applied sciences

MDPI

Article

Antivirus Evasion Methods in Modern Operating Systems

Dominik Samociuk

Institute of Informatics, Silesian University of Technology, 44-100 Gliwice, Poland; dsamociuk@polsl.pl

Abstract: In order to safeguard one's privacy while accessing the internet, it is crucial to have an antivirus program installed on the device. Despite their usefulness in protecting against malware, these programs are not foolproof. Cybercriminals have access to numerous techniques and tools for circumventing antivirus software, which can greatly aid them in their illicit activities. The objective of this research was to examine the most prevalent methods and tools for bypassing antivirus software and to demonstrate how readily accessible and simple they are to use. The aim of this paper is to raise awareness among readers about the associated risks and to assist internet users in protecting themselves from potential threats. The findings of the research indicate that the efficacy of evasion tools is positively correlated with their age and popularity. Tests have shown that, with the latest updates, contemporary antivirus software is capable of resisting virtually all of the tested methods generated using default settings. However, the most significant aspect of this paper is the section presenting experiments with basic but powerful modifications to established evasion mechanisms, which have been found to deceive modern, up-to-date antivirus software.

Keywords: network security; antivirus evasion; cybersecurity

check for updates

Citation: Samociuk, D. Antivirus Evasion Methods in Modern Operating Systems. *Appl. Sci.* **2023**, *13*, 5083. https://doi.org/10.3390/app13085083

Academic Editors: Zbigniew Lubniewski, Tadeus Uhl and Przemysław Falkowski-Gilski

Received: 9 March 2023
Revised: 17 April 2023
Accepted: 17 April 2023
Published: 19 April 2023

Copyright: © 2023 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In the present day, the internet is widely accessible to people of all ages and backgrounds, allowing them to use it for communication, entertainment, and information acquisition. However, hackers also have access to the internet and can use malicious software to compromise user privacy and important files.

To protect against such attacks, users often install antivirus software that aims to reduce the risk of device infection. Downloading files from unknown sources, even well-known programs like web browsers, can lead to infection. Therefore, it is essential to download content from trusted and verified sources. If a suspicious file is downloaded, the antivirus software will typically display an appropriate message and move the infected file to quarantine, where it can be cleaned or deleted. User trust in antivirus software is essential, and regular updates of the signature database and the tool itself are important.

Hackers use various bypassing and evasion techniques to gain access to devices. Antivirus software developers continuously develop new security tactics and update signature databases to combat emerging threats. However, the increasing number of new viruses implies that current protection technologies may not always be sufficient [1].

This article discusses popular bypassing techniques and tools that should be detectable by most antivirus programs but also shows that complex attack chains that combine evasion techniques can bypass modern and commonly-used antiviruses. The paper is structured into nine sections: introduction, related work, security mechanisms, evasion techniques, testing environment, results of research on bypassing antiviruses, discussion, conclusions, and further research directions.

2. Related Work

The issue of circumventing antivirus security measures starts with understanding the behavior of malware samples [2]. A thorough analysis of malware is crucial for enhancing

the operation of antivirus engines, ultimately improving user security and privacy. Techniques to bypass antivirus security measures have been developed since the inception of antivirus software. The perpetual competition between hackers and antivirus developers results in more advanced attacks.

In 2018, Kalogranis [3] tested four tools, namely AVET, peCloack.py, Shellter, and Veil-Evasion, used to bypass antivirus software. Kalogranis' results demonstrated that AVET and Veil-Evasion were efficient at bypassing antivirus software by 60%, while peCloack.py and Shellter were efficient at 40%.

In a subsequent paper [4], the authors compared the effectiveness of antivirus software bypassing tools on the Windows operating system with Kalogranis' work, extending the research. The authors repeated the tests on the tools used by Kalogranis, added a new antivirus bypass tool called TheFatRat, and used a payload generated with Metasploit to extend the research. Veil-Evasion and Shellter failed to bypass security. TheFatRat bypassed one, PeCloak.py four, while Avet bypassed five out of six antivirus software programs used.

Panagopoulos [5] conducted a study on bypassing antivirus software, utilizing malware generation tools such as TheFatRat, Phantom-Evasion, Hercules, and Veil-Evasion in the tests. Phantom-Evasion was the most effective tool, achieving around 65% efficiency, followed by Hercules with around 47%, TheFatRat with 22%, and Veil-Evasion with the lowest efficiency within the range of limits of 10%.

Another work [6] analyzed Bitdefender as one of the best antivirus solutions, and the developers decided to perform all tests only on this antivirus program. The malware generated was Remote Access Trojan (RAT), which was made available to the victim machine using the Apache server. Nine different antivirus bypass tools were tested, and the authors considered not only whether RAT would be detected by antivirus devices but also whether the antivirus would block the triggered Meterpreter session activated by RAT. The effectiveness of these tools was presented as a percentage of the number of methods used for a given tool.

The use of malware and other malicious software is a persistent threat to computer systems, and as a result, antivirus software has become an essential tool for protecting against these threats. However, as antivirus software becomes more advanced, so do the techniques used to bypass it. We explored research related to state of security of antivirus tools. There is also current research on anti-antivirus techniques.

In [7,8], authors presented a technique of obfuscation that involves modifying the code of a piece of malware to make it more difficult to detect. The modifications may include changing variable names, adding extraneous code, and using encryption to hide the true function of the code. Researchers have developed various tools for detecting obfuscated code, but the effectiveness of these tools depends on the level of obfuscation used.

In [9], the researchers presented trends in antivirus evasion techniques, while [10] proposed new details for fileless malware, a type of malware that does not rely on a file to infect a system. Instead, it resides in the system's memory, making it more difficult to detect. This technique has become more popular in recent years as antivirus software has become better at detecting traditional file-based malware.

In [11,12], the authors focused on polymorphic malware, which is a type of malware that can change its code on the fly to evade detection. This technique involves creating multiple variants of the malware, each with a different code signature. When the malware is executed, it selects one of the variants at random and executes it. This makes it more difficult for antivirus software to detect the malware, as each variant has a unique code signature.

Finally, Ref. [13] covered standard signature evasion, while [14] researched detection of malware run as a virtual machine.

From the literature review, it can be concluded that, while no antivirus is infallible, antivirus software bypass tools have their advantages and disadvantages. As observed, there is a significant difference in the effectiveness of certain antivirus software bypassing tools, which can be attributed to various factors such as research measures, test execution dates, the difference in masked malware, its version, the version of the tested antivirus soft-

ware, and even the pool of tested antivirus solutions. Antivirus tools and tools designed to bypass them engage in an intense war, where one party exploiting the other's disadvantage can lead to a significant difference in results.

As shown, individual bypassing of antiviruses has been studied in the past for old versions of antiviruses. However, to the best of author's knowledge, to date, there has been no comprehensive research conducted on the combination of multiple antivirus bypass techniques. Although individual techniques have been studied and developed, the analysis of their effectiveness in combination has yet to be explored. Given the constantly evolving nature of malware and antivirus software, it is essential to investigate how various techniques can be combined to bypass multiple layers of protection. This research can contribute to improving the understanding of the vulnerabilities in antivirus software and developing more robust and effective security measures.

3. Antivirus Detection Mechanisms

The human immune system, like any system, is prone to errors, and can be compromised, leading to illnesses. Similarly, electronic devices can be compromised by malicious software, and they require protective measures. Antivirus software is essential for devices and should be given priority in terms of installed software. Antivirus software scans files and compares their contents with a blacklist database of known malicious codes. If it detects any potential threats, it removes or quarantines them for cleaning. Antivirus software functions similarly to a doctor by analyzing potential threats and determining how to resolve them. Understanding the methods used by antivirus software can help us better understand attackers' motives when creating new malware. This sections provides an overview of modern antivirus software techniques used to identify malware and protect electronic devices from potential threats.

3.1. Signatures

Signature-based malware detection is a traditional method that has been used for a long time. A signature is a unique pattern or code template of a virus that can be used to identify it [15]. The signature is a short fragment that does not reveal any useful information for virus writers to use. Antivirus software can compare the signature of a scanned file with all the signatures from its blacklist of suspicious files, also known as the signature database. If the signature being compared exists in the database, the scanned file is marked as malicious. This method is effective for threats that are already known, making it a popular threat search technique. However, signature-based detection has some drawbacks [16]. It is powerless against new threats and susceptible to modifications that mask the virus. Scanning a new or modified threat may prove unreliable until the signature database is updated with new or modified signatures. Additionally, storing all signatures leads to an exponential increase in their number, resulting in lower performance and longer search times. However, if the virus software change does not modify the signature directly, the signature can still be useful in identifying the malware.

3.2. Behavioral Detection

Behavioral recognition is a malware detection technique that is based on analyzing the behavior of programs when launched. It was presented by Fred Cohen [17] as a way of defining illegal services and distinguishing them from legal ones within a system. Unlike signature-based detection, which relies on unique patterns, the behavior detector collects programs with similar behavior, which can help in identify different threats based on one behavior signature [18]. This method is particularly useful for dealing with new threats and modifications, as it does not rely on a signature database. However, one disadvantage of behavioral recognition is that it may incorrectly classify safe programs as dangerous if they use device resources in a manner similar to known viruses, resulting in a false positive. Zahra Bazrafshan, Hashem Hashemi, Seyed Mehdi Hazrati Fard, and Ali Hamzeh also highlighted this advantage in their publication [19].

3.3. Heuristic Detection

Dmitry Gryaznov in [20] defined heuristics as a set of rules applied to a program to determine if it contains a virus or not. Seyed Mehdi Hazrati Fard, Ali Hamzeh, Zahra Bazrafshan, and Hashem Hashemi described heuristic-based detection as utilizing machine learning and data mining techniques to learn a program's intentions [18]. The learning model detects suspicious features found in new and modified viruses. Heuristic analysis is classified as static or dynamic. Static analysis compares the possible behavior of the decompiled code with the heuristics database, and dynamic analysis runs a program in an isolated environment to observe its operation [21]. The combination of static and dynamic heuristics allows for the assessment of new and modified threats without exposing the client's device. However, as Gryaznov and Fred Cohen noted, it is impossible to create an algorithm that can definitively distinguish between a virus and an uninfected program with 100% accuracy. Heuristic-based detection also has its limitations. Static analysis may return a false positive, as it can find similarity between safe files and the heuristics database. Dynamic analysis may fail to detect a virus that requires certain conditions to be met before being launched. To address these limitations, combining heuristics with signature-based detection can eliminate these drawbacks.

3.4. Sandboxing

In advanced threat detection methods, such as heuristic-based detection, Sandbox serves as a system that emulates the behavior of a host computer. It functions as a virtual machine where potentially harmful files are executed to monitor their actions. Sandbox offers a secure environment that isolates the executed programs from the client's device, thereby reducing the chances of device infection to negligible levels.

4. Antivirus Evasion Mechanisms

Although modern antivirus software is generally effective in protecting against most current threats, it is not immune to vulnerabilities in its code that can be exploited by hackers. In this section, we will discuss several popular techniques that can be used by hackers to exploit these vulnerabilities.

4.1. Signatures

A vulnerability in antivirus software can lead to the exploitation of the program's flaws by hackers. An example of such a vulnerability is the discovery that poorly-designed signatures in an antivirus program can be used as an attack tool against the user. Researchers have developed a method to obtain the exact virus signatures of a specific antivirus software. The method involves three steps: (1) detecting critical bytes in the malware using feedback from the antivirus, (2) aligning corresponding bytes from samples with the same pattern and combining them into one sequence, and (3) converting the linked sequences into the appropriate format to obtain the original signature. The obtained signatures can be used to implant them in secure data, resulting in the quarantining or deletion of such data, which can lead to significant problems. This vulnerability and its exploitation are detailed in [22].

Currently, there are numerous scientific works utilizing the detection mechanism of signatures for the aforementioned attacks. An exemplary framework was presented in [23], where the authors attacked operating systems by forcing the antivirus to perform specific actions harmful to the user.

4.2. Obfuscation

Obfuscation is a well-known technique employed by attackers to modify the code and signature of a virus to make it difficult for antivirus software to interpret the malicious code. This technique involves adding redundant code, modifying the case of a script, and reordering commands to create confusion and evade detection. Although obfuscation does not affect the malware's functionality, it may reduce its effectiveness [24]. However,

relying solely on obfuscation to deceive antivirus software may not be sufficient, as modern antivirus software utilizes heuristic and behavior-based approaches to detect potential threats. To bypass antivirus detection, attackers often use obfuscation in combination with other techniques, such as polymorphism or metamorphism.

Another research study using a systematic approach for antivirus evasion using malware obfuscation was presented in [25]. The authors focused on comparing different mechanisms of obfuscation for bypassing virus detection on different platforms. Obfuscation is a wide term, so new kinds of methods for code obfuscation are presented in the literature, such as ROP gadgets used for antivirus evasion presented in [26].

4.3. Encryption

Encryption is a process that splits a virus into two components, namely the virus body and the decryption loop. The virus body is encrypted using a suitable method, such as XOR, while the decryption loop is a brief program responsible for encrypting and decrypting the virus body. The virus must be decrypted using the decryption loop before it can function. Similarly, an antivirus needs to obtain the decryption portion of an infected file to read its encrypted content, and only then can it compare the signatures [27].

Unlike code obfuscation, which aims to obscure code, many tools use encryption based on cryptographic principles to limit the possibility of static analysis of malware code. This includes both ready-made frameworks such as PEzoNG presented in [28], as well as implementations of well-known encryption algorithms within ransomware code. An analysis of the latter solution, using Cryptolock as an example, was conducted in [29].

4.4. Morphism

Morphism, specifically polymorphic viruses, is a complex technique that aims to make analysis more difficult by creating an unlimited number of different decryptors. Unlike encryption, no part of the virus is encrypted. Polymorphic viruses use numerous obfuscation techniques to change the appearance of the decryption code from copy to copy, making it harder for antiviruses to recognize the threat. In metamorphism, the entire program code changes instead of just the encryptor's code as in polymorphism. Each copy is built differently, with varying sizes or sequences of code, but its behavior remains unchanged. To detect a virus using metamorphism, advanced behavioral and heuristic detection engines are required [27].

Tools for detecting code morphism strive to keep up with the development of frameworks for generating encrypted code, such as the tool presented in [30] for changing shellcode with each subsequent run.

4.5. Process Injection

Process injection is a technique utilized to camouflage a malicious process by running it in the memory space of another process. Additionally, the injected process can inherit permissions of the host process, potentially providing a hacker with more opportunities. The difficulty in detecting this technique lies in the fact that malware can be injected into a program that operates similar processes. Malware can effectively deceive behavior-based analyses and operate surreptitiously for a prolonged period before being detected. The classic DLL injection is one of the most popular process injection methods. This involves writing the path to the malicious DLL library into the address space of another process, followed by executing a remote thread that calls the malicious library during the injection process [31,32].

Process injection is often used with additional techniques mentioned in previous subsections. Research on how to detect process injection, followed by the process injection mechanism itself, is presented in [33], where the authors tried to detect fileless malware with process injection in various forms. There are many other frameworks where process injection is used to create multi-process malware execution in combination with other

malware evasion mechanims, such as the above-mentioned ROP programming in the ROPE framework in [34].

5. Research Environment and Toolset

Hackers utilize various tools to evade detection by antivirus engines. Some of these tools do not require a high level of expertise in cybersecurity. Inexperienced attackers can simply download the tool and follow the instructions provided by its creators to bypass weaker antiviruses. As technology improves and antivirus software is updated, some of these techniques become less effective over time. The current discussion focuses on describing the tools used in this research to provide better understanding of the tests that were performed.

5.1. Basic Tools

Kali Linux [35] is a Linux distribution that is free of charge and is based on Debian. It was launched in 2013 and is designed for advanced penetration testing and security auditing purposes. Kali provides a plethora of tools for tasks such as penetration testing, security research, reverse engineering, and computer forensics [36]. The operating system contains over 600 penetration tools, and it is important to mention that it is open-source software, which enables users to modify the software provided according to their requirements.

Metasploit is a widely-used framework by penetration testers and hackers that provides a vast array of tools for testing system vulnerabilities and breaking security [37]. With approximately 600 payloads, 2200 exploits, and 45 encoders, Metasploit is a powerful tool that is built into Kali Linux by default, making installation unnecessary. Once the user logs in to Kali Linux and enters "msfconsole" in the terminal, the tool is ready for use, and the user can listen for connections on the port generated by the payload. More details about the tool and its functionalities will be discussed in a subsequent section.

Msfvenom [38] is a payload generation tool provided by the Metasploit framework. It was utilized to generate payloads that were used in subsequent tests. The program can be invoked through the terminal by entering the Msfvenom command with the required arguments. The tool provides a range of options, including the generation of payloads, encoding, selecting the output file format, and adjusting the number of load coding iterations.

Shikata Ga Nai (SNG) [39] is a polymorphic XOR encoder with additive feedback that is available in the Metasploit framework. This tool has been around for some time but is still in use today. It employs code obfuscation techniques such as altering the order of code instructions, randomly changing registers, and adding junk code to evade signature recognition every time the code is compiled. Additionally, the additive feedback allows the output data to be incorrect in subsequent iteration stages when incorrect input data are received. This is due to the algorithm performing XOR operations on subsequent instructions, using a randomly-generated key, and then adding the current instruction to the key. Code decoding involves performing all steps in reverse order [40]. SNG can be used by adding an extra argument to the previously described example of payload generation, where the argument preceded by the -e flag selects the SNG encoder.

Example usage is as follows:

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.224.134 LPORT=8080
-e x86/shikata_ga_nai -f exe > defaultMsfvenom.exe
```

5.2. Antivirus Evasion Frameworks

Hyperion [41] is a real-time encoder that was developed in 2012. The program was initially designed for the Windows operating system. To install it on Kali Linux, it is necessary to install the Wine package [42], which enables support for Windows operations in a Unix environment. Once the Wine package is installed, Hyperion can be downloaded, extracted, and converted into an executable file. Thereafter, the steps involved in using Hyperion are straightforward. The command below can be used:

```
wine hyperion.exe payloadToEncrypt.exe encryptedPayload.exe
```

In the terminal, the use of the Wine package to handle the Hyperion executable is indicated by providing two consecutive arguments. The first argument is the executable file to be encrypted, and the second argument is the output file of the already-encoded executable. The operation of Hyperion can be divided into two parts, namely the encryptor and the container. The encryptor takes the executable file to be encrypted, computes its checksum, and adds it to the file in memory. It then generates a random key that is used to encrypt the input file using the AES-128 algorithm, along with the previously calculated checksum. The container acts as a decryptor, copying and decrypting the encrypted input file and running it. As it lacks the decryption key, it performs a brute-force key space check based on a checksum to determine if the key is valid. Although this may seem like a disadvantage, it actually protects against static and dynamic analysis by antivirus engines. Author Christian Ammann also mentions that the encrypted counterpart has an unknown signature and cannot be analyzed by heuristics, thereby protecting the binary code against reverse engineering or the replacement of encryption procedures with packers to reduce the size of the executable file [43]. However, since the document describing Hyperion is almost 10 years old, the claim about the unknown signature may no longer hold true.

TheFatRat [44] is a potent exploitation tool developed by Edo Maland, used for generating payloads that can run on various systems such as Linux, Windows, Mac, or Android [45]. It uses various evasion mechanisms such as obfuscation and cryptography techniques to hide the payload from antivirus scanners. This includes methods like encrypting the payload, encoding it in various formats, and using anti-debugging techniques. TheFatRat can modify the payload to bypass signature-based detection mechanisms used by antivirus software. This includes changing the file's signature or removing it altogether. Dynamic payload generation: The tool can generate payloads dynamically, meaning that each payload is unique and can not be detected by antivirus software that relies on pre-existing signatures. Polymorphic code: TheFatRat can generate polymorphic code, which means that the code changes each time it is executed, making it more difficult for antivirus software to detect. To install TheFatRat on a Kali Linux machine, the following commands are entered in the terminal:

```
git clone https://github.com/Screetsec/TheFatRat.git
cd TheFatRat
chmod +x setup.sh && ./setup.sh
```

After downloading TheFatRat repository and granting permission to execute the file, the installation process begins. Following this, the program update is carried out using the following commands in sequence:

```
cd TheFatRat
./update && chmod +x setup.sh && ./setup.sh
```

To initiate TheFatRat tool, the user needs to execute a command in the terminal with administrator privileges. The options and tools can be accessed by entering the corresponding numbers assigned to the exploits. To exploit PwnWinds, which is a backdoor program offering the same payload as previously-mentioned tools, option 6 is selected. TheFatRat employs the Msfvenom tool to create payloads. After selecting the desired option, the user is prompted to enter the IP and port for the attack, as well as the name of the output file. Depending on the option selected, the program may also require the user to choose the appropriate type of payload.

Shellter [46] is a software tool that facilitates the dynamic injection of a payload into a secure executable file. Payloads can be either created in Shellter or generated with other tools such as Msfvenom. However, it should be noted that Shellter is currently limited to supporting the injection of code only into 32-bit applications [47]. The installation of Shellter requires the previously-described Wine [42] package to be installed again. The Shellter tool can be installed by running the install command in the terminal.

After installation, the program can be initiated by running the Shellter command, which opens the program window. The user must then choose whether to use automatic or manual mode. For this study, the focus was on the automatic mode to demonstrate the ease of use of the tools employed. Next, the user selects the file to which the previously-generated payload will be injected. The program then prompts the user to enable "stealth mode", which ensures the proper operation of the infected program. Since the attacker typically wants the attack to go unnoticed, the user selects the 'Y' option. Following this, the program presents the option to use pre-existing payloads available in Shellter or to use the previously-generated payload. If the user chooses to use their payload, they can indicate it at this point. However, if a payload from the pre-existing list, such as the reverse_tcp meterpreter, is selected, the user must then indicate the host's IP and port.

Veil-Evasion [48] is a Python-based framework widely used to create payloads, which can effectively bypass antivirus protections [49]. It was designed to operate on Kali Linux, making installation a straightforward process by entering an install command in the terminal.

Once installed, the program can be started by typing the command "veil" in the console, which launches the program window displaying two tools. The tool of interest in this context is the Veil-Evasion option, which provides over 41 payloads, divided into 10 different programming languages, including C, PowerShell, Python, Ruby, and AutoIt. The user can list the payloads by using the "list" command and then selecting the desired payload using the "x" command, where "x" represents the assigned number of the payload. After selecting the payload, a window with various options for the generated payload appears, enabling the user to modify the payload to make it more unique and resistant to antivirus scanners. For payloads with a meterpreter console and reverse TCP connection, the required options are LHOST and LPORT, while Veil-Exploit offers many other optional settings. These include the use of an encoder, RAM checks, and the clicktrack option, which only launches the payload after a specific number of mouse clicks by the victim. After modifying the payload as per the attacker's requirements, the "generate" command generates the payload, and the user can specify the name of the output file and the desired extension.

6. Experimental Procedures and Results for Antivirus Bypass Mechanisms

In order to analyze the effectiveness of antivirus bypass mechanisms, tests were conducted on various antivirus software programs as well as tools developed to combat them. The focus of this section is to present the procedures for and results of these tests.

Attack flow is presented in detail in Figure 1. The first condition that a masking tool must meet to be effective is to bypass static scanning. This means that the software delivered to the victim's machine must be able to bypass the initial static scan to check whether the file is infected or not before the infected file is launched. If the software is detected at this stage, it is pointless to proceed, as the attacker's primary goal is to break through the antivirus defenses and enter the victim's system unnoticed. The second condition for an effective masking tool is to establish a connection to the meterpreter console while the malware is running on the victim's machine.

The research was repeated with the latest versions of both attack tools and antivirus software. In addition, the generated malware was also scanned using the antiscan.me online scanner to visualize the results for even more antiviruses. Usage of multiple antiviruses minimizes the risk of bias of chosen AVs in VM tests. From the obtained results, it can be concluded that, in fact, the best-performing antiviruses were used as stated by [50]. However, scans on virtual machines were given priority during the research. Both the antivirus and malware masking software were downloaded on the same day, meaning that they were the latest versions at the time of testing. However, during re-examination, only the antivirus software was updated to verify whether the results would differ after the antivirus update and whether the antivirus would recognize the old threat over time or not.

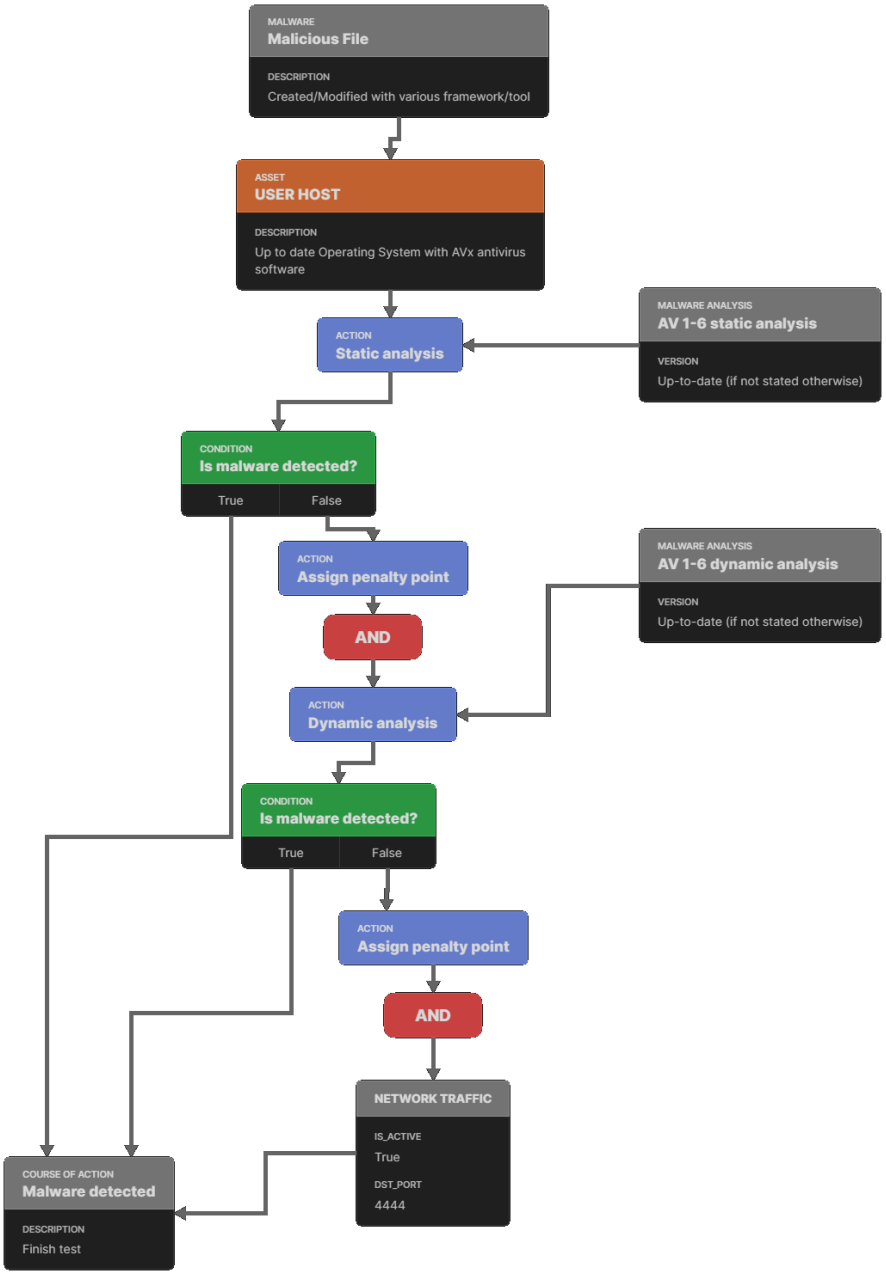


Figure 1. Attack flow and awarding of penalty points during the course of experiments.

Overall, these experimental procedures and results provide valuable insights into the effectiveness of antivirus bypass mechanisms and the importance of staying up to date with the latest versions of antivirus software.

The payload obtained with the Msfvenom tool on Kali Linux was generated as a default file and used as a reference for subsequent tests without any modifications. The payload was then delivered to each of the virtual machines to assess the ability of modern antivirus software to detect it. It was found that all 6 tested software programs immediately detected the potential threat, making it an ideal object for testing other tools. In contrast, an online scan of the payload revealed that only 20 out of 26 available antivirus engines were able to detect the threat, suggesting that the remaining 6 engines may have poor protection and may not be widely used. The same generated payload was used in all test cases; however, each framework modified it in different combinations of evasion mechanisms.

In this way, we compared combinations of antiviruses evasion mechanisms and not the sophistication of the payload itself.

The decision to test free antivirus software was based on a study of nearly 2000 respondents who indicated that users prefer the basic version over the paid one [51]. The choice of antivirus software for testing was informed by a 2021 report from AV-Comparatives [50], which presented the best-performing antivirus programs according to their tests. To maintain confidentiality and prevent misuse by potential hackers, the author of this work uses the acronyms AV1-AV6 to refer to the specific antivirus programs tested. Whole topology, dependencies and connections are presented in Figure 2. All VMs were created on single computer and with a shared hub-like connection to ensure that there were no other security or networking mechanisms blocking malware, such as a firewall or misconfigured routing.

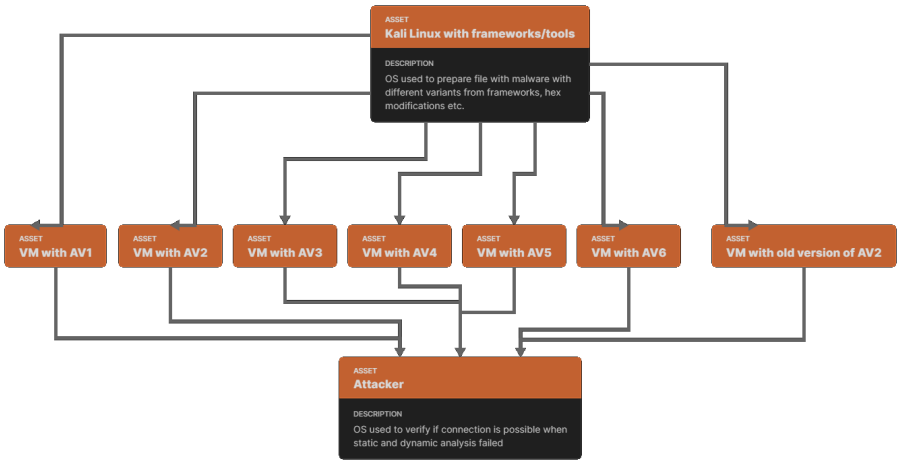


Figure 2. Dependencies between hosts in the created testing topology.

To begin the research, the latest version of the popular malware bypass tool, Hyperion, was downloaded and installed. The research consisted of three distinct parts. First, the payload generated previously was encrypted using Hyperion. The second part involved running the encrypted file in a hex editor to modify specific characters, changing commands to their direct equivalents rather than changing execution flow, in order to generate different hashes/signatures of the payload file. Finally, a new payload was generated using Msfvenom, this time utilizing the Shikata Ga Nai encoder on the payload. The already-encoded file was then subjected to Hyperion's encryption. In the end, all three infected files underwent antivirus scanning and online scanning to assess their effectiveness. Hyperion, being one of the oldest and most widely-used malware bypass tools, was chosen as the starting point for the research.

We used Y/N notation to represent whether the attack was detected (Y) or not (N). As presented in Table 1, the experimentation of expanding the basic payload with encryption or modifying the Hyperion code with a hex editor did not yield significant results for either static scan (S) or for run-time/active state (A). This is likely due to the fact that Hyperion, being an older software created in 2012, is relatively outdated compared to Msfvenom; therefore, similar results were expected. While obfuscation techniques were once effective, contemporary antivirus software has developed heuristic or behavioral detection capabilities that can identify even minor changes in the signature appearance. Notably, one antivirus engine was deceived during online scanning more than with the basic payload. This suggests that, while these basic mechanisms may not be effective against the best software on the market, they may have minimal impact on smaller software producers. In conclusion, it can be stated with confidence that none of the aforementioned tools are currently effective, thus leading to the abandonment of further experimentation with Hyperion.

Table 1. Hyperion results.

Payload	Online Scan	AV Results											
		AV1		AV2		AV3		AV4		AV5		AV6	
		S	A	S	A	S	A	S	A	S	A	S	A
Hyperion	19/26	Y		Y		Y		Y		Y		Y	
+SNG	19/26	Y		Y		Y		Y		Y		Y	
+Hex	19/26	Y		Y		Y		Y		Y		Y	

TheFatRat has been available for a considerable period of time and has a relatively large community, unlike Hyperion. The source code is accessible on GitHub and is regularly updated, which enhances its prospects of evading antivirus software. For the purpose of the study, the research focuses on the solutions that use the same payload as the one initially generated. TheFatRat uses Msfvenom to generate the reverse TCP payload, preserving the idea of the research. After installation and execution of TheFatRat in the console, the sixth option, PwnWinds, was selected from the menu to generate payloads for testing. The first was a file generated in the .bat extension based on PowerShell code, and the second was identical except for the selected port (all previous payloads were functioning on port 8080, whereas this one was set to port 4444). The option that generates a file with an .exe extension based on C# and PowerShell was then selected, followed by a file with an .exe extension based on the C language. To proceed, it was necessary to convert the .bat file to .exe using BatToExe Converter. After generating all the necessary files, the scanning process began.

The results presented in Table 2 are noteworthy. From the table, it is apparent that AV6 performed poorly in the comparison. It was unable to handle payloads generated using C and PowerShell languages. Interestingly, 8 months after the test, AV6 performance remained the same. However, it should be noted that only the static scan level was considered, and it was not possible to establish a meterpreter connection. Furthermore, the payload generated in pure C was also able to evade detection by AV6. A crucial observation is that a payload generated in the .bat extension using PowerShell was more likely to be detected when listening on port 8080 than on port 4444, which is the default port for attacks by various trojans. This finding may suggest that port 8080 was frequently used for attacks in the past, as it is the port commonly used by web browsers. As a result, processes launched by infected files may not have attracted much attention, and, consequently, antivirus developers may have been less attentive to this port in the past. However, half of the tested antiviruses did not detect the static load on port 4444 at the level of static scans.

Due to the longevity of research and the successful evasion of antiviruses, the authors decided to redo all tests after eight months, with updated version of antiviruses and their databases. The installed antivirus updates rendered all six tested antiviruses immune to the old threat. This indicates that antivirus updates are necessary and effective, as evidenced by the results of the online scan. From the results, it can be inferred that the antiviruses tested eight months ago were less effective than the newer ones, except for the generated .bat file that returned worse results in the second study, possibly due to a scan error or a problem with the software engine vendor. In conclusion, the only effective tool appears to be the payload generated in TheFatRat, based on the C language and PowerShell, but only when confronted with the AV6 antivirus.

In another study, we utilized Shellter software to inject a generated payload into a potentially safe program. The process began with downloading an appropriate program that would serve as the host for the payload injection. From a security point of view, the process should be not malicious. Standard processes without changes should not be flagged by antiviruses. The reason for utilizing Shellter was to change not-malicious process into malicious ones, hopefully also bypassing antiviruses. ColorPix was identified as a suitable program for this purpose, given its functionality as a color retrieval tool that does not

require installation. However, we should note that any file will render, in most modern cases, the same results, as the process is not tested but the structure is, which has changed from the antivirus point of view. In other words, the payload should be detected, not the transportation mechanism—in this case, usage of ColorPix. Using Shellter, the basic payload was successfully injected into the downloaded program. To evaluate the effectiveness of the injected payload, we subjected it to static scans by six different antivirus software programs. Surprisingly, as shown in Table 3, the injected payload was able to evade detection by five of the six tested antivirus software programs, with only AV5 exhibiting immunity to the injected payload. These results suggest that Shellter software can be an effective tool for evading antivirus software detection, at least at the static scan level.

Table 2. TheFatRat results.

Payload	Online Scan	AV Results											
		AV1		AV2		AV3		AV4		AV5		AV6	
		S	A	S	A	S	A	S	A	S	A	S	A
Results obtained in January 2022													
TheFatRat 62	13/26	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	**N**	**N**
ThaFatRat 66	10/26	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	**N**	Y
ThaFatRat 61-B	9/26	Y	Y	Y	Y	**N**	Y	Y	Y	**N**	Y	**N**	Y
TheFatRat 61	15/26	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Results obtained in September 2022													
TheFatRat 62	16/26	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	**N**	**N**
TheFatRat 66	14/26	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
TheFatRat 61-B	13/26	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
TheFatRat 61	14/26	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Note: 62 - C+PowerShell; 66 - C; 61 - bat+PowerShell; B - Port 4444.

The experiment was repeated using the same payload configuration but with a .raw extension. The results were even more alarming, with half of the tested antiviruses cheated at the static scanning level, and the other half completely fooled, allowing Shellter to establish a connection to the meterpreter console and run directly on the victim's machine. Subsequent testing after an eight-month interval, with updated software and databases, revealed that the antivirus software had improved in detecting the payload. Only AV4 at the static scan level and AV3 at the static scan level and launch level were deceived by the payload based on the .raw extension. Shellter has the potential to deceive signature-based scanning by injecting payloads into countless programs, with each program displaying a different injected payload. Additionally, it pays little attention to the processes it employs, meaning it is able to deceive most static scanners, and potentially can deceive behavioral or heuristic scans. The study demonstrates that Shellter is an effective tool, initially fooling half of the antivirus software tested, but only one after a few months. This underscores the importance of installing appropriate antivirus updates.

Veil-Evasion is a framework similar to TheFatRat software that provides numerous solutions for bypassing antivirus security. Veil-Evasion has a significant following on GitHub, but the framework has not been updated since 2020, which may suggest a lack of active development. To test Veil-Evasion's effectiveness, three variants of generated payloads using Msfvenom were selected for study. The Veil-Evasion study was divided into three parts, including testing the basic payload generated in Python, generating the payload based on PowerShell and Bash, and manually editing the previously-generated payload with a hex editor. The results, shown in Table 4, indicate that each tested antivirus was effective in threat detection. Differences were observed in online scans, where the Python-based payload was more frequently detected by antivirus scanners.

The other two payloads had similar results, but an interesting situation emerged in re-testing after eight months with up-to-date antiviruses. As expected, more scanners detected the threat, but during this period, the payload generated with PowerShell and Bash was caught by one more scanner than the one that was manually edited in the hex. This finding suggests that some antivirus software did not receive an update, or it may be a distortion from the online scan site.

Table 3. Shellter results.

Payload	Online Scan	AV Results											
		AV1		AV2		AV3		AV4		AV5		AV6	
		S	A	S	A	S	A	S	A	S	A	S	A
Results obtained in January 2022													
Shellter PP	5/26	**N**	Y	**N**	Y	**N**	Y	**N**	Y	Y	Y	**N**	**Y**
Shellter PPR	2/26	**N**	**N**	**N**	**N**	**N**	**N**	**N**	Y	**N**	Y	**N**	Y
Results obtained in September 2022													
Shellter PP	15/26	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Shellter PPR	3/26	Y	Y	Y	Y	**N**	**N**	**N**	Y	Y	Y	Y	Y

Note: PP—Primary payload, PPR—Primary payload with .raw extension.

Table 4. Veil-Evasion results.

Payload	Online Scan	AV Results											
		AV1		AV2		AV3		AV4		AV5		AV6	
		S	A	S	A	S	A	S	A	S	A	S	A
Results obtained in January 2022													
Python	13/26	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Powershell + Bash	10/26	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PS + Bash + Hex	10/26	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Results obtained in September 2022													
Python	14/26	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Powershell + Bash	13/26	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PS + Bash + Hex	12/26	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

In order to be comparable to previous research and to allow for other researchers to reconstruct the authors' tests, the criteria for assessing the effectiveness of the masking tool and antivirus software are similar to those described in previous research. Assigning penalty points for failure to detect malware in static and dynamic scan is a state-of-the-art methodology and thus was used in the research. Instead of using quality measurements or other kinds of rankings, penalty points are direct values that can be compared with past research and can be used by other researchers or in future work. If the antivirus does not detect the malware during the static scan level, it receives one penalty point. If the antivirus does not detect the malware during the static and dynamic scan levels, it receives two penalty points. Similarly, if the payload is not detected during the static scan level, it receives one point, but if it completely bypasses antivirus security and connects to the meterpreter session, it receives two points. The more points the antivirus software receives, the less effective it is. Alternatively, the more points the masking technique obtains, the more dangerous it is.

Table 5 displays the effectiveness of the tested tools, with Shellter ranking as the most dangerous tool in the first two positions: first with a raw load and second with a basic load. TheFatRat with the payload generated using the C language and PowerShell remains in third place, while the other tested mechanisms appear to pose only a slight threat. It

should be noted that the tested tools offer many more possibilities and modifications, but this study focuses on the simplest and most accessible solutions available on the web. This highlights the fact that even with limited knowledge, a potential attacker has all the tools necessary to threaten an unsecured user.

Table 5. Summary results for evasion tools and their combinations.

	Static	Runtime (active)	Summary
Basic payload	0	0	0
Hyperion	0	0	0
Hyperion + SNG	0	0	0
Hyperion + SNG + Hex	0	0	0
TheFatRat 62	0	1	**2**
TheFatRat 66	1	0	**1**
TheFatRat 61-B	3	0	**3**
TheFatRat 61	0	0	0
Shellter PP	5	0	**5**
Shellter PPR	3	3	**9**
V-E Python	0	0	0
V-E PS + Bash	0	0	0
V-E PS + Bash + Hex	0	0	0

To complete the study, older versions of the AV2 antivirus software were tested. Versions 2008 and 2016 were obtained from the website oldversions.com and installed on virtual machines. After installation, infected files were sent to the virtual machines, and the malware scanning process was initiated. The results, as shown in Table 6, indicate that the older antivirus software was unable to detect the threat, even in the case of the basic payload. This is expected, as Msfvenom was developed much later. However, AV2 from 2016 was able to detect only basic threats, which is consistent with the fact that all payloads that were not identified either appeared after 2016 or had multiple updates after that year.

Table 6. Old versions of AV2 results.

	AV2—Version from 2008		**AV2—Version from 2016**	
	S	**A**	**S**	**A**
Basic payload	**N**	**N**	Y	Y
Hyperion	**N**	**N**	Y	Y
Hyperion + SNG	**N**	**N**	Y	Y
Hyperion + SNG + Hex	**N**	**N**	Y	Y
TheFatRat 62	**N**	**N**	**N**	**N**
TheFatRat 66	**N**	**N**	**N**	**N**
TheFatRat 61-B	**N**	**N**	**N**	**N**
TheFatRat 61	**N**	**N**	**N**	**N**
Shellter PP	**N**	**N**	**N**	**N**
Shellter PPR	**N**	**N**	**N**	**N**
V-E Python	**N**	**N**	**N**	**N**
V-E PS + Bash	**N**	**N**	**N**	**N**
V-E PS + Bash + Hex	**N**	**N**	**N**	**N**

7. Discussion

This research has demonstrated that there are readily-available software tools that can be used to mask malicious files, effectively bypassing antivirus security regardless of the level of advancement of the attacker. The effectiveness of these tools depends on several factors, including the popularity of the tool, time, and how the tool works. When a given mechanism is frequently used, identical versions of malware may have been generated before, making the file useless if an antivirus engine detects it. The effectiveness of the software is also influenced by time, as antivirus software often introduces updates related to malware signatures, along with successive patches to fill defense gaps. This forces malware developers to update their programs with newer and newer solutions. The most important factor in the effectiveness of the cloaking mechanism is how it works. Changes in the hex editor or the use of an encoder to obfuscate the code are insufficient in the face of advanced heuristic searches combined with signature-based searches. Among the programs tested, Shellter was found to be the most concerning, as it can create a stable connection of meterpreter sessions on half of the tested antiviruses. The effectiveness of Shellter is also influenced by the injected payload, as the basic payload could bypass most antiviruses, but only on a static level, whereas the payload with the .raw extension could bypass all of them, half of which could be bypassed completely.

8. Conclusions

The study has demonstrated that, regardless of one's knowledge of cybersecurity, every individual is capable of infecting another person's device with varying degrees of success to steal or destroy valuable data. The tests conducted in this study employed a payload that establishes a connection between the victim's machine and the attacker's machine through a meterpreter session. This section of the study concentrates on several threats and commands that a hacker could utilize to launch an attack on someone else's machine. The objective of presenting these threats is to raise awareness of the extent of the threat posed by weak security and how much damage a hacker can inflict on a victim's machine.

9. Future Work

The study of bypassing antivirus mechanisms is an ever-evolving field, with new threats emerging daily and with antivirus software continuously attempting to thwart them. Further research is necessary to improve internet security by exploring more advanced techniques for bypassing antivirus software. The techniques examined in this study were either default ones or minimally modified, and there are many untested and lesser-known tools available on GitHub that could exploit previously-unknown vulnerabilities. Investigating how antivirus masking techniques function on file types other than payloads may also yield different results depending on the malware's triggered processes. While this study focused on Windows, attacks on other systems such as MacOS or Android may produce completely different outcomes. It is important to remember that, in this research, the performance of current computers is powerful and should not be a problem for modern antiviruses. However, in personal and/or Internet of Things devices, the relationship among security, usability, and performance should also be taken into consideration in subsequent research. As smartphones have become ubiquitous and are used for a variety of sensitive activities such as taking pictures, making calls, and conducting financial transactions, it is crucial to maintain the highest level of protection against potential hacker attacks.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Not applicable

Data Availability Statement: Test results are not shared due to ethical restrication/misuse.

Acknowledgments: This research was funded by project SUBB-MN/BKM 2022, grant number 02/120/BKM22/0019.

Conflicts of Interest: The author declares no conflict of interest.

References

1. AV-Test Malware. 2022. Available online: https://www.av-test.org/en/statistics/malware/ (accessed on 15 April 2023).
2. Sharif, M.I.; Lanzi, A.; Giffin, J.T.; Lee, W. Impeding Malware Analysis Using Conditional Code Obfuscation. In Proceedings of the 4th Virus Bulletin International Conference, San Diego, CA, USA, 10 February–13 February 2008.
3. Kalogranis, C. AntiVirus Software Evasion: An Evaluation of the AV Evasion Tools. Ph.D. Thesis, University of Piraeus Department of Digital Systems, Piraeus, Greece, 2018.
4. Aminu, S.A.; Sufyanu, Z.; Sani, T.; Idris, A. Evaluating the effectiveness of antivirus evasion tools against windows platform. *J. Sci.* **2019**, *4*, 112–119.
5. Panagopoulos, I. Antivirus Evasion Methods. Ph.D. Thesis, University of Piraeus Department of Digital Systems, Piraeus, Greece, 2020.
6. Garba, F.A.; Yarima, F.U.; Kunya, K.I.; Abdullahi, F.U.; Bello, A.A.; Abba, A.; Musa, A.L. Evaluating Antivirus Evasion Tools Against Bitdefender Antivirus. In Proceedings of the International Conference on FINTECH Opportunities and Challenges, Karachi, Pakistan, 18 October–19 October 2021.
7. Collberg, C.; Thomborson, C.; Low, D. A taxonomy of obfuscating transformations. *ACM Comput. Surv.* **1997**, *29*, 1–69.
8. Chen, Y.; Liu, Y.; Chen, Y. A survey on android malware obfuscation. *J. Comput. Sci. Technol.* **2016**, *31*, 901–921.
9. F-Secure. State of Cyber Security 2018. 2018. Available online: https://www.f-secure.com/documents/996508/1030743/F-Secure_State_of_Cyber_Security_2018.pdf (accessed on 15 April 2023).
10. Mandiant. M-Trends 2018: A View from the Front Lines. 2018. Available online: https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-2018-mtrends.pdf (accessed on 15 April 2023).
11. Christodorescu, M.; Jha, S.; Seshia, S.A. Static analysis of executables to detect malicious patterns. In Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS'05), San Diego, CA, USA, 27 February–3 March 2005; pp. 1–14.
12. Moser, A.; Kruegel, C.; Kirda, E. Limits of static analysis for malware detection. In Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC'07), Miami Beach, FL, USA, 10–14 December 2007; pp. 421–430.
13. Ligh, M.; Adair, S.; Hartstein, B.; Richard, M. *Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*; John Wiley & Sons: Hoboken, NJ, USA, 2014.
14. Bishop, M.; Gates, C. Combating virtual machine detection and fingerprinting. *IEEE Secur. Priv.* **2009**, *7*, 56–60.
15. Kephart, J.O. Automatic Extraction of Computer Virus Signatures. In Proceedings of the 4th Virus Bulletin International Conference, Abingdon, UK, 1994; pp. 179–194.
16. Zemlyanaya, D.; Boldyrikhin, N.; Svizhenko, A.; Yukhnov, B. Virus signature detection algorithm. *Intell. Inf. Technol. Math. Model.* **2021**, *2131*, 022086. [CrossRef]
17. Cohen, F. Computer Viruses: Theory and Experiments. *Comput. Secur.* **1987**, *6*, 22–35. [CrossRef]
18. Bazrafshan, Z.; Hashemi, H.; Fard, S.M.H.; Hamzeh, A. A Survey on Heuristic Malware Detection Techniques. In Proceedings of the Conference on information and Knowledge Technology, Shiraz, Iran, 28–30 May 2013; pp. 113–120.
19. Cohen, F. Computer Viruses. Ph.D. Thesis, University of South California, Los Angeles, CA, USA, 1986.
20. Gryaznov, D. Scanners of The Year 2000: Heuristics. In Proceedings of the Fifth International Virus Bulletin Conference, Boston, MA, USA, September 1999; pp. 225–234.
21. Kaspersky. usa.kaspersky.com. Available online: https://usa.kaspersky.com/resource-center/definitions/heuristic-analysis (accessed on 15 April 2023).
22. Wressnegger, C.; Freeman, K.; Yamaguchi, F.; Rieck, K. From Malware Signatures to Anti-Virus Assisted Attacks. *arXiv* **2016**, arXiv:1610.06022.
23. Wressnegger, C.; Freeman, K.; Yamaguchi, F.; Rieck, K. Automatically inferring malware signatures for anti-virus assisted attacks. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, Abu Dhabi, United Arab Emirates, 2–6 April 2017; pp. 587–598.
24. Balakrishnan, A.; Schulze, C. Code Obfuscation Literature Survey. *CS701 Constr. Compil.* **2005**, *19*, 31. Available online: https://pages.cs.wisc.edu/~arinib/writeup.pdf (accessed on 15 April 2023).
25. Kaushik, K.; Sandhu, H.S.; Gupta, N.K.; Sharma, N.; Tanwar, R. A systematic approach for evading antiviruses using malware obfuscation. In *Emergent Converging Technologies and Biomedical Systems: Select Proceedings of ETBS 2021*; Springer: Singapore, 2021; pp. 29–37.
26. Ntantogian, C.; Poulios, G.; Karopoulos, G.; Xenakis, C. Transforming malicious code to ROP gadgets for antivirus evasion. *IET Inf. Secur.* **2019**, *13*, 570–578. [CrossRef]
27. Rad, B.B.; Masrom, M.; Ibrahim, S. Camouflage in Malware: From Encryption to Metamorphism. *Int. J. Comput. Sci. Netw. Secur.* **2012**, *12*, 74–83.

28. Bernardinetti, G.; Di Cristofaro, D.; Bianchi, G. PEzoNG: Advanced Packer For Automated Evasion On Windows. *J. Comput. Virol. Hacking Tech.* **2022**, *18*, 315–331. [CrossRef]
29. Scaife, N.; Carter, H.; Traynor, P.; Butler, K.R. Cryptolock (and drop it): Stopping ransomware attacks on user data. In Proceedings of the 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), Nara, Japan, 27–30 June 2016; pp. 303–312.
30. Nicula, S.S. Generating antivirus evasive executables using code mutation. *Inform. Econ.* **2018**, *22*, 81–86. [CrossRef]
31. Redcanary.com. Process Injection. 2022. Available online: https://redcanary.com/threatdetection-report/techniques/process-injection/ (accessed on 15 April 2023).
32. Ashkan Hosseini. Ten Process Injection Techniques. 2017. Available online: https://www.elastic.co/blog/ten-process-injection-techniques-technical-surveycommon-and-trending-process (accessed on 15 April 2023).
33. Sanjay, B.N.; Rakshith, D.C.; Akash, R.B.; Hegde, V.V. An approach to detect fileless malware and defend its evasive mechanisms. In Proceedings of the 2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS), Bengaluru, India, 20–22 December 2018; pp. 234–239.
34. D'Elia, D.C.; Invidia, L.; Querzoni, L. Rope: Covert multi-process malware execution with return-oriented programming. In Proceedings of the Computer Security–ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, 4–8 October 2021; Proceedings, Part I 26; Springer International Publishing: Berlin, Germany 2021; pp. 197–217.
35. Kali Linux Team. Kali Docs. 2022. Available online: https://www.kali.org/docs/ (accessed on 15 April 2023).
36. G0tmi1k. What is Kali Linux. 2022. Available online: https://www.kali.org/docs/introduction/what-is-kali-linux/ (accessed on 15 April 2023).
37. Rapid7. Metasploit. 2022. Available online: https://metasploit.com/ (accessed on 15 April 2023).
38. Rapid7. Msfvenom. 2021. Available online: https://github.com/rapid7/metasploit-framework/blob/master/msfvenom (accessed on 15 April 2023).
39. Rapid7. Shikata Ga Nai Repo. 2017. Available online: https://github.com/rapid7/metasploitframework/blob/master/modules/encoders/x86/shikata_ga_nai.rb (accessed on 15 April 2023).
40. Miller, S.; Reese, E.; Carr, N. Shikata Ga Nai Encoder Still Going Strong. 2019. Available online: https://www.mandiant.com/resources/shikata-ga-nai-encoderstill-going-strong (accessed on 15 April 2023).
41. Belial. Hyperion. 2020. Available online: https://nullsecurity.net/tools/binary.html (accessed on 15 April 2023).
42. Wine Team. WineHQ. 2022. Available online: https://www.winehq.org/ (accessed on 15 April 2023).
43. Christian Ammann. Hyperion: Implementation of a PE-Crypter. 2012. Available online: https://www.exploit-db.com/docs/english/18849-hyperion-implementation-ofa-pe-crypter.pdf (accessed on 15 April 2023).
44. Edo Maland. TheFatRat. 2021. Available online: https://github.com/screetsec/TheFatRat (accessed on 15 April 2023).
45. BlackHat. TheFatRat. 2020. Available online: https://www.blackhatethicalhacking.com/tools/thefatrat/ (accessed on 15 April 2023).
46. kyREcon. Shellter. 2022. Available online: https://www.shellterproject.com/introducing-shellter/ (accessed on 15 April 2023).
47. kyREcon. Shellter Readme. 2022. Available online: https://www.shellterproject.com/Downloads/Shellter/Readme.txt (accessed on 15 April 2023).
48. VeilEvasion Team. Veil. 2020. Available online: https://github.com/Veil-Framework/Veil (accessed on 15 April 2023).
49. Srinivas. Antivirus Evasion Tools. 2021. Available online: https://resources.infosecinstitute.com/topic/antivirus-evasion-tools/ (accessed on 15 April 2023).
50. AV-Comparatives. Summary Report 2021. 2021. Available online: https://www.av-comparatives.org/tests/summary-report-2021/ (accessed on 15 April 2023).
51. Statista Research Department. Which Version of Antivirus Program Do You Use on Personal Devices? 2021. Available online: https://www.statista.com/statistics/1252587/free-vs-paid-antivirus-usage-by-brand-russia/ (accessed on 15 April 2023).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.