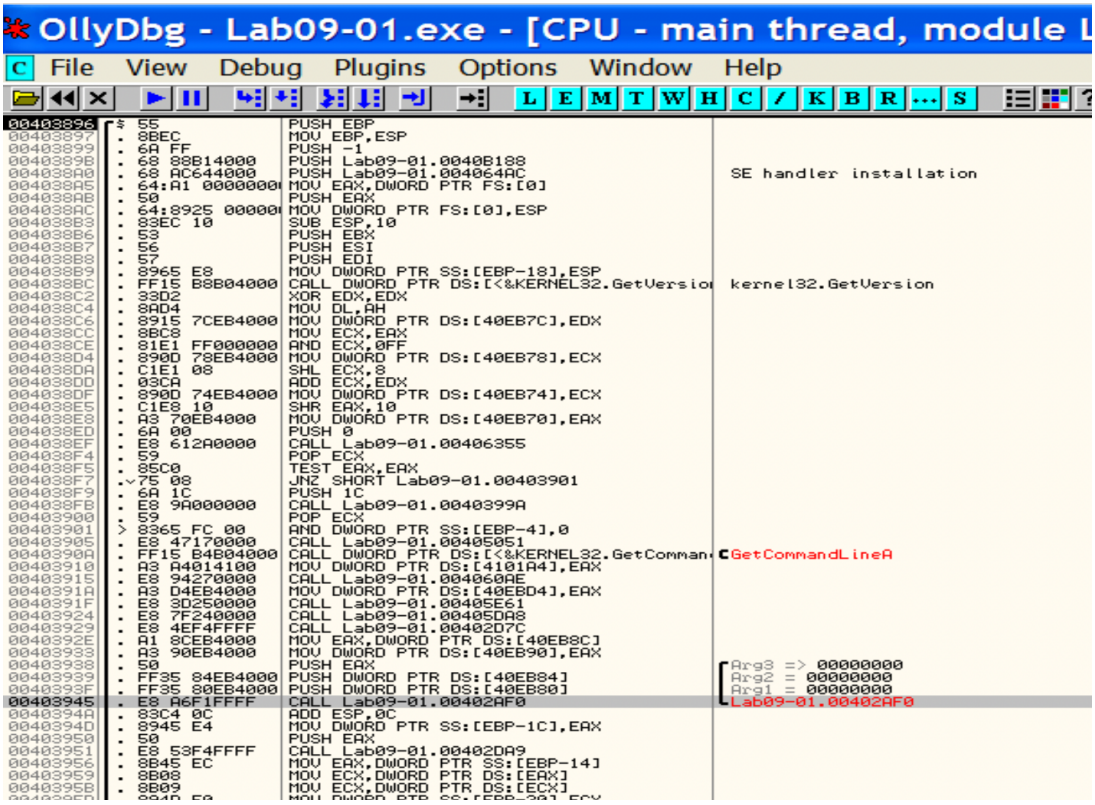
Will Sherrer
HW3 8830

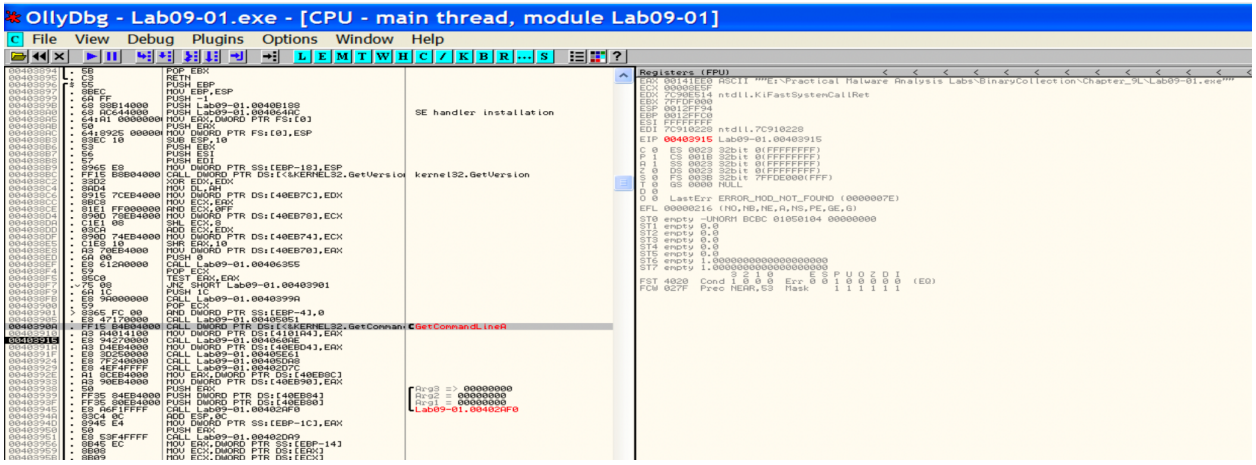
Problem 1

Questions

1. How can you get this malware to install itself?
2. What are the command-line options for this program? What is the password requirement?
3. How can you use OllyDbg to permanently patch this malware, so that it doesn't require the special command-line password?
4. What are the host-based indicators of this malware?
5. What are the different actions this malware can be instructed to take via the network?
6. Are there any useful network-based signatures for this malware?

1. 

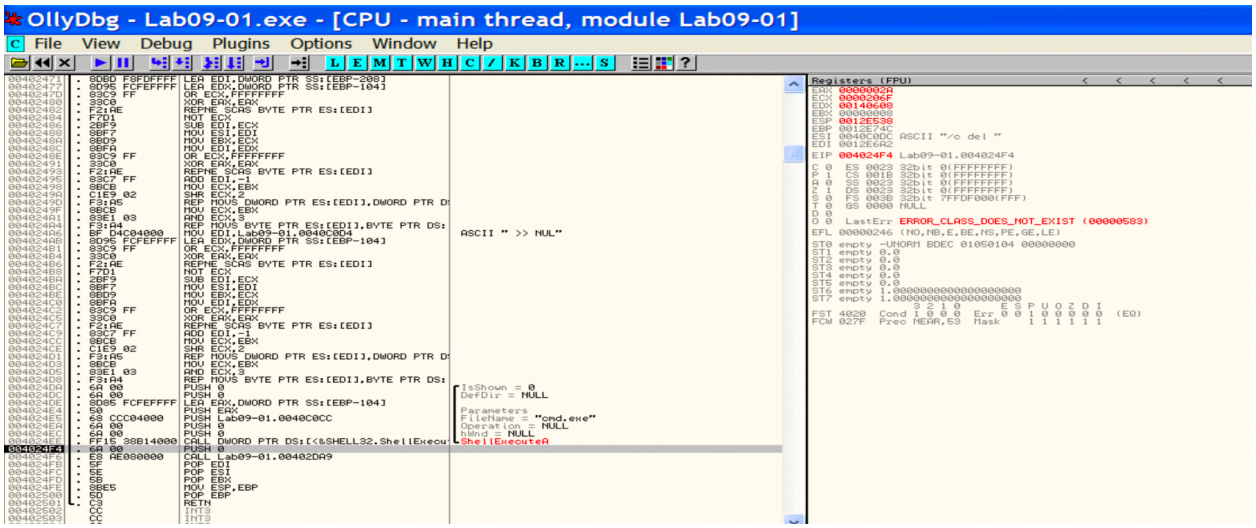
Opening Ollybug shows that it starts at the main function.



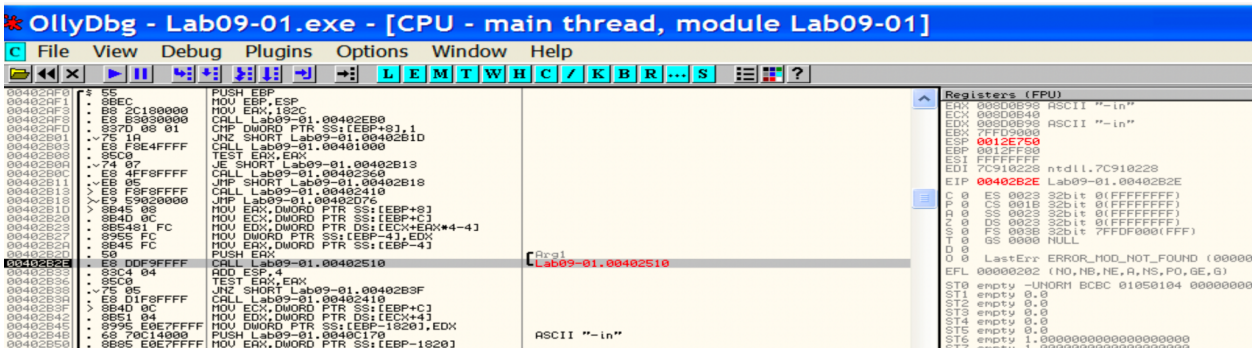
At get command line EAX is updated. I then step into the main function

```
$ 55              PUSH EBP
. 8BEC            MOV EBP,ESP
. 83EC 08         SUB ESP,8
. 8D45 F8         LEA EAX,DWORD PTR SS:[EBP-8]
. 50              PUSH EAX                          pHandle
. 68 3F000F00     PUSH 0F003F                       Access = KEY_ALL_ACCESS
. 6A 00           PUSH 0                            Reserved = 0
. 68 40C04000     PUSH Lab09-01.0040C040            Subkey = "SOFTWARE\Microsoft \XPS"
. 68 02000080     PUSH 80000002                     hKey = HKEY_LOCAL_MACHINE
. FF15 20B04000   CALL DWORD PTR DS:[<&ADVAPI32.RegOpenKe RegOpenKeyExA
. 85C0            TEST EAX,EAX
.v74 04           JE SHORT Lab09-01.00401029
. 33C0            XOR EAX,EAX
.vEB 3D           JMP SHORT Lab09-01.00401066
> 6A 00           PUSH 0                            pBufSize = NULL
. 6A 00           PUSH 0                            Buffer = NULL
```

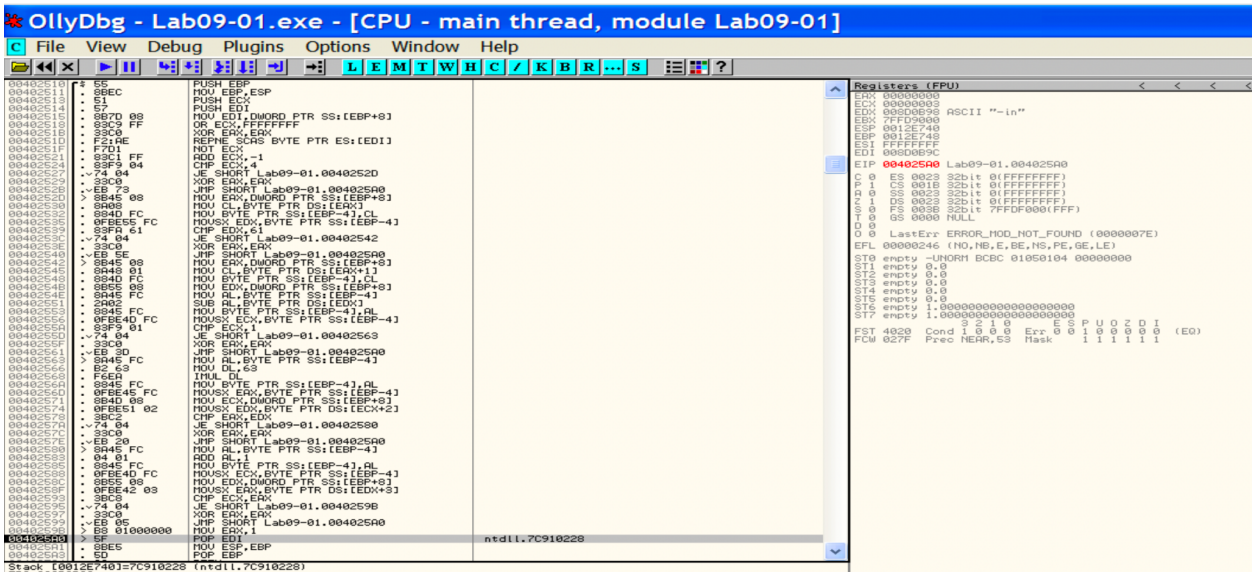
Then the function it goes into compares with a subkey “SOFTWARE\Microsoft\XPS



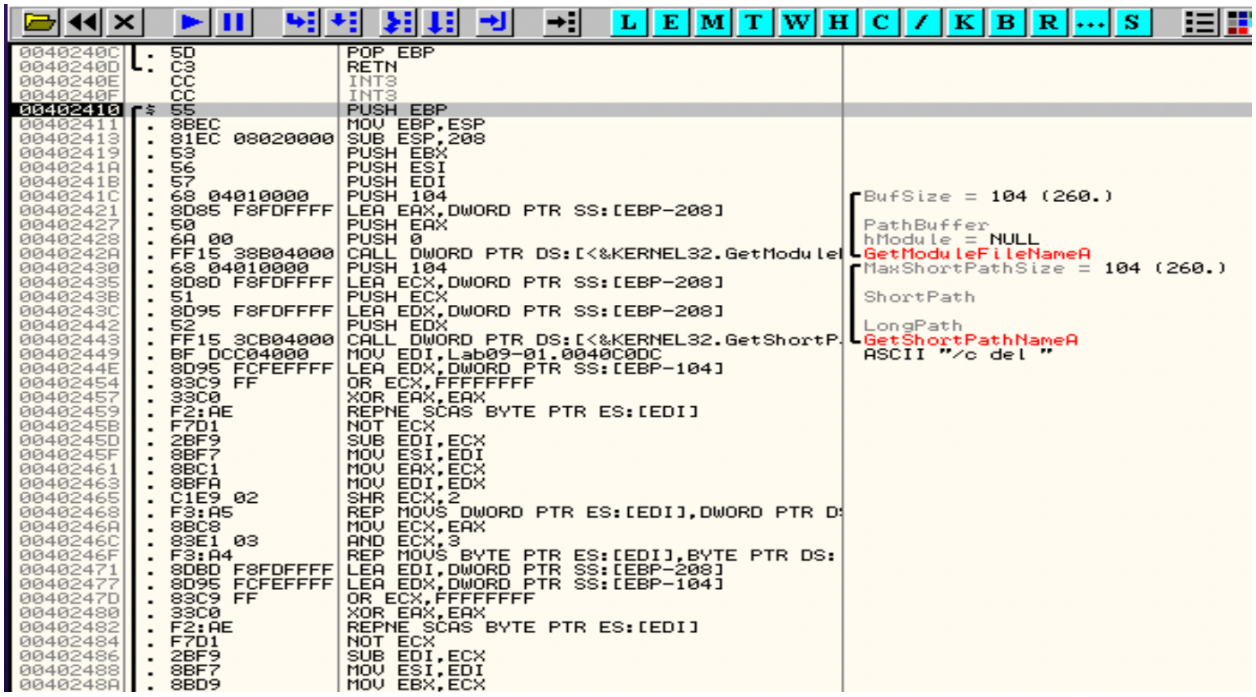
After examining some more I got to this function which seems to delete the malware if does not have any parameters. Therefore running this program isn’t enough to insert it.



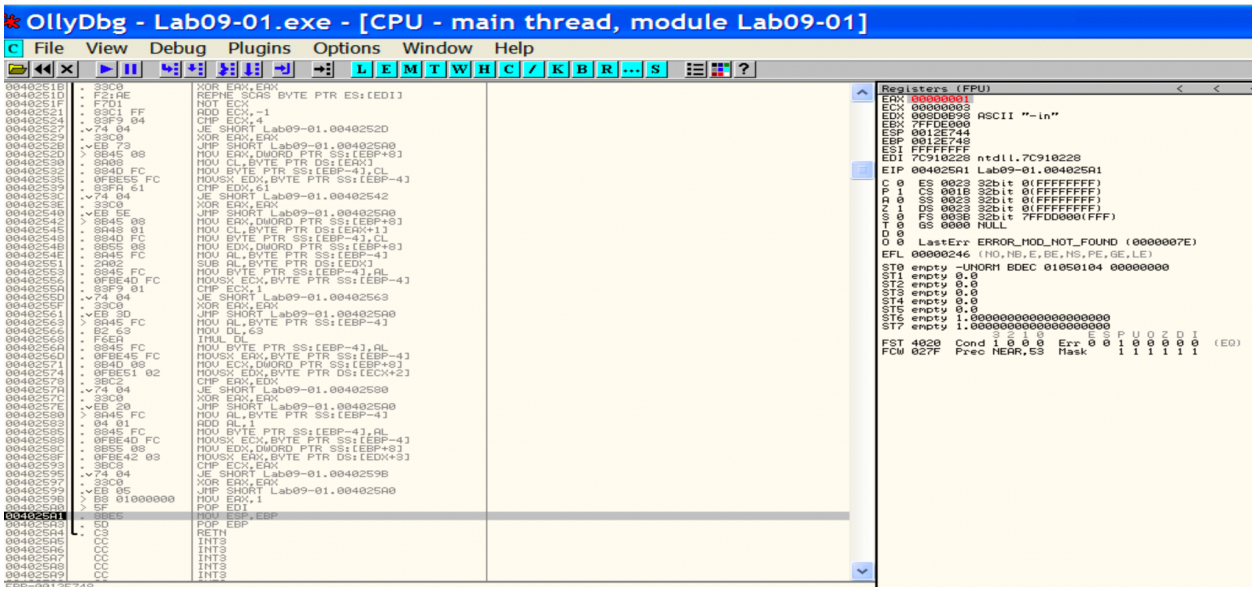
with -in it now jumps and calls the first function.



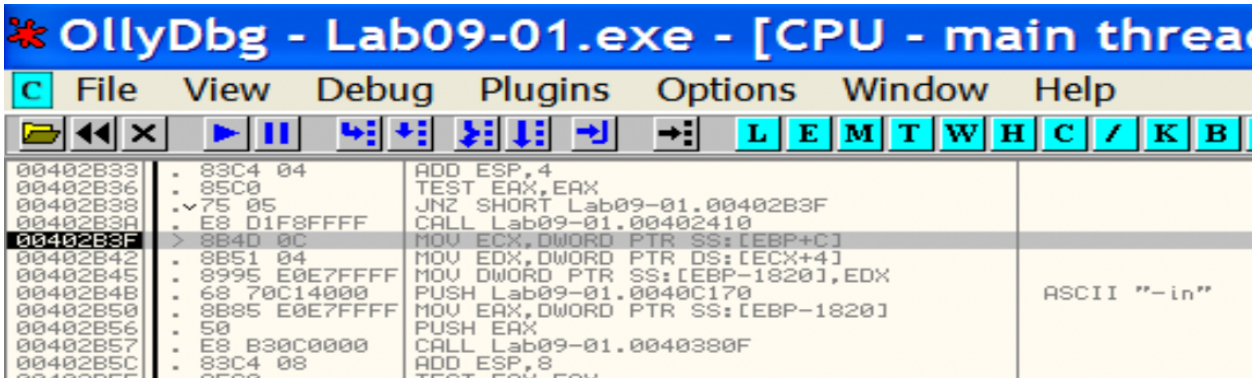
There are several jump calls that are programmed in. When stepping in it jumps to the end of the function and then returns.



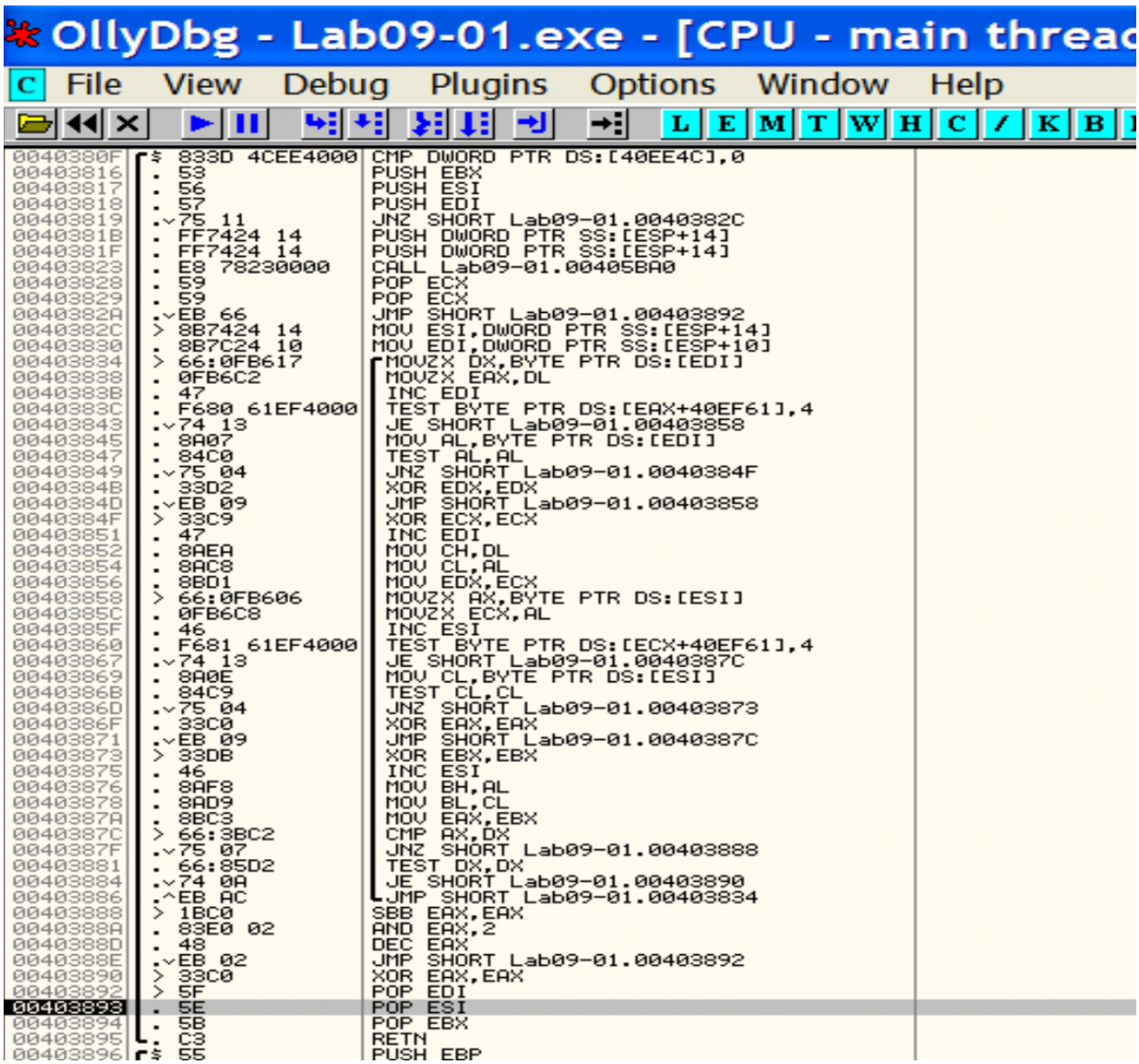
The function then goes back into the function at 402410 which deletes the file. To get back to the function with many jumps (2 screenshots above) I need to make sure that the comparisons are true so that it jumps.



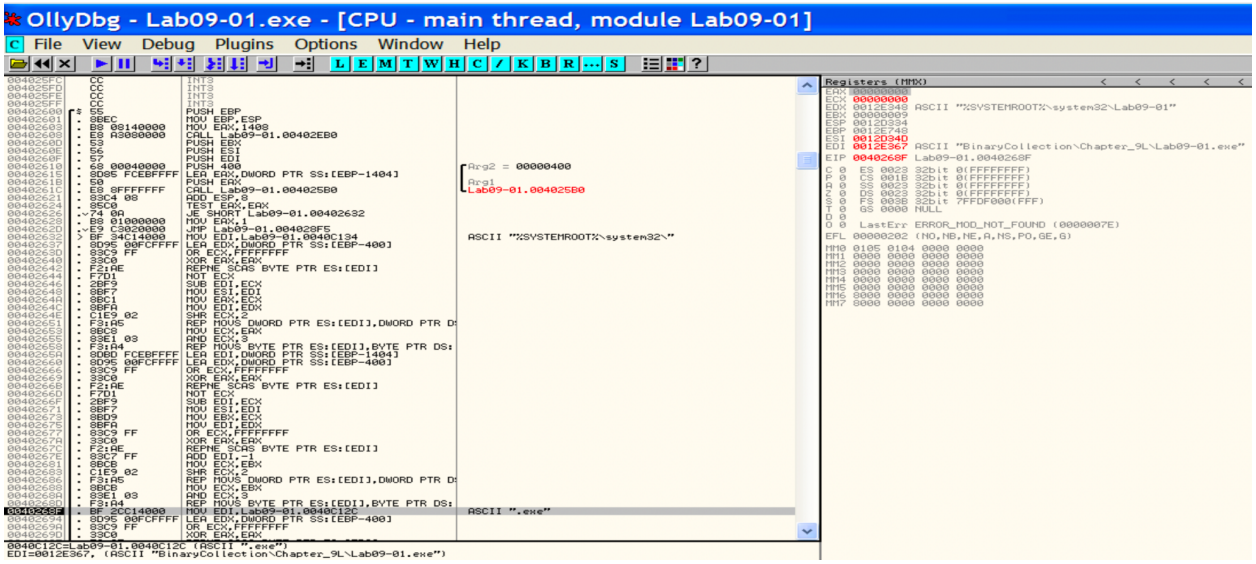
So to bypass this I modify EAX to be 1 so the function returns 1.



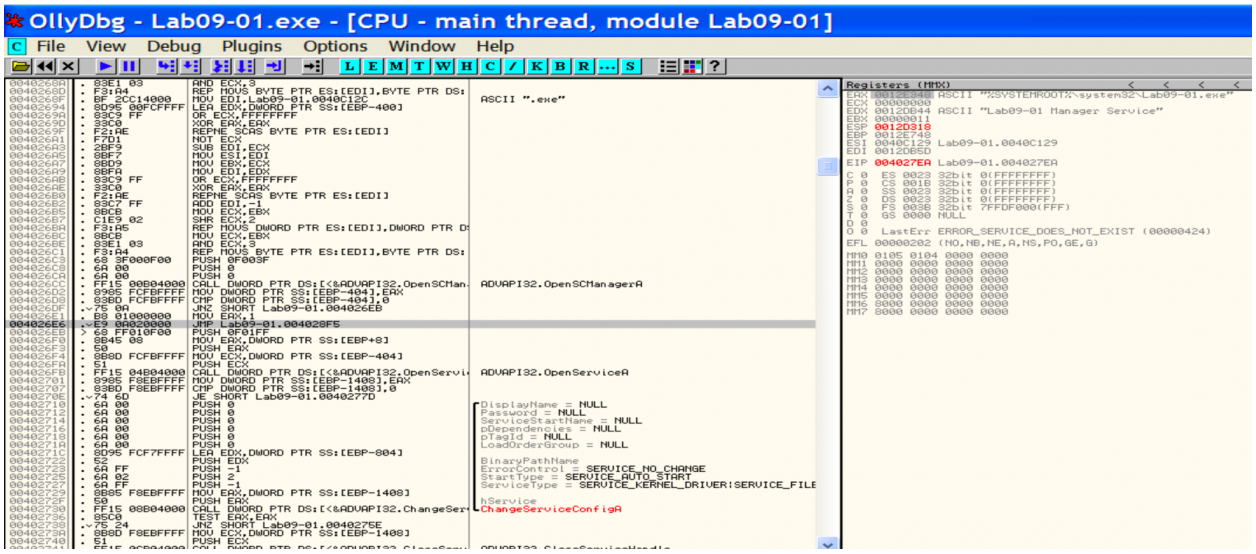
This jumps over the call to 420410 which deletes it.



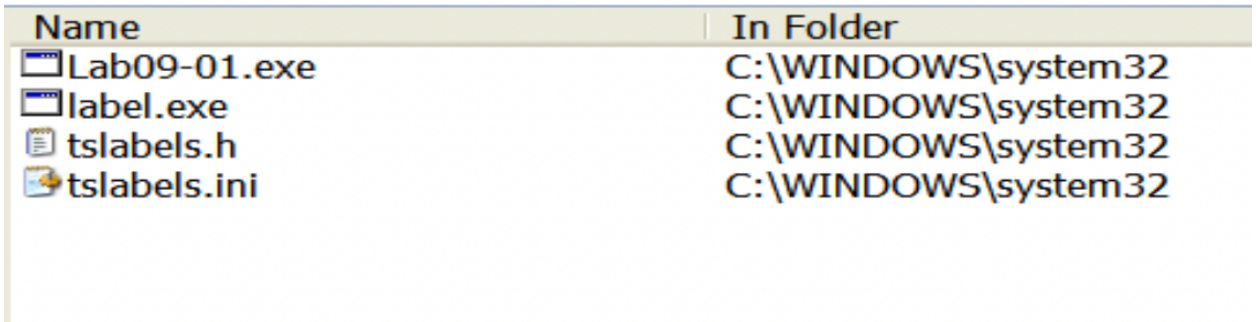
I enter a function at 403380F but fail a jump that compares the arguments number. So again I'm going to run with a random argument



After successfully running and continuous debugging I found something interesting in 40268f. It seems we can see reference to .exe and %SYSTEMROOT%\system32\Lab09-01 which has been taken from the file name passed to the malware.



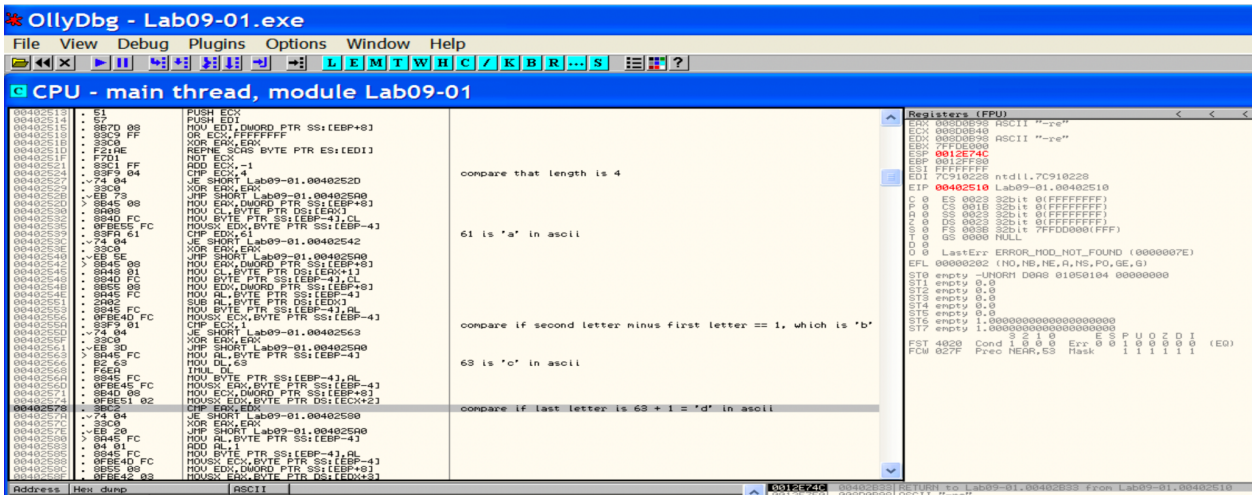
With progressing we can see that it creates a services for the exe file if it doesn't exist



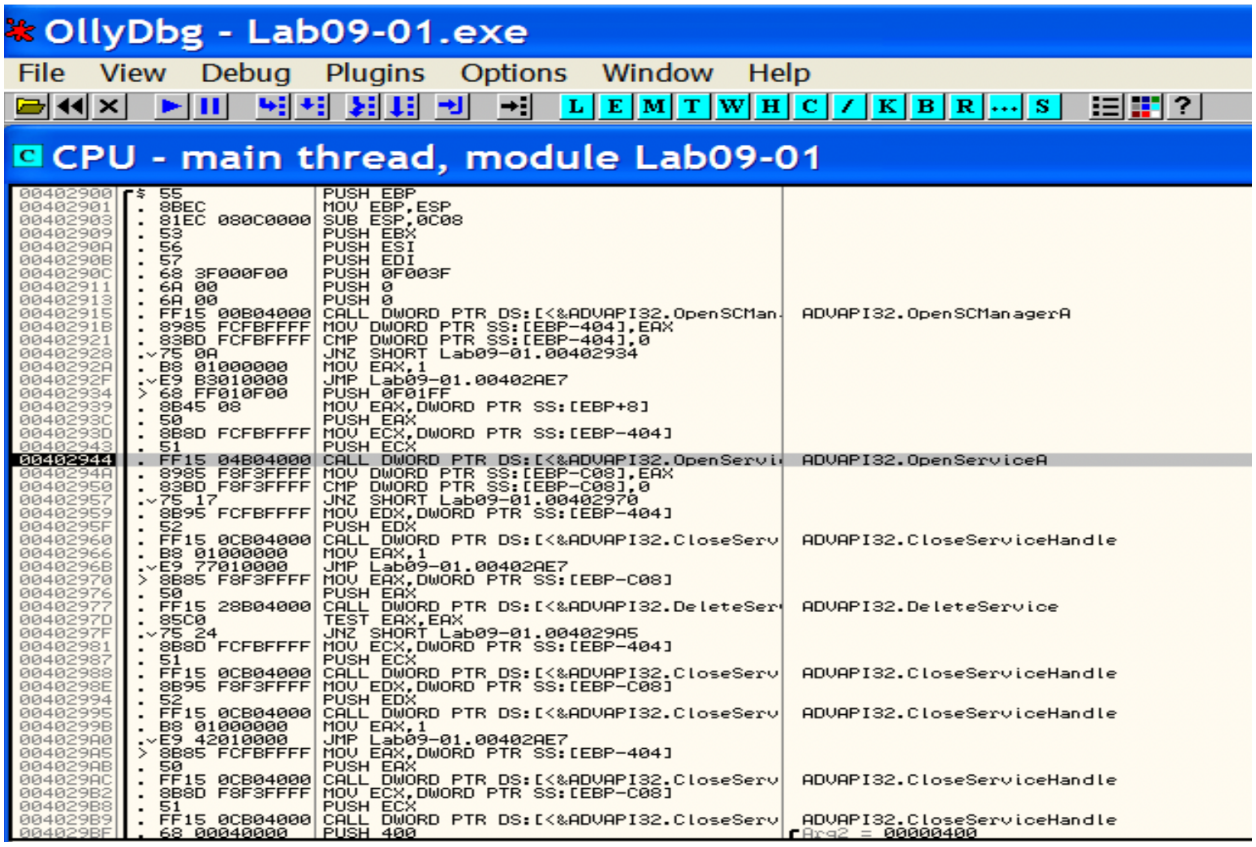
Then after the service was run we see the successful Lab09-01.exe file in system32 (I ran a search for ‘lab’ in the folder)

2. In addition to -in we see that there are also arguments for -re, -r and -cc

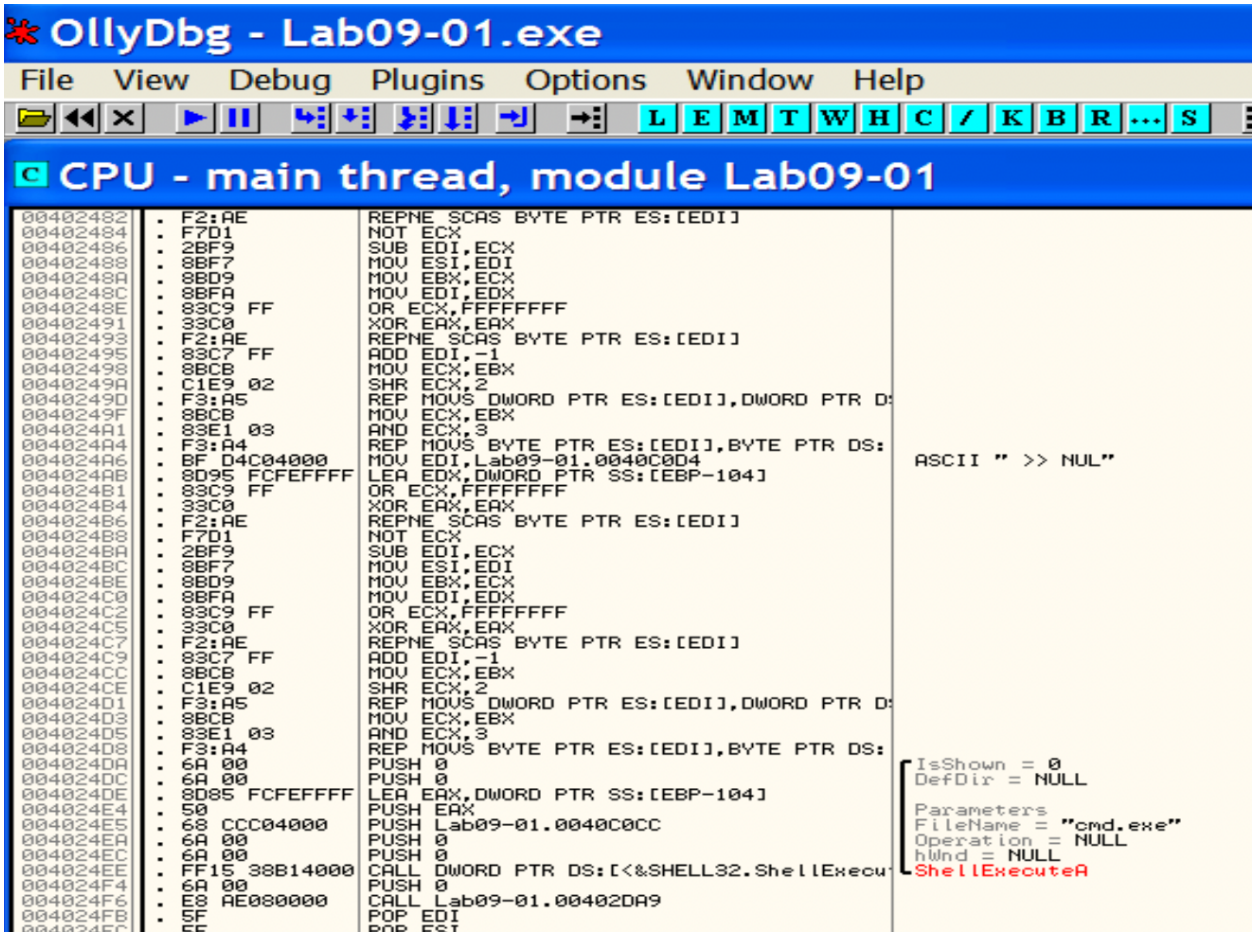
When typing out the -re argument I see that it leads to a comparison in function 402510. I added comments for clarity but the function checks for a password of length 4 and sees that it must be ‘abcd’



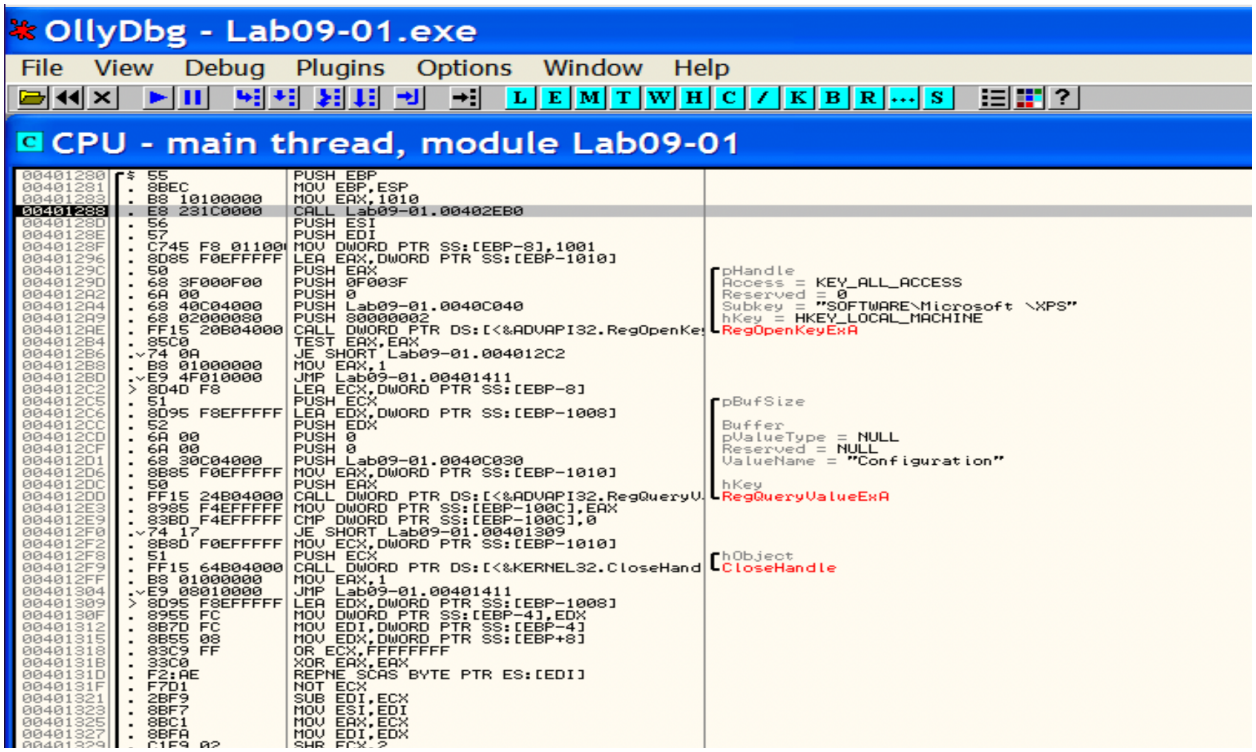
With the -re abcd we see that it enters the second function which calls to delete the service, so I try -c



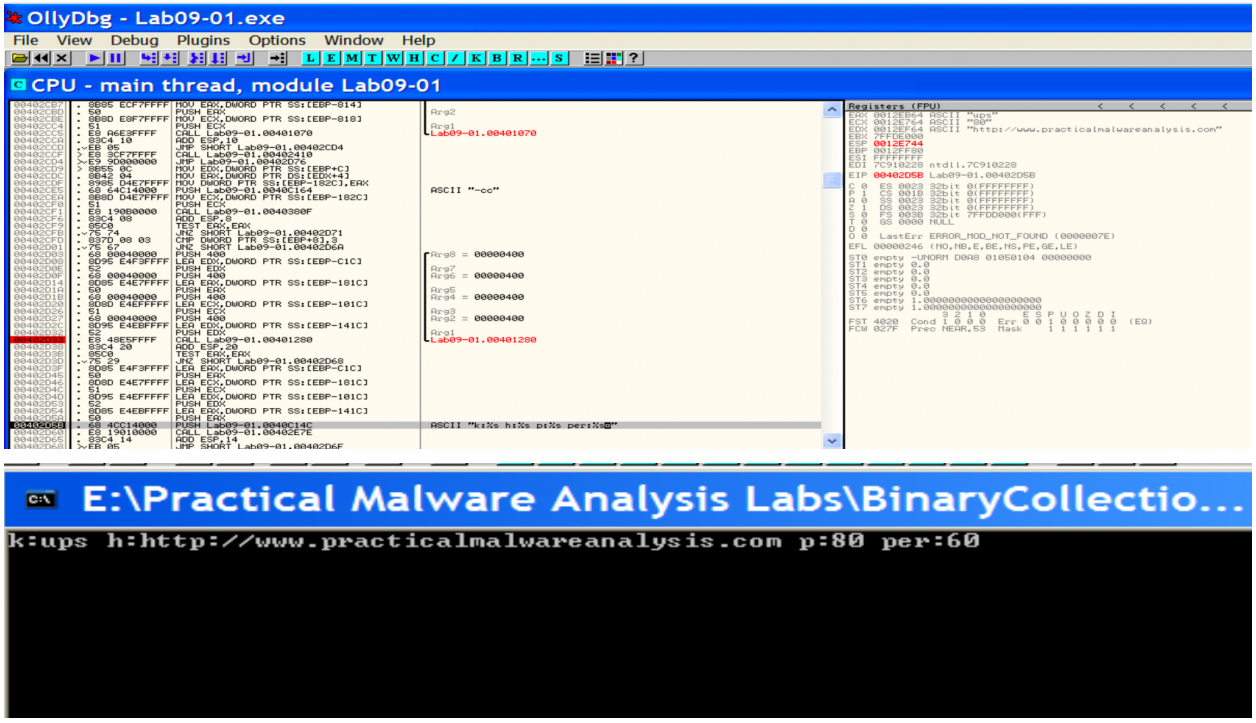
With -c abcd I again end up at the deletion, so I try with -cc



With -cc abcd I see that its looking for a key or something in software\microsoft\XPS



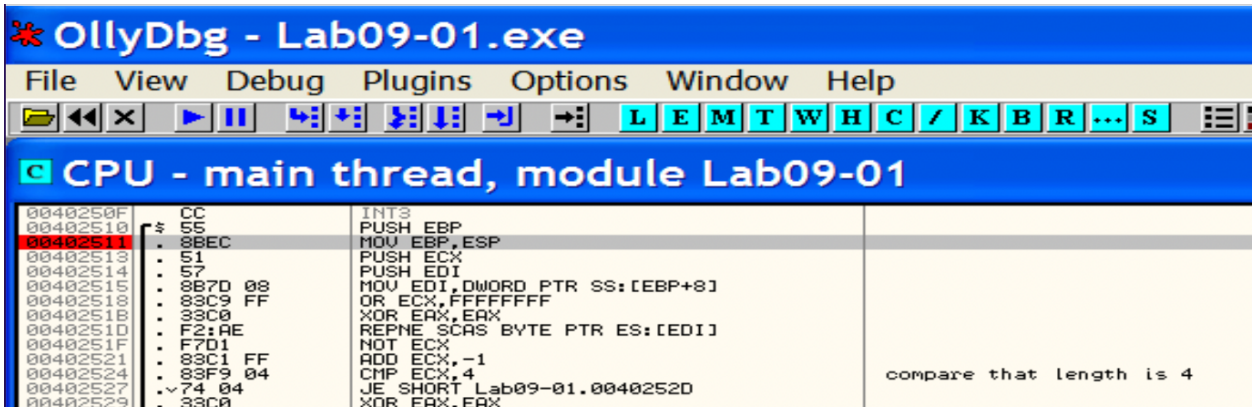
We then see that it then calls something and dumps to the program output.

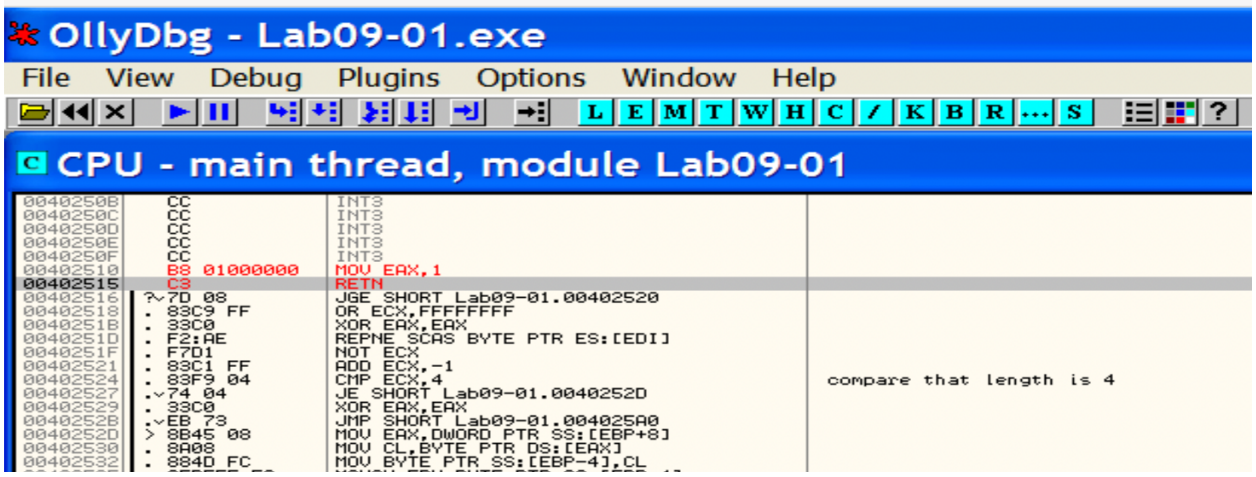


3. Like mentioned in question one, in the code that checks for the password I can change it to always return true.

```
. 33C0          XOR EAX,EAX
.v EB 05        JMP SHORT Lab09-01.004025A0
> B8 01000000   MOV EAX,1
> 5F            POP EDI
. 8BE5          MOV ESP,EBP
. 5D            POP EBP
. C3            RETN
  CC            INT3
```

I see that the command for moc eax,1 is B8 -1 00 00 00 and return is C3

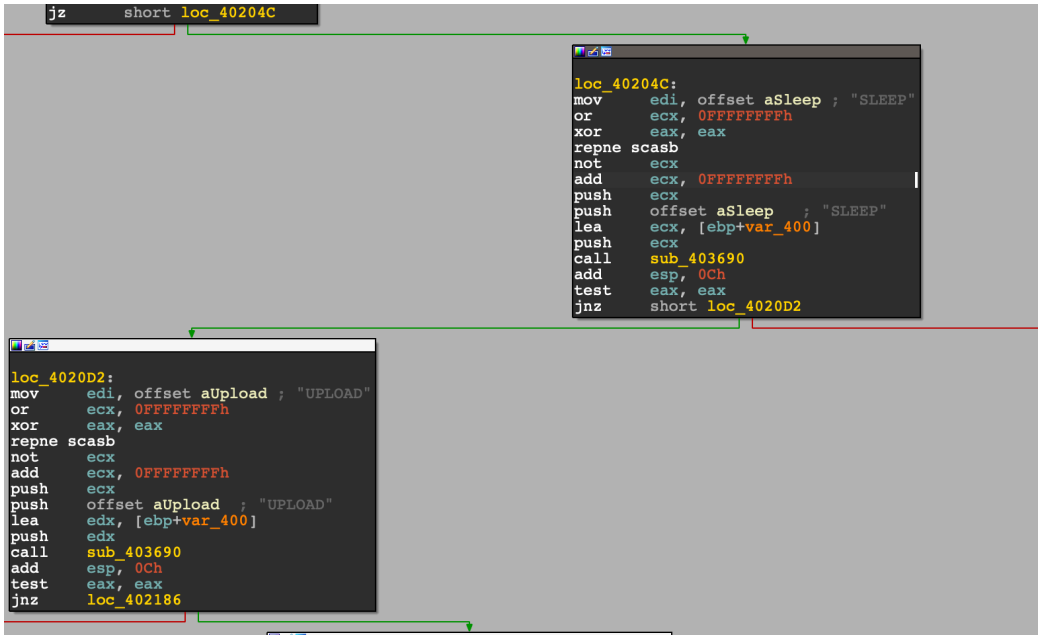


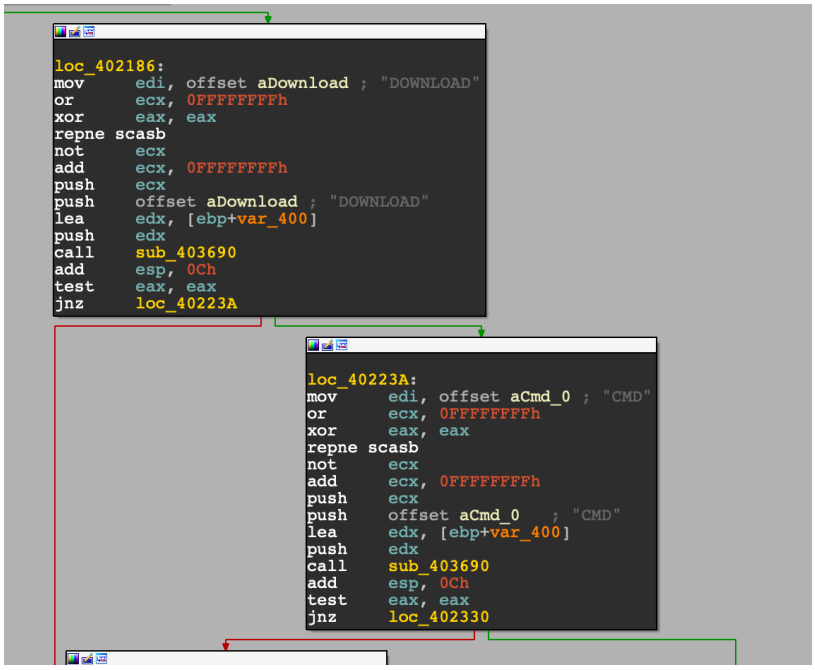


So I went to the beginning of the function and changed the first two lines to this so that it always returns 1. We then copy the patch to the executable

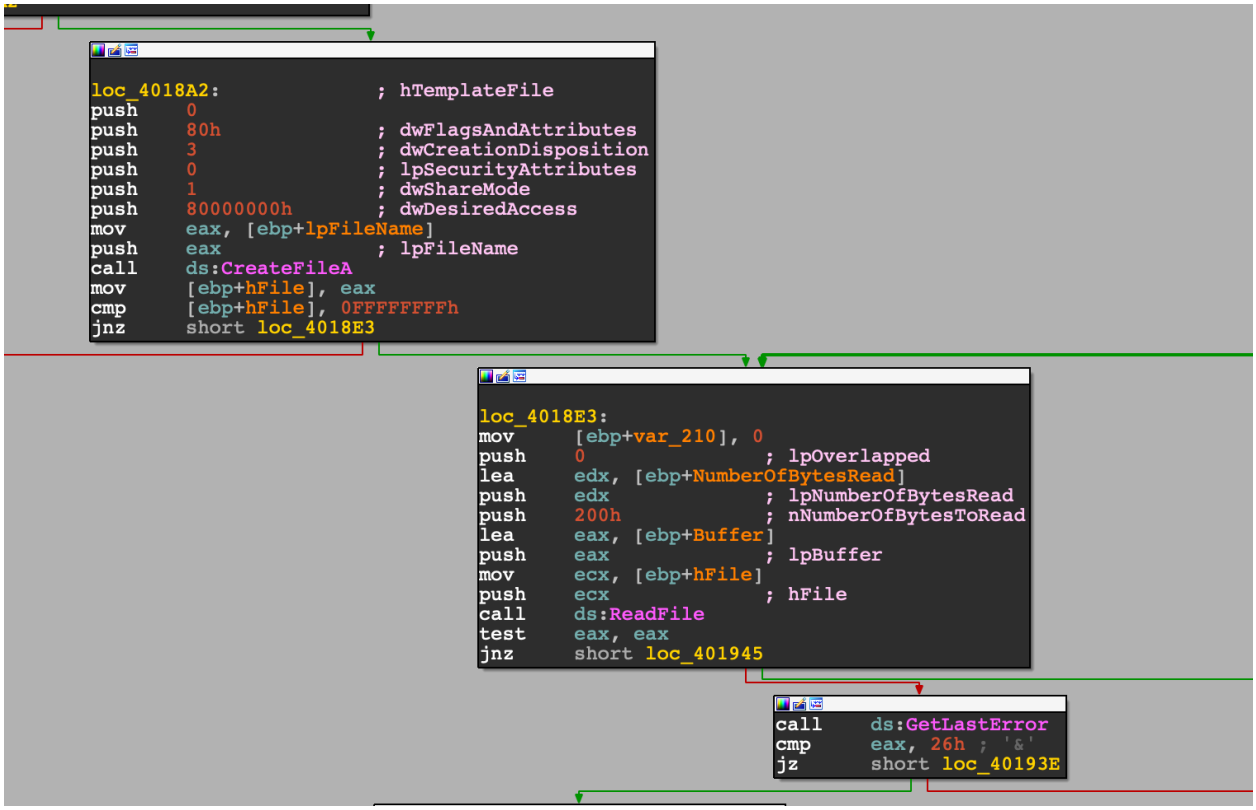
4. Host based signatures include the registry key used to store the malware configuration: at HKLM\SOFTWARE\Microsoft \XPS and also a service created called Manager Service, and also the presence of a new binary in the systemroot system32 folder.

5.

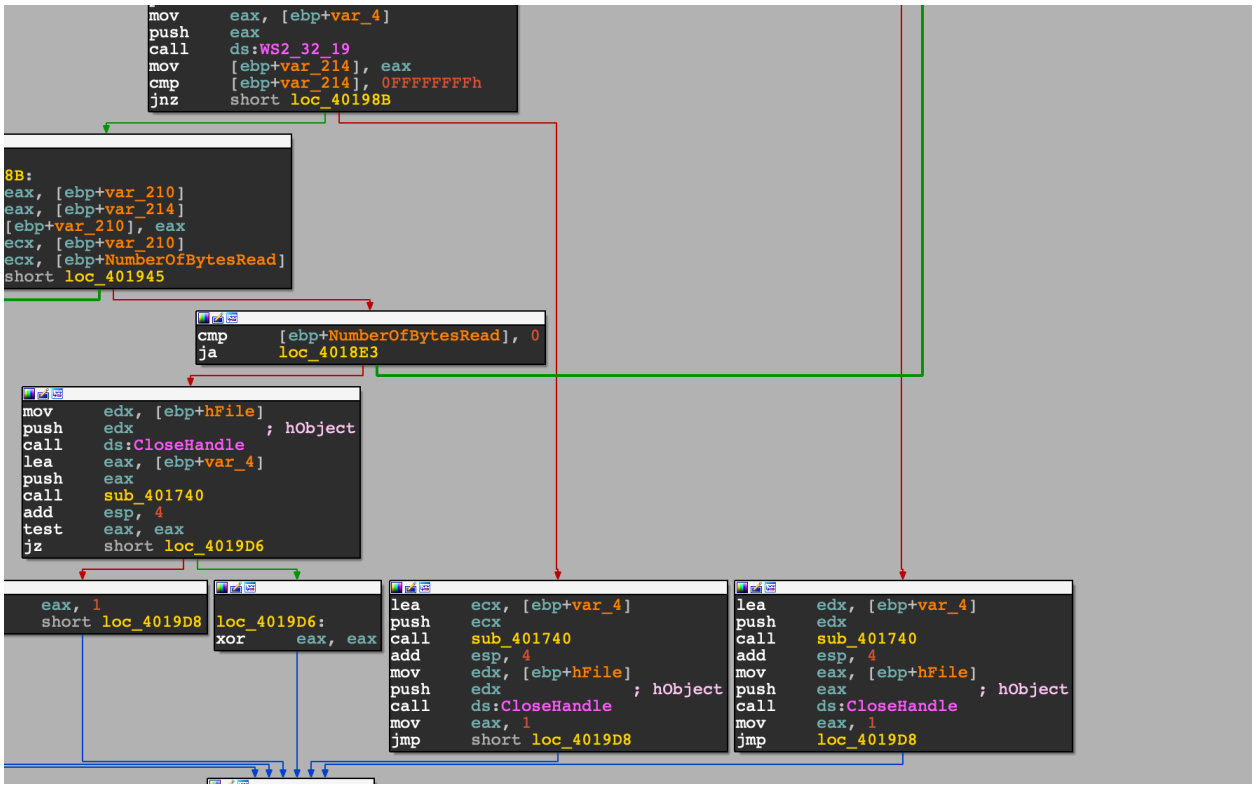




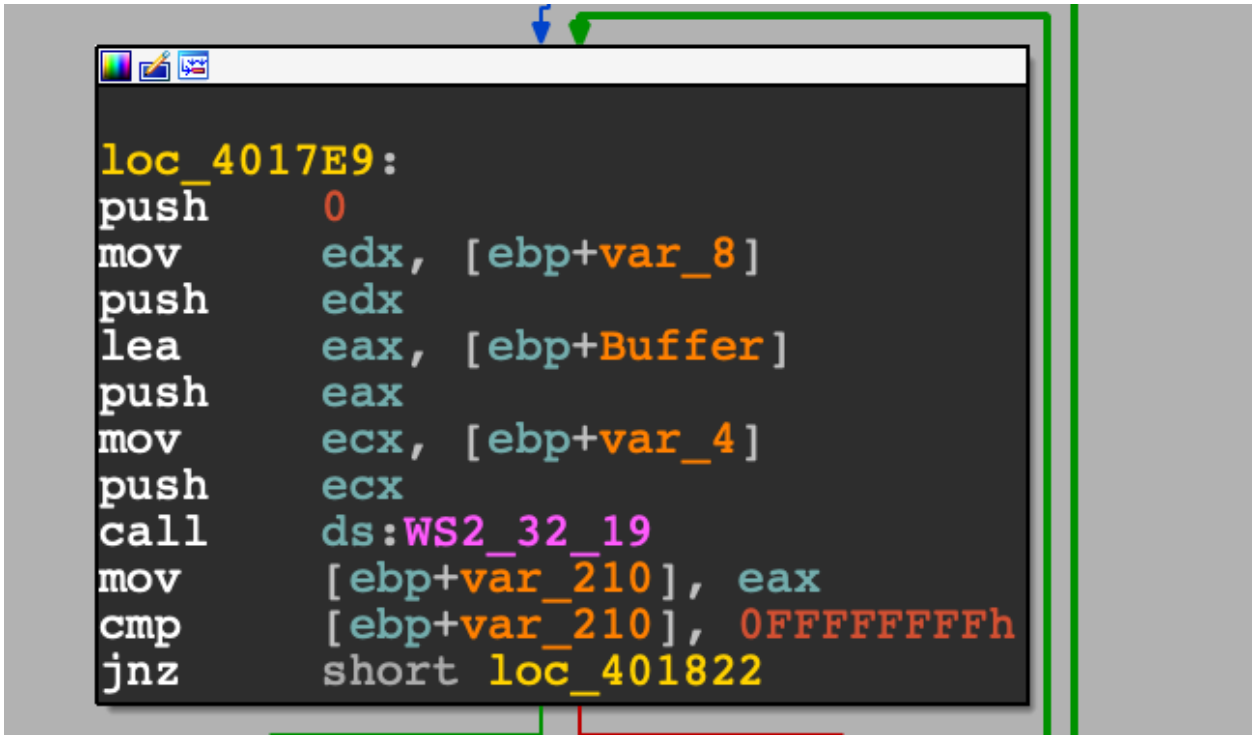
Following some subroutines in ida pro I found this routine that Calls the sleep, upload, download and cmd commands



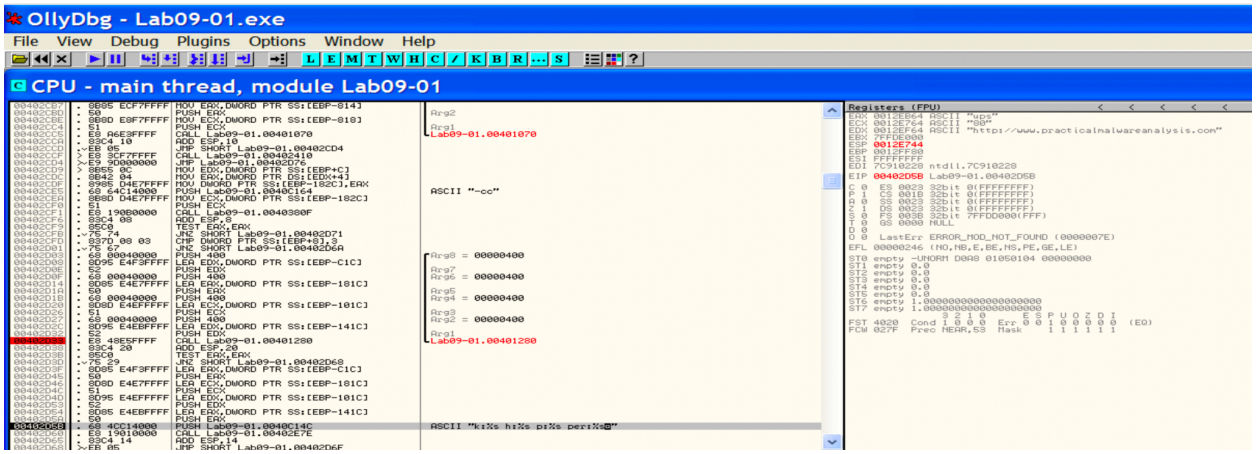
The download command leads to a subroutine that creates a file and seems to sends it over a network.



I found that the command leads to a sub routine that sends something over the network.



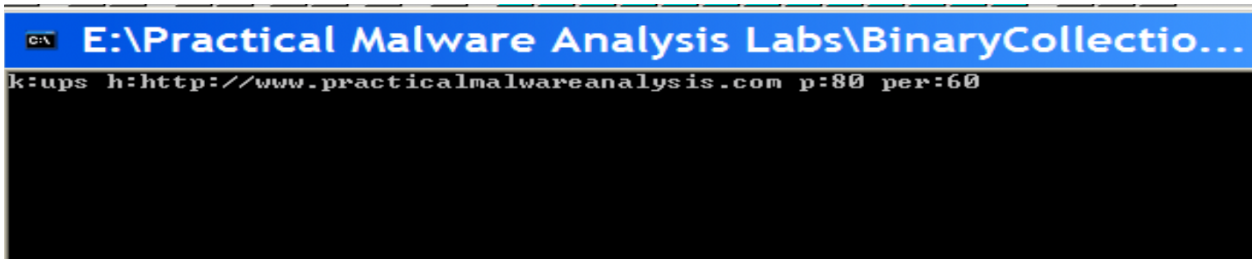
6.



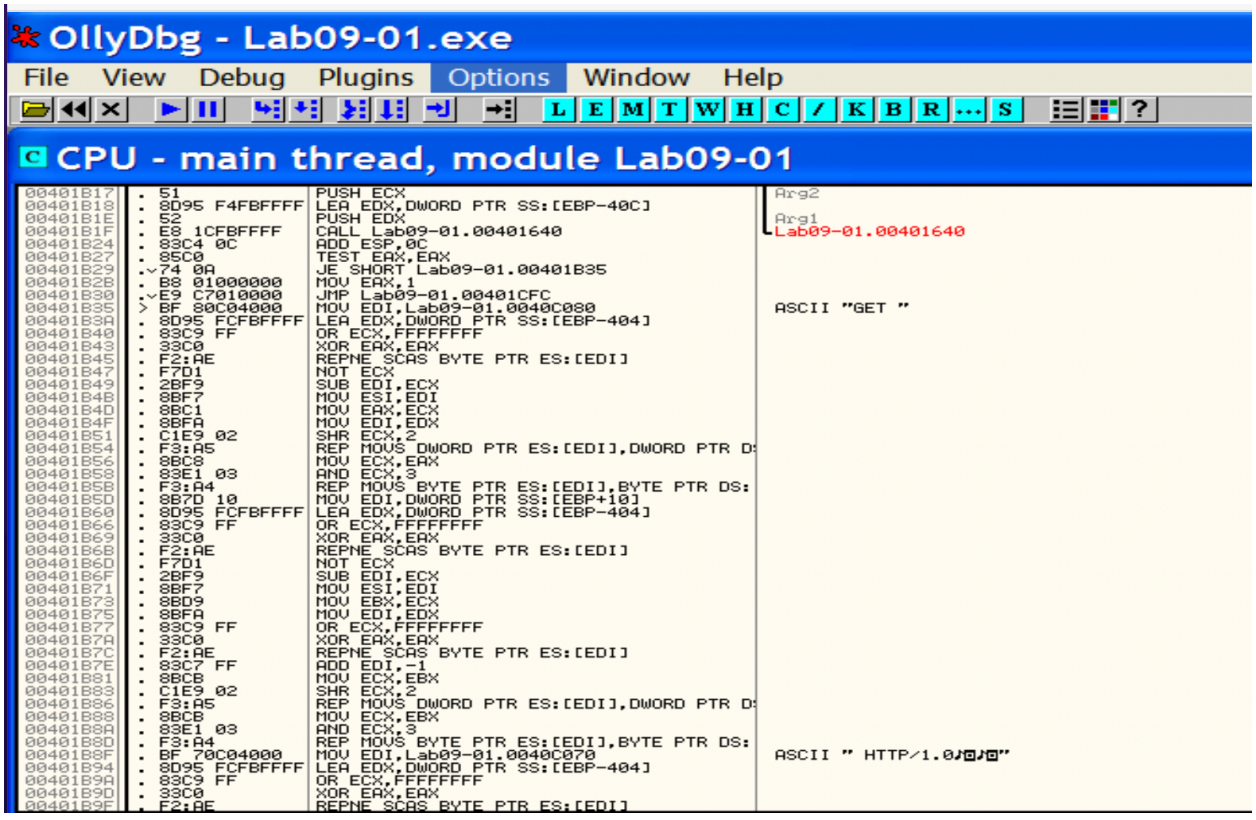
A previous screenshot shows that it communicates with www.practicalmalwareanalysis.com

```
0012BE98  0012BEAC  ASCII "http://www.practicalmalwareanalysis.com"
0012BE9C  00000400
0012BEA0  00000000
0012BEA4  00000000
0012BEA8  00000000
0012BEAC  70747468
0012BEB0  772F2F3A  WININET.772F2F3A
0012BEB4  702E7777
0012BEB8  74636172
0012BEBC  6C616369
0012BEC0  776C616D  SETUPAPI.776C616D
0012BEC4  61657261
0012BEC8  796C616E
0012BECC  2E736973
0012BED0  006D6F63
0012BED4  00000000
0012BED8  00000000
0012BEDC  00000000
0012BEE0  00000000
0012BEE4  00000000
0012BEE8  00000000
```

Along with that I found this in the program. In this screenshot from question 2,



I can see that p:80 could possibly be the port number for the connection.



I also found this “GET” command and an HTTP request in a function.

Problem 2

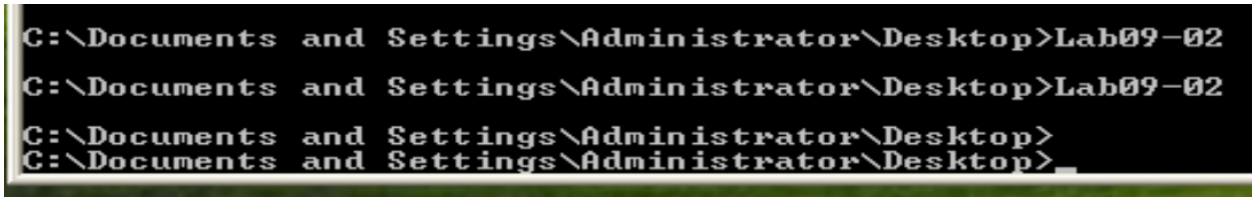
Questions

1. What strings do you see statically in the binary?
2. What happens when you run this binary?
3. How can you get this sample to run its malicious payload?
4. What is happening at 0x00401133?
5. What arguments are being passed to subroutine 0x00401089?
6. What domain name does this malware use?
7. What encoding routine is being used to obfuscate the domain name?
8. What is the significance of the `CreateProcessA` call at 0x0040106E?

1. 

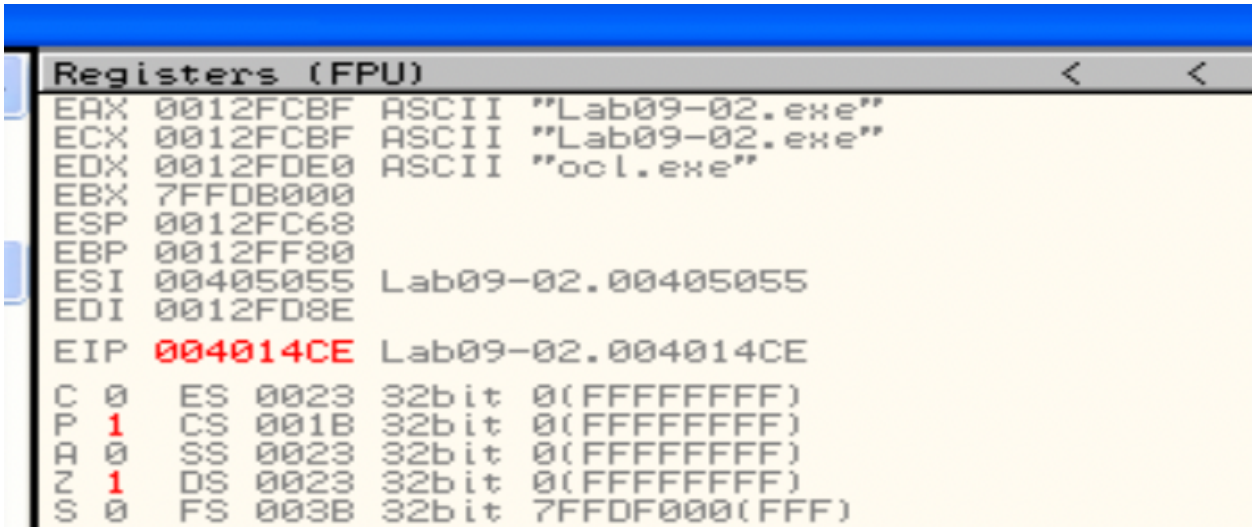
When running strings I found these. There is not much of importance except some internet functions.

2.

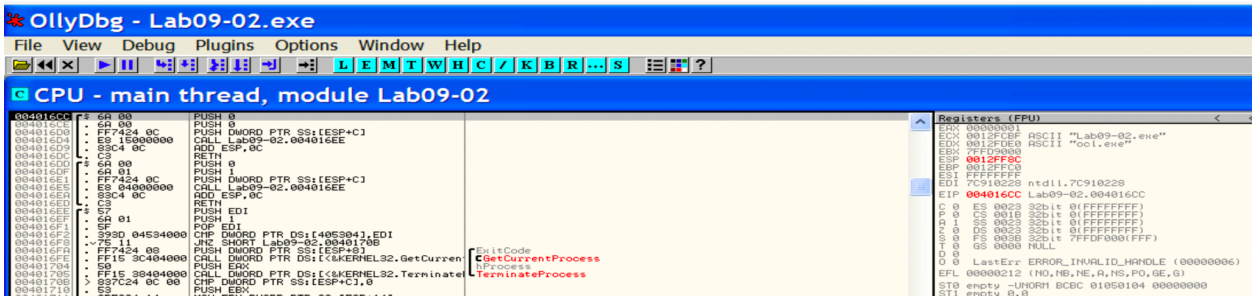


When I ran the binary it finished instantly and nothing seemed to happen

3.

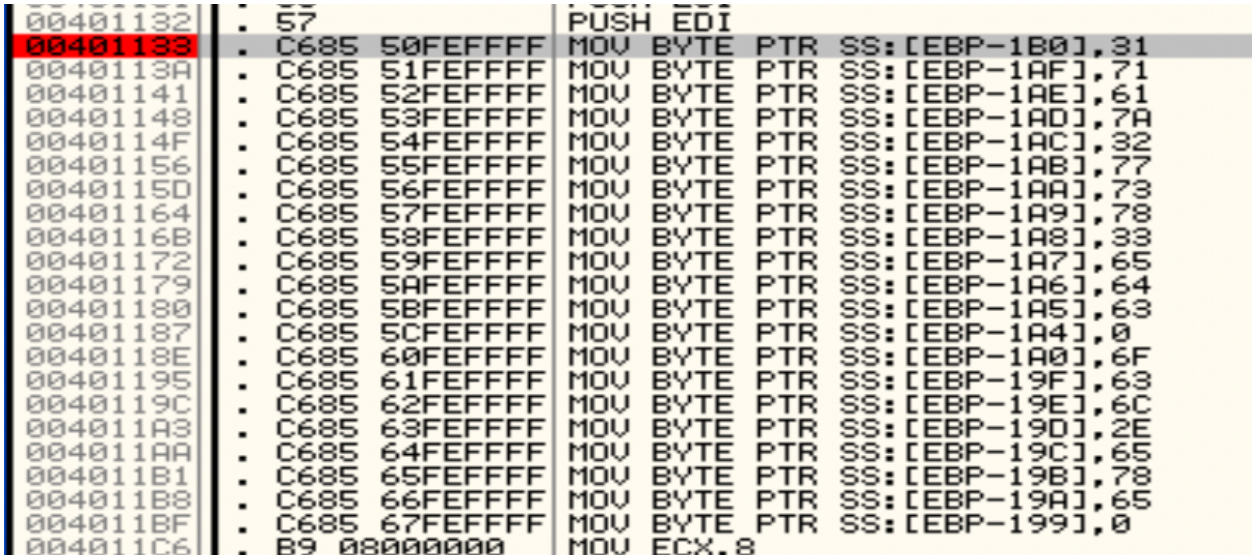


When debugging I see that it has a name called ocl.exe.



I believe that it is comparing the names and terminating if they aren't equal. So I changed the name to ocl.exe. When I debugged the code again it didn't immediately terminate so that's how I assume I can get it to deliver the payload.

4.



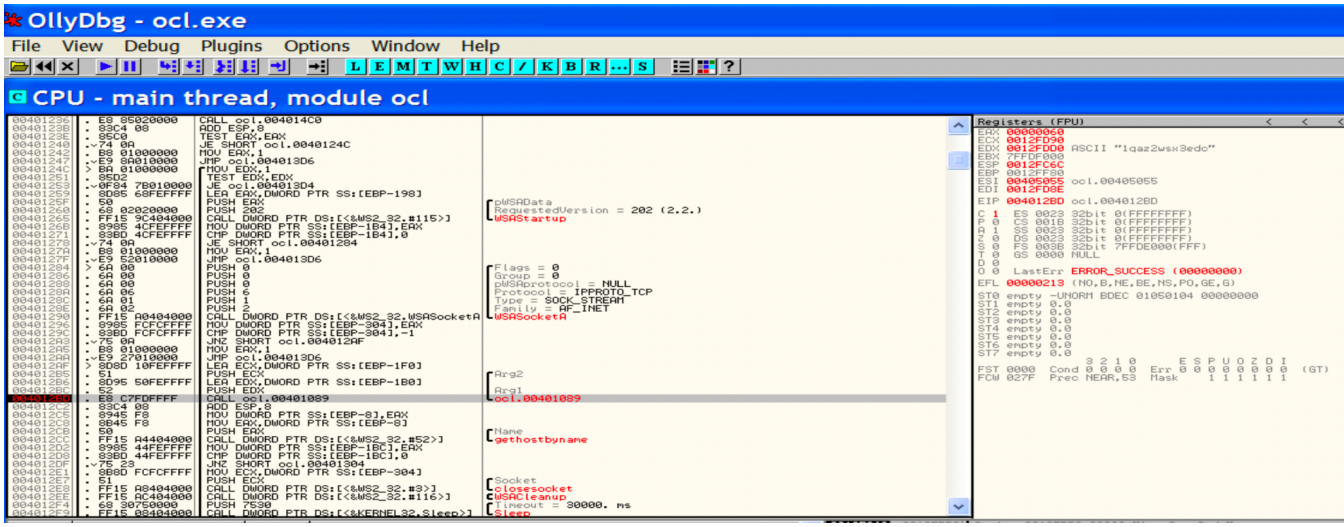
```
00401132  .  57              PUSH EDI
00401133  .  C685 50FEFFFF  MOV BYTE PTR SS:[EBP-1B0],31
0040113A  .  C685 51FEFFFF  MOV BYTE PTR SS:[EBP-1AF],71
00401141  .  C685 52FEFFFF  MOV BYTE PTR SS:[EBP-1AE],61
00401148  .  C685 53FEFFFF  MOV BYTE PTR SS:[EBP-1AD],7A
0040114F  .  C685 54FEFFFF  MOV BYTE PTR SS:[EBP-1AC],32
00401156  .  C685 55FEFFFF  MOV BYTE PTR SS:[EBP-1AB],77
0040115D  .  C685 56FEFFFF  MOV BYTE PTR SS:[EBP-1AA],73
00401164  .  C685 57FEFFFF  MOV BYTE PTR SS:[EBP-1A9],78
0040116B  .  C685 58FEFFFF  MOV BYTE PTR SS:[EBP-1A8],33
00401172  .  C685 59FEFFFF  MOV BYTE PTR SS:[EBP-1A7],65
00401179  .  C685 5AFEFFFF  MOV BYTE PTR SS:[EBP-1A6],64
00401180  .  C685 5BFEFFFF  MOV BYTE PTR SS:[EBP-1A5],63
00401187  .  C685 5CFEFFFF  MOV BYTE PTR SS:[EBP-1A4],0
0040118E  .  C685 60FEFFFF  MOV BYTE PTR SS:[EBP-1A0],6F
00401195  .  C685 61FEFFFF  MOV BYTE PTR SS:[EBP-19F],63
0040119C  .  C685 62FEFFFF  MOV BYTE PTR SS:[EBP-19E],6C
004011A3  .  C685 63FEFFFF  MOV BYTE PTR SS:[EBP-19D],2E
004011AA  .  C685 64FEFFFF  MOV BYTE PTR SS:[EBP-19C],65
004011B1  .  C685 65FEFFFF  MOV BYTE PTR SS:[EBP-19B],78
004011B8  .  C685 66FEFFFF  MOV BYTE PTR SS:[EBP-19A],65
004011BF  .  C685 67FEFFFF  MOV BYTE PTR SS:[EBP-199],0
004011C6  .  B9 08000000    MOV ECX,8
```

It seems to be moving characters I think. So I want to translate the ascii characters into letters and I get.

31 71 61 7a 32 77 73 78 33 65 64 63 = 1qaz2wsx3edc

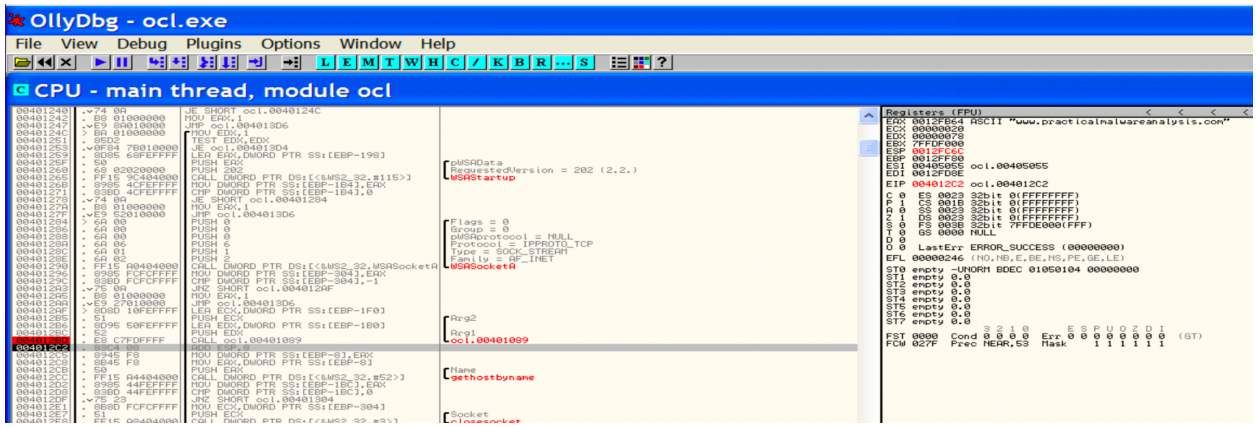
6F 63 6C 2E 65 78 65 = ocl.exe. I assume that this is an obfuscation of the malware or something like that with the name.

5.



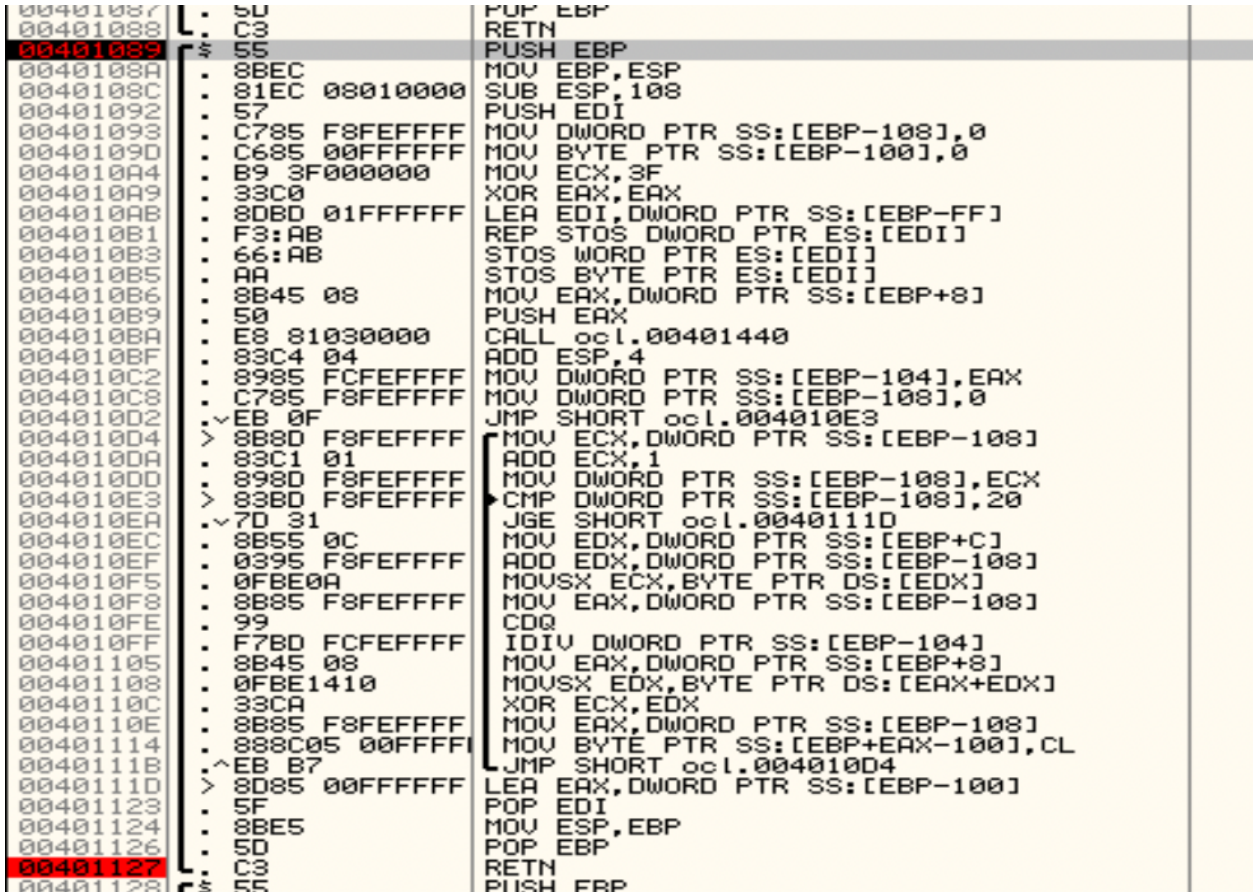
When debugging I can see that the string above 1qaz2wsx3edc is being passed in in register EDX as well as a pointer to 0012FD90 in register ECX.

6.

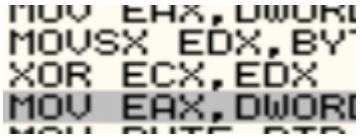


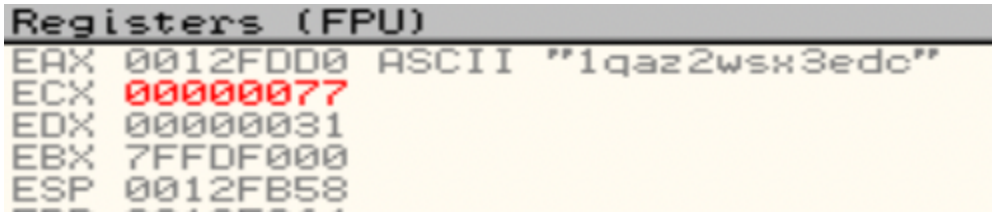
When I ran the function I see on the right that it goes through www.practicalmalwareanalysis.com

7.

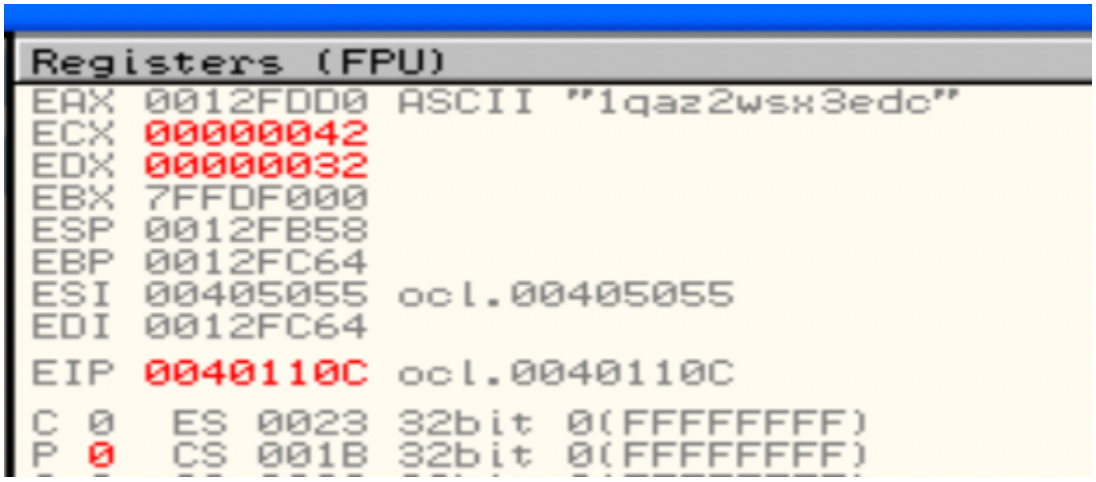


This one is a bit tricky. I started debugging the function at 401089 and found this loop in it. On line 40110C it seems to be using XOR

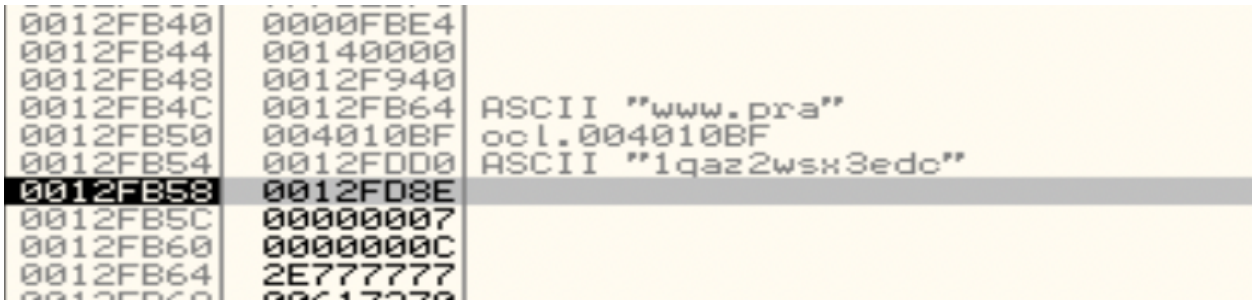




I see that the first letter is in EDX and it's being XOR'd by the number in ECX.



Here's another screenshot when it hits the '2' character (32 in ASCII). Confirming that it goes through the whole string.



I checked the memory around the pointer passed in and saw that the domain was being decrypted in real time

```
0012F940
0012FB64  ASCII "www.practica"
004010BF  ocl.004010BF
0012FDD0  ASCII "1qaz2wsx3edc"
0012FD8E
0000000C
0000000C
2E777777
63617270
61636974
00000000
```

```
0012FB48  0012F940
0012FB4C  0012FB64  ASCII "www.practicalmalwareanalysis.com"
0012FB50  004010BF  ocl.004010BF
0012FB54  0012FDD0  ASCII "1qaz2wsx3edc"
0012FB58  0012FD8E
0012FB5C  00000020
0012FB60  0000000C
0012FB64  2E777777
```

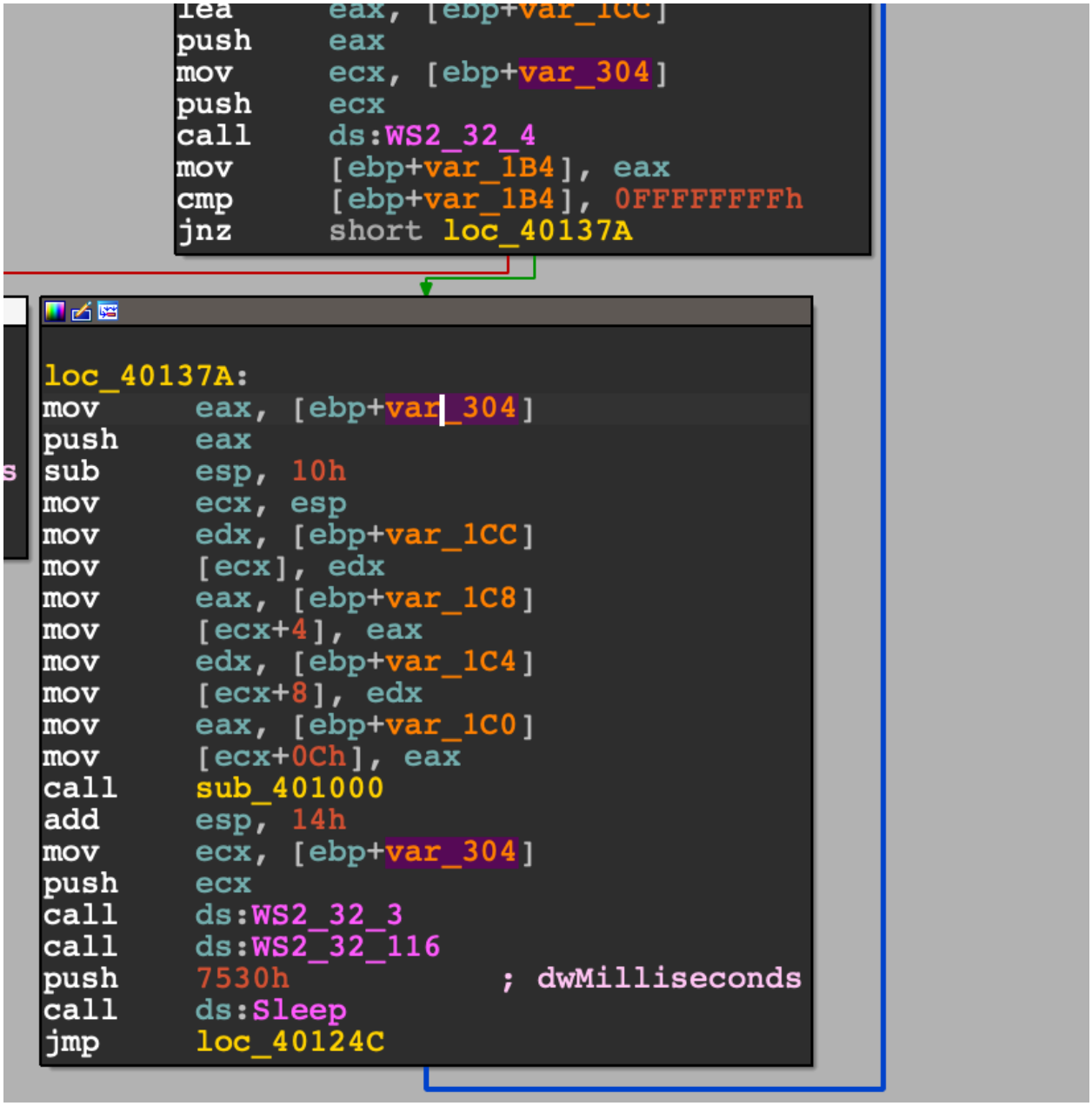
8.

```
00401034  . C745 D4 01010(  MOV DWORD PTR SS:[EBP-2C],101
0040103B  . 66:C745 D8 00(  MOV WORD PTR SS:[EBP-28],0
00401041  . 8B55 18         MOV EDX,DWORD PTR SS:[EBP+18]
00401044  . 8955 E0         MOV DWORD PTR SS:[EBP-20],EDX
00401047  . 8B45 E0         MOV EAX,DWORD PTR SS:[EBP-20]
0040104A  . 8945 E8         MOV DWORD PTR SS:[EBP-18],EAX
0040104D  . 8B4D E8         MOV ECX,DWORD PTR SS:[EBP-18]
00401050  . 894D E4         MOV DWORD PTR SS:[EBP-1C],ECX
00401053  . 8D55 F0         LEA EDX,DWORD PTR SS:[EBP-10]
00401056  . 52              PUSH EDX                                 pProcessInfo
00401057  . 8D45 A8         LEA EAX,DWORD PTR SS:[EBP-58]
0040105A  . 50              PUSH EAX                                 pStartupInfo
0040105B  . 6A 00           PUSH 0                                   CurrentDir = NULL
0040105D  . 6A 00           PUSH 0                                   pEnvironment = NULL
0040105F  . 6A 00           PUSH 0                                   CreationFlags = 0
00401061  . 6A 01           PUSH 1                                   InheritHandles = TRUE
00401063  . 6A 00           PUSH 0                                   pThreadSecurity = NULL
00401065  . 6A 00           PUSH 0                                   pProcessSecurity = NULL
00401067  . 68 30504000     PUSH ocl.00405030                        CommandLine = "cmd"
0040106C  . 6A 00           PUSH 0                                   ModuleFileName = NULL
0040106E  . FF15 04404000   CALL DWORD PTR DS:[<&KERNEL32.CreatePro  CreateProcessA
00401074  . 8945 EC         MOV DWORD PTR SS:[EBP-14],EAX
00401077  . 6A FF           PUSH -1                                  Timeout = INFINITE
00401079  . 8B4D F0         MOV ECX,DWORD PTR SS:[EBP-10]            hObject
0040107C  . 51              PUSH ECX                                 WaitForSingleObject
0040107D  . FF15 00404000   CALL DWORD PTR DS:[<&KERNEL32.WaitForSi
00401083  . 33C0            XOR EAX,EAX
00401085  . 8BE5            MOV ESP,EBP
00401087  . 5D              POP EBP
00401088  . C3              RETN
```

This function isnt called when executing the program. But the first thing I see is after the process is created an infinite loop is run.

```
push    ecx
call    sub_4013E0
add     esp, 0Ch
mov     [ebp+StartupInfo.dwFlags], 101h
mov     [ebp+StartupInfo.wShowWindow], 0
mov     edx, [ebp+arg_10]
mov     [ebp+StartupInfo.hStdInput], edx
mov     eax, [ebp+StartupInfo.hStdInput]
mov     [ebp+StartupInfo.hStdError], eax
mov     ecx, [ebp+StartupInfo.hStdError]
mov     [ebp+StartupInfo.hStdOutput], ecx
lea     edx, [ebp+ProcessInformation]
push    edx             ; lpProcessInformation
lea     eax, [ebp+StartupInfo]
push    eax             ; lpStartupInfo
push    0               ; lpCurrentDirectory
push    0               ; lpEnvironment
push    0               ; dwCreationFlags
push    1               ; bInheritHandles
push    0               ; lpThreadAttributes
push    0               ; lpProcessAttributes
push    offset CommandLine ; "cmd"
push    0               ; lpApplicationName
call    ds:CreateProcessA
mov     [ebp+var_14], eax
push    0FFFFFFFFh      ; dwMilliseconds
mov     ecx, [ebp+ProcessInformation.hProcess]
push    ecx             ; hHandle
call    ds:WaitForSingleObject
xor     eax, eax
```

I decided to go to ida pro to see better visuals around the process and I noticed that the stdinput, stderror, and stdoutput are being passed into another object before cmd.exe is run.



Going to the function that called the subroutine, I found that WS2_32_4 is the connect function. WS2_32_3 is the closesocket and WS2_32_116 is cleanup. With the connection passed into the function that creates an infinite process with stdinput, stderror, and stdoutput, I believe that this is a reverse shell that is created.