparse" indicates that the operation requires further processing, and after reparse, the final result is reported as "success." This indicates that the malware successfully opened the specified registry key.

RED FLAGS AND POTENTIAL THREATS:

- **Unusual Registry Interaction:** The engagement with the registry, especially with a result of "reparse," is uncommon and raises suspicions. This suggests that the malware is employing non-standard techniques, potentially to obfuscate its activities or bypass security measures.

- **Query Value Access:** The desired access of "query value" indicates the malware's interest in retrieving specific values from the registry. This behavior aligns with reconnaissance activities, as the malware may be gathering information about the system configuration.
- **Dynamic Linking with ntdll.dll:** The involvement of ntdll.dll in the stack trace indicates the malware's attempt to leverage advanced system functions. Such dynamic linking is often associated with sophisticated and evasive malware.
- **Potential for Persistence and Control:** Registry manipulation is a common tactic for malware seeking persistence on a compromised system. The ability to open and modify registry keys provides a means for the malware to control its execution and potentially evade detection.

The malware's interaction with the registry, coupled with dynamic linking and the use of non-standard techniques, poses a significant threat. The potential for reconnaissance and the manipulation of critical system settings could lead to unauthorized access, data exfiltration, or further compromise of the system's integrity.

5.4.2 LOGS OF VIRTUAL SANDBOX

After conducting an analysis, it is evident that the malware sample with the process name "1dec94b96of40e3a95e4aaa81f5783c221e4792d429701701052a03686aeb2cd.exe" initiated and concluded its operations both in the virtual and hardware sandboxes. Thus, there is no disparity in the activities performed by the malware in the respective systems. As we can see it in the figure below.

| | | | | | |
|-----------|------------------|------|---|--|---------------------------------------|
| 11:45:... | 1dec94b960f40... | 9368 | Process Start | SUCCESS | Parent PID: 6432, ... |
| 11:45:... | 1dec94b960f40... | 9368 | Thread Create | SUCCESS | Thread ID: 9272 |
| 11:45:... | 1dec94b960f40... | 9368 | QueryNormaliz... C:\Users\asad\Downloads\1dec94b96... | BUFFER OVERFL... | |
| 11:45:... | 1dec94b960f40... | 9368 | QueryNormaliz... C:\Users\asad\Downloads\1dec94b96... | SUCCESS | |
| 11:45:... | 1dec94b960f40... | 9368 | Load Image | SUCCESS | Image Base: 0xce0... |
| 11:45:... | 1dec94b960f40... | 9368 | Load Image | SUCCESS | Image Base: 0x7fe... |
| 11:45:... | 1dec94b960f40... | 9368 | Load Image | SUCCESS | Image Base: 0x7fe... |
| 11:45:... | 1dec94b960f40... | 9368 | Load Image | SUCCESS | Image Base: 0x775... |
| 11:45:... | 1dec94b960f40... | 9368 | CreateFile | C:\Windows\Prefetch\1DEC94B960F4... | NAME NOT FOUND Desired Access: G... |
| 11:45:... | 1dec94b960f40... | 9368 | RegOpenKey | HKLM\System\CurrentControlSet\Contr... | REPARSE Desired Access: Q... |
| 11:45:... | 1dec94b960f40... | 9368 | RegOpenKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS Desired Access: Q... |
| 11:45:... | 1dec94b960f40... | 9368 | RegQueryValue | HKLM\System\CurrentControlSet\Contr... | NAME NOT FOUND Length: 80 |
| 11:45:... | 1dec94b960f40... | 9368 | RegCloseKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS |
| 11:45:... | 1dec94b960f40... | 9368 | RegOpenKey | HKLM\SYSTEM\CurrentControlSet\Con... | REPARSE Desired Access: Q... |
| 11:45:... | 1dec94b960f40... | 9368 | RegOpenKey | HKLM\System\CurrentControlSet\Contr... | NAME NOT FOUND Desired Access: Q... |
| 11:45:... | 1dec94b960f40... | 9368 | RegOpenKey | HKLM\SYSTEM\CurrentControlSet\Con... | REPARSE Desired Access: Q... |
| 11:45:... | 1dec94b960f40... | 9368 | RegOpenKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS Desired Access: Q... |
| 11:45:... | 1dec94b960f40... | 9368 | RegQueryValue | HKLM\System\CurrentControlSet\Contr... | NAME NOT FOUND Length: 24 |
| 11:45:... | 1dec94b960f40... | 9368 | RegCloseKey | HKLM\System\CurrentControlSet\Contr... | SUCCESS |
| 11:45:... | 1dec94b960f40... | 9368 | CreateFile | C:\Windows | SUCCESS Desired Access: E... |
| 11:45:... | 1dec94b960f40... | 9368 | QueryNormaliz... C:\Windows\System32\wow64.dll | SUCCESS | |
| 11:45:... | 1dec94b960f40... | 9368 | Load Image | C:\Windows\System32\wow64.dll | SUCCESS Image Base: 0x7fe... |
| 11:45:... | 1dec94b960f40... | 9368 | ReadFile | C:\Windows\System32\wow64.dll | SUCCESS Offset: 242,176, Le... |
| 11:45:... | 1dec94b960f40... | 9368 | QueryNormaliz... C:\Windows\System32\wow64win.dll | SUCCESS | |
| 11:45:... | 1dec94b960f40... | 9368 | Load Image | C:\Windows\System32\wow64win.dll | SUCCESS Image Base: 0x7fe... |
| 11:45:... | 1dec94b960f40... | 9368 | ReadFile | C:\Windows\System32\wow64win.dll | SUCCESS Offset: 449,024, Le... |
| 11:45:... | 1dec94b960f40... | 9368 | ReadFile | C:\Windows\System32\wow64win.dll | SUCCESS Offset: 323,584, Le... |
| 10:36:... | 1dec94b960f40... | 2760 | RegQueryValue | HKLM\SOFTWARE\Microsoft\Window... | NAME NOT FOUND Length: 20 |
| 10:36:... | 1dec94b960f40... | 2760 | RegCloseKey | HKLM\SOFTWARE\Microsoft\Window... | SUCCESS |
| 10:36:... | 1dec94b960f40... | 2760 | RegOpenKey | HKCU\Software\Microsoft\Windows N... | SUCCESS Desired Access: R... |
| 10:36:... | 1dec94b960f40... | 2760 | RegQueryValue | HKCU\SOFTWARE\Microsoft\Window... | NAME NOT FOUND Length: 20 |
| 10:36:... | 1dec94b960f40... | 2760 | RegCloseKey | HKCU\SOFTWARE\Microsoft\Window... | SUCCESS |
| 10:36:... | 1dec94b960f40... | 2760 | CloseFile | C:\Windows\apppatch\sysmain.sdb | SUCCESS |
| 10:36:... | 1dec94b960f40... | 2760 | QueryBasicInfor... | C:\Users\asad\AppData\Local\Temp\... | SUCCESS CreationTime: 5/15... |
| 10:36:... | 1dec94b960f40... | 2760 | RegOpenKey | HKLM\Software\WOW6432Node\Micr... | SUCCESS Desired Access: R... |
| 10:36:... | 1dec94b960f40... | 2760 | RegSetInfoKey | HKLM\SOFTWARE\WOW6432Node\... | SUCCESS KeySetInformation... |
| 10:36:... | 1dec94b960f40... | 2760 | RegQueryValue | HKLM\SOFTWARE\WOW6432Node\... | NAME NOT FOUND Length: 20 |
| 10:36:... | 1dec94b960f40... | 2760 | RegCloseKey | HKLM\SOFTWARE\WOW6432Node\... | SUCCESS |
| 10:36:... | 1dec94b960f40... | 2760 | CreateFile | C:\Windows\apppatch\sysmain.sdb | SUCCESS Desired Access: G... |
| 10:36:... | 1dec94b960f40... | 2760 | QueryStandardI... | C:\Windows\apppatch\sysmain.sdb | SUCCESS AllocationSize: 4.0... |
| 10:36:... | 1dec94b960f40... | 2760 | QueryStandardI... | C:\Windows\apppatch\sysmain.sdb | SUCCESS AllocationSize: 4.0... |
| 10:36:... | 1dec94b960f40... | 2760 | CreateFileMapp... | C:\Windows\apppatch\sysmain.sdb | FILE LOCKED WI... SyncType: SyncTy... |
| 10:36:... | 1dec94b960f40... | 2760 | QueryStandardI... | C:\Windows\apppatch\sysmain.sdb | SUCCESS AllocationSize: 4.0... |
| 10:36:... | 1dec94b960f40... | 2760 | CreateFileMapp... | C:\Windows\apppatch\sysmain.sdb | SUCCESS SyncType: SyncTy... |
| 10:36:... | 1dec94b960f40... | 2760 | CloseFile | C:\Windows\apppatch\sysmain.sdb | SUCCESS |
| 10:36:... | 1dec94b960f40... | 2760 | CloseFile | C:\Users\asad\AppData\Local\Temp\... | SUCCESS |
| 10:37:... | 1dec94b960f40... | 2760 | Thread Exit | SUCCESS | Thread ID: 2424, ... |

Figure 5.28: Consistent Malicious Operations Across Different Environments.

6

Conclusion

This thesis introduces a pioneering concept for constructing hardware-supported sandboxes, leveraging adaptable hardware. Specifically, the implementation incorporates Leetdma for memory dump, Reboot Restore RX software for system restoration after malware execution, and a KVM switch for seamless control between the host and guest environments. The primary focus is on establishing the architectural requirements for a system virtual machine environment that executes guest machines on dedicated physical hardware within the Leetdma framework. A proof of concept has been implemented, showcasing the application of these architectural requirements to construct an effective hardware-supported sandbox.

A notable advantage of this approach lies in its inherent design, preventing stealth-malware from detecting virtual execution and breaking out of the virtual machine/sandbox. The thesis delves into the theoretical advantages, emphasizing the need for further exploration to validate its effectiveness. Since the system executes malware on actual hardware, it remains resilient to VM/emulation-based detection attacks. The evaluation demonstrates successful monitoring of true malicious behavior in VM/emulation-aware malware samples that remained undetected in emulated or virtualized environments. Following each analysis, the system efficiently restores the system using Reboot Restore RX, ensuring readiness for subsequent analyses.

In conclusion, this thesis presents a hardware-supported sandboxing approach, integrated

with Leetdma, Reboot Restore RX, and a KVM switch, offering a robust solution for malware analysis. By executing malware on dedicated hardware, the system enhances security by preventing detection evasion tactics. The methodology proves effective in uncovering the genuine behavior of sophisticated malware samples, showcasing its potential as a valuable tool in cybersecurity practices.

6.1 LIMITATION

In a virtual sandbox, the host system maintains control over resource allocation, including memory and processors. The proposed concept in this thesis has limitations, as it operates within fixed parameters for memory and processing power. The host's intervention is restricted unless FPGAs are employed to dynamically adjust attached storage and processing capabilities based on the environment. However, for research purposes, a single central processing unit suffices. Unlike the dynamic resource management in a virtual sandbox, where the host system governs resource distribution among virtualized instances using hypervisors or sandboxing software, the thesis's approach faces constraints due to predetermined resource settings. These limitations highlight the need for further exploration and adaptation in addressing varying environmental demands.

References

- [1] “Gen-V Cyber Security - Check Point Software — checkpoint.com,” <https://www.checkpoint.com/pages/gen-v-cyber-security/>, [Accessed 26-10-2023].
- [2] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee, “Polyunpack: Automating the hidden-code extraction of unpack-executing malware,” 01 2007, pp. 289 – 300.
- [3] A. F. Abdul kadir, N. Stakhanova, and A. Ghorbani, “Understanding android financial malware attacks: Taxonomy, characterization, and challenges,” vol. 7, pp. 1–52, 01 2018.
- [4] S. Tomonaga, “Attack Convincing Users to Download a Malware-Containing Shortcut File - JPCERT/CC Eyes — blogs.jpccert.or.jp,” <https://blogs.jpccert.or.jp/en/2019/06/darkhotel-1nk.html>, [Accessed 27-10-2023].
- [5] “LeetDMA (incl. Custom FW) — enigma-x1.com,” <https://enigma-x1.com/store/product/23-leetdma-incl-custom-fw/>, [Accessed 27-10-2023].
- [6] “Artix-7 block diagram,” <https://www.avnet.com/wps/portal/emea/>, [Accessed 27-10-2023].
- [7] <https://forum.huawei.com/enterprise/en/iops-and-throughput/thread/667286038160228352-667213859733254144>, [Accessed 27-10-2023].
- [8] “Computer data storage - Wikipedia — en.wikipedia.org,” https://en.wikipedia.org/wiki/Computer_data_storage, [Accessed 27-10-2023].
- [9] “MalwareBazaar | Malware sample exchange — bazaar.abuse.ch,” <https://bazaar.abuse.ch/>, [Accessed 03-11-2023].
- [10] “AMD Adaptive Computing Documentation Portal — docs.xilinx.com,” https://docs.xilinx.com/v/u/en-US/ds180_7Series_Overview, [Accessed 27-10-2023].

- [11] V. Srivastava and V. Sharma, "Sandbox technology in a web security environment: A hybrid exploration of proposal and enactment*****_*****," 06 2022.
- [12] M. Maass, A. Sales, B. Chung, and J. Sunshine, "A systematic analysis of the science of sandboxing," *PeerJ Computer Science*, vol. 2, p. e43, 01 2016.
- [13] S. Miwa, T. Miyachi, M. Eto, M. Yoshizumi, and Y. Shinoda, "Design issues of an isolated sandbox used to analyze malwares," vol. 4752, 10 2007, pp. 13–27.
- [14] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, and B. Shastry, "Towards taming privilege-escalation attacks on android." in *NDSS*, vol. 17, 2012, p. 19.
- [15] "Virtualization/Sandbox Evasion - How Attackers Avoid Malware Analysis — picussecurity.com," <https://www.picussecurity.com/resource/virtualization/sandbox-evasion-how-attackers-avoid-malware-analysis>, [Accessed 19-11-2023].
- [16] "Virtualization/Sandbox Evasion, Technique T1497 - Enterprise | MITRE ATT&CK; — attack.mitre.org," <https://attack.mitre.org/techniques/T1497/>, [Accessed 19-11-2023].
- [17] M. A. Ajay Kumara and C. D. Jaidhar, "Vmi based automated real-time malware detector for virtualized cloud environment," in *Security, Privacy, and Applied Cryptography Engineering*, C. Carlet, M. A. Hasan, and V. Saraswat, Eds. Cham: Springer International Publishing, 2016, pp. 281–300.
- [18] S. Zhang, X. Meng, L. Wang, L. Xu, and X. Han, "Secure virtualization environment based on advanced memory introspection," *Security and Communication Networks*, vol. 2018, pp. 1–16, 03 2018.
- [19] H. Jazi, "New AgentTesla variant steals WiFi credentials | Malwarebytes Labs — malwarebytes.com," <https://www.malwarebytes.com/blog/news/2020/04/new-agenttesla-variant-steals-wifi-credentials>, [Accessed 19-11-2023].
- [20] C. Nocturnus, "A Bazar of Tricks: Following Team9's Development Cycles — cybereason.com," <https://www.cybereason.com/blog/research/a-bazar-of-tricks-following-team9s-development-cycles>, [Accessed 19-11-2023].

- [21] “Bisonal: 10 years of play — blog.talosintelligence.com,” <https://blog.talosintelligence.com/bisonal-10-years-of-play/>, [Accessed 19-11-2023].
- [22] “This isn’t Optimus Prime’s Bumblebee but it’s Still Transforming | Proofpoint US — proofpoint.com,” <https://www.proofpoint.com/us/blog/threat-insight/bumblebee-is-still-transforming>, [Accessed 19-11-2023].
- [23] “Wayback Machine — web.archive.org,” <https://web.archive.org/web/20151022204649/https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/rpt-apt28.pdf>, [Accessed 19-11-2023].
- [24] <https://www.vmware.com/solutions/nsx-firewall.html>, [Accessed 19-11-2023].
- [25] https://web-assets.esetstatic.com/wls/2021/06/eset_gelsemium.pdf, [Accessed 19-11-2023].
- [26] “Hancitor (AKA Chanitor) observed using multiple attack approaches | Mandiant — mandiant.com,” <https://www.mandiant.com/resources/blog/hancitor-aka-chanit>, [Accessed 19-11-2023].
- [27] “vblocalhost.com,” <https://vblocalhost.com/uploads/VB2021-Kayal-et-al.pdf>, [Accessed 19-11-2023].
- [28] C. Erlich, “The Avast Abuser: Metamorfo Banking Malware Hides By Abusing Avast Executable — chenerlich,” <https://medium.com/@chenerlich/the-avast-abuser-metamorfo-banking-malware-hides-by-abusing-avast-executable-ac9b8b392767>, [Accessed 19-11-2023].
- [29] U. 42, “Russia’s Gamaredon aka Primitive Bear APT Group Actively Targeting Ukraine — unit42.paloaltonetworks.com,” <https://unit42.paloaltonetworks.com/gamaredon-primitive-bear-ukraine-update-2021/>, [Accessed 19-11-2023].
- [30] T. Raffetseder, C. Krügel, and E. Kirda, “Detecting system emulators,” in *Information Security Conference*, 2007. [Online]. Available: <https://api.semanticscholar.org/CorpusID:12025670>
- [31] J. RUTKOWSKA, “Red pill : Detect vmm using (almost) one cpu instruction,” <http://invisiblethings.org/papers/redpill.html>, 2004.

- [32] D. Kirat, G. Vigna, and C. Kruegel, “Barebox: Efficient malware analysis on bare-metal,” in *Proceedings of the 27th Annual Computer Security Applications Conference*, ser. ACSAC '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 403–412. [Online]. Available: <https://doi.org/10.1145/2076732.2076790>
- [33] M. Eckert, J. Haase, D. Meyer, and B. Klauer, “Architectural requirements for constructing hardware supported sandboxes,” in *2016 International Conference on FPGA Reconfiguration for General-Purpose Computing (FPGA4GPC)*, 2016, pp. 37–42.
- [34] J. Smith and R. Nair, *Virtual Machines: Versatile Platforms for Systems and Processes (The Morgan Kaufmann Series in Computer Architecture and Design)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.
- [35] —, *Virtual Machines: Versatile Platforms for Systems and Processes (The Morgan Kaufmann Series in Computer Architecture and Design)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.
- [36] A. P. Namanya, A. Cullen, I. Awan, and J. Pagna Diss, “The world of malware: An overview,” 09 2018.
- [37] A. Jillepalli, D. Conte de Leon, and J. Alves-Foss, “Operational characteristics of modern malware: Pco threats,” 04 2018.
- [38] M. Garetto, W. Gong, and D. Towsley, “Modeling malware spreading dynamics.” vol. 3, 04 2003.
- [39] Z. Chen, G. LX, and K. Kwiat, “Modeling the spread of active worms,” vol. 3, 01 2003, pp. 1890 – 1900 vol.3.
- [40] M. Ramilli and M. Bishop, “Multi-stage delivery of malware,” 11 2010, pp. 91 – 97.
- [41] M. Agrawal, H. Singh, N. Gour, and M. A. Kumar, “Evaluation on malware analysis,” *International Journal of Computer Science and Information Technologies*, vol. 5, no. 3, pp. 3381–3383, 2014.
- [42] W. Yan, Z. Zhang, and N. Ansari, “Revealing packed malware,” *iee seCurity & PrivaCy*, vol. 6, no. 5, pp. 65–69, 2008.

- [43] “Horizon DataSys Horizon DataSys Corporation — horizondatasys.com,” <https://horizondatasys.com/>, [Accessed 30-10-2023].
- [44] R. Stair, F. Moisiadis, R. Genrich, and G. Reynolds, *Principles of information systems*. Cengage Learning Australia, 2011.
- [45] “Can ransomware infect NAS drives and other devices? — xcitium.com,” <https://www.xcitium.com/blog/ransomware/can-ransomware-infect-nas-drive/>, [Accessed 28-10-2023].
- [46] “GitHub - ufrisk/pcileech: Direct Memory Access (DMA) Attack Software — github.com,” <https://github.com/ufrisk/pcileech>, [Accessed 03-11-2023].

Acknowledgments

I would like to thank my supervisor Alessandro Brighente and my co-supervisor Sebastian Biedermann for their continued support.