Camouflage In Malware: From Encryption To Metamorphism

Article · January 2012

CITATIONS
139

READS
18,194

3 authors:

Babak Bashari Rad
Asia Pacific University of Technology and Innovation
49 PUBLICATIONS **815** CITATIONS
SEE PROFILE

Maslin Masrom
Universiti Teknologi Malaysia (Kuala Lumpur)
148 PUBLICATIONS **1,825** CITATIONS
SEE PROFILE

Suhaimi Ibrahim
Universiti Teknologi Malaysia
140 PUBLICATIONS **1,404** CITATIONS
SEE PROFILE

Camouflage in Malware: from Encryption to Metamorphism

Babak Bashari Rad[†], Maslin Masrom [††], Suhaimi Ibrahim[†††]

[†]Faculty of Computer Science and Information System, University Technology Malaysia
[††]Razak School of Engineering and Advanced Technology, University Technology Malaysia
[†††]Advanced Informatics School, University Technology Malaysia

Summary
Camouflage of malware is a serious challenge for antivirus experts and code analysts. Malware use various techniques to camouflage them to not be easily visible and make their lifetime as longer as possible. Although, camouflage approaches cannot fully stop the analyzing and fighting against the malware, but it make the process of analyzing and detection prolonged, so the malware can get more time to widely spread. It is very important for antivirus technologies to improve their products by shortening the detection procedure, not only at the first time facing with a new threat, but also in the future detections. In this paper, we intend to review the concept of camouflage in malware and its evolution from non-stealth days to modern metamorphism. Moreover, we explore obfuscation techniques exploited by metamorphism, the most recent method in malware camouflage.

Keywords:
Camouflage in Malware, Malware Evolution, Malware Encryption, Malware Oligomorphism, Malware Polymorphism, Malware Metamorphism, Obfuscation Techniques

1. Introduction

From the early days of computer malware creation, there is a serious challenge between the malicious code programmers and antivirus specialists. Each opponent endeavors to growth its capabilities to defeat the adversary [1]. In the side of malware producers, one of the most important issues is to prolong the lifetime of the malware in the wild, as much as possible. It is achievable if the malware is able to abscond from the antivirus scanner engines well. Consequently, the camouflage of the malware code is significant factor to make it successful in the wild. In this article, we tend to survey the malware camouflage tactics from the earliest simple viral codes until more advanced introduced techniques, nowadays.
There are four main generations in gradual development of the stealth methodologies [2-3]: Encryption, Oligomorphism, Polymorphism, and Metamorphism. Figure 1 displays the evolution timeline of camouflage techniques in malware.

Manuscript received August 5, 2012
Manuscript revised August 20, 2012

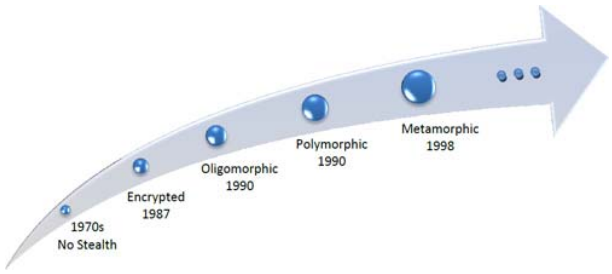

Figure 1: Evolution timeline of Camouflage techniques appearance in malware

2. Primitive Malware

When the story began, virus writing was a kind of programming fun for computer specialists to show off their technical skills, but it gradually became as a tool for other purposes, such as swiping the people's information, like credit card numbers, passwords, or bank account numbers, or for avenges purposes, and so on. In the beginning, there were no techniques invented to escape from the code analyzers or experts who were trying to find malicious code and trap them. Computer programmers liked to satisfy the tempting to make the virus. They were enjoying finding new ways of these amazing programming strengths.

3. Stealth Malware

Malware creators' first attempts in order to escape from trapping redounded to appear of stealth techniques. Stealth virus is able to conceal its signs and traces. Virus normally changes and modifies data resources on the system. For example, a file-hosted virus may append its own code to the end of an executable file. If an application examines the infected file, it can detect the viral code in the file and catch the virus. Stealth virus can hide the changes that it previously applied. When other applications request the parts of resources modified by the virus, stealth code of the malware that dominates the system, delivers the unchanged data instead of the viral code.
The term of "stealth" can be used as a general term for all kind of malicious codes, which are capable to hide

themselves from being visible. They employ many techniques to conceal any existent sign of themselves in the system resources. In general, camouflage actions of the stealth malware may be categorized in two aspects: hiding the trails of the malware or hiding its own code from the human or other programs. The first aspect depends on what the virus is going to perform on the host system. It may change the file system or its components, the memory management system, and so on.

The most considerable reasons to use stealth techniques in malware and hide the viral code and signs of the virus are:

1. To make it invisible from non-expert persons
2. To prevent the static analysis and reverse engineering of the virus code
3. To prolong the lifetime of the virus
4. To prevent modifying the code by other persons

4. Camouflage Evolution

4.1 Encryption

Malware authors always try to improve their program to escape from code analyzer technicians. Accordingly, they could get more time for their produced malware to live in the wild and show off more. The earliest and simplest method employed by the malware programmers to achieve this goal was encryption. The first known encrypted virus, Cascade, was appeared in 1987 [4].

Encrypted virus is composed of two basic sections: a decryption loop and main body. Decryptor, or decryption loop, is a short piece of code, which is responsible to encrypt and decrypt the code of main body. The main body is the actual code of the malware, encrypted, and is not meaningful before it is being decrypted by the decryption loop. When the virus starts to run on the host computer, first the decryptor loop must decode the main body into machine executable code and meaningful data.

Encryption of the code can be carried out via various approaches. For example, a simple encryption may use a 1 to 1 mapping to transform the code byte by byte. In another simple form of encryption, a zero-operand instruction such as INC or NEG can be used. More sophisticated encryption techniques may be utilized, as well, which use reversible instruction, e.g. ADD or XOR with random keys. In addition, the encryption key may be constant value or be a sliding variable value generated by a special algorithm.

Figure 2 depicts the general structure of an encrypted virus.

However, a virus scanner cannot immediately detect the virus using signatures and it first needs to decrypt the virus body to access the whole code. However, it can find the decrypting part, so if this part includes of enough bytes as string signature, it still causes that indirect detection of the virus through string signature be achievable.

However, a virus scanner cannot immediately detect the virus using signatures and it first needs to decrypt the virus body to access the whole code. However, it can find the decrypting part, so if this part includes of enough bytes as string signature, it still causes that indirect detection of the virus through string signature be achievable.

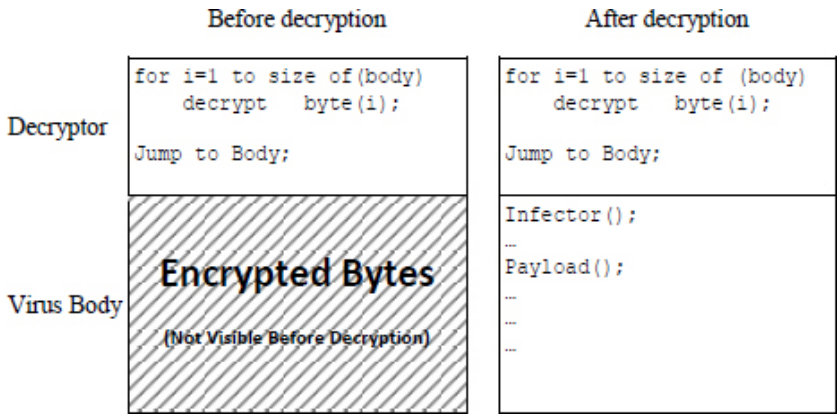

Figure 2: Structure of encrypted virus

4.2 Oligomorphism

The next efforts in advancement of the malware concealment bring about the appearance of oligomorphic viruses. Oligomorphic virus is also called as semi-polymorphic [5]. It was an attempt to make the decryptor loop of encrypted virus different appearance in each new infection. Oligomorphism is an advanced form of the encryption. It contains a collection of different decryptors, which are randomly chosen for a new victim. In such a way, the decryptor code is not identical in various instances. The first known oligomorphic virus was the

Whale, a DOS virus that was appeared in 1990 [2]. Figure 3 displays the structure and mechanism of an oligomorphic virus, schematically.

Oligomorphism is not a major problem for the antivirus software because it only makes a malware slightly more difficult to observe. Unlike encrypted virus, antivirus engine has to check all possible decryptor instances instead of looking for only one decryptor, and it needs a longer time.

4.3 Polymorphism

Polymorphism is actually the most complicated type of oligomorphism and encryption [6]. Polymorphic viruses are similar to encrypted and oligomorphic viruses in usage of code encryption, but the difference is that polymorphics are able to create an unlimited number of new different decryptors [2]. The first polymorphic virus, 1260, a virus of chameleon family appeared by 1990, was developed by Mark Washburn [2].

Polymorphic techniques try to make analysis of virus harder by changing its appearance. The principal rule is to modify the appearance of the code constantly, from a copy to another [7]. It must be carried out in such a way so no permanent common string remain among variants of a virus to be exploited by the antivirus scanner engine for detection purpose. Polymorphic techniques are rather difficult to implement and manage [8].

Polymorphic virus utilizes code obfuscation approaches such as insertion of junk codes or substitution of instructions to mutate its decryptor and build a new one for new infected victim.[9] The section responsible for this process is called mutation engine or obfuscation engine [5].

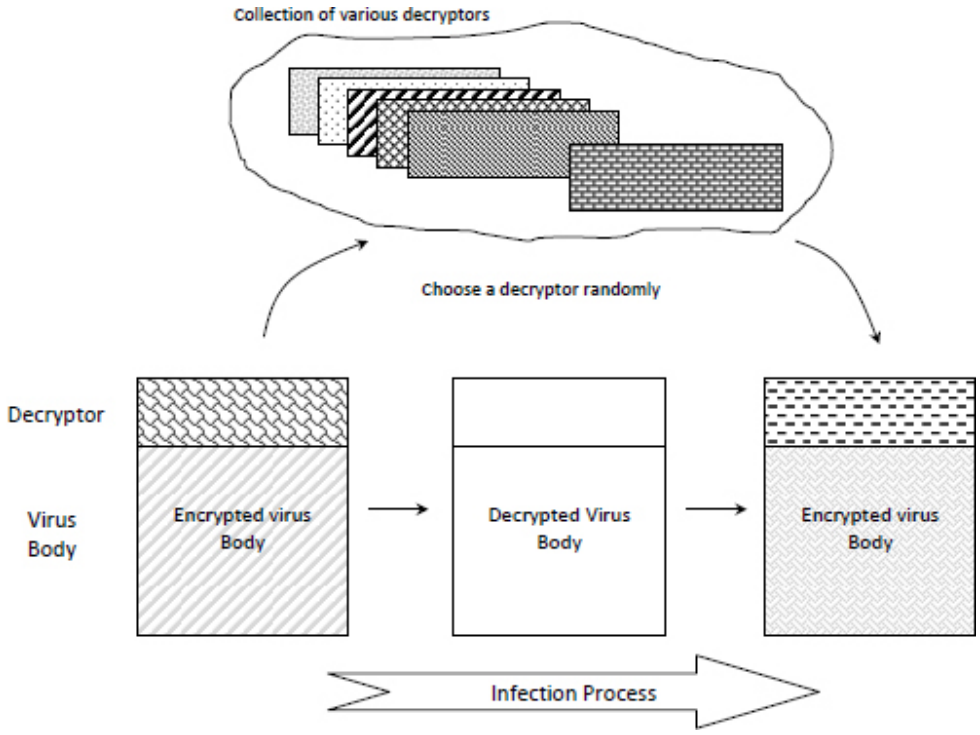

Figure 3: Structure and mechanism of oligomorphic virus

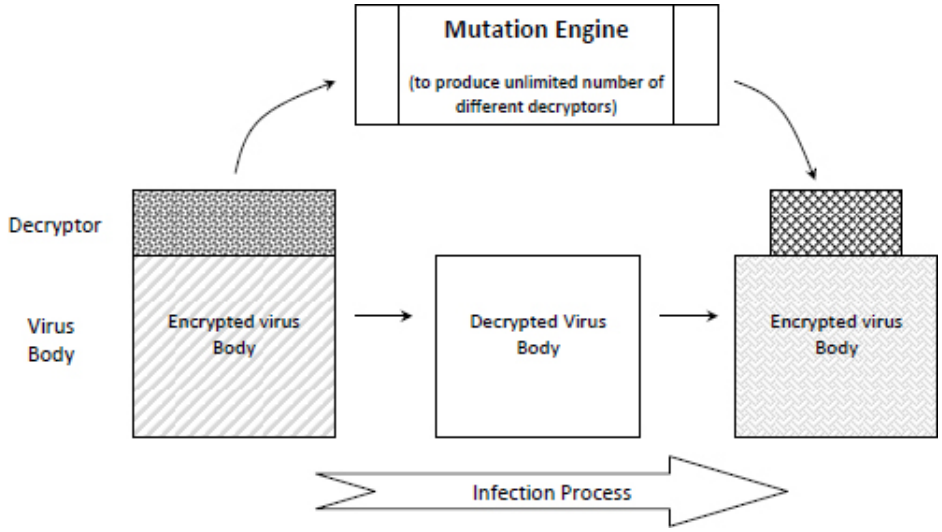

Figure 4: Polymorphic virus structure

Some of most common obfuscation techniques, which are exploited by polymorphic viruses to mutate their code, are [3, 9-11]:

1. Instruction replacement
2. Instruction permutation
3. Variable/Register substitution
4. Junk /Dead code insertion
5. Code transposition

Figure 4 illustrates the structure of a polymorphic virus and its infection process, briefly.

It does not matter how much a polymorphic virus is designed well, after a sufficient emulation of the code, the underlying encrypted code will be revealed and can be detected by a simple string matching [12]. Malicious code writers felt need to a stronger approach to camouflage their products.

4.4 Metamorphism

Unlike the three previous camouflage generations, metamorphic virus has no encrypted part. Therefore, it does not need decryptor, but like polymorphic virus, it employs a mutation engine, as well, Instead of modifying the decryptor loop only, it mutates all its body. The most concise definition of metamorphic viruses is introduced by Igor Muttik “Metamorphics are body-polymorphics” [2]. Each new copy may have different structure, code sequence, size and syntactic properties, but the behavior of the virus does not change. In Figure 5, we illustrate the metamorphic virus propagation scheme.

The first known metamorphic virus that was produced for DOS was ACG, on 1998, and the first efforts on 32-bits metamorphic virus targeting the Portable Executable files was W32.Appartition that spread by 2000 [2].

Anatomy of a metamorphic virus is well explained in [13-14]. A practical metamorphic engine must include the following parts:

1. Disassembler
2. Code analyzer
3. Code transformer
4. Assembler

A complete structure of a metamorphic virus replicator is depicted in figure 6. We create this adapted form from the model introduced by Walnstine et al. in [13].Components of the mutation engine are also displayed. After the virus finds the location of its own code, it needs to convert the code into assembly instruction, which is done by an internal disassembler. The code analyzer is responsible to provide information for code transformer module. The code transformer may need some information such as structure and flow diagram of the program, subroutines, life period of variables and registers, and so on. This information helps the code transformer to work appropriately. Code transformer or obfuscator is the heart of mutation engine. It is responsible to obfuscate the code and change the binary sequence of the virus. In fact, the other modules are designed to prepare the requirements of obfuscation module. It may use all various obfuscation techniques, which are mentioned for the polymorphic virus, as well. The last module, Assembler, converts the new produced mutated assembly code of the virus into machine binary code.

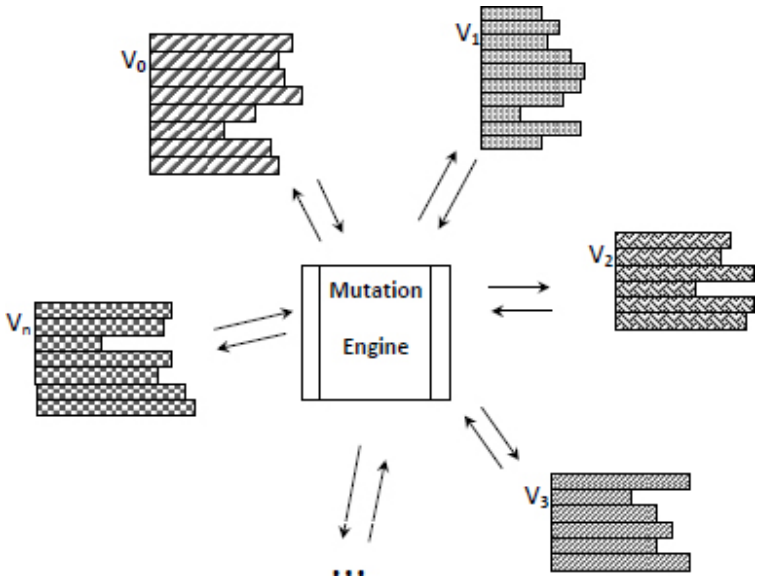

Figure 5: Metamorphic virus propagation scheme

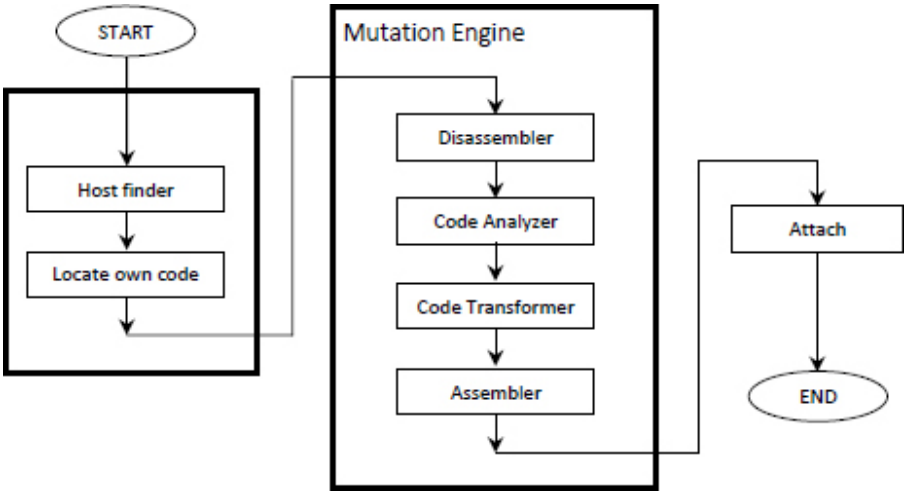

Figure 6: Structure of replicator and mutation engine in metamorphic virus

Well-constructed metamorphic virus does not contain matching string signatures common among its different instances [6]. It means a professional metamorphic malware is able to produce unlimited number of variants, which are similar in behavior and not contain of producing a single pattern vulnerability to be detectable via it. Therefore, antivirus scanning engines must use highly developed heuristics and behavior analysis based detection techniques to catch powerful metamorphic viruses. Although many efforts have done in this area, but a full ideal methodology has not presented yet.

5. Obfuscation Techniques

Different researchers explained and categorized the obfuscation techniques in several ways. Here, we try to cover all the most important techniques, explaining in a clear form and prevent contradiction among them. Most common obfuscation techniques used in malware are [3, 10]:

5.1 Junk/Dead Code Insertion

The insertion of dead code or garbage code is the most effortless solution to modify the binary sequence of the virus program without any effect on the code functionality and behavior [9]. There are various types of garbage codes.

In Table 1, a few examples of junk codes are listed [15]. In this type, the instructions do not change the content of CPU registers or memory and are equal to no-operation (NOP).

Table 1: No-Operation dead code samples

Instruction		Operation
ADD	Reg,0	Reg←Reg+0
MOV	Reg,Reg	Reg←Reg
OR	Reg,0	Reg←Reg | 0
AND	Reg,-1	Reg←Reg&-1

Instructions in Table 1 do not change the value of the operand register; however, they may modify the status of flag register in CPU. For example, adding a zero to a variable or a register, or assigning a register value to itself, do not have any effects on the results of execution.

The second type of this technique, obfuscator place an instruction in the code that probably changes the status of the machine or the content of memory or CPU registers, but before it affects on the result of program, another piece of code undoes it [16]. Two simple examples of this type are listed in Table 2 [10]. A plenty number of nested dead codes makes the analyzing of the code nearly complicated.

Table 2: Reversible dead code samples

Instruction		***Comments***
PUSH	***CX***	*It push value of AX into stack, later it must be turned back to AX before any effects on AX or stack memory*
...	...	
POP	***CX***	
INC	***AX***	*The value of DX increases by 14, and later before any usage of DX, its value must be changed back to its previous value*
...	...	
SUB	***AX, 1***	

However, metamorphic viruses employ composite forms of junk code insertion methods to make the code adequately obscured. The following example in Table 3 is a small part of the W32.Evol virus [2]. The mutation engine of the virus use junk instruction insertion to change the binary sequence of the program code. Table 3 lists two dissimilar instances of W32.Evol [10].

Table 3: Two versions of W32.Evol

Binary Code Sequence	Assembly Code	
C7060F000055	mov	[esi], 5500000Fh
C746048BEC5151	mov	[esi+0004],5151EC8Bh
String Signature:	C7060F000055C746048BEC5151	

Binary Code Sequence	Assembly Code	
BF0F000055	mov	edi,5500000Fh
893E	mov	[esi],edi
5F	pop	edi
52	push	edx
B640	mov	dh,40
BA8BEC5151	mov	edx,5151EC8Bh
53	push	ebx
8BDA	mov	ebx,edx
895E04	mov	[esi+0004],ebx
String Signature:	BF0F000055893E5F52B640BA8BEC5151538BDA895E04	

The table 3 shows that these two forms of the W32.Evol are entirely dissimilar images, but their behaviors are similar. Both of them move the constant value 5151EC8Bh into the memory addressed by

[5500000Fh]. The interesting thing is there is no common string signature to be utilized for static signature-based detection, even using wildcards.

5.2 Variable/Register substitution

Another method used by mutation engines is to exchange registers or memory variables in different instances of the virus. With this technique, the virus tries to defeat the string signature detection, by converting the identical bytes in different generations. W95.Regswap was one of the first viruses that made use of this approach to produce diverse variants of the virus, by December 1998. Obviously, it does not change the behavior of the code, but modifies the binary sequence of the code. Two forms of W95.Regswap are given in Table 4 [10].

The identical bytes in these two instances are highlighted. This technique is not too highly complicated and such wildcards-based scanning can find the morphed versions of the virus simply; there are similar byte sequences as many as necessary to build a signature string.

This relatively simple method in combination with other techniques can produce enough complicated variants of a virus that cannot be easily discovered and make the signature-based detection very impractical.

5.3 Instruction replacement

This technique tries to replace some instructions with their equivalent instructions. Sometimes, a task can be executed in different equal coding instructions set. For example, all different following instructions set the register eax with a zero [10]:

```
mov    eax, 0
xor    eax, eax
and    eax, 0
sub    eax, eax
```

Virus programmers take advantage of this skill in their virus obfuscation engines. it is similar to usage of different synonyms in natural speaking [17].

Table 4: Two versions of W95.Regswap

Binary Code Sequence	Assembly Code	
5A	pop	edx
BF04000000	mov	edi,0004h
8BF5	mov	esi,ebp
B80C000000	mov	eax,000Ch
81C288000000	add	edx,0088h
8B1A	add	ebx,[edx]
899C8618110000	mov	[esi+eax*4+00001118],ebx

Signature: 5ABF040000008BF5B80C00000081C2880000008B1A899C8618110000

Binary Code Sequence	Assembly Code	
58	pop	eax
BB04000000	mov	ebx, 0004h
8BD5	mov	edx, ebp
BF0C000000	mov	edi, 000ch
81C088000000	add	eax, 0088h
8B30	mov	esi, [eax]
89B4BA18110000	mov	[edx+edi*4+00001118],esi

Signature: 58BB040000008BD5BF0C00000081C0880000008B3089B4BA18110000

The example codes in Tables 5, display two different forms of W95.Bistro [2, 10]. Some operations are exchanged by their equivalents, as it is observable in the code. “test esi, esi” is substituted by “or esi, esi”; furthermore, “test edi, edi” is used instead of “or edi, edi” that has the same result, and also, “mov ebp, esp” is exchanged by a couple of consecutive instructions “push esp” and “pop ebp”, that implements similar operation. Obviously, these substitutions metamorphose the binary sequence of program code. Accordingly, the signatures in the given examples of Win95.Bistro are not identical. However, as the other methods, because some fractions of

the binary signatures are alike, scanner can utilize wildcards to detect the variants. The similar parts in the signatures are highlighted in Tables 5.

Table 5: Two Versions of W95.Bistro

Binary Code Sequence	Assembly Code	
55	push	ebp
8BEC	mov	ebp, esp
8B7608	mov	esi, dword ptr [ebp + 08]
85F6	test	esi, esi
743B	je	401045
8B7E0C	mov	edi, dword ptr [ebp + 0c]
09FF	or	edi, edi
7434	je	401045
31D2	xor	edx, edx
Signature:	558BEC8B760885F6743B8B7E0C09FF743431D2	

Binary Code Sequence	Assembly Code	
55	push	ebp
54	push	esp
5D	pop	ebp
8B7608	mov	esi, dword ptr [ebp + 08]
09F6	or	esi, esi
743B	je	401045
8B7E0C	mov	edi, dword ptr [ebp + 0c]
85FF	test	edi, edi
7434	je	401045
28D2	sub	edx, edx
Signature:	55545D8B760809F6743B8B7E0C85FF743428D2	

5.4 Instruction permutation

In many programs, the programmer is able to reorder the sequence of instructions, safely. Through this rearranging process, binary sequences of the code look dissimilar in various generations.

In a situation that some instructions are independent, they can be reorganized in a different order, with no change of the result. Given the following example:

op1 Reg1/Mem1, Reg2/Mem2
op2 Reg3/Mem3, Reg4/Mem4

The above operations can be permutated, If these conditions are exist [18]:

1- Reg1/Mem1 ≠ Reg2/Mem2
2- Reg1/Mem1 ≠ Reg4/Mem4
3- Reg2/Mem2 ≠ Reg3/Mem3

Table 6 display an example, two columns contain same result and code can be arranged in both order, equally [10].

Table 6: Example of Instruction Permutation

Code Order 1	Code Order 2
mov eax, 0F	**add esi, ebx**
push ecx	**mov eax, 0F**
add esi, ebx	**push ecx**

5.5 Code transposition

This approach revise the program structure, in such a way that reorder the program instruction or code flow, but still keeping the execution flow using unconditional or conditional branches. The transformation can be performed on the single instructions level or a code block. Figure 7 illustrate a case of code transposition scheme that is used by Zperm virus [2].

Virtualization obfuscation is another recent method, which is employed by malware creators to defend the malicious code against the reverse engineering [19]. In this technique, instructions and logic of the code are virtualized to hide from analysis. The obfuscator includes a virtual

machine that is required to interpret the logic of the program. This interpreter usually is produced using assembly programming language [20].

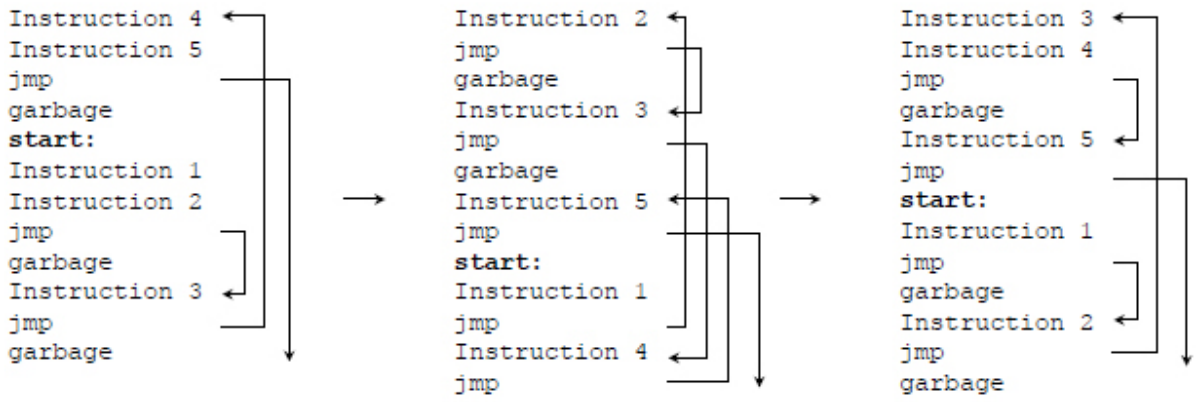

Figure 7: Code transposition in Zperm virus

7. Conclusion

Today, signature-based malware detection is not adequately effective. Because the malware authors attempt to invent advanced camouflage techniques, analysis and study on these techniques by antivirus experts is more required. With the development of information technology, the stealthiness in malware grows up from simple encryption methods to complicated metamorphism. Subsequent to metamorphic virus generation, newer techniques, such as JavaScript malware obfuscation, virtualization obfuscation, and exploit obfuscation are still being developed. Specially, web malware are a serious threat for the increasingly internet users and a significant challenge for antivirus vendors, as well. Consequently, the camouflage techniques in malware need to be investigated and examined by the security specialists, seriously.

Acknowledgement

This study is supported by the Razak School of Engineering and Technology grant funded by University Technology of Malaysia (No. 4B010).

References

[1] M. Sharif, A. Lanzi, J. Giffin *et al.*, "Impeding Malware Analysis using Conditional Code Obfuscation," in The 15th Annual Network and Distributed System Security Symposium (NDSS 2008), San Diego, CA, (2008).

[2] P. Szor, *The Art of Computer Virus Research and Defense*: Addison-Wesley Professional, (2005).

[3] I. You, and K. Yim, "Malware Obfuscation Techniques: A Brief Survey," *Fifth International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA 2010)*. pp. 297-300, (2010).

[4] P. Beaucamps, "Advanced Polymorphic Techniques," *International Journal of Computer Science*, vol. 2, no. 3, pp. 194-205, (2007).

[5] J. Aycock, *Computer Viruses and Malware*, New York, NY, USA: Springer, (2006).

[6] P. O'Kane, S. Sezer, and K. McLaughlin, "Obfuscation: The Hidden Malware," *Security & Privacy, IEEE*, vol. 9, no. 5, pp. 41-47, (2011).

[7] S. Noreen, S. Murtaza, M. Z. Shafiq *et al.*, "Evolvable malware," in Proceedings of the 11th Annual conference on Genetic and evolutionary computation, Montreal, Canada, pp. 1569-1576, (2009).

[8] E. Filiol, *Computer viruses: from theory to applications*, Paris: Springer, (2005).

[9] L. Xufang, P. K. K. Loh, and F. Tan, "Mechanisms of Polymorphic and Metamorphic Viruses," *2011 European Intelligence and Security Informatics Conference (EISIC)*, pp. 149-154, (2011).

[10] B. B. Rad, and M. Masrom, "Metamorphic Virus Variants Classification Using Opcode Frequency Histogram," *LATEST TRENDS on COMPUTERS*. pp. 147-155, (2010).

[11] D. Bruschi, L. Martignoni, and M. Monga, "Code normalization for self-mutating malware," *IEEE Security & Privacy*, vol. 5, no. 2, pp. 46-54, (2007).

[12] M. Jordan, "Dealing with Metamorphism," *Virus Bulletin*, pp. 4-6, October, (2002).

[13] A. Walenstein, R. Mathur, M. Chouchane *et al.*, "The design space of metamorphic malware," *Proceedings of the 2nd International Conference on Information Warfare and Security (ICIW 2007)*. pp. 241-248, (2007).

[14] A. Lakhotia, A. Kapoor, and E. Kumar, "Are metamorphic viruses really invincible?," *Virus Bulletin*, pp. 5–7, December, (2004).

[15] J. M. Borello, and L. Me, "Code obfuscation techniques for metamorphic viruses," *Journal in Computer Virology*, vol. 4, no. 3, pp. 211-220, (2008).

[16] M. W. Bailey, C. L. Coleman, and J. W. Davidson, "Defense against the dark arts," *SIGCSE Bulletin*, vol. 40, no. 1, pp. 315-319, (2008).

[17] A. Karnik, S. Goswami, and R. Guha, "Detecting obfuscated viruses using cosine similarity analysis," *AMS 2007: First Asia International Conference on Modelling & Simulation Asia Modelling Symposium, Proceedings*, pp. 165-170, (2007).
[18] P. Desai, and M. Stamp, "A highly metamorphic virus generator," *International Journal of Multimedia Intelligence and Security* vol. 1, no. 4, pp. 402 - 427, (2010).
[19] K. Coogan, G. Lu, and S. Debray, "Deobfuscation of virtualization-obfuscated software: a semantics-based approach," in Proceedings of the 18th ACM conference on Computer and communications security, Chicago, Illinois, USA, pp. 275-284, (2011).
[20] R. Rolles, "Unpacking virtualization obfuscators," in Proceedings of the 3rd USENIX conference on Offensive technologies, Montreal, Canada, pp. 1-1, (2009).

Babak Bashari Rad is currently a PhD candidate in Computer Science, at University Technology of Malaysia International Campus, Kuala Lumpur. He has completed his Master degree (2002) in Computer Engineering-Artificial Intelligence and Robotic as the outstanding student of Computer Science and Engineering (CSE) department, faculty of engineering at Shiraz University, Iran. He has been working as a faculty lecturer for nine past years in Azad University branches at Iran. His interests and research areas are computer virology, malwares, information security, code analysis, and machine learning methodologies. He is studying on metamorphic virus analysis and detection methodologies for his PhD thesis.

Maslin Masrom received the Bachelor of Science in Computer Science (1989), Master of Science in Operations Research (1992), and PhD in Information Technology/Information System Management (2003). She is an Associate Professor, Razak School of Engineering and Advanced Technology Malaysia, Universiti Technology Malaysia International Campus, Kuala Lumpur. Her current research interests include information security, ethics in computing, e-learning, human capital and knowledge management, and structural equation modeling. She has published articles in both local and international journals such as Information and Management Journal, Oxford Journal, Journal of US-China Public Administration, MASAUM Journal of Computing, ACM SIGCAS Computers & Society and International Journal of Cyber Society and Education.

Suhaimi Ibrahim received the Bachelor in Computer Science (1986), Master in Computer Science (1990), and PhD in Computer Science (2006). He is an Associate Professor attached to Advanced Informatics School (AIS), Universiti Teknologi Malaysia International Campus, Kuala Lumpur. He is an ISTQB certified tester and currently being appointed a board member of the Malaysian Software Testing Board (MSTB). He has published articles in both local and international journals such as the International Journal of Web Services Practices, Journal of Computer Science, International Journal of Computational Science, Journal of Systems and Software, and Journal of Information and Software Technology. His research interests include software testing, requirements engineering, Web services, software process improvement and software quality.