

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 Microsoft 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.

```
000001C03F568390 8A56 0B mov dl,byte ptr ds:[rsi+B]
000001C03F568393 48:81F9 B1010000 cmp rcx,1B1
000001C03F56839A 7D 0F jge 1C03F5683A8
000001C03F56839C 8A46 0B mov al,byte ptr ds:[rsi+B]
000001C03F56839F 48:39D0 cmp rax,rdx
000001C03F5683A2 74 05 je 1C03F5683A9
000001C03F5683A4 88C2 mov dl,al
000001C03F5683A6 48:FFC1 inc rcx
000001C03F5683A9 EB E8 jmp 1C03F568393
000001C03F5683AB 48:85E4 test rsp,rsip
```

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.

000001C03F568645	65 48: 8B 18	mov rbx,qword ptr ds:[rax]
000001C03F568649	48: C7 C0 18 00 00 00	mov rax,18
000001C03F568650	48: 8B 1C 03	mov rbx,qword ptr ds:[rbx+rax]
000001C03F568654	48: C7 C0 20 00 00 00	mov rax,20
000001C03F56865B	48: 8B 1C 03	mov rbx,qword ptr ds:[rbx+rax]
000001C03F56865F	49: 89 DC	mov r12,rbx
000001C03F568662	51	push rcx
000001C03F568663	48: C7 C1 40 00 00 00	mov rcx,40
000001C03F56866A	48: 8B 04 0B	mov rax,qword ptr ds:[rbx+rcx]
000001C03F56866E	59	pop rcx
000001C03F56866F	51	push rcx
000001C03F568670	48: C7 C1 28 00 00 00	mov rcx,28
000001C03F568677	44: 8A 34 08	mov r14b,byte ptr ds:[rax+rcx]
000001C03F56867B	59	pop rcx
000001C03F56867C	41: 80 C6 03	add r14b,3
000001C03F568680	41: 80 FE 4E	cmp r14b,4E
000001C03F568684	74 1C	je 1C03F5686A2
000001C03F568686	51	push rcx
000001C03F568687	48: C7 C1 28 00 00 00	mov rcx,28
000001C03F56868E	44: 8A 34 08	mov r14b,byte ptr ds:[rax+rcx]
000001C03F568692	59	pop rcx
000001C03F568693	41: 80 C6 04	add r14b,4
000001C03F568697	41: 80 FE 6F	cmp r14b,6F
000001C03F56869B	74 05	je 1C03F5686A2
000001C03F56869D	E9 F6 00 00 00	jmp 1C03F568798
000001C03F5686A2	51	push rcx
000001C03F5686A3	48: C7 C1 30 00 00 00	mov rcx,30
000001C03F5686AA	44: 8A 34 08	mov r14b,byte ptr ds:[rax+rcx]
000001C03F5686AE	59	pop rcx

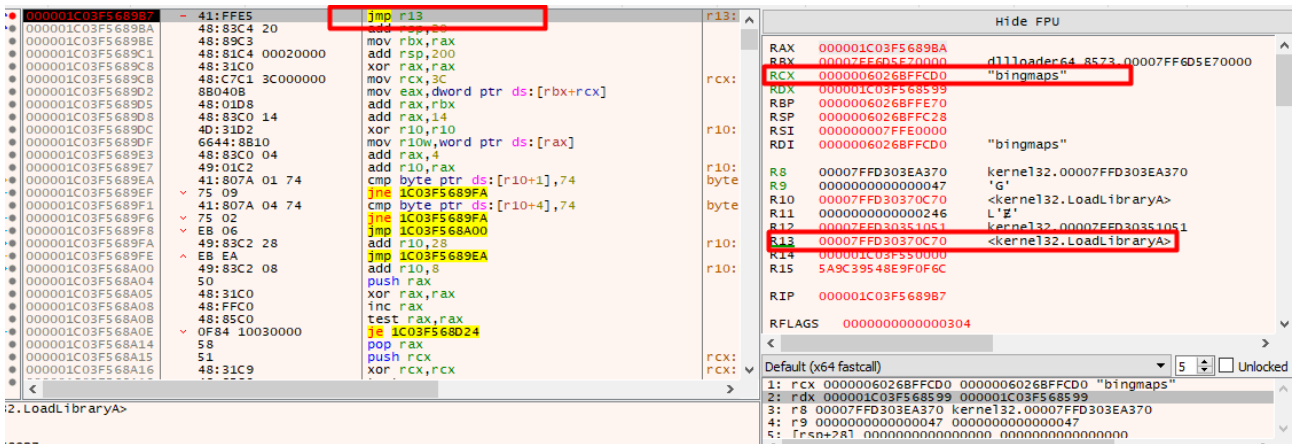
Hashing loop:

```

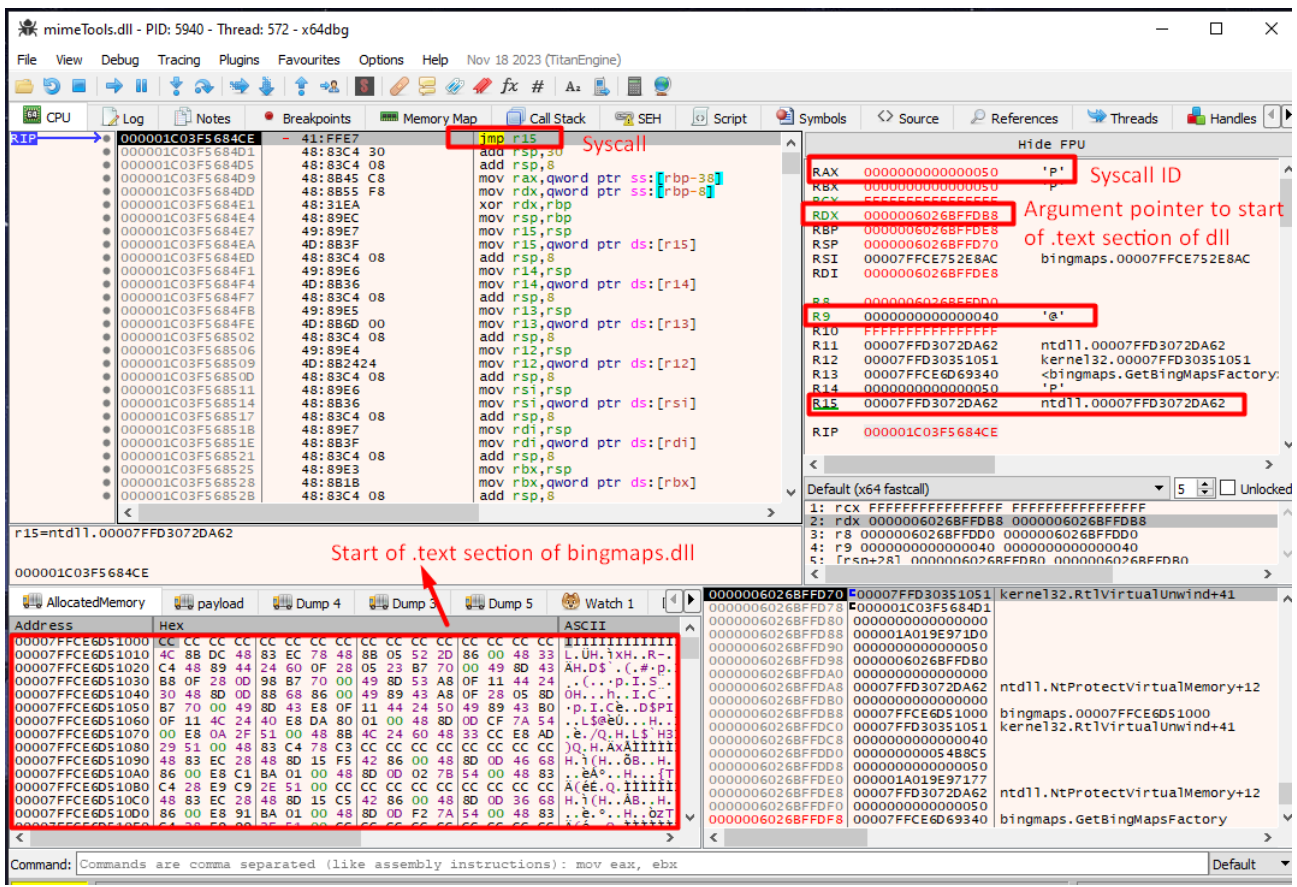
cmp byte ptr ds:[rsi+rcx],0
je 1C03F568E56
xor r13,r13
mov r13b,byte ptr ds:[rsi+rcx]
xor r15,r15
mov r15,qword ptr ss:[rbp-10]
shl r15,6
xor r14,r14
mov r14,qword ptr ss:[rbp-10]
shl r14,10
xor r12,r12
mov r12,qword ptr ss:[rbp-10]
xor rax,rax
mov rax,r13
add rax,r15
add rax,r14
sub rax,r12
mov qword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

```

Call to LoadLibraryA:



First call to NtProtectVirtualMemory:



Memory protection alteration:

0x7f1ce7290000	Image: Commit	2,076 kB	RX	C:\Windows\System32\bingmaps.dll
0x7f1ce6d51000	Image: Commit	5,424 kB	RX	C:\Windows\System32\BingMaps.dll
0x7f1ce6d51000	Image: Commit	5,424 kB	WCX	C:\Windows\System32\BingMaps.dll

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.

```

000001C03F568EB7  ^  FFE2          jmp rdx
000001C03F568EB8  48:83F9 00    cmp rcx,0
000001C03F568EBA  v  7E 47       jle 1C03F568F07
000001C03F568EBE  ^  4D:89C1     mov rcx,rcx
000001C03F568EC0  ^  8A02       mov al,byte ptr ds:[rdx]
000001C03F568EC3  4D:85C9     test r9,r9
000001C03F568EC5  v  74 30       je 1C03F568EE4
000001C03F568ECA  ^  44:30C8     xor al,r9b
000001C03F568ECD  50         push rax
000001C03F568ECE  48:31C0     xor rax,rax
000001C03F568ED1  48:FFC0     inc rax
000001C03F568ED4  48:85C0     test rax,rax
000001C03F568ED7  ^  0F84 61FCFFF  je 1C03F56883E
000001C03F568EDD  58         pop rax
000001C03F568EDE  51         push rcx
000001C03F568EDF  48:31C9     xor rcx,rcx
000001C03F568EE2  48:85C9     test rcx,rcx
000001C03F568EE5  ^  75 DE       jne 1C03F568EC5
000001C03F568EE7  48:FFC1     inc rcx
000001C03F568EEA  48:85C9     test rcx,rcx
000001C03F568EED  ^  0F84 A6F6FFF  je 1C03F568599
000001C03F568EF3  59         pop rcx
000001C03F568EF4  49:C1E9 08   shr r9,8
000001C03F568EF8  ^  EB CB       jmp 1C03F568EC5
000001C03F568EFA  v  8B 80       mov byte ptr ds:[rdi],al
    
```

Shellcode location

bingmaps.dll

```

00007FFCE6D6933C  CC          int3
00007FFCE6D6933D  CC          int3
00007FFCE6D6933E  CC          int3
00007FFCE6D6933F  CC          int3
00007FFCE6D69340  15 5305C99C  adc eax,9CC90555
00007FFCE6D69345  v  7C 00       jl bingmaps.7FFCE6D69347
00007FFCE6D69347  C3         ret
00007FFCE6D69348  CC          int3
00007FFCE6D69349  CC          int3
00007FFCE6D6934A  CC          int3
00007FFCE6D6934B  CC          int3
00007FFCE6D6934C  CC          int3
00007FFCE6D6934D  CC          int3
00007FFCE6D6934E  CC          int3
00007FFCE6D6934F  CC          int3
00007FFCE6D69350  F2:0F1015 68286F00  movsd xmm2,qword ptr ds:[7FFCE74588C0]
00007FFCE6D69358  0F57E4     xorps xmm4,xmm4
00007FFCE6D6935B  80FA 02     cmp dl,2
00007FFCE6D6935E  v  74 08       je bingmaps.7FFCE6D69368
00007FFCE6D69360  0F57DB     xorps xmm3,xmm3
00007FFCE6D69363  0F28CA     movaps xmm1,xmm2
    
```

getbingmapsfact

rbx=dllloader64_8573.00007FF6D5E70000

.text:00007FFCE6D69341 bingmaps.dll:\$19341 #18741

Address	Hex	ASCII
00007FFCE6D69341	53 05 C9 9C 7C 00 C3 CC CC CC CC CC CC CC F2	S.É. .AIIIIII
00007FFCE6D69351	0F 10 15 68 28 0F 00 0F 37 E4 80 FA 02 74 08 0F	...h(o..wä.ú.t
00007FFCE6D69361	57 DB 0F 28 CA EB 08 F2 0F 10 0D 88 28 6F 00 0F	WÜ.(Ëë.ö...c
00007FFCE6D69371	28 DA F2 0F 5F E0 F2 0F 5D CC F2 0F 5C CA 66 0F	(Üö..äb.ÿö.\ë
00007FFCE6D69381	2F CB F2 41 0F 11 08 76 05 F2 41 0F 11 18 C3 CC	/ËöA...v.öA...
00007FFCE6D69391	CC CC CC CC CC CC CC 40 53 48 83 EC 30 0F 29 74	iiiiii@SH.io.
00007FFCE6D693A1	24 20 48 8B DA F2 0F 10 35 BA 28 6F 00 0F 28 CE	\$ H.Üö..5*(o..
00007FFCE6D693B1	E8 5E 87 4F 00 0F 57 C9 66 0F 2F C8 76 04 F2 0F	e^..ö...wEF./Ev.
00007FFCE6D693C1	58 C6 F2 0F 10 0D 2D 28 6F 00 66 0F 2F C8 72 09	xÄö...-(o..f./Ë
00007FFCE6D693D1	0F 57 05 D8 36 6F 00 EB 07 F2 0F 5C F0 0F 28 C6	.W.ö6ö..ë.ö.\ö.

Memory and instructions seen after inject is complete:

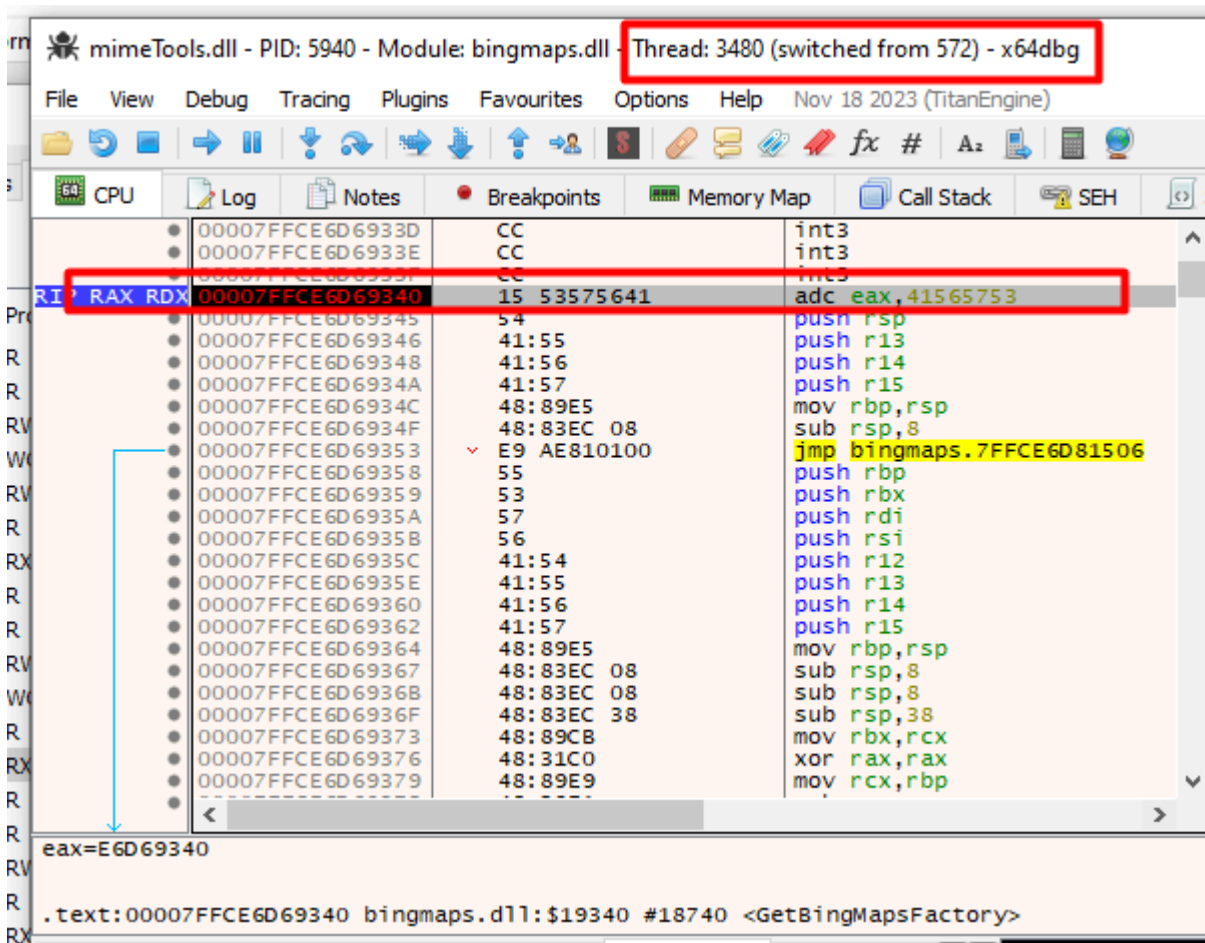
Address	Hex	ASCII
00007FFCE6D69341	53 57 56 41 54 41 55 41 56 41 57 48 89 E5 48 83	WVATAUAVAWH.àH.
00007FFCE6D69351	EC 08 E9 AE 81 01 00 55 53 57 56 41 54 41 55 41	ì.è...USWVATAUA
00007FFCE6D69361	56 41 57 48 89 E5 48 83 EC 08 48 83 EC 08 48 83	VAWH.àH.ì.H.ì.H.
00007FFCE6D69371	EC 38 48 89 CB 48 31 C0 48 89 E9 48 29 E1 48 89	ìSH.ÈH1ÀH.èH)àH.
00007FFCE6D69381	E7 F3 AA 48 89 D9 48 89 4D F8 48 89 55 F0 4C 89	çó^H.ÙH.MøH.UøL.
00007FFCE6D69391	45 E8 4C 89 4D D8 4C 89 5D D0 4C 89 55 C8 48 88	ÈèL.MøL.ÏDL.UÈH.
00007FFCE6D693A1	84 24 88 00 00 00 48 89 45 E0 48 8D 84 24 88 00	.\$...H.ÈàH.\$..
00007FFCE6D693B1	00 00 48 31 C9 48 89 08 B9 D9 BF 4F 88 48 C7 C2	..HIÈH..'UçO.HçÀ
00007FFCE6D693C1	0C 00 00 00 49 C7 C0 17 00 00 00 49 C7 C1 34 00	...IçÀ....IçÀ4.
00007FFCE6D693D1	00 00 48 83 EC 20 48 8D 05 07 00 00 00 50 48 88	..H.ì H.....PH.
00007FFCE6D693E1	45 D8 FF E0 48 83 C4 20 49 89 C6 4D 89 C5 48 88	EöYàH.A I.ÀM.ÀH.
00007FFCE6D693F1	4D D0 48 88 55 F8 4C 88 45 E8 4C 88 4D F0 6A 00	MøH.UøL.ÈèL.Møj.
00007FFCE6D69401	6A 00 48 83 EC 20 48 88 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.ÈL.ðAYà
00007FFCE6D69421	48 83 C4 18 48 83 C4 10 48 8B 55 E0 48 89 EC 41	H.À.H.À.H.UàH.ìA
00007FFCE6D69431	5F 41 5E 41 5D 41 5C 5E 5F 5B 5D 59 FF E2 55 53	_AAAJA\^_[Y]YÀUS
00007FFCE6D69441	57 56 41 54 41 55 41 56 41 57 48 89 E5 48 83 EC	WVATAUAVAWH.àH.ì
00007FFCE6D69451	08 48 83 EC 08 48 83 EC 28 48 89 CB 48 31 C0 48	.H.ì.H.ì(H.ÈH1ÀH
00007FFCE6D69461	89 E9 48 29 E1 48 89 E7 F3 AA 48 89 D9 48 89 4D	.èH)àH.çó^H.ÙH.M
00007FFCE6D69471	E8 48 89 55 F8 4C 89 45 F0 4C 89 4D D8 48 88 44	èH.UøL.ÈøL.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	
00007FFCE6D69368	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	
00007FFCE6D69388	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	
00007FFCE6D69398	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 thread, 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



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:

07FFCE6EC9612	41:81FC 81F9040D	cmp r12d,004F981
07FFCE6EC9619	0F84 4E020000	je bingmaps.7FFCE6EC986D
07FFCE6EC961F	41:81FC A484FAE3	cmp r12d,E3FA84A4
07FFCE6EC9626	0F84 41020000	je bingmaps.7FFCE6EC986D
07FFCE6EC962C	41:81FC AFD7FD7C	cmp r12d,7CFDD7AF
07FFCE6EC9633	0F84 34020000	je bingmaps.7FFCE6EC986D
07FFCE6EC9639	41:81FC 04C5B988	cmp r12d,B8B9C504
07FFCE6EC9640	0F84 27020000	je bingmaps.7FFCE6EC986D
07FFCE6EC9646	41:81FC 0E62A069	cmp r12d,69A0620E
07FFCE6EC964D	0F84 1A020000	je bingmaps.7FFCE6EC986D
07FFCE6EC9653	41:81FC 43EE1760	cmp r12d,6017EE43
07FFCE6EC965A	0F84 0D020000	je bingmaps.7FFCE6EC986D
07FFCE6EC9660	41:81FC B8F5E3CD	cmp r12d,CDE3F5B8
07FFCE6EC9667	0F84 00020000	je bingmaps.7FFCE6EC986D
07FFCE6EC966D	41:81FC 53E5C5C0	cmp r12d,C0C5E553
07FFCE6EC9674	0F84 F3010000	je bingmaps.7FFCE6EC986D
07FFCE6EC967A	41:81FC E0E23BE9	cmp r12d,E93BE2E0
07FFCE6EC9681	0F84 E6010000	je bingmaps.7FFCE6EC986D
07FFCE6EC9687	41:81FC 55FC9E14	cmp r12d,149EFC55
07FFCE6EC968E	0F84 D9010000	je bingmaps.7FFCE6EC986D
07FFCE6EC9694	41:81FC 8EA17FF2	cmp r12d,F27FA18E
07FFCE6EC969B	0F84 C0010000	je bingmaps.7FFCE6EC986D
07FFCE6EC96A1	41:81FC 47CC4F4B	cmp r12d,4B4FCC47
07FFCE6EC96A8	0F84 BF010000	je bingmaps.7FFCE6EC986D
07FFCE6EC96AE	41:81FC 592DD142	cmp r12d,42D12D59
07FFCE6EC96B5	0F84 B2010000	je bingmaps.7FFCE6EC986D
07FFCE6EC96BB	41:81FC AAD7C50E	cmp r12d,EC5D7AA
07FFCE6EC96C2	0F84 A5010000	je bingmaps.7FFCE6EC986D
07FFCE6EC96C8	41:81FC 592DD142	cmp r12d,42D12D59
07FFCE6EC96CF	0F84 98010000	je bingmaps.7FFCE6EC986D
07FFCE6EC96D5	41:81FC 1828FFE	cmp r12d,FE8F2B18
07FFCE6EC96DC	0F84 8B010000	je bingmaps.7FFCE6EC986D
07FFCE6EC96E2	41:81FC 6B4EC1F3	cmp r12d,F3C14E6B
07FFCE6EC96E9	0F84 7E010000	je bingmaps.7FFCE6EC986D
07FFCE6EC96EF	41:81FC 326AE8C1	cmp r12d,C1E86A32
07FFCE6EC96F6	0F84 71010000	je bingmaps.7FFCE6EC986D
07FFCE6EC96FC	41:81FC B685591C	cmp r12d,1C5985B6
07FFCE6EC9703	0F84 64010000	je bingmaps.7FFCE6EC986D
07FFCE6EC9709	41:81FC 0F461E86	cmp r12d,861E460F
07FFCE6EC9710	0F84 57010000	je bingmaps.7FFCE6EC986D
07FFCE6EC9716	41:81FC 9DBE4960	cmp r12d,6049BE9D
07FFCE6EC971D	0F84 4A010000	je bingmaps.7FFCE6EC986D
07FFCE6EC9723	41:81FC 88498A76	cmp r12d,768A4988
07FFCE6EC972A	0F84 3D010000	je bingmaps.7FFCE6EC986D
07FFCE6EC9730	41:81FC 40718DFE	cmp r12d,FF8D7140

Call to Process32Next:

The screenshot displays a debugger interface. On the left, assembly code is shown with the following instructions: `mov rdi,rdx`, `test rdx,rdx`, `je kernel32.7FFD303B5392`, `cmp dword ptr ds:[rdx],130`, `jb kernel32.7FFD303B5392`, `lea rdx,qword ptr ss:[rsp+40]`, `mov dword ptr ss:[rsp+40],238`, `call <kernel32.Process32NextW>`, `and qword ptr ss:[rsp+38],0`, `lea rcx,qword ptr ds:[rdi+2C]`, `and qword ptr ss:[rsp+30],0`, `lea r8,qword ptr ss:[rsp+6C]`, `mov dword ptr ss:[rsp+28],104`, `or r9d,FFFFFFFF`, `mov qword ptr ss:[rsp+20],rcx`, `mov edx,400`, `xor ecx,ecx`, `mov ebx,eax`, `call qword ptr ds:[<wideCharToMultiByte>]`, `nop dword ptr ds:[rax+rax],eax`, `mov rax,qword ptr ss:[rsp+50]`, `mov ecx,dword ptr ss:[rsp+44]`, `mov qword ptr ds:[rdi+10],rax`, `mov eax,dword ptr ss:[rsp+58]`, `mov dword ptr ds:[rdi+18],eax`, `mov eax,dword ptr ss:[rsp+5C]`, `mov dword ptr ds:[rdi+1C],eax`, `mov eax,dword ptr ss:[rsp+60]`. On the right, the register window shows the following values: RAX: 0000000000000001, RCX: 0000006026EFF724 (labeled "[System Process]"), RDX: 0000000000000000, RBP: 0000006026EFF678, RSP: 0000006026EFF380, RSI: 0000000000000000, R8: 0000006026EFF3EC (labeled "L\"System\""), R9: 0000006026EFF678, R10: 0000000000000000, R11: 0000000000000246 (labeled "L'g'"), R12: 0000000000000000, R13: 0000000000000000, R14: 00007FFD303B52D0 (labeled "<kernel32.Process32Next>"), R15: 000000000000001E (labeled "L'A'"), and RIP: 00007FFD303B5336 (labeled "kernel32.00007FFD303B5336").

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:

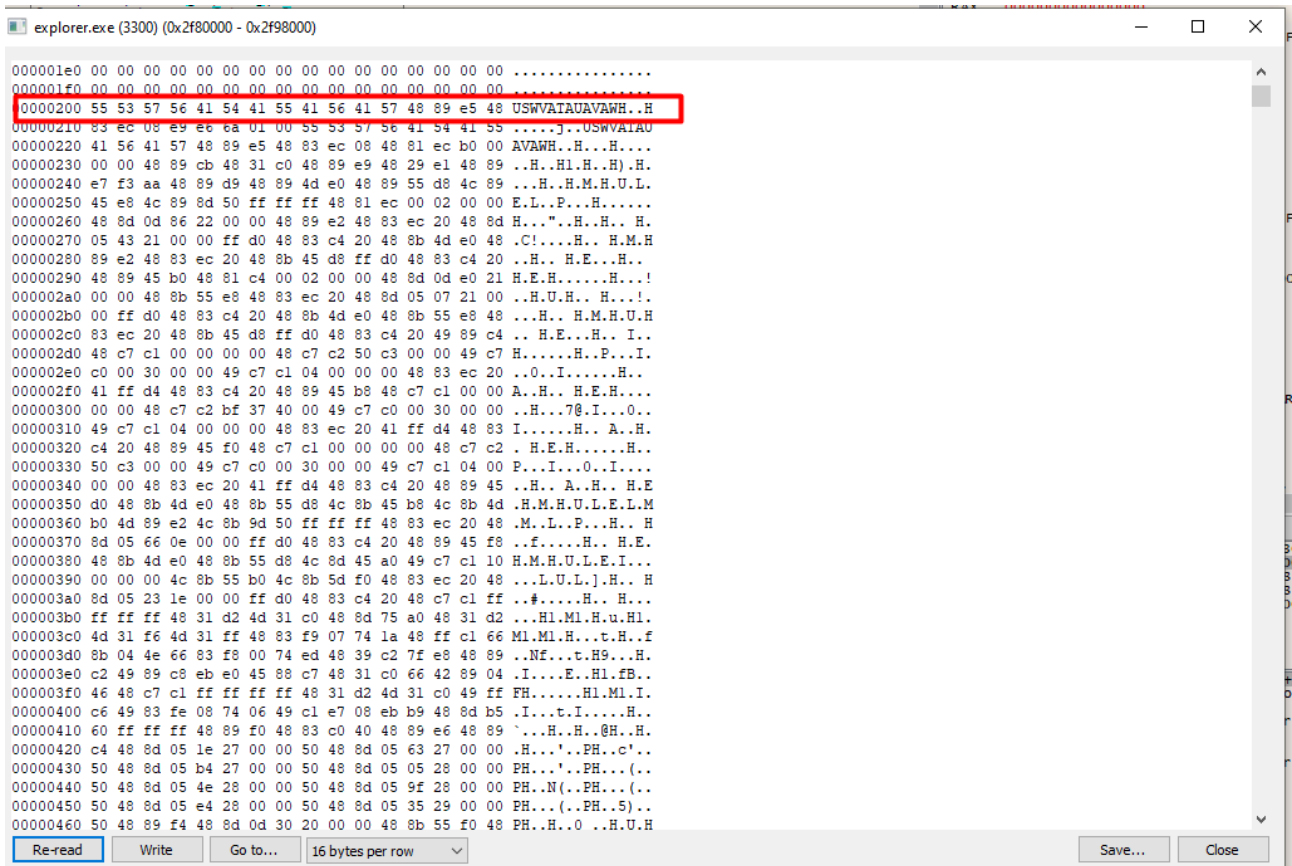
```

explorer.exe (3300) (0x2f80000 - 0x2f98000)

00000000 43 3a 5c 55 73 65 72 73 5c 44 79 6e 61 6d 69 63 C:\Users\Dynamic
00000010 5c 44 65 73 6b 74 6f 70 5c 57 69 6b 69 4c 6f 61 \Desktop\WikiLoa
00000020 64 65 72 5c 62 65 66 30 34 65 33 62 32 62 38 31 der\bef04e3b2b81
00000030 66 32 64 65 65 33 39 63 34 32 61 62 39 62 65 37 f2dee39c42ab9be7
00000040 38 31 66 33 64 62 30 30 35 39 65 63 37 32 32 61 81f3db0059ec722a
00000050 65 65 65 33 62 35 34 33 34 63 32 65 36 33 35 31 eee3b5434c2e6351
00000060 32 61 36 38 5c 6e 70 70 2e 38 2e 36 2e 70 6f 72 2a68\npp.8.6.por
00000070 74 61 62 6c 65 2e 78 36 34 5c 70 6c 75 67 69 6e table.x64\plugin
00000080 73 5c 6d 69 6d 65 54 6f 6f 6c 73 5c 44 4c 4c 4c s\mimeTools\DLLL
00000090 6f 61 64 65 72 36 34 5f 38 35 37 33 2e 65 78 65 oader64_8573.exe
000000a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
    
```

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/bef04e3b2b81f2dee39c42ab9be781f3db0059ec722ae4e3b5434c2e63512a68/>
- <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>