

Malware:- Bumblebee Malware

Hash:- c65c51ed60f91a92789c4b056821ef51252baa2a1679a6513ab008acf0464ccb

Tools:- x64dbg, Cutter

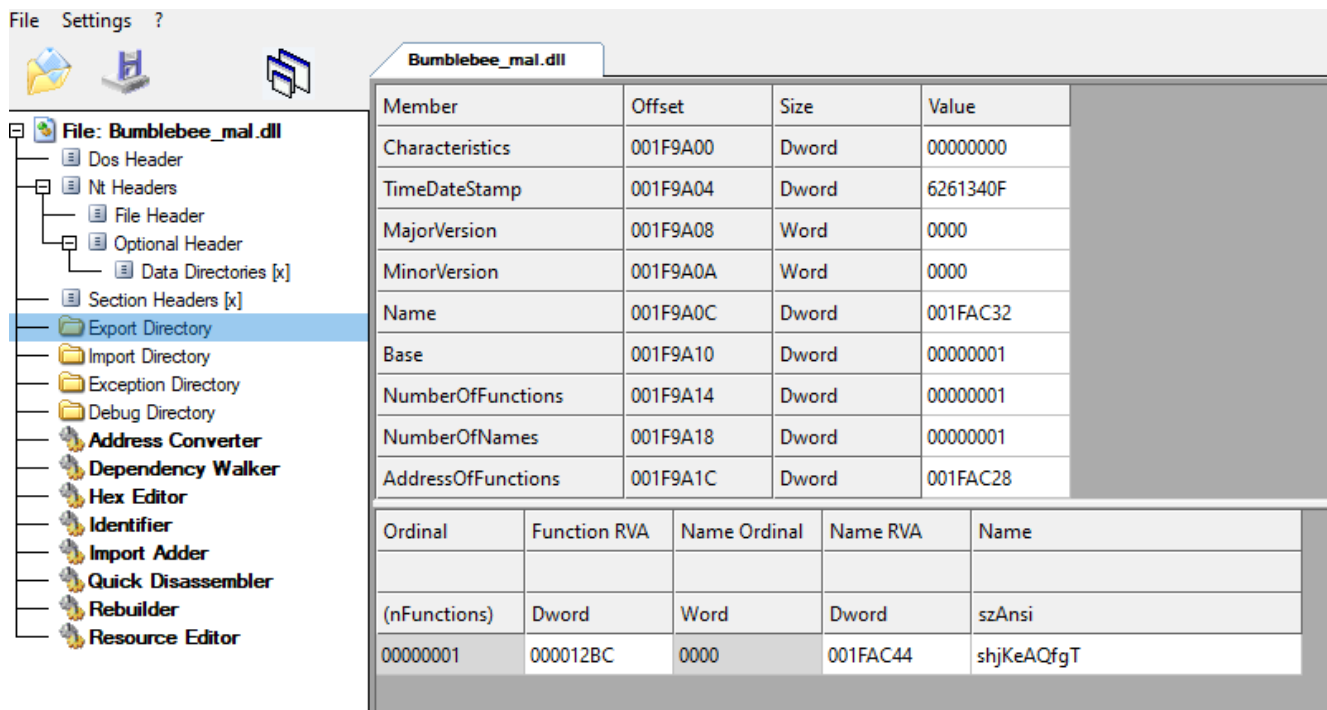
Overview:-

Bumblebee malware acts as a loader primarily used to drop other malware's on the system. It comprises of multi-stage payload delivery. The loader is usually executed using malicious macros or Powershell scripts which goes on to execute the payload. In certain attacks the attacker may use phishing emails that contain a ISO file. Once opened the ISO file contains a .LNK shortcut file which on opening runs a hidden malicious Powershell script which executes the Bumblebee DLL malware.

BUMBLEBEE LOADER DLL ANALYSIS

Export Function

The binary contains an export function called `shjKeAQfgT`, which loads the payload.



Img:- The DLL exports function `shjKeAQfgT`

Dynamic API address retrieval

The malware retrieves APIs using:

- `LoadLibraryA()`
- `GetProcAddress()`

The APIs retrieved using this technique are:

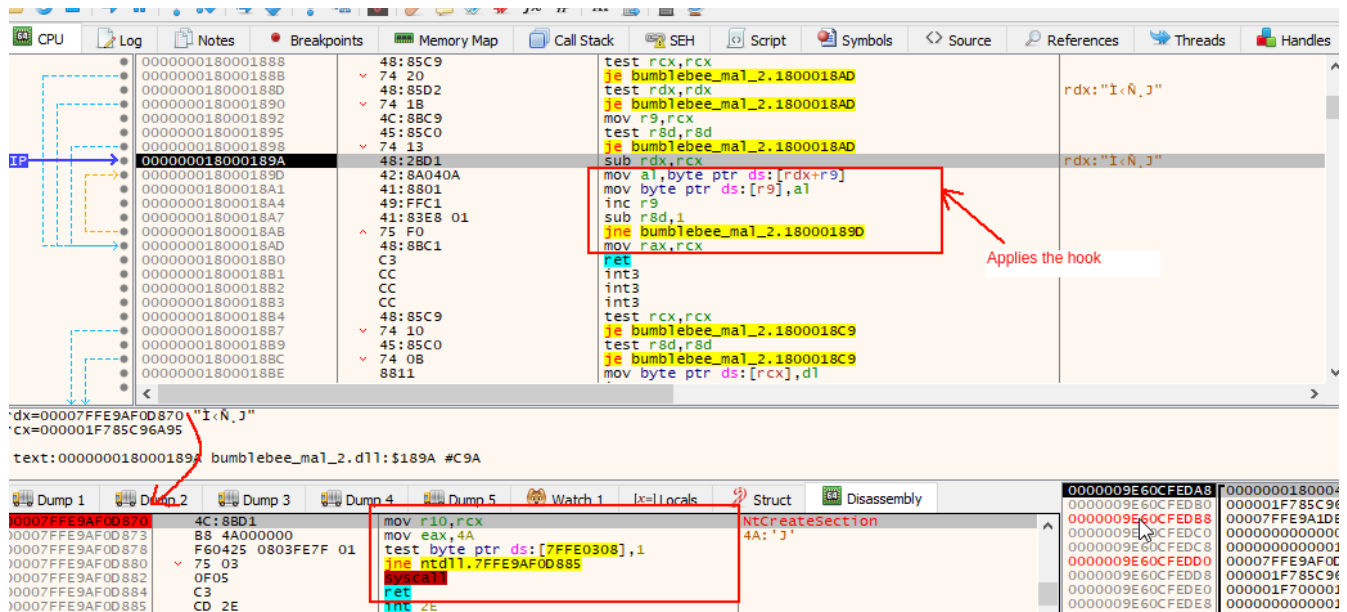
- `ntOpenFile()`
- `ntCreateSection()`
- `ntMapViewOfSection()`
- `VirtualProtect()`

Inline hooking in bumblebee loader

The APIs retrieved in the previous section are hooked in order to execute the code of the malware to load the payload and map it to the memory region of the malware.

In order to apply the hook the malware first changes the memory protection rights of the memory region where the API is located to read, write and execute

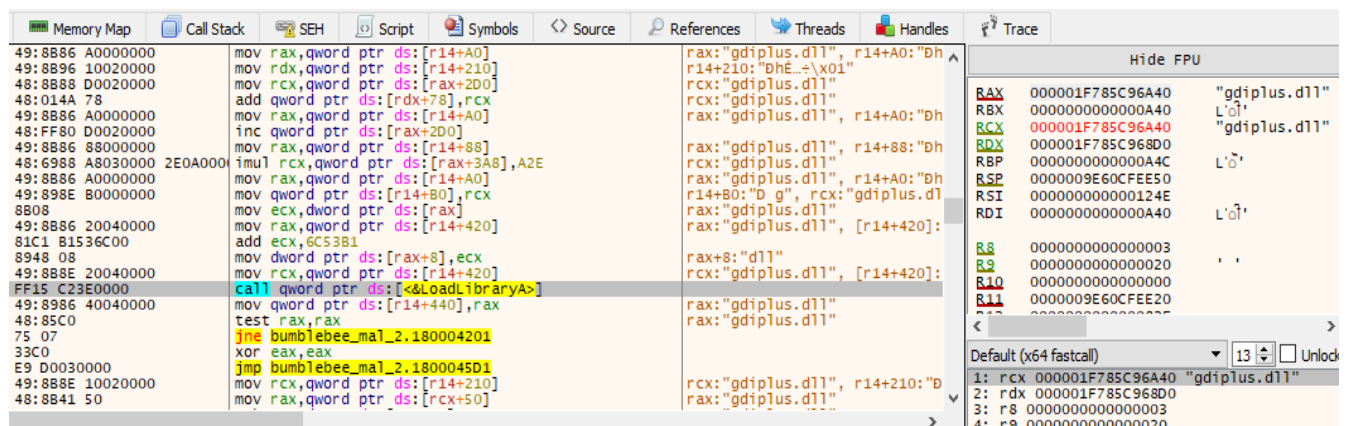
using VirtualProtect() API. Saves the first 0xD bytes. It then applies the hook which will point to some address in the malware code itself. The malware then changes the rights back again to read, write.



Img:- NtCreateSection before getting overwritten by the malware

Using LoadLibrary API to hide the malware functionality

In order to execute the payload DLL the malware itself does not call any function, it uses the inbuilt LoadLibrary API. LoadLibrary internally calls ntOpenFile, ntCreateSection, ntMapViewOfSection to open the requested DLL, create a new section and map that section to the memory region at runtime.



Img:- LoadLibrary to load gdiplus.dll

Therefore the malware patches these APIs as explained in the previous section, so that when the LoadLibrary API is called, the hooked APIs are redirected to malware code.

The entire process is as follows:-

- The malware calls the LoadLibraryA API for the file gdiplus.dll.

- The LoadLibraryA API calls hooked ntOpenFile API.
 - The hooked ntOpenFile jumps to address 0x1800019D8 inside the malware.
 - This function removes the hook from the ntOpenFile and calls the ntOpenFile with parameter gdiplus.dll as the API normally should.
- The LoadLibraryA API then calls the hooked ntCreateSection
 - The hooked ntCreateSection jumps to address 0x18000169c inside the malware.
 - The function removes the hook from the ntCreateSection API and call the API, passing the handle to gdiplus.dll obtained previously.
- The LoadLibraryA API calls the hooked ZwMapViewOfSection API
 - The patched ZwMapViewOfSection does the actual creation of section and mapping for the payload
 - The patched API jumps to address 0x180001F10.
 - This function calls the ntCreateSection API again, which creates a new section which is not backed by file.
 - Section Page Protection :- 0x40 (PAGE_EXECUTE_READWRITE)
 - Allocation Attributes :- 0x80000000 (SEC_COMMIT)
 - The ZwMapViewOfSection API is called passing the handle to the newly created section with parameter:
 - Win32Protect :- 0x40 (PAGE_EXECUTE_READWRITE)
 - After mapping the section, the malware copies the payload to this newly created section.
 - The address of the payload is returned from ZwMapViewOfSection API inside LoadLibrary.
- The LoadLibrary API then loads other DLLs required for the payload and then finally jumps to the entry point of the payload DLL

Bumblebee_mal_2.dll - PID: 1760 - Module: ntdll.dll - Thread: Main Thread 4080 - x64dbg [Elevated]

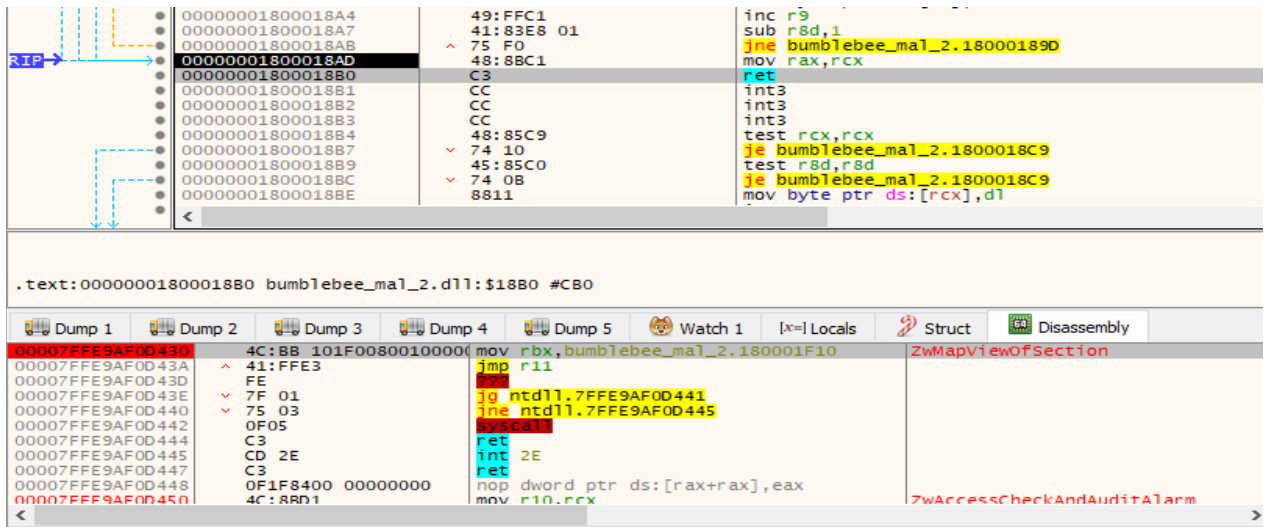
File View Debug Tracing Plugins Favourites Options Help Apr 17 2021 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source Refere

00007FFE9AECFB83	48:8D4D A7	lea rcx,qword ptr ss:[rbp-59]
00007FFE9AECFB87	E8 D4D90300	call <ntdll.NtOpenFile>
00007FFE9AECFB8C	8BD8	mov ebx,eax
00007FFE9AECFB8E	85C0	test eax,eax
00007FFE9AECFBC0	0F88 1FDA0500	js ntdll.7FFE9AF2D5E5
00007FFE9AECFBC6	85DB	test ebx,ebx
00007FFE9AECFBC8	0F88 C3000000	js ntdll.7FFE9AECFC91
00007FFE9AECFBCE	803D 83C81000 00	cmp byte ptr ds:[7FFE9AFDC458],0
00007FFE9AECFBD5	0F85 D4000000	jne ntdll.7FFE9AECFCAF
00007FFE9AECFBD8	BE 00000001	mov esi,1000000
00007FFE9AECFBE0	8577 20	test dword ptr ds:[rdi+20],esi
00007FFE9AECFBE3	0F85 53DA0500	jne ntdll.7FFE9AF2D63C
00007FFE9AECFBE9	833D 38C91000 00	cmp dword ptr ds:[7FFE9AFDC528],0
00007FFE9AECFBF0	0F85 85DA0500	jne ntdll.7FFE9AF2D678
00007FFE9AECFBF6	48:8845 A7	mov rax,qword ptr ss:[rbp-59]
00007FFE9AECFBFA	48:8D4D AF	lea rcx,qword ptr ss:[rbp-51]
00007FFE9AECFBFE	48:894424 30	mov qword ptr ss:[rsp+30],rax
00007FFE9AECFC03	45:33C9	xor r9d,r9d
00007FFE9AECFC06	897424 28	mov dword ptr ss:[rsp+28],esi
00007FFE9AECFC0A	45:33C0	xor r8d,r8d
00007FFE9AECFC0D	C74424 20 10000000	mov dword ptr ss:[rsp+20],10
00007FFE9AECFC15	41:8D51 0D	lea edx,qword ptr ds:[r9+0]
00007FFE9AECFC19	E8 52DC0300	call <ntdll.NtCreateSection>

0007FFE9AF0D590 "I<N,3"

Img:- hooked NtOpenFile and NtCreateSection called inside LoadLibrary



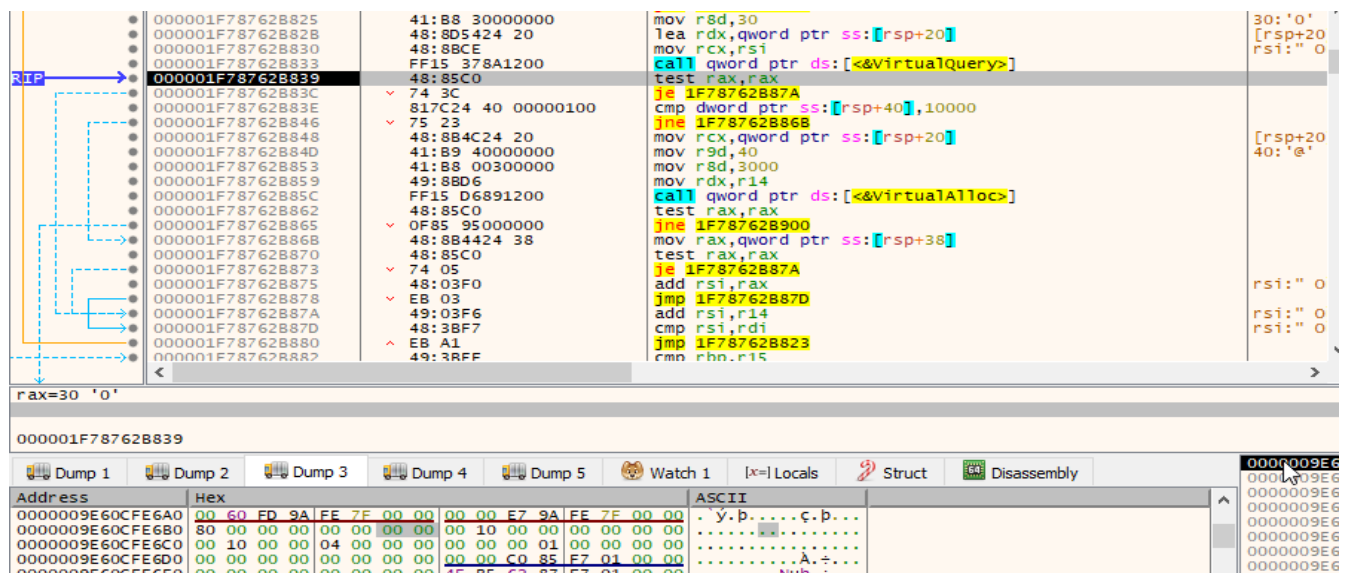
Img:- ZwMapViewOfSection after inline hook placed

PAYLOAD DLL ANALYSIS

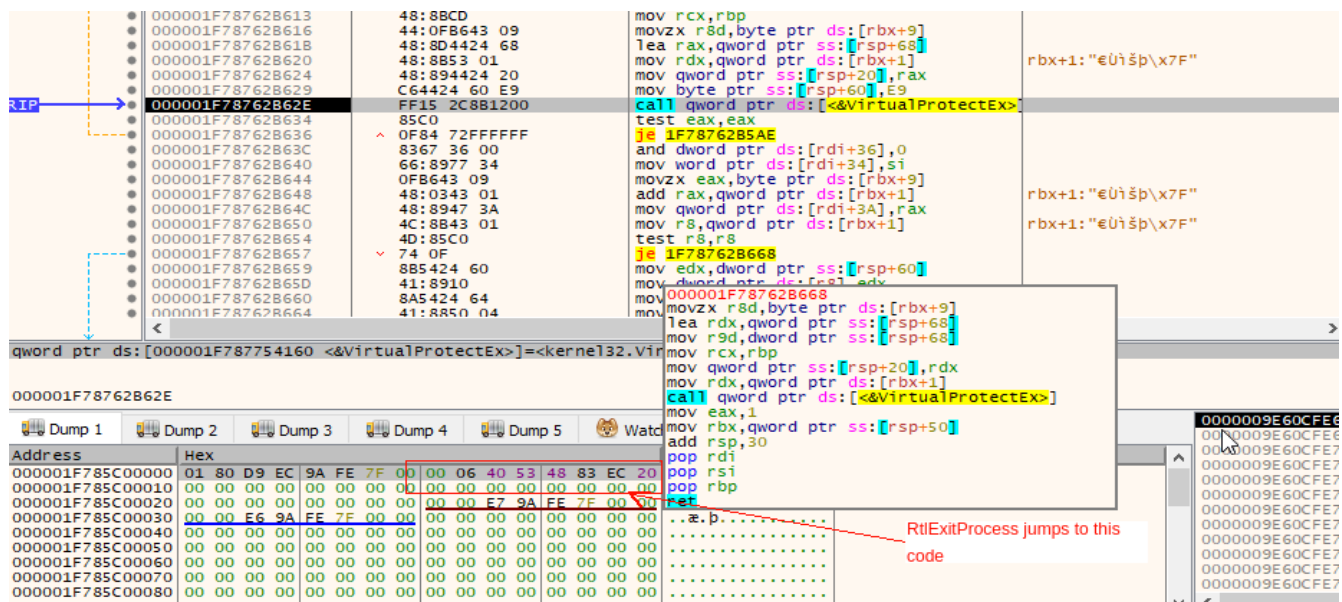
Inline Hooking

After starting execution at the entry point of the DLL, the malware first hooks the `RtlExitUserProcess` API. The malware does this as follows:-

- First uses the base address of the `ntdll.dll` and starts searching for a memory region inside different sections of the DLL whose state is marked as `0x10000` (`MEM_FREE`) using the `VirtualQuery()` API.
- After finding the region it then allocates memory in that region using `VirtualAlloc()` API.
- Copies the bytes to be executed in this region and patches the original `RtlExitUserProcess` to point to this block of code which jumps to code inside the payload DLL.



Img:- VirtualQuery called to check memory state

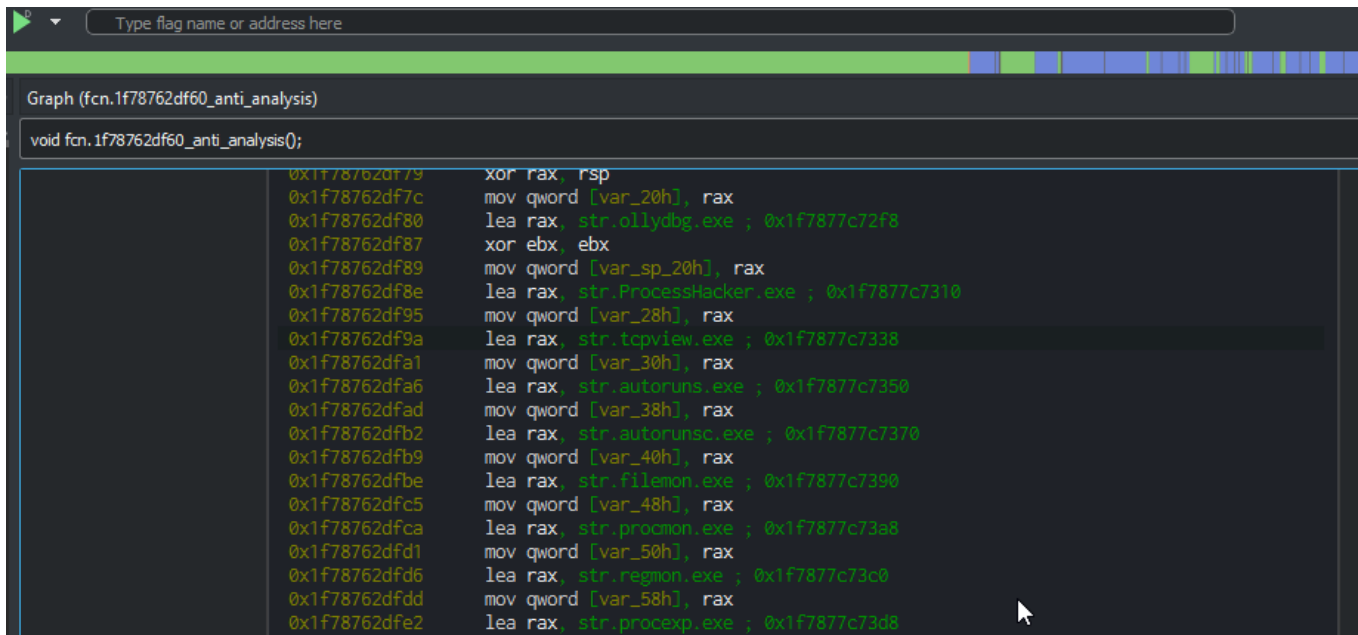


Img:- VirtualProtect called to change rights of RtlExitUserProcess

Multiple Threads and Anti-Analysis tricks

The malware then does the following:-

- Calls CreateThread API with the following parameters
 - lpStartAddress :- fcn.1F787721CA4
 - lpParameter :- fcn.1F7875f90C0 (Thread 2)
- Function fcn.1f787721CA4 executes the instruction “call rdi” where rdi is the parameter fcn.1F7875f90C0.
- Function fcn.1F7875f90C0 is the function that performs all the activities of the payload.
 - The function first calls the fcn.1f78762dc50 which performs the anti-analysis functionality.
 - It then calls the CreateThread API again with the following parameters
 - lpStartAddress :- fcn.1F787721CA4
 - lpParameter :- fcn.1f78762df60 (thread 3)
 - Thread 3 executes whenever thread 2 goes to sleep or waits for certain operation. Therefore at certain intervals thread 3 would execute which will call function fcn.1f78762df60 which perform anti-analysis functionality, in this case check for known processes like x64dbg, processhacker etc

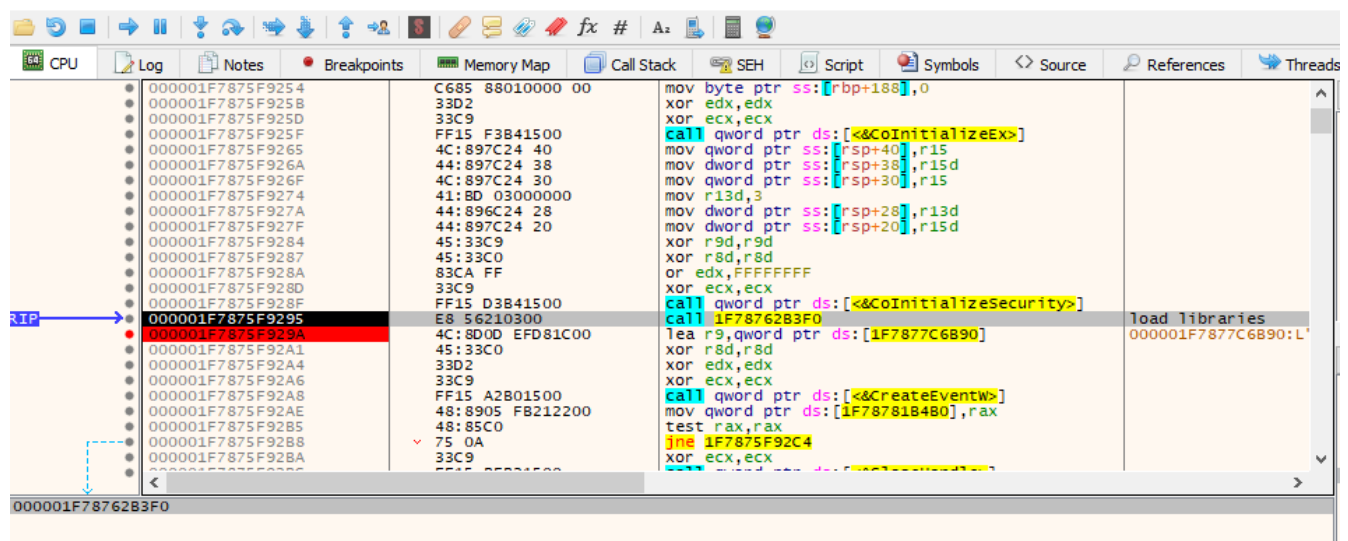


Img:- fcn.1f78762df60 checks for known process in between execution

WMI queries and data exfiltration

The malware uses Windows management instrumentation queries(WMI) to obtain information about the system. The APIs used to perform the queries in order are :-

- CoInitializeSecurity()
- CoCreateInstance()
- IWbemLocator::ConnectServer()
- CoSetProxyBlanket()
- IWbemServices::ExecQuery()
- IEnumWbemClassObject::Next()
- IwbemClassObject::Get()



Img:- CoInitializeSecurity() API called to initialize the environment for WMI

000001F78762D6A1 44: 886D 98 mov byte ptr ss:[rbp-68],r13b
 000001F78762D6A5 4C: 896C24 60 mov qword ptr ss:[rsp+60],r13
 000001F78762D6AA 4C: 896C24 58 mov qword ptr ss:[rsp+58],r13
 000001F78762D6AF 4C: 896D 80 mov qword ptr ss:[rbp-80],r13
 000001F78762D6B3 4C: 896D 88 mov qword ptr ss:[rbp-78],r13
 000001F78762D6B7 48: C745 88 0F000000 mov qword ptr ss:[rbp-78],F
 000001F78762D6BF 4C: 896D 80 mov qword ptr ss:[rbp-80],r13
 000001F78762D6C3 44: 886C24 70 mov byte ptr ss:[rsp+70],r13b
 000001F78762D6C8 48: 8D4424 60 lea rax,qword ptr ss:[rsp+60]
 000001F78762D6CD 48: 894424 20 mov qword ptr ss:[rsp+20],rax
 000001F78762D6D2 4C: 8D0D 9F431800 lea r9,qword ptr ds:[1F787781A78]
 000001F78762D6D9 33D2 xor edx,edx
 000001F78762D6DB 45: 8D45 01 lea r8d,qword ptr ds:[r13+1]
 000001F78762D6DB 48: 8D0D 82431800 lea rcx,qword ptr ds:[1F787781A68]
 000001F78762D6E6 FF15 5C701200 call qword ptr ds:[1F787754748] CoCreateInstance
 000001F78762D6EC 85C0 test eax,eax
 000001F78762D6EE 0F89 A1000000 jns 1F78762D795
 000001F78762D6F4 48: C743 18 0F000000 mov qword ptr ds:[rbx+18],F
 000001F78762D6FC 4C: 896B 10 mov qword ptr ds:[rbx+10],r13
 000001F78762D700 44: 8828 mov byte ptr ds:[rbx],r13b
 000001F78762D703 48: 8D5424 70 lea rdx,qword ptr ss:[rsp+70]
 000001F78762D708 48: 8BC8 mov rcx,rbx
 000001F78762D70B E8 6090FCFF call 1F7875F6770
 000001F78762D710 C74424 50 01000000 mov dword ptr ss:[rsp+50],1
 000001F78762D718 48: 8845 88 mov rax,qword ptr ss:[rbp-78]

qword ptr ds:[000001F787754748]=apphe!p.00007FFE95F21EA0

000001F78762D6E6

Img:- CoCreateInstance() called

000001F78762D790 E9 9B040000 jmp 1F78762DC30
 000001F78762D795 48: 884C24 60 mov rcx,qword ptr ss:[rsp+60]
 000001F78762D79A 48: 8B01 mov rax,qword ptr ds:[rcx]
 000001F78762D79D 48: 8D5424 58 lea rdx,qword ptr ss:[rsp+58]
 000001F78762D7A2 48: 895424 40 mov qword ptr ss:[rsp+40],rdx
 000001F78762D7A7 4C: 896C24 38 mov qword ptr ss:[rsp+38],r13
 000001F78762D7AC 4C: 896C24 30 mov qword ptr ss:[rsp+30],r13
 000001F78762D7B1 44: 896C24 28 mov dword ptr ss:[rsp+28],r13d
 000001F78762D7B6 4C: 896C24 20 mov qword ptr ss:[rsp+20],r13
 000001F78762D7BB 45: 33C9 xor r9d,r9d
 000001F78762D7BE 45: 33C0 xor r8d,r8d
 000001F78762D7C1 48: 8D15 08991900 lea rdx,qword ptr ds:[1F7877C70D0]
 000001F78762D7C8 FF50 18 call qword ptr ds:[rax+18] calls ConnectServer
 000001F78762D7CB 85C0 test eax,eax
 000001F78762D7CD 0F89 8F000000 jns 1F78762D862
 000001F78762D7D3 48: 884C24 60 mov rcx,qword ptr ss:[rsp+60]
 000001F78762D7D8 48: 85C9 test rcx,rcx
 000001F78762D7DD 74 06 je 1F78762D7E3
 000001F78762D7E0 48: 8B01 mov rax,qword ptr ds:[rcx]
 000001F78762D7E3 FF50 10 call qword ptr ds:[rax+10]
 000001F78762D7E8 48: C743 18 0F000000 mov qword ptr ds:[rbx+18],F
 000001F78762D7EF 4C: 896B 10 mov qword ptr ds:[rbx+10],r13
 000001F78762D7F2 C603 00 mov byte ptr ds:[rbx],0
 000001F78762D7F7 48: 8D5424 70 lea rdx,qword ptr ss:[rsp+70]
 000001F78762D7F7 48: 8BC8 mov rcx,rbx

qword ptr ds:[rax+18]=[wbemprox.00007FFE8DDC8198]=wbemprox.00007FFE8DDC1080

000001F78762D7C8

Img:- ConnectServer() called to connect to ROOT\CIMV2 namespace on the local machine

000001F78762DAA1 0F84 11010000 je 1F78762B888
 000001F78762DAA7 49: 8BD4 mov rdx,r12
 000001F78762DAAA 48: 8D4D D8 lea rcx,qword ptr ss:[rbp-28]
 000001F78762DAAE E8 C9AFFFFF call 1F787628A7C
 000001F78762DAB3 90 nop
 000001F78762DAB4 48: 8D55 D8 lea rdx,qword ptr ss:[rbp-28]
 000001F78762DAB8 48: 837D F0 08 cmp qword ptr ss:[rbp-10],8
 000001F78762DABD 48: 0F4355 D8 cmovae rdx,qword ptr ss:[rbp-28]
 000001F78762DAC2 48: 8B4D 90 mov rcx,qword ptr ss:[rbp-70]
 000001F78762DAC6 48: 8B01 mov rax,qword ptr ds:[rcx]
 000001F78762DAC9 4C: 896C24 28 mov qword ptr ss:[rsp+28],r13
 000001F78762DACE 4C: 896C24 20 mov qword ptr ss:[rsp+20],r13
 000001F78762DAD3 4C: 8D4D 10 lea r9,qword ptr ss:[rbp+10]
 000001F78762DAD7 45: 33C0 xor r8d,r8d
 000001F78762DADA FF50 20 call qword ptr ds:[rax+20] IwbemClassObject->
 000001F78762DADD 48: 8B55 18 mov rdx,qword ptr ss:[rbp+18]
 000001F78762DAE1 48: 85D2 test rdx,rdx
 000001F78762DAE4 0F84 8E000000 je 1F78762D878
 000001F78762DAEA 4C: 896D C8 mov qword ptr ss:[rbp-38],r13

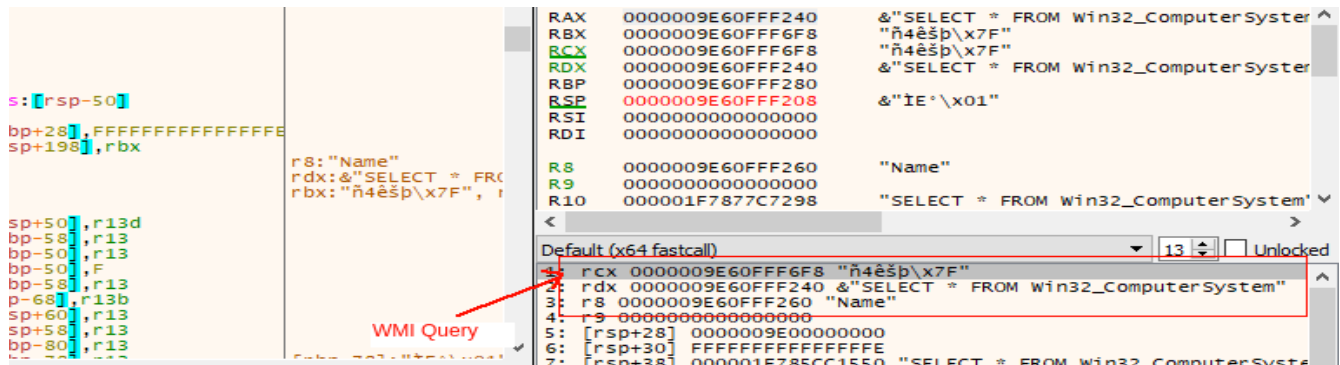
d ptr ds:[rax+20]=[00007FFE87A5EEB8 <fastprox.&?Get@CwbemObject@UEAAJPEBGJPEAutgVARIANT@PEAJ2@Z>]=fastprox.&?Get@CwbemObject@UEAAJPEBGJPEAutgVARIANT@PEAJ2@Z

000001F78762DADA

Img:- Get() method called to retrieve results of the WMI query

The Get() method returns the value requested using the WMI Query. The queries executed by the malware are.

- SELECT * FROM Win32_ComputerSystem → Name
- SELECT * FROM Win32_ComputerSystem → Domain

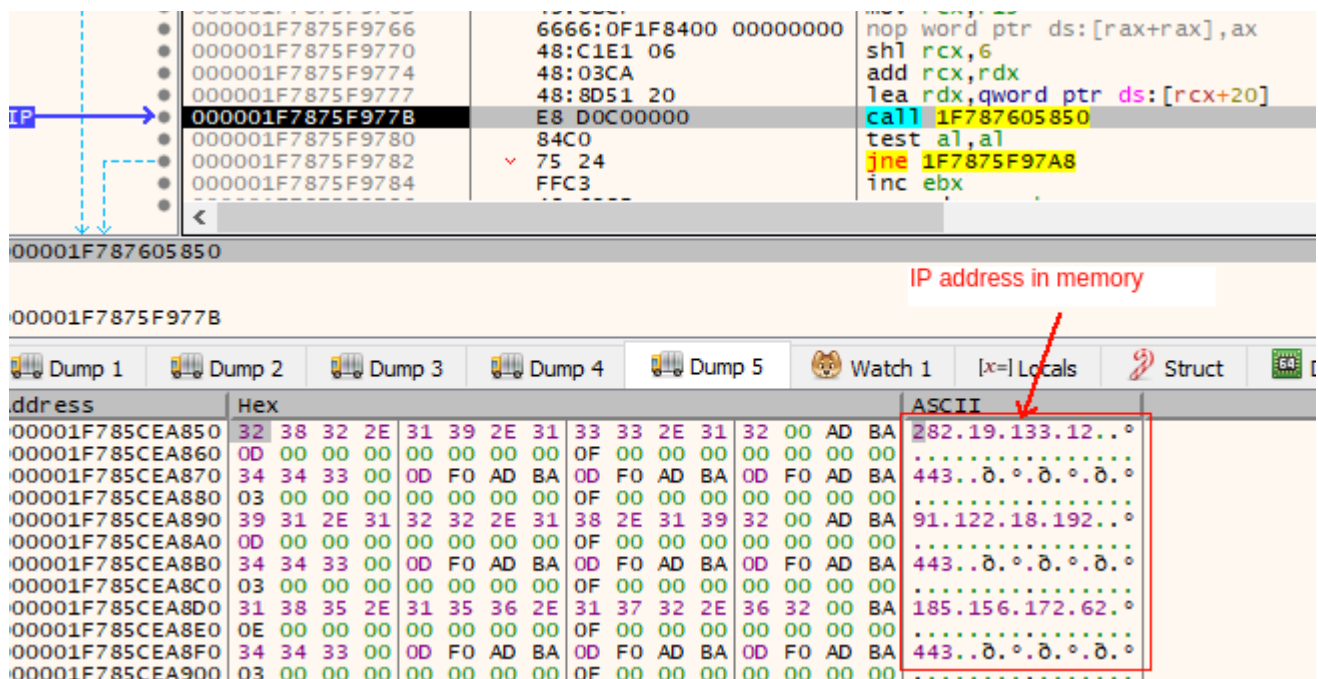


Img:- WMI Query being executed

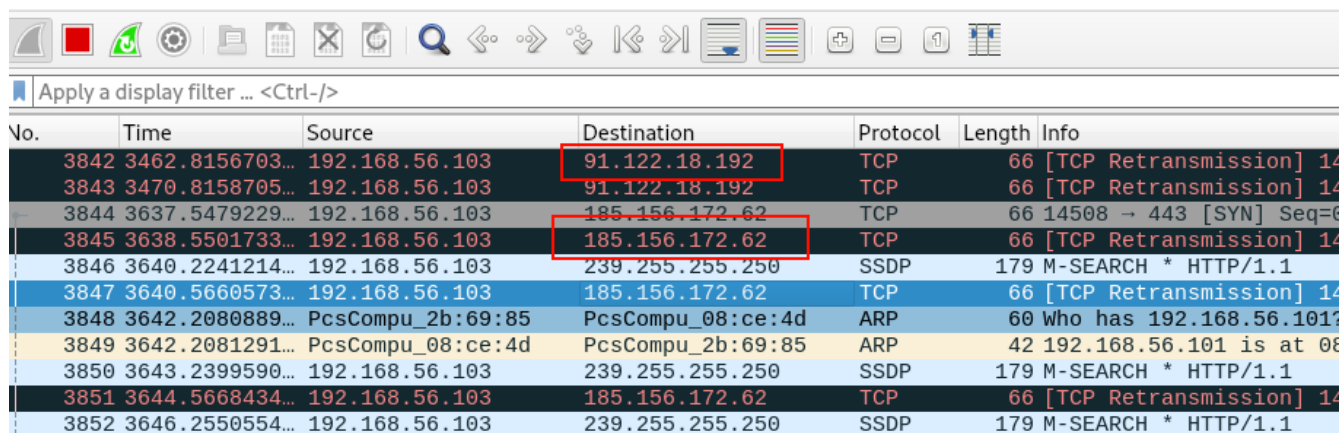
The value returned are stored in memory and later combined to form JSON formatted key value pair data of the form :-

```
{“Client_id”:“d41d8cd98f00b204E9800998ecf8427e”,“group_name”:“2104a”,“sys_version”:“\nDomain name:WORKGROUP”,“client_version”:1}
```

Where Client_id is a generated value and group_name is a value stored in memory. After collecting the required information, the malware tries to connect to the C2 server. If the connection doesn't succeed the malware sleeps and again tries to connect to the next IP stored in memory.



Img:- IP Addresses malware tries to connect to for further operations



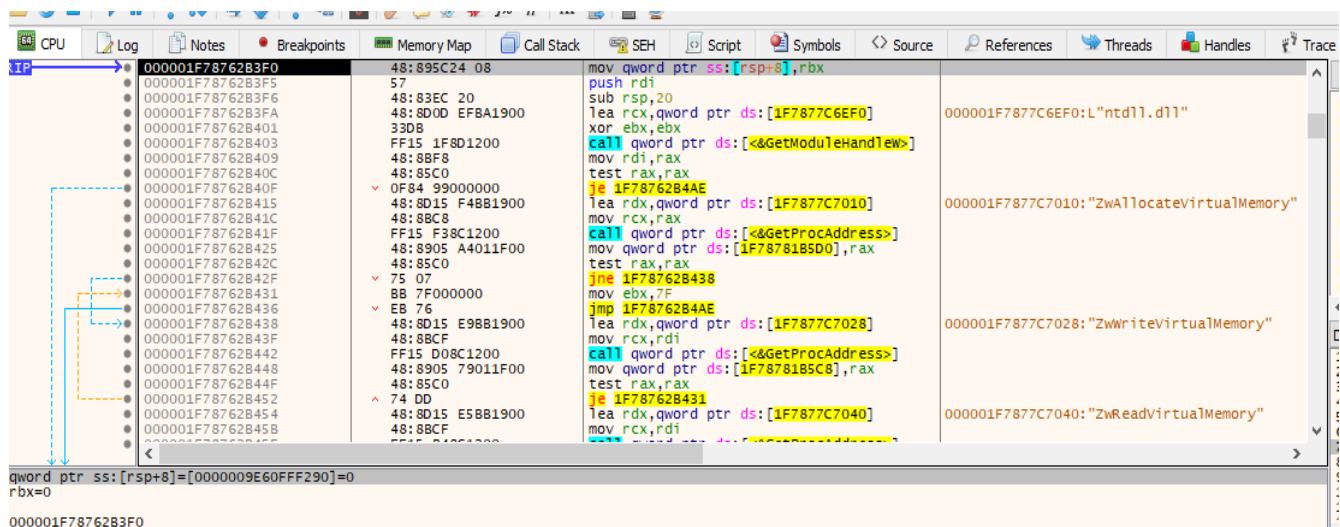
No.	Time	Source	Destination	Protocol	Length	Info
3842	3462.8156703...	192.168.56.103	91.122.18.192	TCP	66	[TCP Retransmission] 14508 → 443 [SYN] Seq=6
3843	3470.8158705...	192.168.56.103	91.122.18.192	TCP	66	[TCP Retransmission] 14508 → 443 [SYN] Seq=6
3844	3637.5479229...	192.168.56.103	185.156.172.62	TCP	66	14508 → 443 [SYN] Seq=6
3845	3638.5501733...	192.168.56.103	185.156.172.62	TCP	66	[TCP Retransmission] 14508 → 443 [SYN] Seq=6
3846	3640.2241214...	192.168.56.103	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
3847	3640.5660573...	192.168.56.103	185.156.172.62	TCP	66	[TCP Retransmission] 14508 → 443 [SYN] Seq=6
3848	3642.2080889...	PcsCompu_2b:69:85	PcsCompu_08:ce:4d	ARP	60	Who has 192.168.56.101?
3849	3642.2081291...	PcsCompu_08:ce:4d	PcsCompu_2b:69:85	ARP	42	192.168.56.101 is at 08
3850	3643.2399590...	192.168.56.103	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
3851	3644.5668434...	192.168.56.103	185.156.172.62	TCP	66	[TCP Retransmission] 14508 → 443 [SYN] Seq=6
3852	3646.2550554...	192.168.56.103	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1

Img:- Malware tries to establish connection to C2

Process injection and checking for infected machine

In addition to the above task performed by the malware the malware also has process injection functionality as it imports the following APIs from the ntdll.dll.

- ZwAllocateVirtualMemory()
- ZwWriteVirtualMemory()
- ZwReadVirtualMemory()
- ZwGetContextThread()
- ZwSetContextThread()



000001F7876283F0 48:895C24 08 mov qword ptr ss:[rsp+8],rbx

57 push rdi

48:83EC 20 sub rsp,20

48:8D0D EFBA1900 lea rcx,qword ptr ds:[1F7877C6EF0] 000001F7877C6EF0:"ntdll.dll"

330B xor ebx,ebx

FF15 1F8D1200 call qword ptr ds:[&GetModuleHandle] 000001F7877C7010:"ZwAllocateVirtualMemory"

48:8BF8 mov rdi,rcx

48:85C0 test rax,rax

0F84 99000000 je 1F7876284AE

48:8D15 F4B81900 lea rdx,qword ptr ds:[1F7877C7010] 000001F7877C7010:"ZwAllocateVirtualMemory"

48:8BC8 mov rcx,rax

FF15 F38C1200 call qword ptr ds:[&GetProcAddress] 000001F7877C7028:"ZwWriteVirtualMemory"

48:8905 A4011F00 mov qword ptr ds:[1F78781B5D0],rax

48:85C0 test rax,rax

75 07 jne 1F787628438

8B 7F000000 mov ebx,7F

E8 76 jmp 1F7876284AE 000001F7877C7028:"ZwWriteVirtualMemory"

48:8BCF lea rdx,qword ptr ds:[1F7877C7028]

48:8BCF mov rcx,rdi

FF15 D08C1200 call qword ptr ds:[&GetProcAddress] 000001F7877C7040:"ZwReadVirtualMemory"

48:8905 79011F00 mov qword ptr ds:[1F78781B5C8],rax

48:85C0 test rax,rax

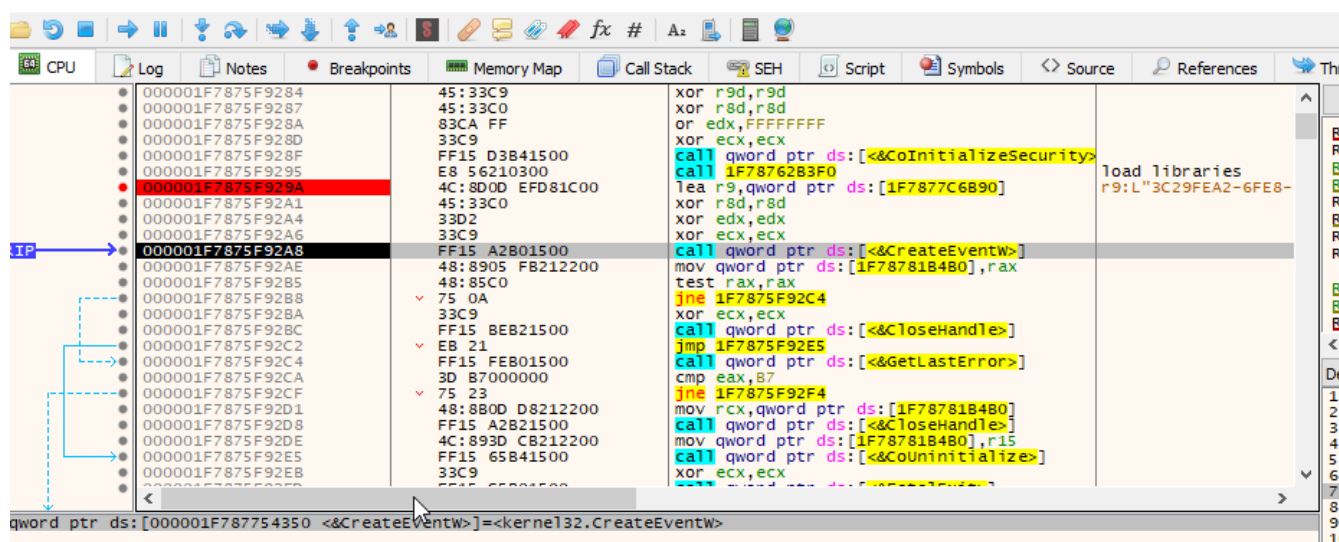
74 DD je 1F787628431

48:8D15 E5B81900 lea rdx,qword ptr ds:[1F7877C7040]

48:8BCF mov rcx,rdi

Img:- Address for ZwAllocateVirtual() and other APIs retrieved

To check if the machine is already infected the malware tries to create an event with the name 3C9FEA2 6FE8 4BF9 B98A 0E3442115F67. If the return value is a handle to the newly created event the malware continues otherwise the error value is checked by comparing it with 0xB7(file already exists). If that is true the malware exits.



Img:- CreateEvent() called with param 3C9FEA2_6FE8_4BF9_B98A_0E3442115F67

INDICATORS OF COMPROMISE

- **Host Based IOCs**
 - File
 - Loader.dll (SHA256):
c65c51ed60f91a92789c4b056821ef51252baa2a1679a6513ab008acf0464ccb
 - Payload.dll (SHA256):
36a54fea5589bdd1f488cbac0412f10d8500121fa06ca1c2fc4c52a987e76204
- **Network Based IOCs**
 - 282.19.133.12:443
 - 91.122.18.192:443
 - 185.156.172.62:443
 - 72.123.65.11:443
 - 149.255.35.167:443
 - 172.241.27.146:443