A Comparative Malware Analysis of XWorm and Nanocore: Laying the Groundwork for Enhanced Detection Strategies

by

Shakil Islam Shanto
24341292
Afsana Mimi Punom
20101267

A thesis submitted to the Department of Computer Science and Engineering in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Shakil Islam Shanto	Afsana Mimi Punom
24341292	20101267

Approval

The thesis titled "A Comparative Malware Analysis of XWorm and Nanocore: Laying the Groundwork for Enhanced Detection Strategies" submitted by

1. Shakil Islam Shanto (24341292)
2. Afsana Mimi Punom (20101267)

Of Fall 2024, has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on November 30, 2024

Examining Committee:

Supervisor:
(Member)

Dr. S M Taiabul Haque

Associate Professor
Department of Computer Science and Engineering
Brac University

Co-supervisor:
(Member)

Md. Faisal Ahmed

Lecturer
Department of Computer Science and Engineering
Brac University

Thesis Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam

Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Dr. Sadia Hamid Kazi

Associate Professor & Chairperson
Department of Computer Science and Engineering
Brac University

Abstract

Malware continues to evolve, posing a significant challenge to global cybersecurity through sophisticated techniques such as obfuscation, process injection, and persistent Command-and-Control (C2) communication. This study conducts a comparative analysis of two active malware strains, XWorm and NanoCore, to uncover shared tactics and unique features that enable their evasion and impact. By employing static, dynamic, and reverse engineering analyses, the research identifies commonalities in delivery methods, persistence mechanisms, and payload obfuscation. Tailored YARA rules are developed to enhance malware detection frameworks, providing practical tools for real-world applications. The findings emphasize the importance of behavior-driven detection strategies and propose generalized mechanisms to address broader malware families, moving beyond isolated strain analysis. This research not only bridges critical gaps in understanding modern malware but also lays a foundation for scalable and adaptive defense systems, contributing to the ongoing battle against evolving cyber threats.

Table of Contents

List of Figures

Chapter 1

Introduction

Since the discovery of the very first malware, Elk Cloner in 1982. They have evolved into multiple classes with specific specializations. From spying to ransoming, every possible way of targeting and exfiltrating for profit from a victim present in a malware nowadays. Besides banking trojans like BBToK, which specializes in extracting assets from the victim using various methods, we have XWORM which does the same but in a more creative way. We also see long running malwares like NanoCore, which have stayed alive for more than a decade. These are only three out of hundreds and thousands of malwares with every single one with distinct features. An anti-virus is not enough to defend against them nor isn't an analysis of one single malware. But at the end of the day, no matter how distinct they are in their attacks and features, some similarities remain constant.

Malware threat has been rapidly evolving into one of the biggest cyber threats besides data breaches. Malwares continue to adapt and evolve by utilizing techniques such as obfuscation, dynamic payload delivery, and persistent Command-and-Control (C2) communication. Among countless number of malwares XWorm and NanoCore stand out due to their relentless persistence and versatile attack strategies.

This research performs comparative analysis on XWorm and Nanocore malwares. Through the analysis, this research finds critical similarities that may usually go unseen using generic malware analysis frameworks. These identification of distinct characteristics helps achieve the proposed contribution of the research. This methodology proofs valuable for the further development of more robust detection frameworks and also, addresses the necessity of in-depth malware analysis.

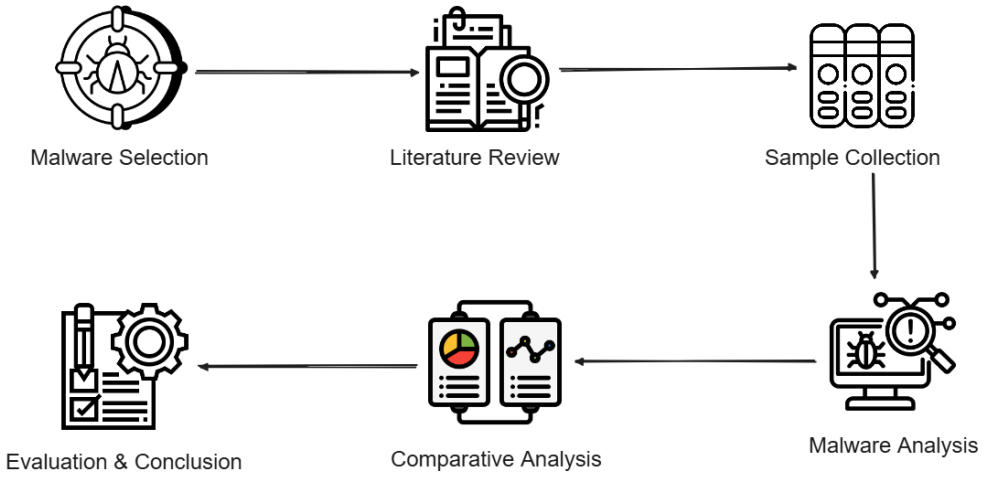

Figure 1.1: Workflow Diagram

1.1 Motivation

Malware attacks have become a major and evolving threat in today's digital world, affecting both businesses and individuals worldwide. Notable incidents, like the NotPetya ransomware attack in 2017, which caused over $300 million in losses for Maersk, and the MyDoom worm in 2004, which led to damages of $38 billion, highlight the serious financial and operational impacts of these threats. Even with improvements in cybersecurity measures, malware keeps changing and adapting. Recent incidents, such as the SolarWinds supply chain breach, show how skilled attackers can exploit even trusted systems, particularly targeting critical sectors and government entities. Modern malware increasingly adopts advanced techniques, including dynamic obfuscation, fileless payloads, and persistence strategies, which allow them to bypass conventional security measures with greater efficacy. The ongoing detection of malware like XWorm and Nanocore, identified as recently as late 2024, highlights the inadequacies of our existing mitigation efforts and stresses the necessity for the development of novel strategies.

This research is motivated by the pressing need to unify disparate malware analyses with broad and applicable insights. By investigating prevalent behaviors such as obfuscation techniques, command-and-control (C2) communications, and persistence tactics, the study aims to uncover patterns that can inform the development of proactive detection and prevention strategies. Understanding these common traits is crucial for tackling not only individual malware variants but also more extensive malware families, thus strengthening defenses against possible future threats.

The relevance of this study extends beyond academic interest, it serves crucial function in combating intricate cyber threats. The research aims to offer useful information for scholars, industry members, and security experts, highlighting the need for defense systems that can grow, adapt, and work together.

1.2 Research Problem

Modern malware, such as XWorm and NanoCore, still manages to avoid detection despite major improvements in cybersecurity. They use clever methods like fileless payloads, dynamic obfuscation, and ongoing Command-and-Control (C2) communication. Previous studies have looked closely at older malware types, but they often miss the newer, more active threats. While automated analysis tools are common, they usually do not provide the detailed insights needed for effective detection strategies.

This research addresses two critical challenges: the insufficiency of current detection systems to adapt to rapidly evolving malware tactics and the underrepresentation of manual, in-depth analysis in existing literature. By focusing on XWorm and NanoCore, two of the most active malware families targeting financial systems and critical infrastructure, this study aims to uncover shared tactics and vulnerabilities that can inform more robust, scalable, and explainable detection mechanisms. Through a layered approach combining static, dynamic, and reverse engineering analysis, this research seeks to provide a foundation for developing proactive defense strategies that bridge the gap between isolated malware studies and holistic, behavior-driven insights.

Research Question

This study aims to address the question: *What common techniques and characteristics can be identified through a comparative analysis of advanced malware families, and how can these insights enhance our understanding of their evolving attack mechanisms to improve detection frameworks?*

1.3 Contributions

This research makes the following significant contributions to the field of malware analysis and detection:

Manual Comparative Analysis: A comprehensive manual analysis of two active malware families, XWorm and NanoCore, is presented. This approach goes beyond the surface-level insights provided by automated tools, delving deeply into persistence mechanisms, obfuscation layers, and Command-and-Control (C2) communications.

Development of YARA Rules: Custom YARA rules are developed to detect the unique features of XWorm and NanoCore. These rules include markers for registry manipulation, process injection, and obfuscated payloads, providing practical tools for improving malware detection frameworks in real-world environments.

Addressing Research Gaps: The study focuses on currently active malware strains, which are underrepresented in existing literature. It also contributes to the limited body of research on malware specifically targeting financial systems, addressing a critical need for timely and relevant analysis.

Foundation for Explainable AI Integration: By identifying behavioral patterns in malware functionality, the research lays the groundwork for the application of explainable AI in detection systems. These findings can be used to build interpretable

and adaptive models for proactive threat mitigation.

Generalized Defense Strategies: The study proposes actionable defense mechanisms that are broadly applicable across malware families. These include enhanced monitoring of system modifications, detection of obfuscated scripts, and traffic analysis of dynamic DNS services, creating a foundation for more effective and generalizable malware defenses.

Through these contributions, this research not only deepens the understanding of XWorm and NanoCore but also offers tools and frameworks that can be applied to broader malware detection and prevention efforts.

1.4 Research Objectives

The following are our study objectives,

1. To investigate the structural, behavioral, and functional aspects of malware, focusing on advanced strains like XWorm and NanoCore.
2. To identify commonalities and differences in attack strategies, persistence mechanisms, distribution methods, and evasion techniques between XWorm and NanoCore.
3. To highlight weaknesses in existing detection frameworks by examining sophisticated techniques like process injection, memory manipulation, and obfuscation, pointing out where deeper monitoring is necessary.
4. To provide guidance on improving detection and analysis frameworks by emphasizing the importance of monitoring abnormal memory behaviors, command-and-control communications, and advanced obfuscation techniques.

Chapter 2

Literature Review

2.1 Malware

Malware is just software with malicious intentions. Like any other programs used in daily life, malware is also a program. The only difference is that, while most programs always obey certain policies and rules, malwares do not. They are like the outlaws of the software city. The first virus ever in the wild, dubbed 'Elk Cloner', was a virus in the boot sector that infected the disk, memory and floppy disk. There was no cryptocurrency back then nor anything worth infecting like we see now. Now, there's all kinds of malware targeting every possible thing from IoT devices to industrial control softwares. Not only this, malwares are not just about stealing information now, malwares can ransom a victim, they can spy on them, if an industrial system falls victim to one, the damage is colossal. Malwares are more victim and demography oriented now. It just doesn't target a simple operating system, it considers everything to amplify the damage.

2.2 XWorm

XWorm demonstrates unique and dangerous capabilities, such as replacing legitimate cryptocurrency wallet addresses with fraudulent ones to steal funds. Its multi-layered distribution approach, using obfuscated PowerShell scripts and legitimate websites, showcases its technical sophistication. Additionally, XWorm offers various payload variants, allowing attackers to adapt to different scenarios and evade detection effectively.

XWorm's impact is evident in its ability to deploy ransomware, as observed in its use by the NullBuldge group to spread LockBit ransomware. LockBit, a highly active ransomware group, has caused over $8 billion in damages globally, targeting critical industries such as finance, healthcare, and infrastructure (Venter et al., 2024). XWorm's targeting of high-risk sectors like transport and healthcare further emphasizes its significance (Trellix, 2024). It has also been associated with the MEME#4CHAN activity cluster, which targeted manufacturing and healthcare facilities in Germany (Ravie Lakshmanan, 2023). The malware's widespread and

calculated targeting strategies underscore its threat to essential industries.

2.3 Nanocore

Nanocore, first detected in 2013, remains highly active and versatile. It enables attackers to execute actions like remote access, keylogging, password theft, and ransomware deployment. Known for its ability to mimic legitimate processes, Nanocore is particularly challenging to detect and neutralize.

Recognized as one of the "Most Wanted Malwares" in 2024, Nanocore has been implicated in several high-profile incidents, including infections that compromised thousands of devices (gmcdouga, 2023). It has caused significant damages, such as data breaches, privacy violations, and hardware failures, resulting in financial and operational losses (Wallen, 2020). The United States Health Sector Cybersecurity Coordination Center has issued warnings about Nanocore, citing its threat to critical infrastructure, particularly in healthcare (HC3, 2020).

2.4 Continued Relevance of XWorm and Nanocore

Both XWorm and Nanocore remain active threats. According to Malware Bazaar, over 800 XWorm samples have been collected since 2022, and more than 7,000 Nanocore samples since 2020. Recent detections on November 26, 2024, for XWorm and November 12, 2024, for Nanocore further highlight their persistent presence (MalwareBazaar, 2024). Platforms like ANY.RUN also reported recent activity, confirming detections as late as November 28, 2024 (ANY.RUN, 2024). The ongoing activity of these malware families underscores their evolving nature and the need for continued research.

2.5 Malware Analysis

This complex structure of the workings and types of malware calls for analysis. Just like how medical doctors find a cure for a virus studying, breaking and even self-infecting the virus. Malware analysis is also not so different. According to Fortinet, "Malware analysis is the study of the unique features, objectives, sources, and potential effects of harmful software and code, such as spyware, viruses, malvertising, and ransomware. It analyzes malware code to understand how it varies from other kinds." [9].

There are many ways to do malware analysis, static analysis, which is a process of analyzing the binary without executing it. It is the easiest to perform and a lot of metadata related to the malware can be achieved through this. It focuses on finding abnormalities in the file itself, not in how it executes. On the other hand, the method where the suspected malware is run in an isolated environment and its behavior is monitored is known as dynamic analysis. Lastly, there's reverse engineering, which involves disassembling a software program. Researchers attempt

to convert the binary instructions to higher level understandable instructions so that they can understand exactly what the malware is supposed to do and pinpoint its area of impact. Only by knowing its details, researchers are able to create solutions which can mitigate the program's intended effects [10].

In this section, similar research done in the past is reviewed. Many methodologies used to achieve their research objectives in these papers can be seen which can be beneficial in the proposed paper. These researches will also help us justify our research objectives.

In paper [1], The authors analyzed specific instances of these attacks, examining their mechanisms and impacts. The analysis involved comparing the approaches of the three malwares and discussing their infiltration and operation strategies. The authors analyzed specific instances of these attacks, examining their mechanisms and impacts. The analysis involved comparing the approaches of the three malwares and discussing their infiltration and operation strategies. The study finds these three malwares are a noteworthy threat to industrial digitization and it also shows the similarities in the attack strategies of these malwares and concludes by showing the rise of these threats. The attacks analyzed within this paper are only limited to digital ICS systems; also it focuses more on describing past events rather than rather new experimental data. Further research should focus on developing and implementing more robust prevention and detection methods against such sophisticated malware attacks in operational technology systems.

The paper [2] studies the lifecycle of IoT malware, it also does a comparison with traditional malware that targets desktop and mobile devices. A large-scale measurement of over 166,000 Linux Based IoT malware samples are collected. The result reveals the differences between IoT malwares and traditional malwares, which is mainly found in the infection part and the C&C part. The results reveal that signature-based detection is less effective for IoT malware because of new detection evasion techniques. The paper concludes with recommendations for IoT device vendors for preventing malware threats. Future work suggests efforts from vendors and improvement in defense mechanisms to counter IoT malware threats.

This paper [3] classifies some malware samples into their respective families. They achieve this my performing a comparative analysis on selected android malwares samples. They focus on the permissions of an application. The experiment is done based on substantial datasets. The datasets are then ran through several machine learning techniques, like decision trees and random forests, support vector machines, logistic model trees, AdaBoost, and artificial neural networks. The paper, although mostly is based on datasets, after examining and analyzing those datasets, they determined the importance of feature engineering which is one of most important things to any malware.

In [4], the researcher's main objective was to understand the inner workings of android malware. In order to achieve these objectives, a Android Malware collection called Kharon is described which gives as much as possible a representation of the diversity of malware type. Analysis on every single malware is also done by dissecting them and reversing their code and also by running them in a controlled environment as if they are executing in real life. By analyzing the behavior of these malwares

in such a low level the researchers can figure out the engineering that goes behind developing them and understand their programming patterns. Future work related to this is to compare these seven malware with large datasets in order to build automatic classification techniques.

The paper [5] talks aims to identify and analyze behavioral characteristics unique to ransomware, as opposed to other types of malware. The study does dynamic analysis and studies malicious code samples to distinguish ransomware features for better detection. The results emphasize on the significance of network and file monitoring to identify ransomware attacks. The paper ends with showing the need for anti-ransomware programs which are attentive to subtle ransomware behaviors.

The focus of paper [6] is to give a thorough insight about the DDoS attack with keeping the IoT Tsunami, DDoS attack in focus where Mirai Botnet was used. It also does an analysis of some known DDoS capable malwares in the IoT and maps their main characteristics. Given the Mirai Botnet source code is publicly available a thorough analysis of this malware is also done. As a result of this research, the authors provided a great insight into the world of IoT malwares which can be useful to the security community as a foundation to tackle future such attacks. One of the strengths of this work is that, through a deep dive into the Mirai botnet, the researchers were able to show us why this botnet caused such chaos in the IoT security community and how such big events can occur in the future. But as the malwares are developed at an exponential pace and there are more IoT and DDoS capable malwares present, more thorough research can be done. This constitutes the researcher's main future works.

Research [7] basically focuses on investigating the existing Industrial Control Systems based malware found in the wild over the past decade through a comparative analysis. They present a comparative analysis framework of ICS-tailored malware in a bi-layered approach and evaluate the proposed method by experimenting with five well-known ICS malware: Stuxnet, Havex, BlackEnergy2, CrashOverride and TRISIS. In the Cyber Threat Intelligence Layer, they modeled the industrial malware using the ICS Cyber Kill Chain model. In the Hybrid Analysis Layer, they analyzed each ICS malware's main features and behaviors by employing static and dynamic analysis techniques. Through the research, they conclude that following best practices and learning from previous cyber attacks is the best defense against ICS-based malware attacks. One of the strengths of this research is that they propose a framework which is focused on ICS-based malware which isn't very researched upon as much as regular malwares. Future plans include using such case studies to develop rigorous theoretical models.

The paper [8], analyzes two popular RATs, namely njRAT and DarkComet. They report on the attackers and victims of these two malwares. For the analysis, all instances of these RATs are collected, then they determine the victims of these campaigns by identifying and registering the expired domains of the controllers and redirecting the traffic to their own measurement infrastructure. The results show that 99are not actual victims and they end the research by reporting on the number of victims, the activation period of the campaigns and the geographic relationship between the victims and the attackers. Future work suggests focusing on doing more similar research to generalize the defense mechanisms against malware threats.

The literature review highlights the evolving sophistication of malware and the limitations of traditional detection methods. Prior studies emphasize the importance of comparative analysis, behavioral profiling, and advanced detection mechanisms. Building on these insights, this research explores shared tactics across malware strains, addressing gaps to enhance understanding and improve defense strategies against emerging threats.

Chapter 3

Malware Sample Collection

In this chapter, malware samples for the respected malwares are collected. There are countless samples available online and most of them work similar. Some starts from javascript some from vbs script etc. We chose the latest malware samples available at the time of analysis.

3.1 XWORM

XWorm's payload can be delivered and propagated through various methods, including the use of Excel and PDF files. Another, less common method involves the use of JavaScript. For our analysis, we selected a JavaScript sample, which represents the initial phase of XWorm's execution process. This sample was retrieved from MalShare.

A review of XWorm's activity trends on ANY.RUN confirms that the malware remains highly active. It is currently sold on the dark web by a user known as "Xcoder" or "EvilCoder" for approximately $400. The malware is typically distributed via phishing emails that include PDF, DOC, or RTF attachments. These emails are often disguised as business-related communications, such as invoices, orders, or tax documents [11].

SHA256	5bc8b1a067ec4b487e88c2bb93728158633f4fdf22b111d5562cbb4ad3426d30
Build	JavaScript
Category	RAT (Remote Access Trojan)
Family	XWorm
Version	5.6

Table 3.1: XWorm Sample

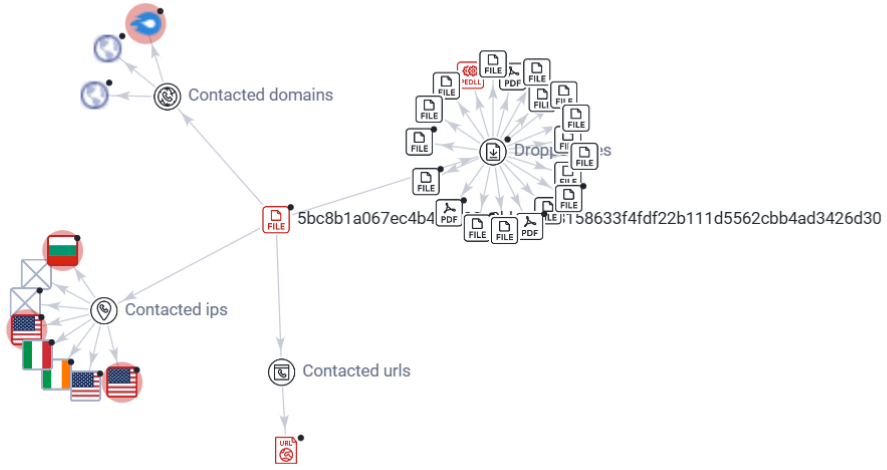

Figure 3.1: XWorm: Virustotal Graph

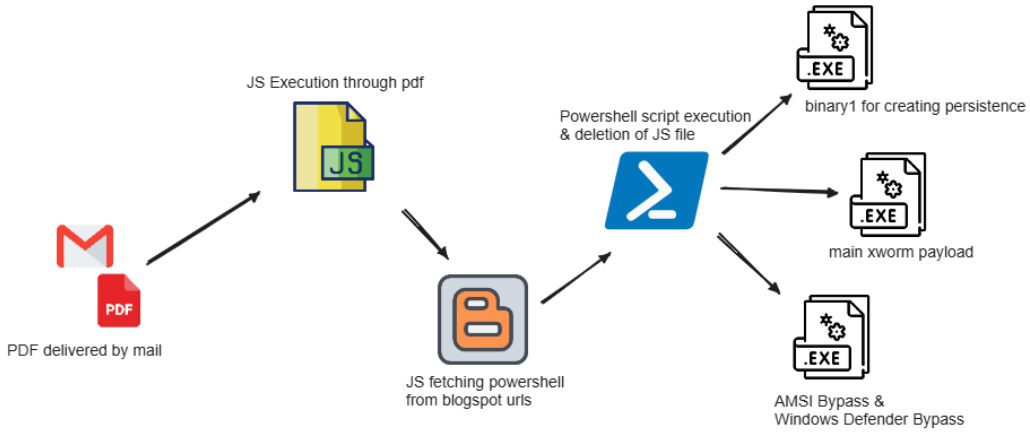

Figure 3.2: XWorm Life Cycle

3.2 Nanocore

Nanocore is one of the oldest players in the game. Even the primary author was sentenced it's still very active [12]. The RAT similar to xworm utilizes social engineering and phishing as the primary way of targeting the victims [13]. The Nanocore version originally sold on nanocore.io was cracked

As reported in [15], Nancore doesn't have any geographical preference, it targets victims from all countries possible.

The first stage malware sample has been taken for analysis which is an exe file. The malware sample has been downloaded from a github repo [16], but the sample is also available on malware bazaar and other databases with different hash.

SHA256	1605f0e74c7088b8a2ca7190b71c83f8dc0381e57d817df3530bda4ac5737511
Build	x86 and dotnet (multiple stages)
Category	RAT (Remote Access Trojan)
Family	NanoCore
Version	1.2.20

Table 3.2: NanoCore Sample

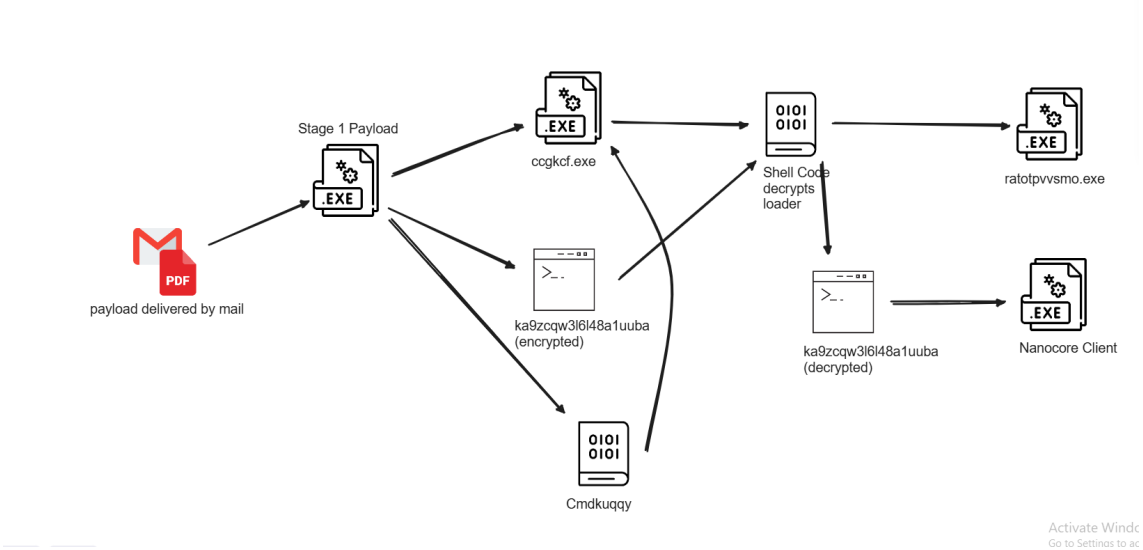

Figure 3.3: Nanocore Life Cycle

Access to malware samples has become relatively easier thanks to all the research done previously. Both of these malware samples are significantly different files. They are known as droppers as they are not the final payload instead they create the final payload through different stages. The main purpose of this is to add a layer of obfuscation to make detection and analysis harder as can be seen in the next chapter.

Chapter 4

Methodology

In the following chapter, we dive into the complicated delivery and execution methods of two malware samples. Initially, the proper environment is set up to confine and sandbox the samples so that they don't cause irreplaceable damage. Then, samples are collected from well-known malware databases. Most samples work the same and the final payload is the same. Only the delivery method differs from each other. Both static and dynamic analysis is performed on both these samples.

4.1 Environmental Setup

In this section, the environment used for the research is shown and a step by step instruction is provided. First step is to create a sandbox for the malware to run independently, so that it's capabilities can be observed. Below are instructions to create such dissecting laboratory for any kind of future malware analysis.

1. Setup VMWare or VirtualBox
2. Install Windows 10/11
3. Configure Windows 10/11 to not detect any suspicious software.
4. Disable windows update.
5. Disable secure script so that further powershell script running is easier.
6. Create a checkpoint in Virtual Machine software so that it's easy to restart in case anything goes wrong.
7. Download FlareVM powershell script and run it using administrator privilege. FlareVM is a all in one malware analysis windows operating system version. This basically installs all the necessary tools required for in-depth malware analysis. Although, not all tools used in the research is not available here but most are.
8. After the complete the installation, it's necessary to create another checkpoint as clean FlareVM install because during analysis of malware a lot of system

file and registry keys are affected that may give out wrong results for the next malware. So, it's best to start clean everytime. If necessary, multiple VM instaces can be used.

The tools that will be used for the analysis are,

- **x32dbg/x64dbg:** Used for analyzing and debugging assembly code.
- **IDAPro:** It's a disassembler perfect for disassembling .NET files for easier analysis and debugging.
- **ProcMon:** Short for process monitor, this software is good for monitoring system changes made by a process. The filter functionality is really helpful to get rid of all the clutters.
- **PEStudio:** Provides metadata information, detects signature, detects hard-coded urls, detets imports, exports, strings etc. for PE binary files.
- **HashMyFiles:** Quickly generates various hashes for multiple files.
- **Process Hacker:** Doesn't come builtin with FlareVM but this tool is very helpful for monitoring process and process injections. Also, the network traffic monitor is easy to understand and analyse.
- **CyberChef:** Known as the Cyber Swiss Army Knife, this tool is used for deobfuscation mostly in this analysis and also used to dump binary data to actual payloads.

4.2 XWORM

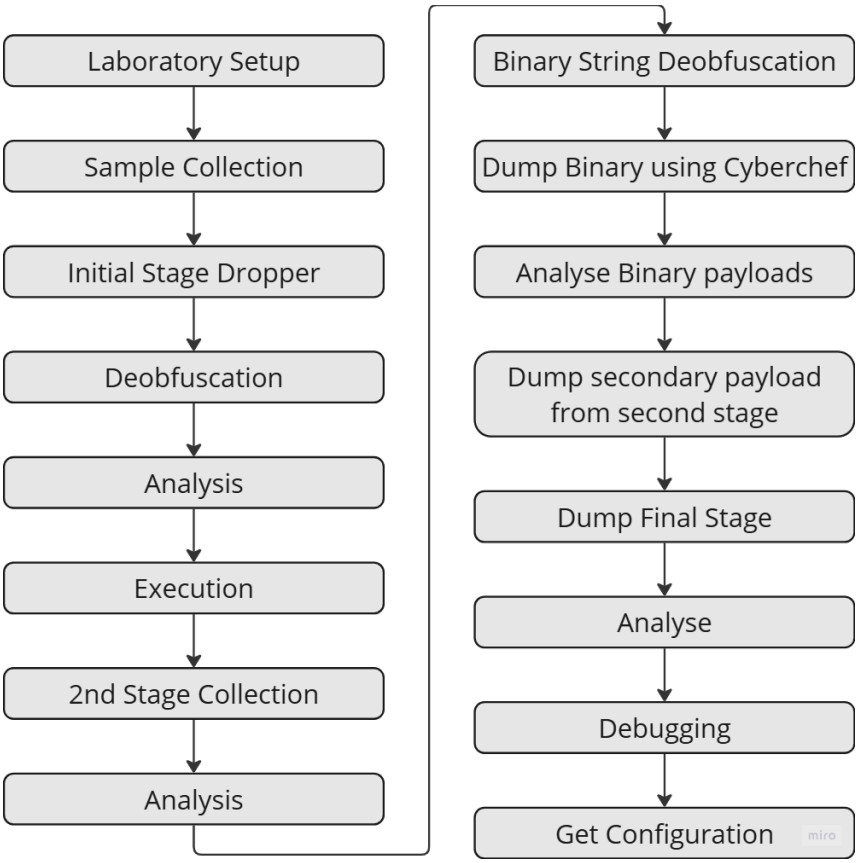

Figure 4.1: XWorm: Oversimplified Analysis Diagram

We start with a simple static analysis of the JavaScript code. We are greeted with somewhat obfuscated JavaScript. A very classic obfuscation method is used by using horrible naming conventions. The main point of this is to slow down the analysis. To de-obfuscate, usually, the names are fixed as we can see the variable names are unnecessarily large. Also, "Akut11" is repeated a lot of times, so removing it greatly shortens the variable name.

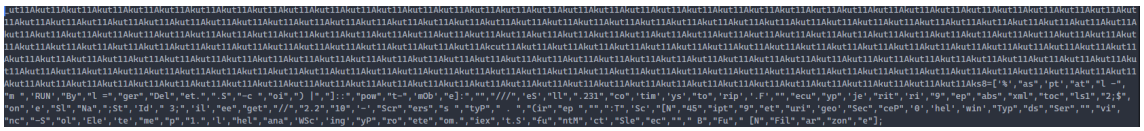

Figure 4.2: XWorm: Obfuscated JS Dropper

Then, what remains is just weird variable names which can easily be replaced with simpler names.

There are in total 4 arrays. Another obfuscation technique is used here. A string is split into an array and the final payload is created through the execution.

At first the string is spread and stored in array1. Then in array2, the elements from array1 are taken in different order and in some cases, some parts of array1 are concatenated and then inserted in the new array. array3 also does the exact same

thing as array2 but it's mostly concatenation. Finally, in array4, which is much shorter, takes values from array3 and stores them in different order and then in the next line the elements are rearranged for the final time. The final string/array is stored in array4. The string is most likely a PowerShell command and the parts of the command are directly used in the next lines. To de-obfuscate, we can simply run the first lines for creating array4 and see the value of array4.

Figure 4.3: XWorm: Deobfuscated

Several things are happening here,

- "atom.xml" file is downloaded from 91.92.245.231
- the script sleeps for 3 seconds most likely for the download to be complete.
- the file is executed using PowerShell.
- The current file is deleted.
- Followed by another 3 seconds sleep delay.

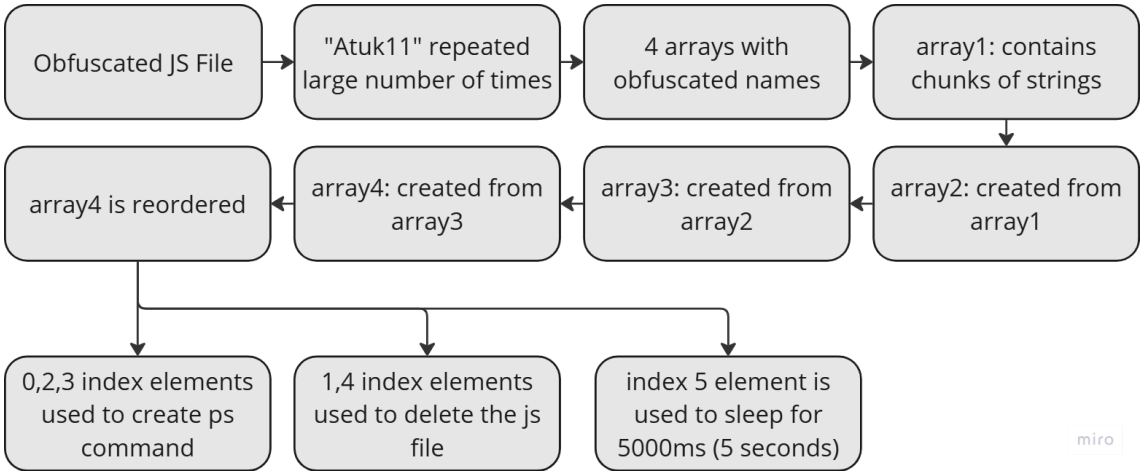

Figure 4.4: XWorm: Stage1 Control Flow Graph

The downloaded atom.xml is a powershell script. Upon analyzing the script a few things instantly catch the eye:

- Process Termination of processes "RegSvcs", "mshta", "wscript", "msbuild"
- If the previous file exists, remove it.

- Downloading a decoy PDF file.

Through analysis of the code, 3 binary arrays can be found inside the file. All of them are very minimally obfuscated with little parts of the binary array being replaced with characters like "-" etc. All of these are passed to a function, the main use of this function is to convert the strings to actual binary.

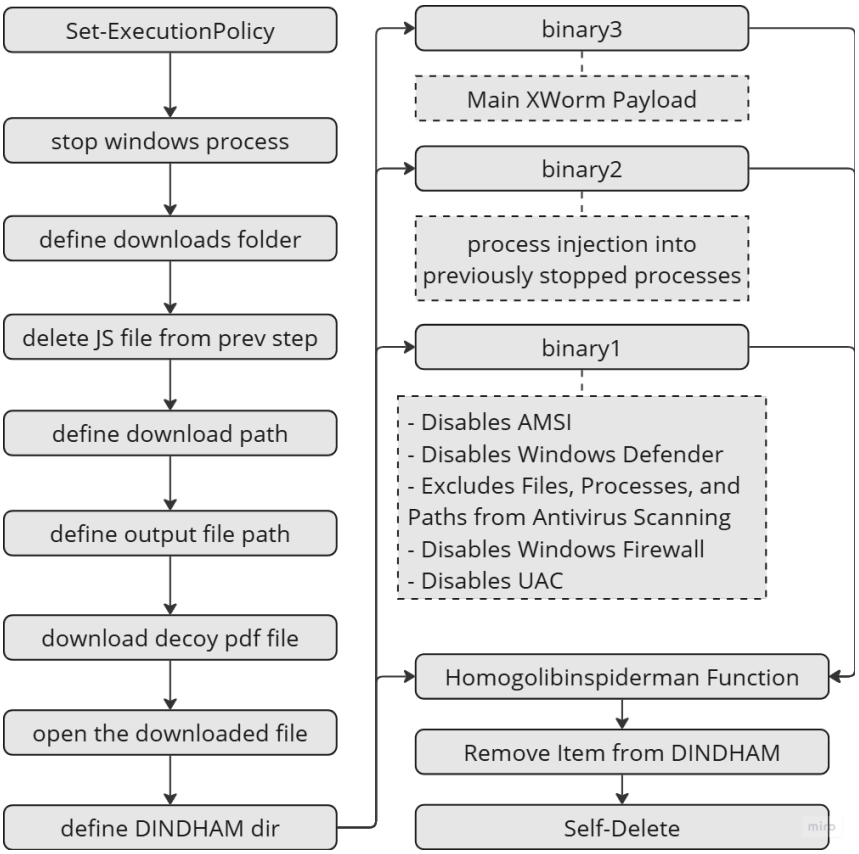

Figure 4.5: XWorm: Second Stage Execution Flow

Dumping the binaries and opening them using Detect It Easy, it can be seen that these are .NET files. The first file is the final payload. In the Main class, the first few lines decrypt encrypted configuration values using a key and sets them as the configuration for the payload.

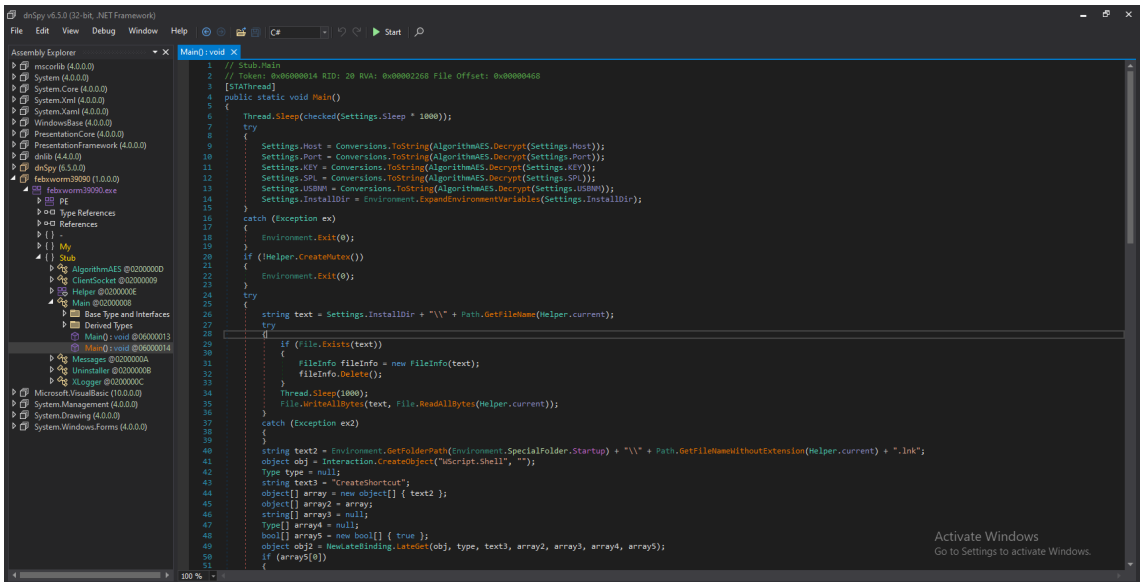

Figure 4.6: XWorm Final Payload

The values look base64 encoded but they are also encrypted using AES and the CipherMode is set to ECB (Electronic Code Block).

Algorithm 1 AlgorithmAES Class

function DECRYPT(*input*)
    Initialize $rijndaelManaged$ as RijndaelManaged
    Initialize $md5CryptoServiceProvider$ as MD5CryptoServiceProvider
    Define $array1$ as byte[32]
    Define $array2$ as $md5CryptoServiceProvider.ComputeHash(UTF8SB(Settings.Mutex))$
    Copy first 16 bytes of $array2$ to $array1$
    Copy last 16 bytes of $array2$ to $array1$
    Set $rijndaelManaged.Key$ to $array1$
    Set $rijndaelManaged.Mode$ to CipherMode.ECB
    Define $cryptoTransform$ as $rijndaelManaged.CreateDecryptor()$
    Define $array3$ as $Convert.FromBase64String(input)$
    return $UTF8S(cryptoTransform.TransformFinalBlock(array3, 0, array3.Length))$
end function
function UTF8SB(s)
    return $Encoding.UTF8.GetBytes(s)$
end function
function UTF8S(b)
    return $Encoding.UTF8.GetString(b)$
end function

The configuration settings for the malware are decrypted during runtime. So, we can obtain the decrypted values by dynamically analyzing the script. First, two breakpoints are added at the first and last variable. Then, we observe all the different values returned by stepping through the lines.

Here, we are able to extract the C2 server which uses DDNS service often used by

threat actors to hide the actual Command and Control Server. The port being used. Also, the key used for decryption.

Host	91.92.245.231
Port	1111
Key	<123456789>
SPL	<Xwormmm>
Mutex	IuFGHmRUIhJoJ1cR
USBNM	USB.exe
Installation Dir	`@"C:\Users\insecure\AppData\Roaming"`

Table 4.1: XWorm: Decrypted Final Stage Configuration

4.3 Nanocore

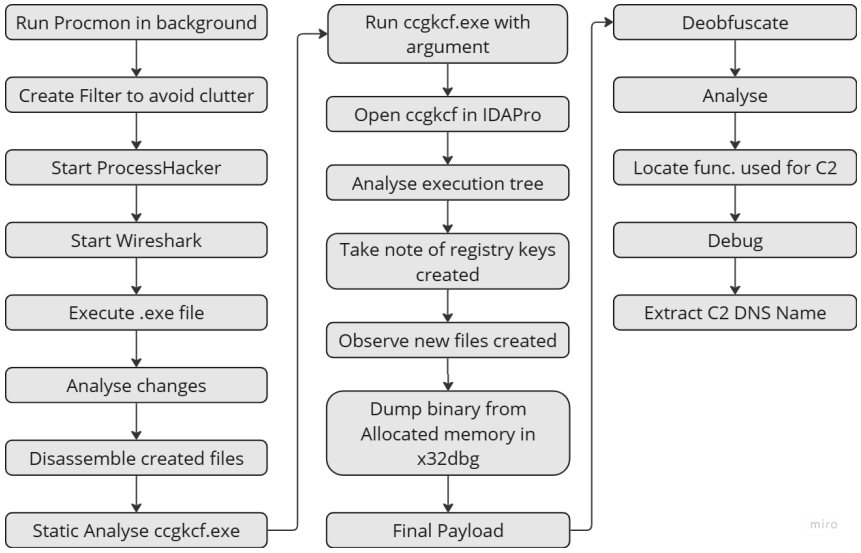

Figure 4.7: Nanocore: Oversimplified Analysis Diagram

The sample is a .exe file. Opening the file in PEStudio flags Strings and Import Functions.

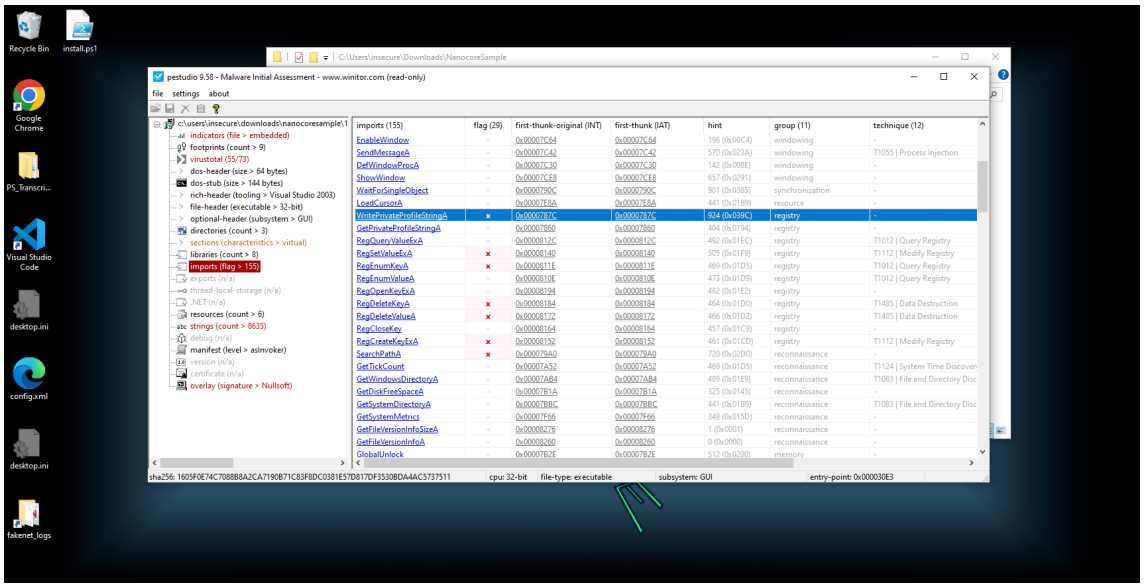

Figure 4.8: NanoCore PEStudio

For Dynamic Analysis, procmon and wireshark is kept running in the background along with process explorer in order to track different processes and network requests done by the file.

Upon execution of the payload, 3 files are created inside the %APPDATA%/Temp folder

1. ccgkcf.exe
2. ka9zcqw3l6l48a1uuba
3. Cmdkuqqy

Static analysis of the ccgkcf.exe file shows few things, first, the Cmdkuqqy file needs to be passed for successful execution of this path. If there's no arguments passed to this file, the file simply terminates itself.
Using IDA, we can observe that the file using different commands to take argument and read file size and finally the file.

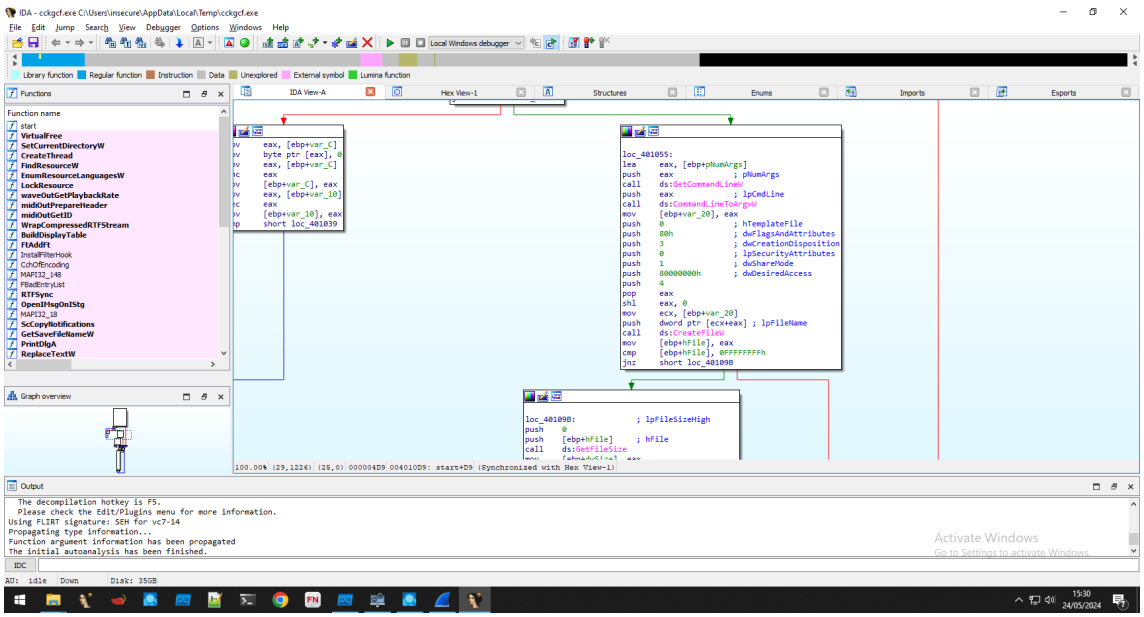

Figure 4.9: Nanocore: Second Stage Static Analysis 1

If the argument and the correct file are passed, the file continues execution.

Algorithm 2 File I/O Operations

```
1: function CreateFileAndWriteToIt(fileName, data)
2:     eax ← CreateFileW(addr(fileName), GENERIC_WRITE, 0, 0, CREATE_ALWAYS, 0, 0)
   ▷ Create the file at address [ebp+8]
3:     if eax ≠ INVALID_HANDLE_VALUE then
4:         ebx ← eax                    ▷ Store the file handle in ebx at address [ebp-4]
5:         ecx ← addr(data)             ▷ Get the address of the data at [ebp+12]
6:         edx ← strlen(data)           ▷ Get the length of the data at [ebp+12]
7:         WriteFile(ebx, ecx, edx, addr(bytesWritten), 0)    ▷ Write the data to the
   file
8:         CloseHandle(ebx)             ▷ Close the file handle at [ebp-4]
9:         return SUCCESS
10:    else
11:        return FAILURE
12:    end if
13: end function
14: function ReadFileAndPrintContents(fileName)
15:    eax ← CreateFileW(addr(fileName), GENERIC_READ, 0, 0, OPEN_EXISTING, 0, 0)
   ▷ Open the file at address [ebp+8]
16:    if eax ≠ INVALID_HANDLE_VALUE then
17:        ebx ← eax                    ▷ Store the file handle in ebx at address [ebp-4]
18:        ecx ← malloc(1024)  ▷ Allocate a buffer of 1024 bytes at address [ebp-8]
19:        edx ← 0        ▷ Initialize the read bytes counter to 0 at address [ebp-12]
20:        while ReadFile(ebx, ecx, 1024, addr(bytesRead), 0) ≠ 0 do
21:            printf("", ecx)     ▷ Print the contents of the buffer at address [ebp-8]
22:            edx += bytesRead ▷ Update the total bytes read at address [ebp-12]
23:        end while
24:        CloseHandle(ebx)             ▷ Close the file handle at address [ebp-4]
25:        free(ecx)                    ▷ Free the allocated buffer at address [ebp-8]
26:        return edx                   ▷ Return the total bytes read at address [ebp-12]
27:    else
28:        return FAILURE
29:    end if
30: end function
31: function Main
32:    fileName ← L"example.txt"        ▷ Set the file name at address [ebp+8]
33:    data ← "Hello, World!"           ▷ Set the data to write at address [ebp+12]
34:    if CreateFileAndWriteToIt(addr(fileName), addr(data)) = SUCCESS
   then
35:        printf("File created and written successfully.\n")
36:    else
37:        printf("Failed to create or write to the file.\n")
38:    end if
39:    fileName ← L"example.txt"        ▷ Set the file name at address [ebp+8]
40:    bytesRead ← ReadFileAndPrintContents(addr(fileName))
41:    if bytesRead ≠ FAILURE then
42:        printf("Read %d bytes from the file.\n", bytesRead)
43:    else
44:        printf("Failed to read the file.\n")
45:    end if
46: end function
```

Going back to IDA, we see, the payload decrypts the "Cmdkuqqy" file and completes the execution using shell code.

The shell code does several things. First, the shell code opens "ka9zcqw3l6l48a1uuba", reads the file and then decrypts the data from the file. Second, it creates a folder named gswccl at the AppData/Roaming directory and inside that directory it creates an exe named "ratotpvvsmo.exe". It also creates a new registry key "hhtktvn" with value set as the path to the "ratotpvvsmo.exe" file. The point of this is to create persistence.

Quick hash comparison shows that, the "ratotpvvsmo.exe" and the "ccgkcf.exe" file are exactly the same.

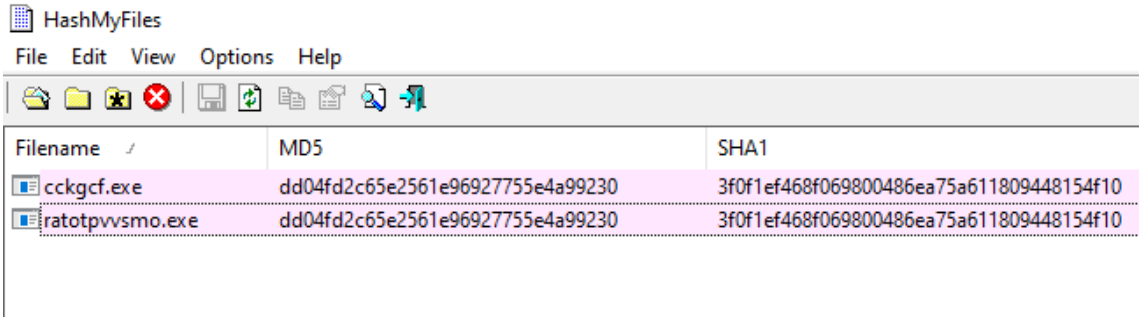

Figure 4.10: Nanocore: Hash Comparison

Next, the shell code again injects itself with the decrypted "ka9zcqw3l6l48a1uuba" file. This in turn starts another shell code which when dumped is the final payload (Nanocore Client).

Similar to XWorm and other RATs, NanoCore Client also shows obfuscated configuration settings which can be extracted using breakpoints like before.

Also, the malware tries to establish a connection to the command and control server during execution.

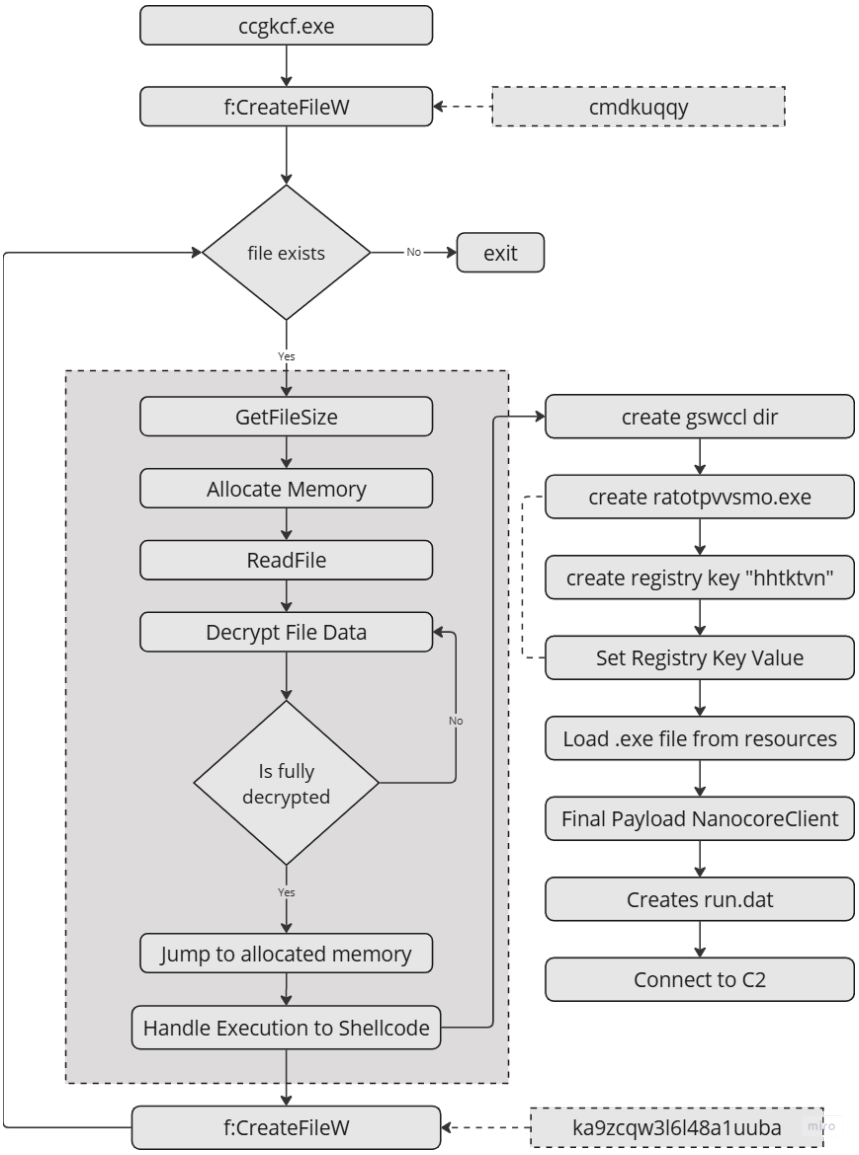

Figure 4.11: Nanocore Execution Flow Graph

4.4 Detection

Based on the analysis conducted in this study, Yara rules have been developed to classify and detect XWorm and NanoCore in real-world environments. YARA, a powerful tool in malware research and detection, utilizes a rule-based approach to define malware families through regular expressions, textual, and binary patterns. Each YARA rule consists of specific strings and logical expressions that help identify malicious behaviors and characteristics, providing an efficient method for detecting these malware strains in the wild.

Listing 4.1: YARA Rule for NanoCore

```
rule NanoCore {
  meta:
    description = "Detects Nanocore malware, including PE
       creation and process injection"
    date = "27-04-2023"
    author = "Shakil Islam Shanto, Afsana Mimi Punom"
    family = "Nanocore"
    version = "1.2.20"
  strings:
    $loader = "cckgcf.exe"
    $shellcode = "cmdkuqgy"
    $loadercopy = "ratotpvvsmo.exe"
    $regname = "hhtkvn"
    $foldername = "gswccl"
    $encNanoloader = "ka9zcqw316l48a1uuba"
    $nanocore = "NanoCore" nocase
    $pdbpath = "C:\\henws\\tzgjsa\\wusq\\17
       bb8d08751340c38ed51f5bd5e7430e\\vrsfIw\\beunbese\\
       Release\\beunbese.pdb"
  condition:
    $loader or $shellcode or $nanocore
}
```

YARA rules represent a valuable asset for the cybersecurity sector, aiding researchers, security experts, and incident response teams in the detection and management of threats associated with XWorm and NanoCore. By integrating these rules into malware detection frameworks, organizations can markedly enhance their ability to identify these specific malware variants in real-time. Furthermore, the flexible design of YARA allows for the development of new rules tailored to emerging malware strains that display analogous characteristics, thus strengthening proactive defense measures.

Listing 4.2: YARA Rule for XWorm

```
rule XWorm_Malware_Detection
{
    meta:
        description = "Detects XWorm malware, including AMSI
           bypass, PowerShell script behavior, and process
           injection"
        author = "Shakil Islam Shanto, Afsana Mimi Punom"
        date = "11-04-2023"
        family = "XWorm"
        version = "5.6"
        reference = "XWorm Malware Analysis"

    strings:
        $amsi_bypass = 'AMSIReaper' ascii
        $amsi_function = 'WriteProcessMemory' ascii
        $amsi_powershell = 'Add-MpPreference' ascii
        $amsi_reg_key = 'HKCU:\Software\Classes\CLSID\{
           fdb00e52-a214-4aa1-8fba-4357bb0072ec}' ascii

        $set_execution_policy = "Set-ExecutionPolicy -Scope
           CurrentUser -ExecutionPolicy Bypass -Force" ascii
        $stop_process = "Stop-Process -Name" ascii
        $remove_item = "Remove-Item -Path" ascii
        $output_file = "James_Tax_2023.pdf" ascii
        $download_url = "https://www.mediafire.com/
           file_premium/37z74w3icn8avhf/2022
           _TAX_RETURN_DOCUMENTS_%2528DAVILA_JONIQUE_L%2529-
           SIGNED.PDF/file" ascii

        $decoy_file = "atom.xml" ascii
        $c2_server = "91.92.245.231" ascii
        $c2_port = "1111" ascii
        $suspect_registry = "HKCU:\Software\Classes\CLSID\{
           fdb00e52-a214-4aa1-8fba-4357bb0072ec}" ascii

    condition:
        $amsi_bypass and $amsi_function or
        $amsi_reg_key or
        $set_execution_policy and $stop_process and
           $remove_item or
        $output_file and $download_url or
        $decoy_file or
        $c2_server and $c2_port or
        $suspect_registry
}
```

The Malware Analysis chapter took XWorm and NanoCore as case studies to study the details. This study brought their complex structure, behavior and how they avoid detection to the light. We found important attack strategies, persistence tricks and hiding techniques and based on the findings yara rules have been created. But

through the complexities of the execution flow, it can be understood that the yara rules may not be enough. They may become irrelevant, with slight change may be the pattern. They also cannot detect memory changes, registry changes, network traffic discrepancies etc. They are not standolone solution, rather complements the dynamic defenses. The use case is still there, so the analysis is easier. As if mere detection framework doesn't stand a chance, this is just a proof that a simple program can be so complicated.

Chapter 5

Comparative Analysis

Based on the findings of the previous section, this sections builds upon them and shows how even the not a single line is same for two malwares. They try to achieve the same goal. Also, this section shows how one single goal can be achieved in so many ways with necessary evidence and then this section concludes with discussion in relevance to the goal of the paper.

5.1 Delivery and Distribution Methods

Both NanoCore and XWorm share social engineering as a primary delivery method, leveraging phishing emails and spam campaigns to trick victims into downloading malicious attachments. The use of social engineering highlights the reliance on human error for initial infection, a common trait in both malware families.

- **NanoCore**: Delivered through phishing emails disguised as price catalogs, fake invoices, and shipment information. Attachments are often encrypted ZIP files, ISO images, or malicious MS Office documents to evade endpoint security.
- **XWorm**: Similarly delivered via phishing emails with business-related themes such as invoices, purchase orders, or tax information. It also uses media-sharing platforms like Mediafire to distribute password-protected RAR files, adding another layer of obfuscation.

Generalized Defense

A robust anti-phishing strategy combined with email filtering and attachment analysis can help prevent both malware strains from reaching end-users. Automated systems that detect unusual file types (e.g., encrypted ZIPs, ISO files) or media-sharing links in emails can be a key defense mechanism.

5.2 Cloud-Based Payload Hosting

Both NanoCore and XWorm take advantage of **cloud services** to host additional malicious payloads, making it easier to deliver larger, more complex attacks.

- **NanoCore**: Hosts additional payloads on platforms like AWS EC2, exploiting reliable infrastructure to store malicious files.
- **XWorm**: Uses Blogspot URLs and Cloudflare tunnels to distribute payloads, making its activity more difficult to trace and block.

Generalized Defense

By monitoring outbound traffic to known cloud service platforms and detecting unauthorized file downloads from unusual sources, a detection mechanism can be implemented to block malicious payloads from cloud-based services.

5.3 Persistence Mechanisms

Both NanoCore and XWorm employ persistence techniques to ensure their continued operation after a system reboot. NanoCore adds itself to the Windows Registry, while XWorm uses Task Scheduler to achieve persistence. Despite the different methods, the goal is the same: maintain control over the infected system.

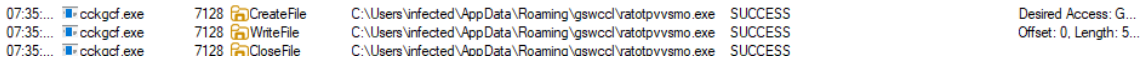

Figure 5.1: Nanocore: Creation of startup file in procmon

Figure 5.2: Nanocore: Creation of registry key

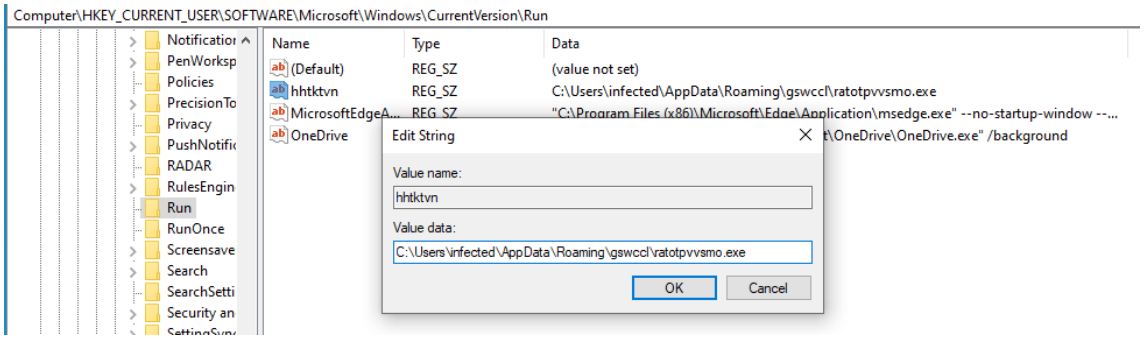

Figure 5.3: Nanocore: registry value set as created file

```
New-Item -P "HKCU:\Software\Classes\CLSID\" -N "{fdb00e52-a214-4aa1-8fba-4357bb0072ec}" -F
New-Item -P "HKCU:\Software\Classes\CLSID\{fdb00e52-a214-4aa1-8fba-4357bb0072ec}\" -N "InProcServer32" -F
New-ItemProperty -Path 'HKCU:\Software\Classes\CLSID\{fdb00e52-a214-4aa1-8fba-4357bb0072ec}\InProcServer32' -N '(Default)' -V "C:\IDontExist.dll" -PropertyType String -Force
```

Figure 5.4: XWorm: Creating Registry Key with dummy value

Generalized Defense

A unified detection approach could focus on identifying unusual additions to startup locations like the Windows Registry or scheduled tasks, flagging any unauthorized changes across both malware types.

5.4 Process Injection

Both malware strains utilize process injection techniques, where they inject malicious code into legitimate processes. This technique is designed to hide the malware within trusted system operations, allowing it to evade detection by security tools. NanoCore and XWorm both leverage this method, making process injection a critical shared feature.

Algorithm 3 Code used for process injection in XWorm

```
1: DATA_1 ← kimkarden decoded bytes
2: DATA_2 ← kimkarden decoded bytes
3: procedure ExecuteCommand
4:     ASSEMBLY ← Load(DATA_1)
5:     TYPE_NAME ← 'A.B'
6:     METHOD_NAME ← 'C'
7:     TYPE ← ASSEMBLY.GetType(TYPE_NAME)
8:     INVOKE_METHOD ← TYPE.GetMethod(METHOD_NAME)
9:     FRAMEWORK_PATHS ← [
   'C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegSvcs.exe',
   'C:\Windows\Microsoft.NET\Framework\v2.0.50727\RegSvcs.exe',
   'C:\Windows\Microsoft.NET\Framework\v3.5\Msbuild.exe'
   $]
10:    Sleep(1)
11:    for path ∈ FRAMEWORK_PATHS do
12:        Sleep(2)
13:        INVOKE_METHOD(NULL_ARRAY, (path, DATA_2))
14:    end for
15: end procedure
```

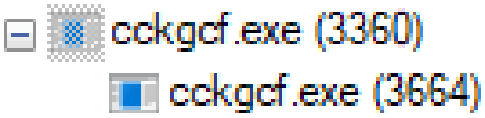

Figure 5.5: Nanocore: Process Injection in process list

Generalized Defense

By focusing on identifying abnormal memory usage or unexpected behavior within legitimate processes, a detection system can be implemented that monitors process

integrity, thus helping to detect a broad range of malware using process injection techniques.

5.5 Command-and-Control (C2) Communication

NanoCore and XWorm both rely on communication with a C2 server to exfiltrate data and receive commands. Both use dynamic DNS services to frequently change their C2 domains, further complicating detection. This reliance on external communication is another key similarity between the two malware strains.

During analysis, it could be seen that XWorm was trying to connect with it's command control server which was '91.92.245.231'. Also, during analysis of Nanocore final payload, it was discover that a function was trying to establish connection with a c2c server which was using a dynamic dns, which was 'stonecold.ddns.net'.

Generalized Defense

Implementing network traffic analysis tools that flag connections to dynamic DNS services or abnormal IP address resolutions can provide a defense mechanism capable of detecting C2 communication across different malware strains.

5.6 Obfuscation Techniques

Both NanoCore and XWorm employ obfuscation to hide their true nature, making it difficult for antivirus software to detect them. They use various techniques to mask their code, filenames, and processes. This common strategy is critical for bypassing traditional detection methods.

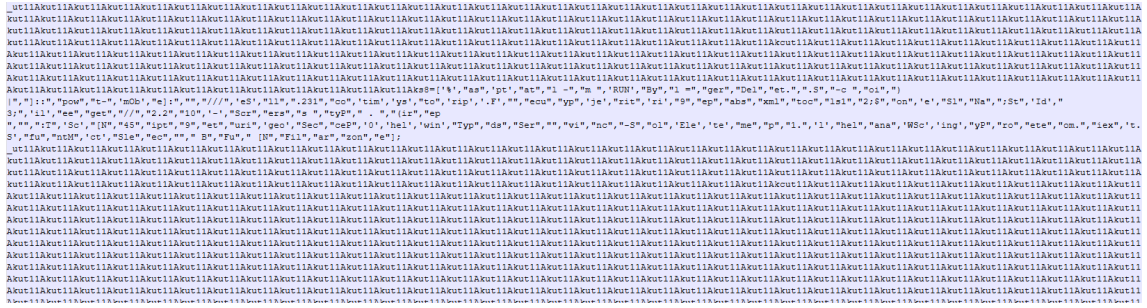

Figure 5.6: XWorm: Obfuscated Stage1 Payload

```
#AM
$Phudigum = "01011011010^^1^10101^1^10-0^10-*101*001*^-101^11*00-110^1*1*1**01011010

(Homogolibinspiderman $Phudigum) | .('{1}{Â°Â°Â°}'.replace('Â°Â°Â°','0')-f'!','I').re

(Homogolibinspiderman $bulgumchupitum)  | .('{1}{Â°Â°Â°Â°Â°Â°Â°Â°Â°Â°Â°Â°Â°Â°Â°Â°Â°
Remove-Item -Path "$DINDHAM\KAMASUTRAKIM.~!!@#!!!!!!!!!!!!!!!!~", "$DINDHAM" -Recurse

'@
[IO.File]::WriteAllText("$DINDHAM\\KAMASUTRAKIM.~!!@#!!!!!!!!!!!!!!!!~", $lulli)

$lulli | .('{1}{Â°Â°Â°Â°Â°}'.replace('Â°Â°Â°Â°Â°','0')-f'!','I').replace('!','ex')
```

Figure 5.7: XWorm: Obfuscated Stage2 Payload

```
[StandardModule]
internal sealed class #=q48p8EJcbwRuSJ9efJfzTZ7uyOBVlFQpnFVv30w93EJA=
{
    // Token: 0x0600001A RID: 26 RVA: 0x00002544 File Offset: 0x00000744
    public static void #=q0zLeEY98tybLc8FS6iVEWjGp4MNZxETphcH7ohzBXuY=(#=qxe_BfLLMHqYa_KBeLsRfpw== #=qW1Ty88c
    {
        switch (#=qW1Ty88cS3yMuRwgBrH3qpw==.#=qKi0KrAcAGUOMcS5S$2tJyg==)
        {
        case 0:
            #=q48p8EJcbwRuSJ9efJfzTZ7uyOBVlFQpnFVv30w93EJA=.#=qVEEdpD96A48uRzPJT7G_w60gIZo4tH1_e21GoRWPFm8=
              (#=qW1Ty88cS3yMuRwgBrH3qpw==.#=qL_Q_RdUm_wJ7VeVwUqRXbA==);
            break;
        case 1:
            #=q48p8EJcbwRuSJ9efJfzTZ7uyOBVlFQpnFVv30w93EJA=.#=q$sTc1AZMnHRC7q_PL2hWs4JIEJoo88_IAFcWtrdNt$4=
              (#=qW1Ty88cS3yMuRwgBrH3qpw==.#=qL_Q_RdUm_wJ7VeVwUqRXbA==);
            break;
        case 2:
            #=q48p8EJcbwRuSJ9efJfzTZ7uyOBVlFQpnFVv30w93EJA=.#=qNsyg$dsR$GJkSvK2TftGTNPuC8S809j_UmmfNnXTTOo=
              (#=qW1Ty88cS3yMuRwgBrH3qpw==.#=qL_Q_RdUm_wJ7VeVwUqRXbA==);
```

Figure 5.8: Nanocore: Obfuscated Final Stage Payload

Generalized Defense

A defense mechanism focusing on detecting obfuscated scripts, unusual file names, or encrypted configuration data at the initial stages of malware execution could serve as a broad-spectrum approach to detecting malware that relies on obfuscation.

5.7 Phishing as an Initial Vector

Phishing is a common initial attack vector for both NanoCore and XWorm. Both strains typically enter the victim's system through malicious email attachments or links. Although the file types and payloads differ, the reliance on social engineering tactics is consistent across both.

Generalized Defense

Anti-phishing measures, such as email filtering and user education, can help block a significant portion of these attacks at their earliest stage, preventing malware from

ever entering the system.

5.8 Analysis

This section interprets the findings from the comparative analysis, focusing on the broader implications for detection, prevention, and understanding of malware behavior.

Both malwares leverage on social engineering but differ in distribution complexity. Both use obfuscation which has become standard nowadays for malwares. XWorm obfuscates the powershell payloads and Nanocore uses dynamic techniques to obfuscate and bypass detection.

Similarities in persistence mechanisms suggest that monitoring common system modification points such as registry keys can detect both malware types. But, differences in obfuscation approaches indicate that defense tools must be adaptable to both static and dynamic obfuscation.

Since both malware families rely on C2C communication, a focus on detecting C2 patterns, such as frequent DNS queries to dynamic DNS services, could thwart multiple threats. Understanding process injection as a shared tactic highlights the need for improved memory and process monitoring solutions.

The reason current defenses mostly fail are still the same, Dynamic DNS services make C2 servers elusive. Fileless payloads bypass traditional antivirus solutions.

	Nanocore	XWorm
Obfuscation	Yes	Yes
Persistence	Yes	Yes
Mailspam	Yes	Yes
C2C	Yes	Yes
Process Injection	Yes	Yes

Table 5.1: Tactics used in XWorm & Nanocore

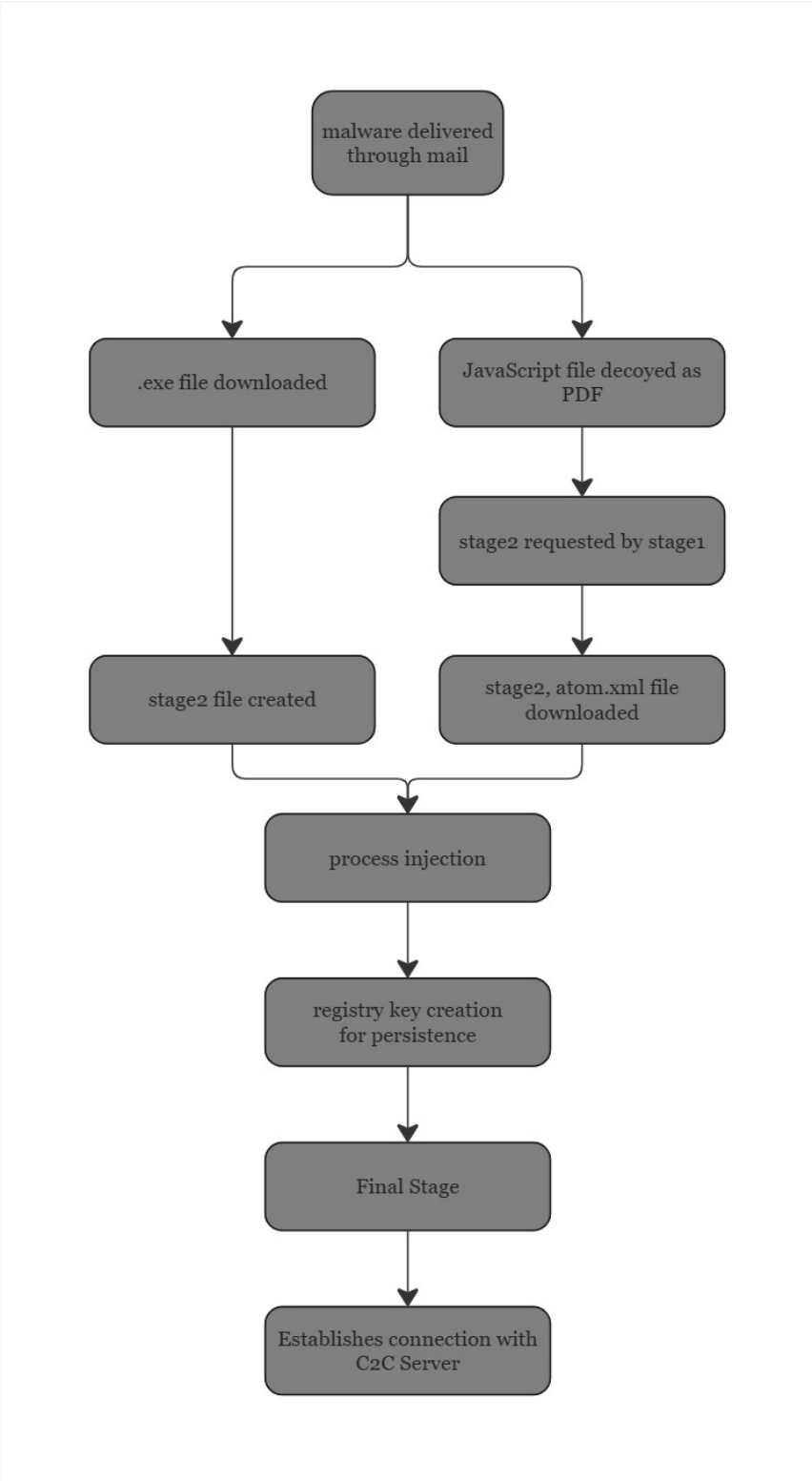

Figure 5.9: similarity workflow graph

This study identifies shared techniques between the advanced malware strains NanoCore and XWORM, offering critical insights into their evasion tactics and attack strategies, which can guide future defense systems by focusing on broader malware families instead of isolated cases. By addressing pressing challenges such as the rise of fileless malware and obfuscation techniques that render traditional

detection methods ineffective, it highlights the persistent threat posed by these strains, with real-world examples like ransomware attacks and crypto thefts. Serving as foundational work, the research provides a blueprint for developing automated detection tools by identifying common features such as persistence mechanisms, obfuscation layers, and C2 communication, which can serve as labeled inputs for training machine learning models to recognize malware behavior patterns. For instance, a supervised learning algorithm could be trained on datasets that include the shared characteristics highlighted in this study to predict new malware instances or classify families. Its interdisciplinary relevance is evident, as researchers can leverage these findings to create such predictive models, industry professionals can integrate behavior-based detection systems into SOC protocols, and educators can use them to teach malware behavior and countermeasures. Additionally, the layered approach of static, dynamic, and reverse engineering analysis offers a replicable methodology and theoretical contributions, particularly in understanding shared persistence mechanisms and process injection techniques, which could be encoded into algorithms for anomaly detection. By generalizing these findings, the results are applicable to a wide range of malware, offering actionable insights for educational use and fostering collaboration between academia and industry to enhance defense systems.

Chapter 6

Future Plan

This study lays the groundwork for the development of resilient and flexible malware detection systems. By gaining insights on complex techniques and their associated sub-techniques, including the diverse methods of process injection, it's clear that the obvious next step is development of a robust framework that will not lose it's capability as malware's evolve. Creating defense rules tailored towards a single malware strain or family is not optimal not even against the same malware strain, as they evolve. But findings through comparative analysis will help apply the defense target broader malware numbers. This level of adaptability will retain that the framework retains its efficacy as malware evolves.

Moreover, utilizing insights derived from comprehensive and comparative analyses allows for the development of superior datasets that are enhanced through effective feature engineering. These enriched datasets are essential for increasing the accuracy and efficiency of AI-driven malware detection systems. Specifically, the profound understanding achieved through manual analysis facilitates the identification of critical features that can significantly improve model training and overall performance. Datasets abundant in features not only bolster detection capabilities but also support the advancement of explainable AI within the domain. The clarity provided by explainable AI guarantees that these systems are both understandable and actionable, thereby enhancing their reliability and broader applicability.

This research sets the stage for future advancements in malware detection systems, encouraging a move towards approaches that are flexible, informative, and capable of tackling the challenges from constantly changing malware threats.

Chapter 7

Conclusion

This study answer the research question by revealing common strategies used by XWorm and NanoCore, including obfuscation, process injection, persistence methods, and Command-and-Control (C2) communication. These results enhance our understanding of their actions and point out weaknesses in current detection methods that often depend on static, signature-based systems. The motivation for the research stemmed from the ongoing threats and importance of XWorm and NanoCore, which highlight the flaws of current defenses. The ability to adapt and evade detection brings into focus the necessity of more flexible and behaviour informed detection systems. While there are well known types of these malware, each of these malware types still evolves to integrate new delivery techniques and new attack methods. It already uncovered an inadequacy of the existing tools, and the need for deeper analysis. The challenge of these types of advanced malware families which evade traditional detection through fileless payload and encryption is what the research problem is. Our own current systems simply aren't adequate to keep up with all these changing threats, leaving us with a huge weakness in our defenses. The difficulty was addressed by this study through detailed manual analysis and comparison methods. The research tasks were to review the structure and functionality of XWorm and NanoCore, identify common points and differences between the two, expose weaknesses of existing systems, and find practical ways of refitting. These were achieved by way of painstaking manual analysis, generating YARA rules and the idea of a detection framework which extends beyond being just static signatures. In addition to closing some immediate gaps, this research establishes a foundation for future advancements in AI based and behavior focused malware detection. This work can open many further avenues for research looking forward. This study provides a strong foundation for further malware detection with a solid framework based on in depth analysis. First of all, using the groundwork established here will lead to a safer digital environment towards scalable, flexible, and clear defense strategies. Finally, we conclude with our findings suggesting the need to develop more efficient and current understanding and detection methodology. The findings and contributions of this research will be of great importance as the phenomena of cyber threats have become more complex.

Bibliography

1. Geiger, Marcus et al. “An Analysis of Black Energy 3, Crashoverride, and Trisis, Three Malware Approaches Targeting Operational Technology Systems.” 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) 1 (2020): 1537-1543.

2. Alrawi, Omar et al. “The Circle Of Life: A Large-Scale Study of The IoT Malware Lifecycle.” USENIX Security Symposium (2021).

3. Chavan, Neeraj et al. “A Comparative Analysis of Android Malware.” International Conference on Information Systems Security and Privacy (2019).

4. Kiss, Nicolas et al. “Kharon dataset: Android malware under a microscope.” (2016).

5. Kihiu, S., and E. Abade. “Comparative Analysis of Distinctive Features of the Ransomware Tactics in Relation to Other Malware.” International Journal of Computer (IJC), vol. 38, no. 1, July 2020, pp. 173-82, https://www.ijcjournal.org/index.php/InternationalJournalOfComputer/article/view/1671.

6. Donno, Michele De et al. “DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation.” Secur. Commun. Networks 2018 (2018): 7178164:1-7178164:30.

7. Mekdad, Yassine, et al. “The Rise of ICS Malware: A Comparative Analysis.” Computer Security. ESORICS 2021 International Workshops, Springer International Publishing, 2022, pp. 496–511. Crossref, https://doi.org/10.1007/978-3-030-95484-0_29.

8. Rezaeirad, Mohammad et al. “Schrödinger’s RAT: Profiling the Stakeholders in the Remote Access Trojan Ecosystem.” USENIX Security Symposium (2018).

9. “What Is Malware Analysis?” Fortinet, https://www.fortinet.com/resources/cyberglossary/malware-analysis.

10. “What is Malware Analysis?” VMWare, https://www.vmware.com/content/vmware/vmware-published-sites/us/topics/glossary/content/malware-analysis.html.html.

11. https://www.trellix.com/blogs/research/old-loader-new-threat-exploring-xworm/

12. https://www.fbi.gov/news/stories/malware-creator-sentenced-070518

13. https://www.blackberry.com/us/en/solutions/endpoint-security/ransomware-protection/nanocore

14. https://www.cyber.nj.gov/threat-landscape/malware/trojans/nanocore

15. Endpoint Protection - Symantec Enterprise. (2015). Broadcom.com.

16. https://github.com/shaddy43/MalwareAnalysisSeries/blob/main/NanoCore1.2.2.0/sample/1605f0e74c7088b8a2ca7190b71c83f8dc0381e57d817df3530bda4ac5737511.rar

17. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5461132/

18. Lucca Runger-Field. (2024, October 22). Destructive Code: The Most Infamous Malware Attacks in History. PIA VPN Blog. https://www.privateinternetaccess.com/blog/worst-malware-in-history/

19. Greenberg, A. (2018, August 22). The untold story of NotPetya, the most devastating cyberattack in history. WIRED; Condé Nast. https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/

20. "Top 10 Malware Q1 2024." *CIS*, www.cisecurity.org/insights/blog/top-10-malware-q1-2024.

21. gmcdouga. (2023, April 10). *March 2023's Most Wanted Malware: New Emotet Campaign Bypasses Microsoft Blocks to Distribute Malicious On ...* Check Point Blog; Check Point Software. https://blog.checkpoint.com/security/march-2023s-most-wanted-malware-new-emotet-campaign-bypasses-microsoft-blocks-to-distribute-malicious-onenote-files/

22. *Malware Creator Sentenced*. (n.d.). Federal Bureau of Investigation. https://www.fbi.gov/news/stories/malware-creator-sentenced-070518

23. May 18, D. W. on, & 2020. (2020, May 18). *NanoCore RAT — Malware of the Month, May 2020*. Security Boulevard. https://securityboulevard.com/2020/05/nanocore-rat-malware-of-the-month-may-2020

24. *Health Sector Cybersecurity Coordination Center (HC3) Sector Note Remote Access Trojan Nanocore Poses Risk to HPH Sector*. (2020). https://www.hhs.gov/sites/default/files/remote-access-trojan-nanocore-poses-risk-hph-sector.pdf

25. Cofense. (2024). *Ransomware in 2024: Top 5 Delivery Methods and Threats to Know*. Cofense.com.

26. Veriti Research. (2024, August 27). *Streamers in the Crosshairs: How XWorm Malware Targets Online Content Creators*. VERITI. https://veriti.ai/blog/streamers-in-the-crosshairs-how-xworm-malware-targets-online-content-creators

27. Ravie Lakshmanan (2023, May 12) . *XWorm Malware Exploits Follina Vulnerability in New Wave of Attacks*. The Hacker News. https://thehackernews.com/2023/05/xworm-malware-exploits-follina.html

28. *MalwareBazaar — Xworm*. (2022). Abuse.ch. https://bazaar.abuse.ch/browse/signature/Xworm

29. *MalwareBazaar — NanoCore*. (2020). Abuse.ch https://bazaar.abuse.ch/browse/signature/NanoCore

30. *Nanocore — Malware Trends Tracker*. (2022, April 30). Nanocore — Malware Trends Tracker. https://any.run/malware-trends/nanocore

31. ANY.RUN. (2023, August 29). *XWorm — Malware Trends Tracker*. XWorm — Malware Trends Tracker. https://any.run/malware-trends/xworm

32. Venter, C., Mikhaeil, C. A., & Ziegelmayer, J. L. (2024, November 24). *Major cybercrime crackdowns signal shift in global cybersecurity strategies*. The Conversation.