Malware:- Bumblebee Malware

Hash:- c65c51ed60f91a92789c4b056821ef51252baa2a1679a6513ab008acf0464ccb

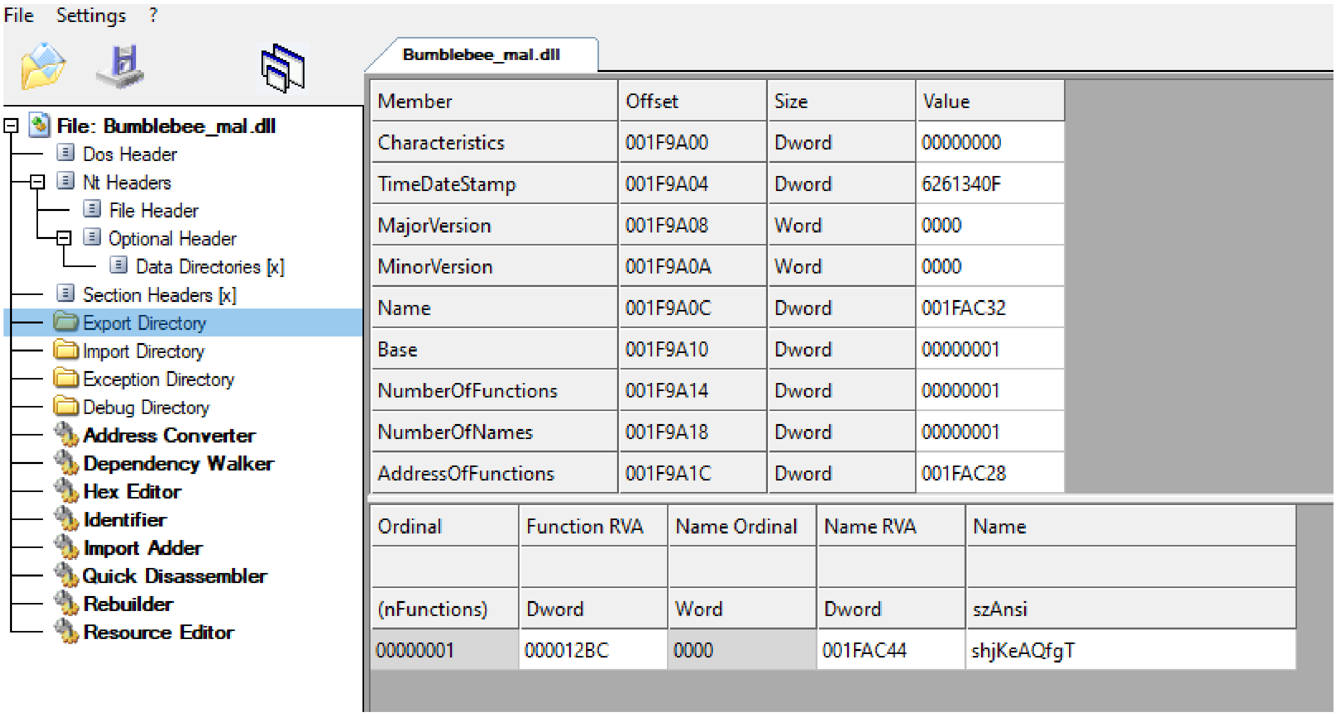
Tools:- x64dbg, Cutter

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()

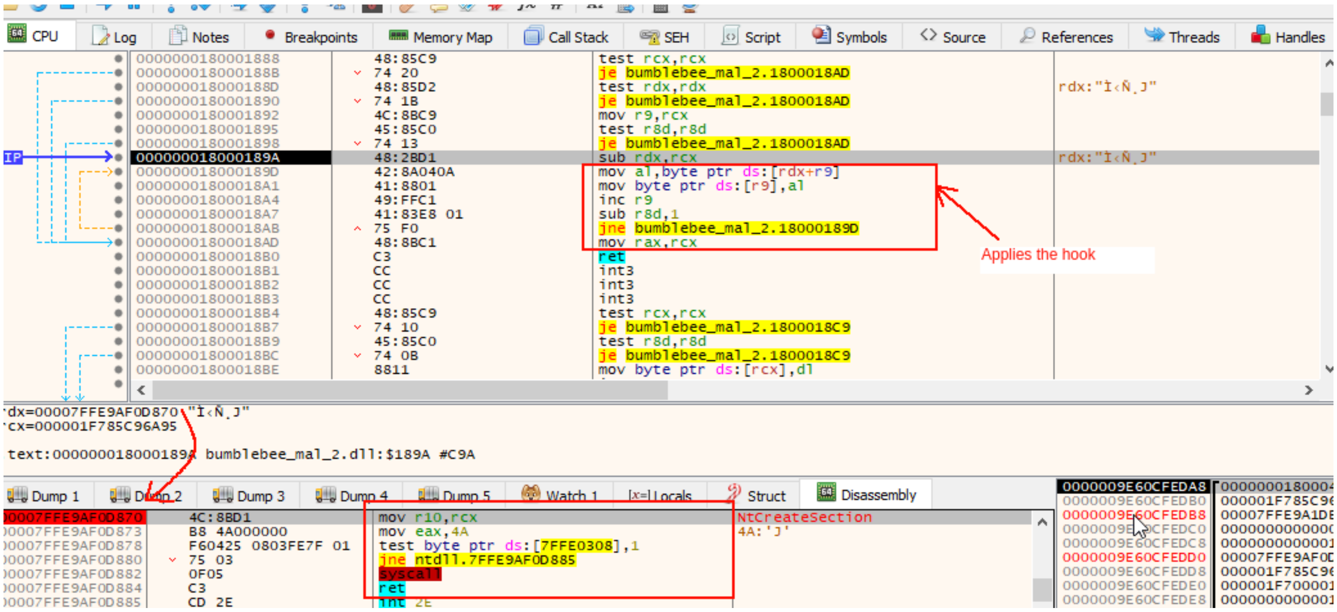
The APIs retrieved using this technique are:

- 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.

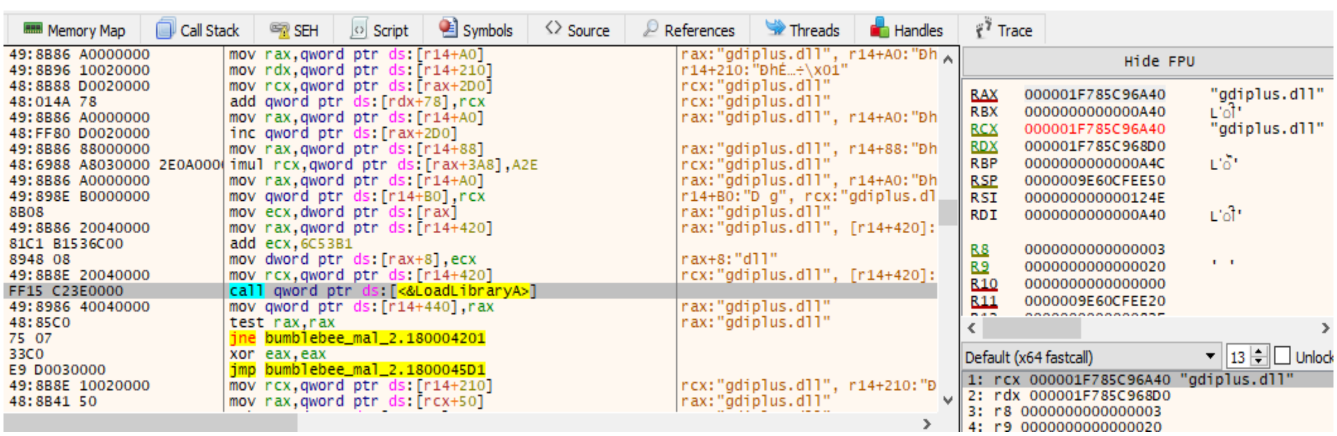
In order to apply the hook the malware first changes the memory protection rights of the memory region where the API is located to read, write and execute using VirtualProtect() API. Saves the first 0xD bytes. It then applies the hook which will point to some address in the malware code itself. The malware 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.



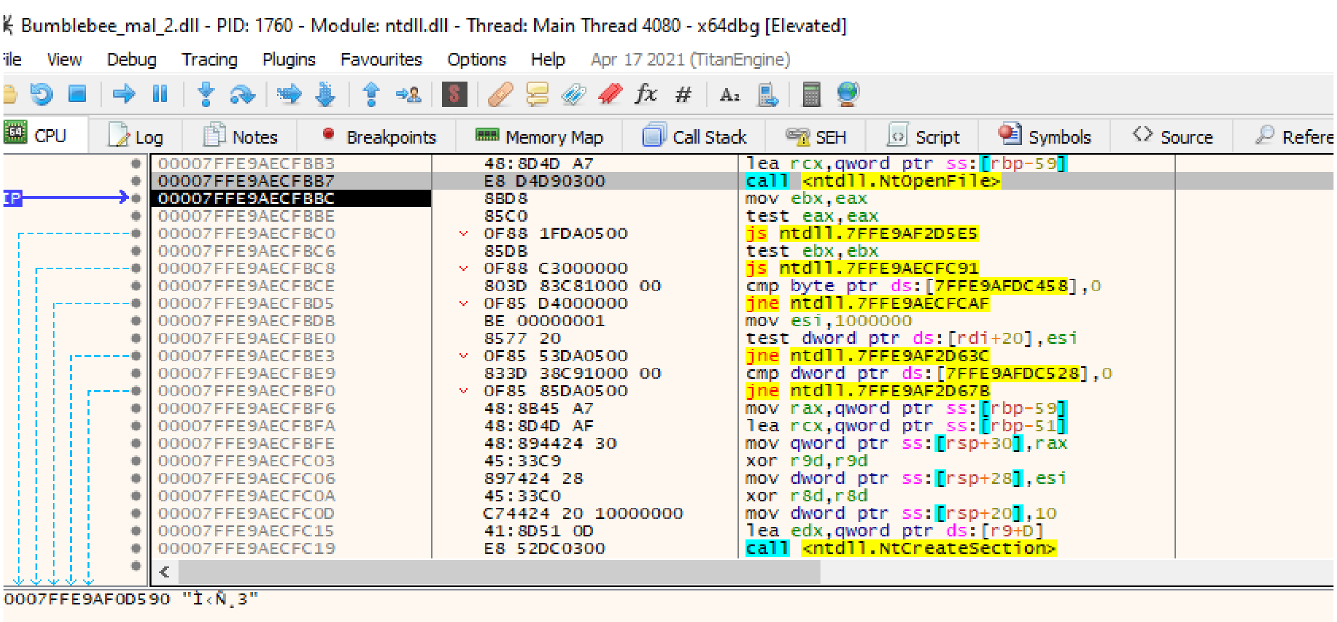
Img:- LoadLibrary to load gdiplus.dll

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.

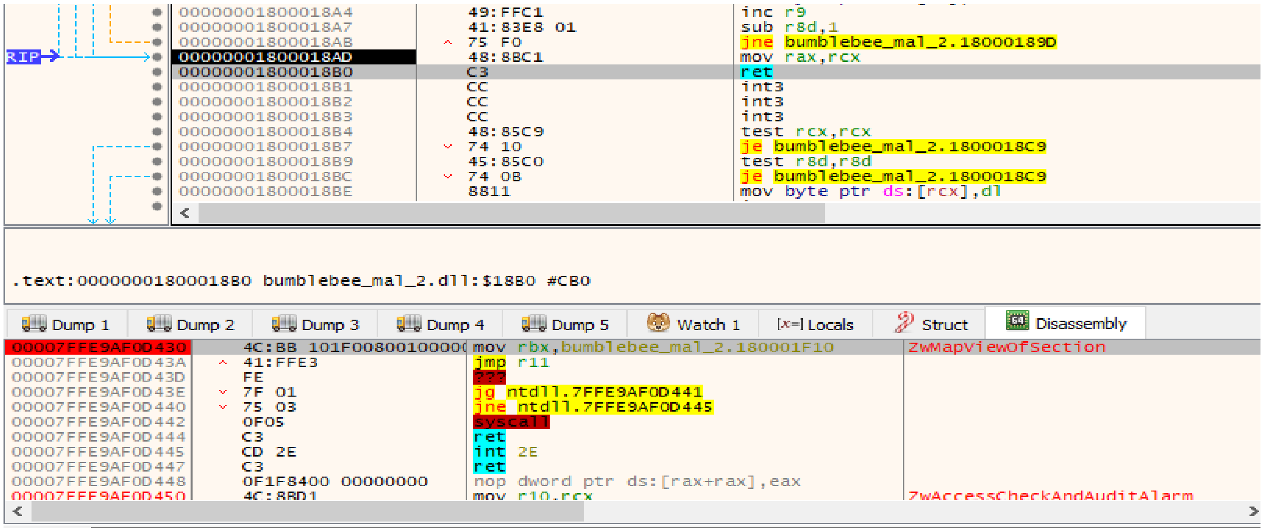
The entire process is as follows:-

- 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 :- 0x8000000 (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



Img:- hooked NtOpenFile and NtCreateSection called inside LoadLibrary



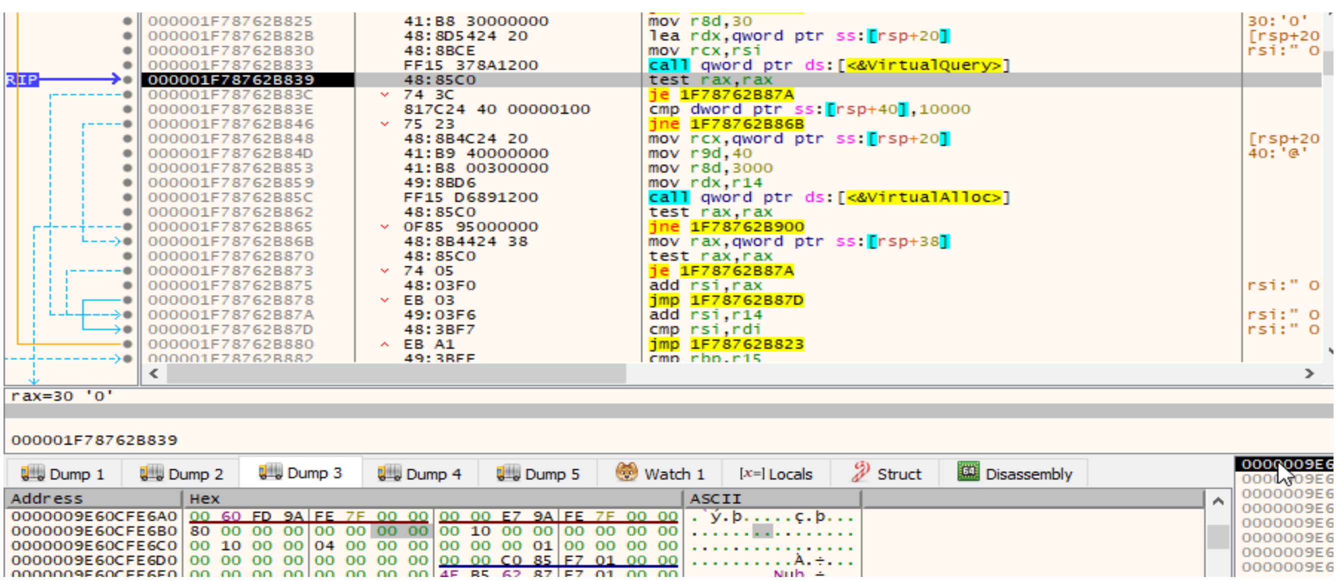
Img:- ZwMapViewOfSection after inline hook placed

PAYLOAD DLL ANALYSIS

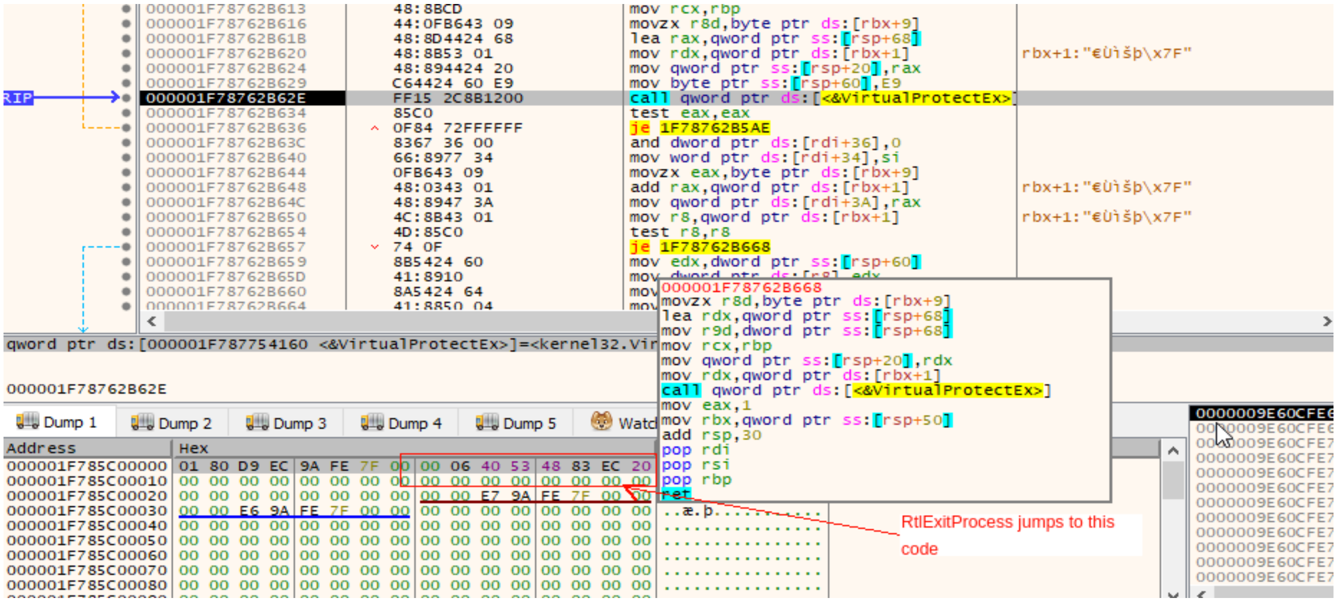
Inline Hooking

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

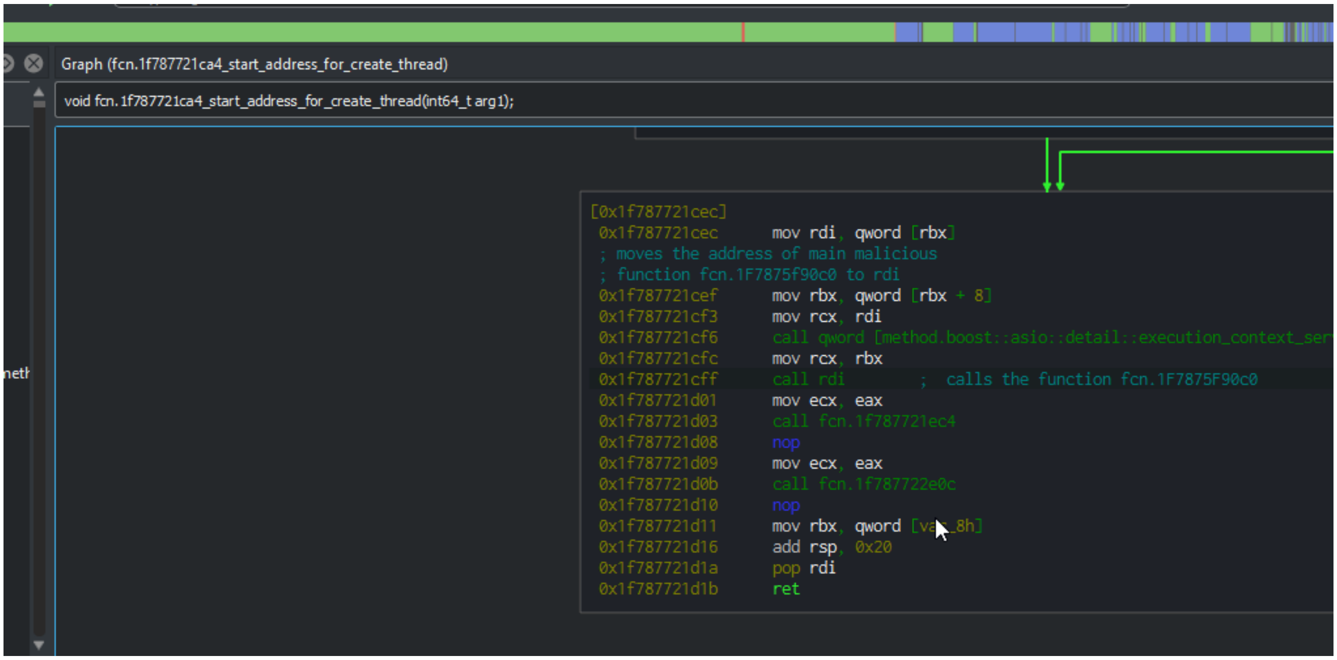


Img:- VirtualProtect called to change rights of RtlExitUserProcess

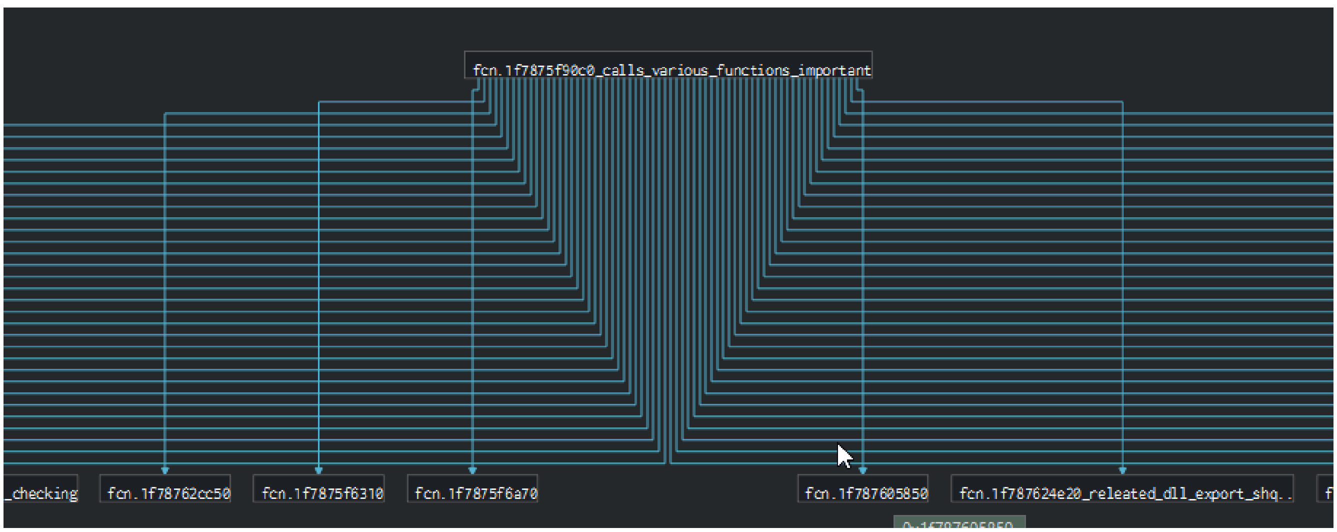
<u>**Multiple Threads and Anti-Analysis tricks**</u>

The malware then does the following:-

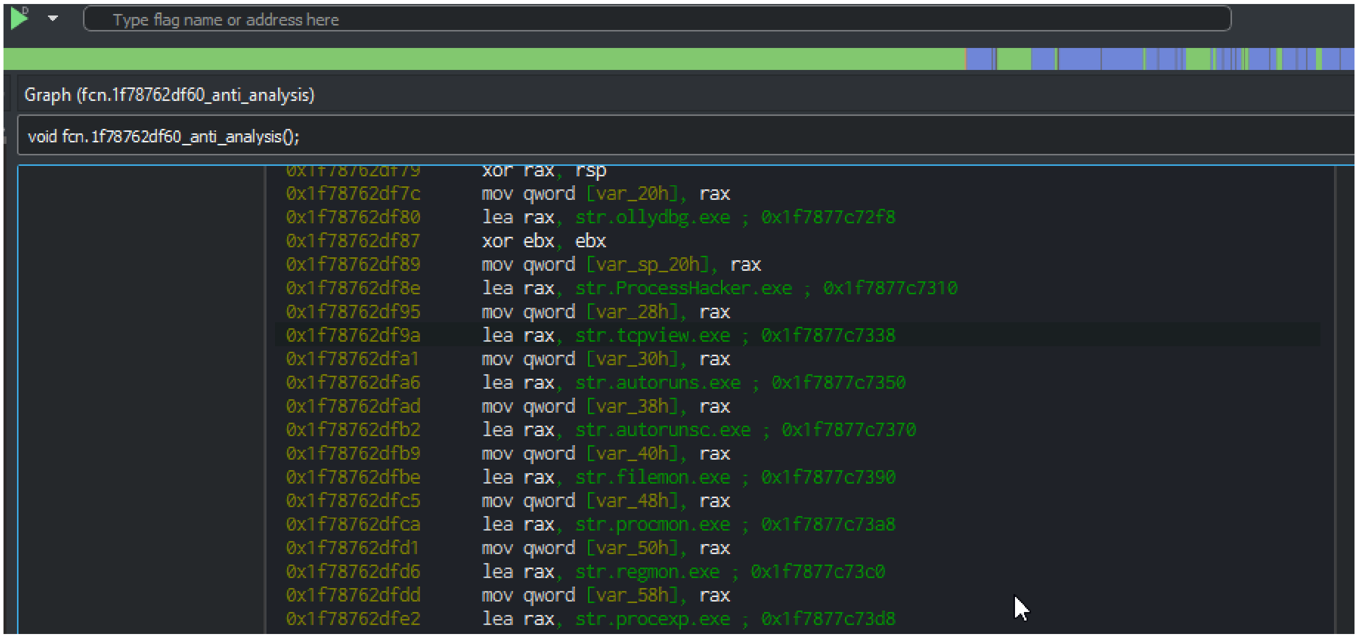
- Calls CreateThread API with the following parameters
  - lpStartAddress :- fcn.1F787721CA4
  - lpParameter :- fcn.1F7875f90C0 (Thread 2)
- Function fcn.1f787721CA4 executes the instruction “call rdi” where rdi is the parameter fcn.1F7875f90C0.
- <u>Function fcn.1F7875f90C0 is the function that performs all the activities of the payload</u>.
  - 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.1f787721ca4 calling fcn.1f7875f90c0



Img:- fcn.1f7875f90c0 performs the main functionality of the payload

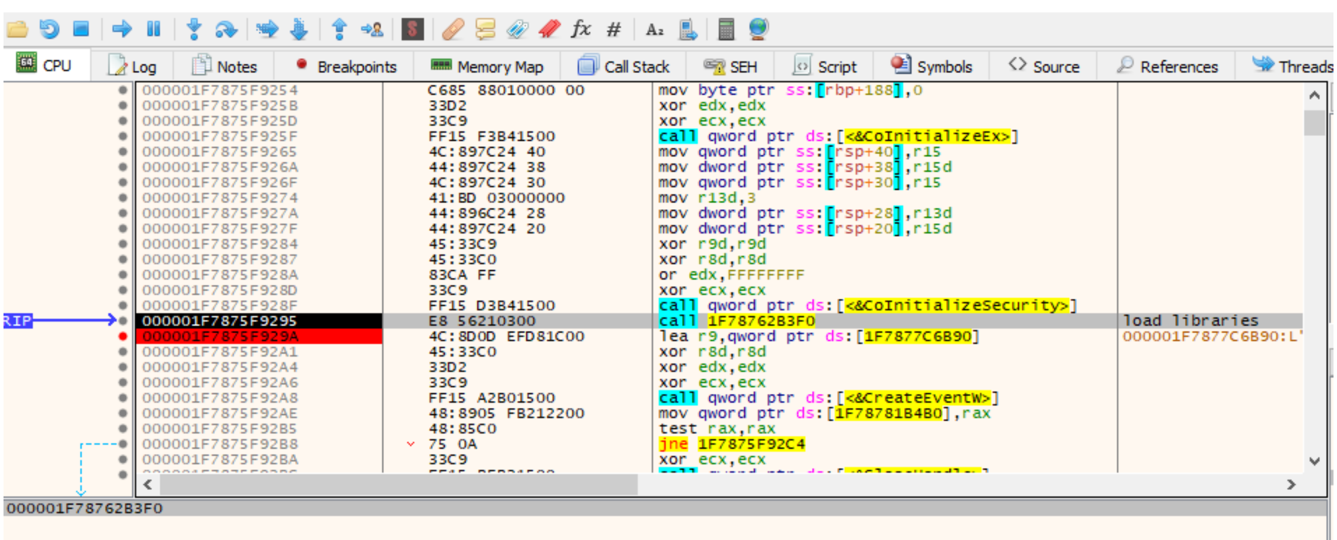


Img:- fcn.1f78762df60 checks for known process in between execution

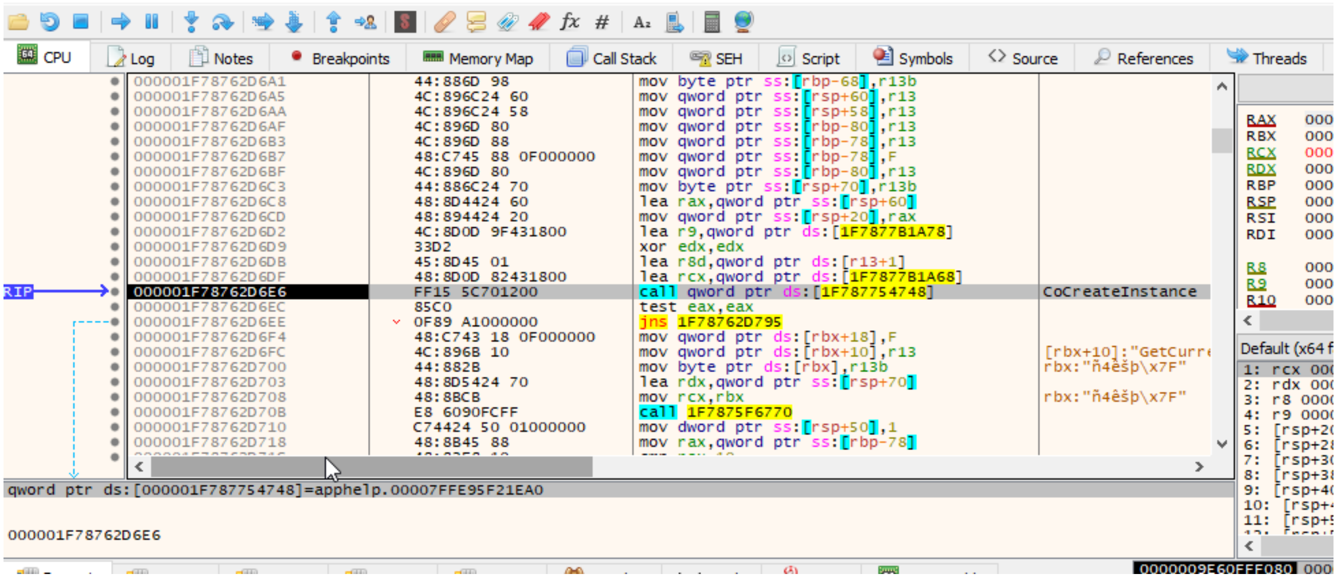
WMI queries and data exfiltration

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

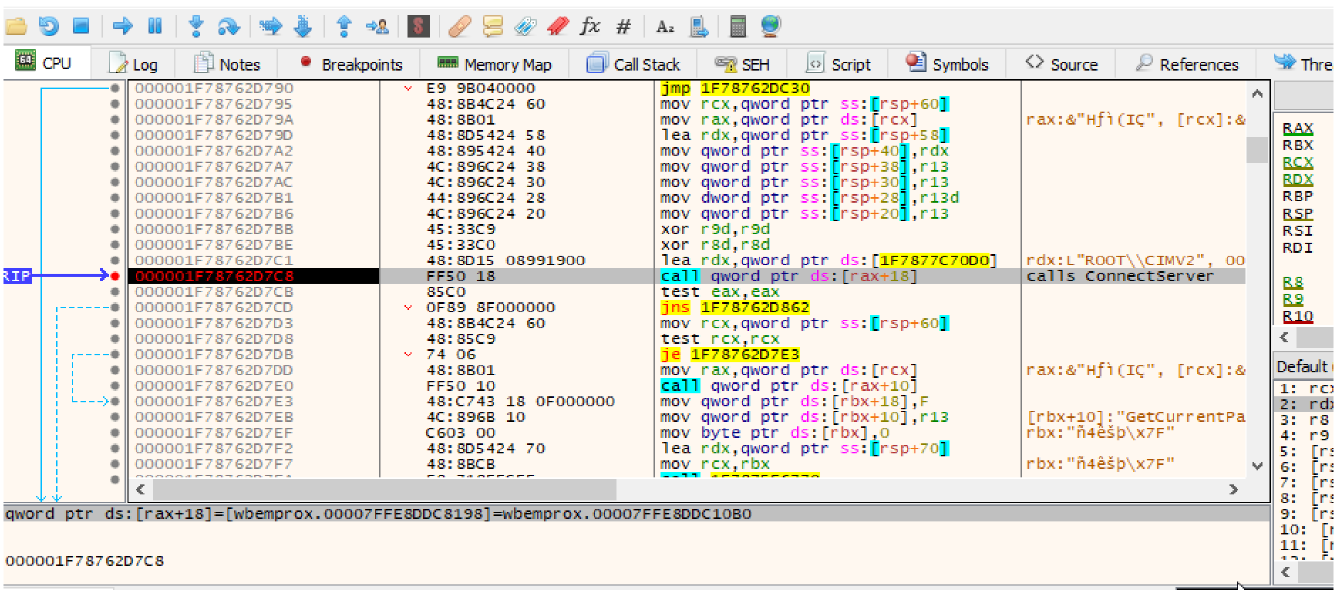
- CoInitializeSecurity()
- CoCreateInstance()
- IWbemLocator::ConnectServer()
- CoSetProxyBlanket()
- IWbemServices::ExecQuery()
- IEnumWbemClassObject::Next()
- IwbemClassObject::Get()



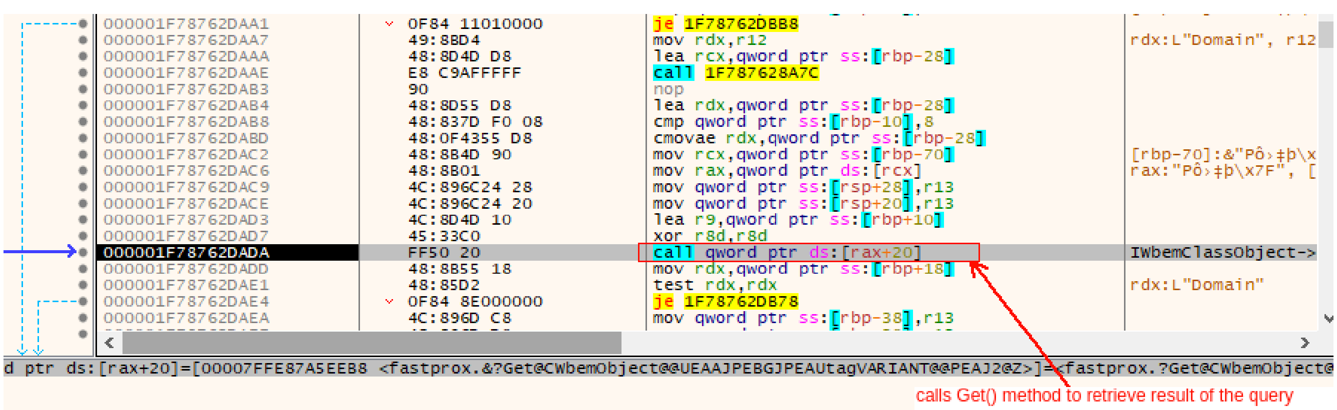
Img:- CoInitializeSecurity() API called to initialize the environment for WMI



Img:- CoCreateInstance() called



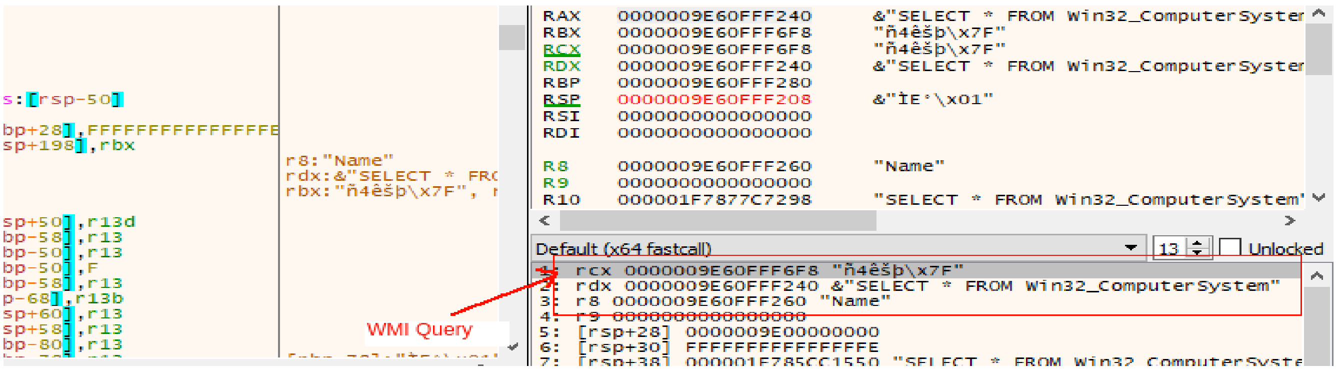
Img:- ConnectServer() called to connect to ROOT\CIMV2 namespace on the local machine



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

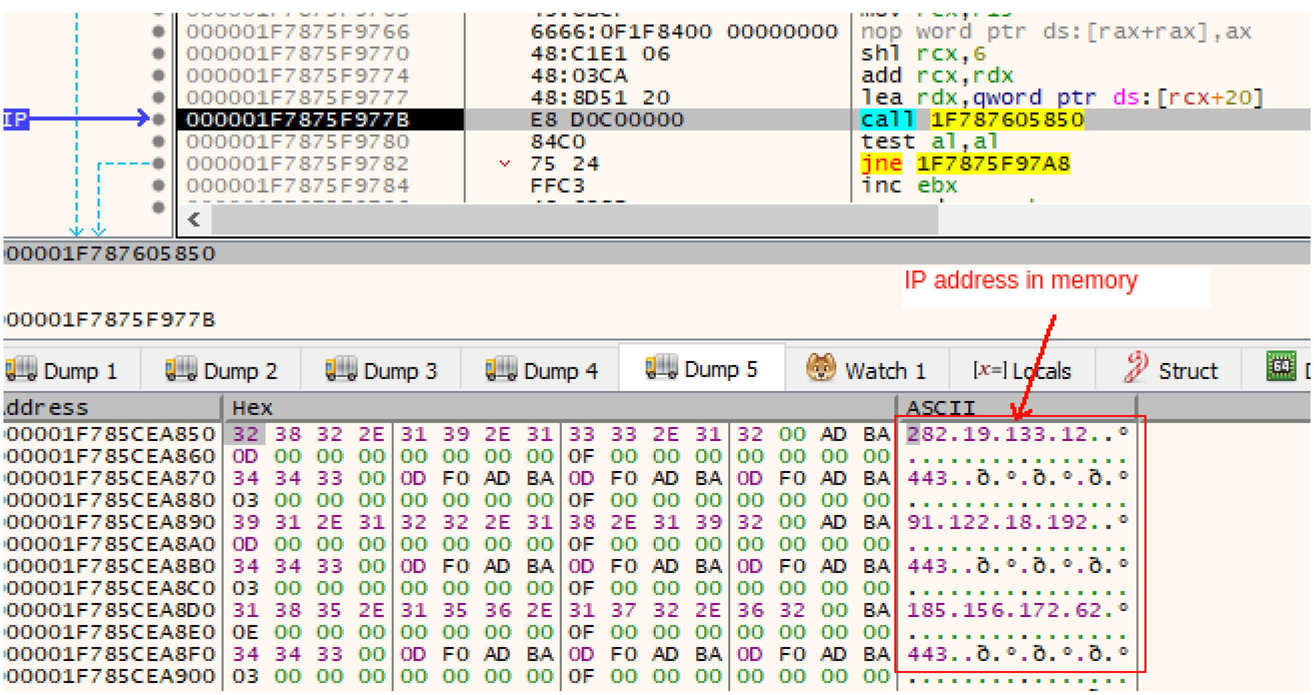


Img:- WMI Query being executed

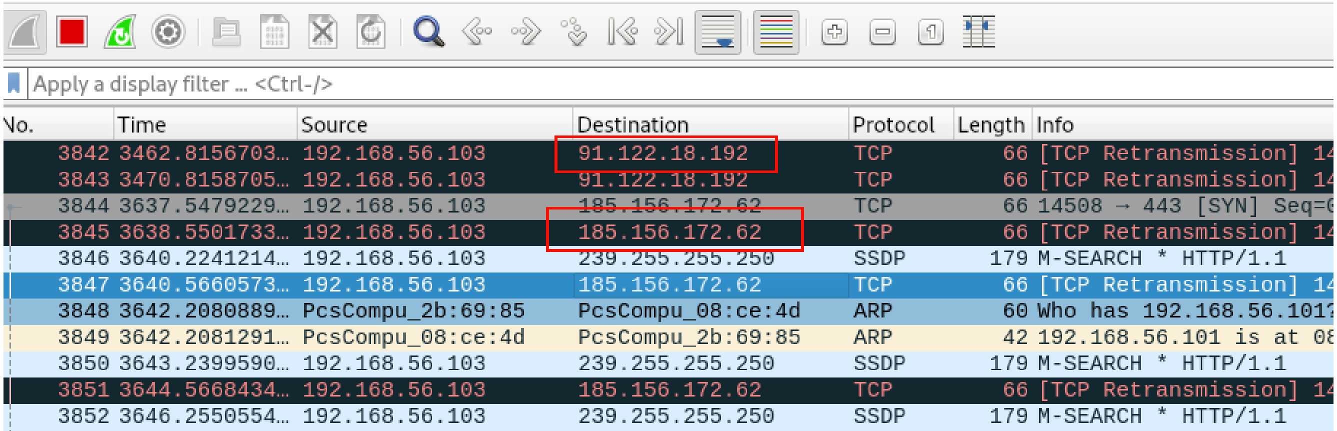
The value returned are stored in memory and later combined to form JSON formatted key value pair data of the form :-

{"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, <u>the malware tries to connect to the C2 server</u>. 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

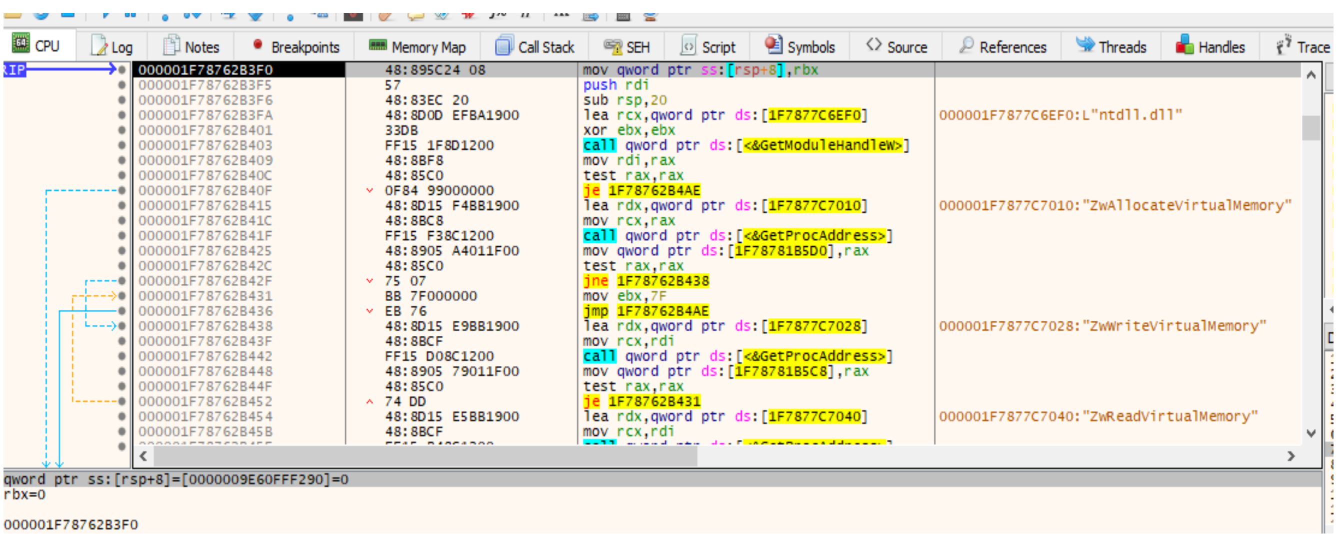


Img:- Malware tries to establish connection to C2

Process injection and checking for infected machine

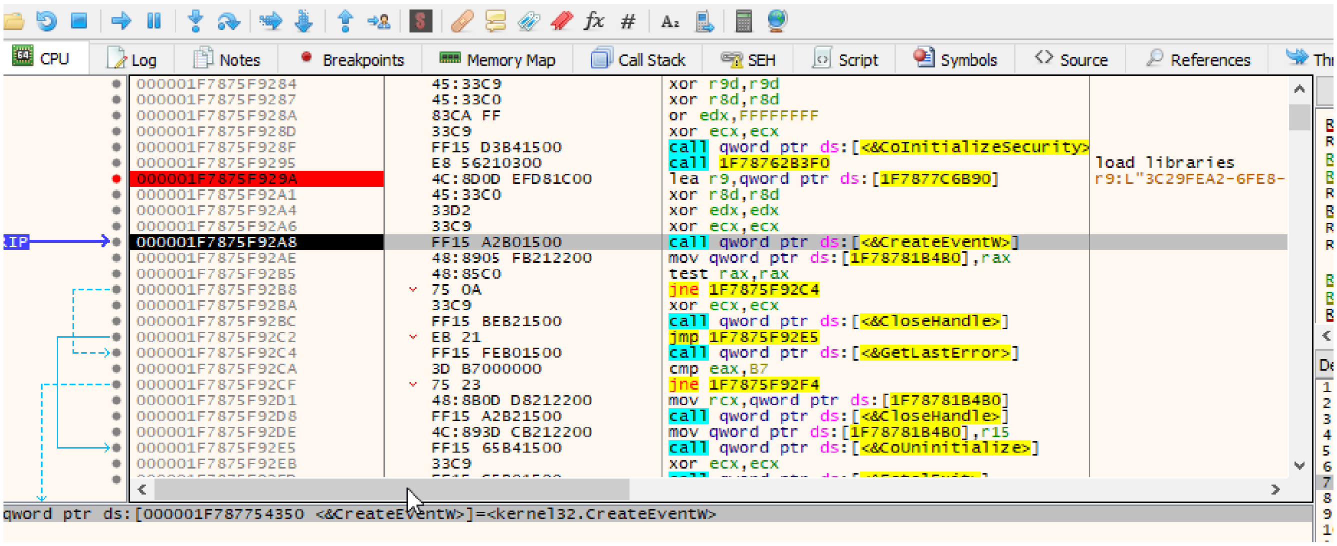
In addition to the above task performed by the malware the malware also has process injection functionality as it imports the following APIs from the ntdll.dll.

- ZwAllocateVirtualMemory()
- ZwWriteVirtualMemory()
- ZwReadVirtalMemory()
- ZwGetContextThread()
- ZwSetContextThread()



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

To check if the machine is already infected the malware tries to <u>create an event with the name 3C9FEA2 6FE8 4BF9 B98A 0E3442115F67</u>. 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