An Overview of Malware Analysis

Stamatia Kalimeri, Niki-Aikaterini Kyriakatou, Nicolas Sklavos
SCYTALE Group
Computer Engineering and Informatics Dept. (CEID), University of Patras
Patra, Greece

Abstract

In cybersecurity, malware poses a significant threat to computer systems and digital infrastructures. Malware infections often alter normal system behavior, resulting in a range of harmful consequences, including data loss, data corruption, and unauthorized privilege escalation, which may further enable infected systems to participate in broader malicious activities, such as DDoS attacks. As IoT continues to expand, malware can easily target connected and embedded devices, creating new security challenges. In today's digital landscape, the need to detect, understand, and prevent malware through the study of its behavior has become more critical than ever. In this paper, two approaches to malware analysis are explored, namely static analysis and dynamic analysis, followed by a comparative evaluation of their effectiveness, strengths, and limitations.

CCS Concepts

• **Malware Analysis**; • **Create isolated virtual environment**; • **Reverse Engineering** → Decompilers; • **Dynamic Analysis** → Determine malware behavior;

Keywords

Malware, Static Analysis, Dynamic Analysis, Function Monitoring, Information Flow

1 Introduction

Computer systems are frequently targeted by malware. Common malicious behaviors include code injection, unauthorized privilege escalation that may lead to backdoor creation, and the encryption or corruption of sensitive data stored on storage devices [1]. Certain forms of malware are designed to mimic the behavior of biological viruses. For instance, self-propagating malware such as worms can spread from an infected system to other devices through network connectivity. Such infections may compromise systems ranging from IoT devices to large-scale infrastructures, causing critical impact across multiple sectors.

The threat posed by malware can be mitigated by effective detection and analysis techniques. To prevent malware infections, understanding the main categories of malware and their respective life cycles is vital. This paper first examines the fundamentals of static analysis, where malicious software is investigated without execution using methods such as decompilation and reverse engineering [2]. Then, dynamic analysis is explored, in which malware is executed within a controlled environment to observe its behavior and determine its functionality [3]. Finally, the paper concludes with a comparative evaluation of static and dynamic analysis in terms of their operational effectiveness, strengths, and constraints.

2 Preliminaries

Malware is software designed to disrupt operations, compromise sensitive data, or gain unauthorized access to computer systems. Malware analysis commonly employs both static and dynamic techniques. In static analysis, information is extracted without code execution, whereas in dynamic analysis, malware behavior is observed during execution within controlled environments. These techniques are commonly performed using virtual machines, which enable the isolated execution of operating systems to improve safety and containment. Network isolation is often reinforced through NAT configurations, allowing virtual machines to operate on private networks without direct external exposure. In addition, sandbox environments provide restricted execution spaces, while emulators simulate software or hardware components for analysis purposes.

Based on [4, 5], the different malware categories and their respective operational phases are summarized in Tables 1, Table 2 and Figure 1.

Type	Core Functionality
Virus	Attaches to programs and propagates upon execution.
Trojan	Hidden malicious logic in benign software.
Worm	Autonomous replication across networks.
Backdoor	Unauthorized remote system access.
Botnet	Remotely managed network of bots.
Ransomware	Data encryption for ransom demands.
Spyware	Covert data harvesting without consent.

Table 1: Malware Types & Core Functionalities

Phase	Description
Dormant	Malware remains inactive after installation.
Propagation	Spreads by attaching itself to files.
Triggering	Activation based on specific system conditions.
Execution	Final stage where malicious payload is deployed.

Table 2: Malware Life-cycle Phases.

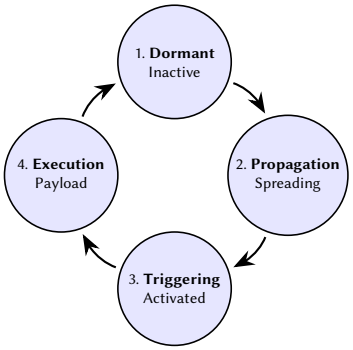

Figure 1: Malware Life Cycle.

3 Evaluating Malware Analysis Methods

3.1 Static Malware Analysis

Prior to sample examination, establishing a secure virtual environment is essential to ensure safe sample handling during static malware analysis. This approach helps prevent malware from spreading from the host system to other devices within the network. Then, an operating system image, such as Microsoft Windows or Kali Linux, is installed within a virtual machine, followed by the configuration of an isolated network. Network settings may vary across virtualization platforms. To maintain containment, the virtual system should not have direct access to external networks. In many virtual machine managers, the network mode can be switched between NAT and isolated configurations. These settings support the following stages of static malware analysis.

The next stage of the analysis involves the collection of malicious code samples from online repositories. An important step of this procedure is the verification of the sample's executable format, as many samples are designed for a specific operating system or processor architectures. This can be determined by identifying formats such as .exe or .elf. After acquisition, the sample can be further identified using online analysis services or antivirus tools. Although optional, this step can make subsequent reverse engineering more targeted and efficient.

Reverse engineering can be conducted using a variety of tools such as decompilers [6]. These tools can extract useful information about the malware sample. For example, tools such as PeiD for Microsoft Windows analyze PE headers to determine whether an executable is packed and to retrieve metadata such as the compilation timestamp, which may be examined for inconsistencies. Some Linux distributions, such as Kali Linux, include pre-installed reverse engineering tools such as Radare2 [7]. When malware employs packing techniques to avoid detection, packer identification tools become necessary for accurate analysis. For a deeper inspection, additional components may also be extracted from the executable. Most decompilation tools enable analysts to enumerate the functions contained within a program. In malicious code, identifiers, exported names, or remaining symbols may be intentionally crafted to appear benign, thereby attempting to evade detection and obstruct analysis.

After identifying suspicious functions, malware analysts can gain initial insight into the behavior of the examined code by analyzing the generated assembly or decompiled C/C++ code. However, the reconstructed code does not always exhibit a one-to-one correspondence with the original code. As a result, analysts and cybersecurity experts focus on understanding the core functionality of the program rather than its exact structure. Additional indicators, such as embedded strings and imported functions or API calls, can provide further valuable analytical insights.

Overall, these indicators not only help analysts understand malware behavior but also contribute to the development of hardware and software systems capable of detecting malware in real time. The steps of static malware analysis are illustrated in Figure 2.

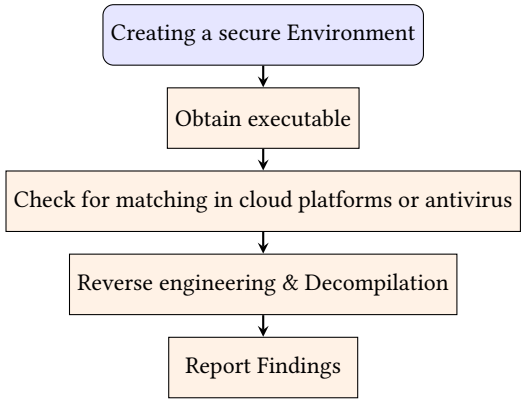

Figure 2: Static Malware Analysis Key Steps.

3.2 Dynamic Malware Analysis

After collecting initial data through static analysis, experts proceed to dynamic analysis. During this phase, the malware sample is executed within a controlled and isolated environment. Various implementation approaches to dynamic analysis have been proposed in the literature [8, 9].

Dynamic analysis can be conducted in either user or kernel space. In user space, it focuses on monitoring function and API calls, providing high-level insight into program behavior and memory activity. In kernel space, system calls and lower-level interactions are observed, offering deeper visibility into malware interaction with the operating system kernel. In addition to these approaches, automated analysis techniques include the use of emulators and sandboxes. Depending on the aspect under investigation, different types of emulation can be deployed, such as memory, CPU, or full-system emulation. Sandboxes provide an isolated environment in which malware can be executed without infecting other systems. However, advanced malware may detect sandbox environments and conceal its behavior. Dynamic analysis can also be conducted within virtual machines using specialized analysis tools.

The next stage of the analysis involves executing the malware sample. Depending on the context, execution may involve running an executable, opening a malicious document, or even visiting a compromised website. During the execution of malware, its behavior can be observed. This is typically achieved by monitoring function calls and their associated parameters. Analysts track these function calls to better understand the underlying functionality of the malware. Such analysis can be performed using debugging tools with breakpoints, as well as specialized frameworks. For instance, TTAnalyze employs API hooking techniques to monitor function activity, while CWSandbox utilizes code injection methods for behavioral analysis [10]. Furthermore, analyzing function parameters, such as the values during execution, provides additional insight into malware detection. Monitoring these parameters enables analysts to correlate function activity and gain deeper understanding of the program's operational logic.

Information flow tracking is another technique used during dynamic analysis. This method allows analysts to track the information flow within a computer system while malware is executed. It can be implemented using various approaches, one of the most common being data tainting, in which suspicious data is labeled to trace its propagation through memory and program execution. When the tainted data reaches a critical operation or sink, analysts

can trace its path backward to identify its origin and better understand the data flow. Several tools, such as Vigilante and TQana, support this type of analysis. For more in-depth investigation, both data and control flow dependencies may be examined.

The final stage of dynamic analysis methodology incorporates tracing. Various forms of tracing can be employed, including instruction tracing, which allows analysts to monitor the sequence of instructions executed during malware analysis. Providing malware with unrestricted network access may pose a risk to external systems and is therefore generally avoided. Instead, controlled network simulations can be used to safely analyze malware communication behavior. Within such environments, analysts can perform network traffic analysis to observe how malware interacts with remote systems. Tools such as Wireshark facilitate this process by capturing and logging network activity, including IP addresses. In Figure 3, an overview of dynamic analysis key steps is presented for better understanding of the procedure.

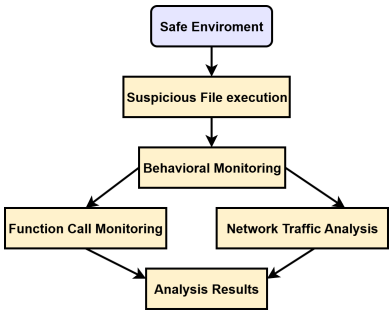

Figure 3: Dynamic Analysis Overview

3.3 Comparison of Static and Dynamic Malware Analysis

Static analysis is generally a faster and simpler approach, as it allows identification of potentially malicious code without executing it. It relies on pattern matching and string analysis techniques derived from reverse engineering methods. However, its effectiveness may be reduced when analyzing packed or previously undetected malware. Several studies have demonstrated that the integration of machine learning techniques can significantly enhance malware detection accuracy [11–13]. Common approaches for malware classification include Convolutional Neural Networks, Recurrent Neural Networks, and Large Language Models.

In contrast, dynamic analysis offers deeper insight into malware functionality. Although more time-consuming, it reveals runtime behavior, including data flow within the system and network communication patterns. However, advanced malware variants, such as context-aware malware, may attempt to evade detection by altering or concealing their behavior when executed in virtualized or monitored environments. Similar to static analysis, the application of machine learning techniques can further improve the accuracy of dynamic analysis.

4 Conclusions

Malware analysis plays a vital role in understanding the functionality of malware and mitigating its impact on computer systems. Static analysis provides a faster, lower-risk approach for examining malicious code without execution, whereas dynamic analysis offers deeper insight into runtime behavior, including memory activity and system-level interactions. Although each method presents distinct advantages and limitations, their combined use provides a more comprehensive understanding of malware behavior, making them complementary rather than competing approaches.

Furthermore, information extracted through malware analysis can support the development of advanced detection mechanisms, including machine learning models and hardware-based security for embedded environments. These capabilities can enhance automated threat identification, incident response, and real-time protection across modern computing systems. As malware continues to evolve in complexity and sophistication, the combined use of static and dynamic analysis remains essential for effective detection, prevention, and long-term cybersecurity resilience.

References

[1] K. Koutsikou J. Milosevic, N. Sklavos. “malware in iot software and hardware”, workshop on trustworthy manufacturing and utilization of secure devices (trudevice’16),barcelona, spain, november 14-16.

[2] Saurabh. Advance Malware Analysis Using Static and Dynamic Methodology. In *2018 International Conference on Advanced Computation and Telecommunication (ICACAT)*, pages 1–5, Bhopal, India, 2018. IEEE. Affiliation: Lakshmi Narain College of Technology & Science. Email: sudosaurabh@gmail.com.

[3] V. Anandhi, P. Vinod, Varun G. Menon, Abhijith Krishna E. R., Akshay Shilesh, Akshay Viswam, and Amin Shafiq. Malware detection using dynamic analysis. *Journal of Computer Science and Engineering*, 2022.

[4] Priya Arora, Rashmi Gupta, Nidhi Malik, and Anil Kumar. Malware Analysis Types & Techniques: A Survey. *International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)*, 9(4):25–30, 2022. Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009.

[5] Harjeevan Singh Gill. Malware: Types, Analysis and Classifications. 2022. Available at ResearchGate and engrXiv.

[6] Andrej Fedák and Jozef Štulrajter. Fundamentals of Static Malware Analysis: Principles, Methods, and Tools. In *Proceedings of the Academy of Armed Forces gen. M. R. Štefánik*, Liptovský Mikuláš, Slovak Republic, 2023. Department of Computer Science. Emails: andrejfedak@gmail.com, jozef.stulrajter@aos.sk.

[7] José Javier de Vicente Mohino, Javier Bermejo Higuera, Juan Ramón Bermejo Higuera, Juan Antonio Sicilia Montalvo, Manuel Sánchez Rubio, and José Javier Martínez Herraiz. MMALE—A Methodology for Malware Analysis in Linux Environments. *Computers, Materials & Continua*, 67(2):1447–1469, 2021. Affiliations: Universidad Internacional de La Rioja, Universidad de Alcalá de Henares.

[8] Ioannis Dermentzis, Vassilis Tsaoussidis, and Marios Anagnostopoulos. Implementation of a dynamic malware analysis environment using elk-stack. In *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 491–496. IEEE, 2021.

[9] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput. Surv.*, 44(2), feb 2012.

[10] Ori Or-Meir, Nir Nissim, Yuval Elovici, and Lior Rokach. Dynamic malware analysis in the modern era: A state of the art survey. *ACM Comput. Surv.*, 52(5), sep 2019.

[11] Matthew G. Gaber, Mohiuddin Ahmed, and Helge Janicke. Malware detection with artificial intelligence: A systematic literature review. *ACM Comput. Surv.*, 56(6), jan 2024.

[12] Yibiao Wu, Honglin Zhuang, Yetao Jia, and Yuting Zhang. A survey of machine learning approaches for malware detection. In *Proceedings of the 2025 5th International Conference on Computer Network Security and Software Engineering (CNSSE 2025)*, CNSSE 2025, pages 1–5, New York, NY, USA, 2025. Association for Computing Machinery.

[13] Farhath Zareen, Ahmed Ghoneim, Mateus A. Fernandes A., Samir Ahmed, and Robert Karam. Malware detection in embedded devices using artificial hardware immunity. *Journal of Hardware and Systems Security*, aug 2025. Published online: 13 August 2025.