Challenges of Malware Analysis: Obfuscation Techniques

Jagsir Singh*, Jaswinder Singh*

*Department of Computer Science and Engineering, Punjabi University Patiala, India.

‡ Jagsir Singh, Punjabi University Patiala, India, Tel: +91-9592523807, email: erjagsirsingh18@gmail.com

ORCID ID: 0000-0003-0221-2691, 0000-0002-9201-4834

Research Paper Received: 19.08.2018 Revised: 14.09.2018 Accepted: 26.09.2018

Abstract - It is a big concern to provide the security to computer system against the malware. Every day a millions of new malware are developed and the worse thing is that new malware are highly sophisticated which are very difficult to detect. Because the malware developers use the various obfuscation techniques to hide the actual code or the behaviour of malware. Thereby, it becomes very hard to analyze the malware for getting the useful information in order to design the malware detection system because of anti-static and anti-dynamic analysis technique (obfuscation techniques). In this paper, various malware obfuscation techniques are discussed in detail.

Keywords - Dynamic Analysis; Malware; Obfuscation Techniques; Static Analysis.

1. Introduction

Despite the enhancement in computer security, still the malicious softwares are succeeding in their destructive objectives. Nowadays, it became a big challenge to keep the computer system secure from malware infection. Malware executes the malfunction in order to infect the computer system or computer resources. It can delete the data, slow down the system working or steal the important information. There are two research communities who are working parallel. One is developing malware detection and protection software and other is cracking the defensive system.

In the earlier, the concept of self-reproducing automation was given by John Von Neumann in 1949[[1]]. However, at that time no proper detail of implementation was feasible. The era of malware has been started around the 1980s when first actual computer "Brain" virus was created in 1986. It was created by the Pakistani brothers Basit Farooq Alvi and Amjad Farooq Alvin. But now the time has changed, millions of new malware are written in a day. According to the latest report of AV-test, the millions of new malware are produced every year. Figure 1 shows the statistics of new malware and total malware of last ten years from 2008 to 2017 (AV-TEST, 2017).

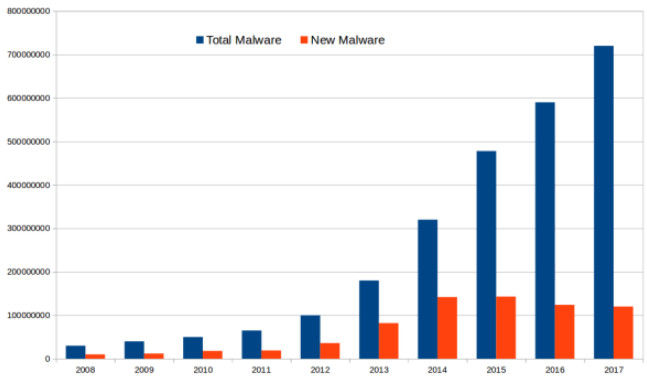

Figure 1. Bar Graph of New Malware and Total Malware of Last Ten Years.

Therefore, it very necessary to keep the system secure from these malware. Computer systems are

compromised by malware for many reasons such as:

- To harm the computer system.
- For financial gain.
- For stealing confidential or private data.
- For making the systems as bots.
- To make the services unavailable to the system.

If we compare the traditional malware with new the malware then we will get the idea how the new malware are so hard to detect. Traditional malware were broad, known, open and one time but now malware are very targeted, zero-day, stealthy and persistent as shown in Figure 2 [[6]]. Several types of new malware and their variants are being programmed by attacker to compromise the security of the computers systems.

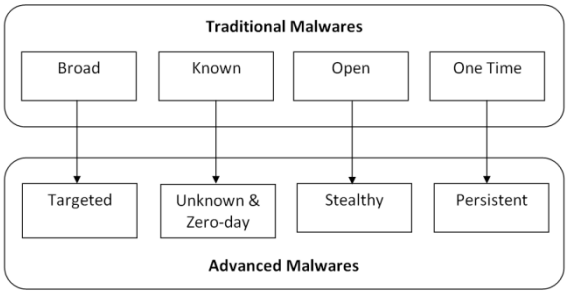

Figure 2 Comparison between Traditional (past) and Advanced Malware (present).

Today the malware are very specific for achieving the particular goal either to disrupt the working of system or any other like stealing important data. In order to avoid malware detector, new variants are created using various obfuscation techniques. In addition to encoding (encryption, base64) and packing techniques create the complex malicious software like polymorphic, metamorphic and packed malware [[7]] which can overrun the malware detection. Therefore, to crack or analyze such kind of malware is very time consuming and also very hard.

The output of malware analysis system must allow to the security organization for updating the malware defending software which can tackle the growth of malware and as a result to thwart the new malware.

The rest of paper is prepared as follows: Section 2 describes the malware analysis methods. Section 3 introduces the anti-static malware analysis techniques. Section 4 presents the dynamic malware analysis obfuscation techniques. Section 5 discusses the countermeasures to some anti-analysis techniques. Finally the paper is concluded.

2. Malware Analysis

Malware analysis is categorized into two main types of static and dynamic which are described as follow:

2.1 *Static Malware Analysis*

It is very basic and powerful phenomenon to analyze the malware without running the malware. In this analysis process code of malware are examined to find out the useful information. On the basis of that information, the malware detection software are designed (antivirus, IDSs etc). The extracted information can be the signature of malware file, program structure, executable format, instruction opcodes etc. For static analysis, code of the binary required. Therefore, reverse engineering is done to convert the executable malware file into the assembly code. Various disassemblers are used to transform the binary files into assembly code such as Ollydbg, IDA Pro [[4]], and Capstone. These disassemblers convert the binary files into the assembly language code, not in the same source code in which the malware file was actually contains. Then, the investigation is done on the assembly code to find the structure or pattern of malicious activity which can be used to detect the malware file or variants of that malware file as well. It is a tedious job to examine a thousand lines of assembly code. To solve this problem various alternatives are followed like the program is broken into parts or grouped on the basis of functioning. Additionally, code obfuscation techniques make the analyst's job harder. Malware writers use various obfuscation techniques such as code encryption, reordering the program

instructions and dead code insertion technique to evade the malware analysis [[7]].

2.2 *Dynamic Malware Analysis*

Dynamic analysis is also known as behavioural analysis. Dynamic analysis is based upon running the malware file then the interaction of malware with the computer system is monitored or observed. For analysis purpose, the malware are run in a controlled environment. In other terms malware files are executed in the virtual environment because if the malware file is run on host system then it will harm the host system. A virtual environment is created using virtualization tools like the Virtual box or VMware. Also, the dynamic analysis environment can be using emulators and hypervisor [[14]]. When a malware file is running in monitored environment various activities are observed such as the creation of new files, deletion of system or user files, new log entries, registry entries, URL accessed, data transmitted etc. Based on these activities, the file is considered as a benign file or malicious file. In the case of static analysis, the files which are not disassembled or not examined properly then those files can be analyzed in the virtual environment to know their behaviour. Various approaches are used in dynamic analyses which are explained as follows:

2.2.1 Tracking the flow of information

When the malware programs are investigated, it is necessary to know how information is being processed by the malware program. In the static analysis, the source code of malware is examined to interpret the flow of information from an instruction to another or from one block to another. However, it is a tedious job because a program file consists of thousands of lines of code. Also, this interpretation is totally based on the analyst capability to investigate the flow of information statically without running the malware. Therefore, running malware is analyzed in the virtual environment (VirtualBox) or in Sandbox (Cuckoo, Norman Sandbox, CWSandbox) in order to get an adequate flow of information. It is done in three basic ways such as following:

- Tainting the source and sinks
- Address Dependencies
- Control flow dependencies

In tainting approach, labels are assigned to the registers or identifiers [[20]]. The data elements which is assignment with the label is called tainted source. The variables also become the tainted if they are assigned from a tainted source. As shown in Figure 3 below the variable k is tainted because it may cause to call or trigger the suspicious activity. If any instruction processes the tainted register is detected as malicious action. On basis of tainted information malware file is detected.

```
                    //Tainted k
C=6                 mov ecx, 06
                    mov eax, ecx
C=C + K             add eax,  K
                    mov ecx, eax
```

Figure 3. Variable k is tainted because it may cause to call or trigger the suspicious activity.

While in address dependency, address tainting is used to observe sensitive information leakage [[21]]. Rather than tainting the data variable, address dependency also tracks the flow of information in an indirect way (using address by pointer). As shown in Figure 4 example pointer k is tainted. It is the base pointer to access array here. To assign a 5^{th} element to variable C using this tainted pointer. When a tainted pointer is assigned with an address of a register then de-referencing of the tainted pointer is detected as malevolent action as shown in figure 4.

```
                    // Tainted pointer k

C = K[5]            mov ecx, [K+5]
```

Figure 4. Pointer k is tainted.

Moreover, control flow dependency is also used to track the flow of information. In the program instructions depend on others instruction and also other instruction depends on that instruction. On the basis of execution of instruction, it is evaluated

the flow of data in order to get know about any suspicious event.

2.2.2 Monitoring the function calls

Function call monitoring is second most used dynamic analysis approach in which malware programs are monitored to know what functions are called [21]. A malware program can call various types of functions related to API (Application Programming Interface), systems calls, window native calls [[21]]. For example, malware calls the function such as CreateFile, DeleteFile, GetProcAddress. It helps to identify the malware files. On the basis of order of the functions calls, malware detection systems are designed to detect the malware and classify them into proper categories. A process is used to intercept the function calls are known as hooking.

3. Anti-Static Analysis Methods

Obfuscation means unclear or obscure which is not understandable. Therefore, the malware writer uses several obfuscation techniques to evade the analysis. From the ancient time; various camouflages have been used to hide the actual information. For example when a king had to send information to another king then they used to use secret and hidden methods to keep the data confidential. The purpose of these approaches was to keep important information secret. In modern computer era, various algorithms are used for confidentially, integrity and authentication of data. Similarly, the malware developers use obfuscation techniques to conceal the malicious code to bypass the malware detection system (Antivirus). Obfuscation techniques can be divided into two categories anti-static and anti-dynamic analysis techniques. In this section mostly used anti-static obfuscation methods are explained as follows.

3.1 Change the order of the code

It is a simple obfuscation approach to change the order of execution of program instructions [[8]]. Unconditional jump statements are inserted into the program code for changing the order of code execution without affecting the actual behaviour of malware program as shown in Figure 5. It seems very simple for this example to find out the original order but for hundred lines of code, it becomes cumbersome for the analyst to find out the actual order.

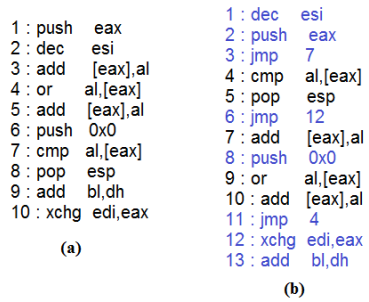

Figure 5 (a) Original x86 assembly code, (b) It shows reordered code using unconditional jump instruction.

3.2 Redundant Data Insertion

Malware writers insert the dead code into the program for creating the new version of same malware just for increasing the overhead of the analyst [[19]]. This approach can evade the signature-based detection systems. When the redundant code is inserted then the different signature is generated. This approach of obfuscation affects the static analysis only because in static analyzes it becomes difficult to distinguish the dead code which has no contribution in the working of malicious software. Thus investigating the dead code is extra overhead for the malware analyst. The redundant code or dead code doesn't affect the original purpose of the malicious software. Unconditional jump statements are used to bypass the redundant code block which retains the original executing order of malware. Moreover, a serious of NOP (Not operation like instruction in x86) statements are inserted in the malware to create new variant as shown in Figure 6.

```
1 : push eax
2 : nop          // Not Operation
3 dec esi
4 : add [eax],al
5 : nop
6 : or al,[eax]
7 : jmp 12
8 : mov [ebp-0x18],esp
9 : mov [ebp-0x14],0x4010e0   // Redundant Code
10 : and eax,0x1
11 : mov [ebp-0x10],eax
12 : add [eax],al
13 : push 0x0
14 : cmp al,[eax]
15 : nop
16 : pop esp
```

Figure 6 Insertion of Redundant code (garbage code) in figure 4(a).

3.3 Equivalent Code Replacement

This obfuscation technique substitutes the originals instructions of malware with other instruction while retaining the semantic of malware [[8], [19]]. Thereby, numbers of variants of same malware files can be created. To handle this problem for every possible variant of same malware the unique signatures is required to detect these variants as well. It is not an impossible task but with the face of increasing new variants of same malware is not an easy task. In every programming language, the same function can be performed in a number of ways. Therefore, the malware writers exactly do the same things. They transform the actual instruction into equivalent instructions. For example, multiplication can be performed using either a series of ADD instructions or a single multiplication instruction (MUL). Figure 7 shows the equivalent code replacement technique.

```
1 : push   eax          1 : push   eax
2 : dec    esi          2 : sub    esi, 1
3 : add    [eax],al     3 : add    [eax], al
4 : or     al,[eax]     4 : or     al, [eax]
5 : add    [eax],al     5 : add    [eax], al
6 : push   0x0          6 : xor    ebx, ebx
7 : cmp    al,[eax]     7 : cmp    al, [eax]
8 : pop    esp          8 : mov    esp, ebx
9 : add    bl,dh        9 : add    bl, dh
10 : xchg  edi,eax      10 : xchg  edi, eax
```

(a) (b)

Figure 7(a) Original assembly code, (b) Transformed code into equivalent form.

3.4 Rename the identifiers

In this obfuscation technique, the identifiers of constants, variables, and registers are changed with other names without altering the semantic [[11]] as shown in figure 8. However, it is expensive obfuscation approach because it requires manual transformation of identifiers of constants, registers, and variables.

```
1 : push   ebx
2 : dec    esi
3 : add    [ebx], al
4 : or     al, [ebx]
5 : add    [ebx], al
6 : push   0x0
7 : cmp    al, [ebx]
8 : pop    esp
9 : add    bl, dh
10 : xchg  edi, ebx
```

Figure 8. Renaming the registers.

3.5 Packing the code

It is advanced obfuscation technique to create more complex and sophisticated variants of malware which makes the static analysis more difficult. In this technique, actual malware code is compressed or encrypted into different form but semantically same [[18], [19]]. Figure 9 shows the packed malware. As a result, new executable file consists of packed or wrapped malware binary code (compressed or encrypted) and an unpacking code. This unpacking code defines the entry point of new packed malware file which is invoked by the operating system. Then, the unpacking code is executed; it unpacks the original malware code into the memory at the runtime. In other words, the unpacking routine represents the original entry point (OEP). Moreover, the unpacking routine handles the imports for actual executable malware file. At last, it returns the control to the original OEP then malware starts performing its actual functioning.

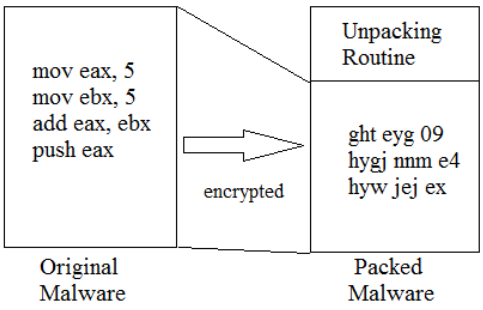

Figure 9. Packed Malware.

To create the new malware variants, the various packers such as UPX, NSPack, UPACK, and FSG are used to compress actual code of malware. Thereby, it hides the malicious code which subverts malware detection. In section 3.3.3, countermeasures for packing malware are discussed.Various variants of a malware are discussed as follow.

a. Encryption

Original malware code is encrypted using an encryption key to generate the encrypted payload. Every encrypted malware consists of an encrypted payload, an encryption key, and decryptor[6,10, 15,19, 23]. In addition, a different encrypted variant of same malware can be produced using a different encryption key. Thereby, it could evade the malware detections. Win95/Mad and Win95/Zombie were examples of the encrypted 32-bit malware in which cascaded encryption was applied for making encrypted virus more complex. The weak point of the encrypted malware is that the same decryptor is used to decrypt the encrypted payload every time. For this reason, malware detection system can be trained can be done on the signature of malware decryptor.

b. Oligomorphic

Oligomorphism implies few structures. It is Greek term combination of two words: oligo (i.e. a small number of) and morphe means form. Oligomorphic malware overcomes the limitation of simple encrypted malware in which the same decryptor is used to create the copies of malware file [19]. In Oligomorphic malware, the decrypt or imitates into different form every time to decrypt the malware file into equivalent form while retaining the same semantic. Win95/Memorial was the Oligomorphic malware which had the capability to create the 96 variants of original file. The problem with Oligomorphic malware is that only a limited number of decryptor can be made. Consequently, a malware detector can use this weakness for detection of every possible variant of malware files.

c. Polymorphic

Polymorphism implies many structures or forms and is gotten from the Greek terms poly (i.e. numerous) and morphe means form. Moreover, it is not just in view of encryption techniques like its forerunner, yet uses the blend of various obscurity procedures, for example, dead code addition. Basically, the polymorphic malware are an advanced version of the Oligomorphic malware. Unlike Oligomorphic, unlimited decryptors can be generated in order to produce the unlimited malware variants [15, 19]. Hence, the polymorphic malware can imitate itself into unlimited numbers of semantically equivalent variants which evade the malware detection. Win95/HPS and Win95/Marburg were the first 32-bit polymorphic malware. Polymorphic malware can use the multiple layers of encryption as well for making the detection much more difficult. For example the win32/Coke and Win32/Crypto were the multi layer polymorphic malware. Despite the fact that the polymorphic malicious software can viably avoid the signature-based detection. But, the static body of this can be used to detect its presence. Even if the code is changed into other form but the semantic of the equivalent code remains same. So, it is feasible to apply the signature matching techniques during the runtime.

d. Metamorphic

Igor Muttik defined metamorphic malware as: “Metamorphics are the bodypolymorphics”. Metamorphic malware doesn’t have a constant body and a decryptor; because metamorphic malware do not use any encryption and packing technique to thwart the analysis [14, 23].These malware transform its binary code dynamically to evade detection. Unlike Oligomorphic and polymorphic, it does not reveal the constant body in the memory. Metamorphic malware imitates another form during runtime in memory. That is why it is known as dynamic code obfuscated malware.

Very early in 1998, Vacna, a malware writer, implemented a metamorphic malware Win95/Regswap by exchanging the used registers in the code as shown in figure 10.

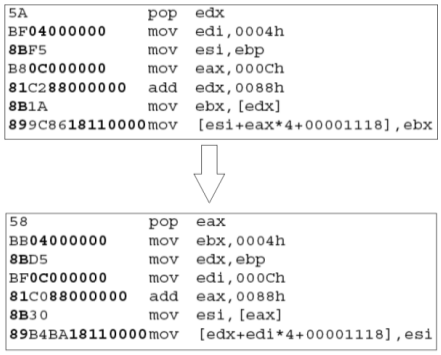

Figure. 10 Win95/Regswap Metamorphic Malware with different registers

For this reason, the metamorphic malware are very hard to detect as compared to other malware.

4. Dynamic Analysis Evasive Techniques

Dynamic analysis is performed by designing a virtual or emulator environment. Also some kind of debuggers' tools can be used to monitor the behavioural artifacts of executing malware. The main requirement while designing the monitoring environment is transparency. The analysis and non-analysis systems must be indistinguishable to each other [[25]]. In [[26]] five main transparency conditions are explained. In Table 1 the transparencies of four dynamic analysis environments are listed. Dynamic malware analysis is applied in many ways like using virtual environment, emulator, hypervisors and bare metal. But each of them has their pros and cons. If the transparency is considered an important point, then bare metal is much more effective because it is immune to timing attacks.

	VM	**Emulated**	**Hypervisor**	**Bare Metal**
Transparency	Low	Low	Medium	High

Table 1. Transparency Level of four Dynamic Analysis Environments.

4.1 Detection of Virtual Environment

It is not likely to execute the malware files onto the host computer as such because the malware files can harm the host computer. Normally for analysis we setup the virtual environment.

The malware can detect the monitored environment in which it is being monitored and hide the actual behaviour [[5], [9],[10],[27],[28]]. Thereby, it fails the malware analysis. Malware writers use the two main features to know the presence of virtual platform such as:**signature of virtual tools** and **fingerprint of the operating system**. Signature of virtual tools means the presence of Virtual Box or VMware over the virtualization has been done. For example, in the case of Microsoft Windows VMware leaves the hint in the registry and creates the many processes such as **VMwareService.exe, VMwareTray.exe** etc. Moreover, in the case of MAC operating system there is specific hardware address (00:0c:29 first three bytes) which exposes the presence of VMware [[15]].

It can also detect the guest operating system over which it is running. In other terms, malware can detect that the guest operating which is installed on VirtualBox or VMware for malware analysis. The guest operating has different kernel data structure than real one when it is installed in the virtual machine. Thereby, the malware writer can exploit these weaknesses or vulnerabilities to known the virtualization and hide the actual behaviour.

4.2 *Network artifacts detection*

Different network behaviour and qualities can be used by malware to detect the analysis. For examples network simulation and isolation [[29]], fast internet service [[33]] and fixed IP addressing. Miramirkhani et al. [[30]] discussed about network behaviour in the virtual environment which is itself an indicator of detection. Some emulator does not perform well like Android SDK which is unable to forward the ICMP packets [[31], [32]].

4.3 Recognition of debuggers

Furthermore, the malware not only detect the virtual environment but also can detect the debugging tools. This is anti-analysis mechanism

when the analyst is using actual host machine to known the actual behaviour of malware. Detection of a debugger can be done in following ways as:

4.3.1 API detection

In MS Windows operating system, the API calls can be used to detect the debugging tools. A malware writer can write the small code to check the **BeingDebugged** flag in Windows OS in order to detect the debugger as shown in Figure 11.

```
Boolean IsDebugPresent ()
{
Return (NtCurrntPeb () - >beingDebbuged);
}
```

Figure 11 Example of detection code used by malware.

There are many API functions which are commonly used by malware to known whether the malware is being analyzed as CheckRemoteDebuggeRpresent(), OutputDebugString() [[8], [4], [19], [23]].

4.3.2. Services and handles

Services and handles are used by various malware. The various fine debuggers have main services which may be used by malicious software to identity their existence.. SoftICE is well-known kernel level debugger tools. Its service NTICE can be used by malware to detect its presence.

4.3.3 Signature of Debuggers

This is very simple and effective anti-analysis approach to detect the present of the debugger by using their signature and address. Like **83 3D 1B 01** was the signature of old version Ollydbg.

4.4 Browser Based Fingerprints

In analysis environment, the browser is also vulnerable which can be exploited by malware in order to confirm the detection environment [[33], [34]]. There are certain discrepancies in features of JavaScript language such as exception handling or parsing which can be a reason of analysis environment detection. Because browser behaves differently in analysis environment compared to host operating system. Also, ActiveX behaves differently in browser in virtual and emulated environment can be a fingerprint of detection. In [[33]], two other feature of browser HTML parsing and Document Object Model can be detected in the emulated environment.

5. Countermeasures to Some Anti-Analysis Techniques

In this section, countermeasures to the anti-analysis techniques are discussed. We have discussed countermeasures for redundant code insertion, reordering of actual malware code and packing malware.

5.1 Countermeasure for Redundant Code

Practically speaking, ClamAV anti-virus programs provided the solutions to NOP instructions [[17]]. This technique just only concentrates on viral byte arrangements and semantic NOP byte instructions are overlooked. It is highly dependent on used regular expression and wildcards. A poor decision can bring about a high false positive rate. Christodorescu et al. proposed a standardization approach where NOP and semantic NOP instructions are distinguished and evacuated by watching the content. However, this technique can't be effective if further obscurity techniques are used in the malware file. If malware writer has used additional obfuscation technique along with NOP instruction then this technique fails to handle NOP redundancy. Thereby, it is not possible to disassemble the malware code accurately. Besides, checking whether a code is a semantic NOP is undecidable [[4]].

5.2 Countermeasure for Reordering of Code

Christodorescu et al. (2007) proposed an approach which uses a CFG(control flow graph) invariant to determine and remove the reordering of malware program. Using invariant CFG, we again reorder the code into the actual order which was before

first reordering. But the requirement of this technique is that malware code must be disassembled properly thereby the CFG can be made appropriately.

5.3 Countermeasure for Packer

Revealing malware secured by archivers is not as tough as the reverse techniques are available and effortlessly reachable. Conversely, beating packers is substantially more troublesome. First of all, one needs to recognize them effectively. This can be either accomplished by searching for section names inside the packed malware program, which can uncover the packer (e.g. UPX0, UPX1 if UPX packer is used or look for different markers for example, few library imports, unusual segment sizes (e.g. size of crude information is 0 while the virtual size is never zero). The other way is to unpack the packed malware program in order to access the code which represents the actual behaviour of malware. In this manner, there are three basic options for unpacking [[15]] such as follows:

a. Static: Automated Unpacking

This approach deals with packed malware without running them and uses some automated tool for unpacking. Most commonly used packing tools are UPack, UPX, NSPack, FSG, ASpack etc. There are various tools such as PEid, PE Explorer and PE view which are capable for unpacking the packed malware files which are packed using these tools. These tools restate the malware executable into original form (unpack) without running the malware file. But, the malware writers can use several anti-packing mechanisms to evade the unpacking such as data encoding (e.g. base64 coding), encryption and, anti-disassembly techniques (multilevel instruction, abuse of pointers and exception handlers) [[23], [24]]. Consequently, to unpack the packed malware is a big challenge for the analyst [[15], [23]].

b. Dynamic: Automated Unpacking

In dynamic unpacking, the malware file is executed. When, the unpacking routine unpacks the malware file then original import table is constructed. The big hurdle of this approach is to find out the beginning of original code (original entry point) and ending of unpacking routine. It requires hard work and expertise. Undesirably, this is a difficult issue to handle automatically. Consequently, manual negotiation is done to determine the starting of original malicious code.

c. Manual Unpacking

It is not an easy task to find out the Original Entry Point(OEP) of malware programs. It requires a lot of hard work and the great understanding about the packing tools in order to get insight about the packed malware file. Unlikely, no such method is there which can determine the entry point of packed file.

6. Conclusion

Analysis of malware is very tedious task. Obfuscation is one of the major factors which affects the analysis of malware. There are two basic ways to analyze the malware signature based (without executing the file) and behaviour-based (running the file mostly in controlled environment). After studying various research papers and whitepapers of security experts it has been shown that the signature-based detection techniques have become obsolete. Also the signature-based detection techniques can't detect the new malware. Now the second alternative is behaviour-based analysis in which malware files are executed for capturing the behavioural artifacts. There is also a possibility that complex obfuscated malware can cheat the execution environments like sandboxes, debuggers due to not executing actual behaviour. Even though behaviour-based system detection systems are far better than signature-based malware detection systems, behaviour-based systems are slow compared to signature-based system. Therefore, the time consuming is also a big concern in order to scan the system and give the decision within instant of time. By considering the pros of both analysis techniques integrated malware detection systems can provide solution to both problems

time efficiency and detecting the unknown malware (new malware).

References

[1]. Malware Statistics & Trends Report | AV-TEST. https://www.av-test.org/en/statistics/malware/ , 2017.

[2]. V. Neumann, "Theory of self-reproducing automata", *Urbana, University of Illinois Press*, 1966.

[3]. J. Blackthorne, A. Bulazel, A. Fasano, P. Biernat, and B. Yener. "AVLeak: Fingerprinting Antivirus Emulators Through Black-Box Testing", *In WOOT'16 USENIX Workshop on Offensive Technologies*, USENIX, 2016.

[4]. B. Beaucamps, and J.Y. Marion, "On behavioral detection", *European Institute for Computer Antivirus Research Annual Conference*, EICAR'09, 2009.

[5]. M. Christodorescu, S. Jha, J. Kinder, S. Katzenbeisser, H. Veith, "Software transformations to improve malware detection", *Journal Computer Virology*, Vol:3(4), pp no:253-265, 2007.

[6]. Xu Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario, "Towards an Understanding of Anti-virtualization and Anti-debugging Behavior in Modern Malware". *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, 2008.

[7]. E. Gandotra, D. Bansal, S. Sofat "Malware Analysis and Classification: A Survey". Journal of Information Security, vol:5(2), pp no: 56-64, 2014.

[8]. I. You, K. Yim, "Malware Obfuscation Techniques: A Brief Survey", *International Conference on Broadband, Wireless Computing, Communication and Applications*, pp no: 297-300, 2010.

[9]. Y. Gao, Z. Lu, Y. Luo, "Survey on malware anti-analysis*", IEEE ,international conference on Intelligent Control and information processing*, 2014.

[10]. T. Vidas, N. Christin, "Evading Android Runtime Analysis via Sandbox Detection", *ACM Symposium on Information, Computer and Communications Security*. ACM, 2014.

[11]. C. Thompson, M. Huntley, C. Link. Virtualization Detection: New Strategies and Their Effectiveness. Technical Report. University of Minnesota, 2010.

[12]. A. Karnik, S. Goswami, R. Guha, "Detecting obfuscated viruses using cosine similarity analysis", *International Conference on Modeling and Simulation*, pp no:165-170, 2007.

[13]. B. Zhang, J. Yin, J. Hao, D. Zhang, S. Wang, "Malicious codes detection based on ensemble learning", *Springer In Autonomic and Trusted Computing*, vol:46(10), pp no: 468-477, 2007.

[14]. M. Cova, C. Kruegel, and G. Vigna, "Detection and Analysisof Drive-by-Download Attacks and Malicious JavaScript Code", *ACM International Conference on World Wide Web*, 2010.

[15]. M. Sikorski, A. Honig, "Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software" *Press, San Francisco,CA, USA*, 2012.

[16]. K. A. Roundy, B. P. Miller, "Binary-code obfuscations in prevalent packer tools", *ACM Computing Survey, Vol:46(1)* pp no:1-4, ClamAV, 2013.

[17]. Clamav, Available at http://www.clamav.net/index.html, 2017.

[18]. S. Alam, R. N. Horspool, I. Traore, I. Sogukpinar, "A framework for metamorphic malware analysis and real-time detection", *Computers & Security* ,vol:48, pp no: 212-233, 2015.

[19]. R. Hedayat, The devil's right hand: An investigation on malware-oriented obfuscation technique. Report, pp no:-31-67, 2016.

[20]. F. Zhang , M. Yang, M. Xu "A malware analysis platform based on taint analysis", *IEEE International Conference on Computer Sciences and Applications, 2013*.

[21]. M. Egele, T. Scholte, E., C. Kruegel, "A survey on automated dynamic malware analysis techniques abd Tools", *ACM computing Surveys*, 2012.

[22]. W. Aman, "A Framework For Analysis And Comparison Of Dynamic Malware Analysis Tools", *International Journal of Network Security & Its Applications (IJNSA)*, Vol.6(5), 2014.

[23]. K. Coogan, G. Lu, S. Debray, "Deobfuscation of virtualization-obfuscated software" *Conference on Computer and Communications Security - CCS '11*, 2011.

[24]. P. Burnap, R. French, F. Turner, K. Jones, "Malware classification using self organising feature maps and machine activity data". *Computers & Security*, vol:73, pp no:399-410, 2017.

[25]. T. Garfinkel, K. Adams, A. Warfield, J. Franklin, "Compatibility is Not Transparency: VMM Detection Myths and Realities", *USENIX Workshop on Hot Topics in Operating Systems*, 2007.

[26]. A. Dinaburg, P. Royal, M. Sharif, W. Lee, "Ether: Malware Analysis via Hardware Virtualization Extensions", *ACM Conference on Computer and Communications Security*, 2008.

[27]. J. Boomgaarden, J. Corney, H. Whittaker, G. Dinolt, J. McEachen, "Challenges in Emulating Sensor and resource-Based State Changes for Android Malware Detection", *IEEE International Conference on Signal Processing and Communication Systems (ICSPCS),* 2015.

[28]. G. Pek, B. Bencsath, L. Buttyan. "Ether: In-guest Detection of Out-of-the-guest Malware Analyzers", *Fourth European Workshop on System Security*, 2011.

[29]. N. Miramirkhani, M.P. Appini, N. Nikiforakis, and M. Polychronakis, "Spotless Sandboxes: Evading Malware Analysis Systems using Wear-and-Tear Artifacts", *IEEE Symposium on Security and Privacy*, 2017.

[30]. D. Maier, M. Protsenko, T. Muller "A Game of Droid and Mouse: The Threat of Split-Personality Malware on Android", *Computers ad Security*, vol:54, 2015.

[31]. D. Maier, T. Muller, M. Protsenko, "Divide-and-Conquer: Why Android Malware Cannot Be Stopped", *IEEE Ninth International Conference on Availability, Reliability and Security (ARES'14),* 2014.

[32]. B. D. Gavitt, Y. Nadji, "See No Evil: Evasions in Honeymonkey Systems", Technical Report 2010.

[33]. M. A. Rajab, L. Ballard, N. Jagpal, P. Mavrommatis, D. Nojiri, N. Provos, and L. Schmidt. Trends in

Circumventing Web-Malware Detection. Technical Report. Google, 2016.

[34]. A. Kapravelos, M. Cova, C. Kruegel, and G. Vigna, "Escape from Monkey Island: Evading High-Interaction Honeyclients", *Springer International Conference on Detection of Intrusions and Malware,and Vulnerability Assessment (DIMVA)*, 2014.