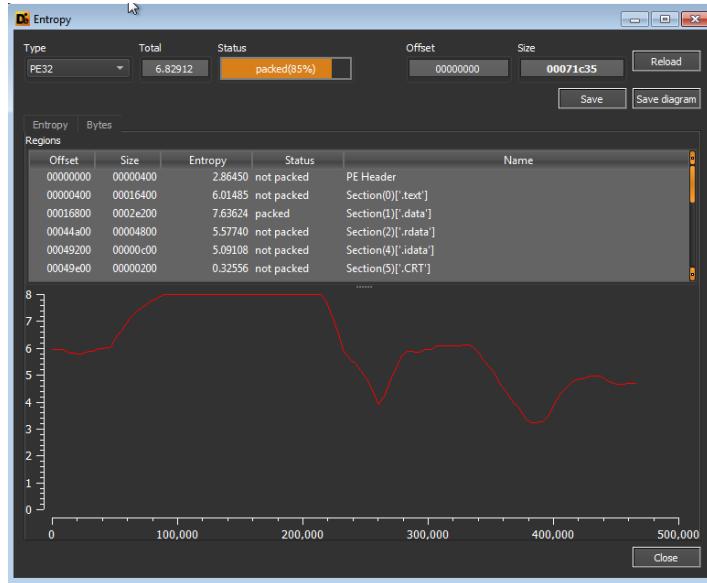
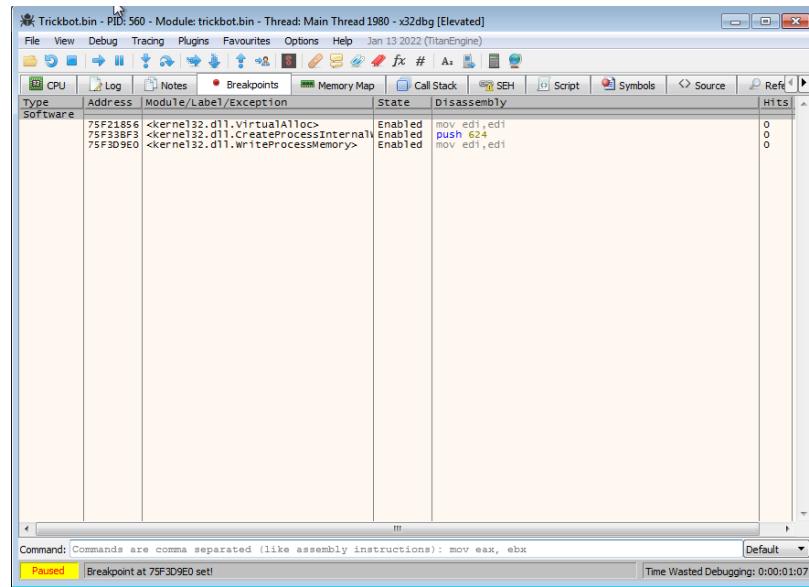


Initially load the sample in Detect it Easy in that we will open the entropy tab and here it is showing the status as packed.



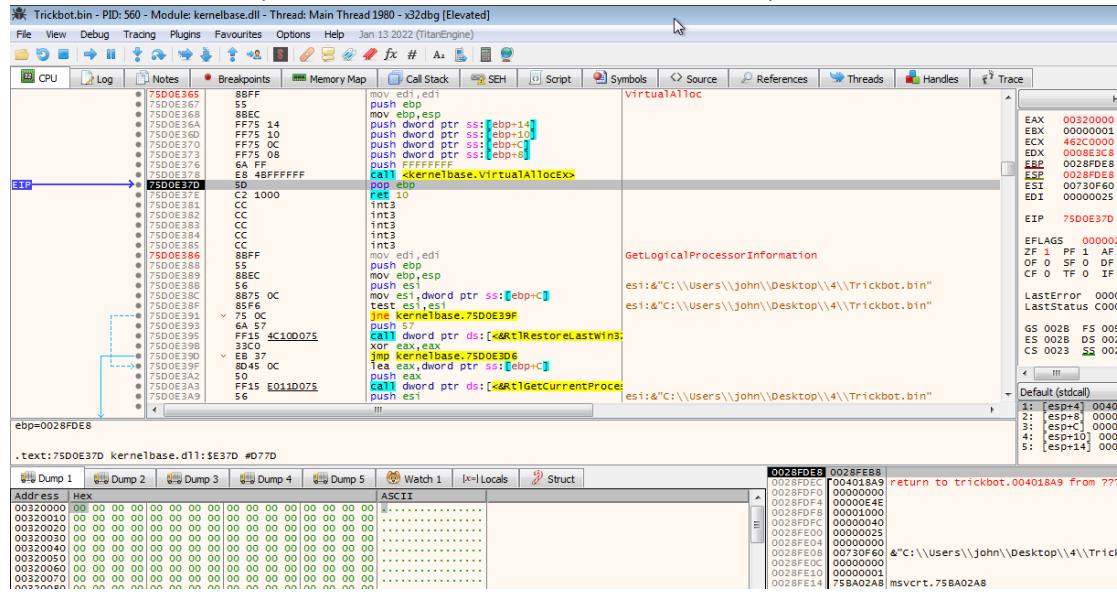
Now we will load the sample in x32dbg, and put breakpoint on VirtualAlloc, CreateProcessInternalW (when malware tries attach to newly created process or it can also create a new process by itself to run certain function), WriteProcessMemory (when malware tries to hijack an existing process by copying part of its code into that process memory in order to be stealthy).



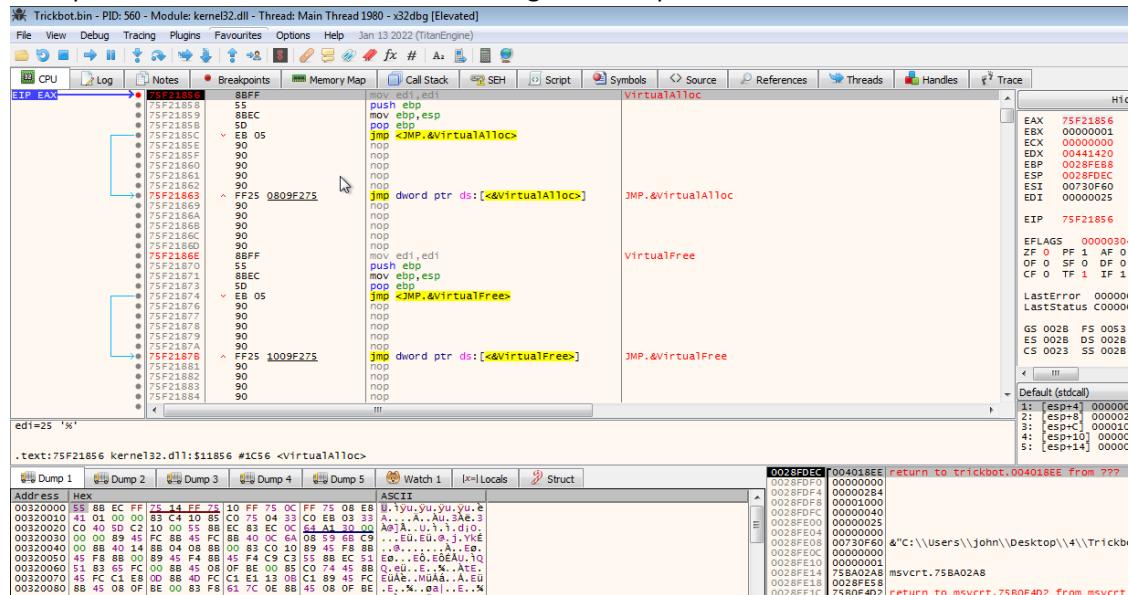
Press F9 and we will hit our first breakpoint at VirtualAlloc, it will return the allocated region address into eax register.

Here inside VirtualAlloc we will step over the kernelbase.VirtualAlloc and will note the address return in eax.

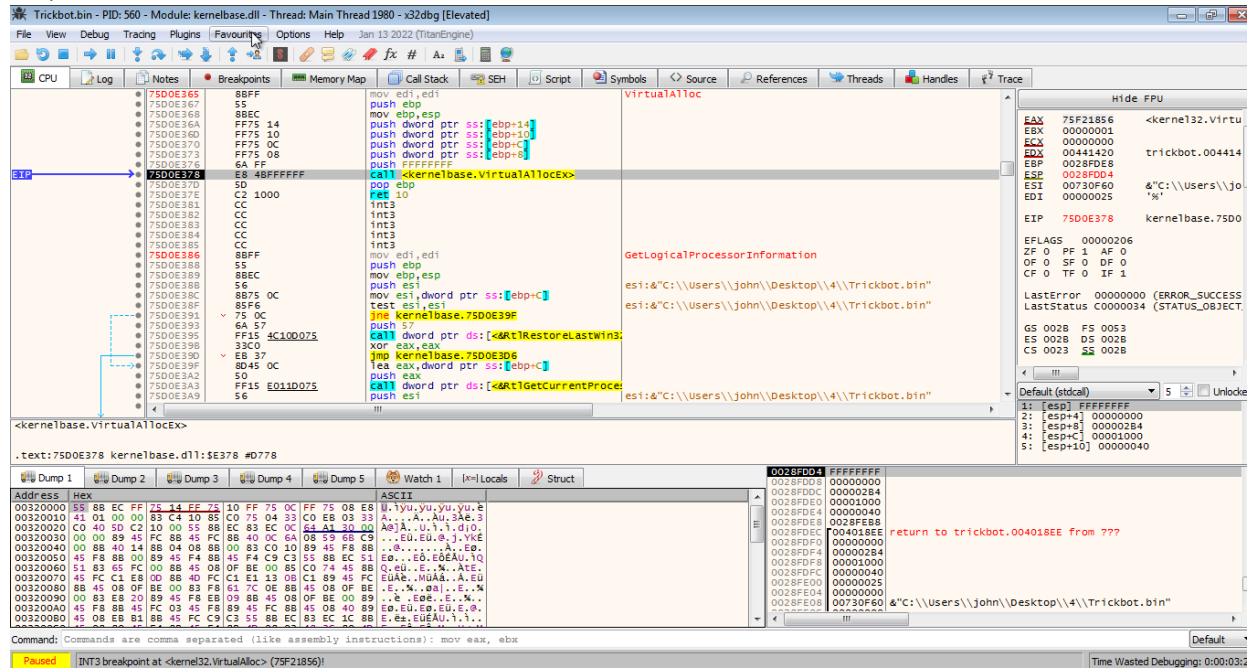
The address return by eax is 320000 follow this location in dump.



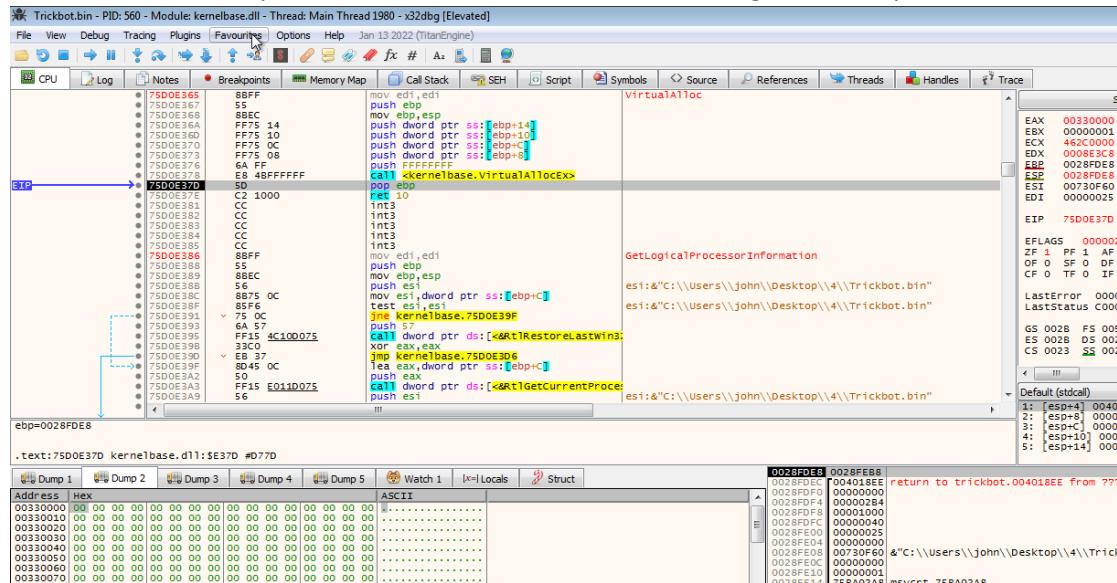
Now press f9 and here we can see the changes in dump also we hit the VirtualAlloc second time.



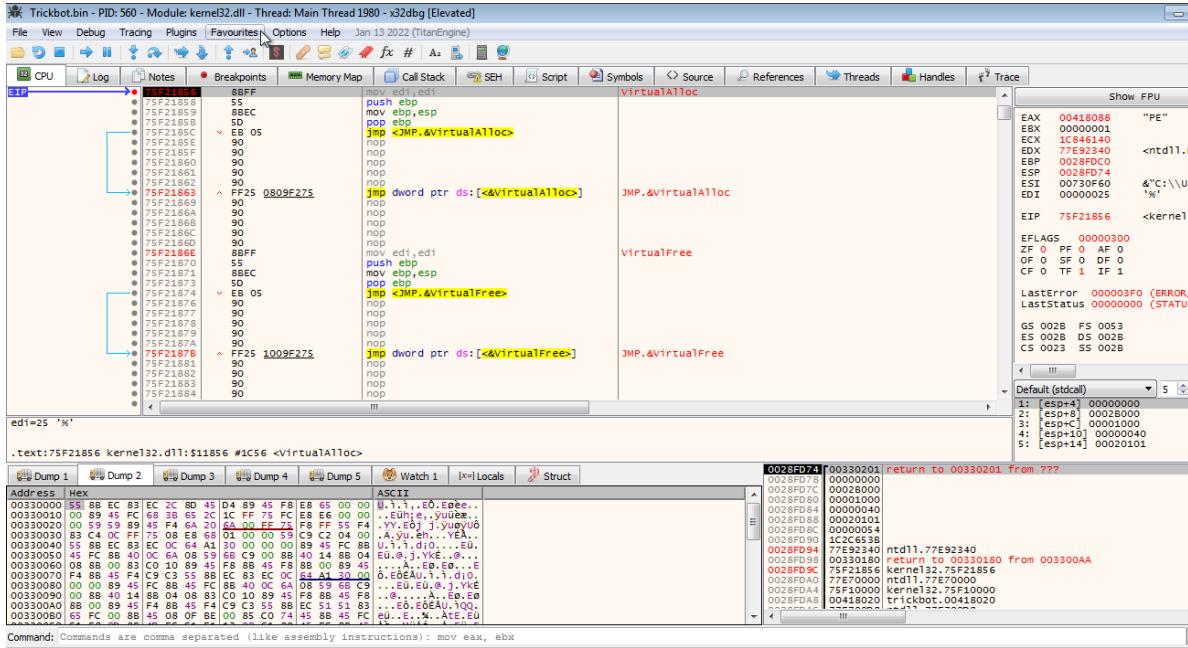
Step over it .



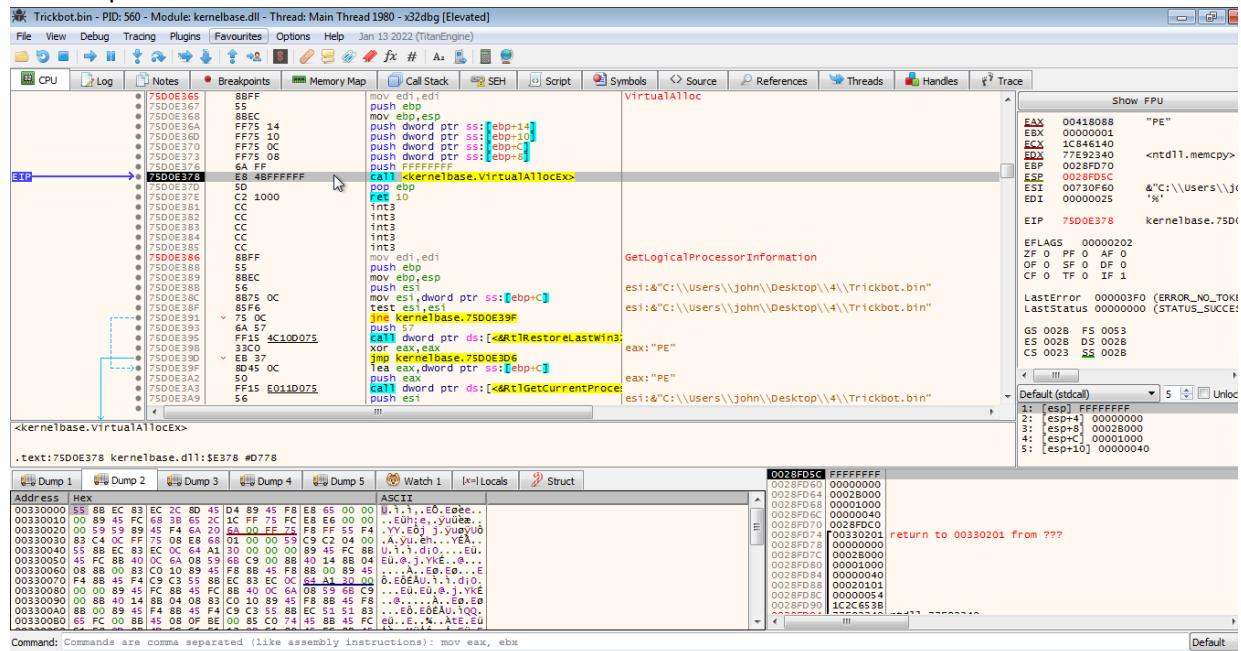
The address return by this VirtualAlloc is 330000 follow this region in Dump 2.



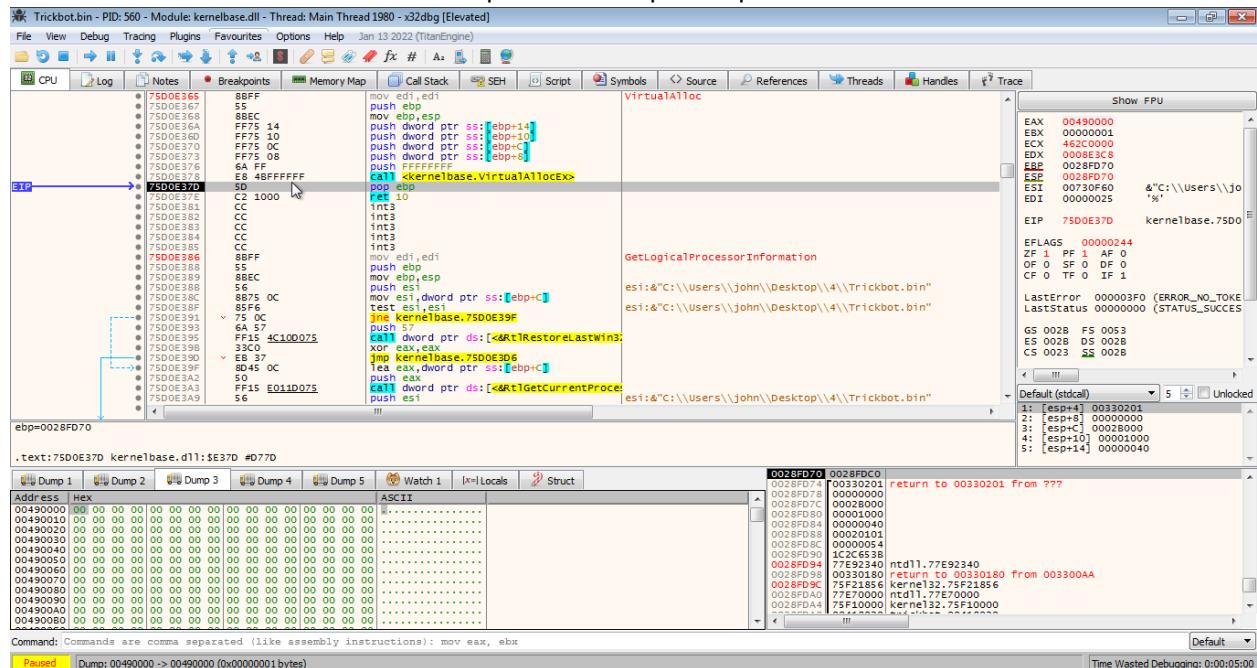
Now again presss f9 here we can see the changes in dump 2 unpacked shellcode and we will hit VirtualAlloc third time.



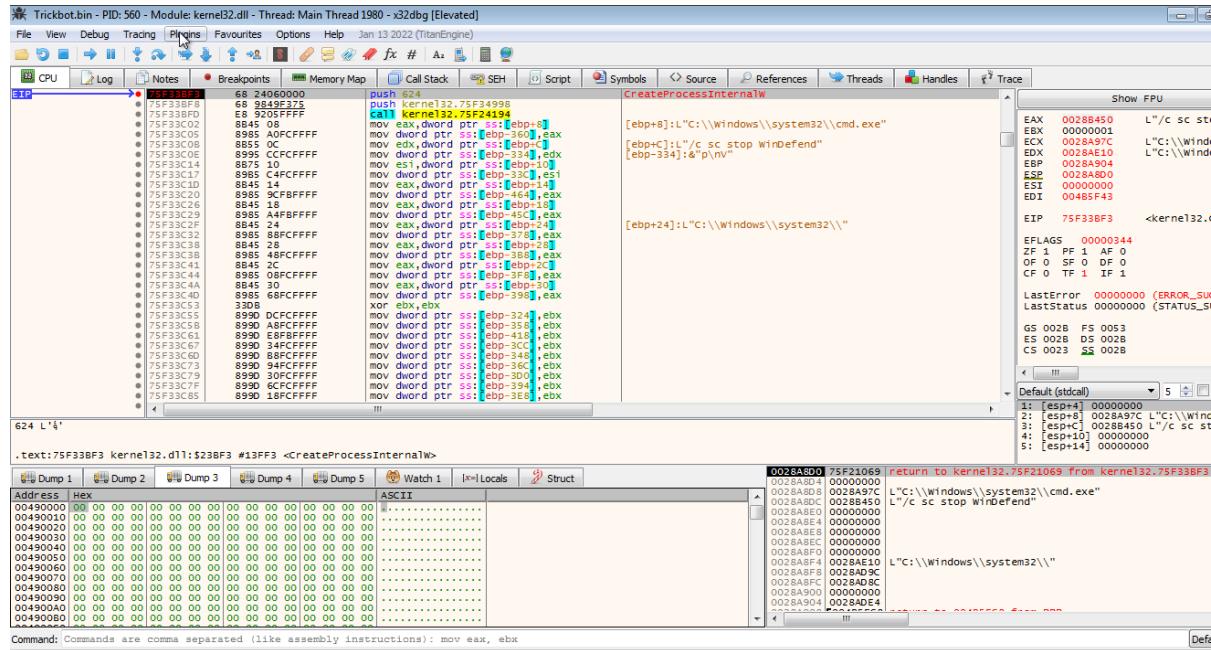
Now step over this.



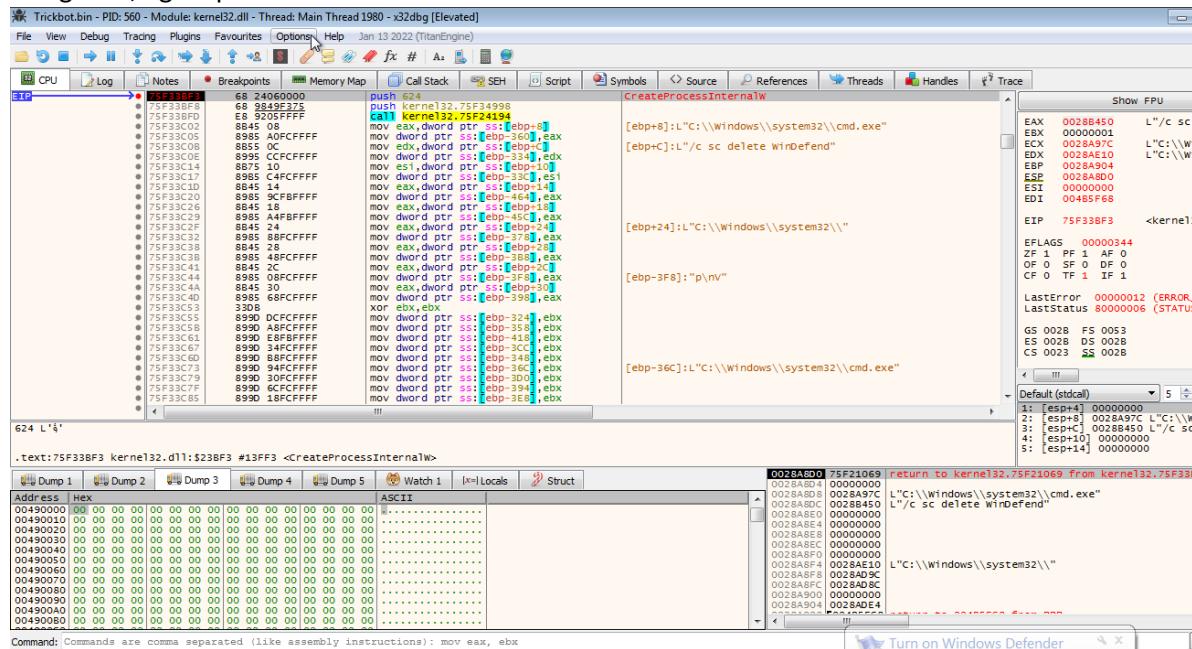
Note the address return in eax and dump that in Dump3 and press F9.



Now we hit another breakpoint CreateProcessInternalW it is using this API to run cmd command in order to stop Window Defender, press f9 to run.



Now here we again hit CreateProcessInternalW now here it is trying to delete window defender service usnig cmd, again press F9.



Now here we again hit CreateProcessInternalW now here it is trying to disable real time monitoring using powershell command, again press F9

```

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
CreateProcessInternalW
File View Debug Tracing Plugins Favourites Options Help Jan 13 2022 [TitanEngine]

0x75F33BF3 68 24060000 push 624
0x75F33BF3 68 24060000 push kernel32.75F34998
0x75F33BF3 E8 9205FFFF call kernel32.75F24194
0x8845 08 mov eax,dword ptr ss:[ebp+8]
0x8845 08 mov dword ptr ss:[ebp-30],eax
0x8845 0C mov edx,dword ptr ss:[ebp-28],eax
0x8845 0C mov dword ptr ss:[ebp-34],edx
0x8845 10 mov esi,dword ptr ss:[ebp-32],eax
0x8845 10 mov dword ptr ss:[ebp-36],esi
0x8845 14 mov eax,dword ptr ss:[ebp-46],eax
0x8845 14 mov dword ptr ss:[ebp-46],eax
0x75F33C26 8845 18 mov eax,dword ptr ss:[ebp-18]
0x75F33C26 8845 18 mov dword ptr ss:[ebp-32],eax
0x75F33C26 8845 24 mov eax,dword ptr ss:[ebp-24]
0x75F33C26 8845 24 mov dword ptr ss:[ebp-37],eax
0x75F33C32 8845 8BFCFFFF mov eax,dword ptr ss:[ebp-35],eax
0x75F33C32 8845 8BFCFFFF mov dword ptr ss:[ebp-35],eax
0x75F33C38 8845 48FCFFFF mov eax,dword ptr ss:[ebp-31],eax
0x75F33C38 8845 48FCFFFF mov dword ptr ss:[ebp-31],eax
0x75F33C41 8845 2C mov eax,dword ptr ss:[ebp-2C]
0x75F33C41 8845 2C mov dword ptr ss:[ebp-39],eax
0x75F33C44 8845 08FCFFF9 mov eax,dword ptr ss:[ebp-38]
0x75F33C44 8845 08FCFFF9 mov dword ptr ss:[ebp-38]
0x75F33C4D 8845 68FCFFFF mov eax,dword ptr ss:[ebp-39],eax
0x75F33C4D 8845 68FCFFFF xor ebx,ebx
0x75F33C53 33DB mov eax,dword ptr ss:[ebp-32],ebx
0x75F33C53 8845 DFCFFFFF mov eax,dword ptr ss:[ebp-32],ebx
0x75F33C58 8990 A8FCFFFFF mov dword ptr ss:[ebp-35],ebx
0x75F33C61 8990 E8BFCFFFFF mov dword ptr ss:[ebp-418],ebx
0x75F33C67 8990 34FCFFFFF mov dword ptr ss:[ebp-3C],ebx
0x75F33C70 8990 94FCFFFFF mov dword ptr ss:[ebp-3D],ebx
0x75F33C73 8990 94FCFFFFF mov dword ptr ss:[ebp-3E],ebx
0x75F33C79 8990 30FCFFFFF mov dword ptr ss:[ebp-30],ebx
0x75F33C7F 8990 B8FCFFFFF mov dword ptr ss:[ebp-39],ebx
0x75F33C85 8990 18FCFFFFF mov dword ptr ss:[ebp-3E],ebx
0x75F33C85 8990 18FCFFFFF

```

624 L'4'

.text:75F33BF3 kernel32.dll:\$#23BF3 #13FF3 <CreateProcessInternalW>

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct

Address	Hex	ASCII
00490000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00490010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00490020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00490040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00490050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00490070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00490080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004900A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004900B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0028A600 75F21069 return to kernel32.75F21069 from kernel32.75F33BF3

EAX 0028A600 00000000 EBX 00000001 ECX 0028A770 L"C:\windows\system32\cmd.exe" EDX 0028A890 L"C:\windows\system32\powershell Set-MpPreference -DisableRealtimeMonitoring [ebp+34]:l:"c:\powershell Set-MpPreference -DisableRealtimeMonitoring [ebp-34]:l:"en-US" EDI 00485FE4 EIP 75F33BF3 <kernel32.C

EFLAGS 00000344 ZF 1 PF 1 AF 0 OF 0 SF 0 DF 0 CF 0 TF 1 IF 1 LastError 00000012 (ERROR_NOACCESS) LastStatus 80000006 (STATUS_NOMEMORY) GS 002B FS 0053 ES 002B DS 002B CS 0023 SS 002B

Default (stdcall) 5

1: [esp+4] 00000000 2: [esp+8] 00000000 L"C:\windows\system32\powershell Set-MpPreference -DisableRealtimeMonitoring [ebp+34]:l:"c:\powershell Set-MpPreference -DisableRealtimeMonitoring [ebp-34]:l:"en-US" 3: [esp+12] 00000000 4: [esp+16] 00000000 5: [esp+20] 00000000

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Now here we again hit CreateProcessInternalW now here it is trying to drop Usjclbpt.exe, again press F9

```

CPU Log Notes Breakpoints Patches Map Call Stack SEH Script Symbols Source References Threads Handles Trace
CreateProcessInternalW
File View Debug Tracing Plugins Favourites Options Help Jan 13 2022 [TitanEngine]

0x75F33BF3 68 24060000 push 624
0x75F33BF3 68 24060000 push kernel32.75F34998
0x75F33BF3 E8 9205FFFF call kernel32.75F24194
0x8845 08 mov eax,dword ptr ss:[ebp+8]
0x8845 08 mov dword ptr ss:[ebp-30],eax
0x8845 0C mov edx,dword ptr ss:[ebp-28],eax
0x8845 0C mov dword ptr ss:[ebp-34],edx
0x8845 10 mov esi,dword ptr ss:[ebp-32],eax
0x8845 10 mov dword ptr ss:[ebp-36],esi
0x8845 14 mov eax,dword ptr ss:[ebp-46],eax
0x8845 14 mov dword ptr ss:[ebp-46],eax
0x75F33C26 8845 18 mov eax,dword ptr ss:[ebp-18]
0x75F33C26 8845 18 mov dword ptr ss:[ebp-32],eax
0x75F33C26 8845 24 mov eax,dword ptr ss:[ebp-24]
0x75F33C26 8845 24 mov dword ptr ss:[ebp-37],eax
0x75F33C32 8845 8BFCFFFF mov eax,dword ptr ss:[ebp-35],eax
0x75F33C32 8845 8BFCFFFF mov dword ptr ss:[ebp-35],eax
0x75F33C38 8845 48FCFFFF mov eax,dword ptr ss:[ebp-31],eax
0x75F33C38 8845 48FCFFFF mov dword ptr ss:[ebp-31],eax
0x75F33C41 8845 2C mov eax,dword ptr ss:[ebp-2C]
0x75F33C41 8845 2C mov dword ptr ss:[ebp-39],eax
0x75F33C44 8845 30 mov eax,dword ptr ss:[ebp-30]
0x75F33C44 8845 30 mov dword ptr ss:[ebp-30]
0x75F33C4D 8845 68FCFFFF mov eax,dword ptr ss:[ebp-39],eax
0x75F33C4D 8845 68FCFFFF xor ebx,ebx
0x75F33C53 33DB mov eax,dword ptr ss:[ebp-32],ebx
0x75F33C53 8845 DFCFFFFF mov eax,dword ptr ss:[ebp-32],ebx
0x75F33C58 8990 A8FCFFFFF mov dword ptr ss:[ebp-35],ebx
0x75F33C61 8990 E8BFCFFFFF mov dword ptr ss:[ebp-418],ebx
0x75F33C67 8990 34FCFFFFF mov dword ptr ss:[ebp-3C],ebx
0x75F33C70 8990 94FCFFFFF mov dword ptr ss:[ebp-3D],ebx
0x75F33C73 8990 94FCFFFFF mov dword ptr ss:[ebp-3E],ebx
0x75F33C79 8990 30FCFFFFF mov dword ptr ss:[ebp-30],ebx
0x75F33C7F 8990 B8FCFFFFF mov dword ptr ss:[ebp-39],ebx
0x75F33C85 8990 18FCFFFFF mov dword ptr ss:[ebp-3E],ebx
0x75F33C85 8990 18FCFFFFF

```

624 L'4'

.text:75F33BF3 kernel32.dll:\$#23BF3 #13FF3 <CreateProcessInternalW>

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct

Address	Hex	ASCII
00490000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00490010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00490020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00490040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00490050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00490070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00490080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004900A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004900B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0028A6F0 75F21069 return to kernel32.75F21069 from kernel32.75F33BF3

EAX 0028A6F0 00000000 EBX 00000001 ECX 0028A770 L"C:\Users\john\AppData\Roaming\wnetwork\usjclbpt.exe" EDX 0028A890 L"C:\Users\john\AppData\Roaming\wnetwork\usjclbpt.exe" EDI 0048520E EIP 75F33BF3 <kernel32.C

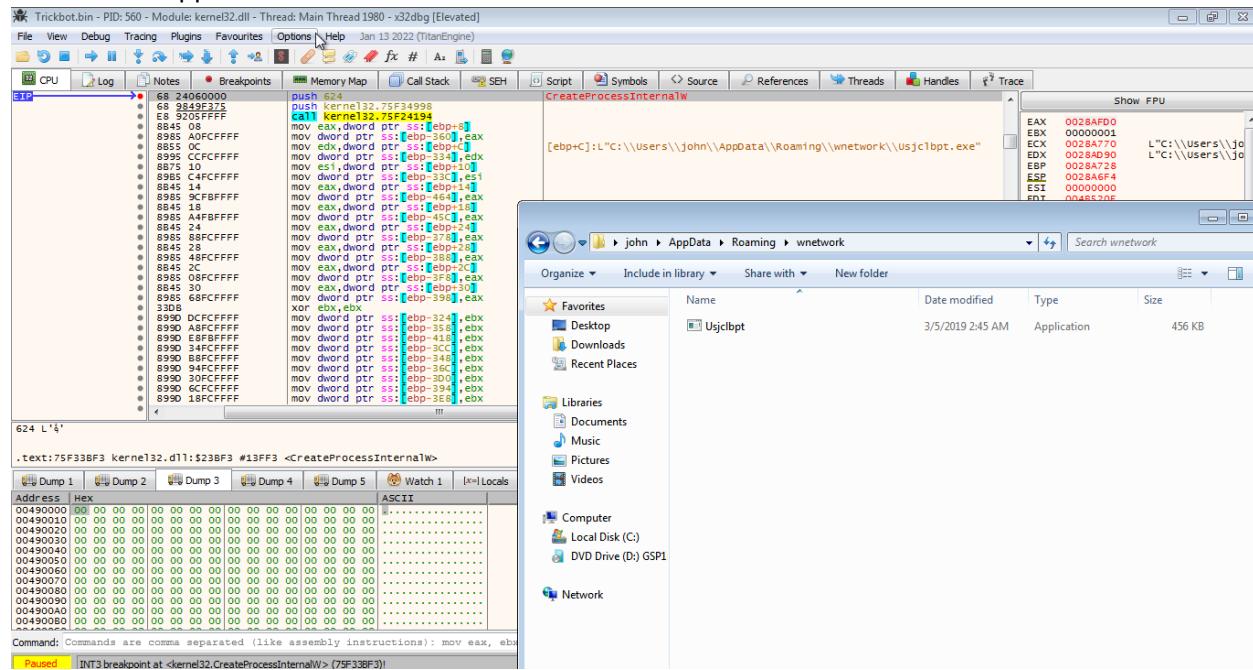
EFLAGS 00000344 ZF 1 PF 1 AF 0 OF 0 SF 0 DF 0 CF 0 TF 1 IF 1 LastError 00000000 (ERROR_SUCCESS) LastStatus 80000000 (STATUS_SUCCESS) GS 002B FS 0053 ES 002B DS 002B CS 0023 SS 002B

Default (stdcall) 5

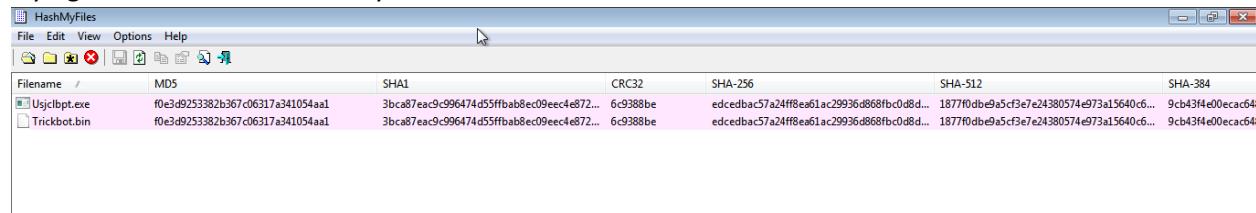
1: [esp+4] 00000000 2: [esp+8] 00000000 L"C:\Users\john\AppData\Roaming\wnetwork\usjclbpt.exe" 3: [esp+12] 00000000 4: [esp+16] 00000000 5: [esp+20] 00000000

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

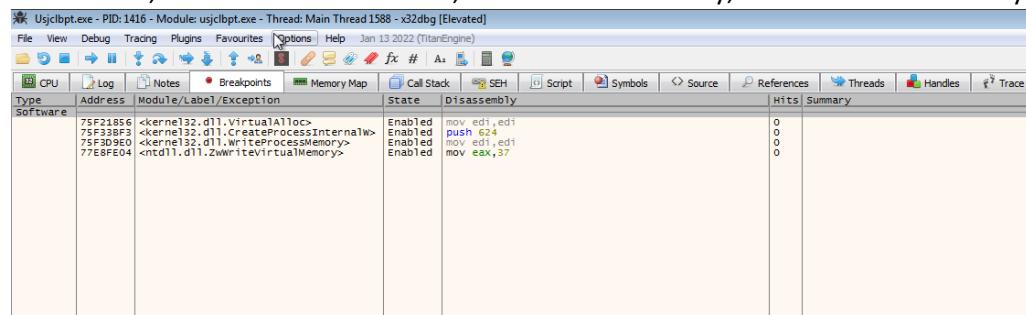
This is the dropped file.



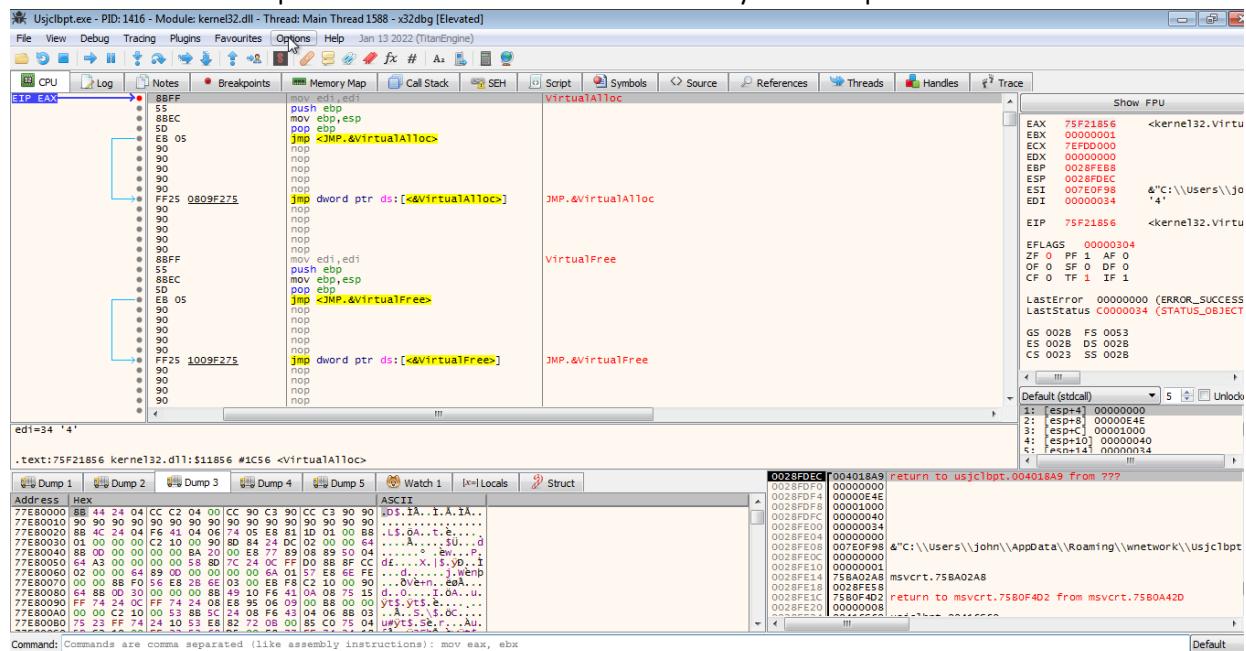
Now compare the hash of this file with the trickbot hash using HashMyFiles. Both file has same hash, trying to become more stealthy.



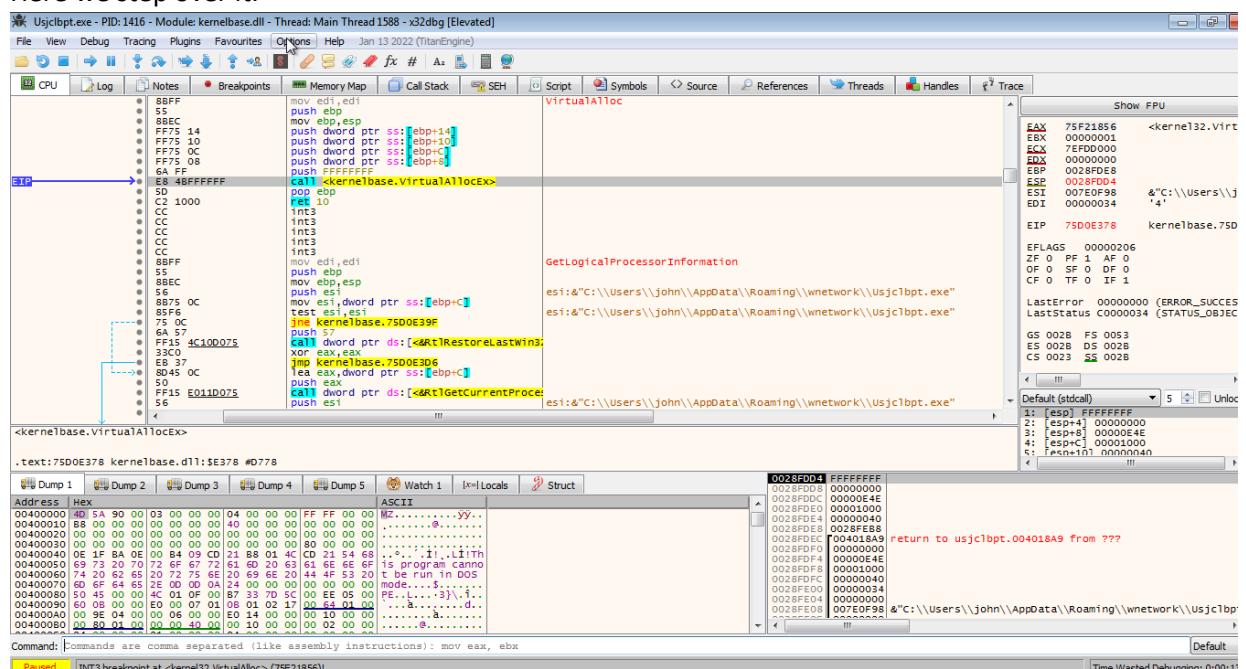
Now stop the ongoing x32dbg and load the newly dropped trickbot in x32dbg and apply the breakpoints VirtualAlloc, CreateProcessInternalW, WriteProcessMemory, ZwWriteVirtualMemory run it.



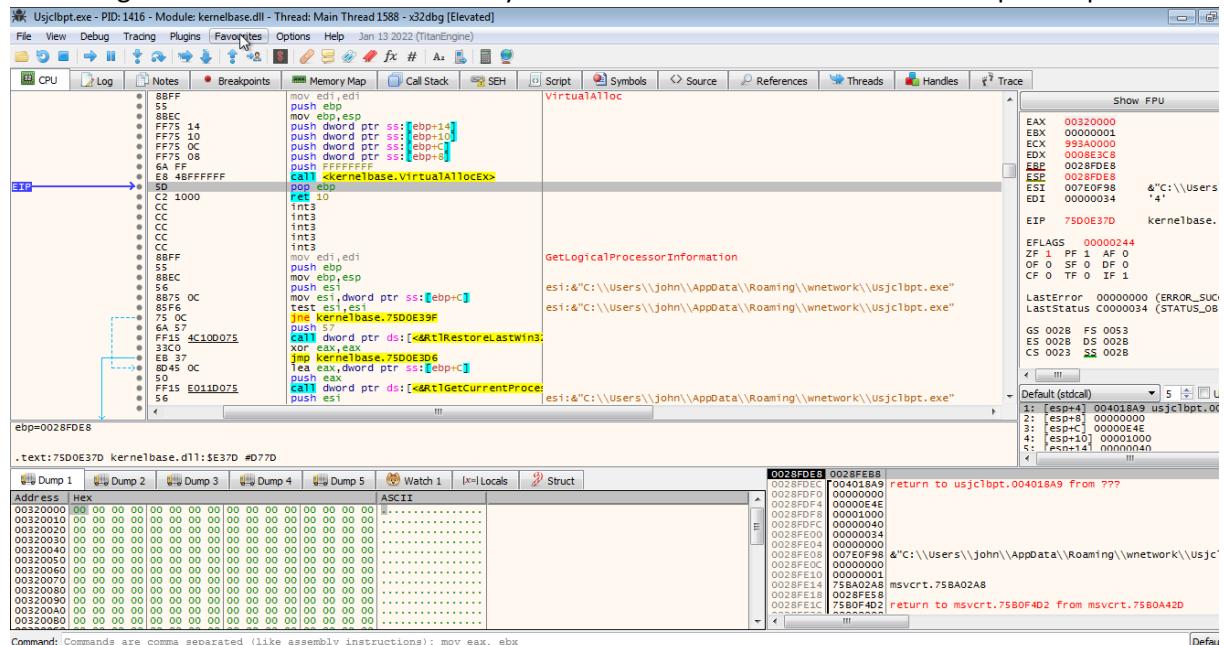
We hit VirtualAlloc step over it and load the address return by it in dump1.



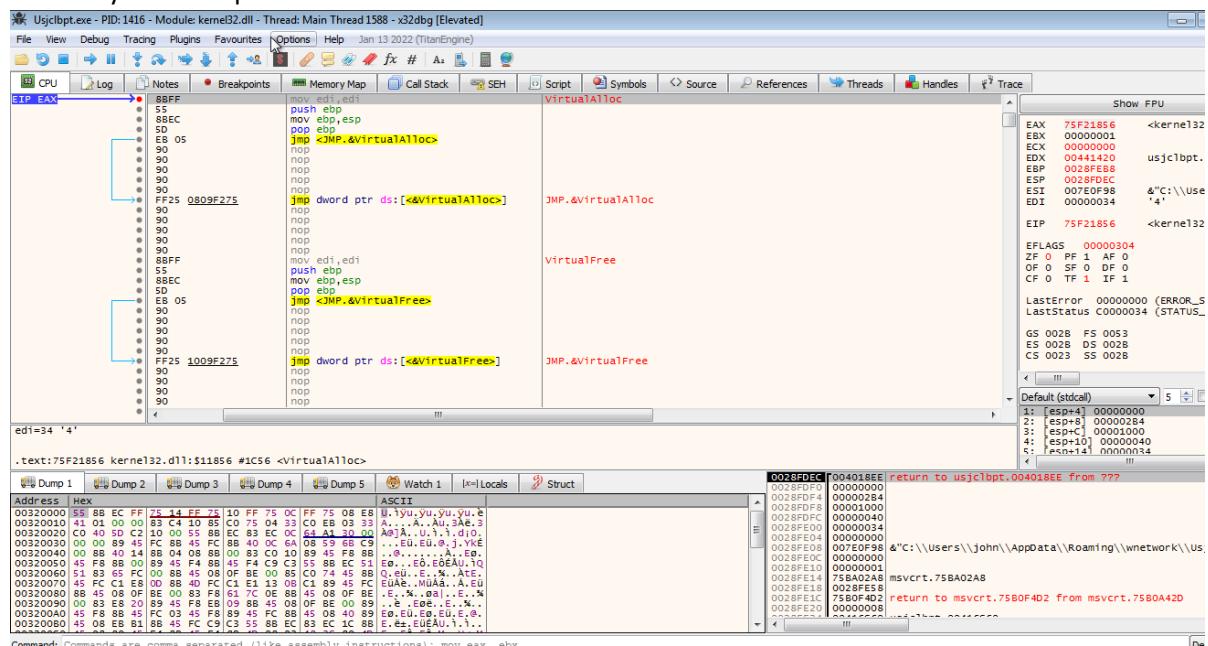
Here we step over it.



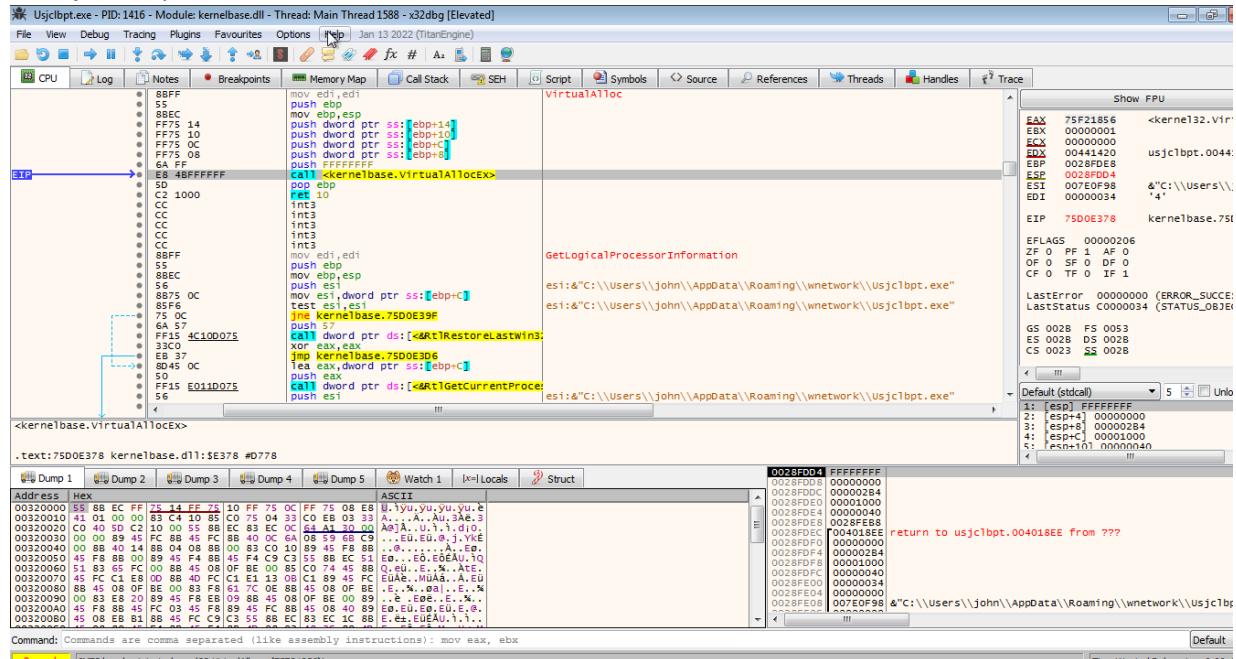
Here we get the return address 320000 by VirtualAlloc i.e in eax and load it in dump1 and press f9.



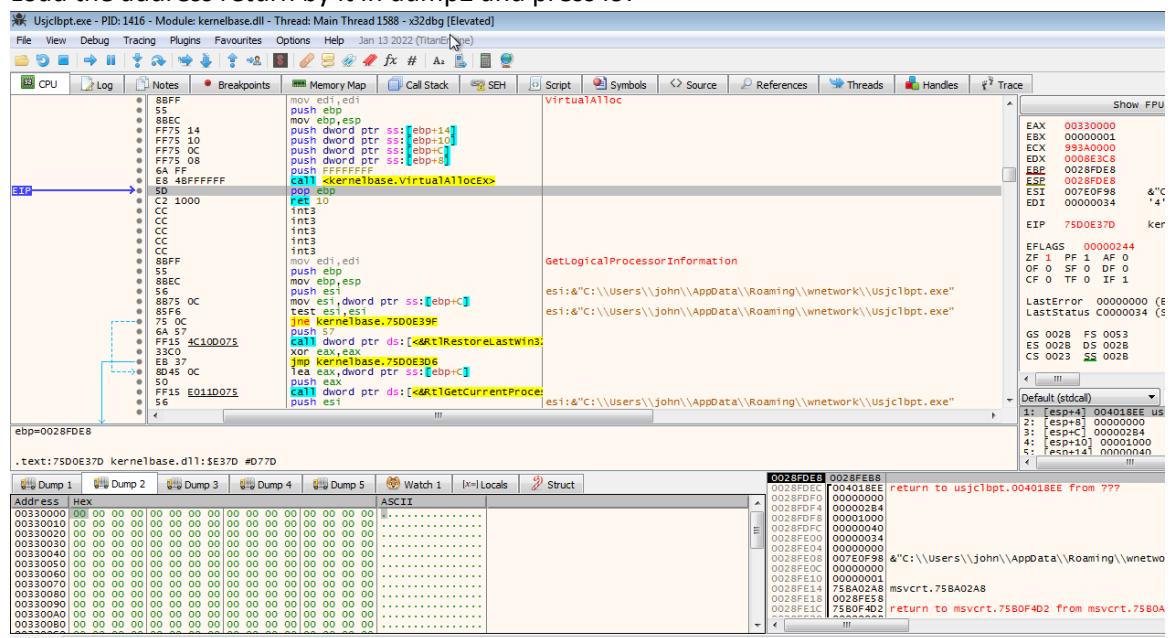
We hit VirtualAlloc step over it and also the dump1 has been filled by shellcode and load the address return by it in dump2.



Here just step over it.



Load the address return by it in dump2 and press f9.



We hit VirtualAlloc step over it and also the dump2 has been filled by shellcode and load the address return by it in dump3.

```

CPU Log Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
EIP 75F21856 VirtualAlloc
00330000 55 8B EC 83 EC 2C 8D 45 D4 89 45 F8 E8 65 00 00 U...1.,.E0.E0... .
00330010 00 89 45 FC 68 3B 65 2C 1C FF 75 FC E8 E6 00 00 .E.Uhie..yuue...
00330020 00 59 45 FC 6A 08 45 F8 E8 65 00 00 .A.Yhie..yuue...
00330030 00 59 45 FC 6A 08 45 F8 E8 65 00 00 .A.Yhie..yuue...
00330040 55 88 EC 83 EC 0C 64 A1 30 00 00 89 45 FC 88 U...1..1.j0... .EU...
00330050 45 FC 88 40 EC 64 A1 30 00 00 89 45 FC 88 E0.0...j.YKE... .@...
00330060 45 FC 88 40 EC 64 A1 30 00 00 89 45 FC 88 E0.0...j.YKE... .@...
00330070 F4 88 45 F4 C9 C3 58 88 EC OC 64 A1 30 00 00 89 45 FC 88 E0.0...j.YKE... .@...
00330080 00 89 45 FC 88 40 EC 64 08 45 F8 E8 65 00 00 U...E.UU... .E0...
00330090 00 89 45 FC 88 40 EC 64 08 45 F8 E8 65 00 00 U...E.UU... .E0...
003300A0 00 89 45 FC 88 40 EC 64 08 45 F8 E8 65 00 00 U...E.UU... .E0...
003300B0 65 FC 00 88 45 08 OF BE 00 85 C0 74 45 88 45 FC eu..E..%.ATE.EU

```

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct

Address	Hex	ASCII
0028FD74	00000000	return to 00330201 from ???
0028FD7C	00028000	
0028FD80	00001000	
0028FD84	00000000	
0028FD88	00020101	
0028FD92	00000054	
0028FD94	1C2C653B	
0028FD98	00030180	ntdll.77E92340
0028FD9B	75F21856	return to 00330180 from 003300AA
0028FD9C	75F21856	kernel32.75F21856
0028FD9D	00010000	ntdll.77E92340
0028FDAD	75F10000	kernel32.75F10000
0028FDA0	00418020	usjclbpt.00418020
0028FDAB	00418020	usjclbpt.00418020

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Here just step over it.

```

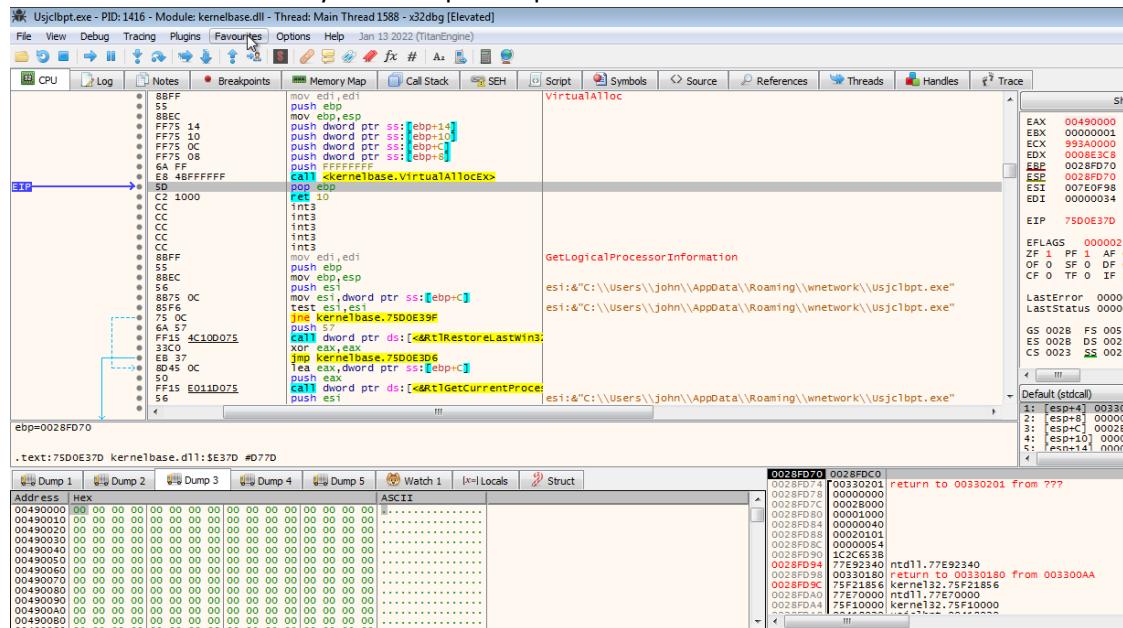
CPU Log Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
EIP 75D0E378 kernelbase.VirtualAllocEx
00330000 55 8B EC 83 EC 2C 8D 45 D4 89 45 F8 E8 65 00 00 U...1.,.E0.E0... .
00330010 00 89 45 FC 68 3B 65 2C 1C FF 75 FC E8 E6 00 00 .E.Uhie..yuue...
00330020 00 59 45 FC 6A 08 45 F8 E8 65 00 00 .A.Yhie..yuue...
00330030 00 59 45 FC 6A 08 45 F8 E8 65 00 00 .A.Yhie..yuue...
00330040 55 88 EC 83 EC 0C 64 A1 30 00 00 89 45 FC 88 U...1..1.j0... .EU...
00330050 45 FC 88 40 EC 64 A1 30 00 00 89 45 FC 88 E0.0...j.YKE... .@...
00330060 45 FC 88 40 EC 64 A1 30 00 00 89 45 FC 88 E0.0...j.YKE... .@...
00330070 F4 88 45 F4 C9 C3 58 88 EC OC 64 A1 30 00 00 89 45 FC 88 E0.0...j.YKE... .@...
00330080 00 89 45 FC 88 40 EC 64 08 45 F8 E8 65 00 00 U...E.UU... .E0...
00330090 00 89 45 FC 88 40 EC 64 08 45 F8 E8 65 00 00 U...E.UU... .E0...
003300A0 00 89 45 FC 88 40 EC 64 08 45 F8 E8 65 00 00 U...E.UU... .E0...
003300B0 65 FC 00 88 45 08 OF BE 00 85 C0 74 45 88 45 FC eu..E..%.ATE.EU

```

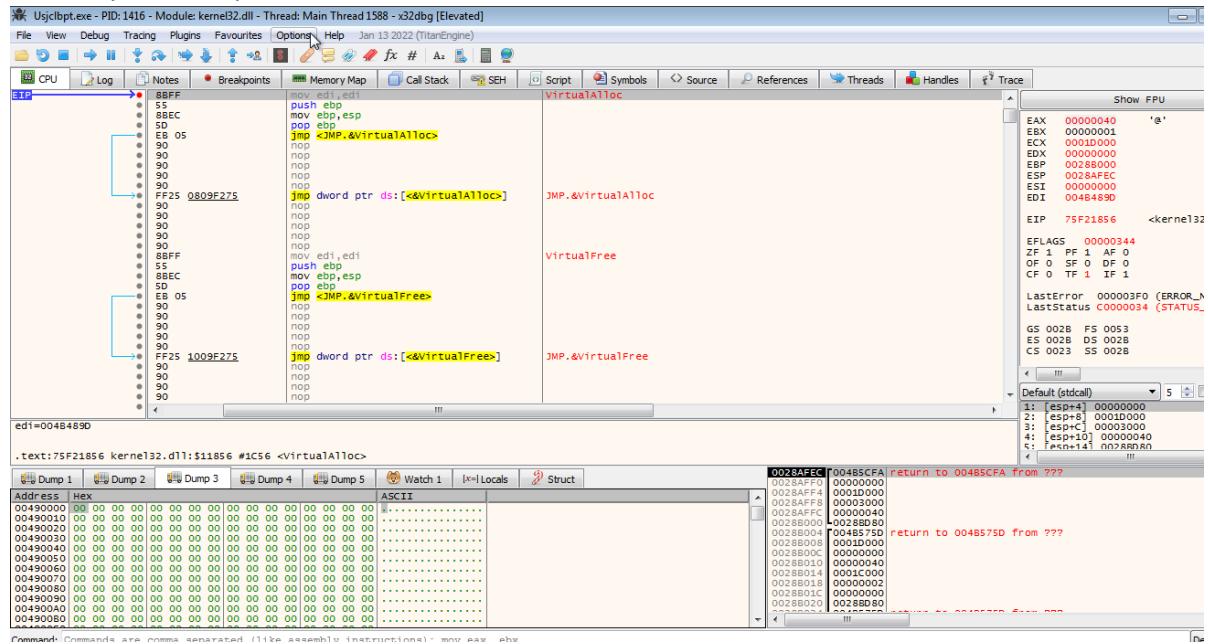
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct

Address	Hex	ASCII
0028FD5C	FFFFFFFFFF	
0028FD60	00000000	
0028FD64	00028000	
0028FD68	00001000	
0028FD70	00000000	
0028FD74	00030201	return to 00330201 from ???
0028FD7C	00028000	
0028FD80	00001000	
0028FD84	00000040	
0028FD88	00000054	
0028FD92	1C2C653B	

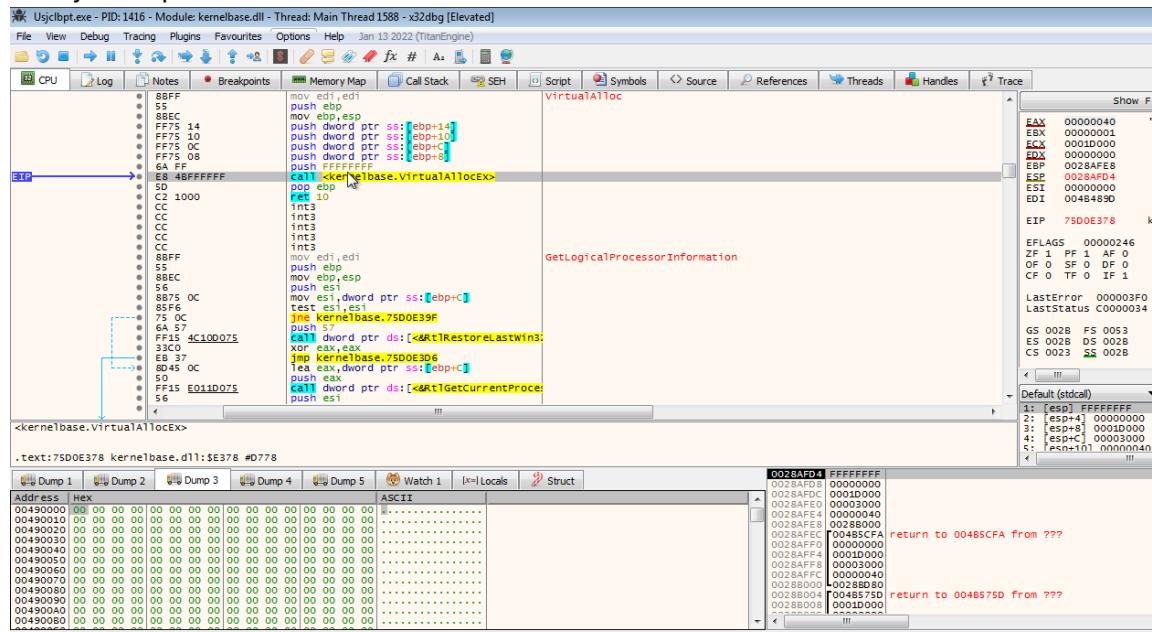
Load the address return by it in dump3 and press f9.



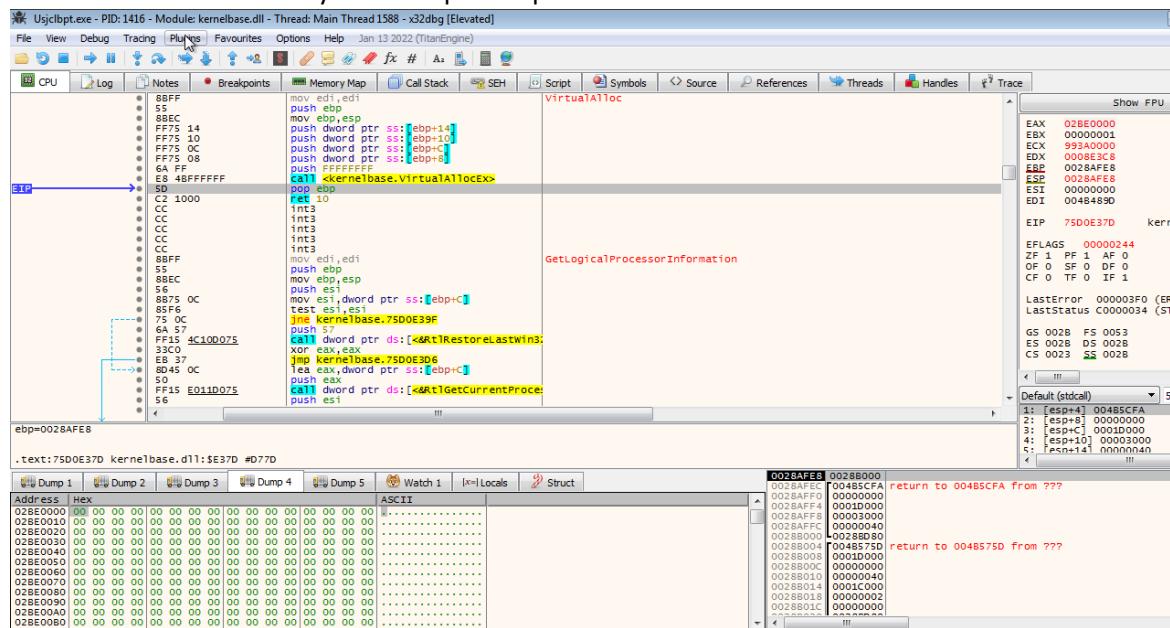
We hit VirtualAlloc step over it and also the dump3 has been filled by shellcode and load the address return by it in dump4.



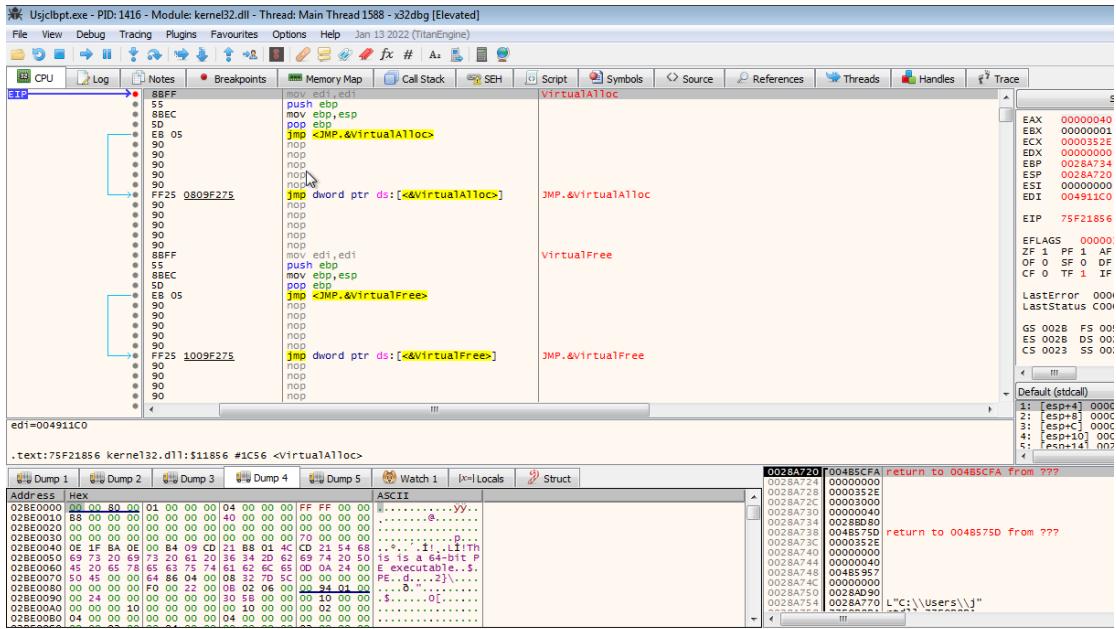
Here just step over it .



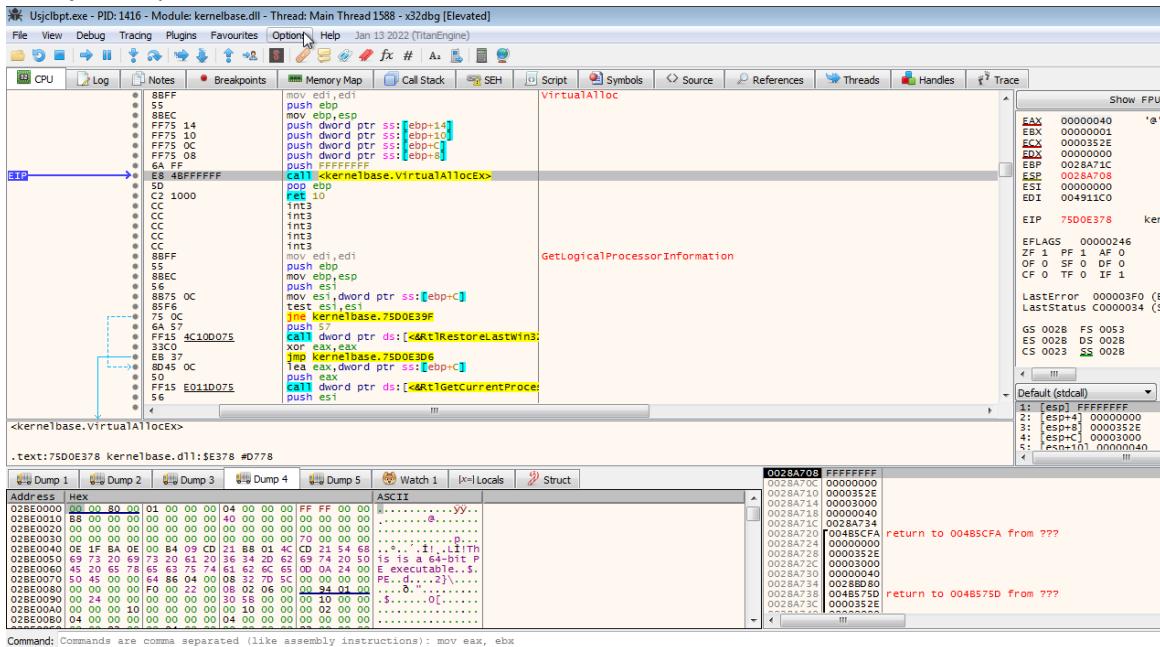
Load the address return by it in dump3 and press f9.



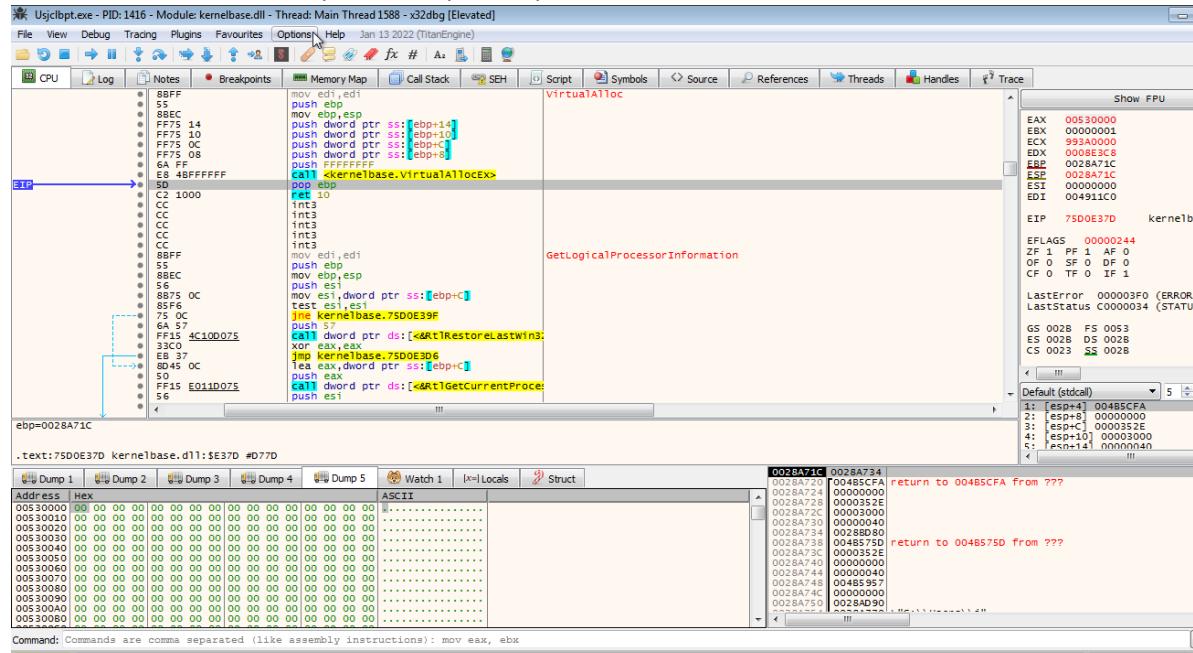
We hit VirtualAlloc step over it and also the dump4 has been filled by shellcode and load the address return by it in dump5.



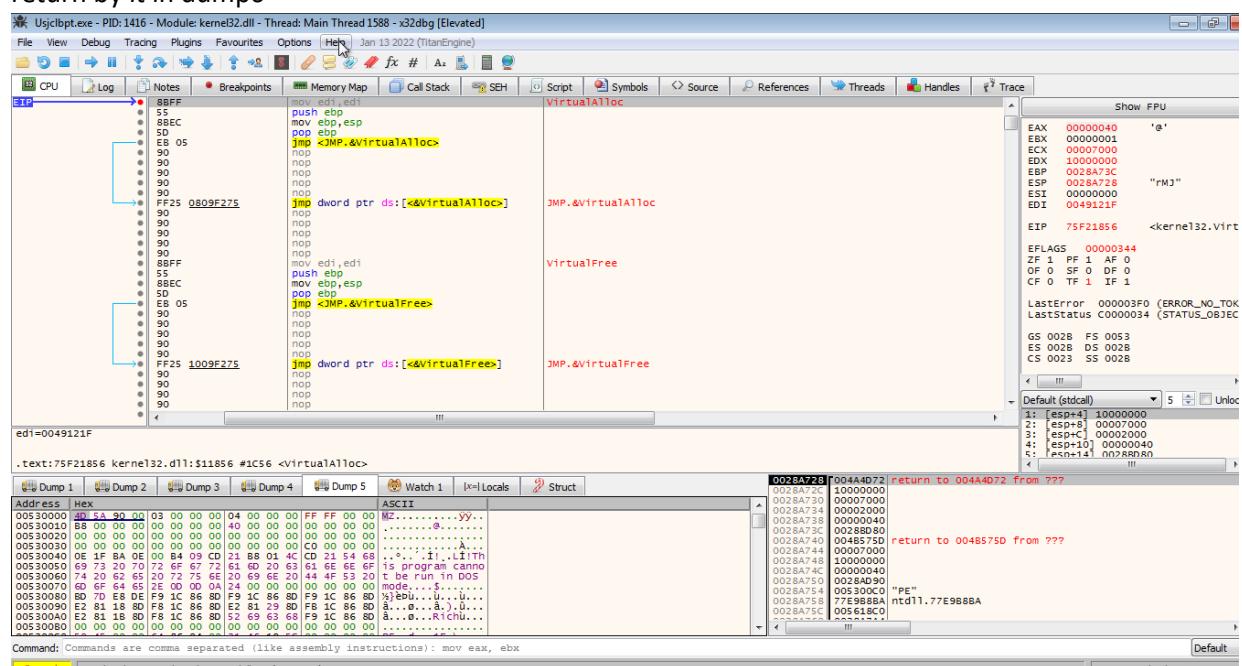
Here just step over it.



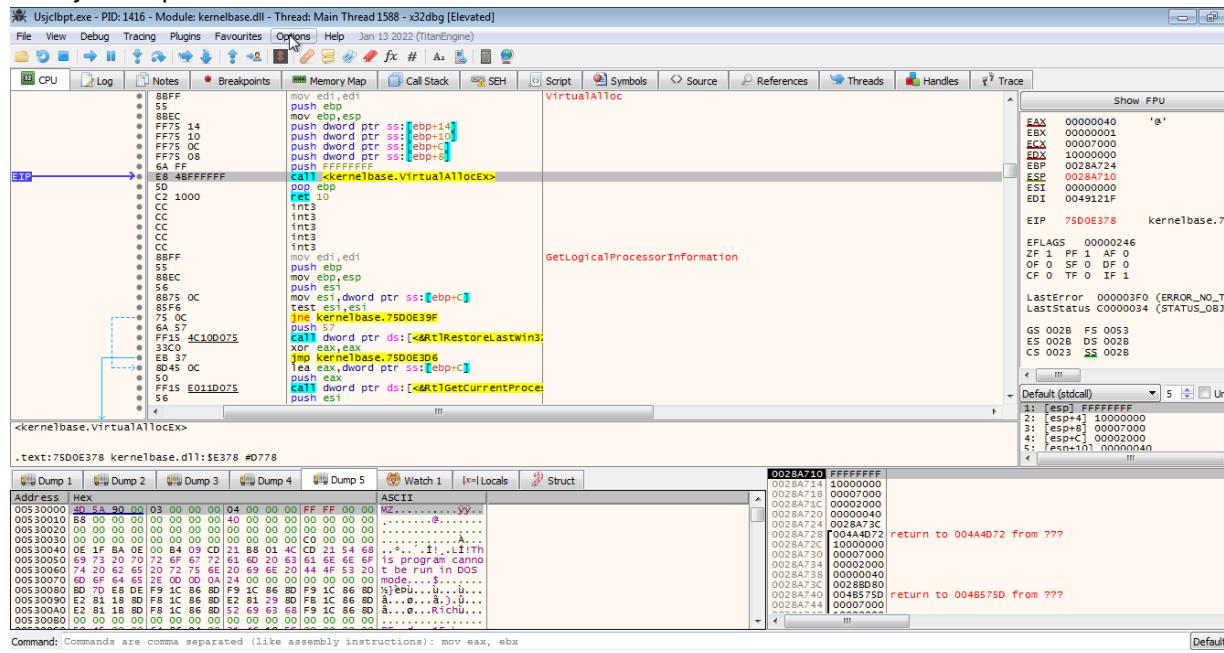
Load the address return by it in dump5 and press f9



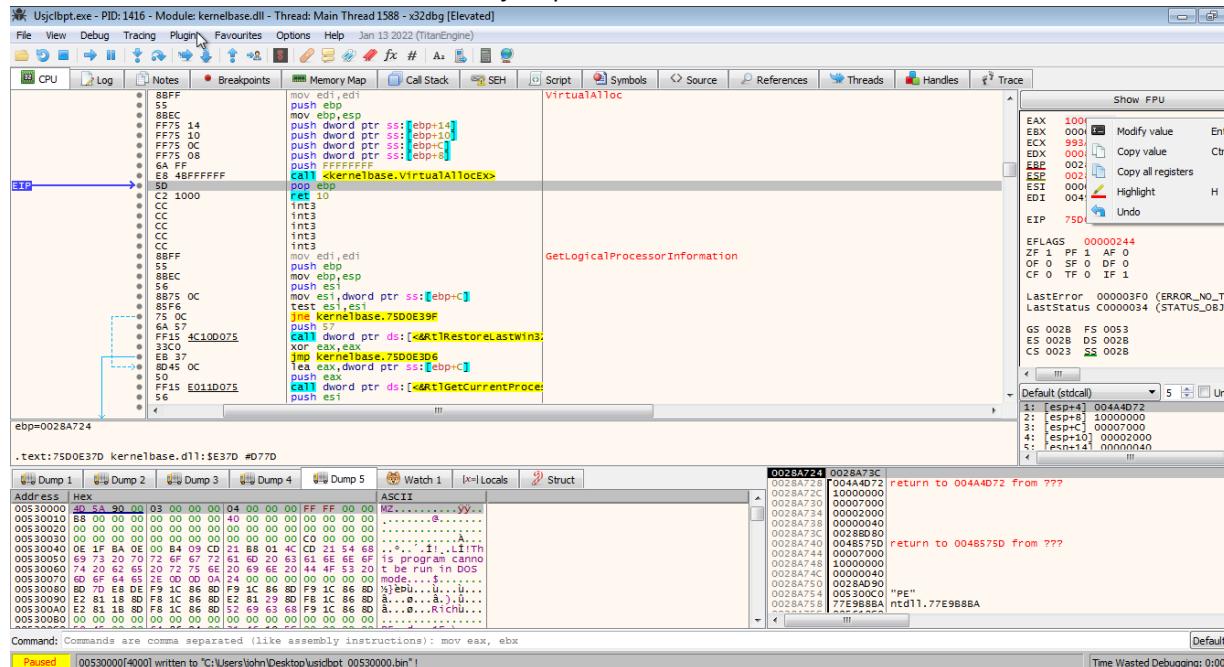
We hit VirtualAlloc step over it and also the dump5 has been filled by shellcode and load the address return by it in dump5



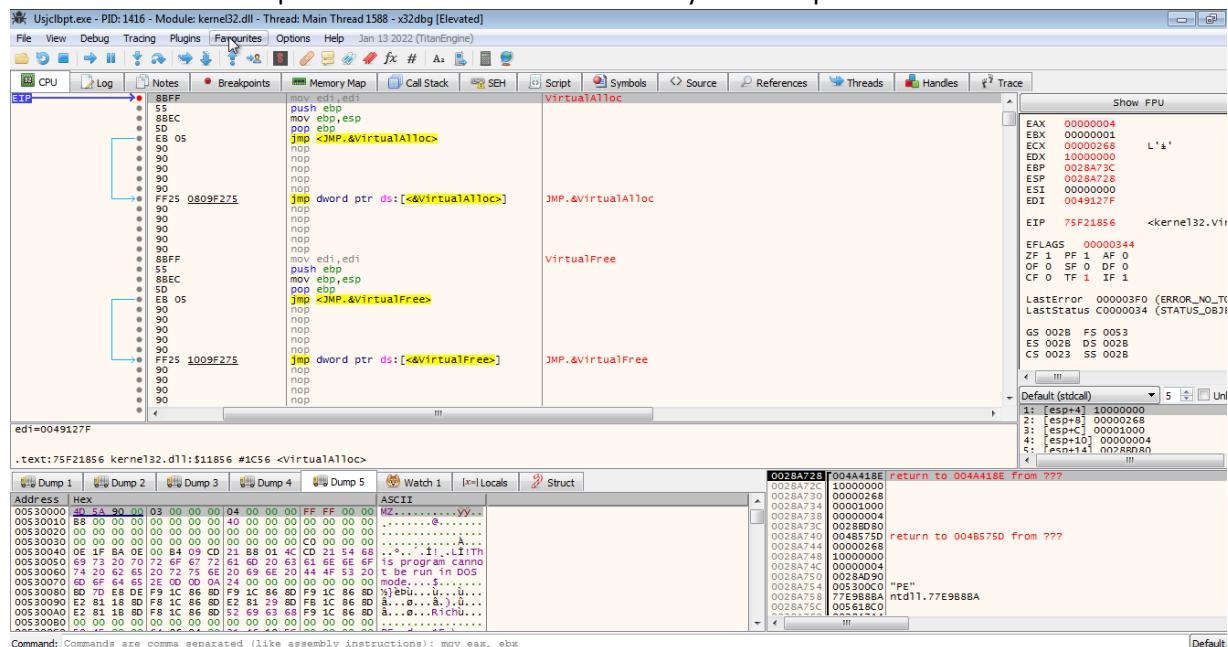
Here just step over it



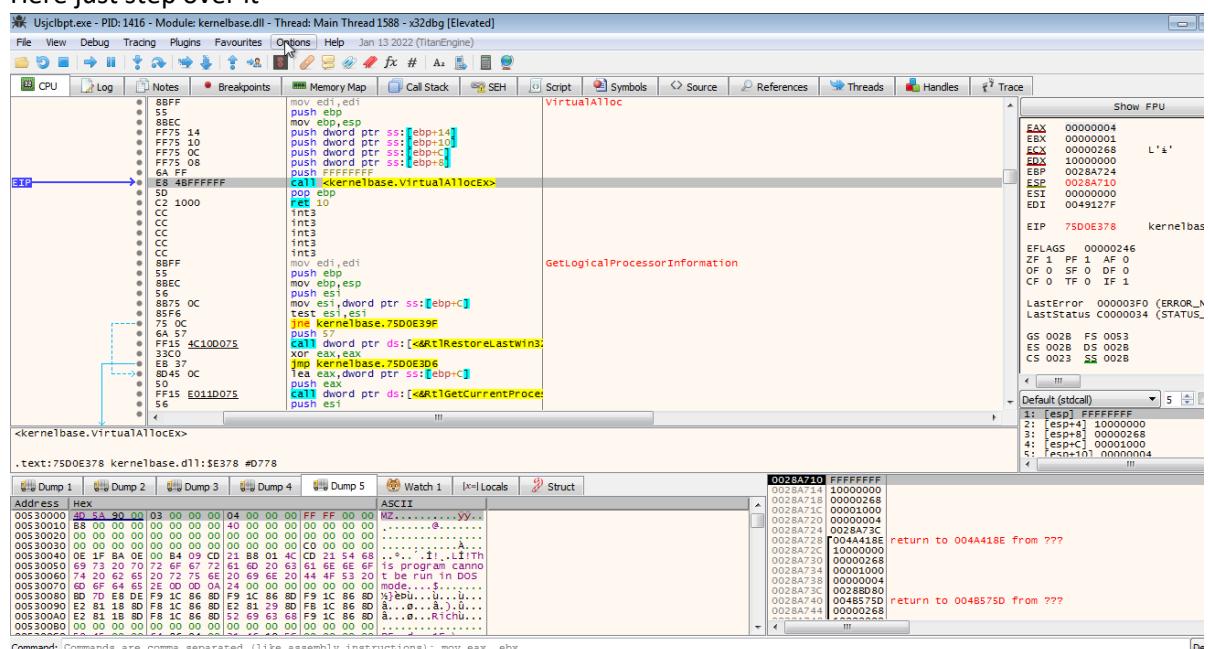
Here we can't load the return address so just press f9.



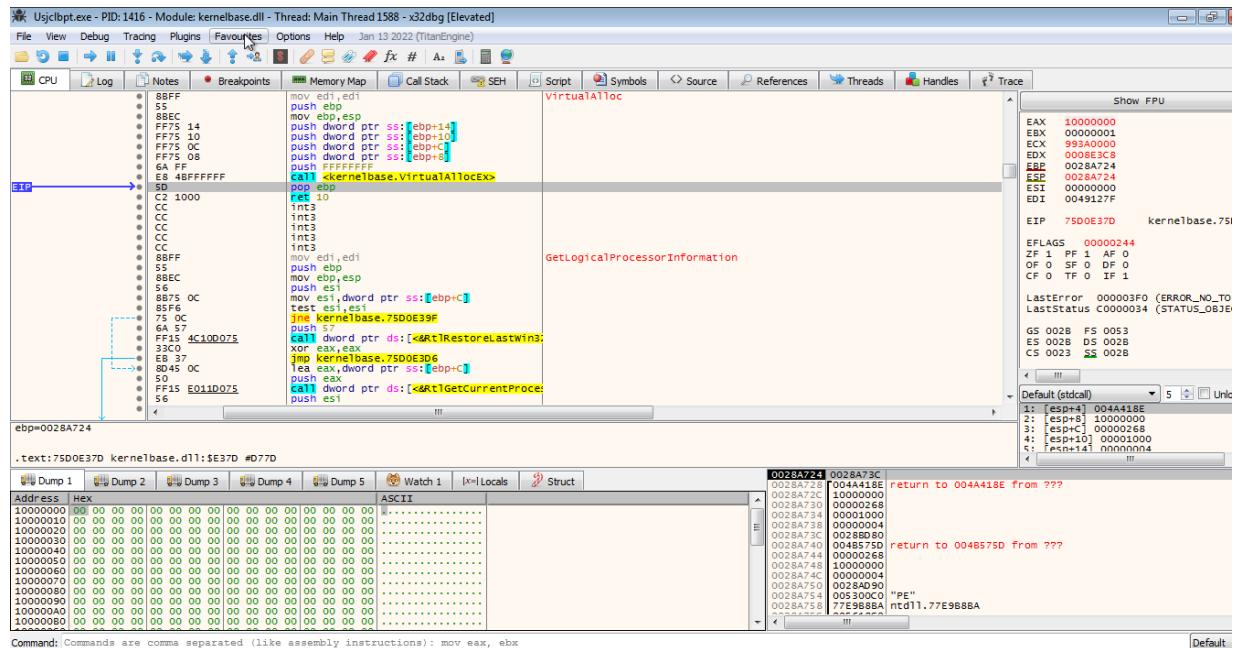
We hit VirtualAlloc step over it load the address return by it in dump.



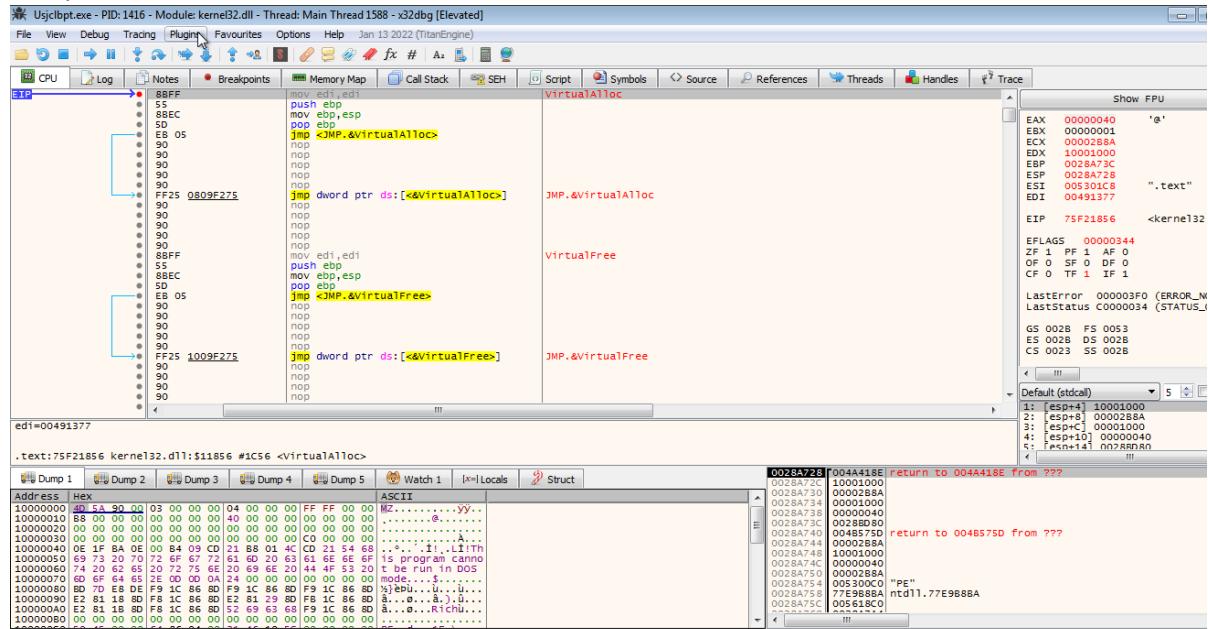
Here just step over it



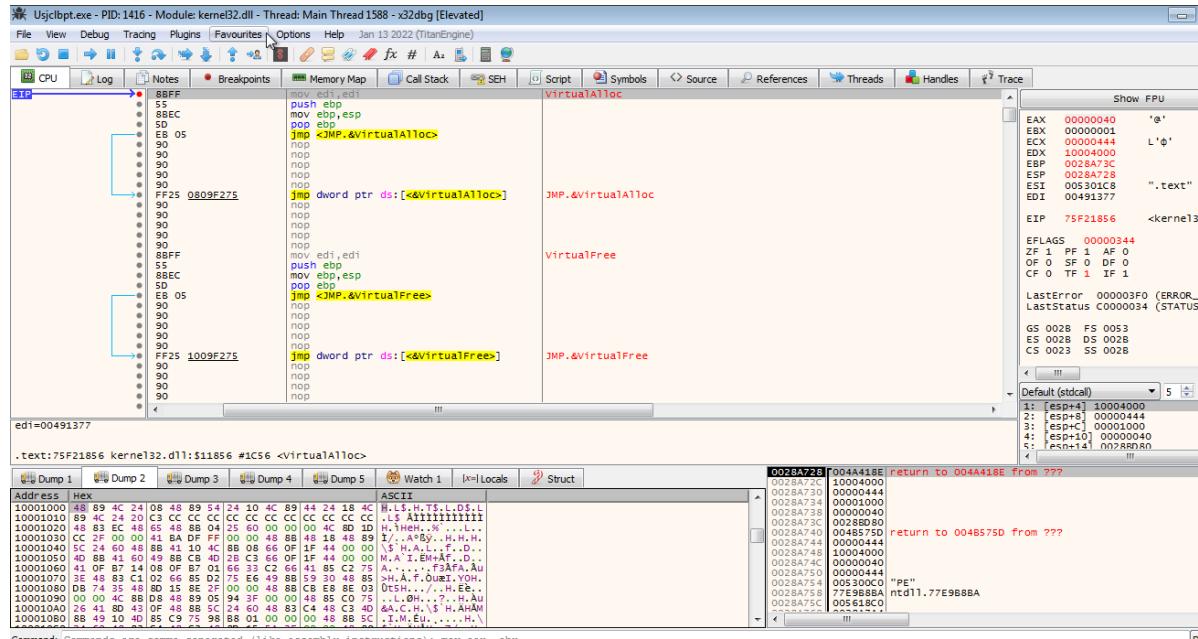
Load the address return by it in dump1 and press f9



We hit `VirtualAlloc` step over it and also the dump1 has been filled and load the address return by it in dump2

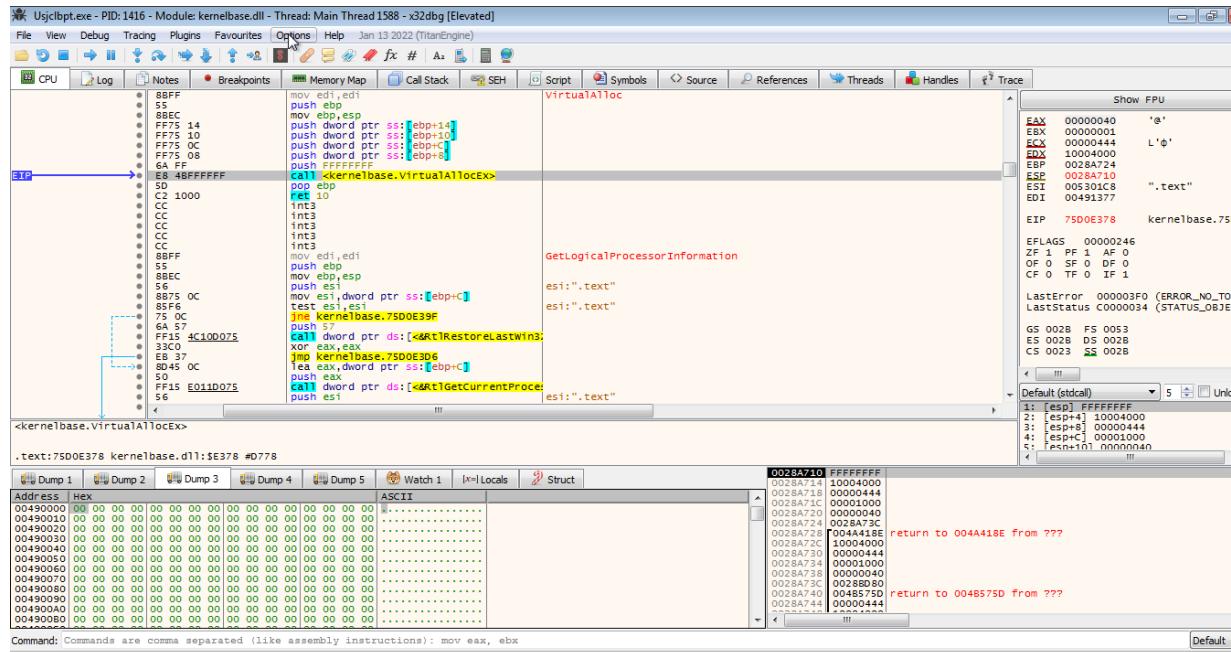


We hit VirtualAlloc step over it and also the dump2 has been filled and load the address return by it in dump3

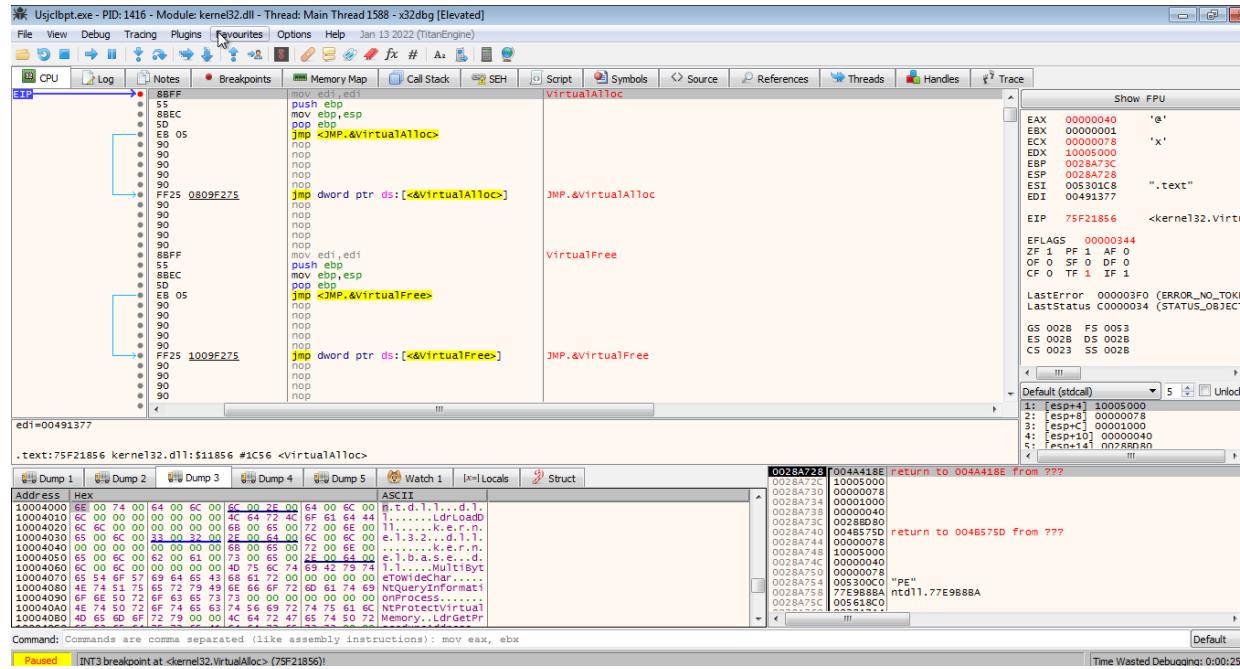


Here just step over it and load the address return by it in dump3.

Note: Here the address `return(10004000)` is in continuation to the previously allocated chunk `(10001000)`.

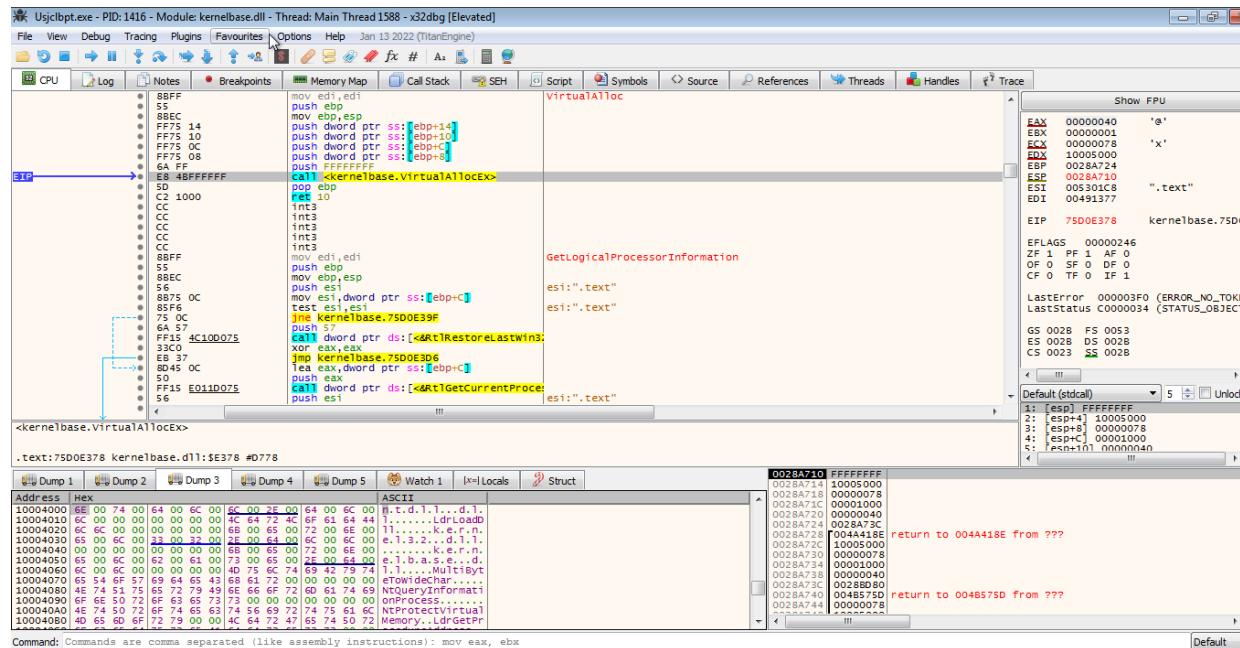


We hit VirtualAlloc step over it and also the dump3 has been filled and load the address return by it in dump4

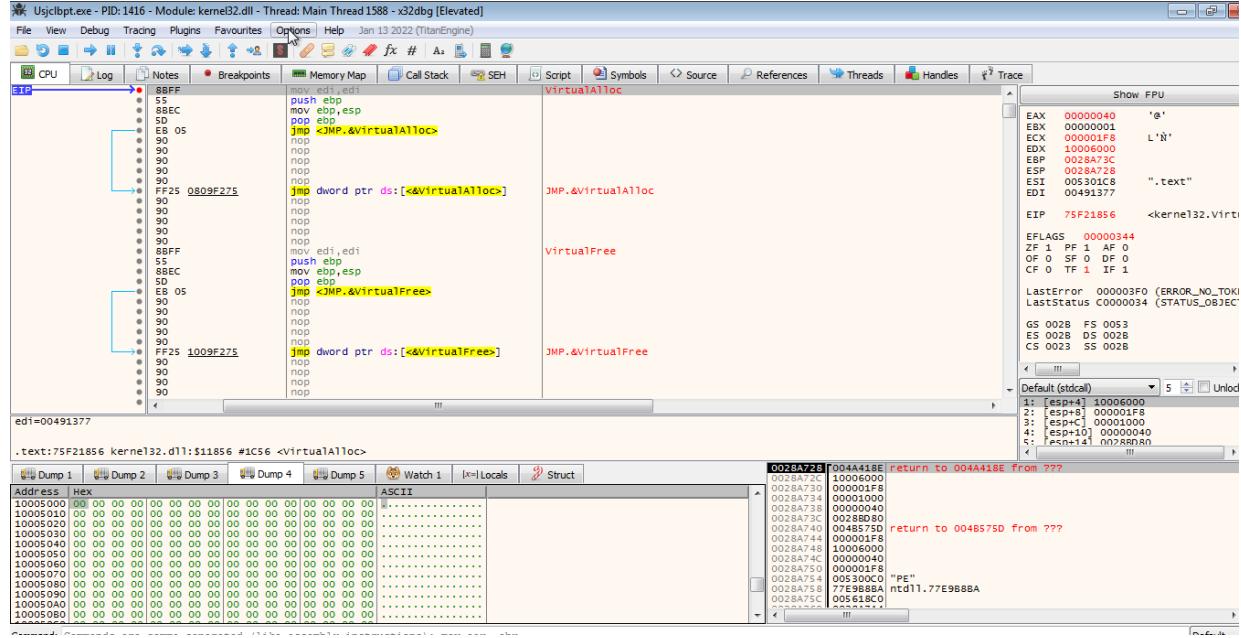


Here just step over it and load the address return by it in dump4

Note: Here the address return(100005000) is in continuation to the previously allocated chunk(100004000)

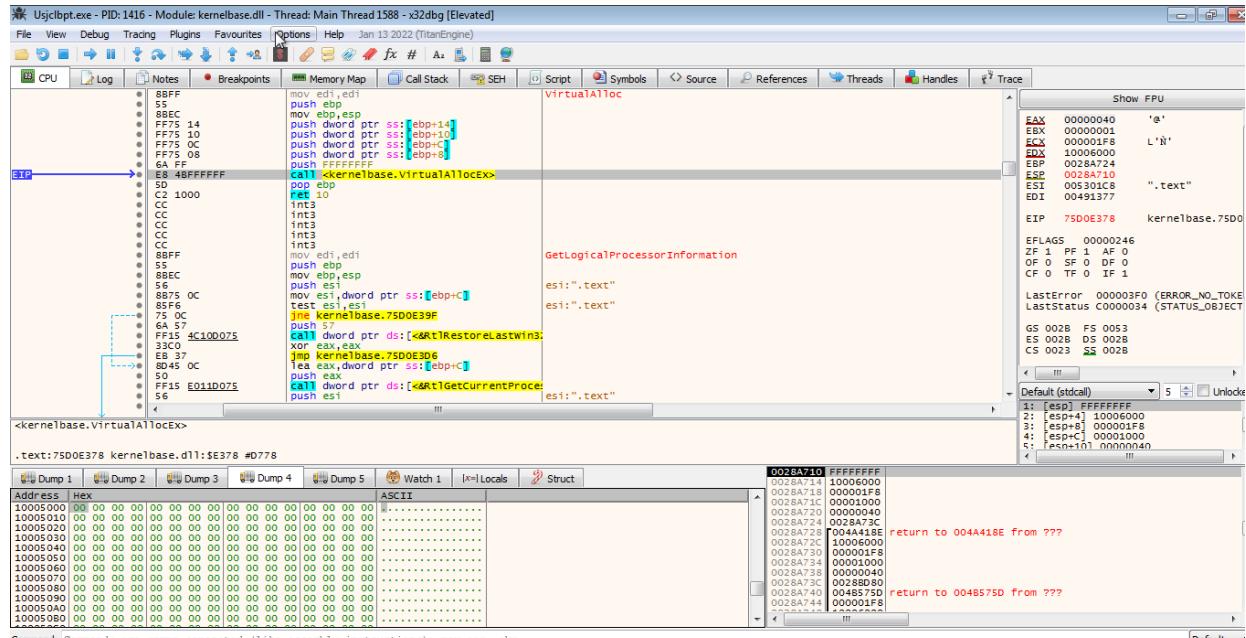


We hit VirtualAlloc step over it and also the dump4 is still empty and load the address return by it in dump5



Here just step over it and load the address return by it in dump5

Note: Here the address `return(10006000)` is in continuation to the previously allocated chunk(`10005000`)



We hit VirtualAlloc step over it and also the dump5 filled with shellcode and load the address return by it in dump1

Assembly code for VirtualAlloc:

```

    mov edi,edi
    push ebp
    mov ebp,esp
    pop ebp
    jmp <JMP.&VirtualAlloc>
    ...
    jmp dword ptr ds:[<&VirtualAlloc>]
    ...
    VirtualFree
    ...
    jmp <JMP.&VirtualFree>
    ...
    jmp dword ptr ds:[<&VirtualFree>]
    ...

```

Registers pane (寄存器) values:

- EAX: 00000040 'g'
- EBX: 00000001
- ECX: 0000001F
- EDX: 00000000
- EBP: 0028A73C
- ESP: 0028A730
- EST: 00530242 "data"
- EDI: 0049170B
- EIP: 75F21856 <kernel32.VirtualAlloc>

Dump5 pane (Dump5) memory dump data:

Address	Hex	ASCII
10006000	20 10 00 00	CE 10 00 00 F4 42 00 00 CE 10 00 000B..I...
10006010	CC 42 00 00	OC 13 00 00 3E 13 00 00 3E 13 00 00AB...>...
10006020	60 13 00 00	CC 42 00 00 3E 13 00 00 CC 42 00 00AB...R...IB...
10006030	60 13 00 00	CC 42 00 00 3E 13 00 00 CC 42 00 00AB...R...IB...
10006040	6D 14 00 00	CO 42 00 00 6D 14 00 00 C5 15 00 00 ... m...AB...m...A...
10006050	A8 42 00 00	CO 42 00 00 EB 15 00 00 90 42 00 00 ... B...A...e...B...
10006060	00 42 00 00	CC 17 00 00 1C 17 00 00 4E 17 00 00B...P...F...
10006070	1C 17 00 00	CC 17 00 00 1C 17 00 00 4E 17 00 00B...P...F...
10006080	00 42 00 00	50 17 00 00 CC 17 00 00 D0 41 00 00 ... B...P...I...0A...
10006090	D0 17 00 00	D0 17 00 00 26 18 00 00 F4 41 00 00 ... 30 18 00 00 D...A...0A...
100060A0	00 42 00 00	50 17 00 00 CC 17 00 00 D0 41 00 00 ... B...P...I...0A...
100060B0	D0 41 00 00	CO 18 00 00 11 19 00 00 E4 41 00 00 ... DA...A...a...a...

Here just step over it and load the address return by it in dump2

Assembly code for VirtualAlloc:

```

    mov edi,edi
    push ebp
    mov ebp,esp
    push dword ptr ss:[ebp+14]
    push dword ptr ss:[ebp+10]
    push dword ptr ss:[ebp+C]
    push dword ptr ss:[ebp+8]
    push FFFFFFFF
    call <kernelbase.VirtualAllocEx>
    ...
    GetLogicalProcessorInformation
    ...

```

Registers pane (寄存器) values:

- EAX: 00000040 'g'
- EBX: 00000001
- ECX: 0000000F
- EDX: 00000000
- EBP: 0028A724
- ESP: 0028A710
- EST: 00530242 "data"
- EDI: 0049170B
- EIP: 75D0E378 <kernelbase.75D0E378>

Dump5 pane (Dump5) memory dump data:

Address	Hex	ASCII
10006000	20 10 00 00	CE 10 00 00 F4 42 00 00 CE 10 00 000B..I...
10006010	CC 42 00 00	OC 13 00 00 3E 13 00 00 CC 42 00 00AB...R...IB...
10006020	60 13 00 00	CC 42 00 00 3E 13 00 00 CC 42 00 00AB...R...IB...
10006030	60 13 00 00	CC 42 00 00 3E 13 00 00 CC 42 00 00AB...R...IB...
10006040	6D 14 00 00	CO 42 00 00 6D 14 00 00 C5 15 00 00 ... m...AB...m...A...
10006050	A8 42 00 00	CO 42 00 00 EB 15 00 00 90 42 00 00 ... B...A...e...B...
10006060	00 42 00 00	CC 17 00 00 1C 17 00 00 4E 17 00 00B...P...F...
10006070	1C 17 00 00	CC 17 00 00 1C 17 00 00 4E 17 00 00B...P...F...
10006080	00 42 00 00	50 17 00 00 CC 17 00 00 D0 41 00 00 ... B...P...I...0A...
10006090	D0 17 00 00	D0 17 00 00 26 18 00 00 F4 41 00 00 ... 30 18 00 00 D...A...0A...
100060A0	00 42 00 00	50 17 00 00 CC 17 00 00 D0 41 00 00 ... B...P...I...0A...
100060B0	D0 41 00 00	CO 18 00 00 11 19 00 00 E4 41 00 00 ... DA...A...a...a...

Here just step over it and load the address return by it in dump, Note:Here the address return is not in continuation to the previously allocated chunk so it means it has finished unpacking so we can ignore this.

Note:these are the dumps with continuous addresses.

.text:75F33BF3 kernel32.dll:\$23BF3 #13FF3 <CreateProcessInternalW>					
Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1
Address	Hex	ASCII			
10000000	AD 5A 90 00	03 00 00 00 04 00 00 00 FF FF 00 00	MZ...yy..		
10000010	B8 00 00 00	00 00 00 00 40 00 00 00@.....		
10000020	00 00 00 00	00 00 00 00 00 00 00 00		
10000030	00 00 00 00	00 00 00 00 00 00 00 00A.....		
10000040	0E 1E 8A 0E	90 B4 09 CD 21 BB 01 4C CD 21 54 68i..LITH		
10000050	69 73 20 70	72 6F 67 72 61 6D 20 63 66 6E 6F	is program canno		
10000060	74 20 62 65	20 72 73 6E 20 69 65 20 44 4F 53 20	be run in DOS		
10000070	00 00 00 00	00 00 00 00 00 00 00 00	mode		
10000080	8D 70 68 DE	F9 1C 86 80 F9 1C 86 80 F9 1C 86 80	%!epu...!..!..!		
10000090	E2 81 18 8D	F8 1C 86 80 E2 81 29 8D F9 1C 86 80	...a...a...!..!..!		
100000A0	00 00 00 00	00 00 00 00 00 00 00 00	...B...R!chu...		
100000B0	00 00 00 00	00 00 00 00 00 00 00 00	...A...!		

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Paused INT3 breakpoint at kernel32.CreateProcessInternalW> (75F33BF3)

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	[x=] Locals	Struct
Address	Hex	ASCII					
10000000	88 49 4C 24	08 46 88 44 24 10 4C 89	!A!T!-!-!-!-!-!-				
10000100	88 4C 24 20	03 CC	H..!H..!H..!H..!H..!H..!H..!H..!H..!H..!H..!H..!H..!H..!H..!H..!H..!				
10000120	48 83 EC 48	65 48 88 04 25 60 00 00 00	4C 8D 1D	H..!H..H..!H..!H..!			
10000130	CC 2F 00 00	41 BA DF FF 00 00	48 88 48 18 48 89	!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!			
10000140	00 00 00 00	48 88 48 18 48 18 48 89	48 88 48 18 48 89	!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!			
10000150	4D 88 41 60	49 88 C8 4D 28 C3 66 0F 1F 44 00 00	00 M..!..!..!..!..!..!				
10000160	41 0F 87 14	08 0F 87 01 66 33 C2 66 41 85 C2 75	A...!..!..!..!..!..!..!..!				
10000170	3E 48 83 C1	02 66 85 D2 75 E6 49 88 58 30 48 85	!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!..!				
10000180	00 00 00 00	48 88 05 94 3F 00 00 48 88 C0 55	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
10000190	00 00 00 00	4C 88 D8 48 88 05 94 3F 00 00 48 88 C0 55	..!..!..!..!..!..!..!..!				
100001A0	26 41 BD 43	0F 48 88 SC 24 60 48 83 C4 48 C3 4D	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
100001B0	88 49 10 4D	85 C9 75 98 BB 01 04 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	[x=] Locals	Struct
Address	Hex	ASCII					
10004000	6E 00 74 00	64 00 6C 00 6C 00 2E 00 64 00 6C 00	!..!..!..!..!..!..!..!..!..!..!				
10004010	6C 6C 00 00	00 00 00 00 00 00 00 00 00 00	4C 64 72 4C 6F 61 64 44	!..!..!..!..!..!..!..!..!..!..!			
10004020	6C 6C 00 00	00 00 00 00 00 00 00 00 00 00	72 00 6E 00	!..!..!..!..!..!..!..!..!..!..!			
10004030	6C 6C 00 00	00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	!..!..!..!..!..!..!..!..!..!..!			
10004040	00 00 00 00	48 88 05 94 3F 00 00 00 00 00	00 00 00 00 00 00 00 00	!..!..!..!..!..!..!..!..!..!..!			
10004050	65 00 60 00	62 00 61 00 73 00 65 00	2E 00 64 00	!..!..!..!..!..!..!..!..!..!..!			
10004060	6C 00 6C 00	00 00 00 00 00 00 00 00 00 00	73 00 6C 00 6C 00 6C 00	!..!..!..!..!..!..!..!..!..!..!			
10004070	4E 74 51 75	65 72 79 49 6E 66 6F 72 6D 61 74 69	!..!..!..!..!..!..!..!..!..!..!				
10004080	4E 74 51 75	65 72 79 49 6E 66 6F 72 6D 61 74 69	N!queryInformati				
10004090	6E 6E 50 72	6F 65 63 73 73 00 00 00 00 00 00	00 00 00 00 00 00 00 00	onProcess.....			
100040A0	4E 74 50 72	6F 65 63 73 73 00 00 00 00 00 00	00 00 00 00 00 00 00 00	N!protectVirtual			
100040B0	4D 65 60 6F	72 79 00 00 4C 64 72 47	65 74 50 72	Memory..!LdrGetPr			

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	[x=] Locals	Struct
Address	Hex	ASCII					
10005000	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	!..!..!..!..!..!..!..!..!..!..!				
10005010	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	!..!..!..!..!..!..!..!..!..!..!				
10005020	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	!..!..!..!..!..!..!..!..!..!..!				
10005030	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	!..!..!..!..!..!..!..!..!..!..!				
10005040	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	!..!..!..!..!..!..!..!..!..!..!				
10005050	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	!..!..!..!..!..!..!..!..!..!..!				
10005060	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	!..!..!..!..!..!..!..!..!..!..!				
10005070	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	!..!..!..!..!..!..!..!..!..!..!				
10005080	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	!..!..!..!..!..!..!..!..!..!..!				

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Paused INT3 breakpoint at kernel32.CreateProcessInternalW> (75F33BF3)

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	[x=] Locals	Struct
Address	Hex	ASCII					
10006000	28 10 00 00	CE 10 00 00 F4 42 00 00 CE 10 00 00	..!..!..!..!..!..!..!..!				
10006010	0C 13 00 00	E0 42 00 00 E0 13 00 00 3E 13 00 00	..ab..!..!..!..!..!				
10006020	CC 42 00 00	3E 13 00 00 52 13 00 00 CC 42 00 00	!..!..!..!..!..!..!..!..!				
10006030	60 13 00 00	1E 14 00 00 34 42 00 00 20 14 00 00	..!..!..!..!..!..!..!..!				
10006040	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00	!..!..!..!..!..!..!..!..!				
10006050	60 13 00 00	1E 14 00 00 34 42 00 00 20 14 00 00	..!..!..!..!..!..!..!..!				
10006060	A6 42 00 00	C0 15 00 00 EB 15 00 00 90 42 00 00	m..!..!..!..!..!..!..!..!				
10006070	F0 15 00 00	7D 16 00 00 28 42 00 00 7D 16 00 00	!..!..!..!..!..!..!..!..!				
10006080	1C 17 00 00	14 42 00 00 1C 17 00 00 46 17 00 00	..!..!..!..!..!..!..!..!				
10006090	DO 17 00 00	26 18 00 00 F4 41 00 00 30 18 00 00	..!..!..!..!..!..!..!..!				
100060A0	7F 18 00 00	EC 44 00 00 B8 18 00 00 B3 18 00 00	D..!..!..!..!..!..!..!..!				
100060B0	DO 41 00 00	CO 18 00 00 11 19 00 00 E4 41 00 00	DA..!..!..!..!..!..!..!..!				

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Paused INT3 breakpoint at kernel32.CreateProcessInternalW> (75F33BF3)

Usjclbpt.exe - PID:1416 - Module: kernelbase.dll - Thread: Main Thread 1588 - x32dbg [Elevated]

File View Debug Tracing Plugins Favourites Options Help Jan 13 2022 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace

VirtualAlloc

```

    mov edi,edi
    push ebp
    mov esp,esp
    push dword ptr ss:[ebp+14]
    push dword ptr ss:[ebp+10]
    push dword ptr ss:[ebp+C]
    push dword ptr ss:[ebp+8]
    push byte 08
    mov edi,edi
    push ebp
    mov esp,esp
    call kernelbase.VirtualAllocEx
    pop ebp
    int3
    int3
    int3
    int3
    int3
    int3
    mov edi,edi
    push ebp
    mov esp,esp
    push esi
    mov es,dword ptr ss:[ebp+C]
    test es,es
    jne KernelBase.75D0E39F
    push byte 0C
    mov es,dword ptr ds:[<Rt>RestoreLastWin32]
    xor eax,eax
    imp kernelbase.75D0E306
    lea eax,dword ptr ss:[ebp+C]
    push eax
    call dword ptr ds:[<Rt>GetCurrentProcess]
    push esi
    esi:"data"
    
```

EAX 007D0000 EBX 00000001 ECX 993A0000 EDX 00000005 EBP 0028A724 ESP 0028A724 ESI 00530242 "data" EDI 004917DB EIP 75D0E370 kernelbase.dll

LastError 000003F0 (ERROR_NO_TOKEN) LastStatus C0000034 (STATUS_ACCESS_DENIED)

GS 002B FS 0053 ES 002B DS 002B CS 0023 SS 002B

Default (stcall) 5 +

1: [esp+4] 004B5CFA
2: [esp+8] 00000000
3: [esp+C] 0000001F
4: [esp+10] 00003000
5: [esp+14] 00000040

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Now we have hit breakpoint at CreateProcessInternalW(using this API it has created svchost after this it will write the previously unpacked code in it)

Usjclbpt.exe - PID:1416 - Module: kernel32.dll - Thread: Main Thread 1588 - x32dbg [Elevated]

File View Debug Tracing Plugins Favourites Options Help Jan 13 2022 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace

CreateProcessInternalW

```

    push 624
    push 75F33BF3
    call kernel32.75F21098
    mov eax,dword ptr ss:[ebp+8]
    mov dword ptr ss:[ebp+60],eax
    mov dword ptr ss:[ebp+C]
    mov dword ptr ss:[ebp-34]
    mov esi,dword ptr ss:[ebp+10]
    mov eax,dword ptr ss:[ebp+C]
    mov eax,dword ptr ss:[ebp-14]
    mov eax,dword ptr ss:[ebp-464]
    mov eax,dword ptr ss:[ebp-14]
    mov eax,dword ptr ss:[ebp+C]
    mov eax,dword ptr ss:[ebp-24]
    mov eax,dword ptr ss:[ebp+C]
    mov eax,dword ptr ss:[ebp-378]
    mov eax,dword ptr ss:[ebp+C]
    mov eax,dword ptr ss:[ebp-388]
    xor ebx,ebx
    mov dword ptr ss:[ebp-324],ebx
    mov dword ptr ss:[ebp-358],ebx
    mov dword ptr ss:[ebp-418],ebx
    mov dword ptr ss:[ebp-3CC],ebx
    mov dword ptr ss:[ebp-36C],ebx
    mov dword ptr ss:[ebp-3D0],ebx
    mov dword ptr ss:[ebp-3E8],ebx
    
```

EAX 0028A730 L"C:\\Windows\\System32\\svhost.exe" EBX 00000001 ECX 0028A518 EDX 0028A308 L"C:\\Windows\\System32\\svhost.exe" EBP 0028A28C ESP 0028A740 ESI 0028A7F7 EDI 00491833 EIP 75F33BF3 <kernel32.CreateProcessInternalW>

LastError 000003F0 (ERROR_NO_TOKEN) LastStatus C0000034 (STATUS_ACCESS_DENIED)

GS 002B FS 0053 ES 002B DS 002B CS 0023 SS 002B

Default (stcall) 5 +

1: [esp+4] 00000000
2: [esp+8] 00000000
3: [esp+C] 0028A518 L"C:\\Windows\\System32\\svhost.exe"
4: [esp+10] 00000000
5: [esp+14] 00000000

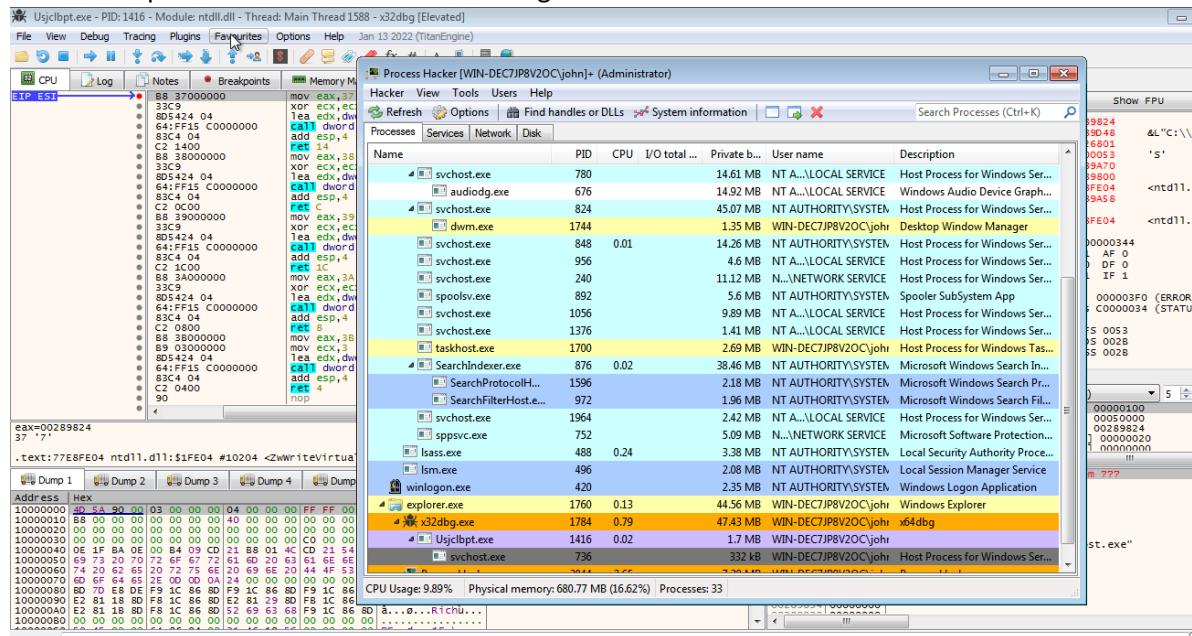
Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Now we are ready to dump, open dump1 and right click on memory 10000000 and open it in memory map.

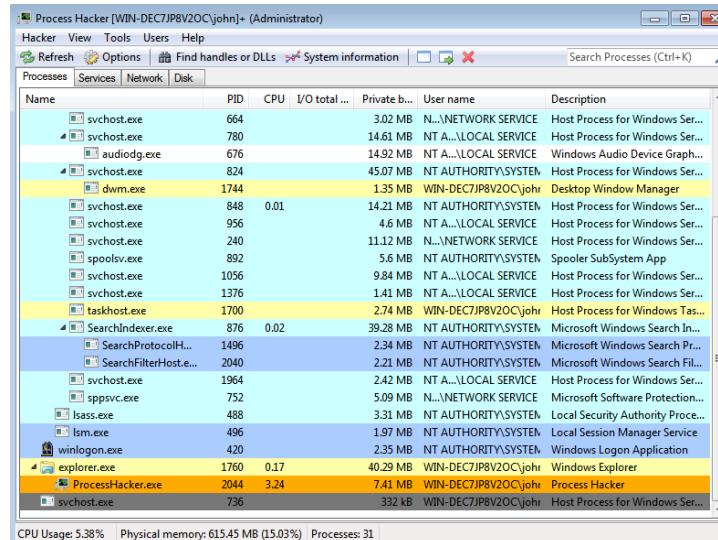
Now from memory one can directly dump that it into a file , by write clicking on Address 1000000. Now press f9.

Now we can note that ZwWriteVirtualMemory it is going to write it into svchost, can verify it by opening process hacker.

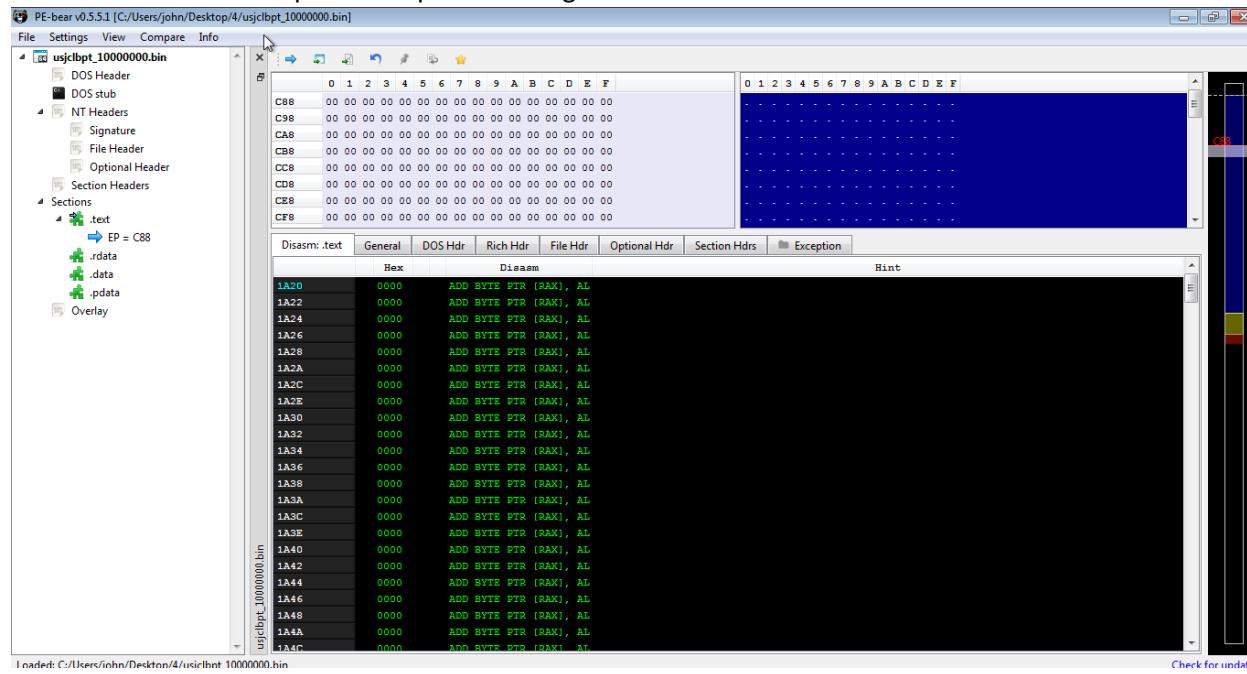
Now if one presses f9 after sometime it will get terminated and a new svchost will start.



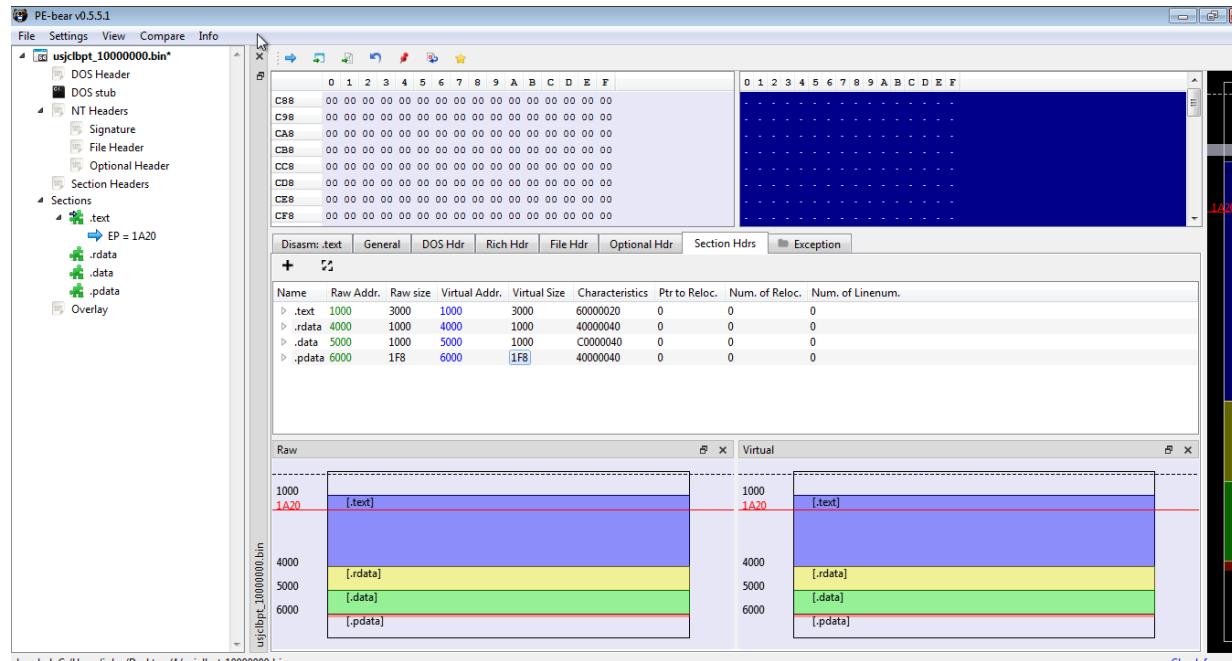
Here the new svchost has started.



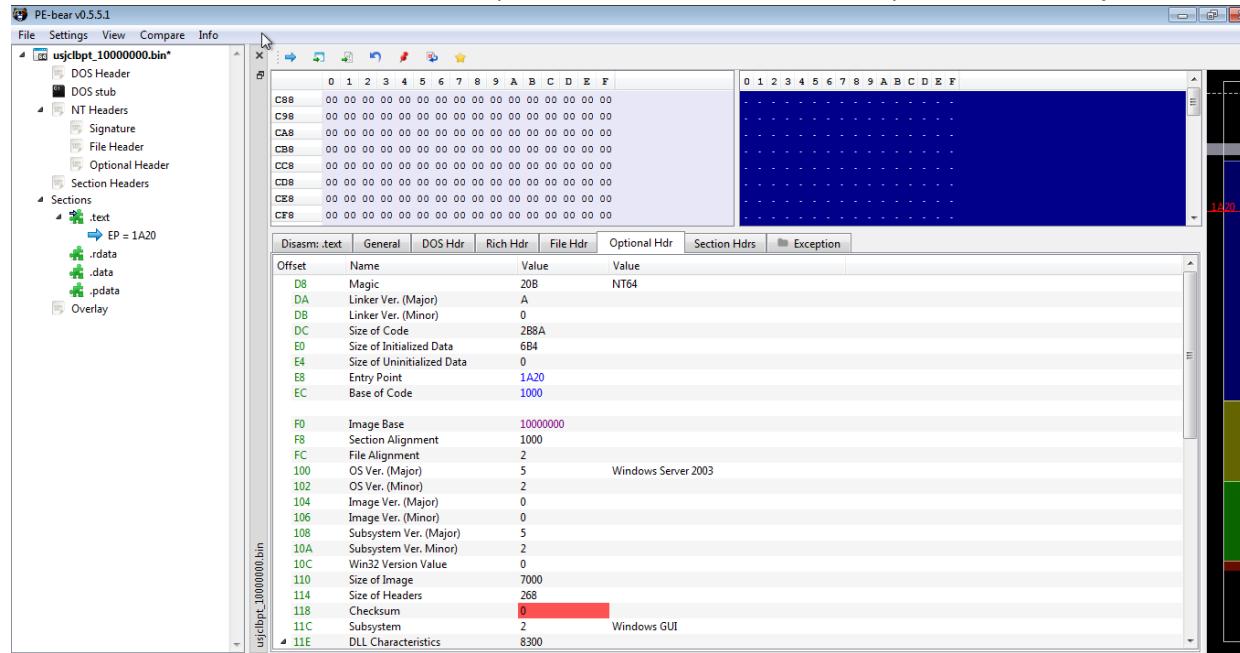
Now here we will unmap the dumped file using PE-bear.



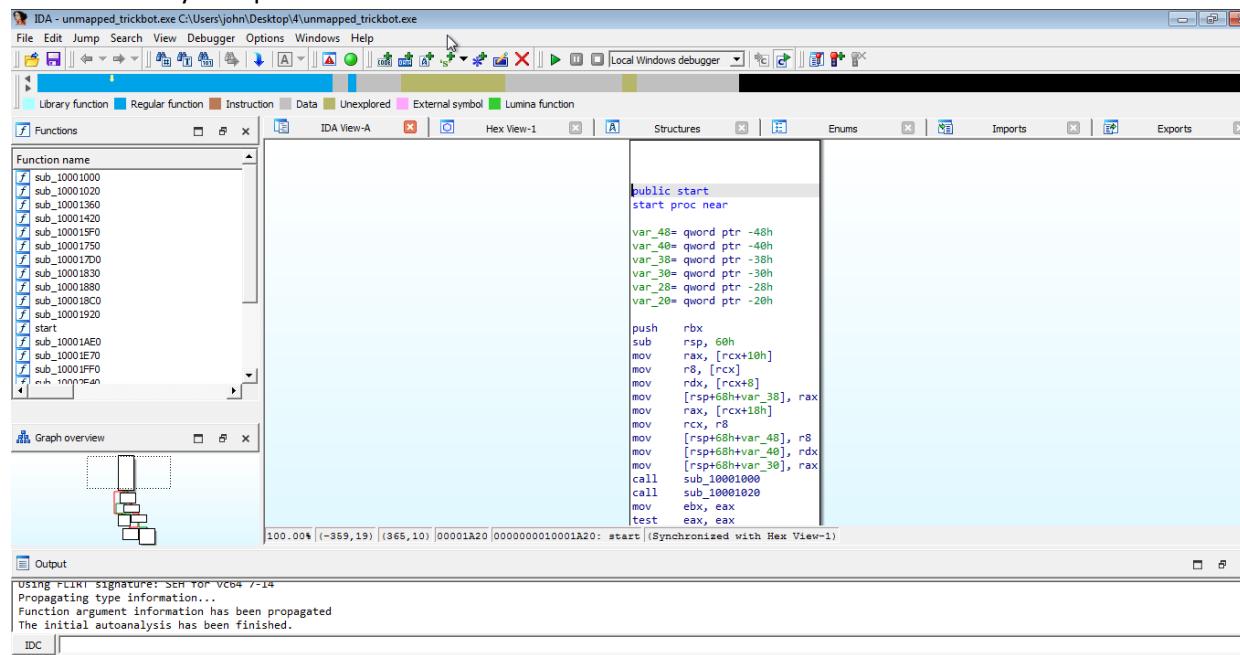
Now here in Section Hdrs tab we need to make raw address same as virtual address , and then calculate the raw size, and then make the virtual size same as raw size.



Now we will check the base address in Optional Hdr tab which was already 10000000, now just save it.



Load the newly dumped file in IDA.



There are no imports here as this trojan will dynamically imports the library as it runs.

