2011 European Intelligence and Security Informatics Conference

Mechanisms of Polymorphic and Metamorphic Viruses

Xufang Li, Peter K.K. Loh
Computer Security Lab
Nanyang Technological University
Singapore
LiXufang, askkloh@ntu.edu.sg

Freddy Tan
Microsoft (Asia) and
International Information Systems Security Certification Consortium
freddyt@microsoft.com

Abstract— **Malware has been generally accepted as one of the top security threats to computer systems around the globe. As malware evolves at a tremendous pace and demonstrates new ways to exploit, infect and victimize the computer systems of enterprises and businesses, remaining economically viable is becoming increasingly difficult. The new trends of malware development are focused on the use of complex and sophisticated code to obstruct analysis as well as spoofing contemporary anti-virus scanners. Polymorphic and metamorphic viruses use the obfuscation techniques to obstruct deep static analysis and defeat dynamic emulators. Malware may also employ metamorphism-based methods, including encryption and decryption engines, multi-packer, garbage code insertion, instruction permutation, code transformation, anti-debugging and virtual machine, registry modification and polymorphic engines. The structural mechanisms of both polymorphic and metamorphic viruses will be presented and discussed in this paper. Finally, the new complex computer viruses such as W32/Fujacks and W32/Vundo were researched as well.**

Keywords-polymorphism, metamorphism, obfuscation, garbage, permutation, decyptor and encryptor, structure

I. Introduction (Heading 1)

A computer virus is a (hidden) program, which invades your computer; much like a biological virus that invades a living cell. A computer virus contains code that has the potential to cause damage and/or perform unwanted/unauthorized functions. Computer viruses have impacted a significant number of computers worldwide over the past few years [1, 2]. One of the most important factors is how to detect complex computer viruses. However, to date, the structures of sophisticated viruses are still unclear and ambiguous.
Contradictory results have been reported for detectable and undetectable computer viruses. The fact that there are no algorithms that can perfectly detect all possible viruses implies that there are computer viruses which no algorithm can detect [3, 22]. In addition, research has also demonstrated that modeling of polymorphic shellcode is intractable by known methods, including both statistical constructs and string signatures [4]. New polymorphic and metamorphic techniques make it more difficult to analyze malware. Existing malware detection techniques based on instruction-level comparison have relatively high miss rates in detecting obfuscated polymorphic and metamorphic malware [5-9]. Many researchers have attempted to detect these complex viruses using dynamic emulator methodologies [10]. Current dynamic analysis methods include registry monitor, API call monitor, file monitor, process monitor, behavior monitor and network monitor [11-13]. A dynamic analysis emulator will execute concurrently with the instrument program and operating system on the same platform. Due to this requirement, it is not always possible to implement a dynamic analysis approach due to scalability issues and /or the analysis environment. New trends on multiple malware research involve machine learning and classification approaches [14-17]. Unfortunately, the mechanisms of polymorphic and metamorphic viruses were not researched in detail by these research works. Others also depicted the normal characteristics of polymorphic and metamorphic viruses with obfuscation, garbage insertion, permutation, polymorphism and metamorphism techniques without directing their efforts to defining structural characteristics [18, 19]. Hence, to-date, the mechanisms of polymorphic and metamorphic viruses have not been fully explained and evaluated yet.

This paper investigates and presents the structural mechanisms of polymorphic and metamorphic viruses with obfuscation, garbage insertion, instruction permutation, code transformation, encryptor, decyptor, polymorphism and metamorphism techniques, as well as methodologies of metamorphic and polymorphic viruses working and evolving. The work was carried out in collaboration with a commercial virus company using static and dynamic analyses for complex computer viruses. The virus algorithms were evaluated to determine their structural models. We derive our motivation from the challenge of successfully modeling these complex computer viruses. Finally, two typical polymorphic and metamorphic viruses are presented with empirically determined structures to clarify how complex computer viruses work, evolve, and replicate themselves.

II. Mechanism of Polymorphic Virus

A polymorphic virus is a computer virus which is capable of mutating itself when it replicates, making it more difficult to identify with antivirus software. It is a sophisticated type of virus that can wreak havoc on a computer system while

978-0-7695-4406-9/11 $26.00 © 2011 IEEE
DOI 10.1109/EISIC.2011.77

avoiding detection. Polymorphic viruses can mutate their decyptors to a large number of different instances that take millions of different form. More sophisticated mutation engines are also used to generate several billions of different decryption routines [18, 20, 21]. Existing polymorphism techniques typically employ two ways of disguising code. The first rewrites the code each time so that it differs syntactically, but retains the same operational semantics. The other approach is to self-cipher; polymorphism is obtained by randomizing the order of these ciphers and using different keys.

A. Polymorphic Engine

A polymorphic engine (called mutation engine or mutating engine) is a computer program that can transform another program into a version that consist of different code with the same functionality by encrypting the target program via different keys and providing a decryption module that can vary widely. The first released polymorphic engine was written by a Bulgarian virus writer nicknamed Dark Avenger.

The mutation engine works in the following way:

```
push [ebp+var_xx]      ; *outBuf (EDI),
call GetSizeOfCode     ;
push ebx               ; sizeOfCode
call SeekStartOfVirus
push ebx               ; *inBuf (ESI)
call MetaEngine
cmp ebx, 0
jz   short EngineFailed
```

The input buffer is a pointer to the virus code in the *inBuf, the output buffer is a polymorphic decryption routine with an encrypted virus body in the *outBuf**,** sizeOfCode is the size of the encrypted virus code. Where ebp and ebx are the general registers to initial the base address for the buffer.

B. Polymorphic Encryptors

Simple encryption:

- *"XOR", "ADD",* A becomes B, B becomes C, C becomes D, where A, B, C, D are the viral encrypted codes

Advanced encryption

- *XYZ, X is encrypted by A, Y is encrypted by B, Z is encrypted by C.*

Long character key for the encryption, decrypt the code with another long key (the key is hard to guess)

- *WVYYX, key is ABCD, Encrypt W with A, V with B, first Y with C, next Y with A, X with B.*

Mutation encrypt is a multiple encrypt method. The code is encrypted again with the same or a different key.
Transposition encrypt is definitely weak because a random key could construct the original opcode.

C. Polymorphic Decryptors

All ciphering algorithms are not equivalent. Some behave like an XOR with a short key and is easy to decrypt. Another well-tested algorithm is the Random Decryption Algorithm (RDA), in which there is no way to retrieve the key.

The weak crypto:

- Deterministic RDA: the computation time is always the same (W32/Crypto)
- Non-deterministic RDA: the computation time cannot be guessed (W32/IHSix)

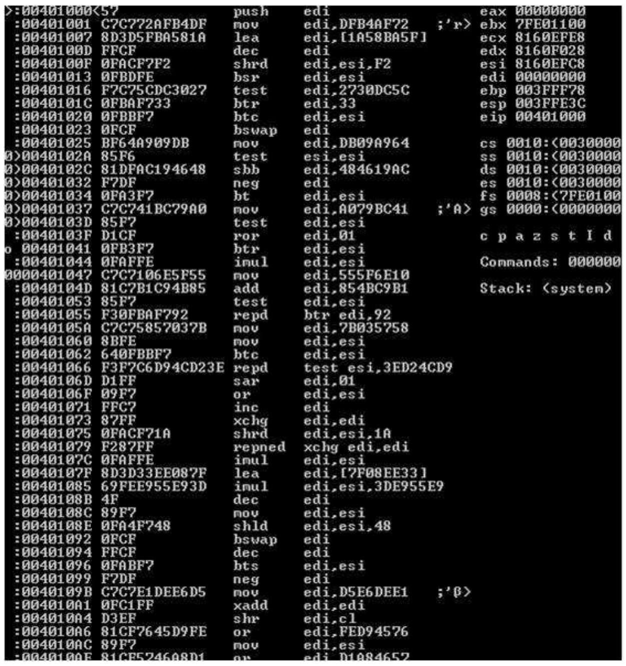
The strong crypto:

- A deciphering function D gathers the information needed to compute the key and decipher the corresponding code.
- Encrypted code EVP_1 (key k_1) contains all the anti-viral mechanisms.
- Encrypted code EVP_2 (key k_2) is in charge of the infection and polymorphism/metamorphism mechanisms.
- Encrypted code EVP_3 (key k_3) contains all the optional payloads.

Pseudo-random generation algorithm (PRGA):

- The PRGA modifies the state and outputs a byte of the key stream. In each iteration, the PRGA increments *i*, adds the value of S pointed to by *i* to *j*, exchanges the values of S[*i*] and S[*j*], and then outputs the element of S at the location S[*i*] + S[*j*] (modulo 256). Each element of S is swapped with another element at least once every 256 iterations [22]

It is perfect and used in the virus of W32/Crypto.



```
>:00401000<57            push    edi                     eax 00000000
 :00401001 C7C772AFB4DF  mov     edi,DFB4AF72    ;'r>   ebx 7FE01100
 :00401007 8D3D5FBA581A  lea     edi,[1A58BA5F]         ecx 8160EFE8
 :0040100D FFCF          dec     edi                    edx 8160F028
 :0040100F 0FACF7F2      shrd    edi,esi,F2             esi 8160EFC8
 :00401013 0FBDFE        bsr     edi,esi                edi 00000000
 :00401016 F7C75CDC3027  test    edi,2730DC5C           ebp 003FFF78
 :0040101C 0FBAF733      btr     edi,33                 esp 003FFE3C
 :00401020 0FBBF7        btc     edi,esi                eip 00401000
 :00401023 0FCF          bswap   edi
 :00401025 BF64A909DB    mov     edi,DB09A964           cs 0010:<0030000
0>0040102A 85F6          test    esi,esi                ss 0010:<0030000
0>0040102C 81DFAC194648  sbb     edi,484619AC           ds 0010:<0030000
0>00401032 F7DF          neg     edi                    es 0010:<0030000
0>00401034 0FA3F7        bt      edi,esi                fs 0008:<7FE0100
0>00401037 C7C741BC79A0  mov     edi,A079BC41    ;'A>   gs 0000:<0000000
0>0040103D 85F7          test    edi,esi
 :0040103F D1CF          ror     edi,01                 c p a z s t I d
o 00401041 0FB3F7        btr     edi,esi
 :00401044 0FAFFE        imul    edi,esi                Commands: 000000
000401047 C7C7106E5F55   mov     edi,555F6E10
 :0040104D 81C7B1C94B85  add     edi,854BC9B1           Stack: (system)
 :00401053 85F7          test    edi,esi
 :00401055 F30FBAF792    repd    btr edi,92
 :0040105A C7C75857037B  mov     edi,7B035758
 :00401060 8BFE          mov     edi,esi
 :00401062 640FBBF7      btc     edi,esi
 :00401066 F3F7C6D94CD23E repd   test esi,3ED24CD9
 :0040106D D1FF          sar     edi,01
 :0040106F 09F7          or      edi,esi
 :00401071 FFC7          inc     edi
 :00401073 87FF          xchg    edi,edi
 :00401075 0FACF71A      shrd    edi,esi,1A
 :00401079 F287FF        repned  xchg edi,edi
 :0040107C 0FAFFE        imul    edi,esi
 :0040107F 8D3D33EE087F  lea     edi,[7F08EE33]
 :00401085 69FEE955E93D  imul    edi,esi,3DE955E9
 :0040108B 4F            dec     edi
 :0040108C 89F7          mov     edi,esi
 :0040108E 0FA4F748      shld    edi,esi,48
 :00401092 0FCF          bswap   edi
 :00401094 FFCF          dec     edi
 :00401096 0FABF7        bts     edi,esi
 :00401099 F7DF          neg     edi
 :0040109B C7C7E1DEE6D5  mov     edi,D5E6DEE1    ;'β>
 :004010A1 0FC1FF        xadd    edi,edi
 :004010A4 D3EF          shr     edi,cl
 :004010A6 81CF7645D9FE  or      edi,FED94576
 :004010AC 89F7          mov     edi,esi
 :004010AE 81CF5246A8D1  or      edi,D1A84652
```

Figure 1. Demo of Entry Point Obfuscation

III. Mechanism of Metamorphic Virus

A metamorphic virus is capable of rewriting its own code with each infection, or generation of infection, while maintaining the same functionality [19]. Metamorphic code is code that can reprogram itself. Often, it does this by translating its own code into a temporary representation, editing this representation and rewriting itself back to normal codes [27].

A. *General Obfuscation*

Code obfuscation techniques are exploited by polymorphic and metamorphic viruses to hide their behavior from antivirus scanners. Due to the reversal protection benefits, several obfuscation techniques have been developed and investigated to generate different computer virus forms via garbage insertion, code transformation, and registry modification etc. Some methods manipulate the aggregation of control or data, while others affect the ordering. Garbage code insertion is a simple technique used by metamorphic and polymorphic engines to change the byte sequence of viral code. They insert code with no effect or function like NOP codes and/or other complex codes that are presented in the literal of the viral Darwinism of W32.Evol as follows:

```
MOV r32, [ebp+Random8]
  - MOV r32, Random32
  - OP r32, Random32; OP = ADC/ADD/AND/OR/SBB/SUB/XOR
  - MOV RandomReg8, Random8
```

Instruction transformation: actual transformations performed by the metamorphic engine on itself are as follows:

```
  B9 00 10 00 00        mov ecx, 1000h
```

Transformed to:

```
  B9 10 B2 00 3C        mov ecx, 3C00B210h
  81 C1 F0 5D FF C3     add ecx, 0C3FF5DF0h; ecx = 1000h
```

B. *Entry Point Obfuscation (EPO)*

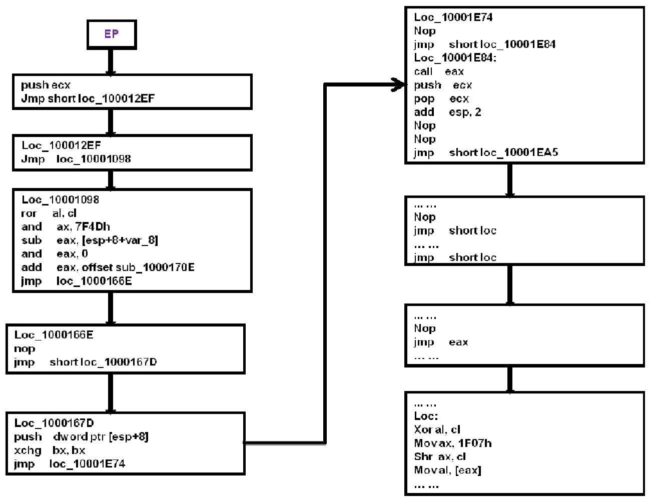
Entry Point Entry Obfuscation (EPO) is a technique used by malware authors to dissuade AV scanners from investigating the files that have been infected. For a virus to activate and acquire control, it needs to place itself within the line of execution. Traditionally this was done by changing the entry point into the target executable to first point to the virus code, which will presumably, at some point, release control back to the host executable. EPO-enabled malware will patch the target executable somewhere in the middle of its execution train with jmp/call instructions and receive control that way [28]. By doing this, EPO will fool the AV scanner that looks for a modified entry point as part of its heuristics engine, as shown in Fig.1. At entry point (EP) with address 401000, the viral codes are obfuscated with different and complex transformation EDI instructions. Some instructions are abnormal with infrequent usage in normal files as *bswap* and *repned*. These are generated by the metamorphic engine to get more forms but with no effect on real file execution. These instructions are inserted after *push* instructions, so we assume that registers will be modified later on. Hence, it is difficult for anti-virus researchers to capture them via a single signature.

C. *Code Transposition – W32/Vundo*

Code transposition can modify the structure of a program at the instruction or module level by physically recoding it, while reversing/altering the execution flow using conditional jumps or unconditional branches as shown in Fig.2 with the W32/Vundo virus. The W32/Vundo engine will generate instructions that are not reliant upon the original code, and of which, the functionality is essentially "do-nothing". However, these instructions actually do alter the original code flow as they are random and inserted in the gap places of jump flows. The control flow methods are used in this kind of viruses with jmp flows [24, 25, 26]. The generation of W32/Vundo virus looks like this:

```
Push ecx ; Entry Point, instructions modified in following insertion
Jmp instruction1
instruction1; Garbage codes insertion eg Nop and/or other transformations
Jmp instructions2
instructions2
Jmp instructions n
Entropy Data ; Packed data, compressed data or encrypted data
```

The measurement of randomness or unpredictability in an event sequence or series of data value was investigated to distinguish between normal files and abnormal files [5]. But it is feasible to absolutely determine the entropy data, which can minimize both false positives and false negatives.



```
EP

push ecx
Jmp short loc_100012EF

Loc_100012EF
Jmp   loc_10001098

Loc_10001098
ror    al, cl
and    ax, 7F4Dh
sub    eax, [esp+8+var_8]
and    eax, 0
add    eax, offset sub_1000170E
jmp    loc_1000166E

Loc_1000166E
nop
jmp    short loc_1000167D

Loc_1000167D
push   dword ptr [esp+8]
xchg   bx, bx
jmp    loc_10001E74

Loc_10001E74
Nop
jmp    short loc_10001E84
Loc_10001E84:
call   eax
push   ecx
pop    ecx
add    esp, 2
Nop
Nop
jmp    short loc_10001EA5

... ...
Nop
jmp    short loc
... ...
jmp    short loc

... ...
Nop
jmp    eax
... ...

... ...
Loc:
Xor al, cl
Movax, 1F07h
Shr ax, cl
Moval, [eax]
... ...
```

Figure 2. Instruction Permutation and Transpoistion

D. *Host Code Mutation*

A metamorphic engine is used to transform executable (binary) code. The behavior of such an engine varies from virus to virus, but many elements remain the same. A metamorphic engine has to implement some sort of an internal disassembly in order to parse the input code. After disassembly, the engine will transform the program code and produce new code that retains its functionality and yet looks different from the original code. The virus body will not only mutate itself in new generations, but it also mutates the code of the host. To do this, the virus uses a randomly executed code-morphing routine.

E. *Anti-Debugging*

Anti-debugging techniques are ways for a program to defend itself if it runs under control of a debugger. They are used by commercial executable protectors, packers and malicious software to prevent or slow-down the process of reverse-engineering. The following morph codes appeared in the Evol virus with a breakpoint over its mutation codes. If the engine finds a breakpoint on debugging, it will jump to the following routine that will result in a crash.

```
AntiDebug:
cmp byte ptr [ebx+7], 0BFh ; are we in kernel mode?
jnz short ret_AntiDebug
mov ecx, 1000h              ; counter = 1000h
mov edi, 40000000h
or edi, 80000000h
add edi, ecx                ; edi = C0001000h
rep stosd                   ; copy bytes to [edi]
ret_AntiDebug:
retn                        ; this will result in a crash
```

F. *W95 / Zmist Code Integration*

W95/Zmist was written by Zombie, a Russian legal polymorphic and metamorphic virus writer. It is also an advanced metamorphic virus with entry point obfuscation, polymorphic decryptor, anti-heuristics, and code integration techniques. The following are characteristics of this virus.

- Entry point obscuring virus
- Randomly use an additional polymorphic decryptor
- Code integration technology
- Mistfall engine – it is capable of decompiling portable executable files to its small elements
- Regenerate code and data references, including the relocation information and rebuild the executable
- Insert jump instructions after every single instruction of the code section
- Perfect anti-heuristics virus: Instead of alerting infected codes directly by making the section writable, an anti-heuristic trick is used for decrypting the virus codes.
- Reliable

W32/Zmist virus implemented an even more sophisticated technique, named *code integration* by the researcher Peter Ferrie and Peter Szor, which has never been seen in any previous virus [29]. Zmist's engine can decompile portable executable (PE) files to their smallest elements, requiring 32MB of memory. Then, the virus moves code blocks out of the way, inserts itself into the block, regenerates codes and data references, and rebuilds the execution as shown in Fig.3. Code integration and morphing techniques are presented and exploited by metamorphic and polymorphic viruses in the form of cavity infection, multi-packers, payload, and stealth etc.

IV. METHODOLOGY OF W32/FUJACKS

W32/Fujacks is a variant of metamorphic virus, which is famous for the panda icon after infection. This virus is written by a 25-year old, believed to be "WhBoy", the infamous nickname that is embedded in most variants of W32/Fujacks.

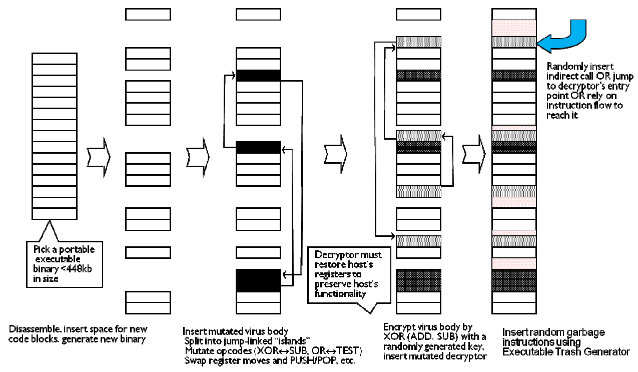

Pick a portable executable binary <448kb in size
Randomly insert indirect call OR jump to decryptor's entry point OR rely on instruction flow to reach it
Decryptor must restore host's registers to preserve host's functionality
Disassemble, insert space for new code blocks, generate new binary
Insert mutated virus body
Split into jump-linked "islands"
Mutate opcodes (XOR↔SUB, OR↔TEST)
Swap register moves and PUSH/POP, etc.
Encrypt virus body by XOR (ADD, SUB) with a randomly generated key, insert mutated decryptor
Insert random garbage instructions using Executable Trash Generator

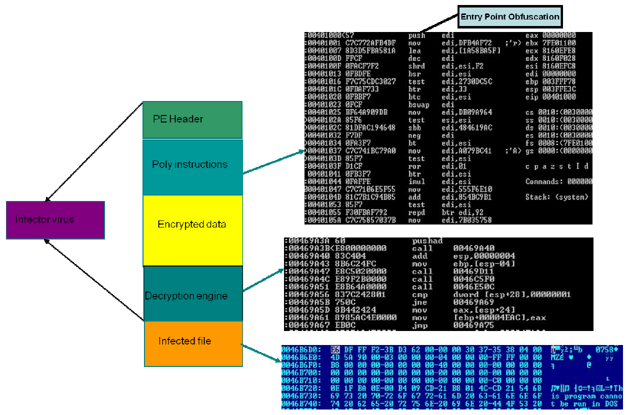
Figure 3. Zmist decompiling PE file and integrating codes

A. *Mechanism of W32/Fujacks*

Fujacks is also another advanced metamorphic virus with morphing techniques. It has been propagated in the wild in china. The characteristics are as follows:

- Entry point obfuscation (EPO): Advanced obfuscation viral codes are inserted after the PUSH instructions to emulate later in other forms.
- Garbage code insertion: they strength their effects on the seldom usage instructions and registers modification technique to the junk codes insertion.
- Instruction permutation: Instructions are permutated after the PUSH EDI instructions.
- Register randomization (*EDI, and ESI)*: The other forms have different obfuscated codes based on the random registers changed.
- Anti-debugger
- Anti-Virtual Machine
- Internally packed: The packer are organized and hidden in the obfuscated codes to avoid the deep unpacking.

Algorithmic Decompression: a typical algorithm is generated to decompress the files, then infecting all the packing or compressed files. Fig.4 shows the structural model of the structure of Fujacks virus with extra/additional poly instructions, encrypted data, decryption engine and infected files. The infector virus concludes four elements (PE header, Poly instructions, Encrypted data, Decryption engine). In poly instructions part, the complex transformation EDI instructions are inserted after the entry point PUSH instructions, which modify registers in the later emulation forms with the entry point obfuscation technique. The junk codes are inserted after the entry point with PUSH EDI instruction. Below the junk codes and poly instructions element, the encrypted data is packed with the multi-packers, which is restored in the data section. The packer is included in the deception engine element with PUSHAD instructions, which encrypt the main viral codes internally. The encryptor and decryptor engines are also packed in the complex codes. After decrypting the viral codes, Fujacks runs to infect the host files and makes more multi-vector threats. The infected files are shown in blue with the MZ label.



```
:00401000<57              push   edi                  eax 00000000
:00401001 C7C772AFB4DF    mov    edi,DFB4AF72   ;'r>  ebx 7FE01100
:00401007 8D3D5FBA581A    lea    edi,[1A58BA5F]       ecx 8160EFE8
:0040100D FFCF            dec    edi                  edx 8160F028
:0040100F 0FACF7F2        shrd   edi,esi,F2           esi 8160EFC8
:00401013 0FBDFE          bsr    edi,esi              edi 00000000
:00401016 F7C75CDC3027    test   edi,2730DC5C         ebp 003FFF78
:0040101C 0FBAF733        btr    edi,33               esp 003FFE3C
:00401020 0FBBF7          btc    edi,esi              eip 00401000
:00401023 0FCF            bswap  edi
:00401025 BF64A909DB      mov    edi,DB09A964         cs 0010:(0030000
>0040102A 85F6            test   esi,esi              ss 0010:(0030000
>0040102C 81DFAC194648    sbb    edi,484619AC         ds 0010:(0030000
>00401032 F7DF            neg    edi                  es 0010:(0030000
>00401034 0FA3F7          bt     edi,esi              fs 0008:(7FE0100
>00401037 C7C741BC79A0    mov    edi,A079BC41   ;'A>  gs 0000:(0000000
>0040103D 85F7            test   edi,esi
:0040103F D1CF            ror    edi,01               c p a z s t I d
 00401041 0FB3F7          btr    edi,esi
:00401044 0FAFFE          imul   edi,esi              Commands: 000000
00401047 C7C7106E5F55     mov    edi,555F6E10
:0040104D 81C7B1C94B85    add    edi,854BC9B1         Stack: (system)
:00401053 85F7            test   edi,esi
:00401055 F30FBAF792      repd   btr edi,92
:0040105A C7C75857037B    mov    edi,7B035758
```

```
:00469A3A 60                pushad
:00469A3B<E800000000        call    00469A40
:00469A40 83C404            add     esp,00000004
:00469A43 8B6C24FC          mov     ebp,[esp-04]
:00469A47 E8C5020000        call    00469D11
:00469A4C E89F2B0000        call    0046C5F0
:00469A51 E8B64A0000        call    0046E50C
:00469A56 837C242801        cmp     dword [esp+28],00000001
:00469A5B 750C              jne     00469A69
:00469A5D 8B442424          mov     eax,[esp+24]
:00469A61 8985AC4E0000      mov     [ebp+00004EAC],eax
:00469A67 EB0C              jmp     00469A75
```

```
0046B6D0:  E6 DF FF F2-3B D3 62 00-00 00 30 37-35 38 04 00   [illegible]    0758
0046B6E0:  4D 5A 90 00-03 00 00 00-04 00 00 00-FF FF 00 00   MZÉ ♥    ♦    ÿÿ
0046B6F0:  B8 00 00 00-00 00 00 00-40 00 00 00-00 00 00 00   ╕       @
0046B700:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
0046B710:  00 00 00 00-00 00 00 00-00 00 00 00-C0 00 00 00               └
0046B720:  0E 1F BA 0E-00 B4 09 CD-21 B8 01 4C-CD 21 54 68   [illegible]!Th
0046B730:  69 73 20 70-72 6F 67 72-61 6D 20 63-61 6E 6E 6F   is program canno
0046B740:  74 20 62 65-20 72 75 6E-20 69 6E 20-44 4F 53 20   t be run in DOS
```

Figure 4. Structure of W32 / Fujacks

B. *Attacking approach*

Fujacks doesn't raise an alert to the entry point of the host. Instead it appends itself to the code. They are multi-vector threats, usually including an aggressive downloader that updates itself frequently. They can infect both executable and non-executable files over insecure media such as open network shares and USB drives, thus slipping through the cracks of loosely managed IT policies. Once successful, trusted media files can be further infected with malicious code or hyperlinks through PE file infection, web-based exploits over HTML or media files targeted against dispatched and vulnerable applications. The characteristics are as follows:

- File size: 173,580 bytes
- A variant of metamorphic virus
- Infect the PE, Compress files (.RAR.ZIP et al) and possibly HTML files with malicious hyperlinks of windows ANI *0-day exploit*
- Upon execution. It drops itself (TXPlatform.exe) and infected files (regedit.exe)
- Create Desktop_1.ini: Located in infected folder and tag the infected data-detected as *W32/Fujacks.remnants virus (McAfee)*
- Autorun worm virus
- Infected type: Pre-pending

Fig.5 shows a typical process of virus infection and execution on the host files to illustrate how Fujacks works and evolves. There are three types of infecting methodology (pre-pending, appending, and cavity). Cavity is more complex than others but Fujacks just uses pre-pending method to infect all possible host files without internal codes changing. Fujacks makes more obfuscation on the entry point, packing itself with multi-packers. Firstly, it runs itself on the entry point obfuscation viral codes to defeat the reversing and detection; meanwhile, other mechanisms are exploited to anti debugging and anti virtual machine running the codes. While unpacking main functional codes that is loaded in the data section, it will drop another similar malicious code and attach itself to the host PE header. It will infect all files launched on all the folders, decomposing files, downloading viral codes, updating itself, hyper linking malicious URL on HTML and updating itself until all the folders are labeled with desktop.ini.

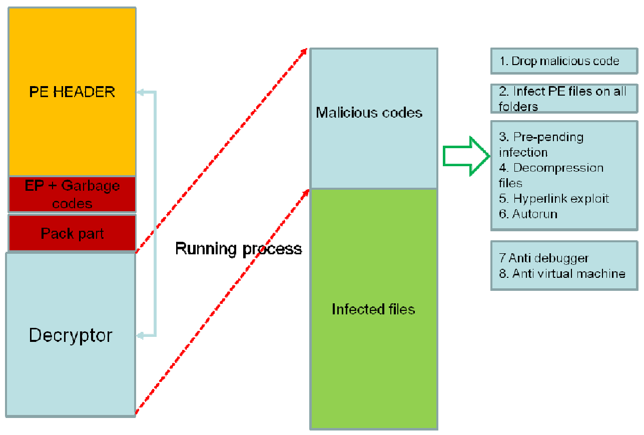


Figure 5. Infecting and Executing of W32 / Fujacks

V. Conclusion

In this paper, we present the mechanisms of polymorphic and metamorphic viruses with more complex polymorphism and metamorphism techniques like encryptor and decryptor, entry point obfuscation, general obfuscation, anti-debugger, code integration, code transposition, host code mutation, polymorphic and metamorphic engines etc. A novel mechanism of Fujacks virus is presented to demonstrate the methodology of how the morphing viruses work and construct themselves to defeat scanners and deep reversing. The polymorphic and metamorphic creations will come very close to the concept of a theoretically undetectable virus. In future, we will pay more attention to the detection of these morphing codes, saying no to the virus writers and stepping ahead of them once again.

References

[1] Information Technology Security Report Lead Agency Publication R2-002, Future Trends in Malicious Code – 2006 Report.

[2] McAfee. McAfee virtual criminology report, Virtually Here: The Age of Cyber Warfare, McAfee, Inc, 2009.

[3] David M. Chess and Steve R. White, An Undetectable Computer Virus, Virus Bulletin Conference, Sep 2000, IBM Thomas J. Watson Research Center, Hawthorne, New York, USA.

[4] Song Y, Locasto ME, Stavrou A, Keromytis AD, Stolfo SJ, On the infeasibility of modeling polymorphic shellcode, Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS), 2007.

[5] Robert Lyda and Jim Hamrock, Using Entropy Analysis to Find Encrypted and Packed Malware, IEEE Security & Privacy, vol 5, pp. 40-45, 2007.

[6] Adrian E. Stepan, Defeating Polymorphism beyond Emulation, Virus Bulletin Conference, October 2005.

[7] Igor Santos, Felix Brezo, Javier Nieves, et al, Idea: Opcode-Sequence-Based Malware Detection, Lecture Notes in Computer Science, Engineering Secure Software and System, vol 5969, pp. 35-43, 2010.

[8] Yong Tang, Bin Xiao, Xicheng Lu, Using a bioinformatics approach to generate accurate exploit-based signatures for polymorphic worms, Journal of Computer & Security, vol 28, pp. 827-842, 2009.

[9] Yong Tang, Bin Xiao, Xicheng Lu, Using a bioinformatics approach to generate accurate exploit-based signatures for polymorphic worms, Journal of Computer & Security, vol 28, pp. 827-842, 2009.

[10] Ulrich Bayer, Engin Kirda and Christopher Kruegel, "Improving the Efficiency of Dynamic Malware Analysis", Proceedings of the 2010 ACM Symposium on Applied Computing, Track on Information Security Research and Applications, Lusanne, Switzerland, March 2010.

[11] Alsagoff, S.N, Malware self protection mechanism issues in conducting malware behavior analysis in a virtual environment as compared to a real environment, 2010 International Symposium in Information Technology (ITSim), vol 3, pp.1326-1331, September 2010.

[12] Ulrich Bayer, Engin Kirda and Christopher Kruegel, Improving the Efficiency of Dynamic Malware Analysis, 25th Symposium on Applied Computing (SAC), Track on Information Security Research and Applications, 2010.

[13] Younghee Park, Reeves, D.S. Identification of Bot Commands by Run-Time Execution Monitoring, Computer Security Applications Conference, ACSAC'09. pp. 321-330, 2009.

[14] Roberto Perdisci, Andrea Lanzi and Wenke Lee, McBoost: Boosting Scalability in Malware Collection and Analysis Using Statistical Classification of Executables, Proceedings of the 2008 Annual Computer Security Applications Conference, pp. 301-310, 2008.

[15] Gregory Conti, Sergey Bratus, Anna Shubina, et al, Automated mapping of large binary objects using primitive fragment type classification, The proceedings of the tenth annual Digital Forensics Research conference, vol 7, pp. (7) S3-S12, August 2010.

[16] Lorenzo Cavallaro, Prateek Saxena and R.Sekar, On the limits of information flow techniques for malware analysis and containment, Lecture Notes in Computer Science, Vol 5137, Detection of Intrusion and Malware, and Vulnerability Assessment, pp. 143-163, 2008.

[17] Andrew Walenstein, Rachit Mathur, Mohamed R. Chouchane, Arun Lakhotia, Constructing malware normalizes using term rewriting, Journal of Computer Virology, pp. (4) 307-322, 2008.

[18] Understanding and Managing Polymorphic Viruses, the Symantec Enterprise Papers (1996).

[19] Szor, P and Ferrie, P, Hunting for Metamorphic, Virus Bulletin Conference (2003).

[20] Evgenios Konstantinou, Stephen Wolthusen, Metamorphic Virus: Analysis and Detection, Technical Report, RHUL-MA-2008-02.

[21] Polymorphic Code: http://en.wikipedia.org/wiki/Polymorphic_code

[22] Pseudo-random generation algorithm: http://en.wikipedia.org/wiki/RC4

[23] Spinellis, D, Reliable identification of bounded–length viruses is NP-complete. Information Theory, IEEE Transactions, Vol 49, pp. 280-284, January 2003.

[24] Danilo Bruschi, Lorenzo Martignoni, Mattia Monga, Detecting self-mutating malware using control-flow graph matching, Lecture Notes in Computer Science, Detection of Intrusions and Malware & Vulnerability Assessment, Vol 4046, PP. 129-143, 2006.

[25] Arlington, Virginia, Malware Detection and Classification From Byte-Patterns to Control Flow Structures to Entropic Defenses, Cyber Genome Project DARPA workshop, University of New Orleans, Department of Computer Science. December 2009.

[26] Metamorphic Code: http://en.wikipedia.org/wiki/Metamorphic_code

[27] Sharif, M., Lanzi, A., Giffin, J., Lee, W, Impeding malware analysis using conditional code obfuscation, In Network and Distributed System Security Symposium. San Diego, CA, 2008.

[28] Peter Szor, Zmist opportunities, Virus Bulletin Conference, pp. 6-7, March 2001.