From Fundamentals to Forensics: Exploring the Spectrum of Malware Analysis

Venkata Pranay Kumar Yerikala[1]
{ypranaykumar1002@gmail.com[1]}

Department of Advanced Computer Science and Engineering, Vignan's Foundation for Science, Technology & Research (Deemed to be University), Vadlamudi, Guntur (Dt), 522213, Andhra Pradesh, India[1]

Abstract. This paper seeks to bring together fundamental and advanced techniques of malware analysis and understanding to prepare the malware analysts with strategic skills towards better investigations and insights [1]. The ever-increasing challenges posed by malware on computer networks require computer security specialists to possess a deep knowledge of the malware's inner details and behaviors during execution. This paper outlines the malware analysis technique, starting from the basic approaches to the advanced techniques. The basic analysis includes both static and dynamic, where static is the examination of malware that has not been executed Operations like sample hashing, string analysis, and Portable Executable (PE) file header examination are collected, whereas in dynamic analysis the executable is run in a safe environment with INetSim running in the background, and its behavior is observed using software tools like Procmon, Wireshark, and TCPView. This document also discusses advanced static analysis consisting of disassembly and decompilation to analyze malware's hidden code to gain insight on how it operates and the logic it uses. The use of Cutter as a program of API flow analysis and program flow visualization will help us to understand a lot more about the malware. The malware executables are also analyzed with x32dbg and x64dbg to allow more detailed examination of the malware's behavior to see how it manipulates memory, control flow of execution, and the code that gets executed within.

Keywords: Malware Analysis, Basic Static Analysis, Basic Dynamic Analysis, Advanced Static Analysis, Disassembly, Decompilation, Advanced Dynamic Analysis, Debugging, PE File Analysis, String Analysis, Wireshark, Procmon, Cutter.

1 Introduction

Malicious software, or malware, always plays a role in the most common and observed computer intrusions and security incidents. To define malware, it is software that is built or programmed for causing harm to a user, computer, network, or any organization and can be termed as malware. Viruses, Trojan horses, worms, rootkits, scareware, and spyware are well-known variants of malware. Malware analysis is an activity in which it is dissected to understand how it actually works, how to identify it, and how to eliminate it from further installing its persistence in the host or network. When analyzing malware, one should always note what that particular binary can do, how to detect it on the network or host systems, and how to prevent and contain damage from it. Once you suspect or identify a file or executable as malware, your first and foremost job is to identify malware characteristics. Determine attack vectors, understand the potential scope of infection, collect, and then develop the host-based signatures, namely file modifications, registry changes, memory artifacts, and network-based signatures, namely protocol anomalies, communication attempts, and suspicious traffic flows, which can

ICITSM-Part I 2025, April 28-29, Tiruchengode, India

DOI 10.4108/eai.28-4-2025.2357850

be helpful to detect malware infections on computers. When performing malware analysis, you'll need to use a variety of tools in order to see the full picture of that executable. There are two fundamental approaches to malware analysis, namely static and dynamic. Static analysis involves identifying the functionalities, like API calls, strings, hashes, and binary code, of malware without running it [2]. Dynamic analysis involves running the malware in a simulated network to monitor its behavior across the host and network [3]. And both of these techniques are further categorized as basic and advanced [4]. This paper mainly focuses on malware found on the Windows operating system.

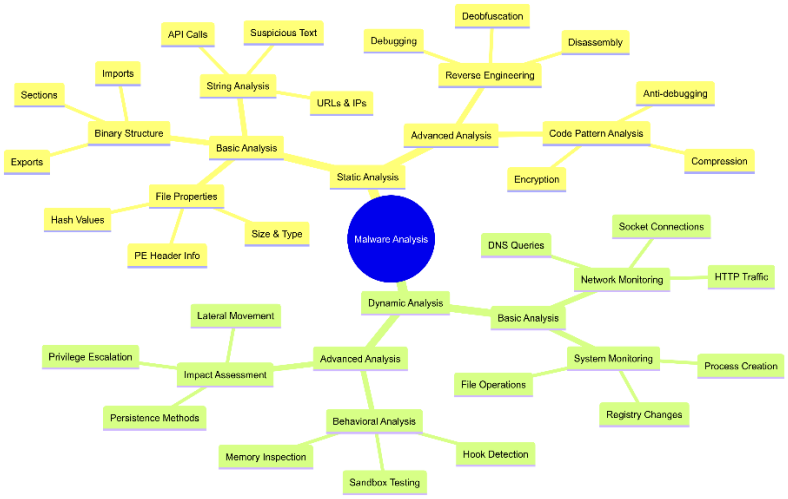

Fig. 1. Malware analysis framework.

It can be helpful to follow some procedures when doing malware analysis to ensure useful results. First is to control the overwhelming details of a piece of malware, especially the advanced persistent ones. Focus on the most irrefutable functionalities and behaviors. Another thing is to know that there are a great number of tools and techniques, each designated for its purpose. If a tool is not useful, use another one or use a different approach to the malware and analyze it from that angle for added perspectives. Last but not least, remember that this is a field of study that is changing continuously as analysts vie with malware creators. When analysts come up with new methods, malware writers find new ways to conceal their work. To be successful, one has to be flexible and learn how these changes are made. Fig 1 shows the Malware analysis framework.

2 Basic Static Analysis

The first step in studying/analyzing malware is static analysis. It is like checking the contents of the malware, e.g., source code and other related information, without running it. The goal of static analysis is to get an idea of what malware can do and confirm whether it is malicious or not [5], [9].

A. Information gathered during basic static analysis includes

- Antivirus software can also be used here to check if the file is dangerous. This helps in validating whether it is malware or not [7].
- A hash is created, which serves as the malware's identifier, like a fingerprint [8].
- Strings can be located within the file. Those strings (words) are helpful in revealing what the malware is programmed to accomplish, such as specific websites it could potentially visit [6].
- PE header, one of the file headers that obtain essential data that can allow you to comprehend the file [5].
- Verifying whether the malware is packed or obfuscated. This means that some of the parts were hidden to make analysis harder [5].

B. What a malware analyst should focus on during basic static analysis data
   - They should mainly focus on every feature rather than focusing and getting lost on every small detail because malware can be very big.
   - They should try to uncover critical information from strings, the import functions that the file uses, and also the headers of the file.
   - Looking at website patterns or file paths can make analysis faster.

C. What a malware analyst should not focus too much on
   - Analysts should not spend too much time on one thing if they get stuck, and it's always better to look for a different way.
   - Do not think that just doing basic static analysis is needed to fully analyze and understand the malware behavior.
   - Do not be so sure of what the malware does because we need to run the malware to know that. Always be aware that static analysis is just guessing.

D. What tools are used?
   - Antivirus programs to Ascertain viruses like Virustotal [7], Kaspersky etc.
   - Tools to get file hashes like SHA256 and MD5SUM [8].
   - Floss (and maybe strings) to get all the available words, sentences, and strings from the files [10].

3 Safe and Controlled Execution

Virtual machines allow one to run malware safely in an isolated environment without risking the host computer or network, making them crucial for malware analysis [12], [13].

A. Why are virtual machines preferred over physical machines?
   - Although physical workstations not connected to the internet (air-gapped) can be used, they come with disadvantages.
   - Air-gapped networks do not have internet access, making it impossible to utilize many malware samples that require full functionality [14].
   - Tools like Veeam, Rescuezilla, Clonezilla, and Macrium Reflect that help in backup imaging often make the removal of malware on physical machines impossible.
   - A major drawback is that certain malware is capable of identifying that they're being executed in a VM and alter their behavior to make analysis more difficult [13]. Even

so, due to the high risk of using physical machines, virtual machines are the most commonly used for dynamic analysis [12].

Despite VMware providing a safe environment, one must keep in mind that certain malware will employ anti-VM techniques to detect virtualization. Malware authors will detect them and execute different algorithms or behaviors to change execution [13].

B. Setting up a virtual machine

Steps to Create Your Malware Analysis Machine:

- One of the prerequisite conditions is having virtual machine software, like Oracle's VirtualBox or VMware Player you wish to use [12].
- Set the hardware settings. If you do not have particular requirements, the default settings should work fine.
- You should install a guest OS. Since most malware targets Windows, you should get a copy of Windows 10 [12].
- Append all the tools and applications required for malware analysis. The most prescribed method is to use Mandiant's FlareVM, which has a lot of malware analysis tools installed in it [15].

C. Configuring VMware for Malware Analysis

- Disconnecting the network is an option but is not recommended since it limits your ability to analyze certain network activity.
- It is preferable to set up a custom network where the VM running the analysis is either isolated or connected to a second VM that offers mock network services. In the source, Fig 2 and 3 illustrate a custom configuration with two virtual machines interconnected [14]. Fig 2 shows the Virtual Lab's Network Adapter setup.

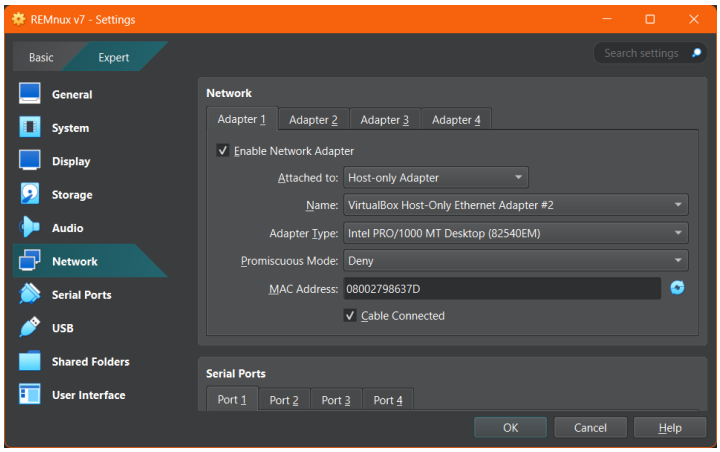

Fig. 2. Virtual Lab's Network Adapter setup.

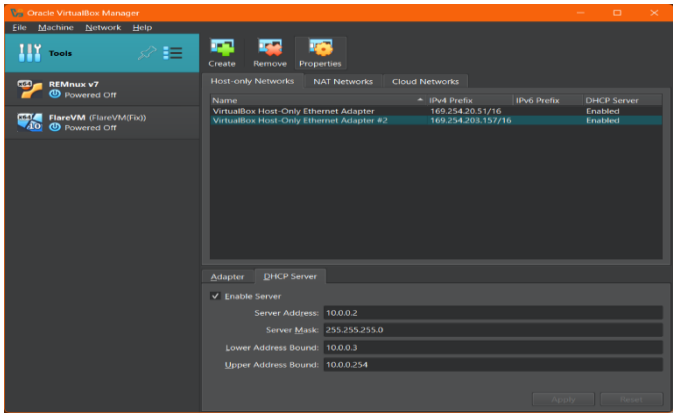

Fig. 3. Virtual Lab's Network DHCP server setup.

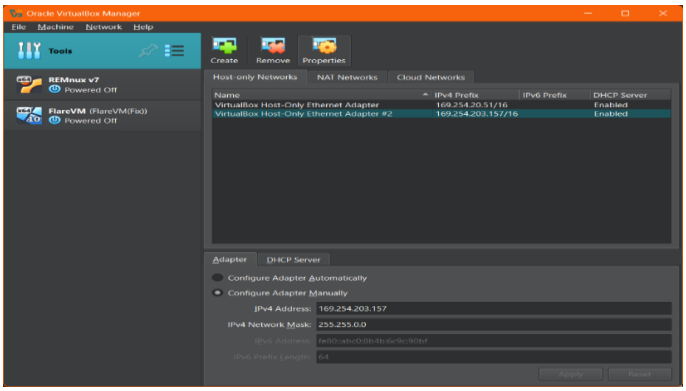

Fig. 4. Attaching Guest OS with new network adapter.

- For some malware, it might be necessary to simulate some of the network services that the malware depends on, e.g., HTTP, DNS, and others, to enable full exercise of malware's network functionality. Such tools like INetSIM, usually installed on REMnux, a Linux VM on the virtual network [14].
- You have the option to control the access of peripheral devices to the guest OS, allowing for connecting and disconnecting.
- After completing installation of the OS, the required tools, and network configuration, you can take a snapshot as this will be your clean base image. Take this snapshot and before analyzing a new malware piece, make sure to revert back to the snapshot [12].
- VMware's Snapshot Manager allows managing snapshots where you can take multiple snapshots and revert to a previous state enabling branching your snapshot history [12]. Fig. 3 shows the Virtual Lab's Network DHCP server setup. Fig. 4 shows the Attaching Guest OS with new network adapter.

4 Basic Dynamic Analysis

This is a crucial approach to malware analysis, which concerns controlling and executing malware in a virtual environment and observing how it behaves. This approach is most commonly used after performing basic forms of static analysis to further understand how the malware operates, especially when static analysis methods are complicated because of encryption or compression. Analysts can see the real-time actions taken by the malware, including the ones that would otherwise not be visible using static methods [1], [3], [4], [16].

A. Definition and Purpose
- The primary methods of basic dynamic analysis consist of executing malware and watching the results that come in to assist in creating tools that effectively remove infected files and devising effective signatures [1].
- One of the main primary aims is to ascertain how much damage a binary file is capable of causing, the means available for identifying its presence within a network, as well as how its damage can be contained and measured [1].
- Support and confirm the results of basic static analysis, as dynamic analysis works hand in hand with these types of analysis processes [3].
- It makes it possible to detect unattended events like changes in files and directories, changes in the structure of the system registry, changes in processes, and communication with other computers as well as reconnaissance activities directed toward the network [1], [17].
- Analysts are able to create simulations of network services that might be necessary for the malware to work by observing the malware in action [18].

B. Key Activities and Focus Areas
- **Safe and controlled execution:** Running malware in a safe and unreachable environment eliminates any risk to the analyst's computer or network during system execution [12], [14].
- **Behavioral Monitoring:** Analysts should primarily concentrate on listed actions:

– Tracking events such as file creation, deletion, modification, or opening. – Tracking changes for the creation, deletion, or modification of keys and values within the Windows Registry.
– Monitoring processes spawned and terminated by the malware.
– Tracking network connections, DNS queries, and data transfer.
– Recognizing mutex creation and observing creation, modification, or deletion of services on Windows

These activities are crucial for understanding the behavior of malware [1], [17].

- **Snapshotting:** Taking snapshots pre and post malware execution is critical to system restoration while maintaining a secure analysis environment [12], [14].
- **Document Findings:** Keep detailed notes of observed behaviors, tool outputs, and any generated indicators.

C. Limitations

- During a single session of dynamic analysis, it is possible that not all executable code paths contained within the malware will execute [1], [16].
- More advanced malware might adopt some form of antiVM technology to recognize that it is working under a VM and either change its execution path or stop running altogether to avoid being analyzed [1], [16].
- There are many pieces of malware programmed to perform an act of destructive behavior solely after certain predetermined trigger events like a particular date and time or a specific user action that is most often difficult to imitate in dynamic analysis [1].
- Virtual machines, for instance, are known to pose threats to themselves as well as to their primary host machines. Using dynamic analysis exposes the malware to possible unmapped threats within lapses of vulnerability on the machine's primary operating system. Patching systems that run dual operating slots has tremendous importance [12], [14].

D. Commonly Used Tools
- **Procmon:** One of the most advanced tools for monitoring activity in real time such as file activity, registry activity, and process activity [18].
- **Process Explorer:** Lists all running processes, including CPU and memory consumption, loaded DLL files, and the structure of parent and child processes [18].
- **Wireshark:** An application that serves the purpose of monitoring protocols to capture network traffic caused by certain malware.
- **TCPView:** A tool for Windows that provides extensive information on all listening and established TCP and UDP connections.
- **INetSim:** A suite designed to provide realistic simulation of internet services.
- **Regshot:** A tool that makes a vault of registers and allows you to change a single register for comparison purposes, letting you pinpoint the changes [18].
- **GFI Sandbox:** Performs automated execution and reporting of malware activity.

5 Advanced Static Analysis

Unlike the primary techniques of analysis, Advanced Static Analysis delves deeper into the analysis of malware, exploring the internal components and operation of malware without executing it. This approach focuses on how the program is functioning at the processor level by reverse-engineering through its binary code.

A. Key aspects of Advanced Static Analysis
- **Disassembly and Decompilation:** The advanced static analysis process start when the executable is scanned in a disassembler which exposes the assembly textual representation of the code. At this stage, IDA Pro is one of the foremost tools. In other cases, advanced static analysis will aim at translating assembly code to a higher-level language through decompilation for easier comprehension of the malware's logic and operation, especially its concealed parts. [19]
- **Understanding Program Instructions:** Advanced static analysis requires specialized skills on disassembly which increases with understanding of the Ascendingly, knowledge of the x86 instruction set is paramount. This means they

have to identify the various opcodes, operands, and registers functions. To achieve this level of sophisticated analysis, knowledge of assembly is paramount. [1]

- **Utilizing Disassembler Tools:** Tools such as IDA Pro have very great capabilities for performing advanced static analysis. These tools do function discovery, stack and local variable analysis which makes it easier for the analyst to relate to the understanding of the original source code. Also, IDA Pro identifies and names library code through code signature FLIRT. Moreover, it is used as an interactive static analysis tool where analysts can add comments, label data and name functions which can be saved in IDA Pro databases. Furthermore, Cutter is useful for API flow analysis and program flow visualization. Although primarily dynamic analysis tools, x32dbg and x64dbg disassemble control flow and can be used to analyze the control flow statically. [19]
- **Identifying High-Level Constructs in Assembly:** In advanced static analysis it is very important to identify the common high-level constructs from lower level programming languages such as C to disassembled code. It becomes easier for the analysts to comprehend the high-level functionality of assembly code without looking at every single instruction painstakingly. One knowing how most compilers change high level code into assembly is also an added advantage. [19]
- **Analyzing Malicious Windows Programs:** With static Windows malware analysis, having a deep comprehension of the Windows Operating System is critical, to understand how its components interrelate with malware Windows suffers from. This familiarity entails knowledge of Windows API, how denial of service attacks through the host system (the registry, file system, ADS) is implemented, executing code bypassing the main executable (through DLLs and process injection), and the degree to which malware uses kernel mode for stealth and advanced operations. For this part, knowing important windows functions is necessary. [1]
- **Analyzing packed and obfuscated files:** For samples suspected of being packed or obfuscated, basic static analysis is usually non-existent due to a lack of resources. However, advanced techniques apply, such as analyzing the stub for unpacking or loading specific program sections into IDA Pro, and manually defining code segments when full unloading isn't feasible. Detecting packers using tools like PEiD becomes useful. [10], [19]
- **Shellcode Examination:** The higher levels of static analysis encompass analyzing shellcode which does not have an executable file, therefore it must be manually loaded into disassemblers like IDA Pro. [1], [19]
- **Steps Towards Further Analysis:** The steps of advanced static analysis aim to provide all detail possible regarding malware's hidden features and capabilities and functions.

This understanding enables the construction of educated predictions that inform the groundwork for later dynamic or active analysis. [9]

6 Advanced Dynamic Analysis

Advanced Dynamic Analysis utilizes sophisticated techniques of malware analysis, which incorporate inspecting the internal processes of an executable file with a debugger employed in dynamic analysis. This type of analysis surpasses watching the malware's actions in a sandbox (basic dynamic analysis) and deeply intuits the mechanisms of its functioning. The techniques

of advanced dynamic analysis serve the very important goal of obtaining additional, deeper information from the executable when such information is hard to obtain using static or basic dynamic analysis.

A. Key aspects of Advanced Dynamic Analysis

- **Debugging as a Core Technique:** More advanced methods of dynamic analysis rely primarily on debugging. This is the process of commanding the execution of the malware using specific software tools known as debuggers. A debugger permits control over the execution, allowing for the stepping through of instructions, examination of memory, and observation of register values during the malware's running. [19]
- **Examining Internal State:** The critical feature of advanced dynamic analysis is observing the internal state of a malware being executed. This involves:

– **Memory Manipulation:** Watching the processes of reading and writing to memory by the malware. Analysts can observe specific areas of memory which are being used and altered by the malware, using debug tools.

– **Control Flow of Execution:** Stepping through the instructions in the debugger enables analysts to control and understand the execution flow of the program: which parts of the code are run and the order they are executed. This is vital in reasoning about understanding complex logic or anti-analysis techniques configured to stop analysis.

– **Register Values:** The values held in the CPU's registers, which are essential for the smooth execution of the program, together with its current state that can be analyzed by the debugger.

– **Stack Analysis:** Using a debugger allows viewing various parts of the call stack which indicates the functions executed preceding the current execution pointer for using set value. [18], [19]

- **Overcoming Limitations of Other Techniques:** Advanced dynamic analysis is useful when there are too many difficulties in collecting data using other techniques. For example, it may assist in figuring out how some custom network packets are used or revealing the purposes of certain imported functions that other methods of basic dynamic analysis may only show being sent or received. [16], [19]
- **Integration with Advanced Static Analysis:** Information retrieved from static analysis, such as functions and strings that were discovered, assists greatly in informing the debugging step, including breaking the code at specific checkpoints of interest. On the other hand, explanations coming from debugging, as in, explanations of what is happening during debugging involve clarification of disassembled files which have been disassembled beforehand in static analysis. [3], [19]
- **Analyzing Packed and Obfuscated Malware:** While dynamic analysis is vital in unpacking malware, the use of a debugger is very crucial when analyzing packed and obfuscated malware. Often, analysts can execute the code until they reach the permanently unpackaged code; getting to the original code can sometimes be a challenge, but tools such as OllyDump can aid in the process. For example, this tool can dump the method that has already been removed from the container for other methods from the memory for thorough examination in the future. [19]
- **Analyzing Anti-Reverse Engineering Techniques:** Debuggers play a vital role in discovering and circumventing anti-debugging techniques used by malware developers. Analysts are often able to notice suspicious behavior, such as abrupt program-ending jumps or execution inhibiting conditions, which pyrotechnic anti-

debugging techniques of brilliant logic-scientists employ. Debuggers further permit the real-time alteration of instruction pointers to disable these checks. Anti-Virtual Machine Techniques: The same applies to observing the behavior of the malicious software while inside a virtualized environment. Debuggers aid in understanding and, at times, even circumventing anti-virtual machine strategies. [19]

7 Siko Mode Malware Analysis

This part discusses the exploration on the" Siko Mode" malware sample which is meant to be a challenge for both novice and advanced malware analysists. My goal is to run an analysis in order to try to see its functionality and its behavior. This requires the use of both static and dynamic analysis, and multiple tools and technologies.

A. Scenario Summary
The Siko Mode specimen came from a link. This analysis is meant to find out how deep it runs.

B. Goals

The examination of the Siko Mode malware sample is set on the following primary goals:

- To analyze the sample both statically and dynamically.
- To obtain relevant details that would give insight on what actions the malware performed.

C. Methodology
The mixed-method approach to the Siko Mode binary analysis is recommended, emphasizing the use of different phases of analysis simultaneously. Basic static and dynamic analysis should be performed first, and then advanced static (with decompilers and disassemblers) and advanced dynamic analysis (with debuggers) should be performed as necessary.

D. Initial Basic Static Analysis
The initial steps in the analysis involve basic static techniques to gather preliminary information about the malware. This typically includes:

- **Hashing the sample:** Calculating file hashes (like SHA256) to uniquely identify the malware. This allows for checking if the sample has been previously analyzed. Fig. 5 shows the Calculating file hashes.

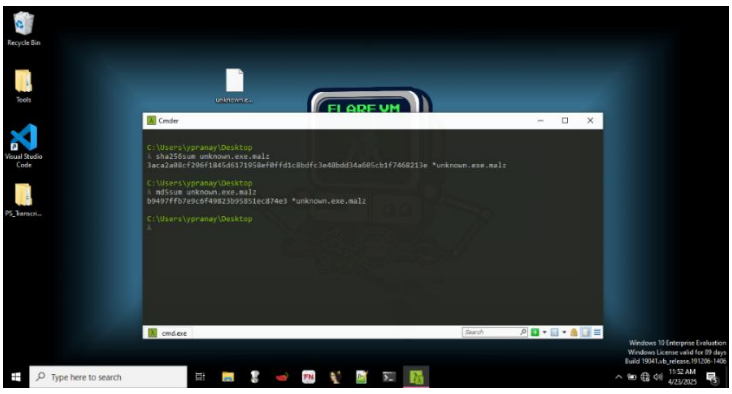

Fig. 5. Calculating file hashes.

- **Submitting hashes to VirusTotal:** Checking online malware repositories like VirusTotal using the calculated hashes to see if the sample is known and if any reports are available. Fig. 6 shows the Analysing the file hashes.

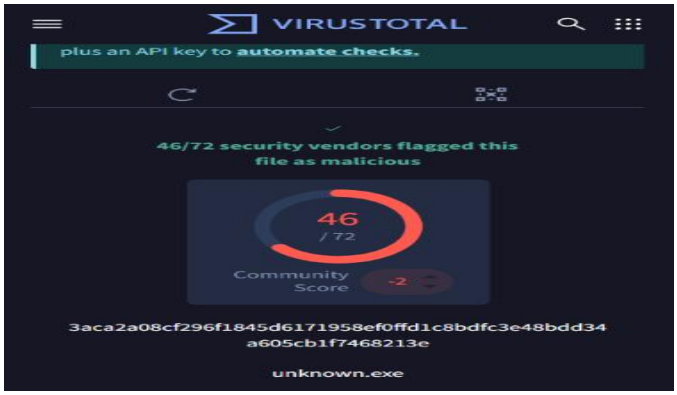

Fig. 6. Analysing the file hashes.

- **String analysis using Floss:** Employing tools like Floss to extract human-readable strings embedded within the binary. These strings can provide initial insights into the malware's potential functionalities, such as URLs, file paths, or commands. Fig. 7 shows the Extracting strings from the malware.

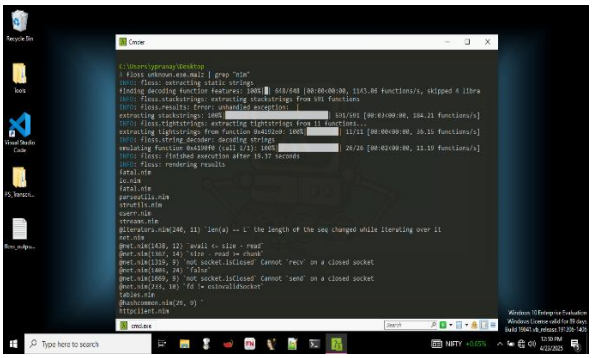

Fig. 7. Extracting strings from the malware.

E. Basic Dynamic Analysis

- **Execution and Network Monitoring:** The analysis starts with executing the unknown.exe binary in a sandbox environment integrated with InetSim and Wireshark. This setup enables monitoring network interactions of the binary in real-time. Fig. 8 shows the INetSim running the internet simulation.

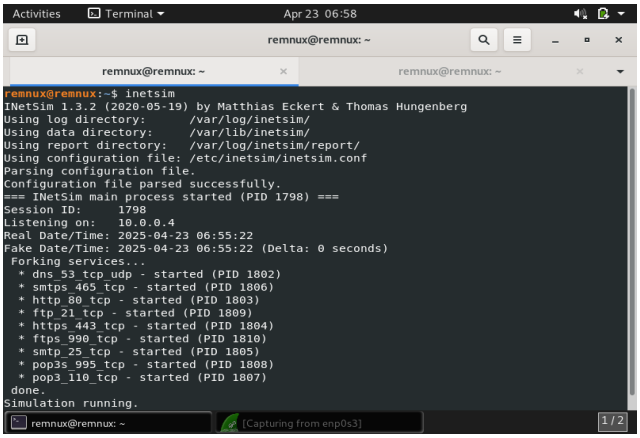

Fig. 8. INetSim running the internet simulation.

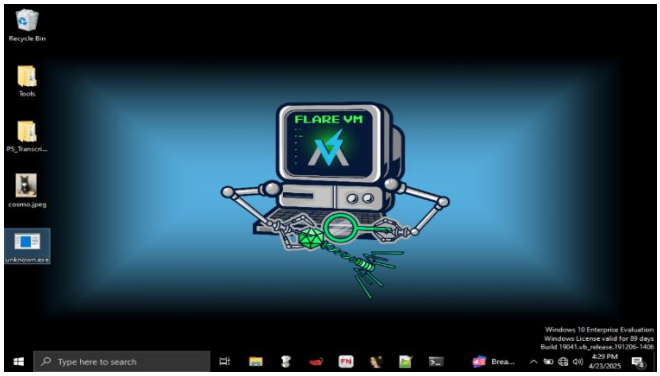

Fig. 9. Detonating malware.

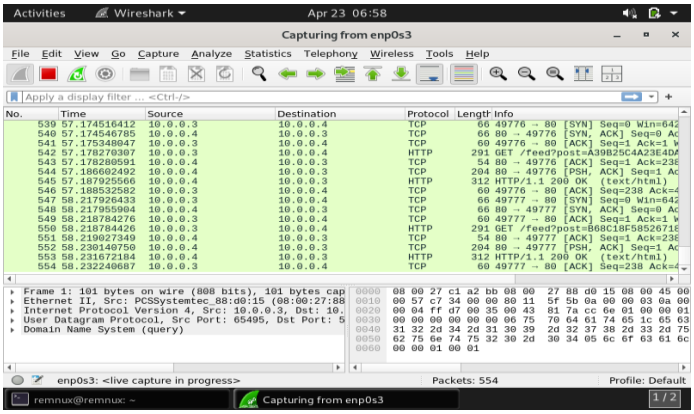

Fig. 10. Wireshark for packet capturing.

- Self-Induced Deletion: One of the primary findings identified during initial dynamic analysis stages is the capability of the binary to delete itself. Fig. 9 shows the Detonating malware. Fig. 10 shows the Wireshark for packet capturing.

– **Condition 1:** Failure to Connect to First Callback Domain: When InetSim is idle or turned off, the binary tries to reach its first callback domain. If it is unable to connect, it releases the handle to the URL and erases itself from the system. This can be shown by executing the binary standalone; the unknown.exe file rapidly disappears post-execution.

– **Condition 2:** Interruption During Execution: If InetSim is active and passing the required data, discontinuing the timeline (for instance, stopping InetSim mid-session) while the binary is executing will also cause self-erasure.

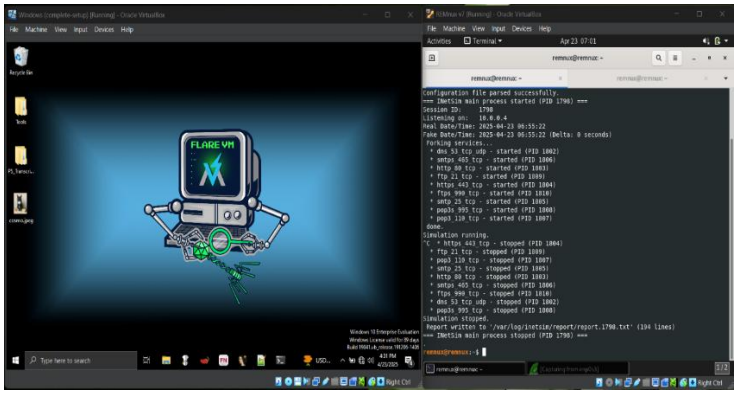

Fig. 11. Malware auto-deletes.

- **Persistence Check:** Basic dynamic analysis includes searching for signs of system persistence within the host using Procmon to check for changes in the registry keys or files saved on disk, capturing file system activities. The result shows that this binary does not incorporate any persistence techniques.
- **First Callback Domain Identification:** The first callback domain was determined to be located at http://update.ec12-4-109-278–3-Ubuntu.20.04. local after executing the binary on a virtual machine with InetSim while simultaneously monitoring its network traffic using Wireshark. This can be proved due to the fact that this domain gets accessed right after the TCP handshake during the HTTP handshake phase. Also, this domain is not one of the domains noted in the strings of the binary, which means it can only be detected through dynamic analysis. Fig. 11 shows the Malware auto-deletes.

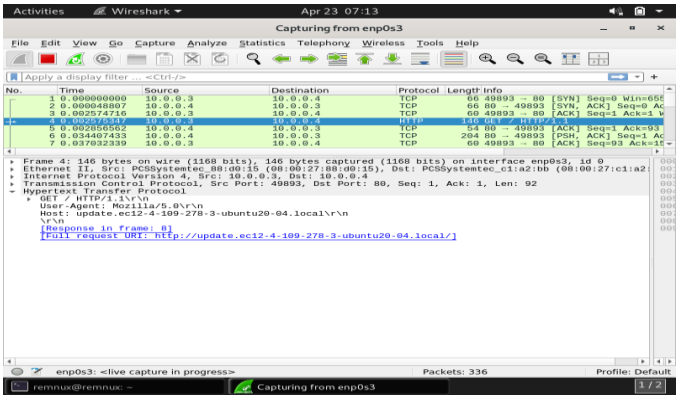

Fig. 12. Malware calling a callback URL.

- **Conditions for Data Exfiltration:** It has been found out that the exfiltration of data occurs at the moment binary is able to establish a successful connection with the first callback domain. Should this connection be established, this binary will automatically exfiltrate data. Fig. 12 shows the Malware calling a callback URL.
- **Exfiltration Domain:** Further network monitoring has disclosed another domain that appears to be serviced by the data exfiltration: http://cdn.alimter.local. As noted, the binary divides its HTTP calls between the first callback domain and this exfiltration domain.
- **Type of Exfiltrated Data and Transmission Method:** The file that is being exfiltrated is identified as 'cosmo.jpeg '. The file's information seems to be extracted and kept in an encoded or compressed format to be sent as a value of a post field in a GET request destined for the /feed URI of the exfiltration domain. Although technically categorized as a GET request, the data retrieved in the header under the post section is in fact, forwarded to the web server and can be recorded. Fig. 13 shows the Malware sending request to domain for data exfiltration

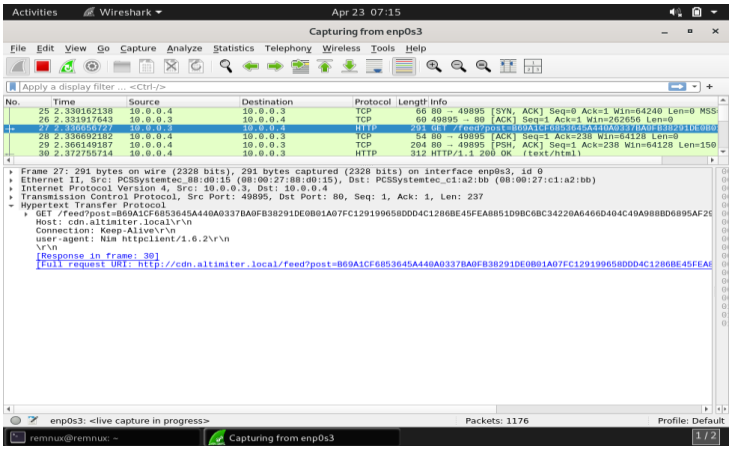

Fig. 13. Malware sending request to domain for data exfiltration.

F. Advanced Static Analysis

- **Language Confirmation:** Initial static analysis done with Floss suggested the presence of strings and function calls (nim main, nim main inner, nim main module) implied the binary was written in NIM. More sophisticated static analysis corroborated this claim by examining the scope and features of the compiled binary.
- **Architecture Confirmation:** Basic static analysis with PE View gave us some bare-bones hints regarding x64 was possible, but limited. PE Studio validated that the binary was indeed x64, meaning it was designed for a 64-bit CPU. Also inspecting the disassembly with some other tools like Cutter showed us the presence of 64-bit registers RAX and RIP which adds further proof.
- **Encryption Algorithm Identification:** A careful analysis of the output strings (Floss can also be used) and the imported libraries shows some references to "RC4". To be specific, there is a two RC4 method call and a for sure library import of RC4.nim.c. That is enough proof that RC4 is the encryption algorithm used here.
- **Finding the Encryption Routine:** An analyst may search for the two RC4 method within the binary using a disassembler like Cutter. An analysis of control flow and cross-references show nim.steel stuff method calls the two RC4 method, which means this part of the code must do the encryption. More detail on the loop within sim.steel stuff suggests that the data being imported into the program is eventually encrypted using RC4. Fig. 14 shows the Function for RC4 Encryption Algorithm.

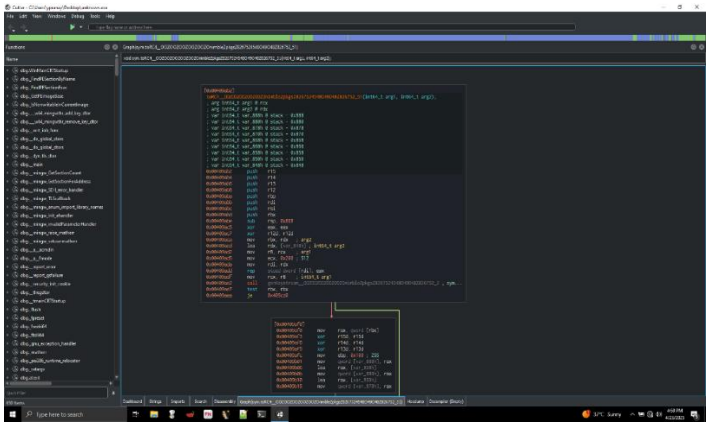

Fig. 14. Function for RC4 Encryption Algorithm.

- **Identifying the Self-Deletion Function:** Conducting a search in Cutter for the string "Houdini" reveals that this is also the name of the method responsible for self-deletion.
- **Assessing the Importance of "Houdini":** Examining nim main module function in Cutter reveals from where the Houdini function is called.

– It is called if the check against the "kill switch URL" (the first callback domain) is not true.

– Called after running the main routine (unpack resources and steal stuff) whether or not it was successful.
– Also called if the execution of the main routine is broken off at any point during the process. Fig. 15 shows the Self-Deletion Houdini Function – 1. Fig. 16 show sthe Self-Deletion Houdini Function – 2.

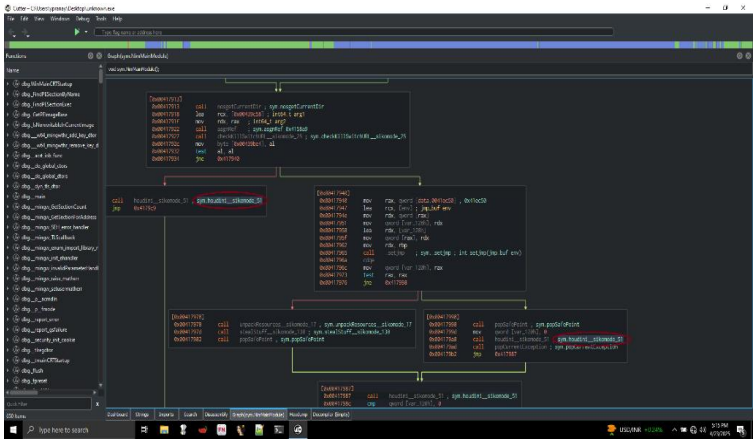

Fig. 15. Self-Deletion Houdini Function – 1.

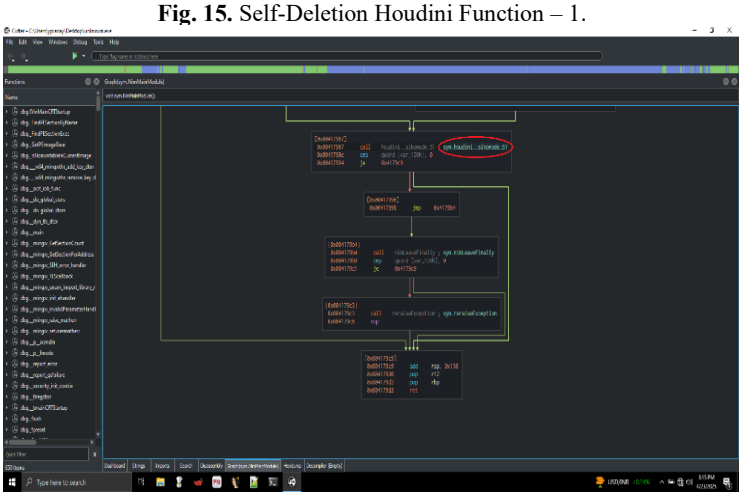

Fig. 16. Self-Deletion Houdini Function – 2.

G. Advanced Dynamic Analysis

- **Finding the Encryption Key:** Outlined in earlier sections, the file RC4 has an encryption key that can be found using advanced dynamic analysis with Procmon. There's notable file access on password.ext that's visible in C:\Users\Public as well as on CreateFile.

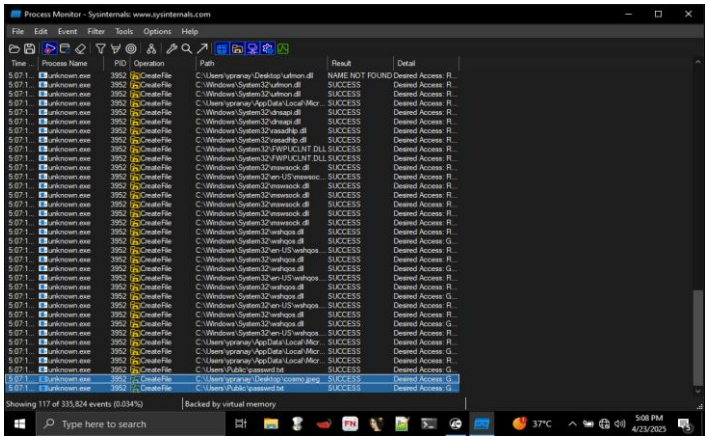

Fig. 17. Finding the Encryption key using procmon.

- **Extracting the Encryption Key:** The previously mentioned file appears to contain a string with the password object. The password (key) for the RC4 encryption routine along with its parameters and procedures is termed SikoMode. Fig. 17 shows the Finding the Encryption key using procmon.

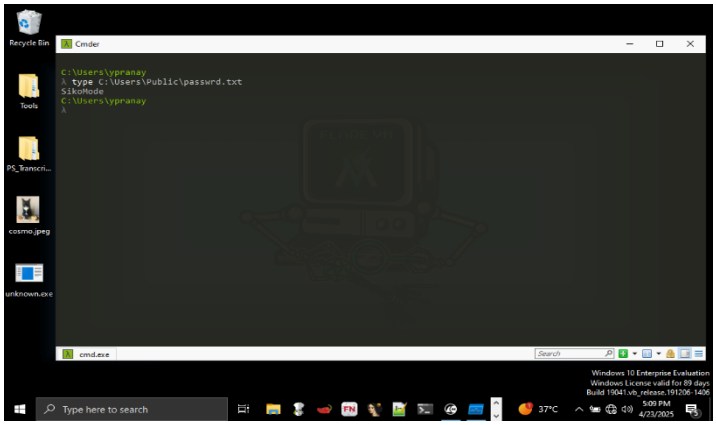

Fig. 18. Key for RC4 Encryption.

Summarizing in short, further breakdown of static analysis displays it quite clearly features malware designed in NIM around a 64-bit architecture. Also, it contains the means for self-deletion (if not initially connecting to the initial callback, interrupted, or completion of operations). A binary containing cosmo.jpeg is extracted using a GET request to its first callback's domain as a post parameter. It is also discovered that RC4 is the algorithm of choice for encryption when the domain is first contacted. It was further determined that the path C:\Users\Public \password.txt is where the key "Siko mode" is stored. The function "Houdini" is responsible for isolation self-deletion and goes off during numerous alternates through operation of the binary. Fig. 18 shows the Key for RC4 Encryption.

8 Conclusion

To summarize, this paper analyzed current malware analysis with a wide array of depth and breadth. It is quite necessary for any cybersecurity professional to have a grasp of the methodologies due to rapidly evolving malware threats. For example, we have covered basic static analysis, which includes the non-executive inspection of malware samples through hashing, string extraction, and PE file header analysis, as well as sample examining heuristics. These methods unlock important possibilities of what a malware sample is capable of and provides critical insight to the structure of the sample. The paper also focused on basic dynamic analysis, where malware is run in a controlled environment such as a virtual machine running INetSim to better understand the sample's processes. The behavior of malware processes on the host and the network has to be captured, and Procmon, Wireshark, and TCPView do just that. Such active monitoring help uncover some of the crucial initial indicators of compromise and give clearer insights of what the malware does.

After the basics were introduced, the discussion moved to advanced static analysis, which involves examining the malware's components and functionality without execution. In this step, disassembly and decompilation will be applied to so that analysts examine the malware's code and logic at the processor level. In understanding malware execution, Cutter aids in API flow analysis and program flow visualization.

Lastly, the paper briefly discussed sophisticated dynamic analysis where debuggers such as x32dbg and x64dbg are used to analyze the behavior of the malware in real-time. With this advanced technique, analysts can check how memory is being modified, the flow of execution of programs, and the command execution by the malware. These actions which dynamic analysis does not readily provide meaningful insights into aids in understanding control at lower levels.

I have stressed in this paper the need for a systematic method in conducting malware analysis alongside the application of varying methodologies and tools. An adaptive and everevolving relationship between an analyst and the malware is key in understanding emerging malware threats.

References

[1] S. M. A. Othman et al., "A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis," Int. J. Inf. Technol. Comput. Sci., vol. 10, no. 10, pp. 1–20, 2018.
[2] M. M. Masud, L. Khan, and B. Thuraisingham, "Static malware analysis using machine learning methods," in Advances in Digital Forensics X, Springer, 2014, pp. 155–168.
[3] S. S. Rao and K. V. N. Sunitha, "Integrated static and dynamic analysis for malware detection," Procedia Comput. Sci., vol. 46, pp. 302–309, 2015.
[4] Aqua Security, "Malware analysis: Static vs. dynamic and 4 critical best practices," 2023.
[5] Infosec Institute, "Static malware analysis," 2015.
[6] N. Bencherchali, "Malware Analysis Techniques — Basic Static Analysis," 2019.
[7] Theta432, "Malware Analysis Part 1: Static Analysis," 2020.
[8] TechTarget, "Top static malware analysis techniques for beginners," 2021.
[9] CrowdStrike, "Malware Analysis: Steps & Examples," 2023.
[10] Mandiant, "FLARE Obfuscated String Solver," accessed 2025.
[11] Winitor, "PEStudio," accessed 2025.
[12] L. Zeltser, "How to Get and Set Up a Free Windows VM for Malware Analysis," 2019.

[13] GeeksforGeeks, “Virtual Machine for Malware Analysis,” 2023.
[14] M. Abdulkabir, “Building a Malware Analysis Lab with VMware Workstation: A Step-by-Step Guide,” 2023.
[15] Mandiant, “FLARE VM,” GitHub, 2023.
[16] TechTarget, “How dynamic malware analysis works,” searchsecurity.techtarget.com, March 1, 2024.
[17] S. Ilic, M. Gnjatovic, I. Tot, B. Jovanovic, N. Macek, and M. G. Bozovic, “Going beyond API Calls in Dynamic Malware Analysis: A Novel Dataset,” Electronics, vol. 13, no. 17, p. 3553, 2024.
[18] J. Gohel, “Dynamic Malware analysis,” Medium, April 8, 2024.
[19] M. Sikorski and A. Honig, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, No Starch Press, 2012.