MalwareAnalysisReports/WikiLoader/WikiLoader Shellcode pt2.md at main · VenzoV/MalwareAnalysisReports

By VenzoV

Archived: 2026-04-05 19:27:16 UTC

Summary part 1

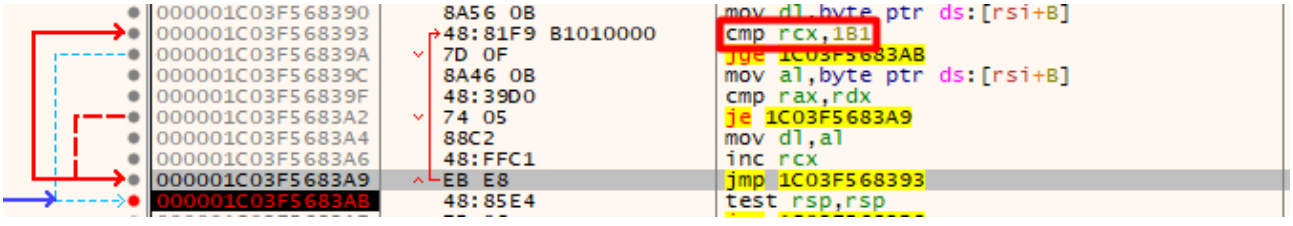
In part one we looked at how shellcode was decrypted by using the Micorsoft Bcrypt library. AES CBC mode was used to decrypt shellcode located in the file "certificate.pem". Once this was done a new thread is created and entry point changed to point to the newly decrypted payload allocated within the memory. We switched threads and followed the shellcode.

Overview

- Loading bingmaps.dll
- Long busy loop to slow down execution.
- Retrieving once again API via PEB walking
- Function used to load API from: Kernel32.dll
- Function Used to load native API to perform indirect syscalls: ntdll.dll
- Checks if native calls are hooked.
- New thread is created and execution is switched, the thread points to bingsmap.dll and jumps back and forth to shellcode.

Loading bingsmap.dll & First Indirect syscall

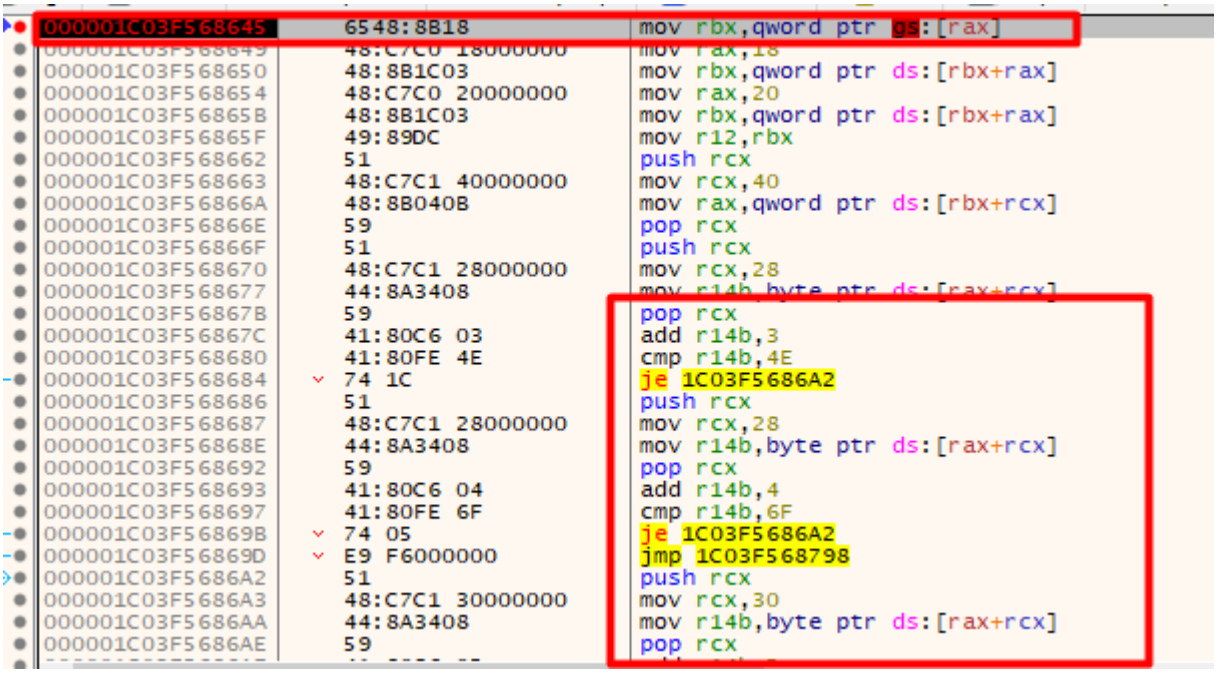
The shellcode enters firstly into a long loop which takes a bit of time to execute until rcx value is 0x1b1. This is likely to slowdown analysis or in general to delay execution of malicious code.



PEB reference is fetched and as in part one we enter the code section where it is parsed to fetch export table. Like part one, it goes through the InMemoryOrderModuleList and compares the result to some chars to get KERNEL32.dll or NTDLL.dll based on what API it needs. So what we expect is the following:

- loops through all the functions
- Calculates a hash for each one

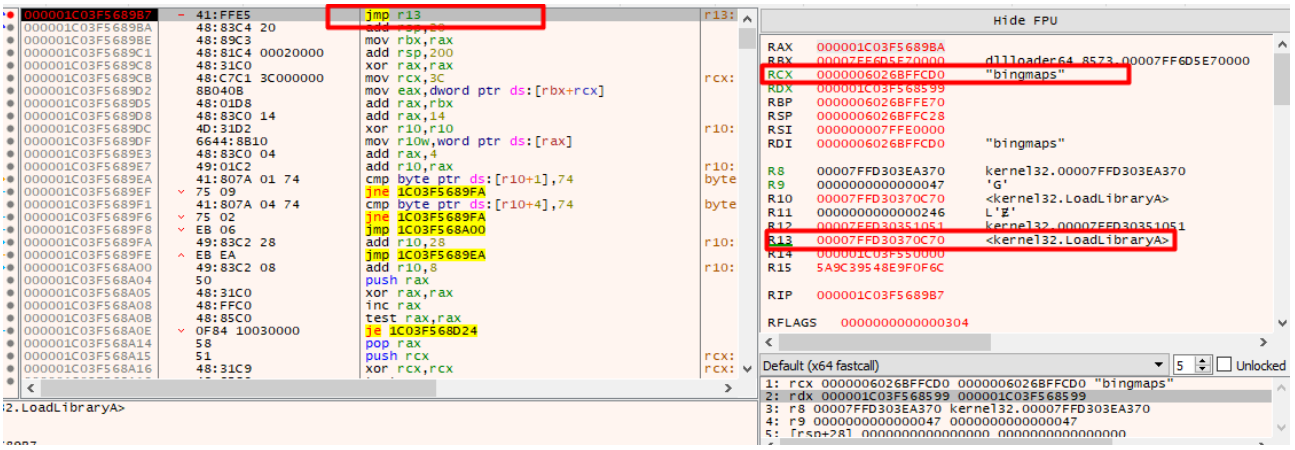
- Find LoadLibraryA()
- JMP to LoadLibraryA()with "bingmaps.dll" as argument
- Fetching NtProtectVirtualMemory
- Call NtProtectVirtualMemory on bingmaps.dll location with 0x40 rx -> wrx
- Overwrites code of the exported function bingmaps.dll (GetBingsMapFactory). It reads from the shellcode and writes to this location, essentially injecting a part of itself into the legitimate binary. The bytes are xored before writing.



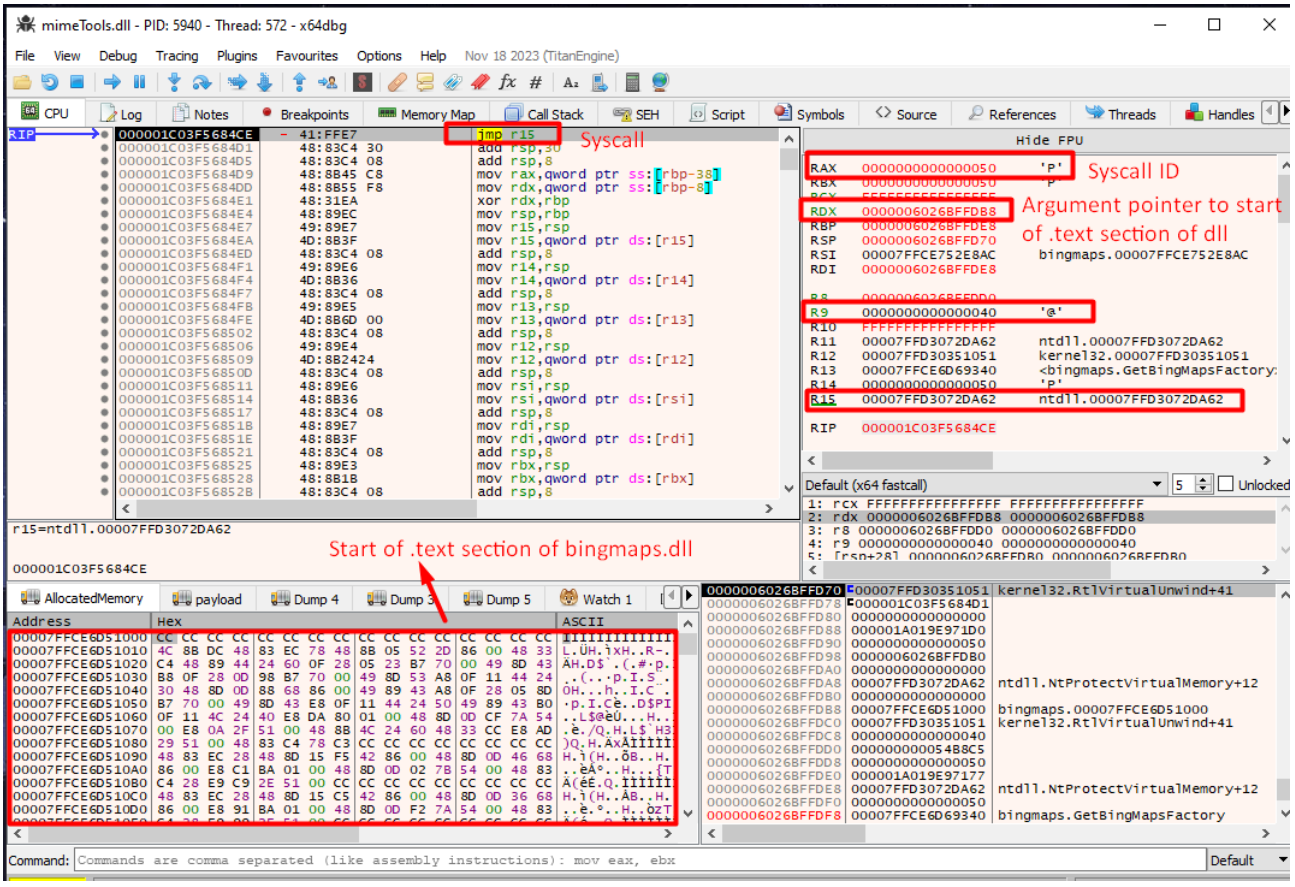
Hashing loop:

```
cmp byte ptr ds:[rsi+rcx],0
je 1C03F568E56
xor r13,r13
mov r13b,byte ptr ds:[rsi+rcx]
xor r15,r15
mov r15,qword ptr ss:[rbp-10]
shl r15,6
xor r14,r14
mov r14,qword ptr ss:[rbp-10]
shl r14,10
xor r12,r12
mov r12,qword ptr ss:[rbp-10]
xor rax,rax
mov rax,r13
add rax,r15
add rax,r14
sub rax,r12
mov qword ptr ss:[rbp-10],rax
inc rcx
jmp 1C03F568E14
```

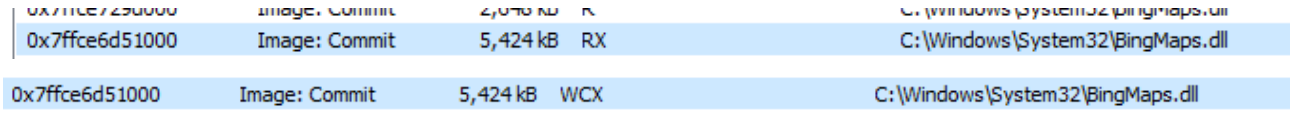
Call to LoadLibraryA:



First call to NtProtectVirtualMemory:



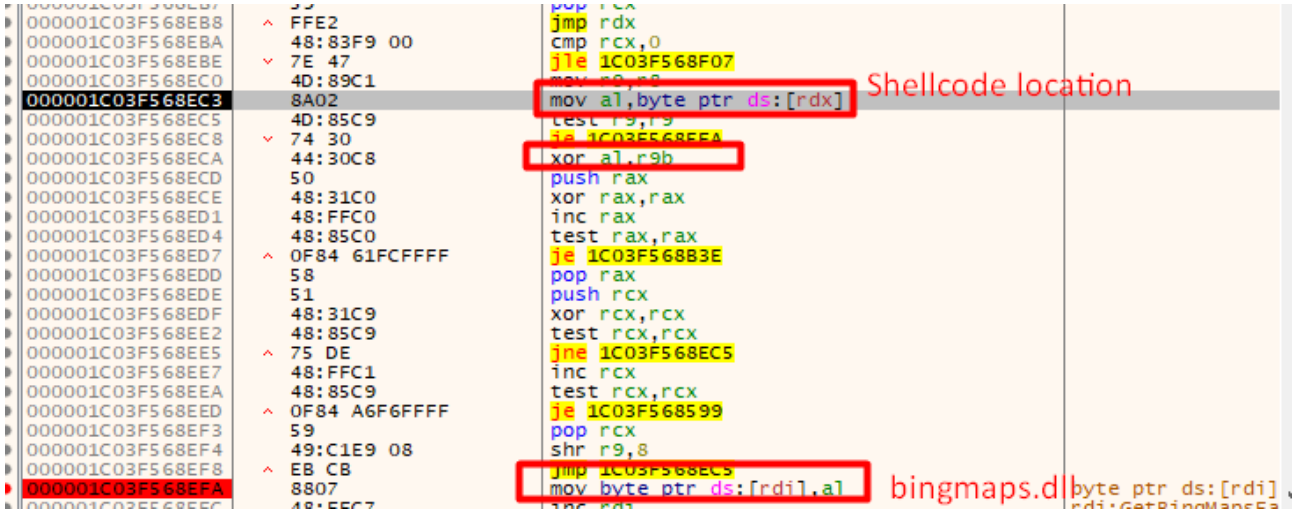
Memory protection alteration:

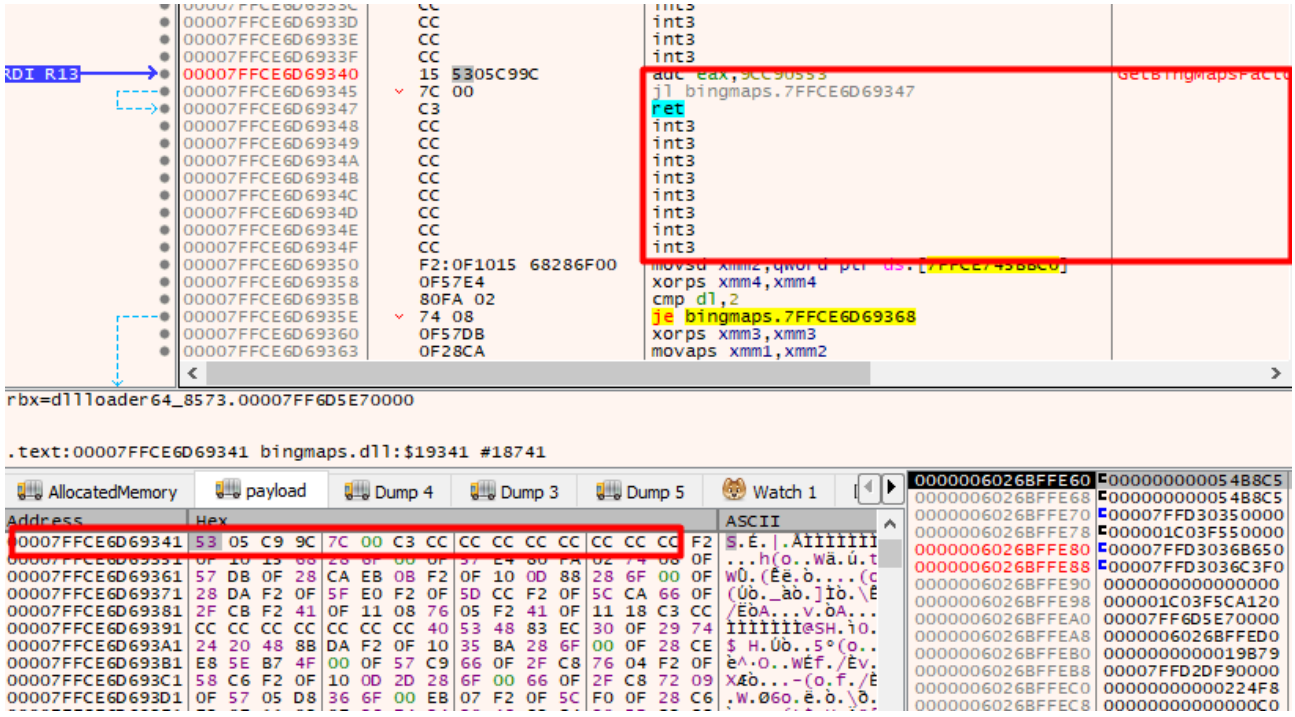


Shellcode DLL injection

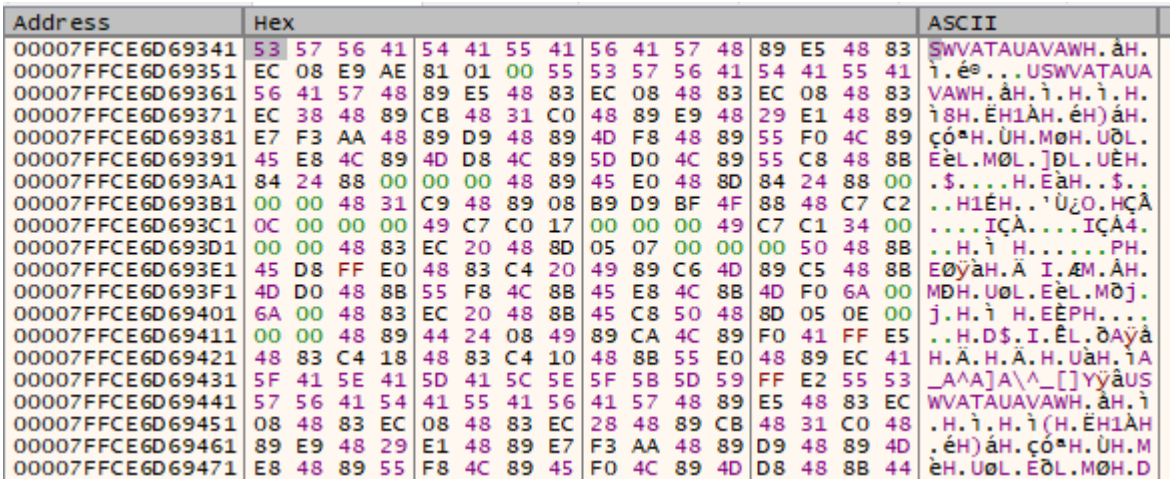
Next, malware injects part of its shellcode into the exported function of bingmaps.dll GetBingsMapFactory. Once it has finished injecting code it will call once again NtProtectVirtualMemory with 0x20

PAGE_EXECUTE_READ on the .text section of bingmaps.dll. Reverting it back to RX permissions.

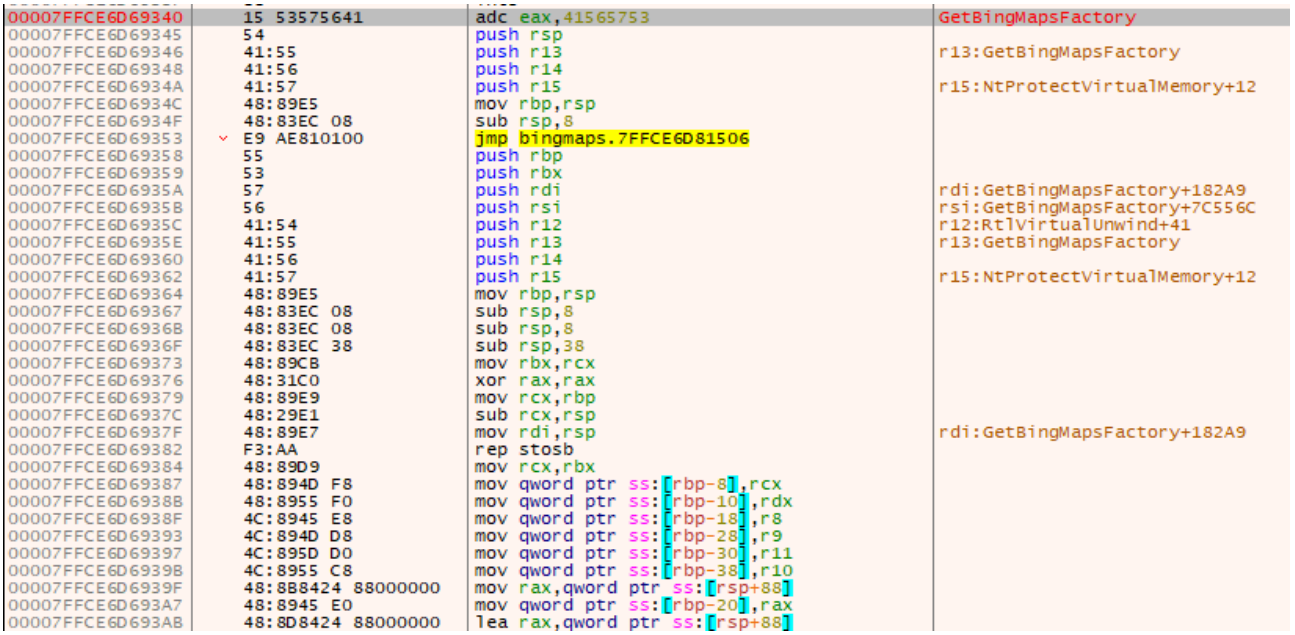




Memory and instructions seen after inject is complete:



Address	Hex	ASCII
00007FFCE6D69341	53 57 56 41 54 41 55 41 56 41 57 48 89 E5 48 83	SWVATAUAVAWH.åH.
00007FFCE6D69351	EC 08 E9 AE 81 01 00 55 53 57 56 41 54 41 55 41	ì.é®...USWVATAUA
00007FFCE6D69361	56 41 57 48 89 E5 48 83 EC 08 48 83 EC 08 48 83	VAWH.åH.ì.H.ì.H.
00007FFCE6D69371	EC 38 48 89 CB 48 31 C0 48 89 E9 48 29 E1 48 89	ì8H.ËH1ÀH.éH)áH.
00007FFCE6D69381	E7 F3 AA 48 89 D9 48 89 4D F8 48 89 55 F0 4C 89	çóªH.ÙH.MøH.UðL.
00007FFCE6D69391	45 E8 4C 89 4D D8 4C 89 5D D0 4C 89 55 C8 48 8B	EèL.MØL.]ÐL.UÈH.
00007FFCE6D693A1	84 24 88 00 00 00 48 89 45 E0 48 8D 84 24 88 00	.$....H.EàH..$..
00007FFCE6D693B1	00 00 48 31 C9 48 89 08 B9 D9 BF 4F 88 48 C7 C2	..H1ÉH..'Ù¿O.HÇÂ
00007FFCE6D693C1	0C 00 00 00 49 C7 C0 17 00 00 00 49 C7 C1 34 00	....IÇÀ....IÇÁ4.
00007FFCE6D693D1	00 00 48 83 EC 20 48 8D 05 07 00 00 00 50 48 8B	..H.ì H......PH.
00007FFCE6D693E1	45 D8 FF E0 48 83 C4 20 49 89 C6 4D 89 C5 48 8B	EØÿàH.Ä I.ÆM.ÅH.
00007FFCE6D693F1	4D D0 48 8B 55 F8 4C 8B 45 E8 4C 8B 4D F0 6A 00	MÐH.UøL.EèL.Mðj.
00007FFCE6D69401	6A 00 48 83 EC 20 48 8B 45 C8 50 48 8D 05 0E 00	j.H.ì H.EÈPH....
00007FFCE6D69411	00 00 48 89 44 24 08 49 89 CA 4C 89 F0 41 FF E5	..H.D$.I.ÊL.ðAÿå
00007FFCE6D69421	48 83 C4 18 48 83 C4 10 48 8B 55 E0 48 89 EC 41	H.Ä.H.Ä.H.UàH.ìA
00007FFCE6D69431	5F 41 5E 41 5D 41 5C 5E 5F 5B 5D 59 FF E2 55 53	_A^A]A\^_[]YÿâUS
00007FFCE6D69441	57 56 41 54 41 55 41 56 41 57 48 89 E5 48 83 EC	WVATAUAVAWH.åH.ì
00007FFCE6D69451	08 48 83 EC 08 48 83 EC 28 48 89 CB 48 31 C0 48	.H.ì.H.ì(H.ËH1ÀH
00007FFCE6D69461	89 E9 48 29 E1 48 89 E7 F3 AA 48 89 D9 48 89 4D	.éH)áH.çóªH.ÙH.M
00007FFCE6D69471	E8 48 89 55 F8 4C 89 45 F0 4C 89 4D D8 48 8B 44	èH.UøL.EðL.MØH.D

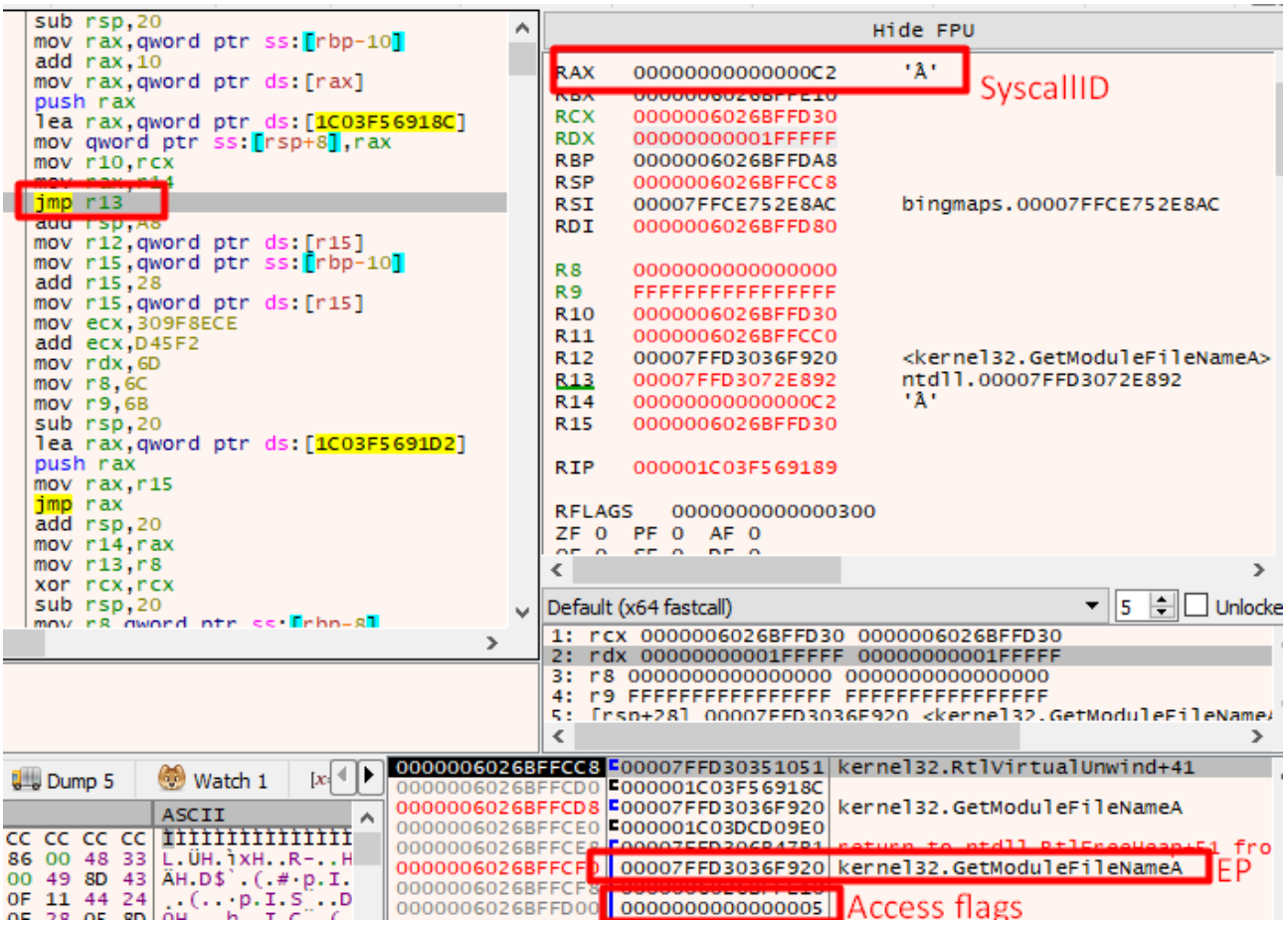


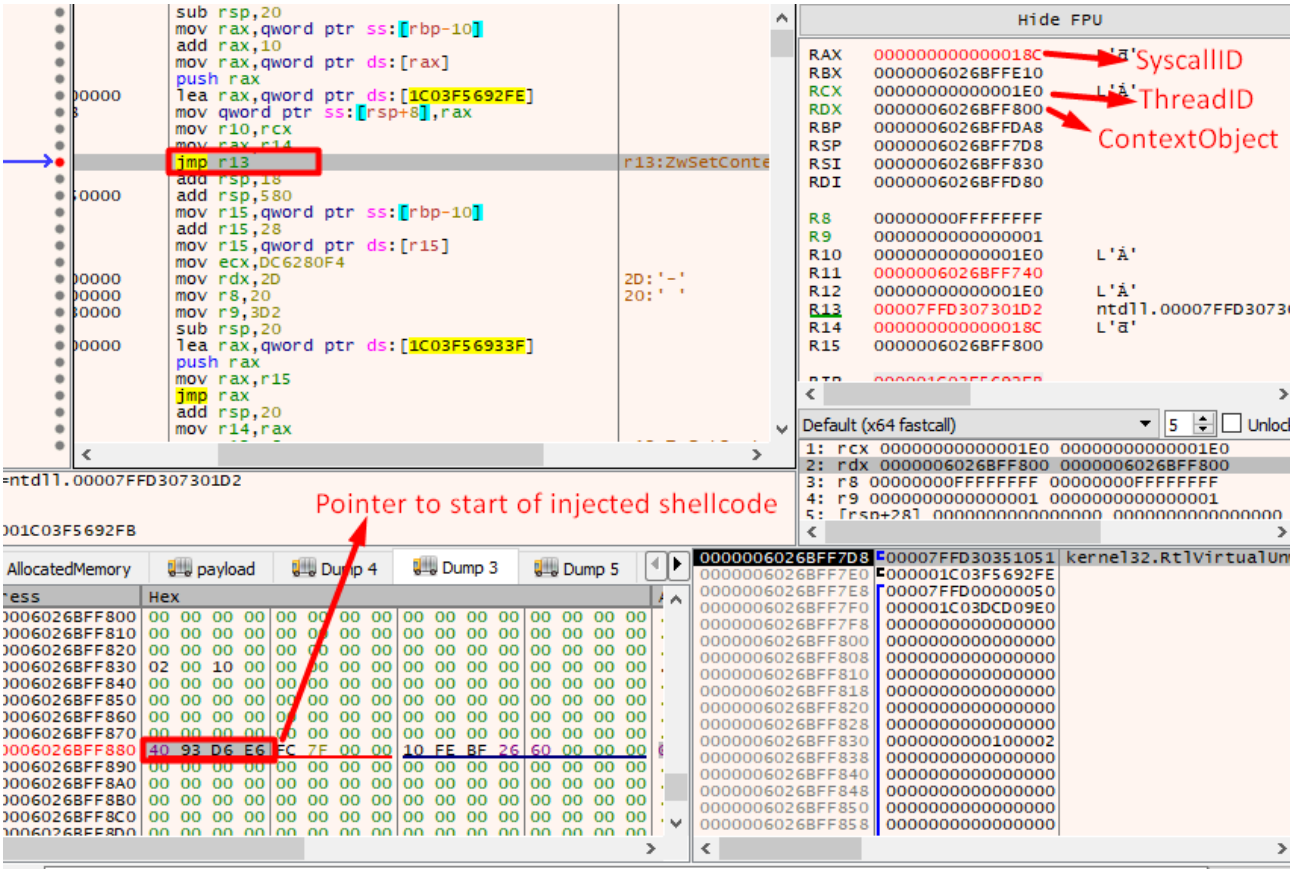
```
00007FFCE6D69340    15 53575641         adc eax,41565753                GetBingMapsFactory
00007FFCE6D69345    54                  push rsp
00007FFCE6D69346    41:55               push r13                        r13:GetBingMapsFactory
00007FFCE6D69348    41:56               push r14
00007FFCE6D6934A    41:57               push r15                        r15:NtProtectVirtualMemory+12
00007FFCE6D6934C    48:89E5             mov rbp,rsp
00007FFCE6D6934F    48:83EC 08          sub rsp,8
00007FFCE6D69353    E9 AE810100         jmp bingmaps.7FFCE6D81506
00007FFCE6D69358    55                  push rbp
00007FFCE6D69359    53                  push rbx
00007FFCE6D6935A    57                  push rdi                        rdi:GetBingMapsFactory+182A9
00007FFCE6D6935B    56                  push rsi                        rsi:GetBingMapsFactory+7C556C
00007FFCE6D6935C    41:54               push r12                        r12:RtlVirtualUnwind+41
00007FFCE6D6935E    41:55               push r13                        r13:GetBingMapsFactory
00007FFCE6D69360    41:56               push r14
00007FFCE6D69362    41:57               push r15                        r15:NtProtectVirtualMemory+12
00007FFCE6D69364    48:89E5             mov rbp,rsp
00007FFCE6D69367    48:83EC 08          sub rsp,8
00007FFCE6D6936B    48:83EC 08          sub rsp,8
00007FFCE6D6936F    48:83EC 38          sub rsp,38
00007FFCE6D69373    48:89CB             mov rbx,rcx
00007FFCE6D69376    48:31C0             xor rax,rax
00007FFCE6D69379    48:89E9             mov rcx,rbp
00007FFCE6D6937C    48:29E1             sub rcx,rsp
00007FFCE6D6937F    48:89E7             mov rdi,rsp                     rdi:GetBingMapsFactory+182A9
00007FFCE6D69382    F3:AA               rep stosb
00007FFCE6D69384    48:89D9             mov rcx,rbx
00007FFCE6D69387    48:894D F8          mov qword ptr ss:[rbp-8],rcx
00007FFCE6D6938B    48:8955 F0          mov qword ptr ss:[rbp-10],rdx
00007FFCE6D6938F    4C:8945 E8          mov qword ptr ss:[rbp-18],r8
00007FFCE6D69393    4C:894D D8          mov qword ptr ss:[rbp-28],r9
00007FFCE6D69397    4C:895D D0          mov qword ptr ss:[rbp-30],r11
00007FFCE6D6939B    4C:8955 C8          mov qword ptr ss:[rbp-38],r10
00007FFCE6D6939F    48:8B8424 88000000  mov rax,qword ptr ss:[rsp+88]
00007FFCE6D693A7    48:8945 E0          mov qword ptr ss:[rbp-20],rax
00007FFCE6D693AB    48:8D8424 88000000  lea rax,qword ptr ss:[rsp+88]
```

New thread

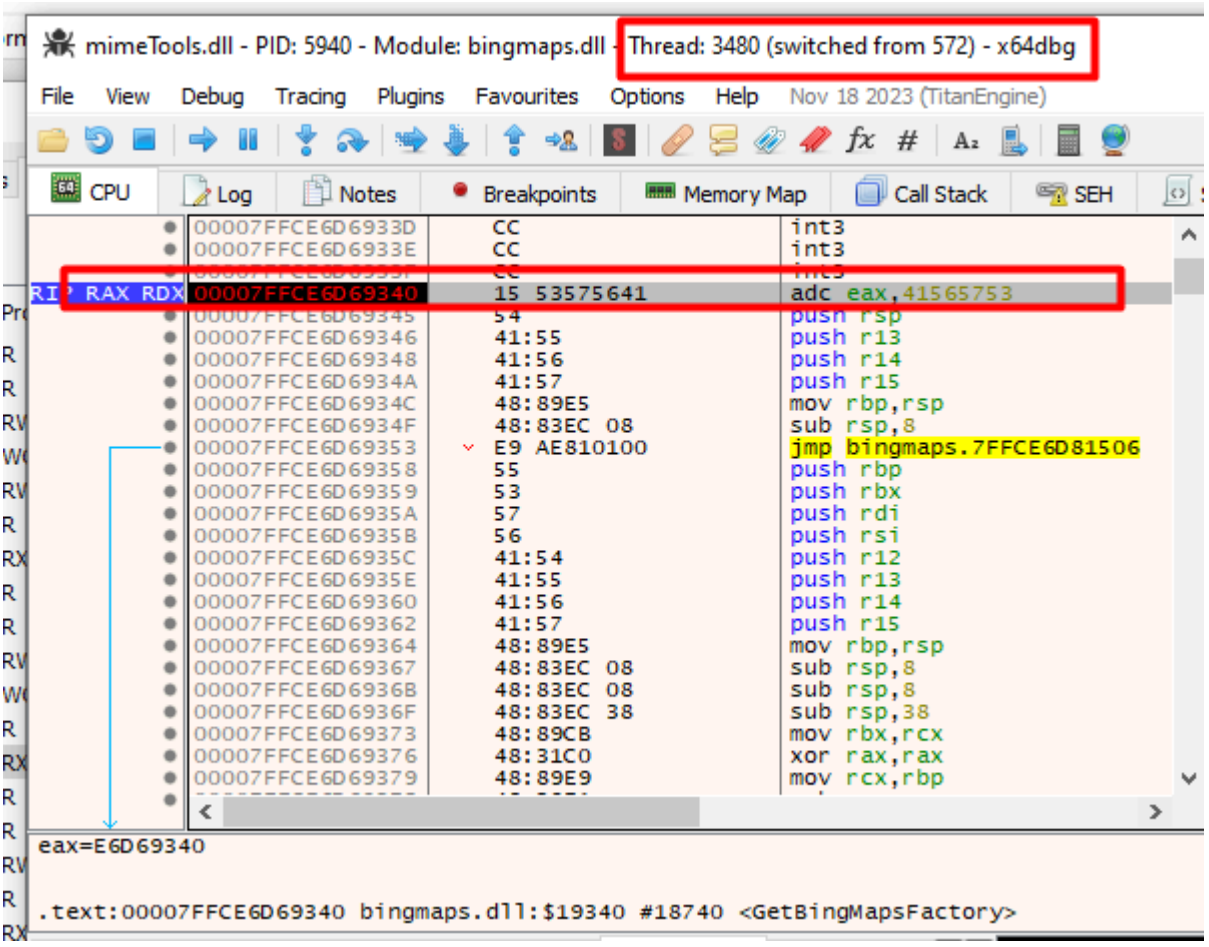
Malware will once again run through PEB , Ntdll.dll & Kernel32.dll in search for the following API:

- GetModuleFileNameA -> This will be the entry point of the new thead, but RIP will be the injected code seen before
- NtCreateThreadEx -> creates thread in suspended state and hidden from debugger. Value 0x5 we can change it to 0x1 if we want to see the thread in the debugger.
- NtGetContextThread
- NtSetContextThread -> Inside the context object in memory there is pointer referencing the start of injected code which will be the threads RIP. (BP will be set here to follow execution)
- NtResumeThread
- NtWaitForSingleObject





Running execution we finally reach the new thread and code!



Anti Debug

From the bingmaps.dll injected code, the malware proceeds with an initial anti debug trick. It seems it has a hash table hardcoded and checks running processes if they match, if so it proceeds to exit. For this, from the PEB it fetches the following API:

- CreateToolHelp32SnapShot
- Process32First
- Process32Next

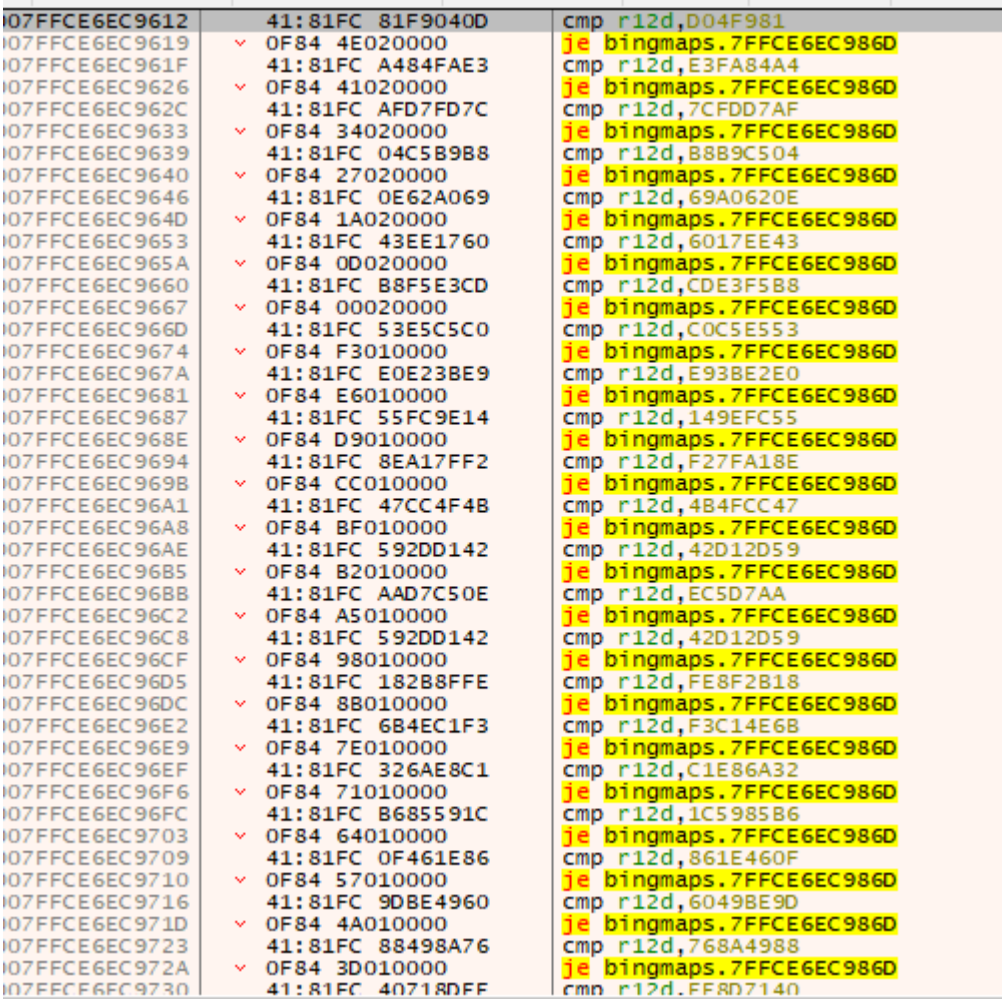
Each name from CreateToolHelp32SnapShot section, is hashed with the same algorithm identified above and then compared with the hashtable results. I haven't checked all the hashes but for it seems it quit when seeing x64dbg.exe and pe-bear.exe which I had running.

Example hash of Pe-Bear -> C0A4EC617E002F0 -> check performed on 17E002F0 portion.

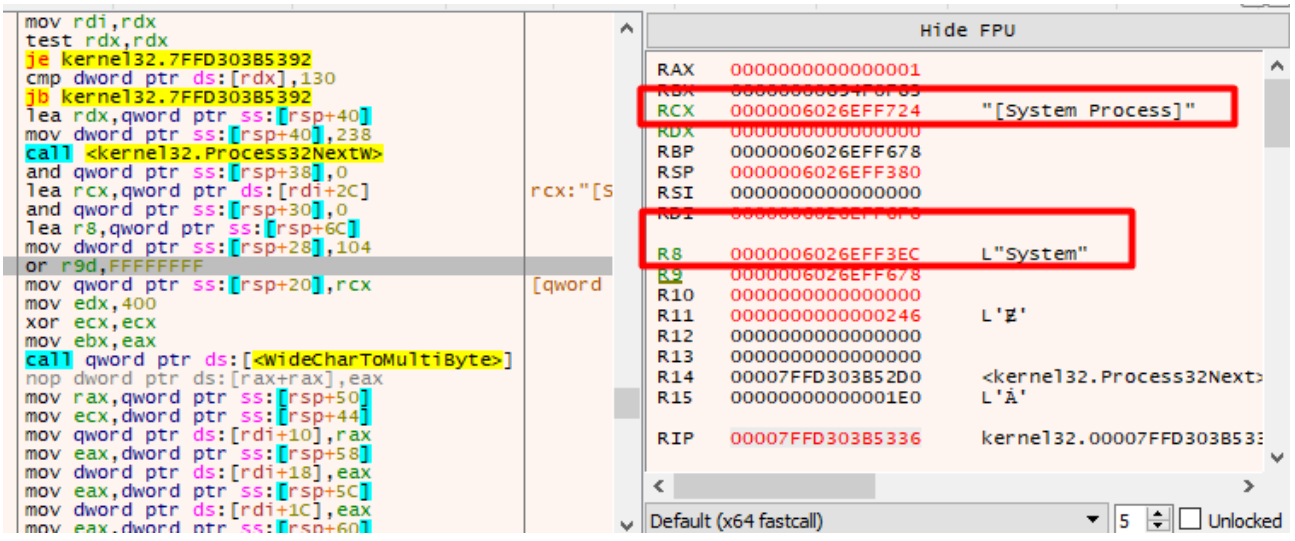
To avoid this, from Process Hacker I opened the section created by CreateToolHelp32SnapShot and modified the hex values to match svchost.exe, and it worked, we get to the next stage.

Also, worth noting that there is a specific check for explorer.exe, when it matches it runs another section of code that saves the PID of the process. This is needed for the next step.

Hashtable:



Call to Process32Next:



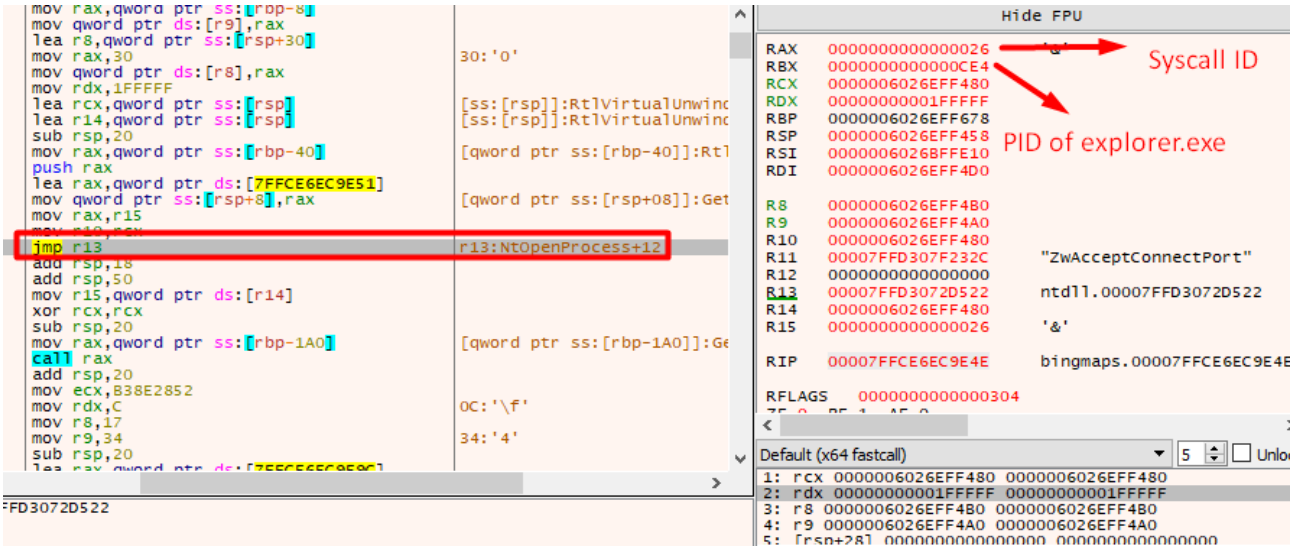
Injecting 3rd shellcode into explorer.exe

Malware from the context of the bingmaps.dll will now proceed to inject other shellcode to explorer.exe. To do this the following is done:

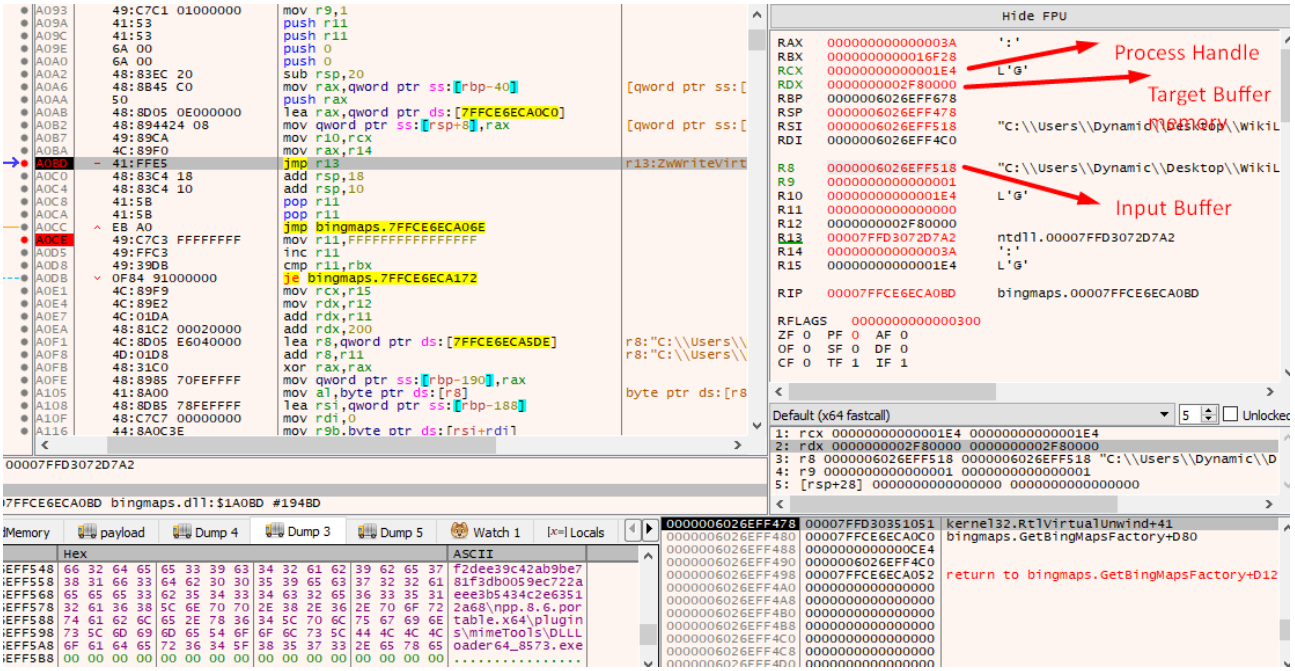
- ZwOpenProcess -> supplying the PID of explorer.exe
- ZwAllocateVirtualMemory -> Allocates two memory buffers inside explorer.exe.
- GetModuleFileA -> Get the full path name of the current process running all the malware
- ZwWriteVirtualMemory -> Writes the shellcoded and the results of GetModuleFileA to explorer.exe section.

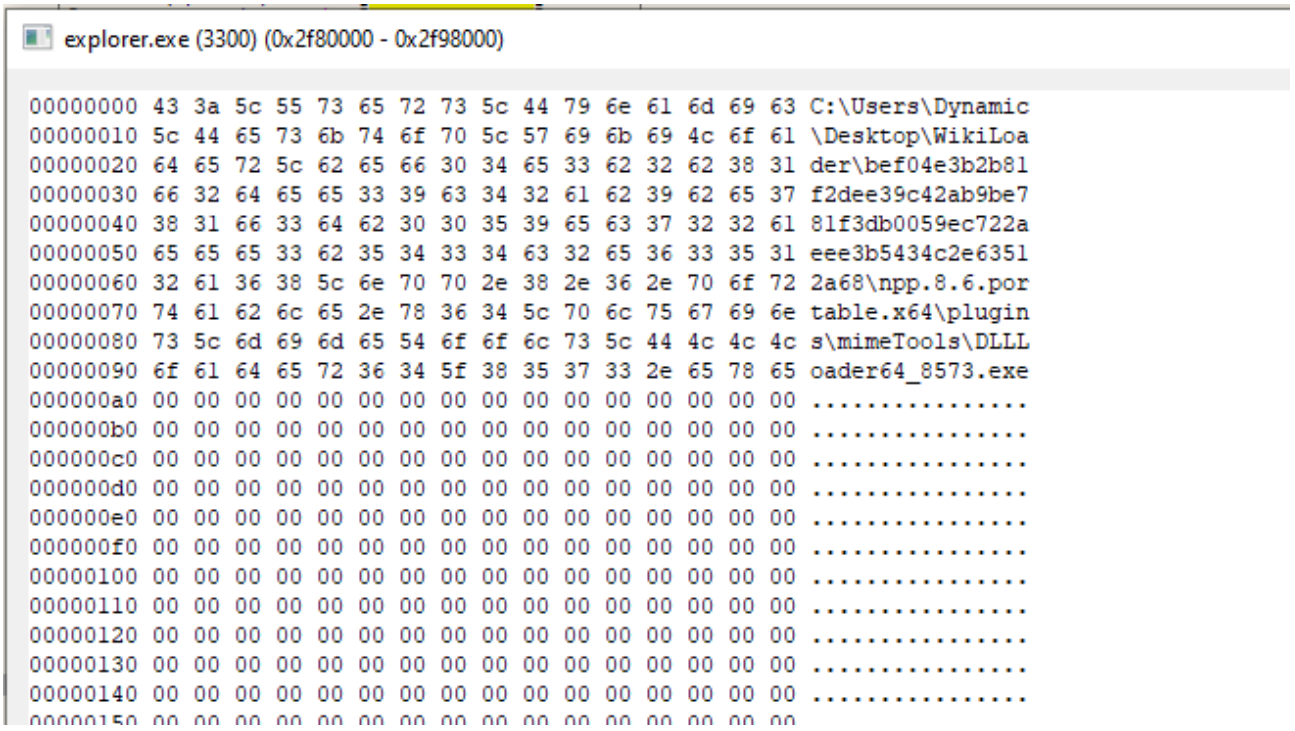
For the shellcode, a single byte is written at a time. The byte to write is obtained by a single XOR loop before the call, the result is loaded into R8 which is the 3rd argument for ZwWriteVirtualMemory.

Call to open process:

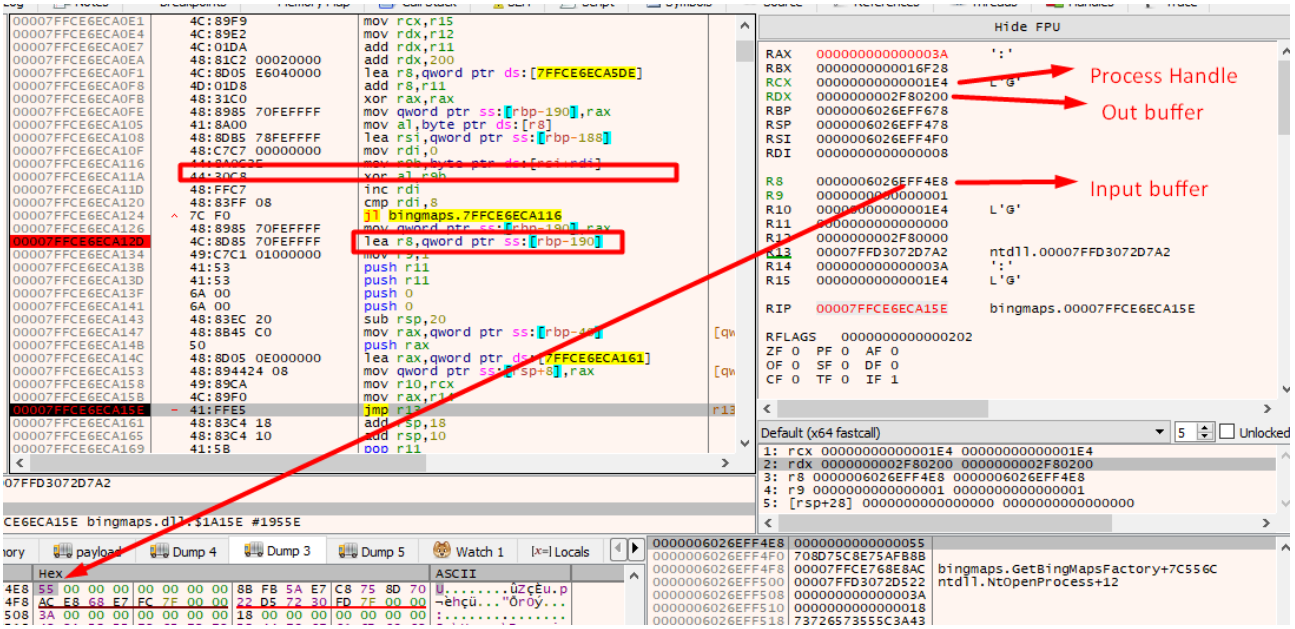


Call to ZwWriteVirtualMemory for current process name:

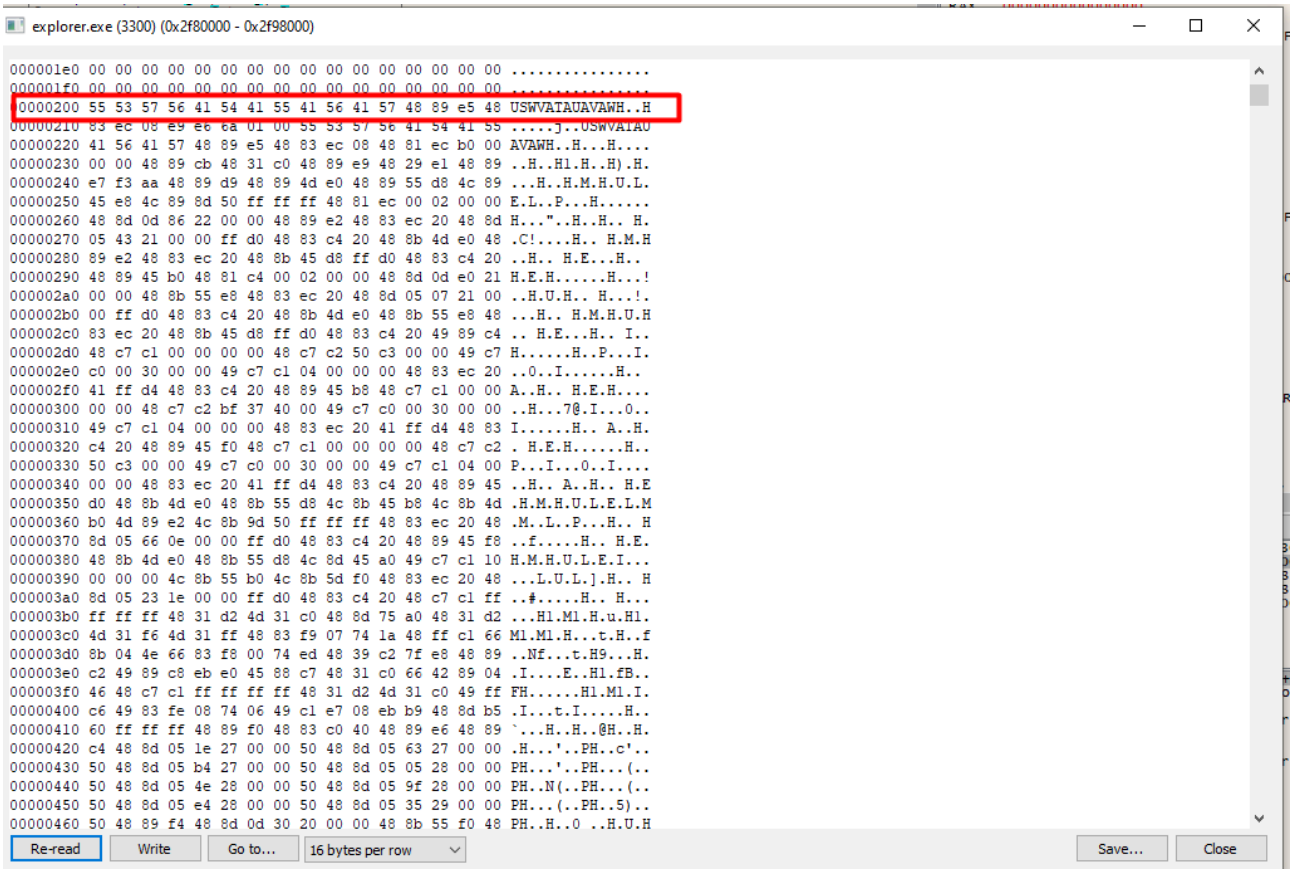




Call to ZwWriteVirtualMemory for shellcode:



Memory inside explorer.exe after loop write:



End of part 2

So we have found another shellcode inject and the code seems to proceed with some other anti analysis checks as I saw it loads:

- ZwQueryInformationProcess
- RtlWow64GetCpuAreaInfo

But I will look into this next time with part 3!

References

- https://bazaar.abuse.ch/sample/bef04e3b2b81f2dee39c42ab9be781f3db0059ec722aeee3b5434c2e63512a68/
- https://www.unpac.me/results/612d6d2c-c45d-47ba-a2bb-a218ec753d3f
- https://twitter.com/Cryptolaemus1/status/1747394506331160736
- https://www.proofpoint.com/us/blog/threat-insight/out-sandbox-wikiloader-digs-sophisticated-evasion
- https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/api/pebteb/peb/index.htm
- https://mohamed-fakroud.gitbook.io/red-teamings-dojo/shellcoding/leveraging-from-pe-parsing-technique-to-write-x86-shellcode

---