

Analyzing Malware: A Comparative Study of Static and Dynamic Methods through Reverse Engineering

Fransiskus Krismayandri Ekaristi^{a,i}

^aInformation Master Study Program, Nusa Putra University, Sukabumi, Indonesia

ARTICLE INFO

Keywords:

Malware
Reverse Engineering
Static Analysis
Dynamic Analysis

ABSTRACT

Malware remains a critical and evolving threat to modern computing environments, necessitating advanced defensive strategies from cybersecurity professionals. This study explores malware analysis through both static and dynamic approaches within a Windows-based environment, emphasizing the role of reverse engineering. Static analysis involves inspecting malware binaries, structure, and metadata without execution. Tools such as IDA Pro and CFF Explorer allow analysts to uncover embedded exploits, operational logic, and potential payloads, offering deep structural insight into the malware's functionality. In contrast, dynamic analysis focuses on observing malware behaviour in real time by executing it within an isolated virtual machine. This method, supported by tools like Process Monitor (ProcMon), enables analysts to track runtime activities, such as API calls, file system alterations, registry modifications, and network interactions. The integration of these tools provides a comprehensive view of malware behaviour, both at the static and operational levels. The paper also highlights the growing sophistication of malware, which increasingly employs obfuscation and evasion techniques. Thus, combining static and dynamic analyses—augmented by antivirus tools and SIEM platforms like Wazuh—enhances detection capabilities. The study concludes that a hybrid reverse engineering strategy significantly strengthens malware defence, offering actionable recommendations to reduce future threats within enterprise-level Windows environments.

1. Introduction

1.1. Overview

In the context of increasingly interconnected digital infrastructures, malware has evolved into a pervasive and adaptive threat impacting private users, enterprises, and governmental systems alike. Malware, a contraction of “malicious software,” denotes a broad spectrum of software artifacts engineered to infiltrate, manipulate, or exploit computing environments for unauthorized purposes. The proliferation of polymorphic and metamorphic variants—capable of dynamically altering their code signatures—has rendered traditional signature-based detection approaches increasingly ineffective. This paradigm shift necessitates a more rigorous analytical framework, particularly grounded in reverse engineering techniques, to expose the operational logic, concealment strategies, and behavioral characteristics of modern malware.

ⁱCorresponding author.

 fransiskus.krismayandri@nusaputra.ac.id (F.K. Ekaristi)
ORCID(s):

Static analysis, a fundamental aspect of reverse engineering, allows for the deconstruction of malicious binaries without running them. Using disassemblers, decompilers, and string analysis tools, practitioners may examine control flows, API calls, and embedded payloads. However, this strategy is severely limited by advanced obfuscation methods, encrypted payloads, and packers designed to evade analytical inspection. Dynamic analysis supplements this by running the malware in isolated environments (such as sandboxes or virtual machines) and monitoring real-time behaviors such as file system changes, process injections, and network communications. Nonetheless, malware using anti-VM, anti-debugging, or delayed execution methods can detect and disrupt dynamic analysis environments, restricting visibility into their destructive algorithms.

Given these constraints, modern malware researchers are increasingly using hybrid methodologies that include static, dynamic, and behavioral analysis. Memory forensics has emerged as a critical discipline for detecting ephemeral or fileless malware that exists solely in volatile memory. Analysts can recover important

memory artifacts, such as disguised programs and injected DLLs, using forensic frameworks like Volatility and Rekall. Furthermore, the use of machine learning has substantially improved the capacity to identify anomalies and categorize emerging threats at scale, strengthening detection pipelines against ever-changing malware types.

In essence, malware is one of the most powerful and automated tools available to cyber adversaries, allowing them to execute persistent and large-scale attacks across digital ecosystems. The growing diversity of malware needs similarly complicated protection. Robust reverse engineering, memory forensics, and AI-powered threat intelligence are required to maintain analytical depth and provide proactive security in an increasingly hostile cyberspace.

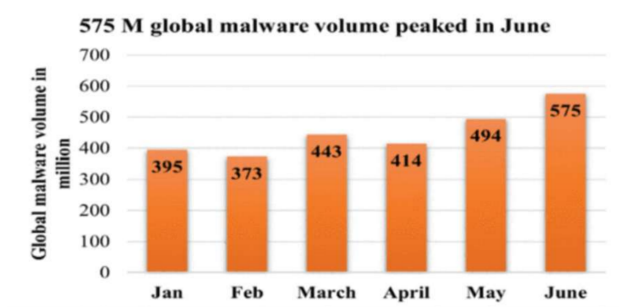


Figure 1: The increasing rate of global malware volume (2023).

Figure 1 presents a temporal distribution of global malware volumes, measured in millions, across the first half of 2023. Beginning at 395 million in January and experiencing a slight decline to 373 million in February, the data thereafter reveals a sustained and accelerating increase—rising to 443 million in March, 414 million in April, and 494 million in May. The apex is reached in June, with 575 million malware instances recorded, representing a 46% increase from the 395 million hits recorded in January [Ferdous et al. \(2023\)](#). This progressive amplification of malicious activity underscores the intensifying scale and velocity of global cyber threats, necessitating more robust, adaptive, and anticipatory cybersecurity frameworks.

2. Study Literature

2.1. The Importance of Malware Analysis Study

Malware poses a significant threat to computer systems, networks, and sensitive data. To combat these threats effectively, cybersecurity experts rely on malware analysis using reverse engineering techniques. This paper provides an in-depth exploration of malware analysis through reverse engineering with static and dynamic methods, detailing the tools and challenges involved. The goal is to empower cybersecurity professionals with

a comprehensive understanding of reverse engineering as an indispensable technique to analyse and neutralize malware effectively.

[Yusirwan et al.](#) investigated the combination of static and dynamic analysis, highlighting the analytical depth gained by hybridization [Yusirwan et al. \(2015a\)](#). However, they highlight a critical operational limitation—significant time consumption—which necessitates optimization solutions that maintain analytical integrity while increasing throughput. Innovations like the Kunai framework [Blázquez and Tapiador \(2023\)](#) demonstrate improvements in static analysis scalability for Android platforms, although issues in handling large datasets remain. Simultaneously, [Arif Islam and Mohan Kumar](#) urge for automation in dynamic analysis to reduce reaction time and unload routine investigative tasks from human analysts [Arif Islam and Mohan Kumar \(2023\)](#).

The outcomes of these studies suggest a paradigm change toward adaptive, multi-layered malware analysis systems. Emerging tools, such as AI-driven classification models [Chen et al. \(2023\)](#) and enhanced memory forensics for ransomware deconstruction [Alsharabi et al. \(2023\)](#), are critical in navigating the obfuscation and anti-analysis strategies used by modern threats. According to [Cletus et al.](#), the transition from traditional, signature-based detection to behaviour-centric and memory-resident analysis models will be critical in ensuring resilient and future-proof cybersecurity infrastructures that can withstand targeted, persistent, and stealth-based malware attacks [Cletus et al. \(2024\)](#).

2.2. Types of Malware

Several common categories of malware are frequently encountered in the wild:

- **Viruses:** One of the earliest and most well-known forms of malware.
- **Worms:** Malware that reproduces on its own and spreads throughout networks without human interaction.
- **Trojans:** Malicious programs or files that impersonate trustworthy ones in order to trick users.
- **Ransomware:** A dangerous kind of virus that encrypts a victim's files and renders them inaccessible until the attacker receives a ransom payment.
- **Spyware:** Designed to covertly gather data on a user's activities, including browsing patterns, login details, and sensitive information.
- **Adware:** A less dangerous but nonetheless intrusive form of malware that shows users unwelcome advertising.
- **Botnets:** Networks of compromised computers are known as botnets.

- **Polymorphic malware:** A type of virus that can change its code while retaining its core functionality.
- **Metamorphic malware:** A kind of malware that modifies its source code with each new infection.

2.3. Structure of Malware

Malware commonly exhibits the following structural behaviours:

- **Persistence:** A common tactic for malware to continue its operations is to ensure persistence. Malicious actors can exploit these features to install malware that automatically reactivates every time the system boots.
- **Obfuscation:** A technique used in malware development to conceal its malicious actions, thereby potentially evading security defences and complicating debugging efforts.
- **Defence evasion:** Techniques used by attackers to avoid detection, such as disabling security measures, manipulating authentication systems, and escalating privileges. Examples include disabling Defender and manipulating access tokens.
- **Dropper:** A script or program used to help the installation of malware. When a dropper is executed, it initially downloads and installs malware that carries out any additional programmed tasks.
- **C2 → Web crawlers:** These malware entities engage in the unauthorized retrieval of users' or enterprises' data through the transmission of malicious crawler data.

2.4. Malware Analysis Method

2.4.1. Static Analysis

Static analysis is a fundamental approach in malware research that involves inspecting source code, binaries, or deconstructed representations without execution. This approach uses control and data flow analysis, symbolic execution, and pattern matching to identify hidden vulnerabilities or malicious logic. Figure 2 illustrates advanced static analysis, which uses abstract interpretation, formal model checking, and IDA-based disassembly to extract granular insights into linked libraries, function calls, and encoded strings. While capable of identifying complex threats early in the development or analysis lifecycle, it has scaling limitations, especially in big or obfuscated codebases, and suffers with runtime-specific behaviors found in dynamically loaded or self-modifying malware.

2.4.2. Dynamic Analysis

Dynamic analysis, also known as Basic Dynamic Analysis, is the running of a program in a controlled environment to observe its real-time activity. This

method uses debugging, profiling, fuzzing, and taint analysis to monitor API calls, memory access patterns, registry changes, and control flows. Tools accomplish this by instrumenting the execution environment and collecting precise runtime data. While dynamic analysis provides greater accuracy for behavioral evaluation, its efficacy is strongly dependent on input coverage and contextual integrity. Advanced techniques, such as symbolic and concurrent execution, attempt systematic path exploration but are hampered by path explosion. Dynamic analysis, as seen in advanced workflows, involves malware debugging, registry scanning, and system-level evaluation in Windows settings.

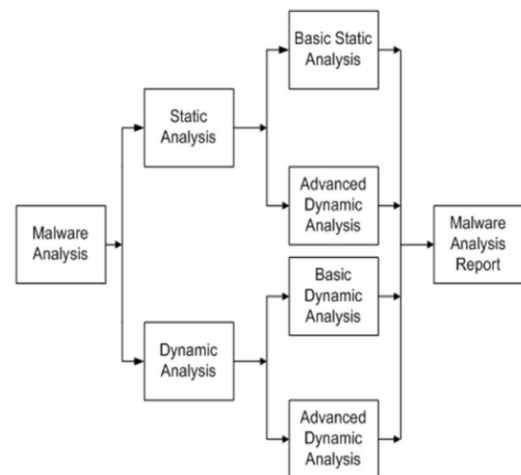


Figure 2: Analysis Malware (Static and Dynamic).

2.5. Reverse Engineering

Reverse engineering is a process of taking parts of software or hardware, analysing the function and the information obtained, and then translating that process into a language that is more easily understood by humans. The purpose of reverse engineering is usually to duplicate or improve the functionality of the original product. In malware analysis, the purpose of reverse engineering is to understand the workings of a malware [Eilam \(2003\)](#).

Static analysis environments use tools like IDA Pro and CFF Explorer to disassemble executable programs. CFF Explorer allows for extensive inspection of Portable Executable (PE) file structures and analysis without affecting internal settings [Manohar Venkat et al. \(2023\)](#); [Megira et al. \(2018\)](#). IDA Pro breaks down binaries into assembly code, providing detailed insights into program behavior. Although decompilers may not be fully architecture compatible, they can help translate binaries into native code for further analysis [Islam et al. \(2023\)](#). Key static indicators, such as header fields, section entropy, printed characters, and function calls, may be retrieved to identify malware sources and build attacker signatures [Maniriho et al. \(2024\)](#); [Sihwail et al. \(2018\)](#).

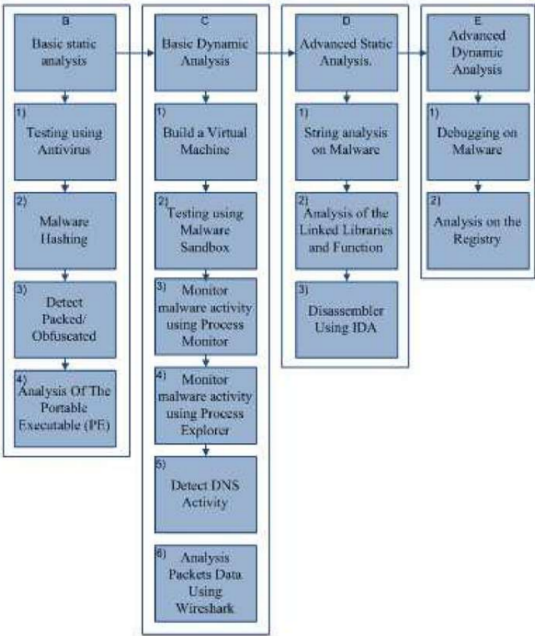


Figure 3: Detail of Analysis Malware Yusirwan et al. (2015b).

The Figure 3 depicts a complete malware analysis process that includes fundamental and advanced static and dynamic techniques. It includes utilities like antivirus scanners, IDA, Process Monitor, and Wireshark. This systematic technique enables a thorough knowledge of malware behaviour, such as registry interactions, PE file analysis, network activities, and function disassembly in virtual environments.

3. Research Methodology

3.1. Static Method

The malware analysis process started with static analysis after selecting the appropriate tools. Static analysis tools were used to collect information from the PE header of the .exe file. It will be possible to learn whether the file actually includes malware, what kind of malware it is, and what programming language it uses.

3.2. Dynamic Analysis

Dynamic analysis, also known as behaviour analysis, runs suspect files in a supervised setting such as a virtual machine, emulator, or simulator. The simple reason that the infected files must be examined in a covert environment is that some malware is supported by anti-virtual machine and anti-emulator capabilities.

Figure 4 shows the flow of research environment analysis. It depicts a malware analysis pipeline that includes preparation, basic and dynamic analysis in a virtual environment utilizing tools such as CFF Explorer, IDA Pro, and Procmon. Integration with SIEM (such as Wazuh) and antivirus programs improves detection. The procedure concludes with a report outlining malware activity, trends, and mitigating techniques.

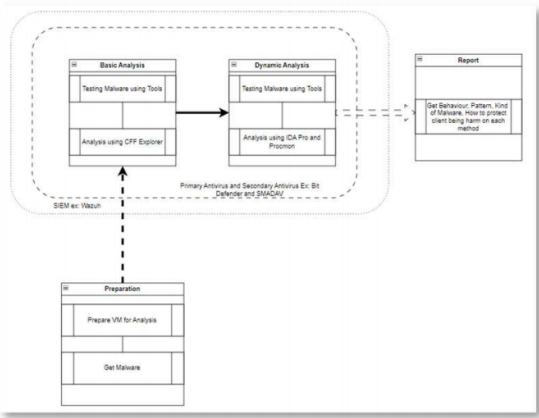


Figure 4: Flow Research Environment Analysis.

3.3. SWOT Methodology

Table 1 delineates the strategic implementation of a SWOT (Strengths, Weaknesses, Opportunities, and Threats) analytical framework applied to a suite of cybersecurity tools utilized in static and dynamic malware analysis, as of February 2024. The tools under review—CFF Explorer, IDA Pro, Process Monitor, along with primary and secondary antivirus solutions, and SIEM platforms such as Wazuh—are critically examined to evaluate their operational capabilities, contextual limitations, integration potential, and exposure to evolving threat vectors.

Table 1: SWOT Methodology.

Reference SWOT	Subject SWOT	Date
Static, Dynamic Analysis	CFF Explorer, IDA Pro, Procmon, Primary Antivirus and Secondary Antivirus, Security Tools (SIEM) Ex: Wazuh	February 2024

4. Discussion

4.1. Pre-Processing Data

Figure 5 reveals that the malware is capable of executing solely within the Windows Console environment, while the subsequent illustration confirms its inability to operate under a DOS-based system. This behavioural constraint strongly suggests that the malware is architected with platform-specific dependencies, thereby restricting its operational scope to the Windows operating system. Such specificity implies a deliberate design to exploit Windows-native functionalities, rendering the malware ineffective in non-Windows environments. Accordingly, any resultant system compromise—ranging from service disruption to full-scale cyberattacks—can be attributed to vulnerabilities unique to the Windows ecosystem, highlighting a focused threat vector within that particular platform domain.

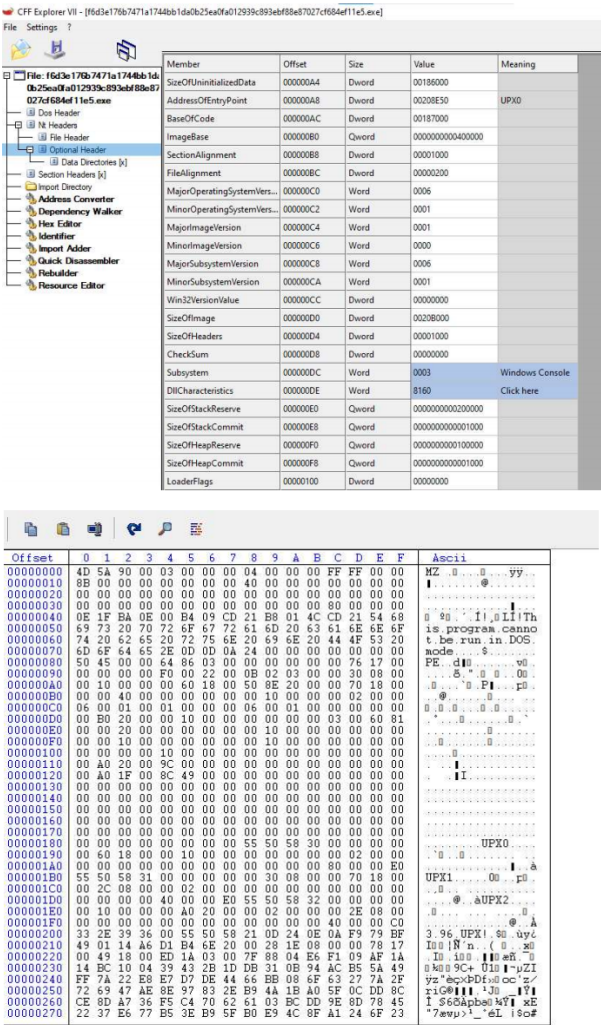


Figure 5: Reverse Engineering on Static Method Using CFF Explorer.

4.2. Reverse Engineering on Static Method (CFF Explorer & IDA Pro)

As depicted in Figure 6, several code segments are observable, providing insight into the malware’s operational behaviour. These include file permission settings (read/write/execute), confirmation of a 64-bit Windows execution environment, and a binary view of the malware. Such details facilitate a deeper understanding of the malware’s structural and functional characteristics.

Platform-specific analysis showed that some malware variants were designed for 32-bit Windows systems and functioned only on specific operating system versions, particularly from Windows Vista to Windows 10. Nonetheless, static analysis proved insufficient in exposing runtime behaviour, network communication, or memory-level modifications, thereby highlighting the necessity of complementary dynamic analysis approaches. Several samples exhibited embedded DLL dependencies, indicating modular design, while others employed packing or encryption mechanisms to evade detection as shown in Table 2.

Table 2: Static Method Evaluation.

Observation	Details
PE Structure & Execution Type	Malware samples were found to be Windows GUI-based executables and some were not executable in DOS environments.
Malware Functionality	IDA Pro revealed the internal flow and logic of malware execution, including function calls, API hooking, and obfuscation techniques.
Embedded Components	Some malware samples contained DLL dependencies, indicating modularity in execution. Others showed packer or encryption mechanisms to evade detection.
Platform-Specific Execution	Certain malware samples were built for 32-bit Windows environments and were compatible only with specific OS versions (Vista to Windows 10).
Static Detection Limitations	While static analysis provided insight into malware internals, it could not reveal runtime behaviour, network activity, or memory modifications.

4.3. Reverse Engineering on Dynamic Method (Procmon & Wazuh)

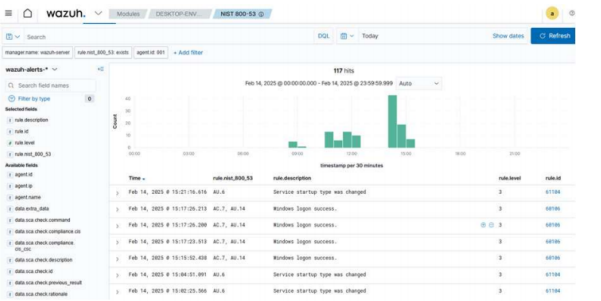


Figure 8: Wazuh (SIEM) Monitoring for Threat Activity.

Table 3: Dynamic Method Evaluation.

Observation	Details
System & Process Behaviour	Procmon captured process creation, registry modifications, and suspicious file access, but lacked full transparency into malware execution.
Application Termination	Some malware variants forcefully terminated monitoring tools, preventing real-time behaviour analysis.
Network Activity	Certain malware samples exhibited UDP communication anomalies, suggesting potential data exfiltration or command-and-control (C2) activity.
System Modification	Some malware samples attempted to alter login security policies or make persistent changes to the OS.
Procmon’s Limitation	Dynamic monitoring provided insights into runtime execution, but due to anti-debugging and sandbox evasion techniques, certain activities remained hidden.
Wazuh’s Limitation	Wazuh failed to detect specific malware activities because security policies automatically blocked execution, leading to incomplete forensic visibility.

Figure 7 illustrates that, upon execution, the malware displays no overtly complex or distinctive behavioural patterns. Instead, it initiates a basic operation by attempting to access several dynamic-link library (DLL) files via the explorer.exe process, indicating

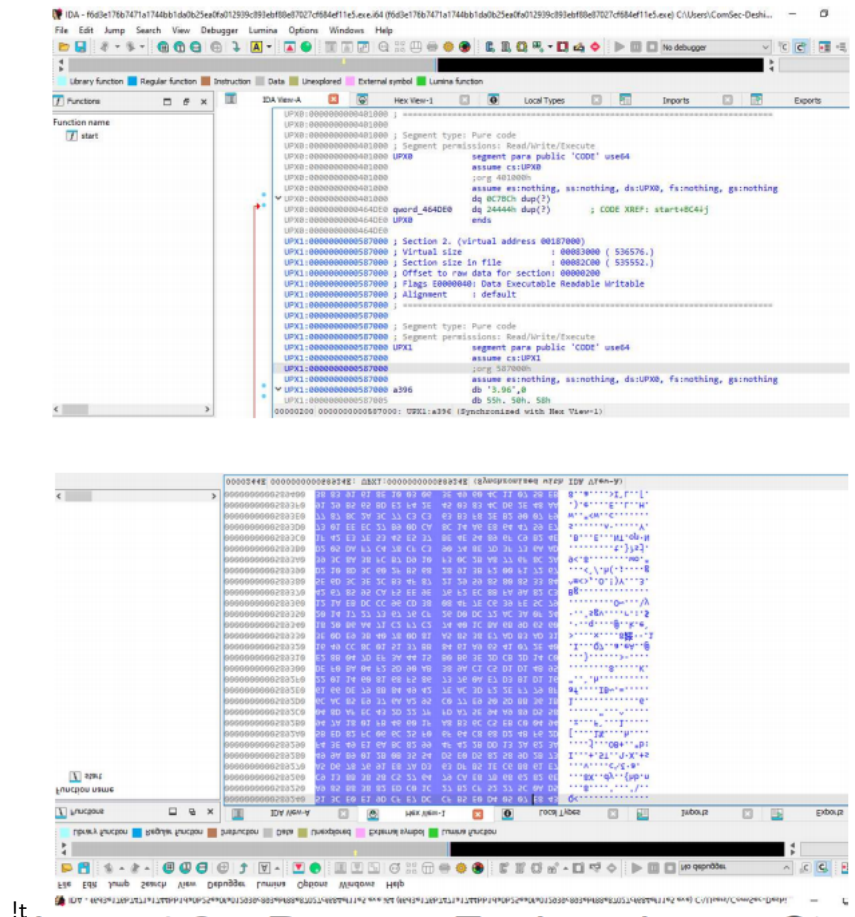


Figure 6: Reverse Engineering on Static Method Using IDA.

limited observable activity during the initial execution phase.

Figure 8 presents a behavioural analysis utilizing the Wazuh security monitoring platform within a controlled research environment, aiming to assess whether the malware’s impact was confined to the local host or exhibited propagation across the network. However, the findings indicate that Wazuh did not register any notable indicators of malicious activity during the execution period, suggesting either limited behavioural manifestation or evasion of detection mechanism.

The dynamic analysis of malware samples on the environment analysis revealed several critical insights and limitations. Procmon successfully captured key system activities, including process creation, registry modifications, and suspicious file access; however, its visibility was hindered by anti-debugging and sandbox evasion techniques. Some malware variants deliberately terminated monitoring tools, thereby obstructing real-time behavioural analysis. Furthermore, certain malware samples attempted to modify login security policies and introduce persistent changes to the operating system. Despite its capabilities, Wazuh was unable to detect specific threats, as automated security policies

preemptively blocked execution, leading to incomplete forensic visibility. These findings highlight the inherent limitations of static and dynamic analysis tools when confronting sophisticated and evasive malware as shown in Table 3.

4.4. Antivirus (Primary & Secondary AV)

As illustrated in Figure 10, the detection of Trojan:Win32/Wacatac.H!ml, categorized as a severe threat by Windows Defender, indicates a significant breach of system security. This malware strain is typically linked to persistent infection techniques and potential command-and-control functionalities, necessitating immediate containment, in-depth forensic analysis, and prompt remediation to preserve system integrity and mitigate further compromise.



Figure 9: Antivirus Secondary (Smadav) not capture detail type of Malware.

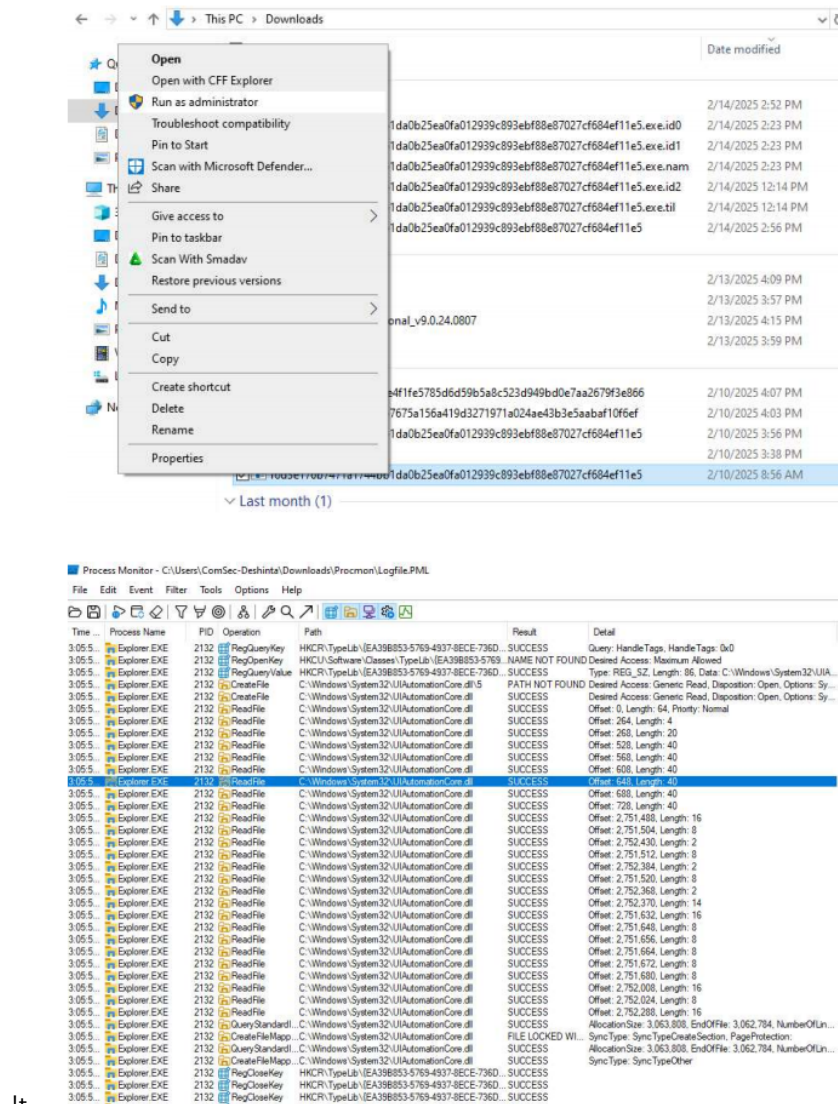


Figure 7: Reverse Engineering on Dynamic Method Using ProcMon.

However, as depicted in Figure 9, the Smadav antivirus failed to identify this threat, highlighting a potential limitation in its detection capabilities.

The evaluation of antivirus effectiveness revealed notable capabilities and inherent limitations in malware detection. Signature-based detection accurately identified known threats such as Trojan:MSIL, AsyncRAT, and Blackmoon. Complementing this, heuristic analysis detected anomalous files lacking definitive malware signatures. Nonetheless, certain detections—such as Program:Win32/Wacapew.C!ML—were later classified as false positives. Real-time blocking mechanisms, while protective, occasionally hindered comprehensive dynamic analysis using tools like Procmon. In other instances, malware evaded initial detection entirely, despite later evidence of system compromise and application instability.

Virus & threat protection

Protection for your device against threats.

Current threats

Threats found. Start the recommended actions.

Trojan:Win32/Wacatac.H!ml
2/14/2025 2:53 PM (Active)

Severe

Figure 10: Antivirus Primary (Windows Defender) capture detail type of Malware.

Moreover, several samples exhibited Trojan-like functionalities, including credential theft, unauthorized system alterations, and remote access capabilities, underscoring the need for multi-layered detection strategies as summarized in Table 4.

Table 4: Antivirus Evaluation.

Observation	Details
Signature-Based Detection	Primary antivirus successfully detected known threats like Trojan:MSIL/AsyncRAT, Win32/Blackmoon, Win32/GulLoader.RSM!MTB, and others.
Heuristic Analysis	Secondary antivirus provided additional detections, identifying threats that were initially flagged as suspicious but lacked clear malware signatures.
False Positives	Some detections, such as Program:Win32/Wacapew.C!ML, were flagged as malware but were later found to be potentially false positives.
Real-Time Blocking	Antivirus blocked some samples before execution, preventing deeper dynamic analysis with tools like Procmon.
Missed Threats	In certain cases, antivirus scans reported no detected threats, yet later analysis showed system compromise and application crashes.
Trojan Functionality	Some malware samples were linked to data theft (password harvesting), system modifications, and remote access functionalities (e.g., Emotet, AsyncRAT, Remcos).

5. Conclusion and Recommendation

Based on data pre-processing conducted using three scenarios of reverse engineering on static and dynamic methods using CFF Explorer, IDA, Process Monitor (Procmon), Wazuh, and antivirus software, this study provides a multidimensional perspective for more thorough threat investigation. Contemporary advancements in malware analysis offer significant opportunities to strengthen cybersecurity frameworks.

The integration of automated analysis pipelines—utilizing machine learning [Gulmez et al. \(2024\)](#) and sandboxing technologies—facilitates the efficient classification and behavioral tracing of malicious software. Enhancements in detection algorithms, particularly those capable of identifying polymorphic and metamorphic malware, contribute to more adaptive and resilient security systems. Furthermore, the application of memory forensics, using tools like Volatility, enables the identification of in-memory threats and fileless malware, while network threat monitoring assists in the detection of command-and-control (C2) communication and potential data exfiltration activities. Moreover, recent innovations in reverse engineering—particularly those incorporating artificial intelligence for disassembly and deobfuscation—enhance the accuracy of static malware analysis. Lastly, the promotion of enhanced threat intelligence sharing through community-driven threat feeds facilitates proactive defense against evolving malware variants. Collectively, these opportunities underscore the necessity of a layered and integrative

approach to modern malware defense within formal cybersecurity education and practice.

The analysis of various malware samples across different virtual machines highlights modern malware detection and analysis methodologies. Static analysis using IDA and CFF Explorer provided detailed insights into the structure and functionality of the malware, revealing execution constraints and dependency requirements. However, dynamic analysis with Process Monitor (Procmon) exposed runtime behaviours, though it was limited in observing stealthy malware that terminated security processes upon execution. Primary and secondary antivirus solutions, while effective in detecting known threats, struggled against polymorphic and metamorphic malware due to their reliance on signature-based detection [Cletus et al. \(2024\)](#). Wazuh, as a security monitoring tool, demonstrated capability in logging and detecting system modifications but lacked granular real-time behavioural detection, particularly against rapidly executed malware.

To address these limitations, a multi-layered approach is essential. Behavioural heuristics, memory forensics, and machine-learning-driven anomaly detection should complement traditional static and dynamic methods. Threat intelligence integration (e.g., MITRE ATT&CK, VirusTotal, and MISP) can enhance proactive defence. Signature-based AVs fail to detect evasive malware, while AI/ML-based heuristics improve detection accuracy [Gulmez et al. \(2024\)](#). Sandboxing solutions like Cuckoo can provide safer analysis environments. Ultimately, a combination of advanced EDR solutions, proactive threat hunting, and Zero Trust security policies is required to counter increasingly sophisticated malware threats [Verizon \(2023\)](#). By adopting these strategies, organizations can significantly enhance their cyber resilience and malware detection effectiveness, which could reduce malware attacks in the coming years.

Acknowledgments

The author would like to thank the Information Master Study Program at Nusa Putra University for the support and resources provided during this research.

References

- Alsharabi, N. et al. (2023). Analysis of ransomware using reverse engineering techniques to develop effective countermeasures. *Journal of Advances in Information Technology*, 14(2):284–294.
- Arif Islam, M. and Mohan Kumar, P. (2023). Cyber safety analysis using reverse engineering. *International Journal of Research Publication and Reviews*, 4(1):399–403.
- Blázquez, R. T. and Tapiador, J. E. (2023). Kunai: A static analysis framework for Android apps. *SoftwareX*, 22:101370.
- Chen, L. et al. (2023). Efficient Windows malware identification and classification scheme for plant protection information systems. *Frontiers in Plant Science*, 14.
- Cletus, A. et al. (2024). An evaluation of current malware trends and defense techniques: A scoping review with empirical

- case studies. *Journal of Advances in Information Technology*, 15(5):649–671.
- Eilam, E. (2003). *Reversing: Secrets of Reverse Engineering*. Wiley Publishing, Inc., Indianapolis.
- Ferdous, S. et al. (2023). A review of state-of-the-art malware attack trends and defence mechanisms. *IEEE Access*, 11:121118–121141.
- Gulmez, A. et al. (2024). Analysis of the zero-day detection of metamorphic malware. In *Proceedings of the 2024 9th International Conference on Computer Science and Engineering (UBMK)*, pages 1–6, Antalya, Turkiye.
- Islam, A. et al. (2023). Static malware analysis using reverse engineering: Tools and techniques.
- Maniriho, P. et al. (2024). Static-analysis techniques of malware reverse engineering.
- Manohar Venkat, S. et al. (2023). Malware reverse engineering to find the malicious activity of Emotet. In *Advances in Transdisciplinary Engineering*.
- Megira, K. et al. (2018). Malware analysis and detection using reverse engineering technique. *Journal of Physics: Conference Series*.
- Sihwail, R. et al. (2018). A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. *International Journal of Advanced Science and Engineering Information Technology*, 8(4-2):1662–1671.
- Verizon (2023). 2023 data breach investigations report: Phishing trends. Technical report, Verizon.
- Yusirwan et al. (2015a). Combination of static and dynamic malware analysis: An analytical depth perspective. *International Journal of Computer Applications*. Cited as Yusirwan et al., 2015b.
- Yusirwan et al. (2015b). Implementation of malware analysis using static and dynamic analysis method. *International Journal of Computer Applications*, 117(6):11–15.