Antivirus Performance Evaluation against PowerShell Obfuscated Malware

Radostin Dimov
Artillery, AD and CIS Faculty,
National Military University „V. Levski",
Shumen, Bulgaria
rsdimov95@gmail.com

Zhaneta Savova
Artillery, AD and CIS Faculty,
National Military University „V. Levski",
Shumen, Bulgaria
zh.savova@yahoo.com

Abstract. **In recent years, malware attacks have become increasingly sophisticated, and the methods used by attackers to evade Windows defenses have grown more complex. As a result, detecting and defending against these attacks has become an ever more pressing challenge for security professionals. Despite significant efforts to improve Windows security, attackers continue to find new ways to bypass these defenses and infiltrate systems. The techniques covered in this paper are all currently active and effective at evading Windows defenses. Our findings underscore the need for continued vigilance and the importance of staying up to date with the latest threats and countermeasures.**

Keywords: AMSI Evasion, Antivirus bypass, Defense Evasion, EDR Evasion, PowerShell Obfuscation, Undetected Payload

I. Introduction

With the growing complexity of cybersecurity threats, it is becoming increasingly important to secure computer systems against malicious attacks. One of the most commonly targeted operating systems is Microsoft Windows. As a result, many organizations and security professionals are deploying various defensive measures to protect their systems from malware attacks.

PowerShell (PS) is a powerful scripting language that comes built into Microsoft Windows, making it a popular choice for both defenders and attackers. While PS can be used to implement defensive measures, attackers also leverage its capabilities to evade defenses and compromise Windows systems. PS scripts are highly visible and can be easily detected by antivirus software, but attackers can use obfuscation techniques to hide their scripts and make them more difficult to detect.

This paper examines various PS obfuscation techniques that attackers may use or combine as methods to bypass commonly used antiviruses (AVs) and evade detection. By understanding these techniques, defenders can better protect their systems against malicious attacks that utilize PS. This paper explores how these techniques work and why they are effective. It also provides readers with a comprehensive understanding of the risks associated with PS scripts and the methods that attackers use to evade windows defenses.

The analysis presented in this paper is based on research and testing, as well as real-world observations of attacks using these techniques. By highlighting these techniques, we hope to contribute to the ongoing effort to improve Windows security and protect against malicious attacks.

The research objective is to evaluate the performance of Antimalware Scan Interface (AMSI) and twelve different AV software against obfuscated PS payloads.

A. AV Detection Methods

AV software uses a variety of techniques to detect malware:

Signature-based detection - involves scanning files for known malware signatures [1]. When the AV software encounters a file that matches a known signature, it will quarantine or delete the file [2]. However, this kind of detection has some limitations as it is ineffective against new and unknown threats.

Heuristic-based detection – relies on set of rules for analyzing the file to determine whether it is malicious or not [3]. This can include looking for specific patterns in the code or program calls.

Behavior-based detection – involves analyzing the behavior of processes running on a system to detect malicious activity [4]. The AV software monitors the system for suspicious behavior such as the process attempting to communicate with a known malicious IP address or downloading stage from a remote host. If a process exhibits such behavior, it may be flagged as malware [5].

Sandbox Detection – part of behavior-based detection which involves running a file or process in a controlled environment to observe its behavior. The AV software

Print ISSN 1691-5402
Online ISSN 2256-070X
https://doi.org/10.17770/etr2024vol4.8201

creates a virtual sandbox where the file can be executed safely without affecting the rest of the system [6]. By observing the behavior of the file within the sandbox, the AV software can determine whether it is malicious or not [7].

However, in general the AV software is integrated with AMSI which adds another layer of security. If a PS payload performs to successfully bypass the AV static signature detection, it has to handle with the AMSI runtime detection [8]. Explaining how AMSI works and its integration with the AV software is beyond the scope of this research.

B. AV Evasion key techniques

There are various techniques which can be utilized to bypass different AV solution, but in general we can classify them as follows:

Encoding – technique used to hide the true nature of the malware code from antivirus software. By transforming the code into a different format using a scheme, such as base64 or hexadecimal, the malware can bypass signature-based detection. However, encoding is a reversible process, and this technique is becoming less effective as antivirus software improves its ability to detect and decode encoded malware.

Encryption – uses encryption algorithms such as XOR or Advanced Encryption Standard (AES) to encrypt the payload. After execution the encrypted code is decrypted in memory [9].

Obfuscation – consists of sub techniques for modifying the code of the malware to change its signatures and make it more difficult to detect [10]. This includes modifying/reorganizing the source code, object concatenation, splitting and merging techniques so the new relevant signatures are not flagged as malicious [11].

Packers – tools used by attackers to compress and encrypt executable files to make them harder to analyze by security tools and detect malware. These tools are used to evade detection by antivirus software and other security tools. Packed executables are unpacked at runtime, making it harder for security tools to detect and analyze the original code [12].

Reflective Code Loading – technique used to load code directly into a target process's memory, without creating any files on the disk. Commonly used by stager payloads for in-memory code execution. This allows the malware to evade detection by traditional AV software, which often relies on scanning for malicious files or processes [13].

Sandbox Evasion – techniques used to avoid detection when running in a sandbox (virtualization) environment such as time-based evasion or system checks.

C. Review of related works

In 2018, Jagsir Singh and Jaswinder Singh [14] have analyzed various obfuscation techniques including code replacement, code reorganization, packing, renaming and encryption. The research also reviewed some of the AV detection mechanisms and highlighted effective countermeasures to detect malware obfuscation techniques.

Another research conducted by Kalogranis [15] evaluated four tools, namely AVET (Antivirus Evasion Tool), peCloack.py, Shellter, and Veil-Evasion, against five of the most popular AV solutions – Avast, Bitdefender, ESET Nod32, McAfee and Avira. The AV products selection was based on the products' market share at that time. The research demonstrated that AVET and Veil Evasion had the best performance.

In 2019, a group of authors evaluated the effectiveness of AV evasion tools against windows platform extending Kalogranis' work in a subsequent research [16]. The authors added the Metasploit payload generator and a new tool – TheFatRat, repeating the same tests used by Kalogranis. In comparison to Kalogrins' research, the results showed that AVET and peCloack.py achieved the best effectiveness against the tested AVs. Of course, we have to keep in mind that some of the tools are still in progress and are updated continuously while the tests were performed in 2019.

Similarly, in [17], the author utilized Metasploit payload generator, Hyperion, TheFatRat, Veil-Evasion and Shellter against six AV platforms. The researcher used and combined multiple techniques during the tests. The results highlight Shellter as the most dangerous tool followed by TheFatRat.

Evaluation of Bitdefender AV against different evasion tools was conducted in [18]. The authors analyzed the mentioned AV as one of the best AV platforms and decided to evaluate the effectiveness of nine different open-source tools against only this AV software. The results showed that Phantom Evasion, Onelinepy and PayGen have the highest percentage evasion score against Bitdefender.

In [19] the authors presented a new packer product – PEzoNG which is successor of PEzor – an existing open-source PE and shellcode packer. However, authors mentioned that PEzoNG is a completely different project from PEzor as they only share a part of the name and the building environment. The framework automates the process of creating undetectable binaries targeting Windows Environment. The new product features custom loader, polymorphic obfuscation, anti-sandbox and anti-analysis evasion mechanisms. The effectiveness of the framework for AV detection is tested against 29 different AV solutions and the product is compared to other similar tools.

Even though several studies have evaluated the defense evasion performance of automated tools that may utilize PS as a feature, this research highlights manual evasion methods which allows attackers to personalize their techniques to the target environment and evade specific detection methods. Manual obfuscation doesn't rely on obfuscation algorithms and adversaries can customize the malware manually, making it harder to detect. Adversaries can use a range of techniques, such as renaming variables, splitting code into multiple functions, adding unnecessary code, and encoding or encrypting the code, to make it more difficult to analyze.

II. MATHERIALS AND METHODS

In this research, a virtual lab is developed using VMware ESXi virtualization software to conduct experiments that evaluate the detection capabilities of

AMSI and twelve different AV software against PS defense evasion techniques. By using a virtual lab, we can simulate real-world attack scenarios and assess the performance of AV software against modern cyber threats. The virtual environment - Fig. 1 consists of 13 virtual machines: a Kali Linux 2023.1 attacker box and twelve fully updated Windows Server 2022 sandboxes each running different AV platform with enabled AMSI services. All virtual machines are connected in a separate subnetwork with an IPv4 address range of 192.168.64.0/24 with the attacker box located at 192.168.64.128/24 and the sandboxes at 192.168.64.131-142/24. The attacker box has an opened Netcat listener on TCP port 4444, which will wait for TCP reverse shell connections while testing the AVs detection capabilities against different obfuscation techniques.

The AV platforms that have been selected have demonstrated their exceptional detection capabilities over the years. These platforms have consistently provided accurate and reliable protection against various types of threats. The utilized AV programs have been rigorously evaluated and have proven their effectiveness in detecting and mitigating known and emerging threats. Additionally, the platforms have received numerous accolades and recognition from reputable organizations in the cybersecurity industry. Their track record of success and continuous improvement make them a reliable and trustworthy choice for protecting against evolving cyber threats.

For the research objective an initial standard PS reverse shell one-liner payload on Fig.2 is used developed by Nikhil Mittal [20]. The payload is then obfuscated with different techniques (Fig. 3) and distributed to the AV sandboxes.

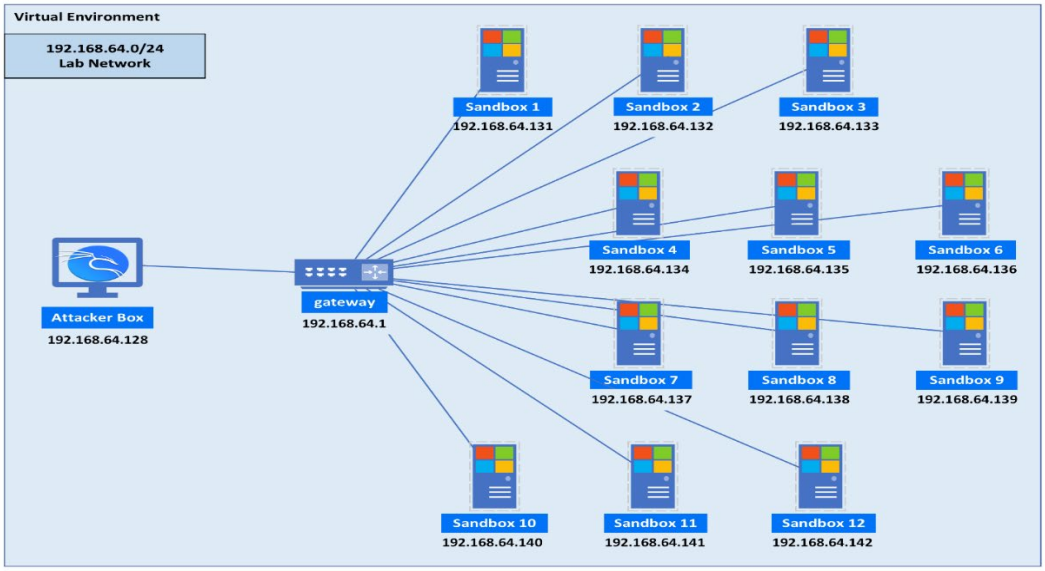

Fig. 1. Experimental network map

```
$client = New-Object System.Net.Sockets.TCPClient('192.168.64.128',4444); $stream
= $client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i =
$stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName
System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data 2>&1 |
Out-String );$sendback2 = $sendback + 'PS ' + (pwd).Path + '> ';$sendbyte =
([text.encoding]::ASCII).GetBytes($sendback2);
$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush()};$client.Close()
```

Fig. 2. Initial PowerShell Reverse Shell script [20]

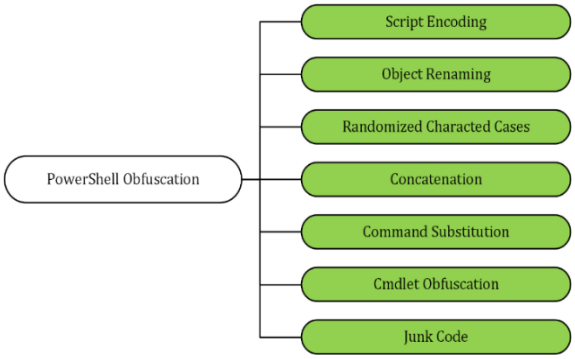

Fig. 3. PowerShell Obfuscation techniques taxonomy

However, as mentioned, AMSI is usually integrated with the AV program which means that we have an additional runtime security layer provided by AMSI. The experimental procedure follows the flowchart on Fig. 4.

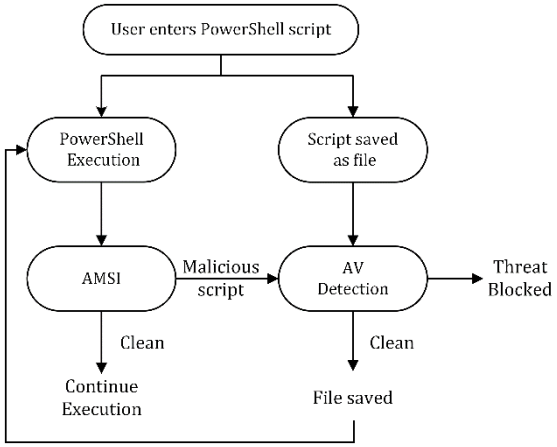

Fig. 4. PowerShell execution flowchart

In the flowchart, there are two paths:

1. If the user enters the PS script directly, it is executed, and the AMSI scans it for malicious code. If the script is not detected as malicious, PS continues the execution. If AMSI detect the code as malicious, an alert is passed through the AV software and script execution is blocked.

2. If the user saves the PS script as a file on the disk, the AV will compare the content of the script file to a predefined database of signatures to identify known malware. If a match is found, the AV software will generate an alert and quarantine or remove the file. If no match is found, the file will remain on the filesystem.

The proposed method can be useful in evaluating the effectiveness of PS manual obfuscation against AMSI runtime detection and AV software for improving their capabilities to detect and prevent attacks that use defense evasion techniques.

Overall, our approach provides a controlled environment for testing AV software and enables us to evaluate their detection capabilities against real-world threats. The findings of this research may contribute to enhancing the effectiveness of AMSI and AV software in protecting against modern cyber threats.

III. Results and Discussion

If we try to save the script on Fig. 2 on the filesystem as PS script file with .ps1 extension, it will be immediately flagged as malicious by Windows Defender real-time protection. Real-time protection performs a static signature scan against every new file saved in the filesystem. On table 1 are shown the AVs detection results of the tested script file.

TABLE 1 INITIAL REVERSE SHELL DETECTION RESULTS

№	AV Software	**Detection**
Results collected November 2023		**Static Detection**
1	Microsoft Defender	Detected
2	Avast Antivirus	Detected
3	AVG Anti-Virus	Detected
4	Avira Antivirus	Undetected
5	Bitdefender Total Security	Detected
6	ESET NOD32 Antivirus	Detected
7	Fortinet Antivirus	Undetected
8	Kaspersky Internet Security	Detected
9	McAfee Endpoint Protection	Detected
10	Sophos	Detected
11	Malwarebytes	Undetected
12	Symantec	Detected
Results collected November 2023		**Runtime Detection**
1	AMSI	Detected

The script is saved as PS file with .ps1 extension and distributed to the AV sandboxes. The results show that the script file is detected by most of the AVs (9/12) either by signature-based detection or heuristic detection. Also, it is detected when executed directly in PS by AMSI. As the script is quite popular and already known to most of the AVs providers, it is also detected as malicious by AMSI. The next experiments perform obfuscation techniques to evaluate the AVs performance and their detection capabilities against PS payload.

A. Encoding (EN)

The proposed experiment aims to evaluate the AVs static signature detection capabilities against encoded PS payloads. The content of the PS Script file – Fig. 2 is base64 encoded with a fixed number of iterations. The encoded payload is then distributed to the AVs sandboxes. After execution the script is decoded in memory. The experiment is repeated with 1, 5, and 10 iterations of base64 encoding the fig. 2 code. The results are presented in table 2. With 10 iterations of encoding, we managed to break 11 of 12 AVs static signatures detection, but not AMSI runtime detection.

As was mentioned earlier encoding is a reversible process. In this example when the encoded script is passed to PS, it is first decoded from base64 and then executed which triggers AMSI runtime detection [21]. An efficient way to bypass AMSI is to encode the strings and decode them within the code [22]. Fig. 5 and 6 shows a brief example where the first command is detected by AMSI as malicious while the encoded one remains undetected.

TABLE 2 BASE64 ENCODING DETECTION RESULTS

№	**AV**	**i=1**	**i=5**	**i=10**
Results collected November 2023			**Static Detection**	
1	Defender	Detected	Detected	Undetected
2	Avast	Undetected	Undetected	Undetected
3	AVG	Undetected	Undetected	Undetected
4	Avira	Undetected	Undetected	Undetected
5	Bitdefender	Detected	Detected	Undetected
6	NOD32	Undetected	Undetected	Undetected
7	Fortinet	Undetected	Undetected	Undetected
8	Kaspersky	Undetected	Undetected	Undetected
9	McAfee	Undetected	Undetected	Undetected
10	Sophos	Detected	Detected	Detected
11	Malwarebytes	Undetected	Undetected	Undetected
12	Symantec	Undetected	Undetected	Undetected
Results collected November 2023			**Runtime Detection**	
1	AMSI	Detected	Detected	Detected

```
"Invoke-Mimikatz"
# Can be encoded as:
[System.Text.Encoding]::Unicode.GetSt
ring([System.Convert]::FromBase64Strin
g('SQBuAHYAbwBrAGUALQBNAGkAbQBpAGsAY
QB0AHoA'))
```

Fig. 5. Encoding Commands

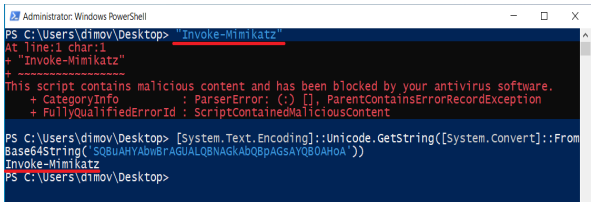

Fig. 6. AMSI Evasion with string encoding

Overall, using base64 encoding, the proposed experiment performed to successfully bypass AVs static signatures detection and provides an effective approach to evade AMSI detection.

B. Object Renaming (OR)

A technique which can be utilized for breaking signatures detection and involves changing the names of variables/functions/classes in the code, without changing the functionality of the code itself. This can be done manually or by using tools that automatically rename the objects. Fig. 7 is an example of renaming the variables within the initial PS code on Fig. 2 with low entropy string values. As the script on Fig. 2 contains 8 variables, the experimental results on table 3 evaluate AVs detection capabilities against this technique depending on the number of substituted variables (n).

```
$client          $aaaaaaaaaaaaaaaaaaaa
$stream          $bbbbbbbbbbbbbbbbbbbb
$bytes           $ccccccccccccccccccc
$i               $ddddddddddddddddddddd
$data            $eeeeeeeeeeeeeeeeeeee
$sendback        $fffffffffffffffffff
$sendback2       $ggggggggggggggggggg
$sendbyte        $hhhhhhhhhhhhhhhhhhh
a) Variables     b) Renamed Examples
```

Fig. 7. Renaming Variables

Substituting all variables within the script results in evading 7 of 12 AVs detection. This technique is also valid for other programming languages such as python, C# or C++. By renaming variables, the malware author can change the "signature" of the code, making it more difficult for AV software to detect. However, AMSI detection is still present. In other words, the script can be saved on the filesystem, but after execution it is detected as malicious by AMSI.

TABLE 3 OBJECT RENAMING DETECTION RESULTS

№	AV Software	n=1	n=4	n=8
Results collected November 2023		**Static Detection**		
1	Defender	Detected	Undetected	Undetected
2	Avast	Detected	Detected	Undetected
3	AVG	Detected	Detected	Undetected
4	Avira	Undetected	Undetected	Undetected
5	Bitdefender	Detected	Detected	Detected
6	NOD32	Detected	Detected	Detected
7	Fortinet	Undetected	Undetected	Undetected
8	Kaspersky	Detected	Detected	Detected
9	McAfee	Detected	Undetected	Undetected
10	Sophos	Detected	Detected	Detected
11	Malwarebytes	Undetected	Undetected	Undetected
12	Symantec	Detected	Detected	Detected
Results collected November 2023		**Runtime Detection**		
1	AMSI	Detected	Detected	Detected

By splitting the script and performing part code execution, we see that the last part of the code – "*$Client.Close()*" is triggering the AMSI detection. This doesn't mean that this part of the code is malicious itself, but combined with all the other parts of the script leads to a malicious result.

C. Randomize character cases (RC)

This leverages the fact that PS is case-insensitive, meaning that the casing of commands, variables, and arguments does not affect their execution. Exploiting this characteristic, obfuscators introduce variations in character capitalization throughout the code, making it visually distinct from the original form. This alteration disrupts simple string-based pattern matching techniques, as the obfuscated code no longer matches known signatures or standard conventions. The obfuscated code can feature a range of randomizations, including uppercase-to-lowercase, lowercase-to-uppercase, or even selectively mixing capitalization within words or commands [23]. These modifications are applied to specific characters, leaving the overall structure and functionality of the code intact. The goal is to create a visually jumbled representation that evades detection algorithms and human analysis, while still allowing the interpreter to execute the malicious instructions correctly. Fig. 8 represents an example of showcasing the randomization of char cases within a PS code.

```
IEX Get-Process    iEx "GeT-PrOcEsS"
$var1="example"    $vAr1="eXaMplE"
a)Original code    b)Obfuscated code
```

Fig. 8. Char cases randomization

This method is implemented into the initial PS script code – Fig. 2. Then the updated script is distributed to the AVs sandboxes. The results are shown on table 4. It may look simply but the detection results represent how powerful this technique could be in breaking AVs signatures.

D. Concatenation (CC)

Concatenation is the process of combining multiple strings or variables into a single string. This operation is frequently used in PS scripts to create more complex and meaningful output, for example, to construct a custom error message, generate a file path or URL, or format text for display. There are several ways to concatenate strings in PS, including the use of the "+" operator, the "-join" operator, and the string interpolation feature [24]. An example of concatenating strings is shown on Fig. 9.

```
$string='192.168.1.121'
$string='192.16'+'8.1.121'
$string='192.1'+'68.1'+'.121'
$string='19'+'2.16'+'8.1.1'+'21'
```

Fig. 9. String Concatenation

The fourth line of code for example creates a new string by concatenating four separate string literals, '19', '2.16', '8.1.1', and '21'. After the fourth line of code runs, the original value of the $string variable, '192.168.1.121', is replaced with the concatenated string. The resulting value of $string is again '192.168.1.121'. A simple concatenation as shown on Fig. 9 is applied into the

strings of the initial script – Fig. 2. AVs detection results for the discussed method are presented in table 4.

E. Commands Substitution (CS)

Substituting commands with similar ones that have the same functionality can be used as a technique by malware authors to evade detection by AV systems. By replacing suspicious or known malicious commands with benign or less detectable alternatives, malware can bypass signature-based detection mechanisms and appear innocuous to security software. This technique takes advantage of the vast number of available PS cmdlets and functions that provide similar functionality but have different names or syntax. By using these alternative commands, malware authors can camouflage their malicious activities and make the code less recognizable to AV engines. Fig. 10 provides an example approach in substituting *pwd* cmdlet. This technique allows malware to evade signature-based detections, as the substituted commands do not match known malicious patterns.

```
# Utilizing .NET Framework
[System.IO.Directory]::GetCurrentDire
ctory()
# Using different cmdlet
Get-location
# using Get-location Alias
gl
```

Fig. 10. pwd cmdlet substitution example methods

F. Cmdlet obfuscation (CO)

In PS, cmdlets can be obfuscated by adding single or double quotes between the characters. The cmdlets can be broken up into multiple segments, and single/double quotes are added around each character. For example, consider the cmdlet *Get-ChildItem*. Using this method, the cmdlet can be broken up into multiple segments, with single or double quotes around each character, as follows:

G'e't'-'C'h'i'l'd'i't'e'm

When interpreted by PS, the concatenated example is equivalent to the original cmdlet Get-ChildItem. Implementing this technique in the initial reverse shell code – Fig. 2, then the following cmdlets can be substituted with the values shown on Fig. 11. This makes the string harder to read, but again, it is still functional when interpreted by PS [25]. The AVs detection results of the discussed method against the PS code on Fig. 2 are shown on table 4.

```
# iex           # Out-String
i'e'x           O'u't-S't'ri'n'g
i''ex           Ou""t-S"tr"i"n"g
i"e"x           O'u't-St"r"i'n'g
i''e"x"         Ou""t'-'S""t'r'in''g
i""e''x""       Ou""t'-'S''t"r"i'n'g
```

Fig. 11. Cmdlet Obfuscation

TABLE 4 SINGLE TECHNIQUES RESULTS

№	AV	RC	CC	CO
Results collected November 2023		**Static Detection**		
1	Win Defender	Undetected	Detected	Undetected
2	Avast	Undetected	Undetected	Undetected
3	AVG	Undetected	Undetected	Undetected
4	Avira	Undetected	Undetected	Undetected
5	Bitdefender	Detected	Detected	Detected
6	NOD32	Detected	Detected	Detected
7	Fortinet	Undetected	Undetected	Undetected
8	Kaspersky	Detected	Detected	Detected
9	McAfee	Undetected	Undetected	Undetected
10	Sophos	Undetected	Detected	Detected
11	Malwarebytes	Undetected	Undetected	Undetected
12	Symantec	Detected	Detected	Detected
Results collected November 2023		**Runtime Detection**		
1	AMSI	Detected	Detected	Detected

G. Adding junk code (JC)

To evade detection, attackers may intentionally insert extraneous or irrelevant code into their PS payloads. This additional code serves no functional purpose and is designed to confuse or obfuscate the actual malicious commands. By adding junk code, the attackers can make their payloads more difficult for security systems to analyze and identify as malicious. Here's a brief example:

```
# These lines serve no purpose
# Some irrelevant code
# Actual malicious code
iex "malicious command"
# More unnecessary code
```

Fig. 12. (A) Add Commented-out code block

```
$randomvar1 = "Hello";
$randomvar2 = 123
Function unnecessaryfunc {
      # Irrelevant code
}
# Actual malicious code
iex "malicious command"
```

(B) Add unnecessary variables and/or functions

```
Sleep 0.1; sleep 0.2;
iex "malicious command";
sleep 0.3
```

(C) Add unnecessary sleep timers

```
Get-Process | Out-Null;
Get-Date | Out-Null;
iex "malicious command"
```

(D) Add unrelated function calls

These examples illustrate how junk code can be introduced to PS payloads, making it more challenging for security systems to identify and analyze the actual malicious commands. However, it's important to note that these evasion techniques can vary widely depending on the specific context and objectives of the attacker.

H. Summary – Putting all together

The following experiment integrates several techniques outlined in sections *A* to *G*, incorporating them directly into the initial PS script – Fig. 2. By utilizing different combinations, malware authors can tailor their obfuscation strategy based on their specific goals and the anticipated defense mechanisms they aim to bypass. The resulting outcomes are presented in table 5. The combined techniques aim to enhance malware obfuscation and evasion capabilities in various ways. The results show that script execution can successfully establish TCP session with an attacker and bypasses AMSI runtime detection as shown on Fig. 13. Table 5

provides a comprehensive overview of the effectiveness and impact of each technique when combined with another. Note that the purpose of the research was to evaluate the effectiveness of the discussed techniques in terms of detection evasion, code obfuscation, and overall impact on the detection rate of AV systems. By combining these techniques, the experiment sought to demonstrate the potential of employing multiple obfuscation strategies to increase the resilience of PS-based malware against detection and analysis.

TABLE 5 INTEGRATING ALL TECHNIQUES RESULTS

№	AV Software	OR+RC	OR+RC+CC	OR+RC+ CC+CS	OR+RC+ CC+CS+CO	OR+RC+CC+ CS+CO+JC	EN+OR+RC+ CC+CS+CO+JC
Results collected November 2023				**Static Detection**			
1	Microsoft Defender	Undetected	Undetected	Undetected	Undetected	Undetected	Undetected
2	Avast	Undetected	Undetected	Undetected	Undetected	Undetected	Undetected
3	AVG	Undetected	Undetected	Undetected	Undetected	Undetected	Undetected
4	Avira Antivirus	Undetected	Undetected	Undetected	Undetected	Undetected	Undetected
5	Bitdefender	Detected	Detected	Detected	Undetected	Undetected	Undetected
6	ESET NOD32	Detected	Detected	Detected	Detected	Detected	Undetected
7	Fortinet	Undetected	Undetected	Undetected	Undetected	Undetected	Undetected
8	Kaspersky Antivirus	Detected	Detected	Detected	Undetected	Undetected	Undetected
9	McAfee	Undetected	Undetected	Undetected	Undetected	Undetected	Undetected
10	Sophos	Undetected	Undetected	Undetected	Undetected	Undetected	Undetected
11	Malwarebytes	Undetected	Undetected	Undetected	Undetected	Undetected	Undetected
12	Symantec	Detected	Detected	Detected	Detected	Undetected	Undetected
Results collected November 2023				**Runtime Detection**			
1	AMSI	Detected	Detected	Undetected	Undetected	Undetected	Undetected

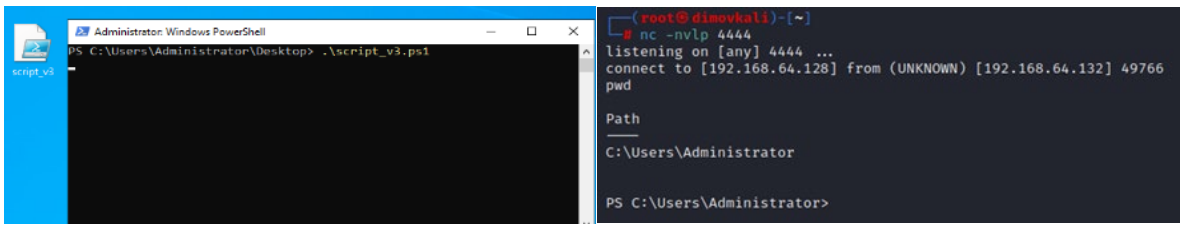

Fig. 13. (A) Script Execution (B) Session Establishment

IV. Conclusions

In conclusion, this research paper examined different obfuscation techniques employed in PS malware and their impact in AV detection. A virtual environment lab was prepared emulating an attacker box and malware distribution against 12 different AV sandboxes. At the time that these tests were performed, ESET NOD32 demonstrated the best performance among the discussed obfuscation techniques, followed by Symantec. However, as manual obfuscation does not rely on algorithms, it is important to note that the overall effectiveness of the AV systems strongly depends on the specific implementation of these techniques within the malicious script. Notwithstanding, the gathered results may still change over time as AVs signatures are frequently updated to detect new and changed payloads.

The findings showcased that utilizing a single obfuscating technique does not necessarily affect AV detection capabilities. On the other hand, the integration of multiple obfuscation techniques significantly enhance the malware's evasion capabilities resulting in a reduced detection rate and increased difficulty in analyzing the malicious code. These results highlight that the recommended approach in breaking both static-signature detection and runtime detection is by combining different obfuscation techniques, particularly in the context of red team activities.

Moving forward, future research could focus on exploring new obfuscation techniques and developing different detection methods to counter emerging threats. Additionally, continued collaboration between academia and industry will be crucial in advancing cybersecurity.

ACKNOWLEDGEMENTS

This publication was prepared in fulfillment of National Scientific Program – Security and Defense, financed by the Ministry of Education and Science of the Republic of Bulgaria.

REFERENCES

[1] P. Shijo and A. Salim, "Integrated Static and Dynamic Analysis for Malware Detection," in *International Conference on Information and Communication Technologies*, 2015.

[2] A. B. Ajmal, A. Anjum, A. Anjum and M. A. Khan, "Novel Approach for Concealing Penetration Testing Payloads Using Data Privacy Obfuscation Techniques," in *IEEE 18th International Conference on Smart Communities: Improving Quality of Life Using ICT, IoT and AI (HONET)*, Karachi, Pakistan, 2021.

[3] F. Pecorelli, F. Palomba, D. D. Nucci and A. D. Lucia, "Comparing Heuristic and Machine Learning Approaches for Metric-Based Code Smell Detection," in *IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, Montreal, QC, Canada, 2019.

[4] Ö. Aslan and R. Samet, "A Comprehensive Review on Malware Detection Approaches," *IEEE Access,* vol. vol. 8, pp. 6249-6271, 2020.

[5] M. J. e. a. Faruk, "Malware Detection and Prevention using Artificial Intelligence Techniques," in *IEEE International Conference on Big Data (Big Data)*, Orlando, FL, USA, 2021.

[6] A. Sharma, B. B. Gupta, A. K. Singh and V. Saraswat, "Orchestration of APT malware evasive manoeuvers employed for eluding anti-virus and sandbox defense," *Computers & Security,* vol. Volume 115, 2022.

[7] N. Miramirkhani, M. Appini, N. Nikiforakis and M. Polychronakis, "Spotless Sandboxes: Evading Malware Analysis Systems Using Wear-and-Tear Artifacts," in *IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, USA, 2017.

[8] D. Hendler, S. Kels and A. Rubin, "Detecting Malicious PowerShell Commands using Deep Neural Networks," in *ASIACCS '18: Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 2018.

[9] A. Al-Hakimi and A. Bakar Md Sultan, "Hybrid Obfuscation of Encryption," IntechOpen, 2023.

[10] K. Oosthoek and C. Doerr, "SoK: ATT&CK Techniques and Trends in," in *In: Chen, S., Choo, KK., Fu, X., Lou, W., Mohaisen, A. (eds) Security and Privacy in Communication Networks SecureComm 2019*, 2019.

[11] H. Xu, Y. Zhou and J. Ming, "Layered obfuscation: a taxonomy of software obfuscation techniques for layered security," *Cybersecurity 3, 9,* 2020.

[12] O. Or-Meir, N. Nissim, Y. Elovici and L. Rokach, "Dynamic Malware Analysis in the Modern Era—A State of the Art Survey," *ACM Computing Surveys,* vol. vol. 52, 2019.

[13] Sudhakar and S. Kumar, "An emerging threat Fileless malware: a survey and research challenges," *Cybersecurity 3, 1,* 2020.

[14] J. Singh and J. Singh, "Challenge of Malware Analysis: Malware obfuscation Techniques," *International Journal of Information Security Science,* vol. 7, no. 3, pp. 100-110, September 2018.

[15] C. Kalogranis, AntiVirus Software Evasion: An Evaluation of the AV Evasion Tools. Ph.D. Thesis, Piraeus, Greece,: University of Piraeus, Department of Digital Systems, 2018.

[16] S. Aminu, Z. Sufyanu, T. Sani and A. Idris, "Evaluating the effectiveness of antivirus evasion tools against windows platform," *FUDMA Journal of Sciences Vol. 4 No. 1,* pp. 89-92, 2020.

[17] D. Samociuk, "Antivirus Evasion Methods in Modern Operating Systems," *Applied Sciences,* vol. 13(8):5083, 2023.

[18] F. Garba, F. Yarima, K. Kunya, F. Abdullahi, A. Bello, A. Abba and A. Musa, "Evaluating Antivirus Evasion Tools AgainstBitdefender Antivirus," in *In Proceedings of the International Conference on FINTECH Opportunities and Challenges*, Karachi, Pakistan, 2021.

[19] G. D. C. D. &. B. G. Bernardinetti, "PEzoNG: Advanced Packer For Automated Evasion On Windows.," *Journal of Computer Virology and Hacking Techniques 18,* p. 315–331, 2022.

[20] N. S. Mittal, "week of powershell shells day 1," May 2015. [Online]. Available: http://www.labofapenetrationtester.com/2015/05/week-of-powershell-shells-day-1.html. [Accessed July 2023].

[21] D. Hendler, S. Kels and A. Rubin, "AMSI-Based Detection of Malicious PowerShell Code Using Contextual Embeddings," in *In Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIA CCS '20)*, New York, NY, 2020.

[22] M. Mimura and Y. Tajir, "Static detection of malicious PowerShell based on word embeddings," *Internet of Things,* vol. Volume 15, 2021.

[23] D. Ugarte, D. Maiorca, F. Cara and G. Giacinto, "PowerDrive: Accurate De-obfuscation and Analysis of PowerShell Malware.," in *In: Perdisci, R., Maurice, C., Giacinto, G., Almgren, M. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment DIMVA*, 2019.

[24] J. Klasmark, Detecting PowerShell Obfuscation Techniques using Natural Language Processing, Dissertation, KTH Royal Institute of Technology, 2022.

[25] A. Rousseau, "Hijacking .NET to Defend PowerShell," *Malware Research and Threat Intel,* 2017.