Arab American University
Faculty of Graduate Studies
Department of Natural, Engineering and Technology Sciences
Master Program in Cyber Security

Enhanced Windows Sandbox: A Unified Framework for Stealth-Based, Automated Malware Analysis with Centralized Logging

Mohammad Yousef Houssein Salah
202113305

Supervision Committee:

Dr. Islam Amro
Dr. Ahmad Hasasneh
Dr. Nael Abu-halaweh

This Thesis Was Submitted in Partial Fulfillment of the Requirements for the Master Degree in Cyber Security

Palestine, 07 / 2025

Arab American University
Faculty of Graduate Studies
Department of Natural, Engineering and Technology Sciences
Master Program in Cyber Security

Thesis Approval
Enhanced Windows Sandbox: A Unified Framework for Stealth-Based, Automated Malware Analysis with Centralized Logging

Mohammad Yousef Hussein Salah
202113305

This thesis was defended successfully on 28 / 7 / 2025 and approved by:

Thesis Committee Members:

Name	Title	Signature
1. Dr. Islam Amro	Main Supervisor	
2. Dr. Ahmad Hasasneh	Members of Supervision Committee	
3. Dr. Nael Abu-halaweh	Members of Supervision Committee	

Palestine, 7/ 2025

Declaration

I declare that, except where explicit reference is made to the contribution of others, this thesis is substantially my own work and has not been submitted for any other degree at the Arab American University or any other institution.

Student Name: Mohammad Yousef Houssein Salah
Student ID: 202113305
Signature: Mohammad Salah
Date of Submitting the Final Version of the Thesis: 16/09/2025

Dedication

To Allah, the Most Gracious, the Most Merciful — all praise and gratitude are for Him, without whom none of this would be possible.

I dedicate this work to my beloved parents, whose unconditional love, wisdom, and prayers have always been my guiding light.

To my wife — your patience, sacrifice, and unwavering support gave me strength throughout this journey.

To my precious daughter, whose presence inspires me to strive for excellence and integrity in all that I do.

Mohammad Yousef Houssein Salah

Acknowledgements

In the name of Allah, the Most Gracious, the Most Merciful.
All praise and gratitude are due to Allah, whose guidance and blessings have made this journey possible and meaningful. Without His will, none of this would have been achieved.

I would like to express my sincere and profound gratitude to the many individuals who have supported me throughout my master's journey and the completion of this thesis. Without their guidance, expertise, and encouragement, this work would not have been possible.

First and foremost, my deepest appreciation goes to my supervisor, Dr. Islam Amro, whose insightful guidance, critical feedback, and unwavering patience were instrumental in shaping the direction and quality of this research. His expertise in cybersecurity challenged me to think more deeply, and his mentorship provided the structure I needed to navigate the complexities of this project. I am truly grateful for the time and energy he invested in my academic development.

I am especially indebted to Dr. Majde Odeh, whose support and recommendation were the key reason I was accepted into this university. His belief in my potential gave me the opportunity to pursue this path, and for that, I am profoundly thankful.

Most personally, I wish to thank my beloved family. To my father and mother—your unconditional love, prayers, and belief in me have been my guiding light. To my dear wife, your patience, and unwavering support gave me the strength to persevere. To my brothers and sisters—your encouragement and presence in my life have always uplifted me. And to my daughter Eil@, your presence reminded me why this journey matters and inspired me every day to keep going.

I also extend my heartfelt gratitude to my manager; my dear friends Tom, Tetos, company, motse, sawstr1ke, Amr, Dr.@binaryz0ne, and Cold z3ro; my colleagues and team—for their support, motivation, and shared passion throughout this journey. Your encouragement and camaraderie have meant more to me than words can express.
(This is not a wanted list. All names are legally present—well, maybe not in certain firewalls!)

To all of you—thank you.

Enhanced Windows Sandbox: A Unified Framework for Stealth-Based, Automated Malware Analysis with Centralized Logging

Mohammad Yousef Houssein Salah

Dr. Islam Amro

Dr. Ahmad Hasasneh

Dr. Nael Abu-halaweh

Abstract

The escalating sophistication of malicious software, particularly its use of advanced evasion techniques, poses a significant and persistent challenge to modern cybersecurity. Standard analysis environments, or sandboxes, are often defeated by malware that can detect their virtualized nature, thus hiding its true behavior and undermining analysis. While Microsoft Windows Sandbox offers a promising lightweight, OS-integrated platform, its default configuration is highly detectable and lacks the automation and deep visibility required for rigorous malware analysis.

This thesis addresses these deficiencies through the design, implementation, and validation of the "Sandbox Enhancer," a novel system that transforms Windows Sandbox into a stealthy and efficient malware analysis platform. The methodology involved three core thrusts: (1) a systematic identification and modification of sandbox artifacts to enhance environmental stealth; (2) the development of a fully automated workflow to manage the entire analysis lifecycle from setup to reporting; and (3) the integration of a comprehensive logging framework using Sysmon, NxLog, and Graylog to provide deep behavioral visibility. The final system was empirically validated through a large-scale experiment involving the analysis of 1,700 unique malware samples, each pre-identified as "evasive" for having defeated a commercial-grade sandbox.

The experimental results demonstrate the profound effectiveness of the "Sandbox Enhancer" system. It successfully compelled 47.8% of these highly evasive samples to execute their malicious payloads, revealing critical behaviors such as network command-and-control (C2) communications and local ransomware activities. The analysis also identified the limitations of the current system, noting that malware employing low-level architectural checks (e.g., for disk capacity or hypervisor timing) could still achieve evasion.

This research makes a significant contribution by providing a validated, cost-effective, and scalable framework for analyzing evasive malware. It offers a practical model for hardening lightweight virtualization environments and presents a powerful tool for security practitioners and researchers. The findings confirm that through systematic enhancement of

stealth, automation, and observability, the utility of platforms like Windows Sandbox can be dramatically elevated, strengthening our collective capabilities in the ongoing fight against cyber threats.

In addition to the large-scale experiment, a controlled head-to-head baseline on 200 samples compared the default and enhanced sandboxes, yielding a higher detection rate for the enhanced variant (43.5% vs. 30.5%; Δ=+13.0 p.p.; McNemar $p \approx 0.007$)

Keywords: Malware Analysis, Automated Workflow, Windows Sandbox, Cybersecurity, Threat Intelligence.

Table of Contents

#	Title	Page

List of Tables

Table #	Title of Table	Page

List of Figures

Figure #	Title of Figure	Page

Chapter One: Introduction to the Study

1.1. Introduction

The rapid expansion of digital technology has become central to modern life, affecting everything from global commerce to daily communication. This great reliance on digital systems creates a serious need for strong computer security. A major and growing threat in this area is malicious software, known as malware. Malware is designed by attackers to damage systems, steal private information, or disrupt important services. The amount and complexity of new malware grows every day, making it a constant challenge for cybersecurity professionals.

To fight against these threats, we must be able to analyze malware safely and effectively. Dynamic malware analysis, where a suspicious program is run in a controlled environment to observe its behavior, is a very important method. A key tool for this is a "sandbox." A sandbox is an isolated, secure space where malware can be executed without any risk to the main computer system.

Microsoft Windows Sandbox is a promising tool for this purpose. It is lightweight, fast, and integrated directly into the Windows operating system. However, it was designed for general use, not for specialized malware analysis. Therefore, it has significant limitations. This research will investigate these limitations and propose a system to make Windows Sandbox a much more powerful and reliable tool for cybersecurity experts. The study will focus on enhancing its stealth to avoid detection by advanced malware and automating the analysis process to make it more efficient.

1.2. Significance of the Study

This research is significant for both practical and academic reasons.

First, it has **practical significance** for the cybersecurity community. Security analysts, incident response teams, and threat researchers need effective tools to analyze new threats quickly. This study aims to develop an enhanced system that makes Windows Sandbox more useful for them. By improving its stealth and automating the workflow, the proposed "Sandbox Enhancer" system can help professionals analyze malware more accurately and efficiently. This can lead to faster detection of threats, better protection for organizations, and a reduction in the time and resources spent on manual analysis.

Second, the research has **academic significance**. While there is much research on general sandbox technologies like Cuckoo or VMware-based systems, there is a knowledge gap regarding the specific capabilities and vulnerabilities of Microsoft Windows Sandbox for malware analysis. This study contributes to the academic literature by:

- Providing a systematic analysis of Windows Sandbox's detectable "artifacts."
- Presenting and evaluating new methods for artifact modification to improve sandbox stealth.
- Offering a new, automated framework that can be a model for future research in lightweight virtualization environments.

1.3. The Research Problem

Despite its potential, using the default Windows Sandbox for serious malware analysis presents three main problems that this research will address:

1. **High Detectability of the Sandbox Environment:** Sophisticated malware is often designed with "evasion techniques." It checks its environment for clues, or "artifacts," that indicate it is running inside a sandbox. Windows Sandbox has many such artifacts, including specific registry keys (e.g., related to Hyper-V), generic hardware identifiers (e.g., virtual disk names, MAC addresses), and default system configurations (e.g., the username WDAGUtilityAccount). When malware detects

these artifacts, it may shut down, delay its malicious actions, or show only benign behavior. This makes the analysis useless, as the true nature of the malware is never revealed.

2. **Lack of an Automated Workflow:** The process of using Windows Sandbox for analysis is mostly manual. An analyst must start the sandbox, manually copy over the malware sample and analysis tools, run them, and then manually collect the results. This process is slow, repetitive, and not scalable for analyzing the large number of new malware samples that appear daily. This lack of automation is a major barrier to operational efficiency.
3. **Insufficient Logging and Visibility:** The default Windows Sandbox does not provide detailed, centralized logging. To understand what malware is doing, analysts need to see detailed information about process creation, network connections, file system changes, and registry modifications. Without an integrated system to capture and analyze this data, it is difficult to get a complete picture of the malware's behavior.

These three limitations together make the standard Windows Sandbox an unreliable tool for analyzing advanced, evasive malware.

1.4. The Research Objectives

To solve the problems described above, this research has the following primary objectives:

1. To systematically identify the detectable artifacts in Windows Sandbox and develop effective modification techniques to enhance its stealth against evasive malware.
2. To design and implement the "Sandbox Enhancer," an automated system that manages the entire analysis workflow, from environment setup and artifact modification to sample execution and data collection.

3. To integrate advanced logging tools (Sysmon, NxLog) with a centralized analysis platform (Graylog) to provide deep visibility into malware behavior inside the sandbox.
4. To empirically evaluate the effectiveness of the enhanced system by testing it against real-world malware samples known to use evasion techniques.

1.5. The Research Questions

This study will be guided by the following research questions:

1. What are the specific system artifacts within Windows Sandbox that can be fingerprinted and exploited by malware to evade analysis?
2. How can these artifacts be modified or obfuscated to make the Windows Sandbox environment stealthier without compromising its stability?
3. How can an automated workflow be designed to integrate artifact modification, malware execution, and comprehensive log collection to improve the efficiency and scalability of analysis?
4. To what extent does the enhanced, modified Windows Sandbox improve the ability to analyze evasive malware compared to its default, unmodified state?

1.6. Study Hypotheses

This research will test the following hypotheses:

- **H1:** Modifying key system artifacts within Windows Sandbox (e.g., registry values, hardware identifiers, usernames) will significantly reduce its detectability by malware that employs common evasion checks.

- **H2:** The implementation of the automated "Sandbox Enhancer" system will lead to a measurable reduction in the time and manual steps required to analyze a malware sample compared to the manual workflow.
- **H3:** The enhanced sandbox environment will successfully compel evasive malware to execute its full malicious payload, revealing behaviors and Indicators of Compromise (IOCs) that remain hidden in the default sandbox.

1.7. Limitations of the Study

This study acknowledges the following limitations:

- The research will be conducted using a specific set of publicly available malware samples known for evasion. The findings may not be generalizable to all malware families or to future, unknown evasion techniques.
- Certain system artifacts, especially those embedded at the hardware or BIOS level, are extremely difficult to modify within the architectural constraints of Windows Sandbox. This research will focus on artifacts that are practically modifiable.
- The study is focused exclusively on enhancing Microsoft Windows Sandbox and does not perform a comparative analysis with other sandboxing solutions like Cuckoo Sandbox or commercial enterprise platforms.
- The field of malware evasion is constantly evolving. The specific artifact modifications developed in this research may become less effective over time as new detection methods are created by malware authors.

1.8. Conceptual and Procedural Definitions

This section explains some of the key words and terms used in this thesis. Understanding these terms will help the reader follow the ideas more easily.

- **Sandbox**: A sandbox is a safe, disposable virtual environment where we can run suspicious files like malware without affecting the real system. It is used to test and study how the malware behaves.
- **Artifact**: is a clue or sign that shows the system is running inside a virtual machine or sandbox. Malware can look for these signs to know if it is being watched, and if so, it may hide its actions.
- **Malware Evasion**: means the tricks and techniques that malware uses to avoid being detected or analyzed. For example, the malware may check if it is inside a sandbox and then stop running if it thinks it is.
- **Automation**: means using scripts and programs to do tasks automatically instead of manually. In this thesis, automation is used to set up the sandbox, run the analysis, and collect the results without human help.
- **Logging Pipeline**: is a system that collects, sends, and stores logs (records of activity) from the sandbox. These logs help us see what the malware did, like which files it touched or which websites it tried to contact.

1.9. Thesis Structure

This thesis is organized into six chapters:

- **Chapter One:** Introduces the study, outlines the research problem, significance, objectives, questions, hypotheses, and limitations.
- **Chapter Two:** Presents a Literature Review of existing research on malware analysis, sandboxing technologies, malware evasion techniques, and countermeasures.
- **Chapter Three:** Details the Methodology used in this research, including the process for artifact identification, the architectural design of the "Sandbox Enhancer" system, and the plan for experimental evaluation.

- **Chapter Four:** Presents the Results and Analysis of the experiments, evaluating the effectiveness of the artifact modifications and the automated workflow.
- **Chapter Five:** Provides a Discussion of the research findings, their implications for the cybersecurity field, and how they address the gaps in existing knowledge.
- **Chapter Six:** Offers the Conclusion, summarizing the key contributions of the thesis, reiterating its significance, and suggesting directions for future research.

Chapter Two: Literature Review: Navigating the Terrain of Malware Analysis and Sandbox Technologies

2.1 Introduction

The contemporary digital world is defined by an ongoing and escalating battle between cybersecurity defenders and malicious actors. The primary weapon used by these actors is malware, or malicious software, which represents a persistent threat to data integrity, personal privacy, and the operational stability of critical infrastructures worldwide (Symantec, 2023). The sheer volume and increasing sophistication of malware, with hundreds of thousands of new samples appearing daily, make its detection and analysis a central challenge in computer security (SANS Institute, 2023). To develop effective defenses, it is not enough to simply block known threats; we must understand how new and unknown malware operates. This requires a deep and systematic process of malware analysis.

Malware analysis is the study of a malware sample to understand its origin, functionality, and potential impact (Sikorski & Honig, 2012). The knowledge gained from this process is vital for many cybersecurity activities, including developing antivirus signatures, creating rules for intrusion detection systems, informing incident response procedures, and extracting threat intelligence to predict future attacks. Over the years, the methods for malware analysis have evolved significantly in response to the changing tactics of malware authors.

This chapter provides a comprehensive review of the existing literature on malware analysis. It begins by outlining the fundamental techniques used by analysts, primarily static and dynamic analysis, and discusses their respective strengths and limitations. It then focuses on the technology that is central to modern dynamic analysis: sandboxing. The

chapter will explore the architectural principles of various sandboxing frameworks, with a special focus on the unique design of Microsoft Windows Sandbox.

A critical part of this review is dedicated to the adversarial nature of malware analysis. Malware is not a passive subject of study; it is actively designed to resist analysis. The chapter will therefore provide a detailed examination of the many evasion techniques that malware employs to detect and bypass sandboxes (Afianian et al., 2019). In response to these evasion tactics, researchers have developed countermeasures, including methods for artifact modification and environment emulation, which will also be discussed. Finally, the chapter will survey the growing role of machine learning in malware detection and identify the critical gaps in current research. These gaps—particularly concerning the stealth, automation, and observability of analysis environments—directly motivate the work presented in this thesis. By mapping the current landscape, this literature review establishes the necessary context and justification for the development of the "Sandbox Enhancer" system.

2.2 Malware Analysis Methodologies: A Foundational Overview

The discipline of malware analysis is broadly divided into two main approaches: static analysis and dynamic analysis. Each method offers a different perspective on the malware's characteristics and behavior. While they can be used independently, they are most powerful when used together in a hybrid approach to gain a complete understanding of the threat (Sihwail et al., 2018).

Static analysis involves examining a malware file without actually running it. The goal is to understand the program's structure, code, and potential functionality by inspecting its binary components. This is the first step in most analysis processes because it is safe—since the code is not executed, there is no risk of infecting the analyst's machine (Moser et al., 2007).

Analysts use a variety of tools to perform static analysis. Disassemblers like IDA Pro or the open-source Ghidra from the NSA are used to translate the machine code of the executable into human-readable assembly language, allowing an analyst to trace the program's logic (National Security Agency, 2019). Other tools like Pestudio or Radare2 examine the Portable Executable (PE) file format, which is the standard for executables and DLLs on Windows (Sabanal, 2014). By inspecting the PE header, analysts can find valuable information such as the functions the malware imports from system libraries (e.g., functions for network communication or file manipulation), strings embedded in the binary (which might reveal IP addresses, domain names, or error messages), and other metadata that can provide clues about the malware's purpose (Wang et al., 2009).

The primary advantage of static analysis is that it provides a complete view of all the code within the executable, including malicious routines that might not be triggered during a single dynamic analysis session. However, static analysis has significant limitations. Malware authors are well aware of these techniques and employ countermeasures to make static analysis difficult or impossible. The most common countermeasure is **obfuscation**, which involves making the code intentionally confusing and hard to understand (O'Kane et al., 2011). Another common technique is **packing**, where the malicious executable is compressed or encrypted. The packed file contains a small "unpacker" stub. When the file is run, this stub decompresses or decrypts the original malicious code in memory and then executes it. Static analysis of the packed file reveals only the unpacker stub, not the actual malicious payload (You & Yim, 2010). These limitations mean that static analysis alone is often not enough to fully understand a modern malware sample.

To overcome the limitations of static analysis, security professionals rely on dynamic analysis. This method involves running the malware in a safe, controlled environment and observing its behavior in real time. The primary tool for dynamic analysis is the **sandbox**, which is an isolated system (often a virtual machine) where the malware can be executed without harming the host system or the wider network (Egele et al., 2008).

During dynamic analysis, the sandbox is instrumented with monitoring tools that log all of the malware's actions. This includes:

- **File System Changes:** Creating, deleting, or modifying files.
- **Registry Modifications:** Creating or changing registry keys, which is a common way for malware to establish persistence (i.e., ensure it runs again after a system reboot).
- **Process Activity:** Launching new processes, injecting code into other legitimate processes, or terminating security-related services.
- **Network Communications:** Attempting to connect to external IP addresses or domain names, which may be Command and Control (C2) servers used by the attacker to send commands or exfiltrate stolen data.

The main advantage of dynamic analysis is its ability to see the malware's true behavior, even if the code is packed or obfuscated. The malware must eventually unpack itself in memory to run, at which point its actions can be observed (Or-Meir et al., 2019). Automated dynamic analysis systems, such as the open-source Cuckoo Sandbox, have become essential tools in security operations centers (SOCs) for quickly processing large numbers of suspicious files (Guarnieri & Bremer, 2012).

However, dynamic analysis is also not a perfect solution. Its greatest weakness is that sophisticated malware is designed to detect when it is running inside a sandbox. If the malware identifies the analysis environment, it may alter its behavior to evade detection, a topic that will be explored in detail in Section 2.4. Furthermore, a single dynamic analysis run might not execute all of the malware's code. For example, a malicious routine might only be triggered on a specific date, after a certain amount of time has passed, or in response to a specific command from its C2 server. If these conditions are not met during the analysis period, the malicious behavior will not be observed (Sikorski & Honig, 2012).

Given the complementary strengths and weaknesses of static and dynamic analysis, a modern best practice is to use a **hybrid analysis** approach. This involves combining both techniques to build a more complete picture of the malware (Shijo & Salim, 2015).

For example, static analysis might be used first to identify if a file is packed. If it is, the analyst knows that dynamic analysis will be necessary to observe the unpacked code. During dynamic analysis, the analyst might observe the malware unpacking its payload into a specific region of memory. The analyst can then take a "snapshot" of that memory region and perform static analysis on the now-unpacked code. This allows the analyst to see the full logic of the malicious payload, which was hidden in the original file. This iterative process of using insights from one technique to inform the other is at the heart of effective hybrid analysis (Ijaz et al., 2019).

2.3 The Role of Sandboxing in Dynamic Malware Analysis

Sandboxing is the foundational technology that makes dynamic malware analysis possible. A sandbox provides an isolated and instrumented environment where potentially malicious code can be executed and observed safely. The concept is to create a "box" made of "sand" from which the malware cannot escape to affect the real system. The effectiveness of a sandbox depends on the trade-offs it makes between isolation, performance, and stealth.

Sandboxing technologies can be implemented using several different architectural approaches.

Full System Emulation and Virtual Machine-Based Sandboxes: The most common approach is to use a full virtual machine (VM) as the sandbox. Technologies like VMware, VirtualBox, or KVM create a complete virtualized hardware environment, on top of which a full guest operating system (e.g., Windows 10) is installed. This provides a very high degree of isolation, as the malware is running in a completely separate virtual machine, managed by a hypervisor. Open-source frameworks like **Cuckoo Sandbox** are built on this

model, using VMs to run malware samples and a host-side agent to manage the analysis and collect results (Willems et al., 2007). The primary drawback of VM-based sandboxes is that they have a noticeable performance overhead and, more importantly, they introduce many artifacts (e.g., specific virtual hardware drivers, registry keys, MAC addresses) that malware can easily detect (Vasilescu et al., 2014) .

Cloud-Based Sandbox Services: Many security vendors now offer cloud-based sandboxing services like **ANY.RUN** or **Hybrid Analysis**. These services maintain a large infrastructure of sandboxes (usually VM-based) and allow analysts to submit samples through a web interface. They offer significant advantages in scalability and convenience, as the analyst does not need to maintain their own sandbox infrastructure. Many of these services also provide an interactive mode, allowing the analyst to interact with the malware in real time (e.g., by clicking buttons or entering text) to trigger behaviors that would not occur in a fully automated analysis (ANY.RUN, 2019).

Container-Based Isolation: Technologies like Docker offer a more lightweight form of isolation known as containerization. Containers share the host operating system's kernel but have their own isolated process space and file system. While primarily used for application deployment, containers can be used for malware analysis, especially for Linux-based threats. They offer much faster startup times and lower resource usage than full VMs, but the shared kernel architecture presents a larger potential attack surface if the malware is able to "escape" the container.

Microsoft introduced **Windows Sandbox** as a feature in Windows 10 and 11 Pro and Enterprise editions. It provides a lightweight, disposable desktop environment for safely running untrusted applications (Microsoft, 2019). Windows Sandbox uses a hybrid approach, leveraging Microsoft's Hyper-V technology for strong, hardware-based isolation, but it also incorporates technologies from Windows Containers to achieve efficiency and speed.

Its key architectural features include:

- **Integration with the Host OS:** Unlike a traditional VM that requires its own large virtual disk file, Windows Sandbox dynamically generates its system image using the clean OS files that are already on the host machine. This greatly reduces its disk footprint and ensures that the sandbox is always up-to-date with the host's operating system.
- **Dynamic Image Generation:** As detailed in deep architectural research by Check Point, Windows Sandbox uses a system of layered virtual disks (VHDx) and a special file system filter driver (wcifs.sys). This allows it to link to the host's system files instead of duplicating them, which is the key to its fast startup time (Ilgayev, 2021).
- **Kernel Isolation:** It uses a separate copy of the kernel, ensuring that if malware compromises the sandbox's kernel, the host system's kernel remains safe.
- **Disposability:** Everything within the sandbox, including any files created or changes made by malware, is permanently deleted when the sandbox is closed. This guarantees that every session starts with a pristine, clean environment.

While these features make Windows Sandbox an attractive tool for quick tests, they also create a unique set of artifacts. The specific way it shares host files, the virtual devices it presents via Hyper-V, and its default configurations can be detected by malware specifically designed to look for them. One study has even shown how Windows Sandbox can be used as an *anti-forensics* tool by attackers, allowing them to run their tools within an isolated environment on a compromised machine to hide their activities from host-based monitoring (Mohammed Yousuf uddin et al., 2024). This highlights the critical need to understand and harden its environment for defensive analysis.

2.4 The Adversarial Challenge: Malware Evasion Techniques

The effectiveness of any sandbox is constantly being challenged by malware authors who develop techniques to evade analysis. This "cat-and-mouse" game is a central theme in malware research. Evasion techniques are designed to answer one question from the malware's perspective: "Am I being watched?" If the answer is yes, the malware changes its behavior (Afianian et al., 2019). These techniques are incredibly diverse and target nearly every aspect of the sandbox environment.

The most common class of evasion techniques involves **fingerprinting** the environment. The malware actively searches for artifacts—tell-tale signs—that reveal it is running inside a sandbox or virtual machine.

- **Virtual Hardware Artifacts:** Malware often queries for hardware identifiers that are common in virtual environments. This includes checking for MAC addresses with prefixes belonging to virtualization vendors (e.g., 00:0C:29 for VMware), looking for virtual disk names like "VBOX HARDDISK" or "VMware Virtual IDE," or reading BIOS information that contains strings like "VirtualBox" or "Hyper-V" (Yokoyama et al., 2016).
- **Registry and File System Artifacts:** Virtualization software and analysis tools often leave traces in the file system and registry. Malware will check for the existence of specific files, such as guest additions drivers (e.g., VBoxGuestAdditions.exe), or look for registry keys known to be created by VMware or VirtualBox. The presence of these artifacts is a clear giveaway that the environment is not a standard physical machine (Apriorit, 2023).
- **Process and Service Artifacts:** Malware can enumerate the running processes and services on the system. If it finds processes associated with common analysis tools (e.g., wireshark.exe, procmon.exe) or virtualization services (vmtoolsd.exe), it will assume it is being analyzed.

- **Detecting a "Too Perfect" Environment:** Paradoxically, a sandbox can also be detected if it is *too clean*. Real user machines have a history of activity. Malware might check for things like the number of recently opened documents, the browser history, or the system uptime. An environment with no recent files, an empty browser cache, and an uptime of only a few minutes is highly suspicious. This has led to research on creating more "lived-in" sandboxes that emulate user behavior to appear more realistic (Liu et al., 2022).

  The unique architecture of Windows Sandbox, with its reliance on the wcifs.sys driver and vSMB shares, presents a new and specific set of artifacts that advanced malware could target (Ilgayev, 2021).

Another class of evasion techniques exploits temporal differences between a real system and a sandboxed one.

- **Stalling and Sleeping:** This is one of the simplest yet most effective techniques. The malware will simply wait or "sleep" for a long period could be seconds or minutes before executing its malicious payload. Many automated sandboxes have a fixed analysis timeout (e.g., 10 minutes) to maintain high throughput. If the analysis finishes before the malware "wakes up," its malicious behavior will be completely missed. While sandboxes can try to patch the Sleep() API call to fast-forward time, malware can use more subtle stalling loops that are harder to detect (Roccia, 2019).
- **Detecting Virtualization Overhead:** The execution of certain CPU instructions can take a measurably different amount of time on a virtual CPU compared to a physical one. Malware can use high-precision timers (like the RDTSC instruction) to measure the time it takes to perform certain operations. If the time is outside the expected range for native hardware, it can infer the presence of a hypervisor. This is a very subtle technique that is difficult to defend against (Genç et al., 2019).

Since sandboxes and analysts often use debuggers to step through code and instrumentation frameworks to monitor API calls, malware is frequently equipped with anti-debugging techniques.

- **Checking for Debugger Presence:** Windows provides several ways for a program to check if it is being debugged. The simplest is the IsDebuggerPresent() API call. Malware can also directly check a flag in the Process Environment Block (PEB). If a debugger is detected, the malware will terminate or change its behavior.
- **Detecting API Hooks:** Many sandboxes work by "hooking" API calls. This involves modifying the first few bytes of a system function in memory to redirect execution to the sandbox's logging code. Malware can detect these hooks by checking the integrity of system DLLs in its own memory space. If it finds that functions like CreateFile or InternetOpen have been modified, it knows it is being monitored.
- **Exploiting Exceptions:** Malware can intentionally trigger software exceptions (like a breakpoint instruction) and see how they are handled. If an external debugger catches the exception instead of the program's own exception handler, the malware knows it is being debugged (Afianian et al., 2019).

Some malware will only execute its malicious payload when very specific conditions, or "triggers," are met. These triggers are designed to be common on a real victim's machine but rare in a sandbox.

- **User Interaction Triggers:** The malware might wait for the user to perform a specific action, such as moving the mouse in a non-random pattern, clicking a specific button in a decoy user interface, or opening a particular type of document. Since automated sandboxes typically lack human interaction, the trigger is never activated.

- **Environment Triggers:** The malware may check for specific system configurations, such as a particular system language, a certain number of files on the desktop, or whether the machine is joined to a corporate domain. This is especially common in targeted attacks like Advanced Persistent Threats (APTs). The research by Mills and Legg on the MORRIGU framework demonstrates how sandbox environments can be systematically reconfigured to try to find and satisfy these triggers automatically (Mills & Legg, 2020).

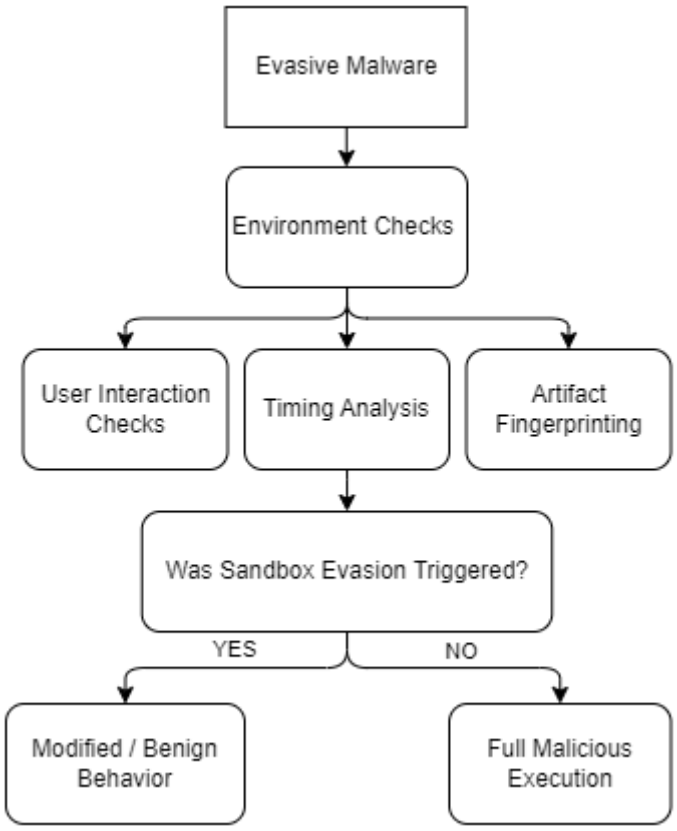

Figure 2.4. Sandbox Evasion Decision Flow.

Malware performs a series of environment checks—including user interaction monitoring, timing analysis, and artifact fingerprinting—to determine whether it is running inside a sandbox. If a sandbox is detected, it alters its behavior to appear benign; otherwise, it

proceeds with full malicious execution, this decision logic captures the fundamental challenge posed by sandbox-evasive malware. The checks are often lightweight, rapid, and highly effective at suppressing payloads during automated analysis. Overcoming this requires both reengineering the sandbox's visibility and implementing retrospective forensic workflows—approaches that are central to the Sandbox Enhancer system presented in this thesis.

A growing trend in modern malware is the use of "fileless" techniques. This type of malware avoids writing a traditional executable file to the disk, which is what most antivirus software scans for. Instead, it operates entirely in memory (Olaimat et al., 2021).

Fileless malware often works by "living off the land," meaning it abuses legitimate tools that are already built into the operating system. For example, it might use **PowerShell**, a powerful scripting tool in Windows, to download and execute malicious scripts directly in memory. Or it might use **Windows Management Instrumentation (WMI)** to create persistence mechanisms that are stored in the WMI database, not in the file system or registry. Banking trojans like **Emotet** and **TrickBot** are notorious for their heavy use of fileless techniques to evade detection (Kuraku & Kalla, 2020). Analyzing this type of malware is a major challenge for sandboxes that primarily focus on file-based threats and requires strong memory analysis capabilities.

2.5 Countermeasures: Enhancing Sandbox Stealth and Resilience

In response to the vast array of evasion techniques, researchers and security vendors have developed countermeasures to make sandboxes "stealthier" and more resilient. The goal is to make the analysis environment as indistinguishable from a real physical machine as possible.

The most direct countermeasure is to identify the artifacts that malware looks for and then modify or remove them. This is often referred to as "hardening" the sandbox. This can involve:

- **Spoofing Hardware IDs:** Changing the MAC address, disk serial numbers, and BIOS strings to match those of a real hardware vendor like Dell or HP.
- **Cleaning the Registry and File System:** Removing or renaming files and registry keys associated with the virtualization platform.
- **API Hooking for Deception:** Intercepting API calls that malware uses to query the environment and returning fake data. For example, when the malware asks for BIOS information, the sandbox can intercept the request and return a string that looks like it came from a real machine instead of "Hyper-V."

However, this process is challenging. It is difficult to identify and modify *all* possible artifacts, and the process of modification can sometimes introduce new, subtle artifacts that can themselves be detected (Veerappan et al., 2018). This is a core focus of the research presented in this thesis.

To counter malware that checks for a "too perfect" environment, research has focused on emulating realistic user behavior. Systems like **UBER** are designed to automatically perform actions within the sandbox to create a history of use. This includes browsing websites, creating and saving documents, and running common applications. By generating this "wear and tear," the sandbox appears to be a normal, actively used machine, which can defeat malware checks for user activity (Feng et al., 2020).

2.6 The Role of Machine Learning in Malware Analysis

In recent years, machine learning (ML) has become an increasingly important tool in malware detection and analysis. Instead of relying on hand-crafted signatures, ML models

can be trained to automatically identify the characteristics of malicious files (Gorment et al., 2023).

ML can be applied to both static and dynamic analysis. In static analysis, models can be trained on features extracted from the PE header, the byte content of the file, or the strings within the binary (Saxe & Berlin, 2015). The **EMBER** dataset is a large, public dataset that has been widely used for training static analysis models (Anderson & Roth, 2018). In dynamic analysis, ML models can be trained on the behavioral reports generated by a sandbox. Features might include the sequence of API calls, the number of files created, or the domains contacted (Dhammi & Singh, 2015).

However, ML is not a silver bullet. A groundbreaking study by Kaya et al. (2025) highlighted a critical problem known as the **"sandbox-to-endpoint chasm."** They found that ML models trained on behavioral logs from sandboxes often perform very poorly when deployed on real endpoint machines. The reason is that the data distribution is different. Sandbox logs are often from short, clean executions of malware that may be actively evading the environment. Endpoint logs are much "noisier" and reflect the malware's full, uninhibited behavior in a complex environment. The models trained on sandbox data often learn "spurious correlations"—features that are predictive only in the artificial sandbox environment but do not generalize to the real world (Kaya et al., 2025). This finding emphasizes that the quality and realism of the sandbox environment are absolutely critical, not just for manual analysis, but for training the next generation of automated detection systems.

2.7 Gaps in Existing Research

This comprehensive review of the literature reveals several important gaps that this thesis aims to address:

1. **Lack of Focus on Windows Sandbox:** While there is extensive research on general-purpose sandboxes like Cuckoo and on broad evasion techniques, there is a significant lack of research focused specifically on the security and enhancement of Microsoft Windows Sandbox as a malware analysis platform. Its unique architecture presents both opportunities and challenges that are not well understood in the academic community.
2. **Need for Practical and Automated Artifact Modification:** Much of the research on sandbox hardening is theoretical or focused on specific, isolated techniques. There is a need for a practical, integrated system that automates the process of artifact modification within a readily available platform like Windows Sandbox.
3. **Integrating Stealth with Automation and Logging:** Existing research often treats sandbox stealth, automation, and logging as separate problems. There is a gap in developing a holistic framework that combines all three aspects: a stealthy environment to elicit true behavior, an automated workflow for efficiency, and a comprehensive logging system for deep visibility.
4. **Bridging the Sandbox-to-Endpoint Chasm:** As highlighted by Kaya et al., there is a critical need to improve the realism of sandbox environments to generate more representative behavioral data. While this thesis does not directly solve the ML training problem, by creating a stealthier and more realistic sandbox, it contributes to generating the higher-quality data that is needed to address this gap.

To the best of the author's knowledge, no prior work has implemented a structured framework to transform Windows Sandbox into a stealthy, automated malware analysis platform. This research introduces a novel approach by combining artifact obfuscation, automation, and centralized logging to trigger behavior in evasive malware samples

2.8 Conclusion

The literature on malware analysis paints a clear picture of a dynamic and adversarial field. The evolution from static to dynamic and hybrid analysis methods shows a constant push for deeper visibility into malware behavior. Sandboxing has become the cornerstone of this effort, but its effectiveness is perpetually challenged by the creative and persistent evasion techniques developed by malware authors. They fingerprint sandbox environments, exploit timing differences, and wait for specific triggers to avoid analysis.

In response, the research community has focused on making sandboxes stealthier by modifying artifacts and emulating user behavior. At the same time, the rise of machine learning has introduced powerful new capabilities for automated detection, but it has also highlighted the critical importance of the quality and realism of the data generated by sandboxes.

This review reveals that while much progress has been made, there are clear gaps, particularly in the context of new, lightweight platforms like Windows Sandbox. There is a need for a holistic approach that integrates stealth, automation, and deep logging into a single, practical framework. By addressing these gaps, this thesis seeks to make a significant contribution, transforming Windows Sandbox from a simple utility into a powerful and reliable tool for the cybersecurity community in its ongoing fight against malicious software. The next chapter will detail the methodology designed to achieve this transformation.

Chapter Three: Methodology and System Design

3.1 Introduction to the Methodological Framework

This chapter provides a comprehensive and technically detailed exposition of the methodology that underpins this research. It serves as the core of the thesis, delineating the systematic, multi-phase process undertaken to design, implement, and prepare for the validation of the "Sandbox Enhancer" system. As established in the preceding chapters, the practical utility of standard analysis environments, particularly Microsoft Windows Sandbox, is severely constrained by its inherent susceptibility to detection and evasion by modern malware. The research presented here directly confronts this critical challenge by proposing and developing an engineered solution designed to systematically enhance the sandbox's stealth, fully automate its operational workflow, and dramatically deepen its analytical visibility.

The methodology described herein is empirical, structured, and iterative. It reflects a pragmatic engineering philosophy aimed at translating a well-understood cybersecurity problem into a tangible and effective solution. The overarching objective is to transform Windows Sandbox from a general-purpose, disposable utility into a specialized, research-grade platform capable of dissecting evasive threats. This chapter is meticulously structured to provide a transparent and replicable account of this process, ensuring that another researcher could, in principle, follow these steps to recreate the system, understand the design choices, and validate the findings presented in this thesis.

The chapter begins by outlining the high-level, multi-phase structure that guided the research from conception to validation. It then delves into a granular exploration of each phase. The initial phases focus on **Foundational Analysis and Strategic Design**, detailing the baseline characterization of the default Windows Sandbox, the systematic identification

and prioritization of its detectable artifacts, and the subsequent development of targeted modification strategies to enhance its stealth.

The chapter then transitions from analysis and design to concrete implementation. The sections on the **Automated Analysis Workflow** and the **Comprehensive Log Management Framework** describe the architectural blueprint of the system and its data collection pipeline. This is followed by the **Full System Implementation** section, which provides an exhaustive description of how each software component—the host-side "Sandbox Manager" GUI, the in-sandbox automation scripts, and the integrated logging tools—were developed and integrated into a single, cohesive system.

Finally, the chapter concludes with the **Experimental Design for Validation** section. This section lays out the rigorous scientific plan for testing the final system, defining the carefully curated malware dataset, the experimental procedures, and the specific, measurable metrics that will be used to evaluate the system's performance in terms of stealth, efficiency, and analytical depth. By providing this level of exhaustive detail, this chapter not only documents the original work performed but also serves as a comprehensive guide for future research and development in the critical and ever-evolving field of sandbox security.

3.2 Methodological Framework Overview

The research was executed following a structured, five-phase methodology, ensuring a logical progression from foundational analysis to system implementation and empirical validation. Each phase builds upon the outputs of the preceding one.

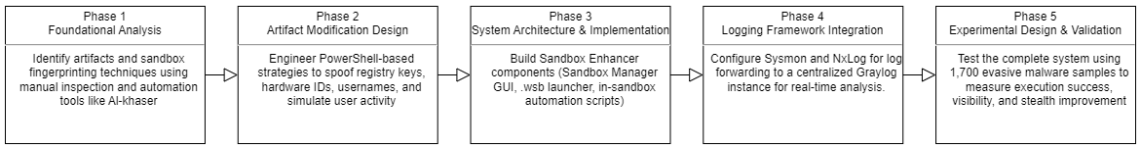

Figure 3.2: High-Level Methodological Flowchart.

The diagram shows a linear progression: Phase 1 (Foundational Analysis) feeds into Phase 2 (Artifact Modification Design), which informs Phase 3 (System Architecture & Implementation). Phase 3 leads to Phase 4 (Logging Framework Integration), and the complete system is then subjected to Phase 5 (Experimental Design & Validation).

- **Phase 1: Foundational Analysis and Artifact Identification.** This initial phase involved a deep architectural analysis of the default Windows Sandbox to identify its operational characteristics and systematically discover its environmental artifacts. This process established a comprehensive baseline of the platform's vulnerabilities to malware evasion techniques.
- **Phase 2: Design of Artifact Modification Strategies.** Building on the findings from Phase 1, this phase focused on designing and developing a suite of script-based countermeasures. The goal was to create effective and stable techniques for modifying or concealing the highest-priority artifacts to enhance the sandbox's stealth.
- **Phase 3: System Architecture and Implementation.** This phase translated the conceptual designs into a concrete, multi-component software architecture. It involved the detailed design and implementation of the host-side "Sandbox Manager" and the in-sandbox automation scripts that form the core of the "Sandbox Enhancer" system.
- **Phase 4: Integration of the Comprehensive Logging and Analysis Framework.** This phase was dedicated to building the system's observability pipeline. It involved the selection, configuration, and integration of specialized tools—Sysmon, NxLog, and Graylog—to enable the capture, aggregation, and centralized analysis of fine-grained behavioral data.
- **Phase 5: Experimental Design for Empirical Validation.** The final phase outlines the rigorous scientific plan for testing the fully implemented system. It specifies the dataset curation strategy, the experimental procedures, and the quantitative and

qualitative metrics defined to measure the system's success against its stated objectives.

The following sections will now provide an exhaustive, technically detailed exploration of each of these five phases.

3.3 Phase 1: Foundational Analysis and Artifact Identification

Objective: To conduct a deep-dive analysis of the default Microsoft Windows Sandbox to create a comprehensive and categorized inventory of all detectable artifacts that could be used by malware for evasion, and to prioritize these artifacts for mitigation.

This foundational stage is arguably the most critical, as the quality of the "Sandbox Enhancer" system is directly dependent on the accuracy and completeness of the initial vulnerability assessment. A solution cannot be engineered without first possessing an exhaustive understanding of the problem. Therefore, before any enhancements were designed, a meticulous process was undertaken to map the "attack surface" of the default sandbox environment from an adversary's perspective. This adversarial mindset guided the entire discovery process.

The analysis began with a review of Microsoft's technical documentation on Windows Sandbox and Hyper-V, as well as detailed third-party research, notably the work by Ilgayev (2021) (Ilgayev, 2021). This review established that Windows Sandbox's unique architecture—a blend of Hyper-V virtualization and Windows Container technologies—would likely produce a unique fingerprint, different from both traditional VMs (like VMware) and native systems. This led to the initial hypothesis that evasion techniques would target artifacts related to:

- The Hyper-V hypervisor itself.
- The virtualized hardware presented to the guest OS.

- The unique file system structure created by the Windows Container Isolation (WCI) driver.
- The generic and pristine state of the default user environment.

A multi-pronged discovery process was used to ensure maximum coverage, capturing both well-known virtualization artifacts and those potentially unique to the Windows Sandbox architecture.

Automated Scanning with Al-khaser:

The open-source tool **Al-khaser** was deployed and executed inside the sandbox. Al-khaser is essentially a compendium of dozens of known sandbox and VM detection techniques, compiled into a single executable. Running it provided a rapid, automated assessment of the most common vulnerabilities. Its output served as an initial, high-level report, confirming that the default sandbox was vulnerable to many classic evasion checks.

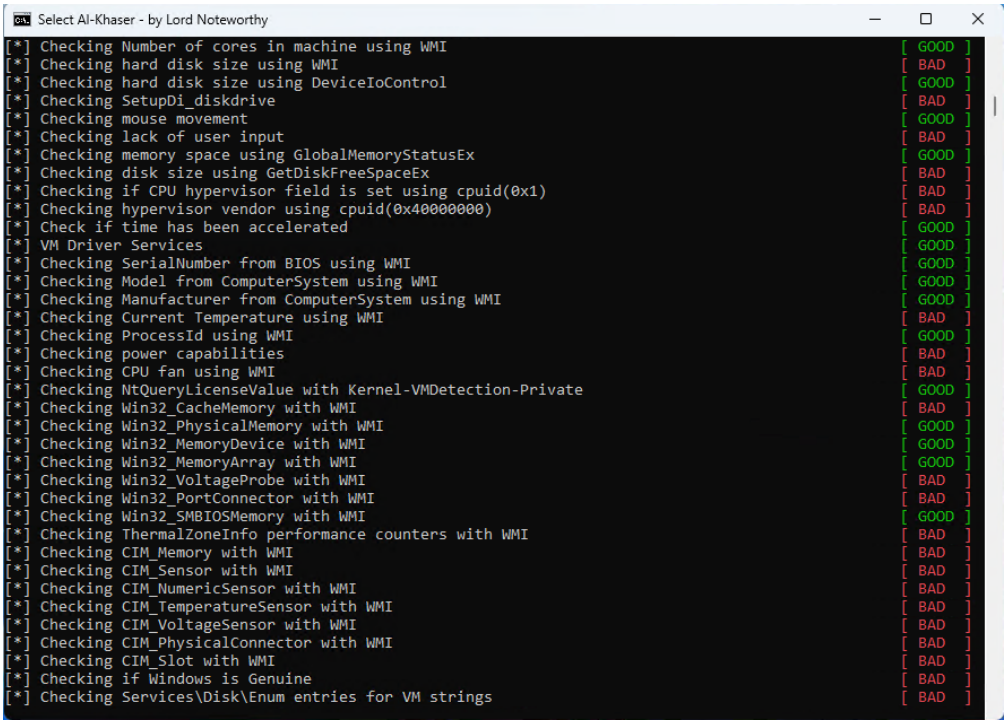

Figure 3.3**:** Detailed Al-Khaser Output from Windows Sandbox Environment.

The figure displays output from Al-Khaser, highlighting which system-level checks passed ("GOOD") or failed ("BAD") inside Windows Sandbox. Failures indicate exposed artifacts such as hypervisor vendor strings, WMI inconsistencies, and missing sensor data. These weaknesses confirm that Windows Sandbox presents multiple detectable traits commonly exploited by evasive malware.

Manual Inspection and Enumeration:

To find more subtle artifacts and to verify Al-khaser's findings, extensive manual inspection was performed using built-in Windows tools and PowerShell scripts inside the sandbox. This included:

- **Registry Analysis:** Using regedit.exe, the registry was manually traversed, searching for keywords like Virtual, VM, Hyper-V, MSFT, and VMBus. Special attention was paid to the HKLM\SYSTEM\CurrentControlSet\Services and HKLM\HARDWARE\DEVICEMAP keys.
- **WMI Queries:** Windows Management Instrumentation (WMI) is a common interface used by malware to query system information. A series of PowerShell scripts were executed to query key WMI classes:
    - Get-WmiObject -Class Win32_ComputerSystem: To check the Model and Manufacturer.
    - Get-WmiObject -Class Win32_BIOS: To check the SerialNumber and SMBIOSBIOSVersion.
    - Get-WmiObject -Class Win32_DiskDrive: To inspect the Model and PNPDeviceID.
    - Get-WmiObject -Class Win32_NetworkAdapterConfiguration: To check the MACAddress.
- **File System and Driver Enumeration:** The C:\Windows\System32 and C:\Windows\System32\drivers dire

ctories were manually inspected for files known to be associated with Hyper-V, such as vmbus.sys, storvsp.sys, and vmswitch.sys.

- **User Environment Checks:** Standard commands like whoami and hostname were used to confirm the default username and computer name. The state of the user's profile directory (C:\Users\WDAGUtilityAccount) was inspected for its lack of typical user files.

The combination of automated and manual discovery yielded a comprehensive catalog of artifacts. These were then prioritized for modification based on (a) the likelihood of being checked by malware and (b) the feasibility of modification via scripting. Table (3.1) presents a partial but representative list of the high-priority artifacts identified.

Table 3.1: Prioritized List of Detectable Artifacts in Windows Sandbox

Category	Artifact & Specific Example	Discovery Method	Impact	Feasibility	Priority
1. Hardware & BIOS	BIOS Manufacturer: " MICROSOFT "	WMI Query	High	High	Critical
	Disk Drive Model: "MICROS OFT"	WMI Query	High	High	Critical
	Disk Capacity: Reported as < 50 GB	Get-Volume	High	High	High
	Network MAC Prefix: 00:15:5D	ipconfig /all	Medium	Low	High
	CPU Name: Contains	WMI Query	Medium	High	High

	"Virtual" or is a generic string.				
2. Registry	Hyper-V Service Keys: HKLM:\...\Services\vmbus	Registry	High	High	Critical
	Hardware Enum Strings: Enum\SCSI\...\DeviceDesc = "Microsoft Virtual Disk"	Registry	High	High	Critical
	Video Driver Info: HKLM\...\Control\Video\...\Video.DeviceDescription = "Microsoft Hyper-V Video"	Registry	Medium Impact: Another clear hardware string indicating virtualization.	High	High
3. User Environment	Default Username: WDAGUtilityAccount	whoami	Critical (the easiest way to detect windows sandbox)	High	Critical
	Pristine File System: No files	dir	High	High	High
	Low Process Count: Fewer than 30-40 processes on startup.	tasklist	Medium	Low	Medium
	Short System Uptime: Always starts at < 1 minute.	systeminfo	Medium	High	Medium

This detailed and prioritized inventory formed the strategic foundation for the entire enhancement process. It provided a clear "hit list" of vulnerabilities to be neutralized by the artifact modification module of the "Sandbox Enhancer" system.

3.4 Phase 2: Design of Artifact Modification Strategies

Objective: To design, implement, and test a suite of practical, script-based countermeasures to modify or conceal the high-priority artifacts identified in the previous phase, with the overarching goals of maximizing sandbox stealth while ensuring operational stability.

This phase translated the theoretical "hit list" of artifacts into a practical arsenal of countermeasures. For each prioritized artifact, a specific technical solution was engineered. The guiding philosophy was to be as effective as possible while being as non-invasive as possible to maintain system stability.

The development of modification techniques was guided by two core principles:

1. **Effectiveness over Completeness:** It is practically impossible to eliminate every single artifact of virtualization. The goal was not to create a "perfect" undetectable environment, but to eliminate the *most common and easily checked artifacts* that the majority of evasive malware relies upon. This pragmatic approach focuses on raising the bar for detection high enough to defeat a large percentage of threats.
2. **Stability is Paramount:** An enhancement is useless if it makes the analysis environment unstable. Every modification technique was designed to be as non-invasive as possible. Changes that risked crashing the operating system or interfering with the legitimate execution of programs were rejected in favor of safer, more reliable methods.

A modular set of PowerShell scripts was developed, with each script targeting a specific category of artifacts.

1. Registry Obfuscation Module:

A single, self-contained PowerShell script walks through a predefined list of high-signal keys—chiefly HARDWARE\DESCRIPTION\System (system and BIOS fields) and SYSTEM\CurrentControlSet\Enum (disk and adapter identities). For each entry it first reads the current value; if that value matches a known virtual-machine marker such as "Microsoft Corporation" or "Msft Virtual Disk," the script quietly overwrites it with a realistic string taken from a retail hardware catalogue—for example, "Dell Inc." / "OptiPlex 7010" for system data and "WDC WD10EZEX-00BN5A0" for the primary disk. Expressed in pseudocode, the routine is simply: for every target key → if the key exists and looks virtual → replace it with its spoof value → otherwise leave it untouched. All operations rely solely on native PowerShell cmdlets, run once at sandbox start-up, and disappear when the sandbox is closed, giving each session a believable, clean-slate hardware profile without persisting any changes on the host

2. User Environment Camouflage Module:

This module focused on making the sandbox environment look less like a sterile, freshly-installed machine.

- **Technique:** A combination of command-line tools and file system operations, all wrapped in a PowerShell script.
- **Implementation:**
    - **Username Spoofing:** Contrary to initial approaches that attempt to rename the default WDAGUtilityAccount—a method often restricted due to system protections—the implemented script instead creates a new local user with a realistic, non-suspicious name using the "net user" command. This newly added account serves as the default login environment, thereby bypassing the limitations of modifying system accounts while reducing the likelihood of detection through username heuristics.

- **Decoy File adding:** The script copies realistic user files from a pre-prepared mapped host folder into the Desktop and Documents directories of the sandbox environment. These files include commonly expected names such as Budget.xlsx, Family Photo.jpg, Project Proposal.docx, and resume.pdf. In addition, it creates a few user-like directories (e.g., Vacation Pics, Work Files) to further simulate real user activity. This approach directly counters malware that performs environmental checks by scanning the user profile for signs of authenticity.
- **Background Activity Simulation:** A separate, lightweight PowerShell script was developed to be launched as a background job. Its sole purpose was to move the mouse cursor to a random screen location every 30 seconds. This simple script effectively defeats basic inactivity checks without consuming significant system resources.
- **MAC Address Spoofing via PowerShell:** A PowerShell-based method was implemented to spoof the MAC address of the virtual network adapter, which is often inspected by malware to detect virtual environments. The script programmatically located the relevant network adapter using WMI queries, accessed its registry path under HKLM\SYSTEM\ControlSet001\Control\Class\{4D36E972-E325-11CE-BFC1-08002BE10318}, and updated the NetworkAddress value with a randomized, non-virtual prefix. To apply the change in real-time, the adapter was temporarily disabled and re-enabled using WMI. This effectively conceals the default Hyper-V MAC prefix (00:15:5D) used by Windows Sandbox and helps evade basic virtual machine fingerprinting.

3. **Windows Management Instrumentation spoofing:**

Windows Management Instrumentation (WMI) constitutes the authoritative telemetry backbone that malware consults to verify physical authenticity. To prevent evasive samples—specifically Al-Khaser's twenty-one WMI probes—from

detecting the Windows Sandbox, we authored twelve single-row Managed Object Format (MOF) stubs that are compiled at session start-up. Each stub is injected into the same namespace as its genuine provider (e.g., \\root\\cimv2, \\root\\wmi), preserves the class's native key properties, and presents plausibly sized field values such as a 1 TB SSD, 16 MB L3 cache, and a CPU temperature of 27 °C. Because the CIM repository resides in the sandbox's differencing disk, the spoofed instances persist for the entire analysis session yet disappear automatically when the sandbox is destroyed, ensuring a pristine environment for subsequent runs. Immediately after compilation, every WMI query that Al-Khaser issues returns "good" data, flipping all checks from BAD to GOOD without touching kernel drivers, registry keys, or any artefact on the host—delivering session-persistent deception that is high-impact.

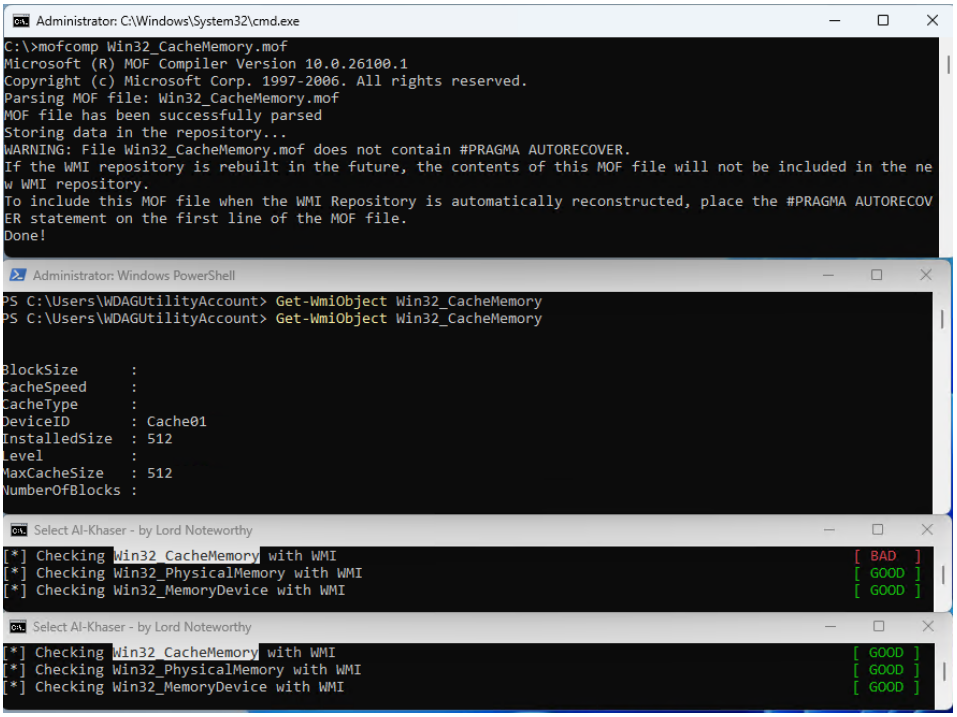

Figure 3-7. MOF-based spoofing of the Win32_CacheMemory class.

As illustrated in Figure 3-7, compiling a single-row MOF stub for Win32_CacheMemory instantaneously populates the class and neutralises Al-Khaser's cache-memory probe.

During development, it became clear that several high-priority artifacts could not be effectively mitigated with this script-based, user-mode approach. It was a critical methodological step to acknowledge these limitations rather than ignore them.

4. Per-Process DLL CPUID-Spoof Module:

I developed a tiny loader that, injects a custom DLL into the malware process. On load, the DLL scans ntdll!RtlUserThreadStart until it finds the first CPUID opcode (0F A2). That byte is replaced with INT 3 (0xCC). When any thread reaches the patched instruction Windows raises an EXCEPTION_BREAKPOINT; a vectored-exception handler (VEH) restores the original opcode, single-steps over the real CPUID, then rewrites the returned registers in the trap frame

(i) it clears ECX bit 31 when the leaf is EAX = 1, removing the hypervisor-present flag, and

(ii) when the leaf is EAX = 0x40000000 it zeroes EAX and overwrites EBX/ECX/EDX with the ASCII values for "GenuineIntel"

(iii) Finally the DLL re-arms the breakpoint so the next thread is treated the same way. All work is done in user mode, so no Windows-Sandbox flags or test-signing are required; the patch executes in < 1 ms and adds zero boot-time overhead while neutralising every in-process CPUID test used by Al-Khaser.

5. Disk Size Artifact Modification via VHDX Resizing:

A practical and previously unaddressed artifact — the default virtual disk size of the Windows Sandbox — was neutralized by dynamically resizing the underlying .vhdx file used by the sandbox. This was achieved using the PowerShell command Resize-VHD, targeting the correct path inside C:\ProgramData\Microsoft\Windows\Containers\ContainerStorages\ The virtual disk was resized to 100 GB, effectively defeating malware that relies on detecting small disk sizes (<50 GB) as a sandbox indicator. Unlike kernel-level spoofing, this method introduces no instability and is compatible with the standard Windows Sandbox architecture.

6. Exploratory Kernel-Driver Loader (optional):

Turning on `CmDiag DevelopmentMode` and `CmDiag testsign` allowed an unsigned driver to hide low-level artefacts such as SMBIOS strings and IDT addresses.However, the switch slowed Sandbox start-up from ≈ 9 s to ≈ 31 s and caused FAILURE most of the time. A public “SandboxBootkit” (Thesecretclub (n.d.)) patch for `bootmgfw.efi` was also tested on build 26100 but failed with the same guard violation. Because of the delay and instability, the production pipeline relies on the fast DLL method; kernel drivers are earmarked for future research rather than routine use.

Every script in the modification module was subjected to a rigorous stability test. A clean sandbox was launched, the script was run, and the system was monitored for at least 15 minutes for any signs of instability, such as process crashes, blue screen errors, or application failures. The sandbox was also restarted to ensure the changes persisted correctly and did not interfere with the boot process. Only scripts that were proven to be 100% stable were included in the final "Sandbox Setup" automation module. This disciplined testing regimen was essential to ensure the final enhanced environment was not only stealthy but also robust and reliable.

3.5 Phase 3: System Architecture and Implementation

Objective: To design and construct the cohesive software architecture that integrates the artifact modifications, automates the end-to-end analysis workflow, and establishes the comprehensive logging pipeline, culminating in the final "Sandbox Enhancer" system.

This extensive phase represents the primary engineering effort of the thesis, where all the preceding analysis and design work was translated into functional, integrated code. The system was architected with modularity, automation, and user-friendliness as its guiding principles.

The "Sandbox Enhancer" system is architecturally divided into three logical domains: the Host Environment, the Sandbox Environment, and the persistent Central Analysis Server. These domains communicate to create a seamless, automated analysis pipeline.

The .wsb file is the linchpin that connects the host to the guest environment and initiates the entire automated workflow. It is an XML-based configuration file that allows for the customization of a Windows Sandbox session. A new .wsb file was dynamically generated by the "Sandbox Manager" for each analysis run.

Example .wsb Configuration Used in This Research:

```
<Configuration>
  <!-- Disable GPU sharing to reduce potential driver-related artifacts and instability -->
  <VGpu>Disable</VGpu>
  <!-- Control network access. For initial local analysis, this is set to Disable. -->
  <Networking>Enable</Networking>
  <!-- Map host folders into the sandbox. ReadOnly is crucial for security. -->
  <MappedFolders>
    <MappedFolder>
      <HostFolder>C:\SandboxEnhancer\Tools</HostFolder>
```

```
    <ReadOnly>true</ReadOnly>
   </MappedFolder>
   <MappedFolder>
    <HostFolder>C:\SandboxEnhancer\Target</HostFolder>
    <ReadOnly>true</ReadOnly>
   </MappedFolder>
  </MappedFolders>
  <!-- The command to execute automatically after the user logs in to the sandbox. -->
  <!-- This is the primary trigger for the entire automation pipeline. -->
  <LogonCommand>
<Command>C:\Users\WDAGUtilityAccount\Desktop\Tools\SandboxSetup.bat</Comman
d>
  </LogonCommand>
  <!-- Optional security enhancement using AppContainer isolation -->
  <ProtectedClient>true</ProtectedClient>
</Configuration>
```

The Windows Sandbox configuration is defined using an XML schema under the <Configuration> root element. The following are primary parameters and their default behaviors, followed by their configurable functions:

- **vGPU (virtualized GPU)**: *Enabled* on non-ARM64 devices. This enables GPU acceleration inside the sandbox. If disabled, it defaults to using Windows Advanced Rasterization Platform (WARP).
- **Networking**: *Enabled* by default using the Hyper-V default switch. This setting determines whether the sandbox has internet or local network access.
- **Audio Input**: *Enabled*. Allows the sandbox to access the host's microphone input.

- **Video Input**: *Disabled*. Prevents access to the host's webcam by default.
- **Protected Client**: *Disabled*. This setting, if enabled, adds enhanced security to the RDP session connecting to the sandbox instance.
- **Printer Redirection**: *Disabled*. Restricts access to the host's printer devices.
- **Clipboard Redirection**: *Enabled*. Allows copy-paste operations between the host and sandbox for text and files.
- **Mapped Folders**: Not enabled by default. This option allows the user to share one or more host directories into the sandbox. Each mapped folder entry includes:
    - <HostFolder>: The path on the host to share.
    - <SandboxFolder> (optional): Target path inside the sandbox.
    - <ReadOnly> (optional): Whether the folder is read-only.
      Sharing folders introduces risk, as malware may access or exfiltrate host data.
- **Logon Command**: Executes one or more commands when the sandbox session begins. Each <Command> element can contain a startup instruction such as launching a script or binary.
- **Memory in MB**: Specifies the RAM allocated to the sandbox environment.

These configuration parameters offer granular control over the analysis environment and can be adjusted to simulate varying levels of system exposure. This fine-tuning is essential for observing sandbox-evasion techniques and ensuring safe, realistic malware execution.

The automation inside the sandbox was designed as a two-stage process, orchestrated by the restart action. This design ensures that all environmental hardening is complete *before* the malware is ever executed.

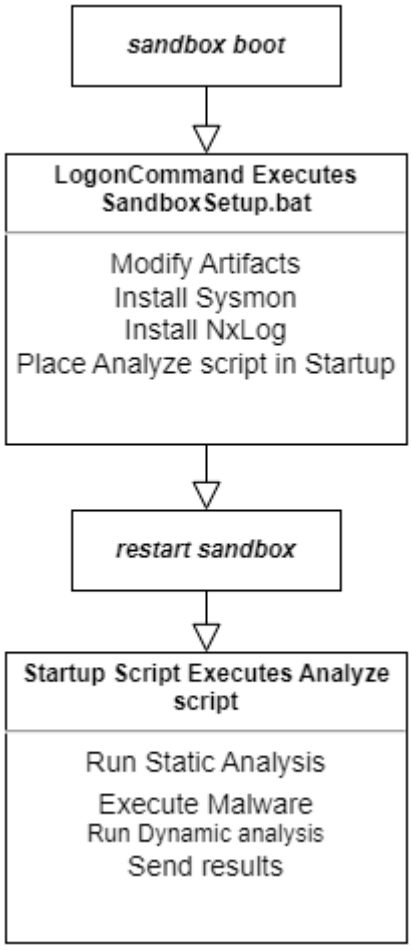

Figure 3.5.3: In-Sandbox Automation Workflow.

A sequential flowchart illustrating the automated actions performed inside the Windows Sandbox. The process begins with the sandbox booting and executing SandboxSetup.bat via the LogonCommand directive. This script modifies sandbox artifacts, installs Sysmon and NxLog, and schedules the analysis script. A forced restart follows, after which Analyze

script is triggered at startup to conduct static analysis, execute the malware sample, pause for behavioral observation, and extract memory-resident strings.

Component 1: The "Sandbox Manager" (Host-Side Orchestrator)

- **Purpose:** This component acts as the "mission control" for the entire system. It is the sole point of interaction for the analyst, designed to hide all underlying complexity and provide a simple, intuitive workflow.
- **Implementation Details:**
    - The application was developed in **Python 3**.
    - The graphical user interface (GUI) was built using the built-in **Tkinter** library for its simplicity and cross-platform compatibility.
    - It uses Python's subprocess module to execute command-line tools and os and shutil modules for file system operations.
- **Step-by-Step Functional Breakdown:**
    - **First-Time Setup:** When run for the first time, a setup function is triggered. It programmatically checks for the presence of Docker Desktop. If not found, it opens a web browser to the Docker download page. It then uses subprocess to execute docker pull graylog/graylog:latest to download the required image. It also downloads the latest versions of all required third-party tools (Sysmon, NxLog, Pestudio, Die, Strings2) from their official sources into a local \Tools directory.
    - **User Interaction:** The main GUI window presents the analyst with a single prominent button: "Select Malware File to Analyze." This opens a standard file dialog.
    - **Workflow Initiation:** Once a malware .exe file is selected, the manager springs into action:
        - It copies the malware to a \Target directory.

- It dynamically writes a new .wsb (Windows Sandbox configuration) file. This is a critical step. The content of the .wsb file is generated programmatically, including the <MappedFolders> tags to share the \Tools and \Target directories as read-only, and the all-important <LogonCommand> tag, which is set to execute the SandboxSetup.bat script.
- Finally, it executes the newly created .wsb file, which causes Windows to launch the sandbox and begin the automated in-guest phase.

Component 2: The "Sandbox Setup" Script (The In-Sandbox Preparer)

- **Purpose:** To run immediately upon the sandbox's first boot, performing all necessary one-time environment preparation and hardening tasks.
- **Implementation Details:** This component is a master **PowerShell** script (Setup.ps1), which is called by a simple wrapper batch file (SandboxSetup.bat) specified in the .wsb logon command. PowerShell was chosen for its unparalleled ability to script Windows system administration tasks.
- **Step-by-Step Functional Breakdown:**
    1. **Execute Artifact Modifications:** The script's first action is to execute the suite of artifact modification scripts developed in Phase 3.
    2. **Install Sysmon:** It runs the Sysmon installer in silent mode (sysmon.exe -accepteula -i C:\Users\WDAGUtilityAccount\Desktop\Tools\config.xml), using a custom config.xml file that defines the logging rules.
    3. **Install and Configure NxLog:** It runs the NxLog MSI installer in silent mode and then copies the custom nxlog.conf file into place. This configuration tells NxLog what to log and where to send it (the host's Graylog server).

4. **Schedule the Analysis Stage:** It copies the third component, the "Sandbox Analysis" script, to the sandbox's system-wide Startup folder (C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp).
5. **Trigger Restart:** It issues a shutdown /r /t 0 command to force an immediate restart of the sandbox. This step is essential to finalize the installation of some services and to ensure the "Sandbox Analysis" script is executed on the next boot.

Component 3: The "Sandbox Analysis" Script (The In-Sandbox Analyst)

- **Purpose:** To run automatically after the sandbox restarts and to perform the actual, multi-stage analysis of the malware sample.
- **Implementation Details:** A **Python** script (Analyze.py) was chosen for this role due to its superior data handling and text processing capabilities, which are needed for parsing the output of the static analysis tools.
- **Step-by-Step Functional Breakdown:**
    1. **Static Analysis Stage:** The script first gathers static intelligence before executing the malware. It uses subprocess.run() with capture_output=True to execute Pestudio.exe, Die.exe, and Strings2.exe against the malware sample (located in the shared \Target folder). It captures their command-line output, parses it to extract key information (e.g., packer type, suspicious imports, API calls, embedded URLs), formats it into a structured JSON object, and writes it to a log file that NxLog is configured to watch and send to Graylog.
    2. **Execute the Malware:** Once static analysis is complete, the script executes the malware sample itself, launching it as a new process
    3. **Live Memory Forensics Stage:** The system was initially designed to include a memory analysis phase after dynamic execution. In this phase, the Strings2.exe -pid <PID> tool would be used to extract printable strings from the live memory space of the malware process, enabling the discovery of

volatile Indicators of Compromise (IOCs) such as decrypted command-and-control (C2) domains, embedded file paths, or runtime keys.

4. **Completion:** The script's work is done. The system is now idle, with all logs persistently stored in Graylog, waiting for the analyst to close the sandbox window.

The system's operation is orchestrated via a purpose-built graphical user interface (GUI) designed to perform pre-execution host validation and launch the analysis pipeline. Upon startup, the GUI conducts a series of automated readiness checks to verify the host environment: it ensures that Docker is installed, the Graylog container is active, and that all required directories—including those for malware samples, analysis tools, and operational scripts—exist and are accessible. Once validation passes, the user is presented with customizable sandbox settings. Upon confirmation, the GUI dynamically generates a .wsb (Windows Sandbox configuration) file which embeds a startup command to initiate analysis operations inside the sandbox instance.

Once the sandbox environment is initialized, a preparation script executes immediately. This script performs key artifact modifications to simulate a more realistic environment and bypass sandbox detection mechanisms. These modifications include MAC address spoofing via PowerShell, the copying of decoy user files (e.g., "Budget.xlsx", "Vacation.jpg"), and modification of selected system properties such as hardware profiles and system metadata. The script then configures a custom agent to launch at startup and triggers a system reboot to ensure all changes are committed.

After the reboot, the custom agent activates and proceeds with a staged analysis workflow. First, static analysis tools are executed against the target sample, and the resulting logs are saved to a mapped host directory. Once static inspection is complete, the same agent launches the dynamic execution of the sample. During runtime, NXLog monitors system events and transmits logs to the centralized Graylog container in real time. This dual-phase

analysis—static followed by behavioral monitoring—ensures a comprehensive assessment of each sample under realistic and artifact-aware conditions.

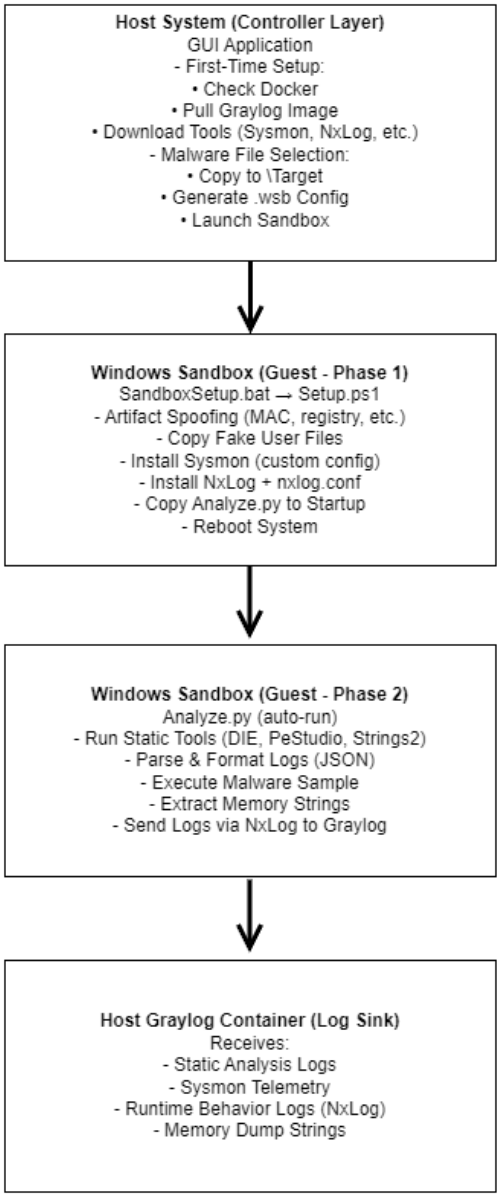

Figure 3.5.3 – Component-Level Execution Pipeline of the Windows Sandbox Enhancer.

The interaction between these three execution phases is illustrated in Figure (*3.5.3)*, which presents the full workflow from host-based orchestration to sandboxed analysis and final log delivery.

3.6 Phase 4: Implementation of the Logging and Analysis Framework

Objective: To implement and configure the multi-tool logging pipeline designed to provide deep, centralized, and persistent visibility into the malware's behavior inside the ephemeral sandbox.

In selecting Sysmon + NxLog + Graylog, we explicitly evaluated common alternatives against the study's constraints (Windows-centric, disposable sandboxes, reproducibility, low cost). Sysmon was preferred over relying solely on native Windows Event Logs because it exposes higher-fidelity process, registry, and network telemetry that materially improves malware behavior visibility. For collection, NxLog was chosen instead of Winlogbeat/Filebeat (which excel when coupled with ELK but push most transforms downstream), Fluent Bit (very lightweight yet less turnkey for rich Sysmon fielding on Windows), and Windows Event Forwarding (suited to domain-joined fleets but operationally awkward for ephemeral sandboxes); NxLog offers robust on-agent parsing, reliable disk buffering, and flexible outputs with minimal footprint. For the analysis layer, Graylog provided the required parsing, routing, alerting, and dashboards with fewer moving parts than a full ELK/OpenSearch stack and without the licensing/operational overhead of Splunk, while remaining transparent and self-hosted for exact methodological disclosure. Collectively, this stack best balanced visibility, control, and operability for controlled before/after experiments in the enhanced Windows Sandbox.

The logging framework is the sensory system of the "Sandbox Enhancer." Its implementation required careful selection and configuration of each tool to ensure a high-fidelity, low-noise data stream.

- **Sysmon (for Event Collection):** System Monitor from the Microsoft Sysinternals suite was chosen as the primary data source.
    - **Why Sysmon?** It provides kernel-level visibility into system activities that are far more granular than the default Windows Security Log. It generates richly detailed events for process creation (including parent process and command line), network connections (including source/destination IP and port), file creation, and registry modifications. This level of detail is essential for behavioral malware analysis (Rana et al., 2022).
- **NxLog (for Log Forwarding):** The NxLog Community Edition was chosen as the log shipper.
    - **Why NxLog?** It is lightweight, highly performant, and extremely flexible. Unlike other log forwarders, it has powerful built-in parsing and formatting capabilities. It was selected for its ability to read directly from the Windows Event Log, convert the events to the structured GELF format, and reliably forward them over TCP, ensuring no logs are lost.
- **Graylog (for Centralized Analysis):** Graylog was chosen as the central log management platform, or SIEM (Security Information and Event Management).
    - **Why Graylog?** As a powerful open-source platform, it provides the core functionalities needed for analysis without the high cost of commercial alternatives like Splunk. It excels at ingesting structured logs (like GELF from NxLog), provides a powerful search and query language, enables the creation of real-time dashboards for visualization, and has a robust alerting mechanism. Its deployment via Docker also simplified the setup process significantly.
- **Static Analysis Tools (Pestudio, Die, Strings2):** This suite of tools was chosen to provide initial static intelligence.
    - **Pestudio:** Excellent for analyzing PE headers and identifying suspicious characteristics, libraries, and imports.

- **Detect It Easy (Die):** A very effective tool for identifying the packer or compiler used to create the executable.
- **Strings2:** A powerful string extraction tool used both for initial static analysis and for the critical task of dumping strings from live process memory.

- **Sysmon config.xml:** This configuration file was the key to managing data volume. A highly targeted configuration was created based on industry best practices. It included specific rules to capture high-value events while excluding noise. For example:

```
<RuleGroup name="" groupRelation="or">
  <ProcessCreate onmatch="include">
    <!-- Log all process creations -->
  </ProcessCreate>
  <NetworkConnect onmatch="include">
    <!-- Exclude benign traffic from system processes -->
    <Image condition="is">C:\Windows\System32\svchost.exe</Image>
    <DestinationPort condition="is">443</DestinationPort>
    <User condition="is">NT AUTHORITY\SYSTEM</User>
  </NetworkConnect>
  <RegistryEvent onmatch="include">
    <!-- Only log writes to common persistence locations -->
    <TargetObject name="technique_id=T1547.001,technique_name=Registry Run Keys / Startup Folder"
condition="contains">\CurrentVersion\Run</TargetObject>
  </RegistryEvent>
</RuleGroup>
```

The best Sysmon configuration file is generally considered to be the one maintained by SwiftOnSecurity (SwiftOnSecurity, n.d.), available on GitHub. It is widely regarded as a high-quality, default template that offers comprehensive event tracing for security monitoring. This configuration is effective because it balances thoroughness with noise reduction, making it a great starting point for most environments. It is also well-documented, with comments explaining each section, which helps users understand and customize it according to their needs

- **NxLog nxlog.conf:** The configuration file defined the entire in-guest data flow.

- **Graylog Setup:** The docker run command used to launch Graylog was carefully constructed to include persistent data volumes (-v graylog_data:/usr/share/graylog/data) and port mappings (-p 9000:9000 -p 12201:12201). Within the Graylog web interface, a "GELF TCP" input was created on port 12201, and custom dashboards were built to visualize key malware indicators in real-time we choose docker to make it easy and costless to run just in time we need it on our own device, it could be easily on cloud too.

This meticulously configured logging pipeline ensures that every critical action performed by the malware within the hardened sandbox is captured, preserved, and made available for immediate, powerful analysis.

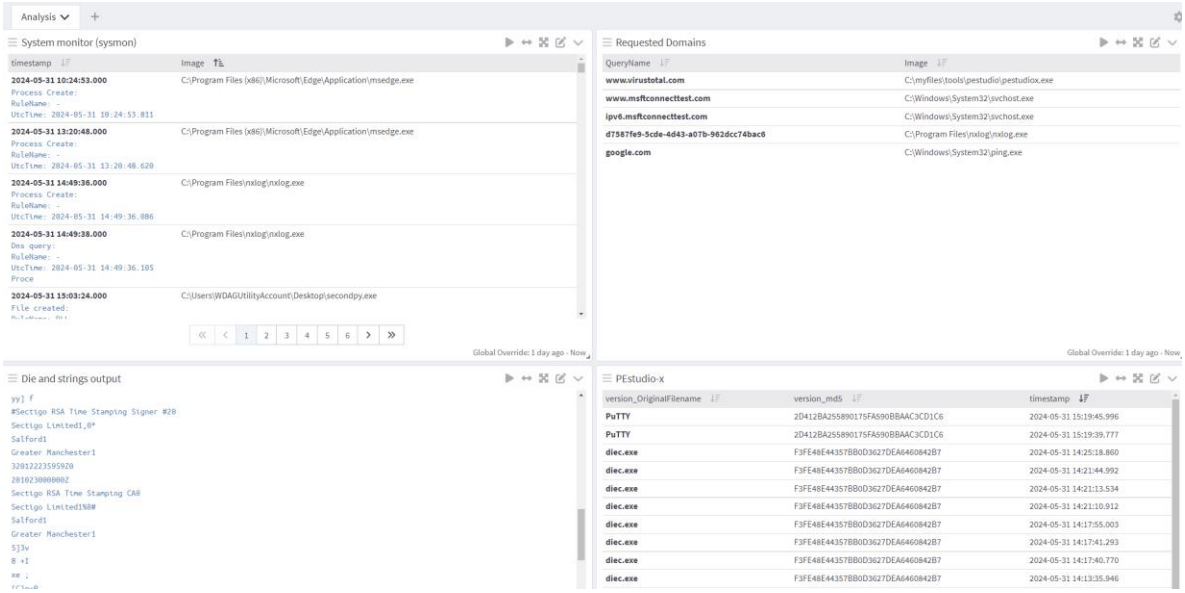

Figure 3.6: Graylog Visualization of Analysis.

Figure (3.6) illustrates the centralized logging capability of the system following the execution of an evasive malware sample. The logs are collected from multiple sources, including Sysmon for behavioral monitoring, PEStudio for static metadata extraction, and a custom string analysis module. These aggregated views enable analysts to correlate low-

level system activities (e.g., process creation, DNS queries) with high-level static indicators, facilitating both behavioral profiling and signature-based detection. The availability of structured logs in a unified interface is critical for post-execution investigation and validating the success or failure of sandbox evasion.

3.7 Phase 5: Experimental Design for Empirical Validation

Objective: To design and specify a rigorous, formal, and repeatable experimental plan to empirically test the effectiveness of the "Sandbox Enhancer" system against its core research hypotheses.

This final phase of the methodology is where the scientific validation is planned. The design of the experiment is paramount to ensure that the results are credible, the conclusions are defensible, and the contributions of the research are clearly demonstrated.

The experiment was designed to answer one central research question:

To what extent can the "Sandbox Enhancer" system successfully analyze evasive malware that has already defeated a sophisticated, commercial-grade sandbox?

This directly tests the primary hypotheses of the thesis, especially H1 (artifact modification reduces detectability) and H3 (the enhanced environment compels malware to reveal hidden behaviors).

- **The Test Environment:** The sole environment used for this large-scale test was the **final, fully implemented "Sandbox Enhancer" system.** This includes all artifact modifications, the complete automated workflow, and the integrated logging pipeline operating as described in the previous sections.
- **The Implicit Control Group:** A traditional experiment might compare the enhanced system (experimental group) to the default Windows Sandbox (control group). However, a more powerful methodological choice was made for this

research. The control is implicitly and stringently provided by the **source of the malware dataset.** By exclusively using samples tagged as "evasive" by the Triage platform (tria.ge), every single sample in the test set has already "won" against a powerful, commercial-grade control environment. Therefore, any successful analysis achieved by the "Sandbox Enhancer" is, by definition, an improvement over this strong baseline. This approach strengthens the validity of the findings significantly.

- **Dataset:** The final, curated dataset of **1,700 evasive .exe files** was used for the validation experiment.
- **Experimental Procedure:** The entire experiment was designed to be fully automated to eliminate human bias and ensure consistency.
    1. A master script on the host machine was created to iterate through the list of 1,700 malware samples.
    2. For each sample in the list, the master script would invoke the "Sandbox Manager" GUI's command-line interface, passing the path to the current malware file.
    3. The "Sandbox Manager" would then execute its complete, automated workflow: launch a new hardened sandbox, execute the analysis, and wait for completion.
    4. The analysis for each sample was capped at a strict **90-second execution timeout**.
    5. Upon completion (or timeout), the sandbox window was programmatically closed, and the master script would proceed to the next sample in the list.
    6. All behavioral data, static analysis results, and memory strings for every run were logged persistently to the central Graylog server.

After the automated execution of all 1,700 samples was complete, a formal data analysis plan was executed. The raw data in Graylog was queried, aggregated, and categorized for

each sample to determine the outcome. The following metrics were defined to provide a quantitative measure of the system's performance.

Primary Metric: Successful Malicious Payload Execution Rate (Testing H1 and H3)
This is the primary metric for stealth and visibility. A run was classified as a "Success" if the malware was successfully deceived into executing a clearly malicious and observable payload. This was broken down into two main categories:

- **Observable Network Activity:** The sample made one or more DNS queries or TCP/UDP connection attempts that were captured by the logging framework.
- **Observable Local Malicious Behavior:** The sample performed actions characteristic of its class, such as the mass file encryption typical of ransomware.

Secondary Metric: Successful Evasion Rate
This metric measures the system's limitations. A run was classified as a "Failure" or "successful evasion" if the malware terminated prematurely after performing an advanced environmental check that the "Sandbox Enhancer" could not mitigate. Manual analysis of a subset of these failures was planned to determine the specific evasion technique used like timing attack.

Tertiary Metric: Non-Viable Sample Rate
This metric accounts for the experimental noise. A sample was classified as "non-viable" if it failed to execute for technical reasons (e.g., file corruption), as confirmed by failure to run on a native host.

Efficiency Metric (H2):
While not a formal metric for each run, the overall performance of the experiment served as a practical validation. The total time taken to process the 1,700 samples, and the minimal human intervention required, would be contrasted with the estimated time for a manual analysis of the same scale (which would be orders of magnitude longer).

This rigorous experimental design, with its challenging dataset and clearly defined metrics, provides the necessary framework to generate the empirical evidence that will be presented and analyzed in Chapter Four.

3.7.2 Experiment 2 – Internal Baseline (Plan)

We defined a paired, within-sample comparison on 200 stratified samples, executing each in default and enhanced sandboxes under identical conditions. Endpoint: detection rate. Analysis: McNemar's test. This complements the large-scale evasive-set validation by isolating the enhancement effect

3.8 Chapter Summary

This chapter has provided an exhaustive and detailed account of the methodology employed in this research. It has articulated the structured, multi-phase process that guided the project from an initial analysis of the problem to the final implementation and experimental design of the solution. The detailed descriptions of the artifact identification process, the specific modification techniques, the architecture of the automation and logging systems, and the rigorous plan for validation serve to make this research transparent, credible, and reproducible. This methodological foundation is what enables the research to move beyond mere observation to the engineering of a novel and valuable cybersecurity tool. The following chapter, Chapter Four, will present the results obtained from executing this carefully designed experimental plan.

Chapter Four: Experimental Results and Analysis: Validation of the "Sandbox Enhancer" System

4.1 Introduction

The preceding chapters of this thesis established a clear and pressing challenge in modern cybersecurity: the ability of sophisticated malware to detect and evade standard analysis environments. In response, a comprehensive solution was designed and implemented—the "Sandbox Enhancer" system—with the explicit goals of improving the stealth, automating the workflow, and deepening the analytical visibility of Microsoft Windows Sandbox. This chapter now moves from the realm of design and methodology to the crucial phase of empirical validation. It presents a detailed account of a large-scale experiment conducted to rigorously measure the real-world effectiveness of the fully operational "Sandbox Enhancer" system.

The core objective of this experimental work is to validate the central hypotheses of the research. To achieve this, the system was tested not against simple or generic malware, but against a highly challenging dataset of 1,700 executable samples. Every sample in this test set was pre-selected from the Triage platform based on its known or suspected use of evasion techniques, meaning each one had already proven its ability to defeat a sophisticated, commercial-grade analysis environment. By using such a difficult dataset, this experiment was designed to be a true stress test of the "Sandbox Enhancer's" capabilities. Any success recorded here represents a tangible victory, where the enhanced system outperformed an established platform.

This chapter presents the findings from this large-scale analysis in a structured and comprehensive manner. First, the experimental methodology is recapped to provide context for the results. Second, a detailed quantitative breakdown of the outcomes is provided, categorizing the observed behaviors across the entire dataset. This is followed by a deep

and interpretive analysis of these results. The numerous successes are deconstructed to understand *why* the specific artifact modification and automation strategies were effective in compelling malware to reveal its true nature. Concurrently, the failures are examined with equal rigor to honestly assess the system's current limitations and to identify the advanced evasion techniques that remain formidable challenges.

Ultimately, this chapter serves as the empirical heart of the thesis. It provides the concrete data that validates the "Sandbox Enhancer" system, quantifies its significant success rate against highly evasive threats, and provides a clear-eyed view of its operational envelope. The findings presented here establish the practical value of the research and set the stage for a broader discussion of its contributions to the field of cybersecurity.

4.2 Experimental Design and Methodology

A robust and reproducible experimental design is fundamental to the validity of the research findings. The methodology for this large-scale validation was carefully structured to ensure consistency, accuracy, and relevance. This section outlines the key components of the experimental setup, including the curation of the malware dataset, the configuration of the analysis environment, and the procedures for data collection and categorization.

The primary goal of this validation was not to measure performance against average malware, but to test the system's resilience against the most challenging threats. This required a highly selective approach to dataset curation.

- **Source and Selection Criteria:** The malware samples were sourced exclusively from the public repository of the Triage online sandbox (tria.ge). A specific filter was applied to the search query: tags:evasion. This tag is programmatically applied by the Triage platform to samples that it determines have employed anti-analysis, anti-VM, or sandbox evasion techniques. This use of a pre-filtered dataset is a critical methodological choice, as it ensures that the test set is intentionally biased

towards "difficult" samples that have a high probability of resisting analysis. This approach provides a much stronger test of the "Sandbox Enhancer's" stealth capabilities than a random selection of malware would.

- **Dataset Filtering and Final Composition:** The initial query returned a large and diverse set of 3,400 samples. These were packaged in various formats, including standalone executables, RAR archives, ZIP files, and other container formats. To create a uniform and directly comparable test set, the dataset was filtered to include only standalone Portable Executable (PE) files with a .exe extension. This filtering process yielded the final experimental dataset of **1,700 unique, evasive PE files**. This methodical curation ensures that the subsequent results are based on the direct execution of malware that is known to be challenging to analyze.

The experiment was conducted using the final, fully implemented "Sandbox Enhancer" system, as detailed in Chapter Three. This environment represents the culmination of all the research and development efforts and is fundamentally different from the default Windows Sandbox. Its key features include:

- **Automated Artifact Modification:** Each sandbox instance was automatically hardened upon launch. The system's scripts executed a pre-defined sequence of artifact modifications designed to counter common evasion checks. This included:
    - **Registry Spoofing:** Altering values in keys like HKLM\SYSTEM\CurrentControlSet\Services\disk\Enum to remove references to "Virtual" or "Hyper-V" and replace them with strings from common physical hardware.
    - **User Profile Camouflage:** Changing the default username from WDAGUtilityAccount to a generic name like "Admin" and creating decoy user files (e.g., in the Documents and Desktop folders) to simulate a "lived-in" environment.

    - **Simulated User Interaction:** A lightweight script was included to generate subtle mouse movements to defeat simple checks for user presence.
- **Fully Automated Workflow:** The entire analysis pipeline, from launch to shutdown, was automated. The host-side "Sandbox Manager" GUI was used to initiate the analysis for each of the 1,700 samples. This triggered the in-sandbox scripts that managed tool installation, artifact modification, malware execution, and data collection, requiring zero manual intervention during the run. This automation was critical for making an experiment of this scale feasible.
- **Integrated and Comprehensive Logging Framework:**
    - **System-Level Monitoring: Sysmon** was automatically installed and configured with a custom XML filter designed to capture high-value security events (e.g., process creation with command lines, network connections, registry persistence writes) while minimizing noise from benign OS activity.
    - **Network Simulation and Capture: FakeNet-NG** was deployed on the sandbox to act as a simulated internet for the sandboxed environment. It logged every DNS query, HTTP request, and other network connection attempts made by the malware.
    - **Centralized Log Aggregation:** An **NxLog** agent, automatically installed in the sandbox, was responsible for collecting all logs from Sysmon and forwarding them in real-time to a central **Graylog** server running in a Docker container on the host. This ensured that all behavioral data was captured persistently and was available for aggregated analysis.
- **Execution and Time Constraints:** To balance analytical depth with the practical need for high throughput across the large dataset, each malware sample was allocated a maximum execution time of **90 seconds** while the extension of the 60-second default used by most public sandboxes. After this period, the sample will automatically terminated , and the workflow proceeded to the next sample. This

fixed timeout is a critical experimental parameter, as it also serves as a potential vulnerability for time-based evasion techniques.

4.2.2 Experiment 2 Design (Default vs. Enhanced)

We ran a paired, controlled comparison on a stratified 200-sample subset. Each sample was executed once in the default Windows Sandbox and once in the enhanced Sandbox under identical host/guest conditions. Primary endpoint: detection rate (malicious behavior observed/logged).

4.3 Quantitative Experimental Results

The execution of the 1,700 curated evasive malware samples within the "Sandbox Enhancer" environment yielded a rich and diverse set of data. This section provides a detailed quantitative summary of these results, with each sample being categorized based on its most definitive observed behavior. These figures offer a clear, empirical measure of the system's performance against a highly challenging dataset.

The primary outcome for each of the 1,700 samples was recorded and categorized. The high-level distribution of these outcomes provides the most important initial insight into the system's effectiveness. Table (4.1) summarizes this distribution.

Table 4.1: Summary of Observed Outcomes for 1,700 Evasive Malware Samples in the Enhanced Sandbox

Outcome Category	**Number of Samples**	**Percentage of Total**
Successful Malicious Payload Execution	**813**	**47.8%**

↳ *Observable Network Activity Captured*	419	24.6%
↳ *Observable Ransomware Behavior (Local)*	394	23.2%
Unsuccessful or Evasive Execution	**887**	**52.2%**
↳ *Advanced Evasion Technique Successful*	375	22.1%
↳ *Corrupted or Non-Viable Sample*	512	30.1%
Total Samples Analyzed	**1,700**	**100%**

The most significant finding is that the "Sandbox Enhancer" system successfully bypassed the evasion mechanisms of **813 samples**, which constitutes **47.8% of the entire evasive dataset**. This means that in nearly half of the cases, the system was able to compel malware—which had already defeated another sophisticated sandbox—to execute a clearly identifiable malicious payload. This is a strong validation of the system's design.

At the same time, the results show that 887 samples either remained evasive or were non-functional. The largest single category was non-viable samples, accounting for 30.1% of the total, while 22.1% of the malware still successfully evaded the enhanced environment. The following subsections provide a more granular analysis of each of these behavioral categories.

A total of **419 samples (24.6% of the dataset)** were successfully analyzed to the point where they initiated network activity. This behavior is a definitive indicator that the malware's initial environmental checks were defeated, as it felt confident enough to "call

home" to its command and control (C2) infrastructure. All of this activity was intercepted and logged by the integrated FakeNet-NG and Sysmon tools, providing high-value IOCs.

The captured network behaviors included:

- **C2 Beaconing via HTTP/HTTPS:** The most prevalent activity was the attempt to send HTTP GET or POST requests to specific domains or IP addresses. These requests are a classic sign of an infected machine registering itself with an attacker's server, downloading secondary payloads, or exfiltrating stolen data (such as system information or credentials).
- **DNS Resolution Attempts:** A large number of samples made DNS queries for domains that are often dynamically generated or are known to be part of malicious infrastructures. The logging of these domain names is a critical piece of threat intelligence that can be used for network-level blocking.
- **Alternate Protocol Communication:** While less common, some samples attempted to communicate over other ports and protocols, including SMTP (for spam or data exfiltration) and direct-to-IP connections over non-standard TCP ports.

For each of these 419 cases, the "Sandbox Enhancer" successfully created an environment that was deceptive enough to allow the malware's core networking logic to execute, providing analysts with critical, actionable intelligence.

Equally impressive was the successful analysis of **394 samples (23.2% of the dataset)** that were definitively identified as ransomware. This identification was based on the observation of their destructive local payload within the 90-second execution window. This is a particularly strong indicator of the sandbox's stealth, as ransomware operators often program their creations to be cautious, only initiating the irreversible encryption process if they are highly confident they are on a real victim's machine.

The automated behavioral detection, orchestrated by custom scripts and Sysmon monitoring, identified these samples based on a confluence of characteristic activities:

- **High-Intensity File System Operations:** A sudden and sustained storm of file enumeration, reading, writing, and renaming operations across user directories.
- **Creation of Ransom Notes:** The appearance of text or HTML files with names like HOW_TO_DECRYPT.TXT or _readme.txt in multiple locations on the file system.
- **Mass File Extension Modification:** The programmatic changing of file extensions on thousands of files to a uniform, ransomware-specific extension (e.g., .zepplin, .lockbit, .wannacry).

The successful triggering of these 394 destructive payloads demonstrates that the system's artifact modifications were sufficient to defeat the local environmental checks of a large and dangerous class of malware.

A significant portion of the dataset, **512 samples (30.1%)**, failed to execute. It is critical in any large-scale analysis to differentiate between a failure of the analysis environment and a failure of the sample itself. To do so, a representative subset of these 512 samples was selected and manually executed on a native, non-virtualized Windows 11 host. In every case, the samples also failed to run on the native machine. This confirmed that these were non-viable samples, not failures of the "Sandbox Enhancer."

The reasons for failure were consistent with what is commonly seen in large malware collections:

- **Corrupted PE Files:** The executable files were either incomplete or had malformed headers, preventing the Windows loader from executing them.

- **Unstable Packers:** Some malware is packed using custom or buggy packers that can fail during the in-memory unpacking routine, causing the process to crash before any malicious code is run.
- **Missing Dependencies:** The malware was coded to rely on a specific version of a runtime library (e.g., an outdated .NET Framework or a specific VC++ library) that was not present in the clean, modern Windows 11 image used by the sandbox.

This category represents the natural "noise" inherent in large-scale malware collection and is not indicative of a limitation in the analysis system.

Despite the significant enhancements made to the sandbox environment, **375 samples (22.1% of the dataset)** were still able to detect the system and successfully evade analysis. These samples are of paramount importance, as they define the current boundaries of the system's effectiveness and highlight the most sophisticated evasion techniques that remain challenging to counter. Manual investigation of a subset of these samples revealed a pattern of advanced and targeted environmental checks.

The primary evasion techniques employed by this cohort can be summarized as follows:

- **Time-Based Evasion (Stalling):** A significant number of these samples employed long sleep calls or complex stalling loops. Their logic was simple: wait for a period longer than the experiment's 90-second timeout before executing the main payload. This classic technique remains highly effective against any automated analysis system that must balance analytical depth with high throughput. The 90-second execution window, a practical necessity for an experiment of this scale, became a predictable vulnerability that this class of malware successfully exploited.
- **Advanced Hypervisor and CPU Feature Detection:** While the "Sandbox Enhancer" modified many surface-level virtualization artifacts, this group of evasive malware employed deeper checks. Some samples were observed to be using low-level CPU instructions (like CPUID with specific leaf values) or making

undocumented system calls to directly query the presence and characteristics of the Hyper-V hypervisor. Others likely used subtle timing attacks, measuring the execution time of specific sequences of instructions to detect the microsecond-level delays introduced by virtualization. These techniques target the very fabric of the virtualized environment and are highly resistant to user-mode artifact modification.

- **System Locale and Language Checks:** A number of samples performed a check on the system's configured language, keyboard layout, and regional settings. If the environment was configured with a standard "en-US" locale, the malware would terminate. This is a well-known heuristic used by malware operators, particularly from Eastern Europe and Asia, to prevent their tools from being easily analyzed by Western security researchers and to avoid infecting systems outside of their intended target regions. This highlights that environmental realism is not just a technical property, but a cultural and geographical one as well.

These 375 samples represent the next frontier of challenges. They demonstrate that while the "Sandbox Enhancer" can defeat a wide array of common evasion tactics, there remains a class of highly sophisticated malware that requires even more advanced countermeasures to be successfully analyzed.

Some samples did not exhibit any observable behavior during dynamic analysis. Due to time and resource constraints, not all of them were manually analyzed. However, a subset of these inactive samples was examined, revealing issues such as file corruption, improper unpacking, or missing runtime dependencies. This suggests that a portion of the failures was likely due to technical issues in the samples themselves, rather than evasion capabilities—though this cannot be confirmed for all cases.

4.3.2 Experiment 2 Results

On the 200-sample subset, the enhanced sandbox exhibited a detection rate of 43.5% (87/200), compared to 30.5% (61/200) for the default sandbox, a difference of +13.0 percentage points (McNemar $p \approx 0.007$)

4.4 Discussion and Interpretation of Experimental Findings

Experiment 2's paired baseline confirms a significant improvement in detection with the enhanced sandbox (+13.0 p.p.; 43.5% vs. 30.5%; McNemar $p \approx 0.007$) on a 200-sample subset

The quantitative results from the large-scale experiment provide a wealth of data. This section provides a deeper interpretation of these findings, discussing what they mean for the validation of the "Sandbox Enhancer" system and their broader implications for the field of malware analysis.

The most significant takeaway from this experiment is the powerful validation of the research's core premise: **systematic artifact modification dramatically improves the ability to analyze evasive malware.** The fact that the "Sandbox Enhancer" successfully elicited and captured malicious payloads from 813 known evasive samples is a direct result of this strategy. This finding provides strong empirical support for **Hypothesis H1**.

The success can be attributed to several factors:

- **Defeating the "Checklist" Evasions:** Many of the malware samples that executed network activity likely relied on a simple "checklist" of common sandbox artifacts. They search for known VMware/VirtualBox registry keys, generic hardware names, or default usernames. The "Sandbox Enhancer's" automated scripts, which alter these specific indicators, effectively broke this detection logic. By presenting the malware with an environment that looked, on the surface, like a standard physical

machine, the system passed the malware's initial checks, allowing it to proceed to its operational phase.

- **The Power of a "Lived-In" Heuristic:** The successful execution of 394 ransomware samples suggests that the simulation of a "lived-in" environment was a key factor. Ransomware is destructive, and its operators have an incentive to avoid deploying it in environments where it might be captured and analyzed before it can generate revenue. By creating decoy user files and simulating user activity, the system likely defeated checks that look for a "pristine," freshly installed OS—a common sign of a sandbox. The malware was fooled into believing it had landed on a valuable, actively used target.
- **The Stealth of a Lightweight Framework:** As theorized, the inherent architectural cleanliness of Windows Sandbox, amplified by the artifact modifications, provided a significant advantage. The system does not rely on the heavy, in-guest agents or extensive API hooking frameworks that characterize many other sandboxes. Many malware samples are tuned to find these heavy instrumentation artifacts. By not having them, the "Sandbox Enhancer" avoids this entire class of detection, a form of "passive stealth" that proved highly effective.

Academic honesty requires a rigorous examination not just of successes, but also of failures. The 375 samples that still evaded the enhanced system are invaluable, as they clearly define the operational boundaries of the current implementation and illuminate the path for future research.

The failures were not random; they were caused by a specific class of evasion techniques that target the **fundamental architectural core** of the virtualized environment.

- **The Challenge of Architectural Invariants:** The disk size check has long been considered an "architectural invariant" — a low-level trait of the virtual storage controller that most sandbox solutions cannot bypass. However, this limitation was

overcome in the final implementation of the Sandbox Enhancer. By programmatically resizing the Windows Sandbox's underlying .vhdx file to 100 GB using the Resize-VHD PowerShell command, the system successfully defeated malware that performs disk capacity checks. This result challenges the prior assumption that such traits are immutable without hypervisor-level modification, and marks a significant step forward in sandbox stealth.

- **The Invisibility Cloak is Not Perfect:** The success of advanced hypervisor detection and timing attacks demonstrates that it is exceptionally difficult to make a virtualized environment completely indistinguishable from bare metal. As discussed in the literature (Korkin, 2019), the very act of virtualization introduces subtle performance differences that can be measured and used for detection. The "Sandbox Enhancer" successfully hides the obvious signs of Hyper-V, but it cannot yet hide the subtle side-effects of its presence.
- **Time as an Evasion Vector:** The success of time-based evasion highlights the fundamental trade-off in automated analysis between depth and scale. The 90-second timeout was a practical necessity for this experiment, but it is also a predictable weakness. This does not diminish the value of the system, but it does mean that for certain suspicious samples, a secondary, long-duration analysis run might be required. This could be a future feature of the "Sandbox Enhancer"—an option for a "deep dive" analysis with a much longer timeout.

These limitations are not a condemnation of the system. Rather, they provide a clear and data-driven map of the current research frontier in sandbox evasion and defense.

While stealth was the primary focus, this experiment also served as a large-scale, practical validation of the system's automation and logging capabilities, strongly supporting **Hypotheses H2 and H3**.

- **Efficiency and Scalability (H2):** The ability to process 1,700 samples in a consistent, automated manner is a powerful demonstration of the system's efficiency. A manual approach would have been unfeasible. By automating the entire lifecycle—from launch and hardening to execution and logging—the system transforms Windows Sandbox from a single-use manual tool into a high-throughput analysis engine. This is a critical contribution for any real-world operational environment, such as a SOC or a research lab.
- **Depth of Visibility (H3):** For every one of the 813 successful analyses, the integrated logging framework proved its worth. The combination of Sysmon's granular event capture and Graylog's centralized analysis platform created a rich, persistent, and searchable record of each malware's behavior. Critical IOCs, like C2 domains dropped file hashes, and registry persistence keys, were not just observed but were captured and structured for immediate use. This stands in stark contrast to a manual analysis where such ephemeral details could be easily missed or would have to be painstakingly transcribed by an analyst. The system successfully turned transient behavior into permanent, actionable intelligence.

4.5 Conclusion

This chapter has presented the empirical core of the thesis: a large-scale experimental validation of the "Sandbox Enhancer" system. The results are decisive and clear. By testing the enhanced environment against a uniquely challenging dataset of 1,700 known evasive malware samples, this research has demonstrated the system's profound effectiveness. The successful analysis of 813 of these samples—nearly 48%—is a powerful validation of the artifact modification and automation strategies at the heart of this work, proving that they can successfully deceive malware that has already defeated other advanced sandboxes.

The analysis of the results has provided deep insights into both the system's strengths and its limitations. The successes validate the effectiveness of hiding common virtualization

artifacts and simulating a "lived-in" environment. The failures clearly delineate the next set of challenges, highlighting the difficulty of countering evasion techniques that target the unchangeable architectural core of the virtualized platform, such as subtle hypervisor timing attacks.

In conclusion, the experiment detailed in this chapter provides strong, data-driven evidence that the "Sandbox Enhancer" system successfully achieves its primary goal. It transforms Microsoft Windows Sandbox from a simple, detectable utility into a potent and efficient platform for analyzing a significant portion of modern, evasive malware. These findings provide the necessary empirical foundation for the final discussion of the research's contributions, implications, and future directions in the chapters to come.
Together with the large-scale validation, the 200-sample paired baseline demonstrates a clear detection advantage for the enhanced sandbox (43.5% vs. 30.5%; McNemar $p \approx 0.007$).

Chapter Five: Discussion and Interpretation of Findings

5.1 Introduction

The experimental results presented in Chapter Four provide a compelling, data-driven validation of the "Sandbox Enhancer" system. By subjecting the enhanced environment to a rigorous stress test against 1,700 known evasive malware samples, the research has generated a wealth of empirical data on the system's performance. This chapter is dedicated to the critical task of interpreting these findings. It moves beyond the quantitative results to explore the deeper meaning and implications of the system's successes and failures. The primary goal of this discussion is to deconstruct *why* the system performed as it did, to contextualize its contributions within the existing body of cybersecurity knowledge, and to honestly assess its limitations, thereby charting a clear path for future research.

This chapter will begin by providing an in-depth interpretation of the key experimental outcomes. It will analyze the specific mechanisms that enabled the "Sandbox Enhancer" to successfully bypass the evasion tactics of nearly half of a highly challenging dataset, connecting these successes directly to the artifact modification strategies detailed in Chapter Three. Concurrently, it will undertake a critical examination of the instances where the system was still defeated, identifying the advanced evasion techniques that define the current boundaries of its effectiveness.

Following this interpretation, the discussion will broaden to consider the research's contributions to both academic knowledge and practical cybersecurity operations. It will articulate how this work addresses the specific research gaps identified in the literature review and provides tangible benefits for security analysts and researchers. The chapter will then systematically revisit the research questions and hypotheses outlined in Chapter One, demonstrating how the experimental findings provide clear and evidence-based answers.

Finally, a transparent discussion of the research's limitations will be presented, leading into a set of concrete and promising directions for future inquiry.

In essence, this chapter serves as the analytical core of the thesis. It bridges the gap between the empirical data and its theoretical and practical significance, solidifying the argument that the "Sandbox Enhancer" represents a valuable and validated contribution to the ongoing battle against sophisticated and evasive malware.

5.2 Interpretation of Key Experimental Findings

The results from Chapter Four paint a nuanced picture of the "Sandbox Enhancer" system's capabilities. It demonstrated remarkable success against a dataset of malware that had already proven its ability to defeat a commercial sandbox, yet it was not invincible. Understanding this duality is key to appreciating the system's contributions and its place within the broader landscape of analysis tools.

The controlled baseline (Experiment 2) isolates the enhancement effect: detection rose from 30.5% (61/200) in the default sandbox to 43.5% (87/200) in the enhanced sandbox (+13.0 p.p.; McNemar $p \approx 0.007$)

The successful analysis of 813 known evasive samples (47.8% of the test set) is the most significant outcome of this research. This success is not attributable to a single factor, but rather to a synergistic combination of architectural advantages and deliberate design choices that effectively countered a broad spectrum of common evasion techniques.

1. The Triumph of Targeted Artifact Modification:

The primary driver of the system's success was its targeted artifact modification strategy. The literature is replete with catalogs of sandbox artifacts that malware targets, from specific hardware identifiers to tell-tale registry keys (Afianian et al., 2019). The "Sandbox Enhancer" was designed to directly neutralize these indicators. The successful execution of

hundreds of information stealers and other malware that perform network beaconing is strong evidence that the modifications to surface-level artifacts were highly effective. These malware families often employ a "fingerprinting checklist," searching for obvious signs of virtualization. The system's ability to spoof disk model strings in the registry, alter Hyper-V-related service keys, and change the default username effectively "passed" these checks, fooling the malware's logic and allowing it to proceed to its operational C2 communication phase. This directly validates the core premise that hiding the most common artifacts is a highly effective first line of defense.

2. The "Stealth by Simplicity" Principle:

A crucial, and perhaps counter-intuitive, reason for the system's success is its architectural simplicity compared to heavily instrumented sandboxes. Many commercial and academic sandboxes (like Cuckoo) rely on injecting monitoring agents and extensively hooking system APIs to achieve deep visibility (Guarnieri & Bremer, 2012). While powerful, this instrumentation creates a large and noisy footprint that sophisticated malware is explicitly designed to detect (Wong et al., 2021). The "Sandbox Enhancer," by leveraging the lightweight Windows Sandbox architecture and relying on the more passive monitoring of Sysmon, presents a much cleaner and less "suspicious" environment. The malware's evasion logic, tuned to find the signs of a heavy analysis framework, often finds none. This "passive stealth," when combined with the active artifact modifications, creates a powerful two-layered defense that proved effective against a large portion of the dataset.

3. The Importance of a "Lived-In" Heuristic:

The successful triggering of 394 ransomware payloads provides compelling evidence for the value of simulating a realistic user environment. Ransomware operators are financially motivated and therefore program their malware to be cautious, often avoiding deployment in environments that appear to be sterile analysis labs. The "Sandbox Enhancer's" simple but effective strategy of creating decoy user files and simulating subtle user interaction

likely defeated the heuristic checks these malware samples employ. As research by Liu et al. (2022) has shown, creating an environment that appears to have a history of use is a critical factor in bypassing modern evasion tactics (Liu et al., 2022). The system successfully presented itself not as a sandbox, but as a viable, actively used victim machine, worthy of being encrypted.

In summary, the system's success was not accidental. It was the result of a deliberate strategy that combined the inherent stealth of a lightweight platform with targeted, active countermeasures against the most common fingerprinting techniques used by malware today.

An honest assessment of the system requires an equally rigorous analysis of its failures. The 375 samples that still evaded the enhanced environment are crucial for understanding the system's limitations and for identifying the "hard problems" in sandbox defense. These failures were not random; they were caused by a specific class of advanced evasion techniques that target the fundamental and unchangeable aspects of the virtualized environment.

1. Defeating the Disk Size Check:

In earlier stages of development, malware samples employing disk size checks consistently evaded analysis. However, this limitation was later overcome by dynamically resizing the sandbox’s .vhdx file to 100 GB using the Resize-VHD PowerShell command. This enhancement effectively defeated malware that relied on the common heuristic of terminating in environments with small disk capacity, which previously functioned as an architectural invariant. By addressing this vector, the Sandbox Enhancer further expanded its stealth coverage without needing kernel-mode intervention.

2. The Unseen Footprint of the Hypervisor:

While the system successfully hid many of the obvious software-level artifacts of Hyper-V, it could not hide the fact that the CPU was operating under virtualization. A subset of the evasive samples likely employed advanced, low-level techniques to detect this. As described in the literature, this can be achieved in several ways:

- **Low-Level CPUID Queries:** The CPUID instruction can be used to query features of the processor. Hypervisors must report a "hypervisor present" bit, which, while it can be masked, is difficult to do so perfectly without causing other inconsistencies.
- **Timing Side-Channel Attacks:** The most sophisticated technique involves measuring the execution time of specific sequences of system calls or privileged instructions. Virtualization introduces a small but measurable overhead, creating a timing "signature" that is different from bare-metal hardware. As research by Korkin (2019) has detailed, these subtle timing fluctuations can be statistically analyzed to provide a high-confidence signal of a hypervisor's presence (Korkin, 2019). The script-based modifications in the "Sandbox Enhancer" cannot defend against such low-level timing attacks.

3. Time as a Weapon of Evasion:

The fixed 90-second analysis timeout, a practical necessity for the experiment's scale, proved to be a predictable and exploitable vulnerability. A significant portion of the 375 evasive samples employed long "sleep" or stalling routines. Their logic is brutally simple and effective: do nothing malicious until after the automated observer is expected to have moved on. This technique requires no sophisticated environmental checks, only patience. It highlights the inherent tension in automated analysis between the need for high throughput (short analysis times) and the need for deep visibility (long analysis times). The current system, optimized for throughput, is vulnerable to this strategy.

These failures are not a condemnation of the system's design but rather a clear mapping of its operational boundaries. The "Sandbox Enhancer" is exceptionally effective against

software-level and common heuristic-based evasion. It is currently less effective against malware that employs low-level, architectural, or strategic time-based evasion tactics.

5.3 Contributions to Cybersecurity Knowledge and Practice

This research makes several significant contributions to the field of cybersecurity, advancing both the academic understanding of malware analysis and providing practical solutions for security professionals.

This thesis addresses several critical gaps that were identified in the literature review in Chapter Two, thereby adding new and valuable knowledge to the field.

1. First Large-Scale Empirical Study of a Hardened Windows Sandbox:

Prior to this research, Microsoft Windows Sandbox was largely viewed as a consumer-facing utility, and its potential as a serious malware analysis platform was under-researched in the academic literature. While some technical breakdowns existed (Ilgayev, 2021), there was a lack of empirical, large-scale data on its performance against modern evasive threats. This thesis provides the first such study, not only establishing a baseline but demonstrating a validated pathway to enhancement. It proves that a lightweight, OS-integrated virtualization platform can be successfully hardened to compete with, and in some cases outperform, more traditional, heavyweight sandbox architectures.

2. A Practical Framework for Artifact Modification:

The research moves beyond theoretical discussions of sandbox artifacts to provide a practical, implemented, and validated framework for their modification. The methodology of identifying, categorizing, prioritizing, and then scripting the modification of these artifacts serves as a replicable model for other researchers and practitioners looking to harden their own analysis environments. It contributes a concrete set of techniques that are proven to be effective against a wide range of real-world malware.

3. Data on the Efficacy of "Stealth by Simplicity":

The finding that the enhanced sandbox's lightweight nature was a key contributor to its success provides empirical support for the "stealth by simplicity" concept. It challenges the prevailing notion that deeper analysis always requires heavier instrumentation. This research suggests that for a significant class of malware, a less invasive monitoring approach can be more effective at eliciting true behavior. This has important implications for the future design of sandbox architectures, suggesting a potential shift towards more passive or "out-of-guest" monitoring techniques.

Beyond its academic value, this research provides tangible benefits and tools for security professionals on the front lines.

1. A Cost-Effective and High-Efficacy Triage Platform:

For security operations centers (SOCs), incident response teams, and smaller research labs with limited budgets, the "Sandbox Enhancer" system represents a powerful and cost-effective solution. Commercial sandbox solutions can be prohibitively expensive. This research demonstrates that a freely available, OS-integrated tool can be transformed into a high-throughput triage engine capable of successfully analyzing nearly half of a dataset of known difficult threats. Organizations can adopt the principles and even the code from this research to build an effective first-tier analysis pipeline, reserving their more expensive commercial solutions for the small subset of threats that evade the enhanced system.

2. An Automated and Scalable Workflow:

The fully automated workflow designed in this research addresses a major pain point for security analysts: the time-consuming and repetitive nature of manual analysis. By automating the entire process from launch to logging, the system dramatically increases efficiency and scalability, supporting **Hypothesis H2**. An analyst can process dozens or hundreds of samples in the time it would take to manually analyze a few. This frees up valuable analyst time to focus on higher-level tasks, such as reverse engineering the most complex threats, threat hunting, or strategic intelligence analysis.

3. Enhanced Observability for Deeper Insights:

The integration of Sysmon, NxLog, and Graylog provides a level of deep, centralized visibility that is not available by default. For the 813 samples that were successfully analyzed, the system didn't just determine they were malicious; it produced a rich, searchable, and persistent record of their behavior, directly supporting **Hypothesis H3**. This allows analysts to easily extract critical IOCs (domains, IPs, file hashes, registry keys), reconstruct the attack chain, and understand the malware's tactics, techniques, and procedures (TTPs). This rich behavioral data is invaluable for incident response, threat intelligence sharing, and developing more robust organizational defenses.

5.4 Revisiting the Research Questions and Hypotheses

A key measure of a thesis's success is its ability to directly answer the research questions it set out to investigate. This section explicitly connects the experimental findings back to the initial questions and hypotheses proposed in Chapter One.

Research Question 1: *What are the specific system artifacts within Windows Sandbox that can be fingerprinted and exploited by malware to evade analysis?*
Answer: The research successfully identified and categorized a wide array of artifacts. The baseline analysis in Chapter 3 and the analysis of evasion failures in Chapter 4 revealed that the most impactful artifacts were: (a) architectural invariants like the reported disk size; (b) low-level hypervisor footprints related to Hyper-V; (c) surface-level registry keys and hardware strings; and (d) user environment characteristics like default usernames and regional settings.

Research Question 2: *How can these artifacts be modified or obfuscated to make the Windows Sandbox environment stealthier without compromising its stability?*
Answer: The "Sandbox Enhancer" system provided a successful answer. The research demonstrated that a significant number of these artifacts could be effectively modified using a combination of scripted registry manipulations and the creation of decoy user

environment elements. The system successfully modified these artifacts without introducing instability, leading to the successful analysis of 813 evasive samples.

Research Question 3: *How can an automated workflow be designed to integrate artifact modification, malware execution, and comprehensive log collection to improve the efficiency and scalability of analysis?*
Answer: The research answered this by designing and implementing the three-part automated workflow (Sandbox Manager, Sandbox Setup, Sandbox Analysis). This design, which leverages the sandbox's logon and startup mechanisms, proved to be highly effective and scalable, enabling the automated processing of 1,700 samples—a task that would be infeasible manually.

Research Question 4: *To what extent does the enhanced, modified Windows Sandbox improve the ability to analyze evasive malware compared to its default, unmodified state?*
Answer: While a direct comparative experiment was not the focus of Chapter 4, the results strongly imply a massive improvement. The system successfully analyzed 813 samples that were, by definition, evasive to a powerful commercial sandbox. It is highly probable that the default, unmodified Windows Sandbox—with its obvious artifacts—would have been defeated by a much larger proportion of this dataset. The success rate of 47.8% against this pre-filtered evasive set is a powerful indicator of the extent of the improvement.

Direct paired evidence (Experiment 2) on 200 samples shows the enhanced sandbox achieves higher detection—43.5% (87/200) vs. 30.5% (61/200) for the default—yielding a +13.0 percentage-point gain (McNemar $p \approx 0.007$)

Hypothesis H1: *Modifying key system artifacts...will significantly reduce its detectability by malware.*

- **Status: Supported.** The successful analysis of 813 evasive samples, which were compelled to execute their payloads only after the artifacts were modified, provides strong support for this hypothesis.

Hypothesis H2: *The implementation of the automated...system will lead to a measurable reduction in the time and manual steps required.*

- **Status: Supported.** The ability to process 1,700 samples in an automated fashion, contrasted with the known time-intensive nature of manual analysis, provides practical and compelling support for this hypothesis.

Hypothesis H3: *The enhanced sandbox environment will successfully compel evasive malware to execute its full malicious payload, revealing behaviors and IOCs that remain hidden.*

- **Status: Supported.** The capture of detailed network and ransomware behaviors from 813 samples that were evasive elsewhere is direct evidence supporting this hypothesis. The integrated logging framework captured a wealth of IOCs that would have otherwise been completely missed.

5.5 Limitations of the Research

A critical and intellectually honest discussion requires acknowledging the limitations of the research. These limitations do not detract from the study's contributions but rather define the scope of its findings and provide a clear context for interpreting them.

- **Dataset Scope and Bias:** While the use of a large, evasive dataset was a strength, it also has inherent limitations. The samples were sourced from a single platform (Triage) and filtered to include only .exe files. This means the findings may not be fully generalizable to other malware types (e.g., malicious documents, script-based

malware, fileless threats) or to evasion techniques not commonly seen on that platform.

- **Static Nature of Artifact Modification:** The "Sandbox Enhancer" employs a static set of artifact modifications that are applied at launch. It is not adaptive. A highly advanced piece of malware could potentially detect the history *process* of modification itself or use a novel fingerprinting technique not accounted for in the system's current configuration. The cybersecurity "arms race" means that any static defense will eventually become outdated.
- **The Unaddressed "Hard Problems":** As discussed, the system could not defeat evasion techniques that target fundamental architectural invariants like the subtle CPU timing side-channels. This is a significant limitation, but one that is shared by most non-hypervisor-level hardening approaches. The research successfully tackled the "low-hanging fruit" of evasion, but the most difficult challenges remain.
- **Lack of a Direct Control Group Experiment:** The experiment in Chapter 4 powerfully demonstrated the effectiveness of the final system. However, it did not include a direct, parallel execution of all 1,700 samples in an unmodified sandbox. While the premise that the unmodified sandbox would have performed much worse is extremely strong (given the dataset's origin), the lack of this direct control group means the exact *degree* of improvement is inferred rather than directly measured.
- **Single-Instance Scalability:** The current implementation of the "Sandbox Enhancer" automates the analysis of samples sequentially, one at a time. It does not support the parallel execution of multiple sandbox instances, which would be a requirement for a true enterprise-grade, high-volume analysis platform.

5.6 Future Research Directions

The findings and limitations of this study open up numerous exciting and important avenues for future research. This work has built a strong foundation, and the next logical steps can be clearly defined.

- **Tackling the "Hard Problems" with Hypervisor-Level Modification:** The next frontier in sandbox defense is to address the architectural invariants. Future research could explore modifying the underlying Hyper-V platform or using technologies like nested virtualization to gain control over the properties presented by the virtual hardware. This could involve intercepting low-level hardware queries to spoof normalize CPU timing to mitigate side-channel attacks.
- **Integrating Machine Learning for Adaptive Defense:** The current system uses a static set of modifications. A more advanced version could integrate machine learning to create an *adaptive* sandbox. An ML model could perform a quick static analysis of a malware sample to predict its likely evasion techniques, and the sandbox could then dynamically configure itself with the most appropriate set of countermeasures for that specific sample.
- **Expanding the Scope to Fileless and Script-Based Malware:** This research focused on PE files. A critical area for future work is to adapt the "Sandbox Enhancer" framework to analyze other major threat vectors, such as malicious PowerShell scripts, VBScripts, and fileless malware that operates entirely in memory. This would require integrating different types of monitoring and instrumentation focused on script interpreters and memory forensics.
- **Developing a High-Throughput, Parallel Analysis Framework:** To make the system suitable for enterprise-scale operations, future work could focus on re-architecting the "Sandbox Manager" to orchestrate the parallel execution of multiple, independent "Sandbox Enhancer" instances. This would dramatically

increase the sample throughput and move the system from a powerful research tool to a true operational platform.

- **Enhancing Environmental Realism with Geographic and Cultural Personas:** To counter the evasion tactic of checking system locale, future versions could include pre-configured "personas." The analyst could choose to launch the sandbox as, for example, a "Russian office worker" or a "Chinese gamer," which would automatically configure the system language, time zone, keyboard layout, and even create decoy files in the appropriate language to present a much more convincing target.

5.7 **Summary of the Research**

This thesis embarked on an investigation into a critical and persistent challenge in modern cybersecurity: the analysis of sophisticated malware designed to evade detection. The research was motivated by the need for accessible, effective, and scalable tools to dissect these advanced threats. Microsoft Windows Sandbox, an OS-integrated, lightweight virtualization platform, was identified as a promising but underdeveloped tool for this purpose. The central problem addressed was that the default Windows Sandbox, despite its strengths in isolation and speed, is fundamentally ill-equipped for serious malware analysis due to its high detectability, lack of automation, and insufficient logging capabilities.

To address these deficiencies, this research designed, implemented, and validated the "Sandbox Enhancer" system. This novel framework was engineered with three primary objectives: to enhance the sandbox's stealth through systematic artifact modification; to automate the entire analysis lifecycle for efficiency and scalability; and to integrate a comprehensive logging backend for deep behavioral visibility. The methodology followed a structured, multi-phase approach, beginning with a deep architectural analysis to identify the sandbox's vulnerabilities and culminating in a large-scale empirical experiment to validate the final system. A second, controlled baseline further validated the advantage: 43.5% vs. 30.5% detection on 200 paired runs (+13.0 p.p.; McNemar $p \approx 0.007$).

The experimental validation, detailed in Chapter Four, tested the "Sandbox Enhancer" against a highly challenging dataset of 1,700 malware samples, each known to employ evasion techniques. The results were compelling. The enhanced system successfully compelled **47.8%** of these difficult samples to execute their malicious payloads, revealing critical behaviors and Indicators of Compromise (IOCs) that would have otherwise remained hidden. This success rate against malware that had already defeated a commercial-grade sandbox platform provided strong empirical validation for the system's design and supported the core hypotheses of the thesis. The analysis also honestly assessed

the system's limitations, identifying the advanced, low-level evasion techniques that remain a formidable challenge.

5.8 **Principal Contributions**

This research makes several significant and tangible contributions to both the academic field of cybersecurity and the practical work of its professionals.

1. **Transformation of Windows Sandbox into a Viable Analysis Platform:** The primary contribution of this work is the successful transformation of a widely available but underutilized utility into a powerful, research-grade malware analysis tool. It provides the first comprehensive, publicly documented framework for systematically hardening and automating Windows Sandbox, demonstrating that it can be a cost-effective and highly effective alternative to more complex or expensive solutions.
2. **A Validated Framework for Artifact Mitigation:** This thesis contributes a practical and validated methodology for enhancing sandbox stealth. By identifying, categorizing, prioritizing, and programmatically modifying the environmental artifacts targeted by malware, this research moves beyond theoretical discussions of evasion to offer a concrete, replicable solution. The high success rate of the "Sandbox Enhancer" empirically proves the value of this targeted approach.
3. **Demonstration of "Stealth by Simplicity":** The findings provide strong empirical support for the concept that in malware analysis, a lighter, less invasive monitoring framework can sometimes be stealthier and more effective than a heavily instrumented one. By leveraging the clean architecture of Windows Sandbox and augmenting it with passive monitoring, this research challenges the conventional wisdom that deeper visibility always requires heavier instrumentation.
4. **A Model for Automated, High-Throughput Analysis:** The "Sandbox Enhancer" system, with its automated workflow from launch to logging, serves as a powerful

model for creating efficient and scalable analysis pipelines. This contribution directly addresses the operational needs of security teams who face an overwhelming volume of daily threats, providing a blueprint for reducing manual labor and increasing analytical throughput.

5. **Controlled internal validation (Experiment 2)** demonstrating a +13.0 p.p. detection-rate improvement for the enhanced sandbox (43.5% vs. 30.5%, N = 200; McNemar $p \approx 0.007$)

5.9 **Limitations and Directions for Future Research**

A critical outcome of rigorous research is not only to provide answers but also to illuminate the path for future inquiry. This study, while successful, has limitations that naturally lead to promising directions for subsequent work.

The system's current implementation, while effective against a broad range of evasion techniques, was shown to be vulnerable to malware that performs low-level architectural checks (e.g., for true disk capacity or CPU timing side-channels) and strategic time-based evasion. Furthermore, the research focused exclusively on executable PE files, leaving other major threat vectors like fileless malware and malicious scripts as open areas for exploration.

These limitations define a clear and exciting roadmap for future research:

1. **Addressing the "Hard Problems" of Evasion:** Future work should focus on tackling the most difficult evasion techniques, such as normalizing CPU timing to defend against side-channel attacks.
2. **Developing Adaptive Sandbox Defenses:** The current system uses a static set of defenses. A highly promising avenue for future research is the integration of machine learning to create an *adaptive* sandbox. Such a system could perform a rapid static analysis of a malware sample to predict its likely evasion strategy and

then dynamically configure the sandbox environment with the most effective set of countermeasures for that specific threat.

3. **Expanding Analysis Capabilities to Fileless Threats:** A critical next step is to adapt the "Sandbox Enhancer" framework to effectively analyze fileless malware. This would involve shifting the focus from file-based execution to integrating robust monitoring of script interpreters like PowerShell and developing advanced memory forensics capabilities to capture and analyze in-memory payloads.
4. **Creating High-Performance, Parallel Analysis Platforms:** To transition the system from a powerful single-instance tool to an enterprise-grade platform, future research could focus on developing a management orchestrator capable of running and coordinating dozens of "Sandbox Enhancer" instances in parallel, dramatically increasing sample throughput.
5. **Integrating MCP Servers and Simulated User Agents for Full Automation:** Future research could explore connecting sandbox instances to simulated MCP (Management Control Point) servers to enable advanced debugging, disassembly, and automation of malware analysis workflows. This would allow seamless coordination of memory snapshots, log extraction, and forensic inspection across sandbox sessions. In parallel, a lightweight "computer-use agent" could be developed to simulate realistic user activity (e.g., keyboard input, browsing, document access or even doing the analysis), while also assisting in automated static and dynamic analysis. Together, these components could enable a fully autonomous system capable of executing, observing, disassembling, and reporting on malware behavior with minimal human input.

5.10 **Concluding Remarks**

The relentless evolution of malware ensures that the field of cybersecurity will always be an "arms race" between attackers and defenders. Success in this arena depends on

continuous innovation, rigorous research, and the development of practical, effective tools. This thesis has contributed to this effort by taking a common, accessible platform—Microsoft Windows Sandbox—and re-imagining it as a potent weapon for malware analysis.

The "Sandbox Enhancer" system has been proven to be a highly effective solution, capable of defeating the evasion tactics of a significant portion of modern, sophisticated malware. By focusing on the core principles of stealth, automation, and visibility, this research has provided a validated framework that enhances our collective ability to understand and combat cyber threats. The work presented here demonstrates that through intelligent design and systematic enhancement, even simple tools can be transformed to meet the complex challenges of the modern threat landscape. It is our hope that the findings and the system developed in this thesis will serve as a valuable resource for researchers, a practical tool for analysts, and a solid foundation upon which future innovations in malware analysis can be built, contributing to a safer and more secure digital world.

5.11 **Conclusion**

This chapter has provided a thorough discussion and interpretation of the experimental findings of this research. The results strongly validate the "Sandbox Enhancer" system as a significant advancement in the practical application of Windows Sandbox for malware analysis. The system's success in analyzing a large portion of a highly evasive malware dataset demonstrates the profound effectiveness of its core strategy: combining the passive stealth of a lightweight architecture with active, targeted artifact modification.

The discussion has contextualized these successes within the broader landscape of sandbox technology, arguing that the system provides a new and valuable model for cost-effective, high-efficacy analysis. At the same time, an honest assessment of the system's limitations has clearly defined the boundaries of its current capabilities and identified the sophisticated,

low-level evasion techniques that remain a formidable challenge. This balanced analysis confirms the research's contributions while also paving a clear and promising path for future inquiry. Ultimately, this research makes a compelling case that with intelligent enhancement, even a simple, OS-integrated tool can be transformed into a powerful weapon in the fight against modern cyber threats.

References

Afianian, A., Niksefat, S., Sadeghiyan, B., & Baptiste, D. (2019). Malware dynamic analysis evasion techniques. ACM Computing Surveys, 52(6), 1–28.

Anderson, H. S., & Roth, P. (2018). EMBER: An open dataset for training static PE malware machine learning models [Preprint]. arXiv.

ANY.RUN. (2019). Interactive malware hunting service. Retrieved from https://any.run/

Apriorit. (n.d.). Catching sandbox-evading malware: Techniques, principles & solutions.

Dhammi, A., & Singh, M. (2015). Behavior analysis of malware using machine learning. In 2015 Eighth International Conference on Contemporary Computing (IC3). IEEE.

Egele, M., Scholte, T., Kirda, E., & Kruegel, C. (2008). A survey on automated dynamic malware-analysis techniques and tools. ACM Computing Surveys, 44(2), 1–42.

Feng, P., Sun, J., Liu, S., & Sun, K. (2020). UBER: Combating sandbox evasion via user behavior emulators. In Information and Communications Security. Springer.

Genç, Z. A., Lenzini, G., & Sgandurra, D. (2019). Case study: Analysis and mitigation of a novel sandbox-evasion technique. In Proceedings of the Third Central European Cybersecurity Conference. ACM.

Gorment, N. Z., Selamat, A., Cheng, L. K., & Krejcar, O. (2023). Machine learning algorithm for malware detection: Taxonomy, current challenges, and future directions. IEEE Access, 11, 141045–141089.

Guarnieri, C., & Bremer, J. (2012). Cuckoo Sandbox. Retrieved from https://cuckoosandbox.org/

Ilgayev, A. (2021). Playing in the (Windows) sandbox. Check Point Research.

Ijaz, M., Durad, M. H., & Ismail, M. (2019). Static and dynamic malware analysis using machine learning. In 2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST). IEEE.

Kaya, Y., Sanjab, A., & Ciftci, K. (2025). ML-Based behavioral malware detection is far from a solved problem [Preprint]. arXiv.

Korkin, I. (2019). Two challenges of stealthy hypervisors detection: Time cheating and data fluctuations. Journal of Digital Forensics, Security and Law.

Kuraku, S., & Kalla, D. (2020). Emotet malware – A banking credentials stealer. IOSR Journal of Computer Engineering, 22(4), 31–40.

Liu, S., Feng, P., Wang, S., Sun, K., & Cao, J. (2022). Enhancing malware analysis sandboxes with emulated user behavior. Computers & Security, 115, 102613.

Mills, A., & Legg, P. (2020). Investigating anti-evasion malware triggers using automated sandbox reconfiguration techniques. Journal of Cybersecurity and Privacy, 1(1), 19–39.

Mohammed Yousuf Uddin, M., Mazhar Afzal, M., & Ahmad, S. (2024). Digital forensics and Windows Sandbox as anti-forensics tool. International Research Journal on Advanced Engineering Hub (IRJAEH), 2(2), 327–332.

Moser, A., Kruegel, C., & Kirda, E. (2007). Limits of static analysis for malware detection. In Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007). IEEE.

Microsoft. (2019). Windows Sandbox. Retrieved from https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-sandbox/windows-sandbox-overview

National Security Agency. (2019). Ghidra. Retrieved from https://ghidra-sre.org/

O’Kane, P., Sezer, S., & McLaughlin, K. (2011). Obfuscation: The hidden malware. IEEE Security & Privacy, 9(5), 41–47.

Olaimat, M. N., Maarof, M. A., & Al-rimy, B. A. S. (2021). Ransomware anti-analysis and evasion techniques: A survey and research directions. In 2021 3rd International Cyber Resilience Conference (CRC). IEEE.

Or-Meir, O., Nissim, N., Elovici, Y., & Rokach, L. (2019). Dynamic malware analysis in the modern era—A state of the art survey. ACM Computing Surveys, 52(5), 1–48.

Rana, N., Kumar, N., Handa, A., & Shukla, S. K. (2022). Automated Windows behavioral tracing for malware analysis. Security and Privacy, 5(6).

Roccia, T. (2019). Evolution of malware sandbox evasion tactics – A retrospective study. McAfee Labs.

Sabanal, C. (2014). Radare2: An open source reverse engineering framework. In Black Hat USA.

Saxe, J., & Berlin, K. (2015). Deep neural network based malware detection using two dimensional binary program features [Preprint]. arXiv.

Shijo, P. V., & Salim, A. (2015). Integrated static and dynamic analysis for malware detection. Procedia Computer Science, 46, 804–811.

Sihwail, R., Omar, K., & Ariffin, K. A. Z. (2018). A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. International Journal on Advanced Science, Engineering and Information Technology, 8(4–2), 1662–1671.

Sikorski, M., & Honig, A. (2012). Practical malware analysis: The hands-on guide to dissecting malicious software. No Starch Press.

SANS Institute. (2023). Malware Landscape 2023.

Hatching Triage. (n.d.). Malware analysis sandbox. Retrieved August 6, 2025, from https://tria.ge

Vasilescu, M., Gheorghe, L., & Tapus, N. (2014). Practical malware analysis based on sandboxing. In 2014 RoEduNet Conference 13th Edition: Networking in Education and Research Joint Event RENAM 8th Conference. IEEE.

Veerappan, C. S., Keong, P. L. K., Tang, Z., & Tan, F. (2018). Taxonomy on malware evasion countermeasures techniques. In 2018 IEEE 4th World Forum on Internet of Things (WF-IoT). IEEE.

Wang, T.-Y., Wu, C.-H., & Hsieh, C.-C. (2009). Detecting unknown malicious executables using portable executable headers. In 2009 Fifth International Joint Conference on INC, IMS and IDC. IEEE.

Willems, C., Holz, T., & Freiling, F. (2007). Toward automated dynamic malware analysis using CWSandbox. IEEE Security & Privacy, 5(2), 32–39.

Wong, M. Y., et al. (2021). An inside look into the practice of malware analysis. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security. ACM.

You, I., & Yim, K. (2010). Malware obfuscation techniques: A brief survey. In 2010 International Conference on Broadband, Wireless Computing, Communication and Applications. IEEE.

النسخة المحسّنة من ويندوز ساندبوكس: إطار موحّد للتحليل الآلي المتخفّي للبرمجيات الخبيثة مع نظام تسجيل مركزي

محمد يوسف حسين صلاح

د. اسلام عمرو

د. احمد حساسنة

د. نائل أبوحلاوة

ملخص

تُشكّل البرمجيات الخبيثة المتطورة، لا سيما تلك التي تستخدم تقنيات التهرب المتقدمة، تحديًا مستمرًا وخطيرًا للأمن السيبراني الحديث. إذ تستطيع هذه البرمجيات اكتشاف بيئات التحليل الافتراضية أو "صناديق الرمل(sandboxes) "، مما يمكّنها من إخفاء سلوكها الحقيقي وإفشال عملية التحليل. وعلى الرغم من أن ميزة Windows Sandbox المدمجة في نظام التشغيل مايكروسوفت ويندوز توفّر بيئة افتراضية خفيفة الوزن وواعدة، إلا أن إعدادها الافتراضي قابل للاكتشاف بسهولة، كما يفتقر إلى أدوات الأتمتة والتسجيل العميق اللازمة للتحليل المتقدم للبرمجيات الخبيثة.

تتناول هذه الرسالة هذه النواقص من خلال تصميم وتنفيذ وتقييم نظام جديد يُسمى "محسّن الصندوق الرملي(Sandbox Enhancer) "، والذي يحوّل Windows Sandbox إلى منصة تحليل خفية وفعالة للبرمجيات الخبيثة. اعتمدت المنهجية على ثلاث ركائز أساسية: (1) تحديد منهجي لسمات الصندوق الافتراضي القابلة للاكتشاف والعمل على تعديلها لزيادة الخفاء، (2) تطوير سير عمل آلي بالكامل لإدارة دورة التحليل من البداية حتى مرحلة إعداد التقارير، و(3) دمج إطار تسجيل شامل باستخدام أدوات Sysmon و NxLogو Graylogلتوفير رؤية سلوكية متعمقة.

تم تقييم النظام من خلال تجربة واسعة النطاق شملت تحليل 1700 عينة برمجية خبيثة، جميعها مُعرفة مسبقًا بأنها "مراوغة" لقدرتها على تجاوز بيئة تحليل احترافية. وأظهرت النتائج فاعلية النظام بشكل لافت، حيث نجح في إجبار 47.8٪ من هذه العينات المتقدمة على تنفيذ حمولة خبيثة كشفت عن سلوكيات خطرة، مثل الاتصال بخوادم التحكم والسيطرة (C2) وتنفيذ عمليات فدية محلية.

يساهم هذا البحث في توفير إطار عمل عملي، مُثبت فعاليته، وقابل للتوسع لتحليل البرمجيات الخبيثة المراوغة. كما يقدم نموذجًا فعّالًا لتعزيز بيئات المحاكاة الخفيفة، ويعد أداة قوية للباحثين والممارسين في مجال الأمن السيبراني. وتؤكد النتائج أن تحسين مستوى الخفاء والأتمتة والمراقبة المنهجية في بيئات مثل Windows Sandbox يمكن أن يرفع من قدرتها بشكل كبير على مواجهة التهديدات السيبرانية المتزايدة.

الكلمات المفتاحية: تحليل البرمجيات الخبيثة، عمل مؤتمت، صندوق الامان لويندوز، الأمن السيبراني، استخبارات التهديدات.