International Journal for Electronic Crime Investigation

ISSN: 2522-3429 (Print)
ISSN: 2616-6003 (Online)

DOI: https://doi.org/10.54692/ijeci.2023.0803200

Vol. 8 issue 3 Jul-Sep 2024

Advanced Techniques of Malware Evasion and Bypass in the Age of Antivirus

Kausar Parveen and Kinza Batool
Department of Computer Sciences, University of Engineering and Technology, Lahore

kbatool5121472@gmail.com
Received: Jul 06, 2024; **Accepted:** Jul 29, 2024; **Published:** Sep 12, 2024

ABSTRACT

The use of antivirus software as the main line of protection against growing cyber threats highlights the necessity of comprehending and resolving its limits. This study provides light on the ease of use and accessibility of tools used by hackers by carefully examining the complex terrain of malware evasion and bypass tactics. The persistent evolution of malware evasion and bypass techniques presents a significant cybersecurity challenge. The main objective is to educate users about the ever-changing hazards and provide them with the knowledge they need to properly strengthen their digital defenses. The literature analysis highlights the necessity for continued attention by establishing a strong correlation between the effectiveness of evasion strategies and their age and popularity. While modern antivirus software shows strong resistance against a range of tried-and-true techniques when updated on a regular basis, the study reveals a crucial component in its testing. This entails applying simple yet effective tweaks to well-known evasion techniques, demonstrating their capacity to fool even the most recent antivirus software. A thorough examination of malware evasion tactics, including both on-desk and in-memory approaches, is given in the methods section. Packing, obfuscators, protectors, reflective DLL injection, remote process memory injection, process hollowing, and inline hooking are all covered in detail in this paper. Subsequently, the study delves deeper into distinct evasion strategies, such defensive evasion through direct system calls and sophisticated evasion tactics, showcasing malware developers' versatility in evading antivirus and endpoint detection and response (EDR) systems.

Keywords: Malware evasion, Malware bypass, Cybersecurity, antivirus, evasion.

1. INTRODUCTION

1.1. Opening Section

The widespread usage of the internet by many sectors of the population has made communication, entertainment, and information retrieval more convenient. But users that take advantage of this accessibility run the risk of being hacked by malicious software that compromises user privacy and sensitive data. As a major protection mechanism against cyberattacks, people frequently resort to antivirus software in reaction to this digital terrain. [1] Selecting trustworthy sources is crucial since downloading data from unknown sources can result in viruses, even with the widespread use of popular software like web browsers. Users can delete or clear suspicious files from quarantine by using antivirus applications, which are essential for alerting users about them. [2] The dependability of these defensive technologies depends on user confidence, which means that the antivirus program and its signature database need to be updated on a regular basis.

1.2. Background of the Research

Hackers are a constant danger to cybersecurity because they use a variety of evasion and bypassing techniques to access equipment without authorization. In order to meet this challenge, antivirus software providers are always creating new protection methods and upgrading signature databases. [3] On the other hand, the ongoing appearance of new viruses raises the possibility that the defenses in place now could not always be enough. By exploring the intricate world of malware evasion and bypass techniques, this study sheds insight on how cyber threats are changing and highlights the need for creative solutions to strengthen digital defenses.

1.3. Statement of the Problem

The increasing complexity of malware evasion and bypass techniques poses a significant cybersecurity concern in the age of sophisticated antivirus programs. Concerns over the effectiveness of present defensive systems are raised by the rising number of new infections, and in spite of antivirus software vendors' constant attempts to create strong protection measures. The goal of this study is to thoroughly investigate the limitations of antivirus software, with a particular emphasis on the accessibility and usability of tools used by hackers. The goal of the research is to improve digital defense techniques by providing useful insights by comprehending and overcoming these constraints.

1.4. Rationale

The understanding of the dynamic nature of cyberthreats and the requirement for a proactive approach to cybersecurity serve as the foundation for this study. It is impossible to exaggerate the significance of having strong antivirus software in light of people' growing reliance on digital platforms. Through investigating the always changing strategies for evading and bypassing malware, [4] this study seeks to support continuous endeavors to fortify digital defenses. The results of the study will provide useful information for antivirus software manufacturers as well as users, resulting in a more robust cybersecurity environment.

1.5. Scope of the Study

This study is important because it may help users learn about the constantly evolving risks posed by cyberattacks

and provide them with the information, they need to strengthen their online defenses. Researchers have established a relationship between the popularity and age of evasion tactics and their efficiency, which is useful information for antivirus software producers as well as consumers. [5] The study's conclusions will further the current cybersecurity conversation by encouraging a more knowledgeable and proactive defense against changing cyberthreats.

1.6. Significance of the Study

This study is important because it may help users learn about the constantly evolving risks posed by cyberattacks and provide them with the information, they need to strengthen their online defenses. Researchers have established a relationship between the popularity and age of evasion tactics and their efficiency, [16,7] which is useful information for antivirus software producers as well as consumers. The study's conclusions will further the current cybersecurity conversation by encouraging a more knowledgeable and proactive defense against changing cyberthreats.

2. RELATED WORK

2.1. Literature Review

Installing antivirus software is seen as a crucial first step in safeguarding one's privacy on the internet. This literature study [1] however, explores the shortcomings of these products, emphasizing the ease of use and accessibility of techniques used by hackers to get around antivirus software. There is a significant association between the popularity and antiquity of evasion tools and their effectiveness, even though modern antivirus software that receives frequent updates works well against them.

Interestingly, the study highlights default configuration weaknesses, showing that even the most recent antivirus software can be tricked by small changes to well-established evasion strategies. The study emphasizes the need for ongoing watchfulness since hackers use easily available resources to take advantage of potential vulnerabilities. The literature's ultimate goal is to increase user awareness of the hazards related to cybersecurity by advising them to stay vigilant and knowledgeable about the latest developments in digital threats.

Extending the research, the authors contrasted in a later paper the efficacy of antivirus software bypassing techniques on the Windows operating system with Kalogranis' work. In order to expand on their research, the authors included a new antivirus bypass tool dubbed TheFatRat [8], replicated the tests using the tools used by Kalogranis, and utilized a payload created with Metasploit. Shellter and Veil-Evasion were unable to get past security. Of the six antivirus applications that were employed, TheFatRat was able to bypass one (PeCloak.py 4) [9], whereas Avet was able to bypass five.

The research [10] chose to limit their testing to Bitdefender after reading an analysis of the antivirus software in another study, which ranked Bitdefender as one of the top options. The target PC was able to access the Remote Access Trojan (RAT) malware through the use of the Apache server. The authors examined nine different antivirus bypass methods, taking into account whether the antivirus program would be able to detect RAT as well as whether it would be able to prevent the

triggered Meterpreter session that RAT activated. As a fraction of the total number of ways for each tool, the effectiveness of these tools was displayed.

This paper [11] presents a new approach to return-oriented programming (ROP)-based code obfuscation. The two main aspects of ROP— automated analysis and creation of ROP chains for a given code and the repurposing of valid code as ROP gadgets—pose problems to standard malware research. The developed program, ROPInjector, uses executable code to patch the ROP chain and convert shellcode to its ROP equivalent. Experimental results on VirusTotals show that ROPInjector can bypass nearly every antivirus program, demonstrating the efficacy of ROP in obfuscating code. This study highlights the need for improved cybersecurity measures by highlighting the possible threat posed by ROP in cyberattack campaigns.

The research [12] concentrated on malware that can change its code on the fly to avoid detection, known as polymorphic malware. This method entails developing several malware variations, each with a unique code signature. Upon execution, the malware randomly chooses and runs one of the variations. Because each form of the malware has a different code signature, this makes it harder for antivirus software to detect the malware.

The literature study leads to the conclusion that, although antivirus software is not perfect, antivirus software bypass technologies do have benefits and drawbacks. The effectiveness of some antivirus software bypassing tools varies significantly, as has been observed. This variation can be ascribed to a number of factors, including research methods, test dates, the type of malware being bypassed, its version, the tested antivirus software version, and even the collection of antivirus solutions that have been tested. Antivirus software and anti-virus software are engaged in a fierce competition in which the advantages of each side might have a substantial impact on the outcome.

As demonstrated, individual antivirus bypassing has been researched in the past for older antivirus versions. To the best of the author's knowledge, no thorough study has been done on the use of many antivirus bypass strategies together, nevertheless. Even while separate strategies have been researched and developed, it has not yet been investigated how efficient they are when combined. Considering the dynamic nature of malware and antivirus software, it is important to explore the ways in which different methods can be blended to get beyond several security levels. By better understanding antivirus software flaws, more resilient and efficient security measures may be created. This research can help.

2.2. Methodology

Malware evasion refers to strategies used to evade security system detection. This can involve using encryption to conceal dangerous payloads, polymorphic code that alters its appearance, and taking advantage of security software flaws. Avoiding detection frequently necessitates constant adjustment to security solutions' countermeasures. Malware evasion can be on desk or in memory.

more, so that they can avoid "Heuristic Detection," which makes it difficult for the program to understand the

instructions from antivirus software.

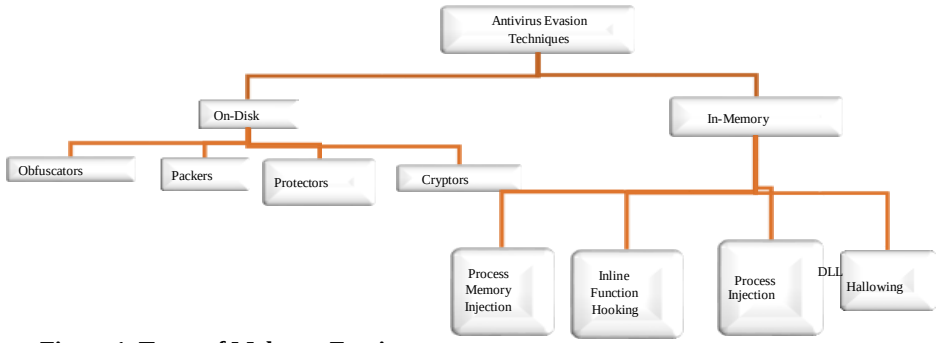

Figure 1. Types of Malware Evasion

2.2.1 Malware Evasion On-Desk

a) Packing

Malware is packed similarly to a compressed file, including new instructions and a larger file size to evade "signature-based detection."

b) Obfuscators

It obfuscates the blacklisted functions, such as VirtualAlloc, VirtualProtect, and more, so that they can avoid "Heuristic Detection," which makes it difficult for the program to understand the instructions from antivirus software.

c) Protectors

Although it complicates the malware's reverse engineering process, the Protectors app [3] is a regular one that wasn't intended for use in evasion, but it still has its uses.

Malware Evasion In-Memory

a) Remote Process Memory Injection

In order to apply this technique, we require certain APIs, such as: We inject our process or payload into a normal process like

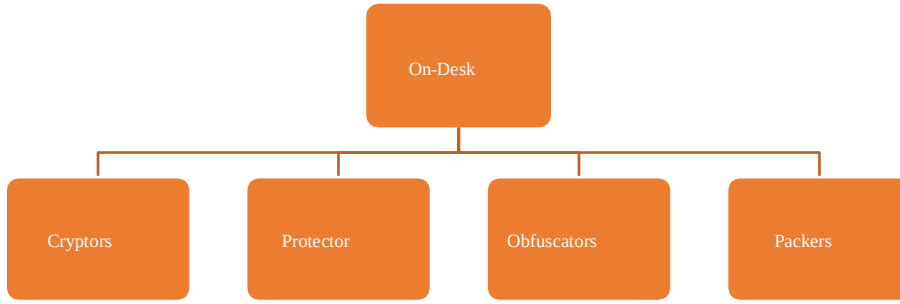

Figure 2: Showing Methods to Evade AV

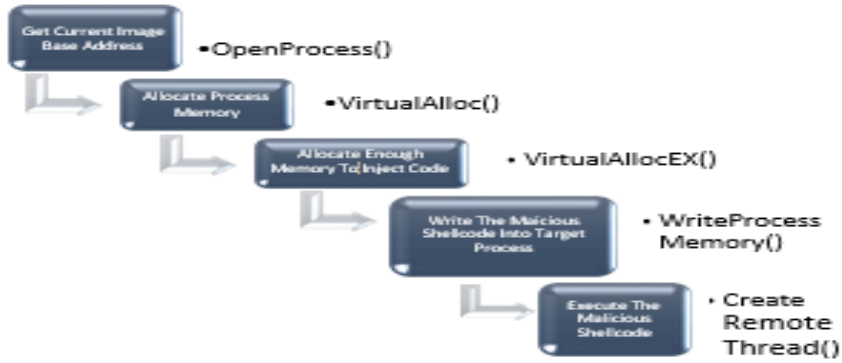

Figure 3: Common APIs for Remote Process Memory Injection

Reflective DLL Injection

An injection method that dispenses with using the conventional Windows APIs to load DLLs in order to load a DLL into the memory of a process. When limitations or security safeguards prevent the use of conventional DLL injection techniques, this can be helpful. The method by which Reflective DLL Injection operates is called manual mapping. The fundamental idea is to execute the DLL directly from memory by mapping it there rather than utilizing the regular Windows API calls. As a result, the DLL can function without raising security alerts or drawing attention from antivirus programs.

b) Process hollowing

We create a fictitious process that consumes space, pause it, modify its content to match our payload, and then restart the process with his updated instructions and content.

A technique called "inline hooking" allows you to change a process's code while it's still executing in memory. Redirecting function calls from the original code to a new location in memory accomplishes this. Although there are other approaches, these are the well-known ones that are employed.

In the current digital era, cyber-attacks are a constantly changing concern. It's critical to stay one step ahead of attackers who are always coming up with new ways to get around established defenses. This research outlines the strategies employed by these attackers, with a particular emphasis on how they evade Endpoint Detection and Response systems. Certain malware can successfully evade detection by employing strategies like the usage of syscalls. These strategies go beyond the first infiltration phase. Attackers use sophisticated tactics like process injection and DLL hijacking to keep control of the system after they have gained access. Regarding analysis, 'Dark Crystel RAT (DCrat)' is highlighted as a leading illustration of contemporary cyber risks. Examining this danger in depth gives readers a thorough grasp of the difficulties this type of malware poses by illuminating how it operates. This information serves as a tool and is not merely academic. Individuals, companies, and organizations can better prepare and safeguard their digital assets in an increasingly hostile cyber environment by being aware of these hazards.

Techniques of Malware Evasion and Bypass

Following are the techniques used:

2.3.1. Technique 1

Defense Evasion Technique Using Direct Sys-calls and Advanced Evasion Methods

In order to escape AV/EDR detection, this strategy entails creating a suite of tools that utilize direct syscalls, evade sandboxes, employ strong encryption, and change procedure names. It also describes how to circumvent security protections and generate memory snapshots using the well- known utility Dumpert, which makes use of direct syscalls [6]. Notably, Microsoft Defender identified

Dumpert after it was created and utilized on the disk. This discovery prompted research into avoidance strategies for both static and dynamic scenarios.

It's essential to understand the specifics of Native APIs and Windows APIs. Applications run in user mode on Windows. They carry out operations using Windows APIs. Security solutions like AV/EDR can't view anything past the native APIs included in ntdll.dll. Consider malicious software that makes use of Windows API functions like WriteProcessMemory, CreateRemoteThread, and VirtualAllocEx. These APIs link to additional ntdll.dll API activities. The majority of the operations in ntdll.dll are sets of instruction steps that initiate kernel system level operations. AV/EDR tools often connect to Native APIs and modify the application's route whenever it performs these

activities, enabling them to detect potentially dangerous activity in the app. EDRs load their DLLs into the process memory at startup in order to monitor the actions of the application.

Defense Evasion Technique: A Two-Part Exploration

Part 1

Using native API function names, the syscalls are discussed in the first section. Next, to further complicate static analysis, the tool is enhanced with name changes. Creating ASM/H pairings with SysWhispers 2, which always utilizes random function names and determines syscalls as they change, is one step in setting up this evading detection technique.

his resolves the function hash into syscalls and make the call.

The native calls show up when you use IDA-PRO to perform a static analysis of the implant. These calls serve as markers of the binary's activity. With this combination, malware researchers may easily infer that the program is carrying out a process injection—a technique frequently used by malware creators for this very goal.

The method uses three sandbox evasion tactics in addition to encryption: determining the RAM capacity, determining the processing speed, and determining the number of core processors. The code above specifies that 8GB of RAM is required; the values for core processors and RAM capacity are adjustable. The application is meant to stop running right away if the RAM is discovered to be less than 4GB.

Even with the use of direct syscalls, which effectively get over most AV/EDR solutions [13], there is still a need to improve the implant's stealth and resistance to analysis. AES encryption is used to further obscure against static analysis. Understanding that the well-known program msfvenom regularly generates shellcodes that are detected by AV/EDR systems, the shellcode was encrypted using AES to strengthen its tealthiness.

Part 2

To increase its stealth, the approach incorporates random naming for operations and functions, as was discussed in Part 1. For this reason, both the prototypes' names and the names of these operations were changed. Notably, prototypes of Native APIs are still easily recognizable even if they are not yet defined.

This version of the implant has function names that are chosen at random. This method is purposefully designed to make static analysis more difficult for malware experts. This foresight also takes into consideration possible future circumstances in which AV/EDR systems could identify the binary using these function names and the signatures that go along with them [14].

The methods were tested on Windows 11 by pitting them against McAfee, Microsoft Defender, and Kaspersky [15]. Surprisingly, none of these security measures were able to identify the implant, suggesting that the static and dynamic assessments required by these security measures were successfully circumvented.

The payload was integrated into explorer.exe. The payload's presence can be observed within the memory

address of explorer.exe, designated as RWX.

AntiScan was also used to evaluate the binary.me to assess the methods' effectiveness in detecting things. Remarkably, the binary escaped detection entirely.

By using randomized procedure names, strong encryption, sandbox evasion techniques, and direct syscalls, it was possible to successfully avoid EDR/XDR detection. In the final part, the strategy that may be used to go past Outflank's Dumpert tool is intended to be explained.

2.3.1.1. Bypass Dumpert Tool (Outflank)

Outflank created an amazing program that creates memory dumps by using straight syscalls. But because it's open-source, the majority of AV/EDRs have updated their signatures to support Dumpert. Instead of changing the signature, a
different and more effective bypass technique was selected, with remarkable results. To begin with, @TheWover's tool 'Donut' was used to create an autonomous shellcode for Dumpert [16] in its raw form. All it takes to convert Dumpert.exe into raw shellcode is a simple command.

In order to avoid Dumpert's static analysis, in- memory execution is used. Although Dumpert's default method for creating memory dumps is through direct syscalls, an injector was also created to load Dumpert shellcode into a remote process. The same approaches that were previously mentioned are incorporated into this loader.

Because direct syscalls are incorporated into the injector to get beyond the user-mode hooking that AV/EDRs impose, this technique effectively gets around AV/EDRs.

2.3.2. Technique 2

Achieving Elevated Reverse Shells via DLL Hijacking and Mock Directories

The goal of this approach is to obtain a high-level privileged reverse shell by circumventing Windows UAC security features through the use of DLL Hijacking and Mock directories. The method, which security experts have identified, uses dummy files in conjunction with a simplified DLL hijacking procedure to get around UAC protections. Tests on Windows 10 were able to successfully disable the UAC security mechanism, raising concerns about how resistant Windows 11 is to similar tactics.

Escalating privileges is usually the next step after gaining initial access, with objectives such as hash dumping or performing [16] privileged actions that enable lateral movement inside a network. Think about a domain user who uses a PC and is also the local administrator. In the event that this user is compromised by an attacker, there is an instantaneous push to elevate privileges in order to dump hashes and utilize that user's NTLM hashes for network authentication. But since there is already an elevated reverse shell in place and a privileged connection to the C2 server established, there is no need for this kind of escalation. This method will explore the principles of DLL hijacking and identify particular Windows binaries that are helpful in executing this attack. The preferred instruments comprise Metasploit for constructing.

Dynamic Link Libraries, or DLLs for short, are repositories of processes and code that facilitate Windows programs. Because they use the Portable Executable (PE) file type, they are similar to EXE files but cannot be executed directly. In
essence, DLL hijacking enables the insertion of malicious code into particular apps or services. This is accomplished by replacing the original DLL with a malicious one, making sure that the malicious DLL launches when the service is turned on. Because of the way certain Windows applications look for and load DLLs, such a swap becomes possible. When the DLL path of a service is not predefined in the system, Windows will automatically look for it in the environment path. By using this search pattern, attackers can place the rogue DLL in a location that Windows is aware of, preparing the way for the malicious code to be executed.

2.3.2.1. UAC – User Account Control

Initially included in Windows Vista and maintained in later iterations, UAC functions as a safeguard. Elevated rights cannot be provided to high-risk apps unless the user confirms it. Microsoft added "exceptions" to the UAC framework in an attempt [17] to improve user experience. This allowed trusted system DLLs stored in C:\Windows\System32\ to automatically rise to higher privileges without triggering a UAC question.

2.3.2.2. Mock Directories

In essence, a fake directory is a mimicked directory that can be identified by its trailing space. Consider the Windows trustworthy directory "C:\Windows\System32." The dummy equivalent would be "C:\Windows\System32," with the trailing space being the main distinction. Here, it's crucial to emphasize that Windows Explorer cannot be used to create mimic directories. PowerShell or the command prompt (cmd) must be used for creation. It is not possible to create "C:\Windows," however it is possible to set up "C:\Windows \System32."

2.3.2.3. TaskManager (taskmgr.exe)

Taskmgr.exe's integrity level was checked throughout the study. Taskmgr.exe is located in "C:\Windows\System32" and loads many DLL files when it runs. Attackers have the chance to use the DLL hijacking technique with this program [18]. This procedure "autoelevates" each DLL it introduces because of its high integrity level by design. It is possible to use many executables in a DLL hijacking attack [19]. "computerdefaults.exe" is the attack executable selected in this method. Attackers use these binaries to increase [20] their level of power in Windows, enabling them to perform DLL hijacking and change registry settings, among other things

2.3.2.4. Exploitation

This section explores the attack's mechanism, showing how an attacker may bypass Windows 11's UAC protections and acquire an administrator shell by using DLL hijacking and fake folders. This method's effectiveness was verified on Windows 11, even while Windows Defender was turned on.

Steps:

1. Crafting a Malicious DLL Constructing
2. Mock Folder and Loading the Malicious DLL
3. Securing an Administrative Reverse Shell
4. Launching Mimikatz

To begin, a shellcode was formulated utilizing Msfvenom in the CSharp format, with Metasploit serving as the C2 server.

"Msfvenom -p windows/x64/shell_reverse_tcp lhost=0.0.0.0 lport=555 -f CSharp".

Following the creation of the shellcode, a straightforward C++ program was developed to produce a DLL file. This program incorporated the previously generated shellcode.

The next step is creating a batch program that creates fictitious folders, copies a file to one of these fictitious directories, and tries to load the malicious DLL. There are a number of ways to use Mimikatz and avoid Windows Defender detection. On the C2 server [6], user hashes were collected when Mimikatz was successfully launched. Numerous network-wide attacks may be carried out to authenticate users using these NTLM hashes.

2.3.3. Technique 3:

Direct System Calls for AV/EDR Evasion, User-Mode vs Kernel Mode

A variety of techniques are employed by contemporary AVs and EDRs to do both static and dynamic analysis. They may look at a variety of signatures, including keys, hashes, and recognized strings, to find out if a file on disk is dangerous.

Nevertheless, attackers have created a wide range of obfuscation techniques, rendering static analysis all but useless. Dynamic/heuristic analysis is the primary emphasis of modern EDRs, which allows them to keep an eye on how each process behaves on the system and search for unusual activity. As a result, if malicious files have been disguised, this approach can download them and perhaps leave the EDR unnoticed [9]. However, as soon as the virus is activated, the EDR will recognize it and stop it. User-land hooks are used by the majority of AVs, EDRs, and sandboxes to monitor and intercept each user-land API call. They are unable to trace a technique that enters kernel mode and conducts a system call.

The fact that system call numbers differ between OS versions and occasionally even between service build numbers presents a problem. Nonetheless, the inmemory NTDLL module may be scanned to retrieve the syscall numbers using a library called inline syscall. The tricky part of this is that this module uses Windows API calls to retrieve the syscall number. These routines will not obtain the right number if an AV/EDR hooks them. Using Syswhispers is one alternate method that this blog discusses. By creating header/ASM files that implants can utilize to start direct system calls, SysWhispers helps in evasion.

2.3.3.1. SysWhispers1 vs SysWhispers2:

Although there is no requirement to specify which Windows versions to support, the usage is nearly comparable to that of SysWhispers1. Behind the scenes, most of the changes take place. It no longer uses @j00ru's syscall tables

and instead uses the @modexpblog-popularized "sorting by system call address" technique, which significantly reduces the size of the syscall stubs. The particular implementation in SysWhispers2 is a modification of the concept of @modexpblog. The function name hashes are randomized with every generation, which is one difference. Notable is also another version that was previewed previously by @ElephantSe4l and is built on C++17. Although it is still accessible, the original SysWhispers repository could eventually be retired.

2.3.3.2. API Hooks and Windows Architecture

AV/EDRs use a technique called "hooking" to intercept function calls and direct code flow to a controlled environment where the call's maliciousness may be examined. It is clear from looking at the Windows Architecture that a library by the name of NTDLL controls how user programs interact with the more complex OS operations.DLL.

The primary link between user-mode apps and the OS is the Native API (NTDLL.DLL). As a result, the OS serves as the interface between all applications. For example, ZwWriteFile and other frequently used Native APIs are stored in NTDLL.DLL. Several DLLs are loaded into a process's memory address space when it is started. When an AV/EDR loads a DLL, it can alter the function's assembly instructions by adding an unconditional jump at the start that points to the EDR's code.

Modern operating systems use multiple privilege levels and virtual memory to isolate and separate running processes. Kernel-mode and user-mode are the two primary privilege levels recognized by the Windows operating system. Windows ensures that apps stay segregated and are unable to directly interact with system resources or critical memory regions by using this technique [18]. Direct access could be dangerous by nature and could cause problems with the system. The CPU switches to kernel mode when a program attempts to carry out a privileged job. Software can enter kernel mode thanks to syscalls, which makes it easier to do privileged tasks like writing files. Take the previously described Win32 API function WriteFile as an example. A process invokes the user-mode WriteFile function when it wants to write a file.

2.3.3.3. Injecting Shellcode Via Windows API

Standard techniques for inserting shellcode into a process are widely known to individuals who are knowledgeable about malware creation. Shellcode injection is frequently carried out by attackers using Windows API calls as VirtualAllocEx, WriteProcessMemory, and CreateRemoteThread. By using this procedure, a section of memory is created where the shellcode may be written. Then a remote thread is started, and the system waits for it to finish. A shellcode that would be inserted into the NOTEPAD.EXE process was created using msfvenom. This shellcode's goal is simple: it shows a message box with the words "Hello, From Red Team Operator." “Msfvenompwindows/x64/messagebo x TEXT="Hi, From Red Team

Operator" -f csharp > output.txt.

This method introduces shellcode into a process by utilizing Windows APIs. The purpose of the presentation is to show that AV/EDR systems can identify such behaviors since they have hooks on these APIs. When memory is allocated to a process and marked as concurrently executable and writable, concerns are aroused. Since the shellcode is transcribed, executed, and created in memory using Windows APIs, it is obvious that AV/EDR systems would detect and flag these events.

2.3.3.4. Windows API Calls

This technique involves generating and injecting shellcode into notepad.exe. To achieve this, either the process name or the process id is required. Thus, the technique retrieves the pid of notepad.exe.

2.3.3.5. Shellcode Injection Through Syscalls

A program that writes the shellcode into the process and allocates memory via direct syscalls was created using the same previously produced shellcode. SysWhispers2, a program that dynamically resolves syscall numbers, was used. Due to SysWhispers1's reliance on the Windows operating system, SysWhispers2 was created and put to use.

The primary operating system for this method was Ubuntu, which posed a problem with the ASM/Header pair generated by SysWhispers2. There is a separate assembly format needed for compilation with Mingww64, and there is a distinct assembly format for MASM. Conor Richard deserves recognition for reworking the current assembly, adding support for x86 (Wow64 & Native) and NASM ASM, and enabling compilation using MinGW and NASM straight from the command line. A malicious program was created [21] that inserts the shellcode—created by msfvenom—into the process using direct syscalls. This time around, all operations—including creating memory and inserting the shellcode into the remote process—are carried out using direct syscalls.

After successfully compiling and executing, program is caught by Windows Defender. Windows Defender discovered this method. The cause is that it made use of Windows APIs, which are often observed by antivirus and endpoint protection programs. These security tools make it easy to discover malicious programs that depend on Windows API calls to carry out such acts because they have hooks on user-land APIs.

Windows Defender discovered this method. The cause is that it made use of Windows APIs, which are often observed by antivirus and endpoint protection programs. These security tools may easily identify malicious applications that rely on Windows API calls to carry out such acts since they have hooks on user-land APIs.

Once the malware was successfully compiled, it was possible to avoid both static and dynamic detection by running the malware in the presence of Windows Defender. Within the project, this method used function names and random variables.

In the past, unsigned char shellcode was used for initialization while creating malware []. Windows Defender was able to identify the infection as a result. The virus was identified by MDE as soon as it came into contact with the

disk, even though it had encrypted the shellcode and masked API calls. Further analysis revealed that the detection was caused by the term ShellCode. As a result, it has been noted that antivirus software occasionally raises a warning based on these patterns. The virus dynamically modifies its variable and function names in order to thwart this and modify the static signature.

This time, Windows Defender did not detect the malware, as direct syscalls were employed. By leveraging [23] direct syscalls, it's possible to evade AV/EDR user-land hooking mechanisms.

This time, not a single antivirus program detected the malware once it was uploaded to AntiScan.me. The outcomes might be explained by the malware's anti-sandbox methods, which include examining CPU speed, RAM capacity, and processor count, or by the usage of direct syscalls. However, the virus was able to effectively avoid both static and dynamic analysis when tested against several AV/EDR solutions.

3. RESULTS

Significant new insights into the dynamic landscape of malware evasion and bypass tactics are provided by the research, which also highlights the continual innovation of measures that undermine the effectiveness of conventional antivirus software. Notably, the study emphasizes the necessity for creative defensive strategies by highlighting the shortcomings of antivirus software. Testing contemporary antivirus software demonstrates its strong resilience to common evasion approaches, but the research also identifies flaws resulting from minute changes to tried and-true tactics. The methodology thoroughly examines in-memory and on-desk evasion strategies, describing methods including packing, obfuscation, and reflection DLL injection. Advanced evasion techniques demonstrate the versatility of malware creators in avoiding detection. One such technique is defensive evasion via direct system calls. The effectiveness of combining encryption, random naming, and sandbox evasion to successfully evade AV/EDR systems is demonstrated by the results of particular evasion approaches. The research also looks at DLL hijacking and fake directories, which may be used to elevate reverse shells and cause issues with Windows UAC protection. Methods for AV/EDR evasion via direct system calls are shown, along with an overview of tools such as SysWhispers2 and the difficulties presented by contemporary security technologies. The study advocates for proactive defensive tactics and ongoing awareness in order to improve cyber resilience in the face of constantly changing cyber threats.

4. CONCLUSION

Proactive defense and awareness are crucial in the face of constantly changing cyberthreats. This study highlights the need for a comprehensive and constantly evolving strategy towards cybersecurity through its discussion of inventive methods and procedures. Conventional defenses still have their place, but ongoing learning and adaptation are also necessary. This research seeks to provide people and organizations with the knowledge necessary to strengthen their digital

defenses through its thorough examination. Let this research serve as a light for improved cyber resilience as we traverse this digital age.

5. REFERENCES

[1] D. Samociuk, "Antivirus Evasions Methods in Modern Operating Systems," *Applied Sciences*, vol. 13, no. 8, pp. 5083, 2023.

[2] D. Waterson, "Managing Endpoints, the Weakest Link in the Security Chain," *Network Security*, vol. 2020, no. 8, pp. 9-13, 2020.

[3] S. Choi, T. Chang, S. Yoon, and Y. Park, "Hybrid Emulation for Bypassing Anti-Reversing Techniques and Analyzing Malware," *The Journal of Supercomputing*, vol. 77, no. 1, pp. 471-497, 2021.

[4] S. Gold, "Advanced Evasion Techniques," *Network Security*, vol. 2011, no. 1, pp. 16–19, 2011.

[5] D. Li, S. Cui, Y. Li, J. Xu, F. Xiao, and S. Xu, "PAD: Towards Principled Adversarial Malware Detection Against Evasion Attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, pp. 1-16, 2023.

[6] A. Monika and R. Eswari, "Prevention of Hidden Information Security Attacks by Neutralizing Stego-Malware," *Computers and Electrical Engineering*, vol. 101, pp. 79-90, 2022.

[7] J. Cabrera-Arteaga, M. Monperrus, T. Toady, and B. Baudry, "WebAssembly Diversification for Malware Evasion," *Computers & Security*, vol. 131, pp. 32-43, 2023.

[8] R. S. Kunwar, "Malware Analysis of Backdoor Creator: FATRAT," International Journal of CyberSecurity and Digital Forensics, vol. 7, no. 1, pp. 72-79, 2018.

[9] "Evading Scanners," The Antivirus Hacker's Handbook, Wiley, pp. 133-164, 2015.

[10] F. A. Garba, F. U. Yarima, K. I. Kunya, F. U. Abdullahi, A. A. Bello, A. Abba, and A. L. Musa, "Evaluating Antivirus Evasion Tools Against Bitdefender Antivirus," Proceedings of the International Conference on FINTECH Opportunities and Challenges, vol. 18, Karachi, Pakistan, 2021.

[11] C. Ntantogian, G. Poulios, G. Karopoulos, and C. Xenakis, "Transforming Malicious Code to ROP Gadgets for Antivirus Evasion," *IET Information Security*, vol. 13, no. 6, pp. 570-578, 2019.

[12] M. Christodorescu and S. Jha, "Static Analysis of Executables to Detect Malicious Patterns," 12th USENIX Security Symposium (USENIX Security 03), 2003.

[13] D. Waterson, "Managing Endpoints, the Weakest Link in the Security Chain," *Network Security*, vol. 2020, no. 8, pp. 9-13, 2020.

[14] H. Anand, N. Kumar, and S. K. Shukla, "Adversaries Strike Hard: Adversarial Attacks Against Malware Classifiers Using Dynamic API Calls as Features," Electronics, pp. 20-37. 2021.

[15] M. Noor, H. Abbas, and W. B. Shahid, "Countering Cyber Threats for Industrial Applications: An Automated Approach for Malware Evasion Detection and Analysis," *Journal of Network and Computer Applications*, vol. 103, pp. 249-261, 2018.

[16] M. A. Titov, A. G. Ivanov, and G. K. Moskatov, "An Adaptive Approach to Designing Antivirus Systems," Safety of Computer Control Systems 1992 (Safecomp' 92), pp. 215-220, Elsevier, 1992.

[17] A. Sharma, B. B. Gupta, A. K. Singh, and V. K. Saraswat,

"Orchestration of APT Malware Evasive Maneuvers Employed for Eluding Antivirus and Sandbox Defense," *Computers & Security*, vol. 115, pp. 10-28, 2022.

[18] H. Liu, W. Sun, N. Niu, and B. Wang, "MultiEvasion: Evasion Attacks Against Multiple Malware Detectors," 2022 IEEE Conference on Communications and Network Security (CNS), pp. 10-18, IEEE, 2022.

[19] J. Chen, C. Yuan, J. Li, D. Tian, R. Ma, and X. Jia, "ELAMD: An Ensemble Learning Framework for Adversarial Malware Defense," *Journal of Information Security and Applications*, vol. 75, pp. 103-114, 2023.

[20] T. Tsafrir, A. Cohen, E. Nir, and N. Nissim, "Efficient Feature Extraction Methodologies for Unknown MP4 Malware Detection Using Machine Learning Algorithms," *Expert Systems with Applications*, vol. 219, pp. 119-127, 2023.

[21] U. Ahmed, J. C. Lin, and G. Srivastava, "Mitigating Adversarial Evasion Attacks of Ransomware Using Ensemble Learning," *Computers and Electrical Engineering*, vol. 100, pp. 107-119, 2022.