## Lab 08-1.malware

## 1. What anti-disassembly techniques are used in the binary?

The binary uses rogue bytes after unconditional jumps which would cause errors in disassemblers that use the linear disassembly strategy. The disassemblers would erroneously try to read these bytes as instructions. Flow-oriented disassemblers like IDA would never reach these, thus never be confused by them. An example is shown below

Additionally, the malware uses several techniques that make analysis more difficult. Most functions in the malware have several misdirecting jumps before they can get called. Lastly, the malware hides its true functionality from plain sight by using SEH (Structured Exception Handling). The malware pushes the function it *really* wants to call into a designated exception handling spot on the stack (the first element of the Thread Information Block, [FS:0x00]), and then intentionally creates an exception by dividing by 0.

```
; CUDE AKEF: SUD_4110551]

sub ecx, ecx

push offset mal_func_jump_func

push large dword ptr fs:0

mov large fs:0, esp

div ecx

add esp, 4

retn

endp; sp = -4
```

S

# 2. What anti-debugging techniques are used in the binary?

The malware makes a call to *IsDebuggerPresent*, and if it detects a debugger, it immediately exits. If it does not detect a debugger, it calls a function which calls *rdtsc*, which gets the number of cycles since the computer has started. It compares this with 0xFFFFF, and if the computer has had less cycles than this (1,048,575), it will continue successfully to the malicious function. If the computer has had more cycles than this, it will jump to the middle of an instruction, intentionally crashing the program. It is

unclear to me why and how the malware author intended to use the rdtsc instruction here.

```
  Image: Section 1

  Image: Section 2

  Image: Section 3

                                          00411300
                                          00411300
                                          00411300
                                           00411300
                                                                                                                         rdtsc_check_func proc near
                                           00411300
                                           00411300
                                                                                                                         ; FUNCTION CHUNK AT 00411136 SIZE 00000005 BYTES
                                           00411300
                                           00411300 OF 31
                                                                                                                         rdtsc
                                           00411302 2B C1
                                                                                                                         sub
                                                                                                                                                eax. ecx
                                          00411304 3D FF FF 0F 00
                                                                                                                                                eax, OFFFFFh
                                                                                                                         CMD
                                           00411309 OF 86 27 FE FF FF jbe
                                                                                                                                                1oc 411136
J
N
■
 0041130F
                                                                                                                                                     ; jumps to middle of instruction, causing crash
0041130F
                                                                              1oc 41130F:
                                                                                                     short near ptr loc_411329+3
0041130F EB 1B
                                                                              jmp
0041130F
                                                                              rdtsc_check_func endp
0041130F
                                                                                🛗 N 👊
                                                                                                                                                               ; START OF FUNCTION CHUNK FOR rdtsc check func
                                                                               00411136
                                                                                00411136
                                                                                00411136
                                                                                                                                                              1oc_411136:
                                                                                                                                                                                     sub_4112E0
                                                                                00411136 E9 A5 01 00 00
                                                                                                                                                              jmp
                                                                                                                                                                  END OF FUNCTION CHUNK FOR rdtsc_check_func
                                                                               00411136
```

This is an atypical use of the rdtsc instruction, as usually the instruction is called twice to calculate the cycle difference between two points in code. If a debugger was present, this cycle difference would be much greater than normal. Once successfully passing the rdtsc anti-debugging trick, the malware calls its malicious function.

#### 3. What does the sample do?

The malicious subroutine creates a file called *kingkai.bat* in the current users /AppData/Roaming/Microsoft/Windows/Start Menu/Programs/Startup directory, and fills it with the contents:

# @echo off shutdown /s /f /t 0

This attempts to shut down the computer every time it starts up.

#### 4. Patch the PE to bypass any techniques found.

The following code patches show how to remove the anti-debugging features from this sample.

<b>™</b> N ∪L	
00411320	
00411320	
00411320	; Attributes: bp-based frame
00411320	
00411320	_main proc near
00411320 55	push ebp
00411321 8B EC	mov ebp, esp
00411323 83 EC 40	sub esp, 40h
00411326 <mark>53</mark>	push ebx
00411327 <mark>56</mark>	push esi
00411328 <u>57</u>	push edi
00411329 FF 15 34 70 41 00	call ds:IsDebuggerPresent
0041132F 85 C0	test eax, eax
00411331 <mark>74 04</mark>	jz short loc_411337

Before Figure

<b>⊞N</b> ₩	
00411320	
00411320	
00411320	; Attributes: bp-based f
00411320	
00411320	_main proc near
00411320 <mark>55</mark>	push ebp
00411321 8B EC	mov ebp, esp
00411323 83 EC 40	sub esp, 40h
00411326 <mark>53</mark>	push ebx
00411327 <mark>56</mark>	push esi
00411328 57	push edi
00411329 90	nop
0041132A <mark>90</mark>	nop
0041132B 90	nop
0041132C 90	nop
0041132D 33 C0	xor eax, eax
0041132F 85 C0	test eax, eax
00411331 74 04	jz short loc_411337

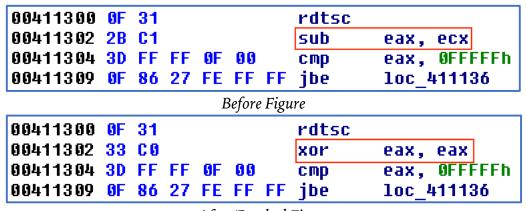
After/Patched Figure

The call to *IsDebuggerPresent* takes up 6 bytes, so I overwrote it with 4 NOPs and a **xor eax**, **eax**, which sets the eax register to 0. By doing so, the next instruction, **test eax**, **eax** always sets the zero flag, and the subsequent conditional jump-if-zero always occurs, which is what happens with the original malware at this point when it doesn't detect a debugger.

Additionally, since this patch address originally contained a function call, this address is in the relocation table, which will likely alter the patched bytes after being loaded into memory. As such, I used a tool called RelocEditor

(<a href="http://www.woodmann.com/collaborative/tools/index.php/RelocEditor">http://www.woodmann.com/collaborative/tools/index.php/RelocEditor</a>) to remove this entry from the table and ensure the **xor eax**, **eax** instruction didn't get altered.

The rdtsc patch was slightly easier. Instead of actually using the return value from the rdtsc instruction (which gets put into edx:eax), I just used the **xor** eax, eax instruction again to guarantee that the next jump instruction would jump to the 'true' path. The 'true' path leads to the malicious function, as shown in the last figure.



After/Patched Figure

