Assignment 5 Report

Group Name: Aw, Snap!

Group Members: Weijia Sun, Xinyu Lyu, Mengmei Ye

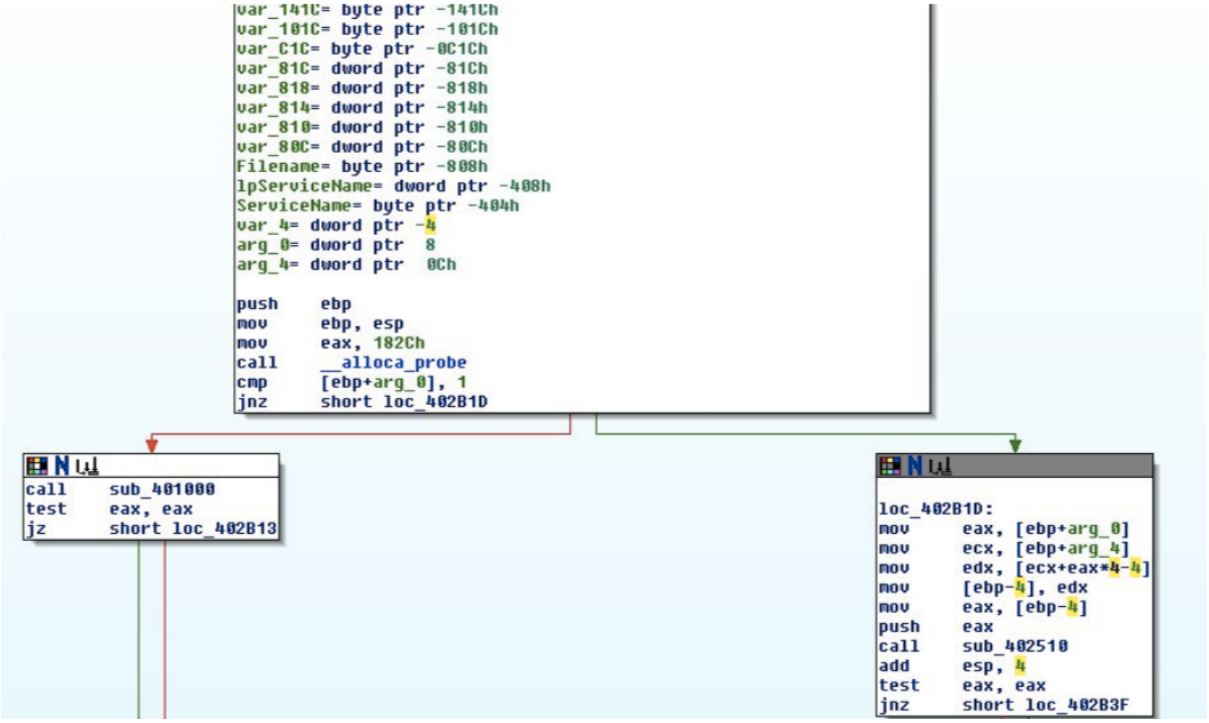
Lab09-01

Analyze the malware found in the file Lab09-01.exe using IDA Pro to answer the following questions. This malware was initially analyzed in the Chapter 3 labs using basic static and dynamic analysis techniques.

Questions

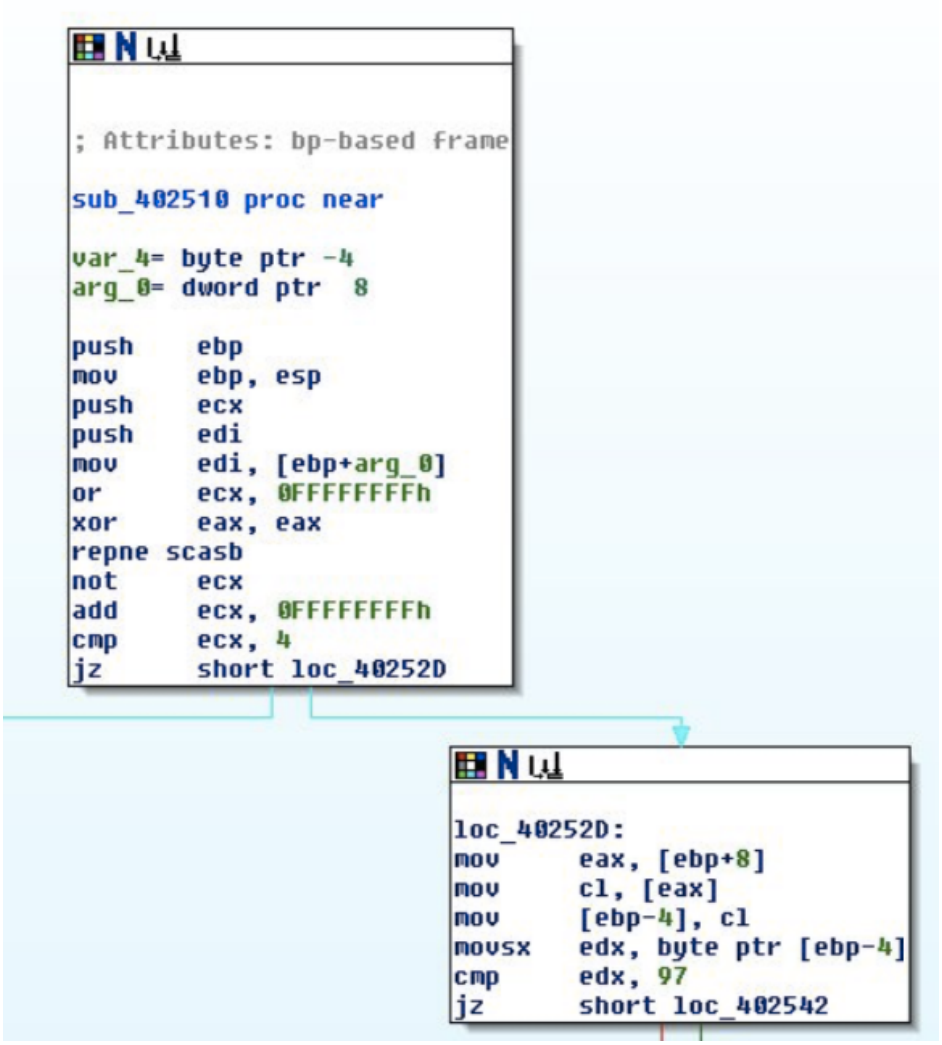
1. How can you get this malware to install itself?

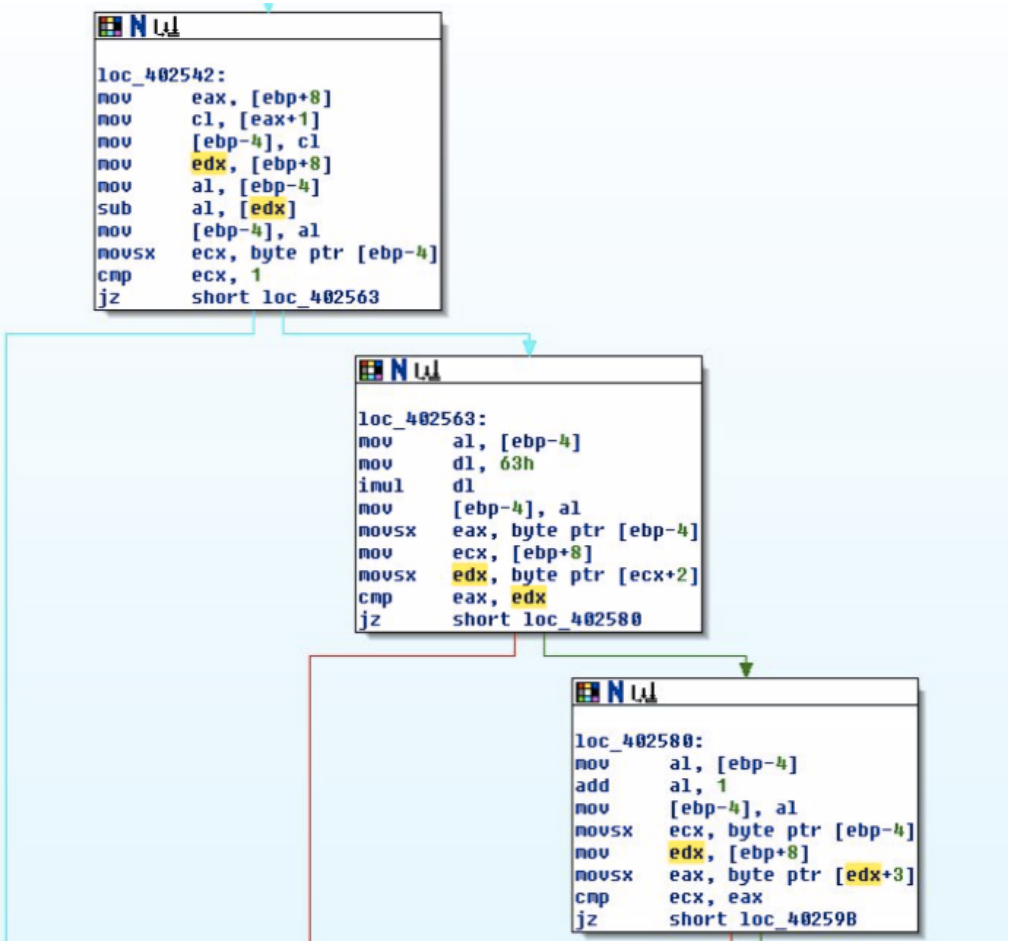
In the main function, you can see the text in edx, it pushes the text to sub402510.



Inside the sub_402510 it compares the first character with 97 which is the ACSII code of ‘a’. After this, it add the code up to 100 to get the password which is abcd.

What’s more, the program still needs more parameters to run itself. ‘-in’ might be the command of install. When type in the install command, you should take it with the password.


; Attributes: bp-based frame
sub_402510 proc near
var_4= byte ptr -4
arg_0= dword ptr 8
push ebp
mov ebp, esp
push ecx
push edi
mov edi, [ebp+arg_0]
or ecx, 0FFFFFFFFh
xor eax, eax
repne scasb
not ecx
add ecx, 0FFFFFFFFh
cmp ecx, 4
jz short loc_40252D
loc_40252D:
mov eax, [ebp+8]
mov cl, [eax]
mov [ebp-4], cl
movsx edx, byte ptr [ebp-4]
cmp edx, 97
jz short loc_402542

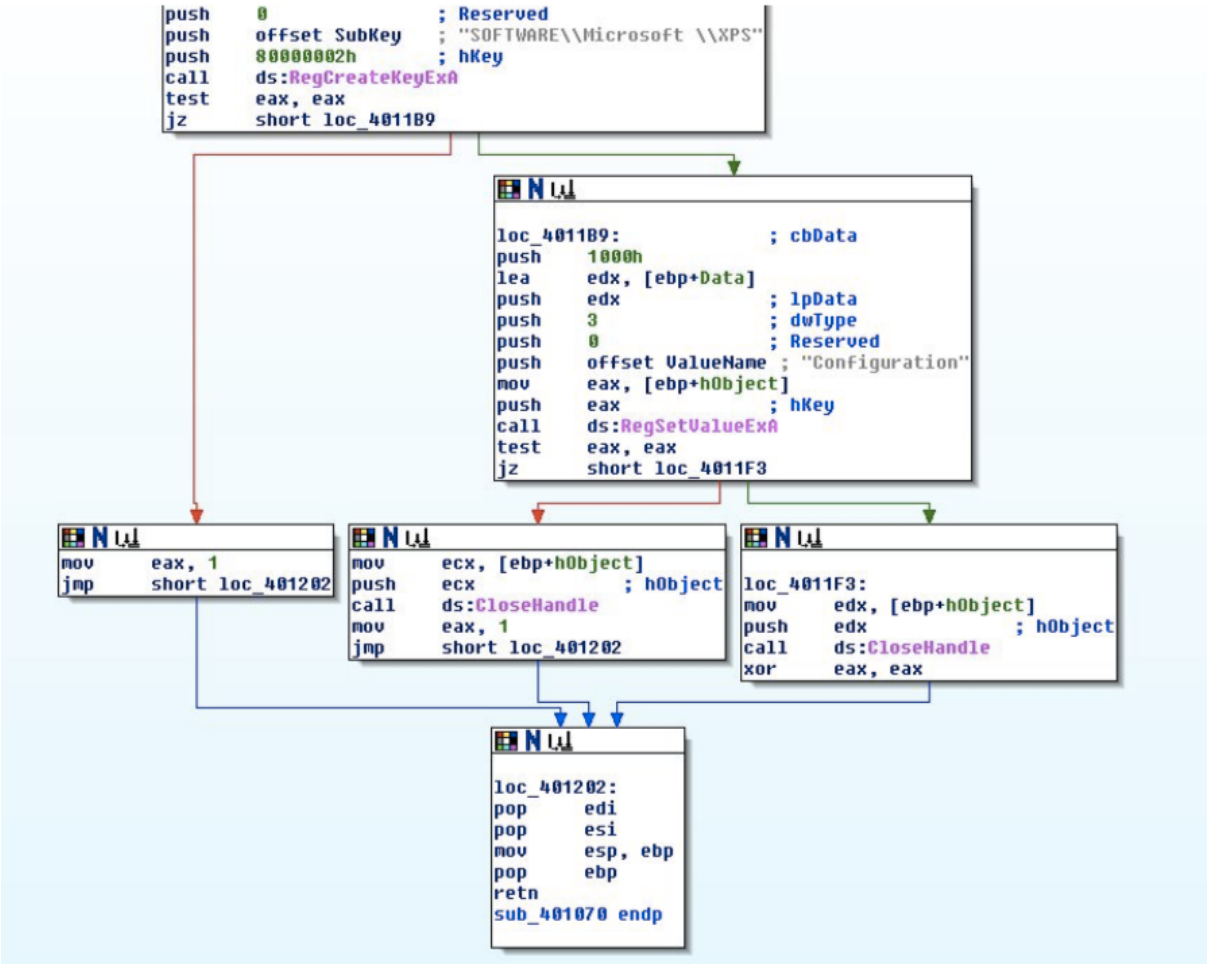


2. What are the command-line options for this program? What is the password requirement?

There are 4 kinds of command line options for this program, which are:

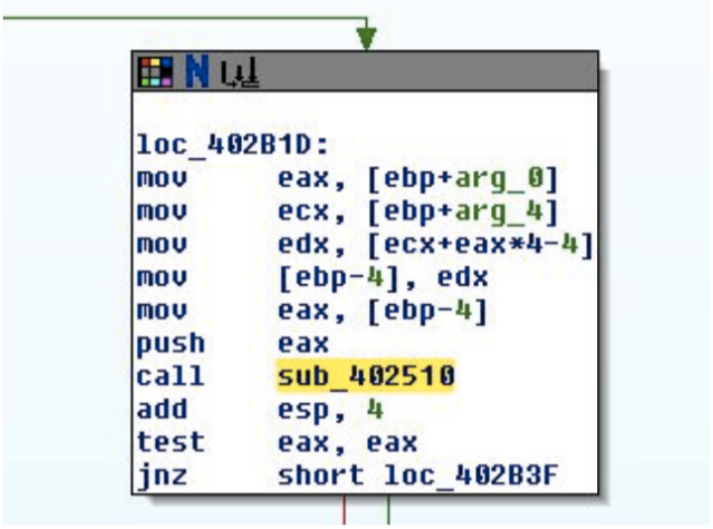
-in; install
-re; uninstall
-cc; parse registry and prints it out
-c; set Registry

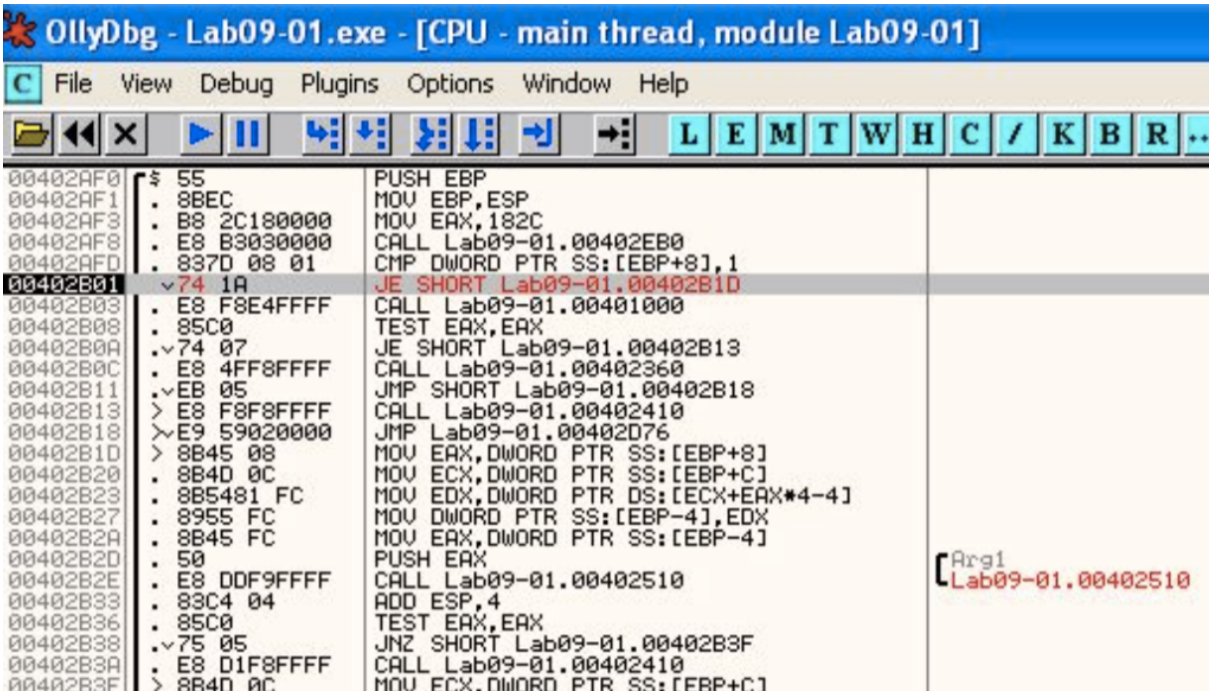
As we mentioned in the first question, the password is abcd which can be derived from sub_402510.



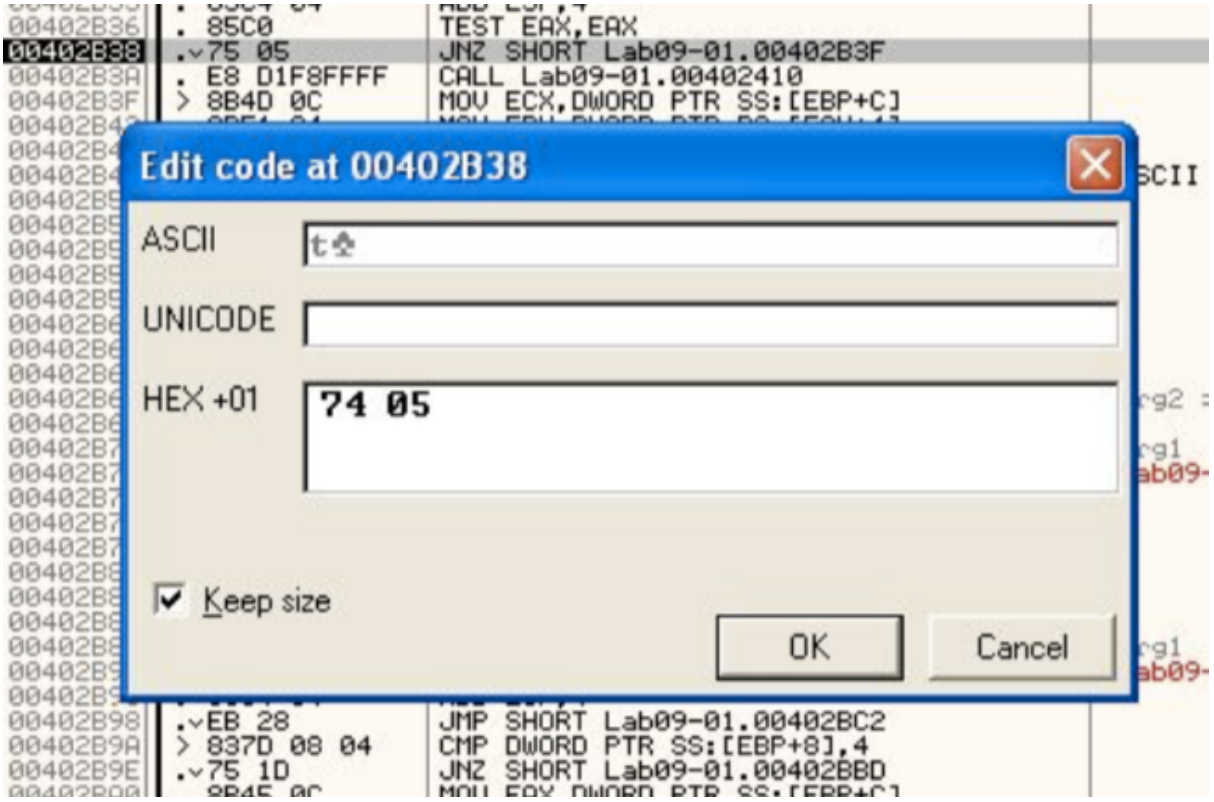
3. How can you use OllyDbg to permanently patch this malware, so that it doesn't require the special command-line password?

The password input locate in the following place. Use OllyDbg on the line.





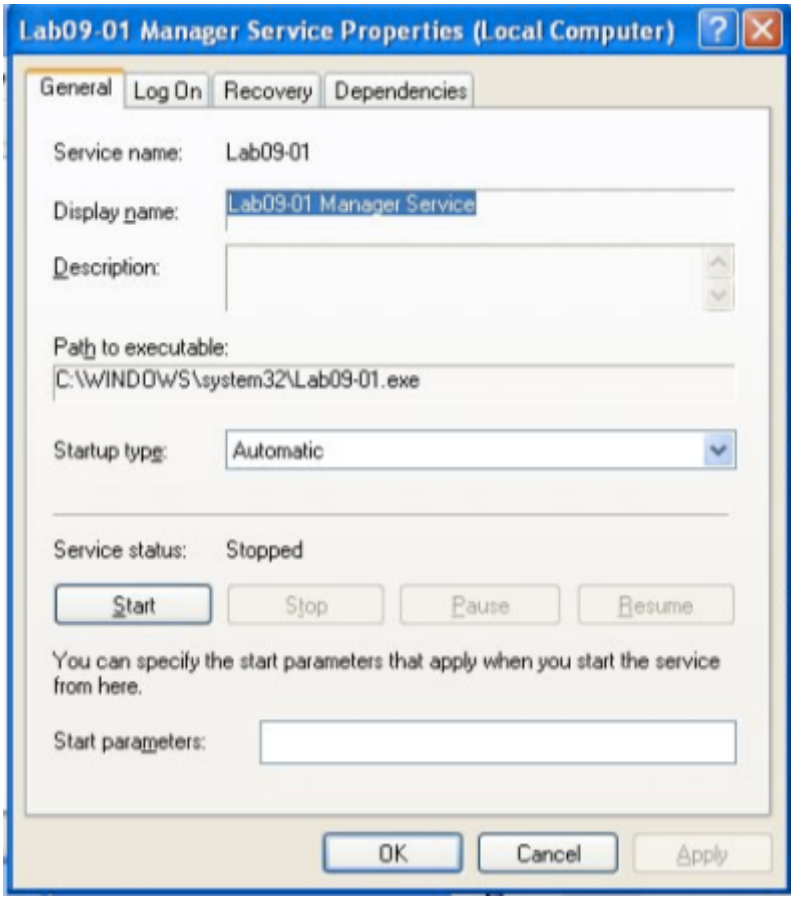
Then, we need to pass the first detection which locates in 00402B01. Modify the text from JNZ to JZ. At last, the program is lead to 00402B38. We do the same change for this code. For now, save the change, and the program doesn't require command line to install

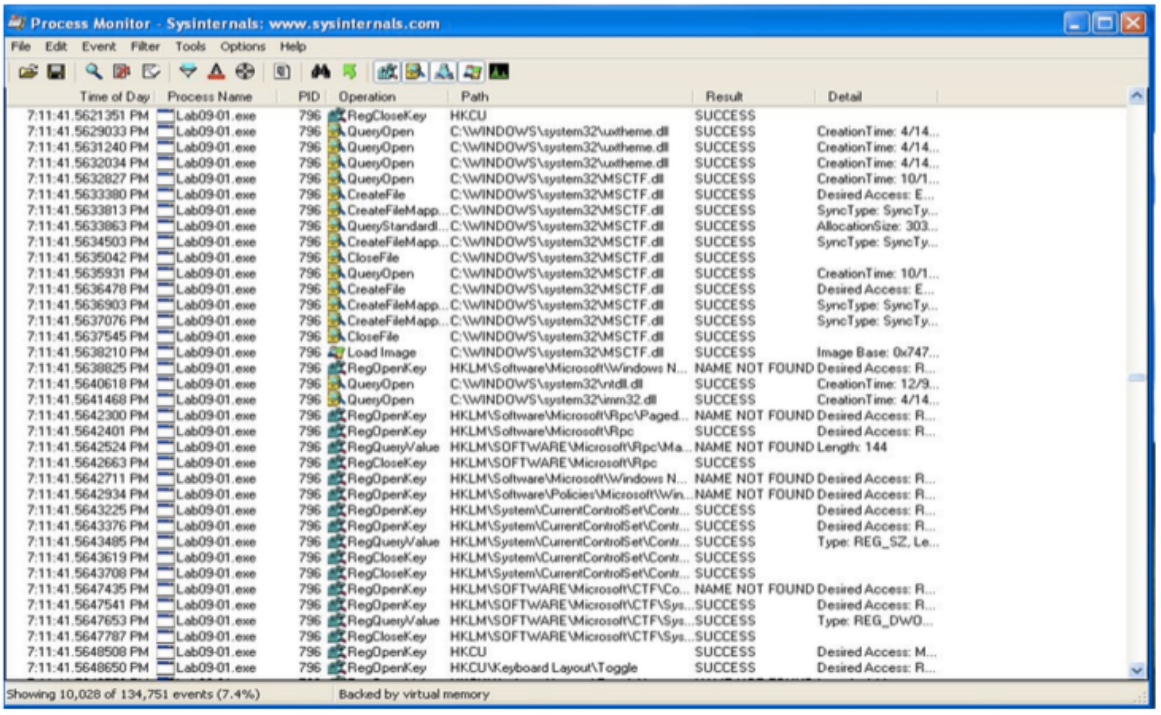


4. What are the host-based indicators of this malware?

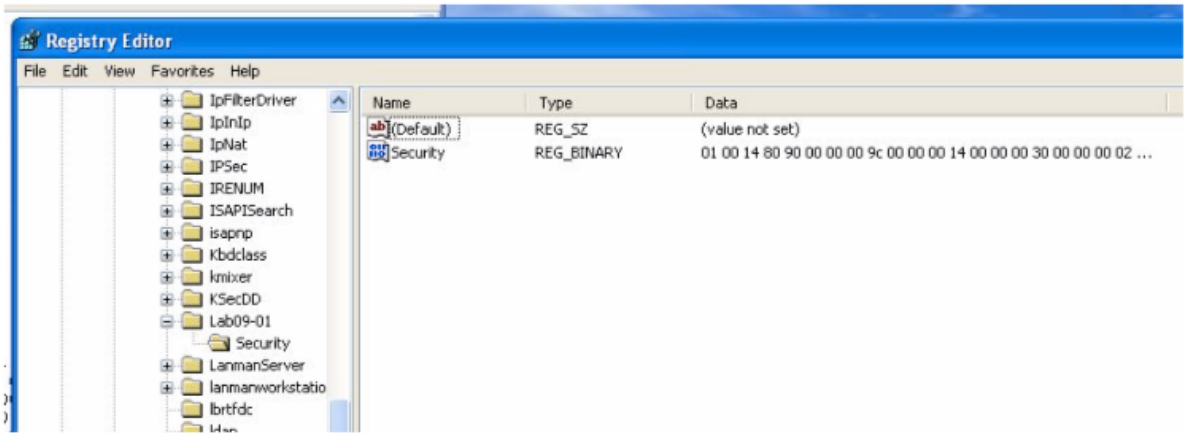
After analysis with IDA Pro in the previous questions, we noticed that the program would call register creating and modifying the functions.

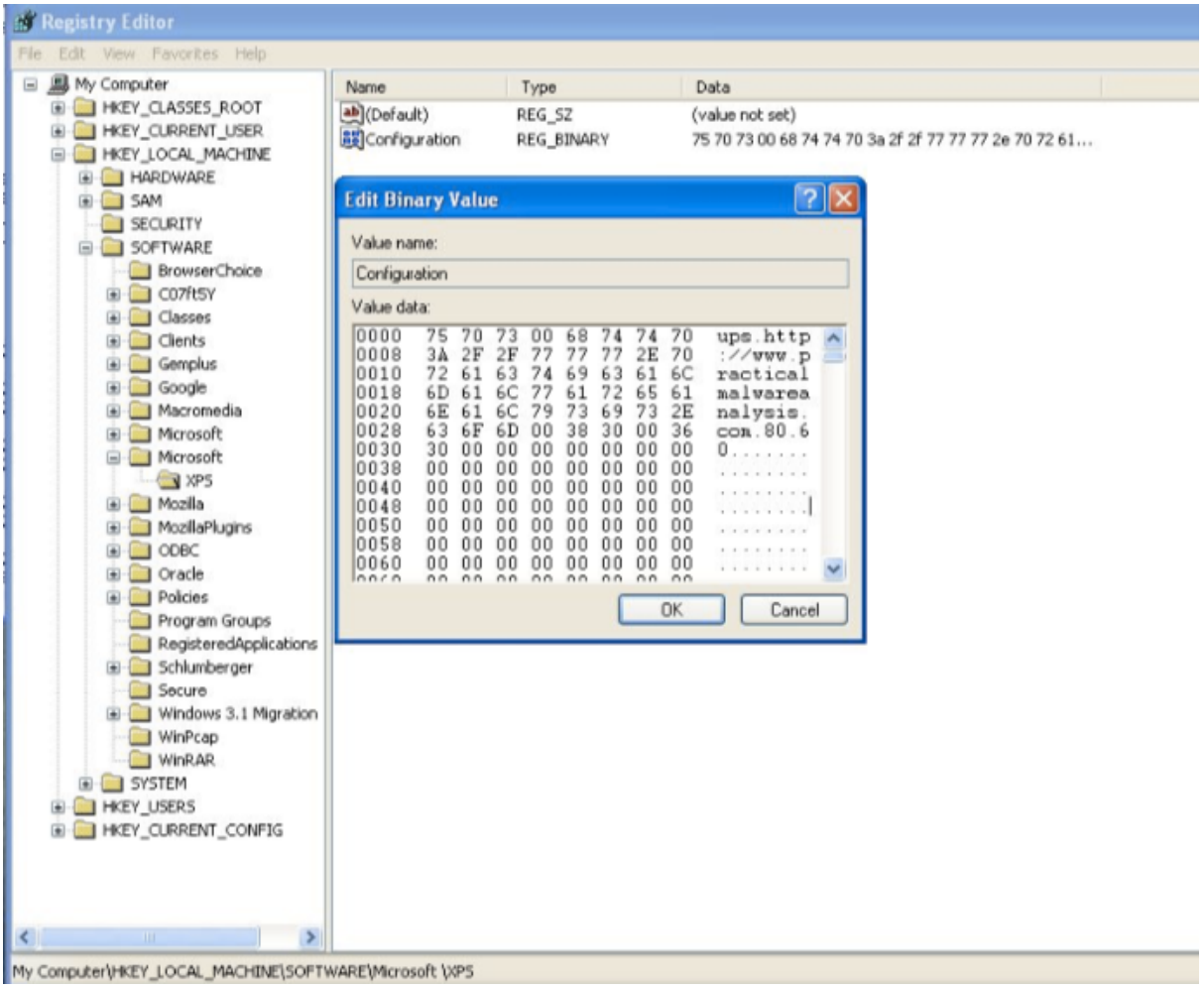
In order to verify this, we check the regshot, process monitor. The malware adds 6 changes to the register and installed a service named Lab09-01 Manager Service after running install command. The result is as follows






```
-----------------------------------
Keys added: 6
-----------------------------------
HKLM\SOFTWARE\Microsoft
HKLM\SOFTWARE\Microsoft \XPS
HKLM\SYSTEM\ControlSet001\Services\Lab09-01
HKLM\SYSTEM\ControlSet001\Services\Lab09-01\Security
HKLM\SYSTEM\CurrentControlSet\Services\Lab09-01
HKLM\SYSTEM\CurrentControlSet\Services\Lab09-01\Security
```

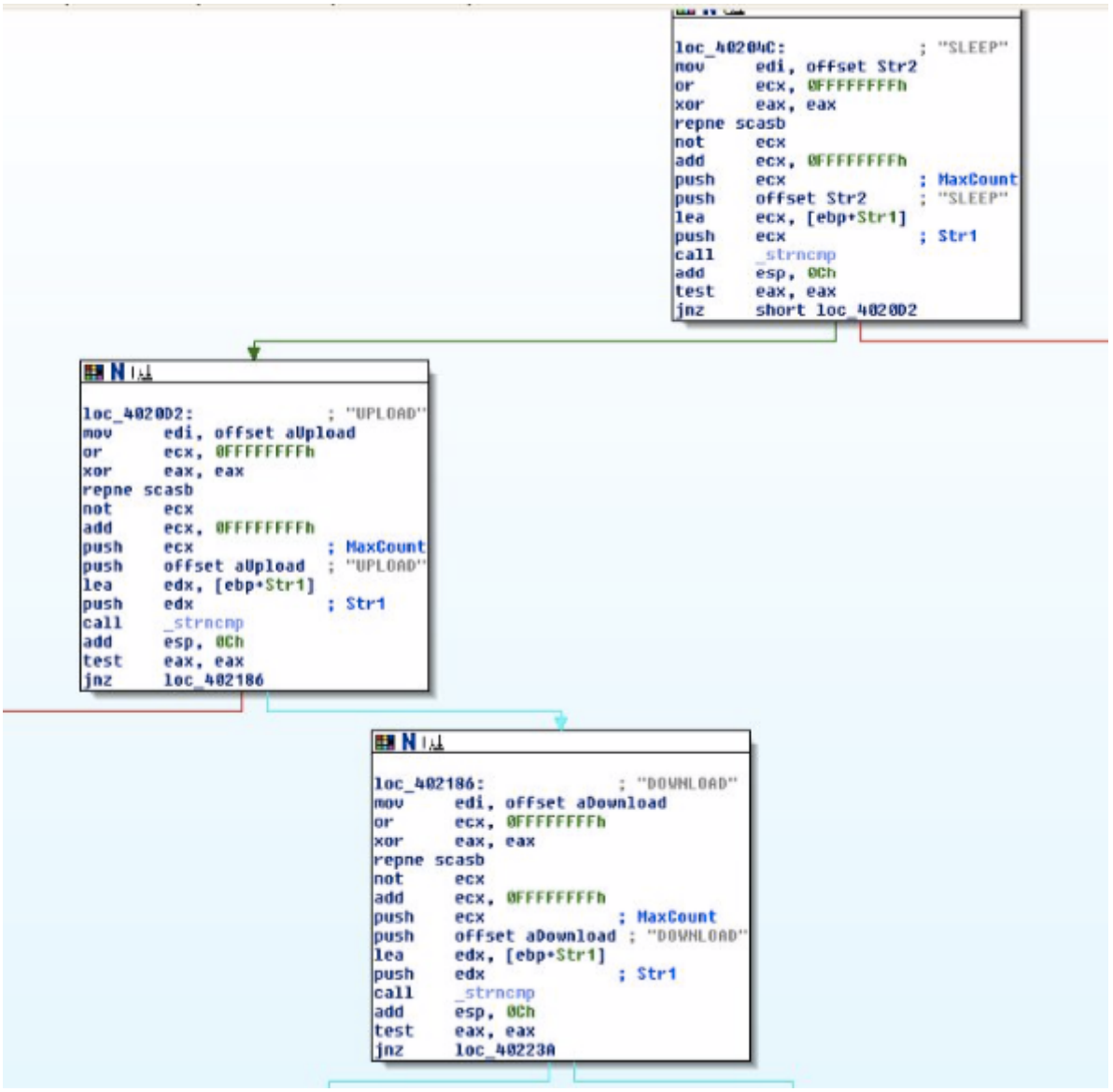




5. What are the different actions this malware can be instructed to take via the network?

If no argument is passed into the executable, the malware will call the function @0x00402360. This function will parse the registry “HKLM\\SOFTWARE\\Microsoft \\XPS\\Configuration and call function 0x00402020 to execute the malicious functions.

Analyzing the function @0x00402020, we can conclude that the malware is capable of doing the following tasks



1. Sleep
2. Upload (save a file to the victim machine)
3. Download (extract out a file from the victim machine)
4. Execute Command

6. Are there any useful network-based signatures for this malware?

After analyzing the malware, we found that the malware is trying to connect to https://practicalmalwareanalysis.com/

No.	Time	Source	Destination	Protocol	Length	Info
1	0.00000000	CadmusCo_3e:cf:05	Broadcast	ARP	42	Who has 10.0.2.2? Tell 10.0.2.15
2	0.00032500	Realteku_12:35:02	CadmusCo_3e:cf:05	ARP	60	10.0.2.2 is at 52:54:00:12:35:02
3	0.00032900	10.0.2.15	192.168.1.1	DNS	99	Standard query 0xc1aa A http://www.practicalmalwareanalysis.com
4	0.09001400	192.168.1.1	10.0.2.15	DNS	145	Standard query response 0xc1aa CNAME practicalmalwareanalysis.com A 192.0.78.24 A 192.0.78.25
5	0.09328600	10.0.2.15	192.0.78.24	TCP	62	nsstp > http [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
6	0.10278300	192.0.78.24	10.0.2.15	TCP	60	http > nsstp [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
7	0.10281600	10.0.2.15	192.0.78.24	TCP	54	nsstp > http [ACK] Seq=1 Ack=1 Win=64240 Len=0
8	0.10297800	10.0.2.15	192.0.78.24	HTTP	84	GET 7S4g/3Wh4.7yS HTTP/1.0
9	0.10323300	192.0.78.24	10.0.2.15	TCP	60	http > nsstp [ACK] Seq=1 Ack=31 Win=65535 Len=0
10	0.11116900	192.0.78.24	10.0.2.15	HTTP	1473	HTTP/1.1 400 Bad Request (text/html)
11	0.11119300	192.0.78.24	10.0.2.15	TCP	60	http > nsstp [FIN, ACK] Seq=1420 Ack=31 Win=65535 Len=0
12	0.11121400	10.0.2.15	192.0.78.24	TCP	54	nsstp > http [ACK] Seq=31 Ack=1421 Win=62821 Len=0
13	0.11176400	10.0.2.15	192.0.78.24	TCP	54	nsstp > http [FIN, ACK] Seq=31 Ack=1421 Win=62821 Len=0
14	0.11202300	192.0.78.24	10.0.2.15	TCP	60	http > nsstp [ACK] Seq=1421 Ack=32 Win=65535 Len=0

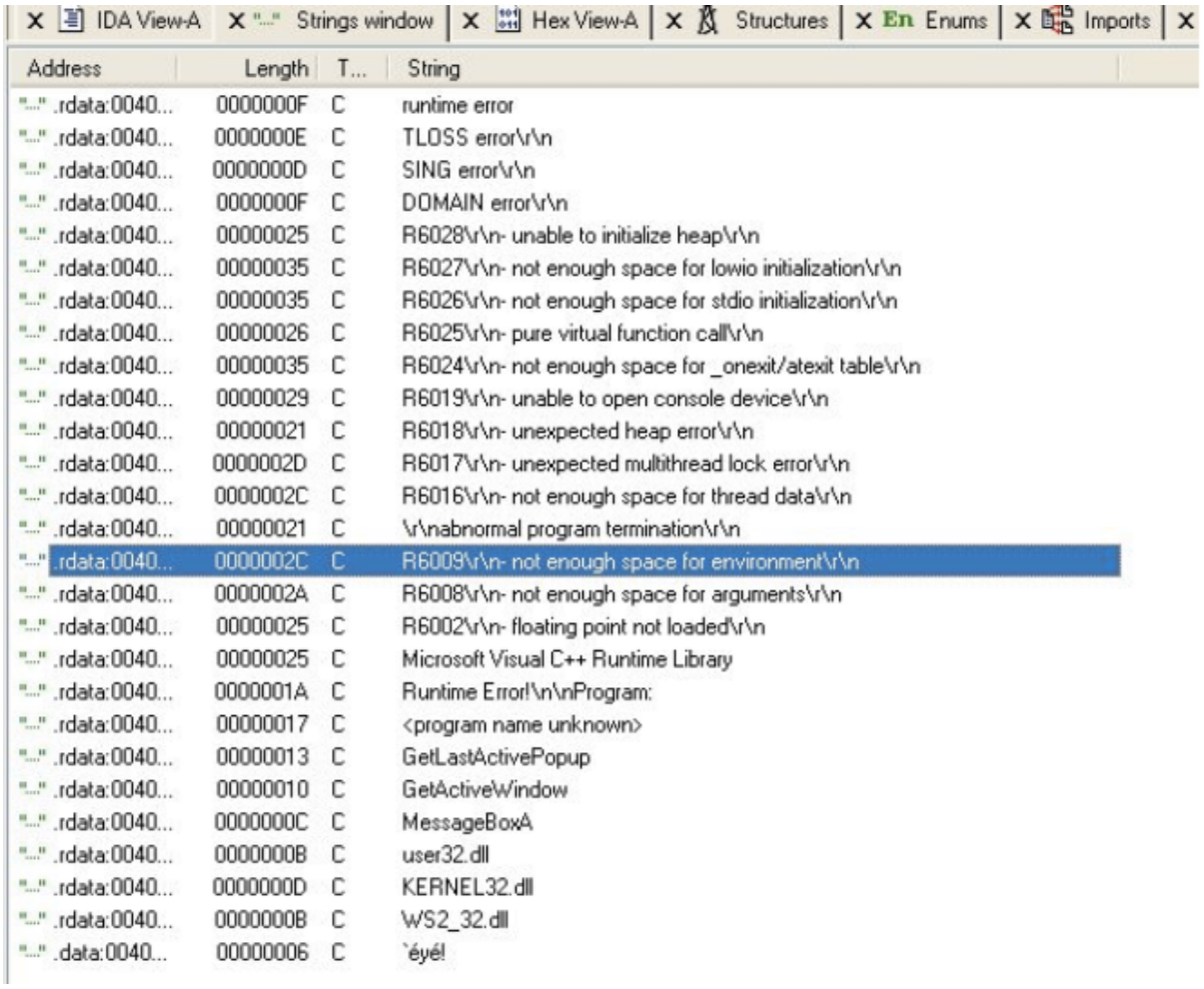
Lab09-02

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

Questions

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

We see some alert messages and text notifications as follows:

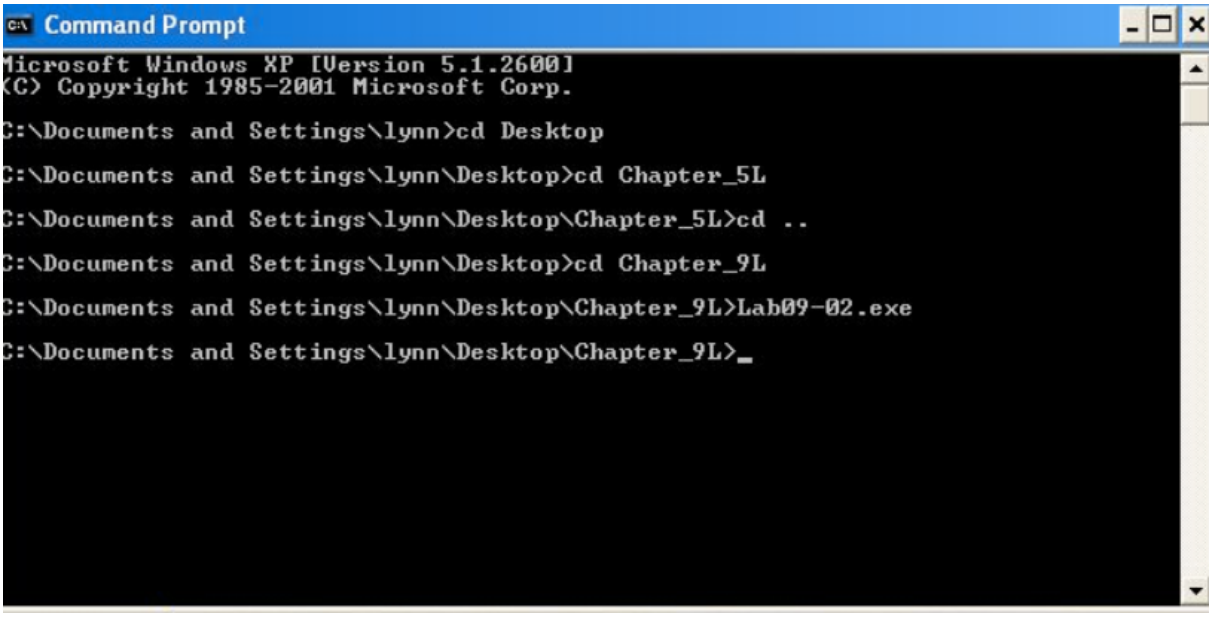


IDA View-A | Strings window | Hex View-A | Structures | Enums | Imports

Address	Length	T...	String
.rdata:0040...	0000000F	C	runtime error
.rdata:0040...	0000000E	C	TLOSS error\r\n
.rdata:0040...	0000000D	C	SING error\r\n
.rdata:0040...	0000000F	C	DOMAIN error\r\n
.rdata:0040...	00000025	C	R6028\r\n- unable to initialize heap\r\n
.rdata:0040...	00000035	C	R6027\r\n- not enough space for lowio initialization\r\n
.rdata:0040...	00000035	C	R6026\r\n- not enough space for stdio initialization\r\n
.rdata:0040...	00000026	C	R6025\r\n- pure virtual function call\r\n
.rdata:0040...	00000035	C	R6024\r\n- not enough space for _onexit/atexit table\r\n
.rdata:0040...	00000029	C	R6019\r\n- unable to open console device\r\n
.rdata:0040...	00000021	C	R6018\r\n- unexpected heap error\r\n
.rdata:0040...	0000002D	C	R6017\r\n- unexpected multithread lock error\r\n
.rdata:0040...	0000002C	C	R6016\r\n- not enough space for thread data\r\n
.rdata:0040...	00000021	C	\r\nabnormal program termination\r\n
.rdata:0040...	0000002C	C	R6009\r\n- not enough space for environment\r\n
.rdata:0040...	0000002A	C	R6008\r\n- not enough space for arguments\r\n
.rdata:0040...	00000025	C	R6002\r\n- floating point not loaded\r\n
.rdata:0040...	00000025	C	Microsoft Visual C++ Runtime Library
.rdata:0040...	0000001A	C	Runtime Error!\n\nProgram:
.rdata:0040...	00000017	C	<program name unknown>
.rdata:0040...	00000013	C	GetLastActivePopup
.rdata:0040...	00000010	C	GetActiveWindow
.rdata:0040...	0000000C	C	MessageBoxA
.rdata:0040...	0000000B	C	user32.dll
.rdata:0040...	0000000D	C	KERNEL32.dll
.rdata:0040...	0000000B	C	WS2_32.dll
.data:0040...	00000006	C	`éyé!

2. What happens when you run this binary?

As follow. Terminate without doing anything.



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

The malware compares the module file name with the letter of its alphabet. It then strip the path using _strrchr. The malware then compares the filename with “ocl.exe”.

```
rep movsd
movsb
mov     [ebp+var_1B8], 0
mov     [ebp+Str], 0
mov     ecx, 43h
xor     eax, eax
lea     edi, [ebp+var_2FF]
rep stosd
stosb
push    10Eh            ; nSize
lea     eax, [ebp+Str]
push    eax             ; lpFilename
push    0               ; hModule
call    ds:GetModuleFileNameA
push    5Ch             ; Ch
lea     ecx, [ebp+Str]
push    ecx             ; Str
call    _strrchr
add     esp, 8
mov     [ebp+Str2], eax
mov     edx, [ebp+Str2]
add     edx, 1
mov     [ebp+Str2], edx
mov     eax, [ebp+Str2]
push    eax             ; Str2
lea     ecx, [ebp+Str1]
push    ecx             ; Str1
call    _strcmp
add     esp, 8
test    eax, eax
jz      short loc_40124C
```

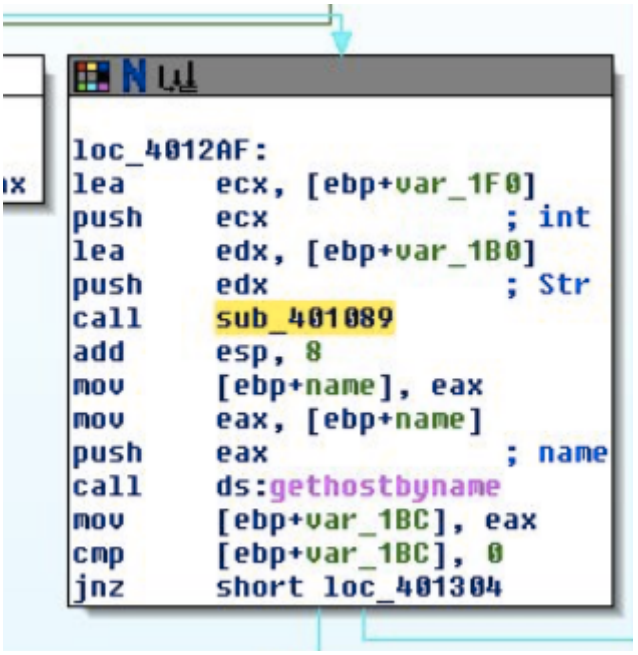
```
push    edi
mov     [ebp+var_1B0], '1'
mov     [ebp+var_1AF], 'q'
mov     [ebp+var_1AE], 'a'
mov     [ebp+var_1AD], 'z'
mov     [ebp+var_1AC], '2'
mov     [ebp+var_1AB], 'w'
mov     [ebp+var_1AA], 's'
mov     [ebp+var_1A9], 'x'
mov     [ebp+var_1A8], '3'
mov     [ebp+var_1A7], 'e'
mov     [ebp+var_1A6], 'd'
mov     [ebp+var_1A5], 'c'
mov     [ebp+var_1A4], 0
mov     [ebp+Str1], 'o'
mov     [ebp+var_19F], 'c'
mov     [ebp+var_19E], 'l'
mov     [ebp+var_19D], '.'
mov     [ebp+var_19C], 'e'
mov     [ebp+var_19B], 'x'
mov     [ebp+var_19A], 'e'
mov     [ebp+var_199], 0
```

4. What is happening at 0x00401133?

The malware generates some word to prevent the IDA pro detecting the normal string which is ‘1qaz2wsx3edc’

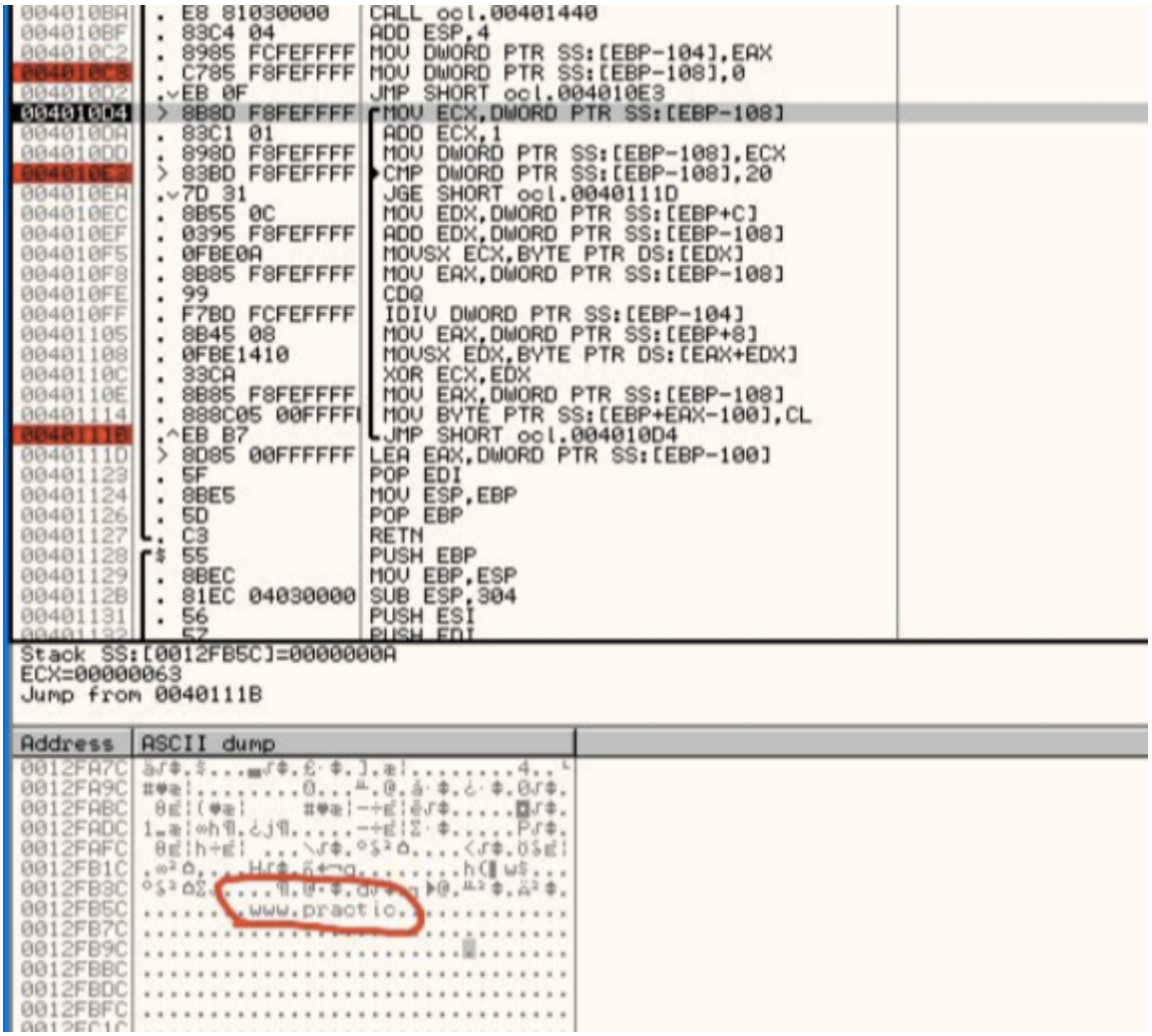
```
push    edi
mov     [ebp+var_1B0], '1'
mov     [ebp+var_1AF], 'q'
mov     [ebp+var_1AE], 'a'
mov     [ebp+var_1AD], 'z'
mov     [ebp+var_1AC], '2'
mov     [ebp+var_1AB], 'w'
mov     [ebp+var_1AA], 's'
mov     [ebp+var_1A9], 'x'
mov     [ebp+var_1A8], '3'
mov     [ebp+var_1A7], 'e'
mov     [ebp+var_1A6], 'd'
mov     [ebp+var_1A5], 'c'
mov     [ebp+var_1A4], 0
mov     [ebp+Str1], 'o'
mov     [ebp+var_19F], 'c'
mov     [ebp+var_19E], 'l'
mov     [ebp+var_19D], '.'
mov     [ebp+var_19C], 'e'
mov     [ebp+var_19B], 'x'
mov     [ebp+var_19A], 'e'
mov     [ebp+var_199], 0
```

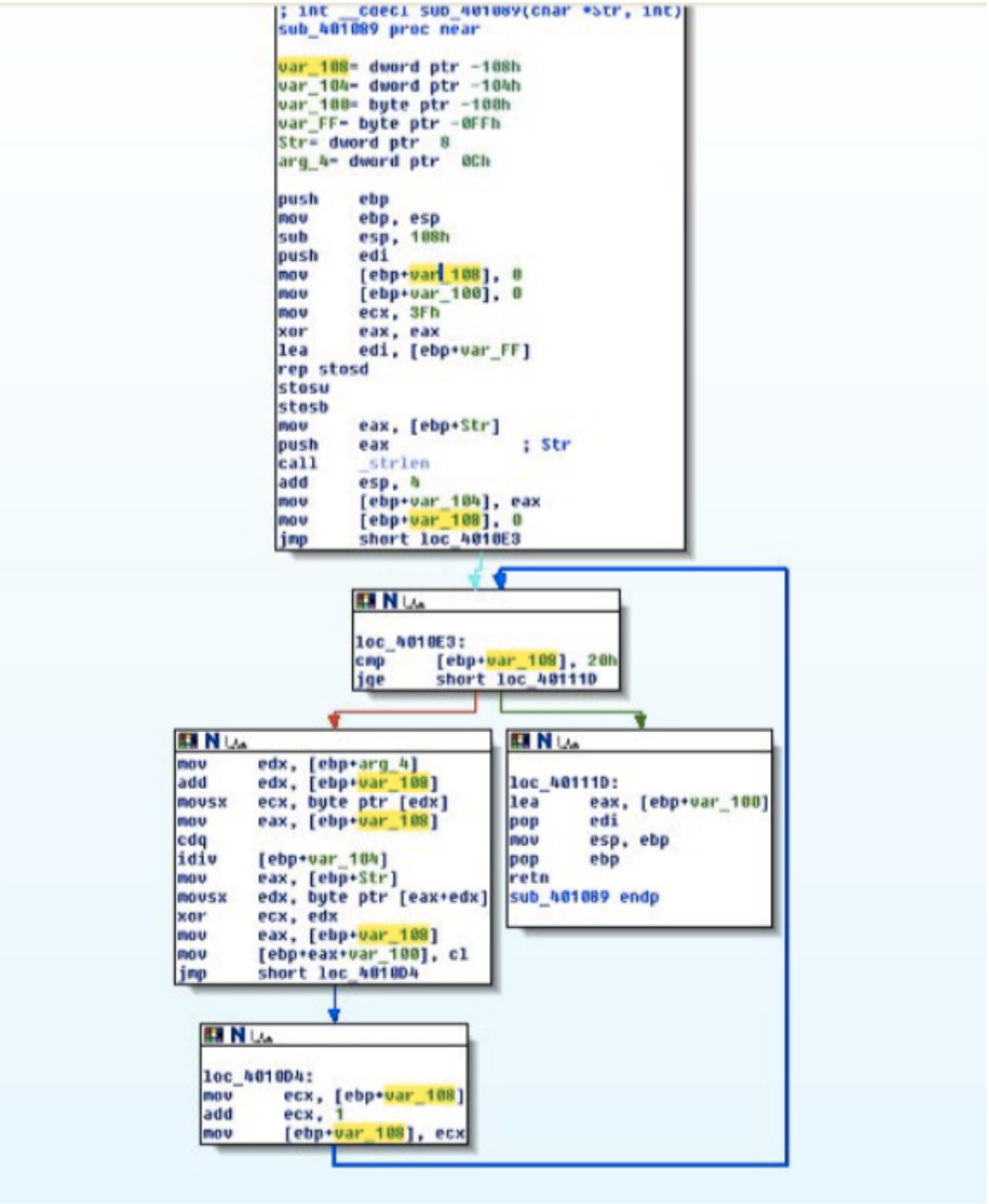
5. What arguments are being passed to subroutine 0x00401089?



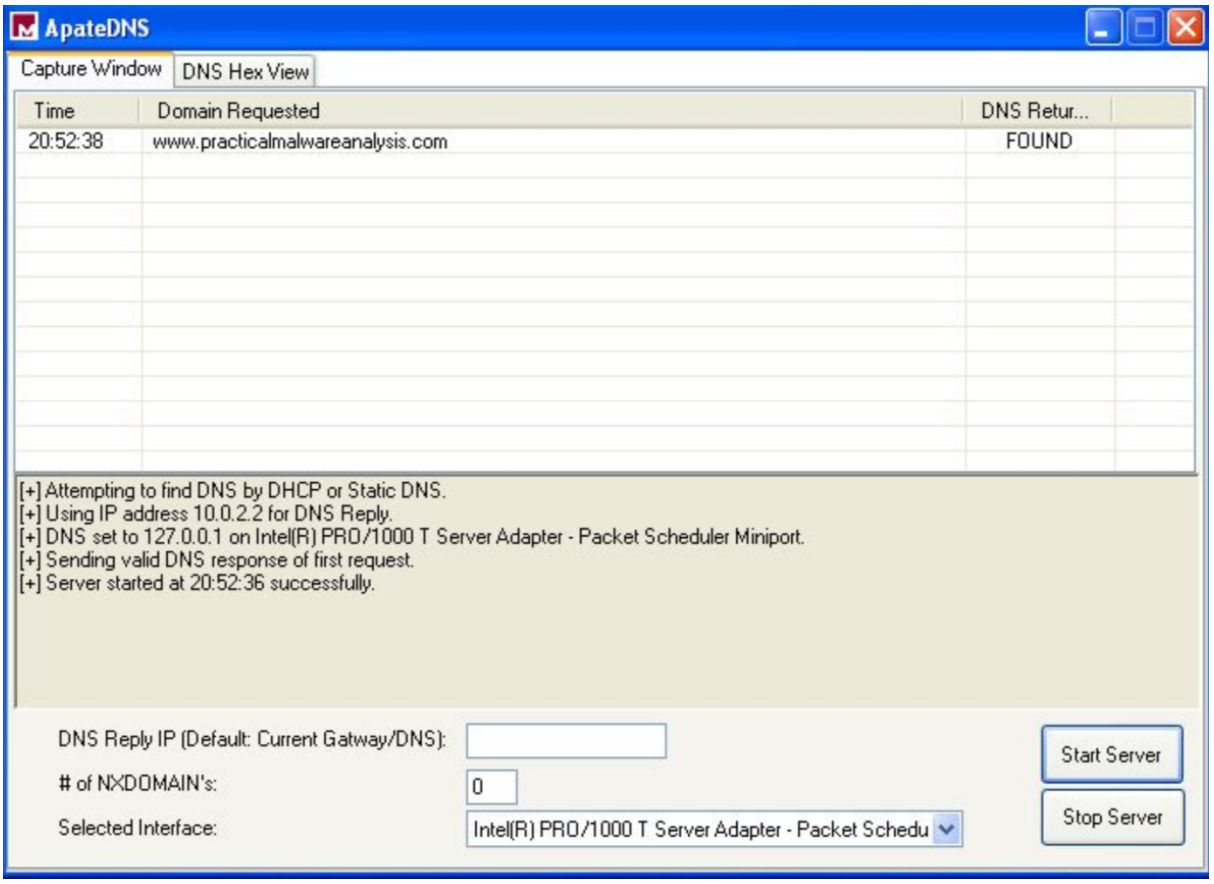
From the picture above we can see the 0x00401089 is called and pass a string and a pointer with it.

Inside the function sub_401089, there is a loop that can read the string and perform XOR operation on it.


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 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 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 00FFFFF 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
00401124 . 8BE5 MOV ESP,EBP
00401126 . 5D POP EBP
00401127 . C3 RETN
00401128 $ 55 PUSH EBP
00401129 . 8BEC MOV EBP,ESP
0040112B . 81EC 04030000 SUB ESP,304
00401131 . 56 PUSH ESI
00401132 . 57 PUSH EDI
Stack SS:[0012FB5C]=0000000A
ECX=00000063
Jump from 0040111B
Address ASCII dump
0012FA7C
0012FA9C
0012FABC
0012FADC
0012FAFC
0012FB1C
0012FB3C
0012FB5C .....www.practic.....
0012FB7C
0012FB9C
0012FBBC
0012FBDC
0012FBFC



6. What domain name does this malware use?



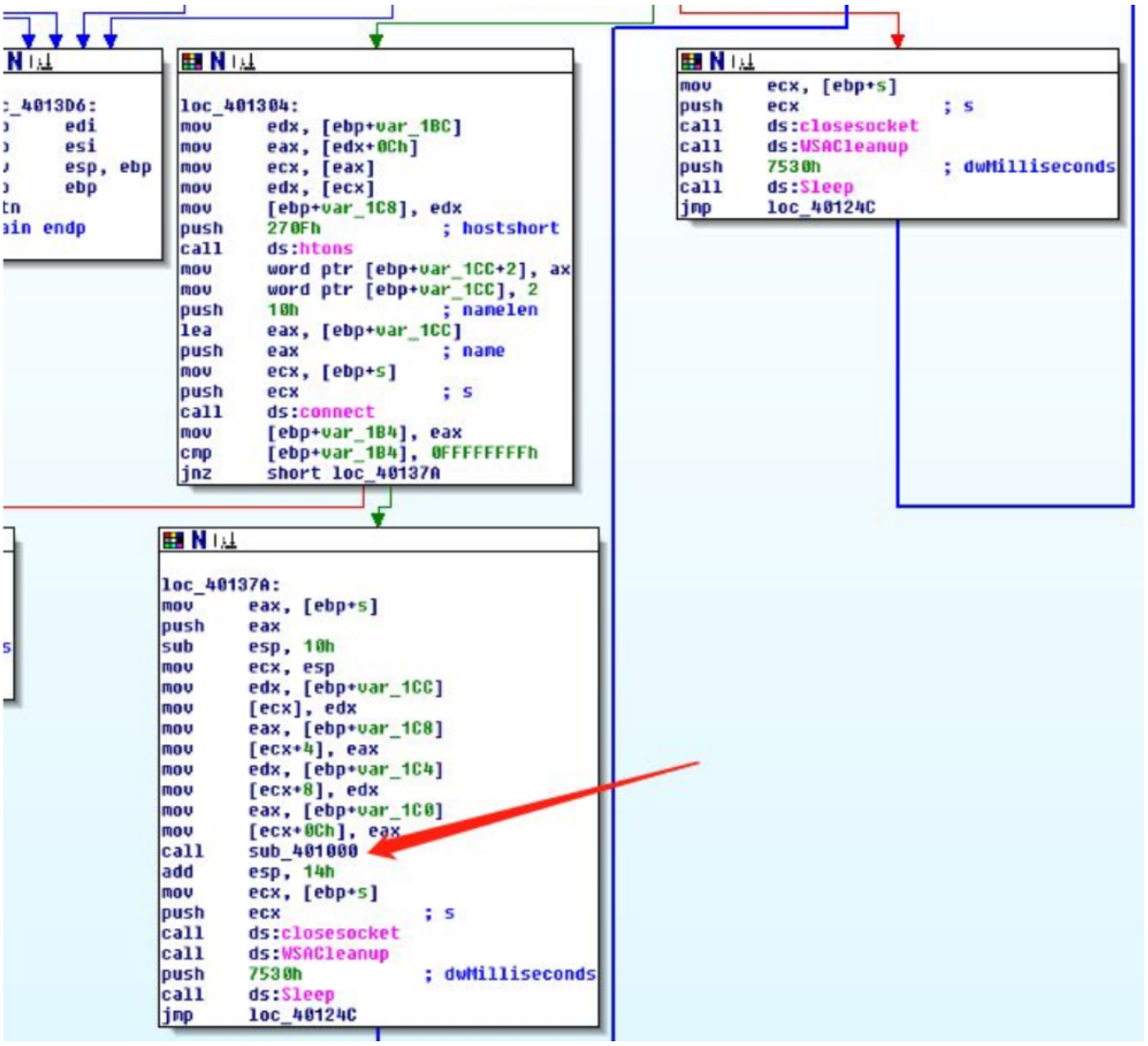
Shown in ApateDNS that the domain name for the malware is www.practicalmalwareanalysis.com.

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

The encoding routine used to obfuscate the domain name is XOR bitwise operation, which has already been mentioned in Lab09-02 question 5.

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

The CreateProcessA has been called in the main function to build a sock connection to the domain. Therefore, the malware runs the CreateProcessA before the socket is closed.



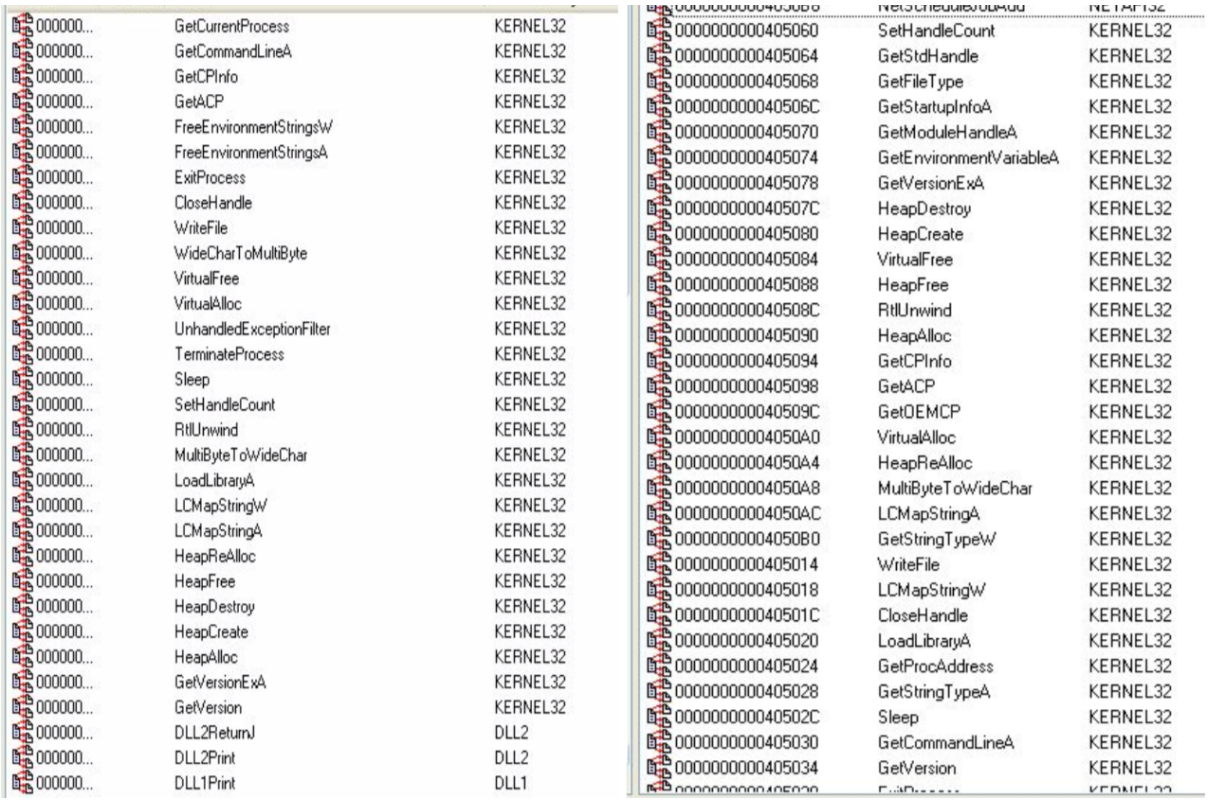
We guess that startupinfo.hstdinput/hstdoutput may be the inputs/outputs from the command window. Therefore, we guess that the socket is to passing the data to the domain and return the inputted data domain which is kind of like the ssh connection from a remote client.

Lab09-03

Analyze the malware found in the file Lab09-03.exe using OllyDbg and IDA Pro. This malware loads three included DLLs (DLL1.dll, DLL2.dll, and DLL3.dll) that are all built to request the same memory load location. Therefore, when viewing these DLLs in OllyDbg versus IDA Pro, code may appear at different memory locations. The purpose of this lab is to make you comfortable with finding the correct location of code within IDA Pro when you are looking at code in OllyDbg.

Questions

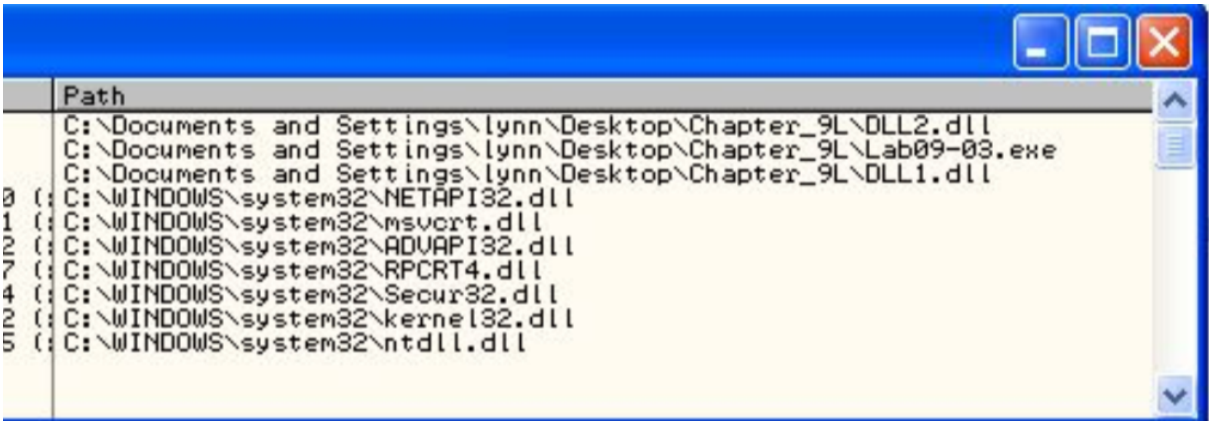
1. What DLLs are imported by Lab09-03.exe?



000000...	GetCurrentProcess	KERNEL32
000000...	GetCommandLineA	KERNEL32
000000...	GetCPInfo	KERNEL32
000000...	GetACP	KERNEL32
000000...	FreeEnvironmentStringsW	KERNEL32
000000...	FreeEnvironmentStringsA	KERNEL32
000000...	ExitProcess	KERNEL32
000000...	CloseHandle	KERNEL32
000000...	WriteFile	KERNEL32
000000...	WideCharToMultiByte	KERNEL32
000000...	VirtualFree	KERNEL32
000000...	VirtualAlloc	KERNEL32
000000...	UnhandledExceptionFilter	KERNEL32
000000...	TerminateProcess	KERNEL32
000000...	Sleep	KERNEL32
000000...	SetHandleCount	KERNEL32
000000...	RtlUnwind	KERNEL32
000000...	MultiByteToWideChar	KERNEL32
000000...	LoadLibraryA	KERNEL32
000000...	LCMapStringW	KERNEL32
000000...	LCMapStringA	KERNEL32
000000...	HeapReAlloc	KERNEL32
000000...	HeapFree	KERNEL32
000000...	HeapDestroy	KERNEL32
000000...	HeapCreate	KERNEL32
000000...	HeapAlloc	KERNEL32
000000...	GetVersionExA	KERNEL32
000000...	GetVersion	KERNEL32
000000...	DLL2ReturnJ	DLL2
000000...	DLL2Print	DLL2
000000...	DLL1Print	DLL1

0000000000405060	SetHandleCount	KERNEL32
0000000000405064	GetStdHandle	KERNEL32
0000000000405068	GetFileType	KERNEL32
000000000040506C	GetStartupInfoA	KERNEL32
0000000000405070	GetModuleHandleA	KERNEL32
0000000000405074	GetEnvironmentVariableA	KERNEL32
0000000000405078	GetVersionExA	KERNEL32
000000000040507C	HeapDestroy	KERNEL32
0000000000405080	HeapCreate	KERNEL32
0000000000405084	VirtualFree	KERNEL32
0000000000405088	HeapFree	KERNEL32
000000000040508C	RtlUnwind	KERNEL32
0000000000405090	HeapAlloc	KERNEL32
0000000000405094	GetCPInfo	KERNEL32
0000000000405098	GetACP	KERNEL32
000000000040509C	GetOEMCP	KERNEL32
00000000004050A0	VirtualAlloc	KERNEL32
00000000004050A4	HeapReAlloc	KERNEL32
00000000004050A8	MultiByteToWideChar	KERNEL32
00000000004050AC	LCMapStringA	KERNEL32
00000000004050B0	GetStringTypeW	KERNEL32
0000000000405014	WriteFile	KERNEL32
0000000000405018	LCMapStringW	KERNEL32
000000000040501C	CloseHandle	KERNEL32
0000000000405020	LoadLibraryA	KERNEL32
0000000000405024	GetProcAddress	KERNEL32
0000000000405028	GetStringTypeA	KERNEL32
000000000040502C	Sleep	KERNEL32
0000000000405030	GetCommandLineA	KERNEL32
0000000000405034	GetVersion	KERNEL32

The malware import the DLLs including kernel32, dll2, dll1 and Netapi32.



Additionally, we find that the malware also import the PRCRT4.dll, msvcrt.dll, ntdll.dll and Secur32.dll when the programs are running.

2. What is the base address requested by DLL1.dll, DLL2.dll, and DLL3.dll?

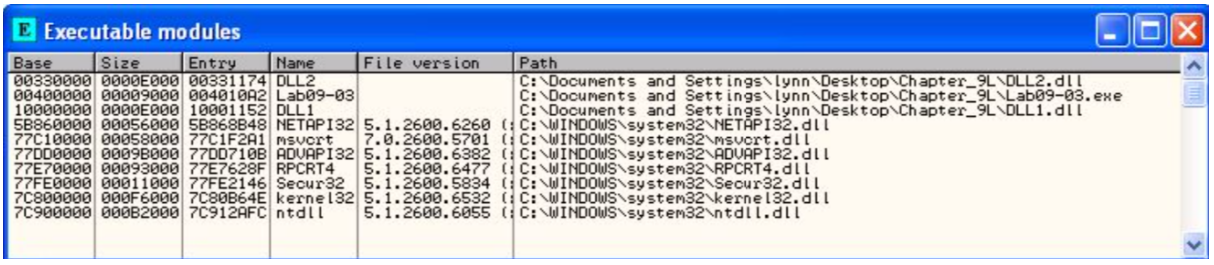
```
;
; Input MD5   : 1F9775ED5D105B4D86B67DEED9C5CF62

; File Name   : C:\Documents and Settings\lynn\Desktop\Chapter_9L\DLL1.dll
; Format      : Portable executable for 80386 (PE)
; Imagebase   : 10000000
; Section 1. (virtual address 00001000)
; Virtual size                  : 000054FA (  21754.)
; Section size in file          : 00006000 (  24576.)
; Offset to raw data for section: 00001000
; Flags 60000020: Text Executable Readable
```

```
; File Name   : C:\Documents and Settings\lynn\Desktop\Chapter_9L\DLL2.dll
; Format      : Portable executable for 80386 (PE)
; Imagebase   : 10000000
; Section 1. (virtual address 00001000)
; Virtual size                  : 0000551A (  21786.)
; Section size in file          : 00006000 (  24576.)
; Offset to raw data for section: 00001000
; Flags 60000020: Text Executable Readable
```

```
; File Name   : C:\Documents and Settings\lynn\Desktop\Chapter_9L\DLL3.dll
; Format      : Portable executable for 80386 (PE)
; Imagebase   : 10000000
; Section 1. (virtual address 00001000)
; Virtual size                  : 0000554A (  21834.)
; Section size in file          : 00006000 (  24576.)
; Offset to raw data for section: 00001000
; Flags 60000020: Text Executable Readable
; Alignment     : default
; OS type         :  MS Windows
; Application type:  DLL 32bit
```

As shown above, the base address requexted by DLL1.dll, DLL2.dll, DLL3.dll is Imagebase: 10000000.

3. When you use OllyDbg to debug Lab09-03.exe, what is the assigned based address for: DLL1.dll, DLL2.dll, and DLL3.dll?



Base address: 0x00330000 for DLL2.dll.
Base address: 0x10000000 for DLL1.dll.
Base address: 0x390000 for DLL3.dll.

4. When Lab09-03.exe calls an import function from DLL1.dll, what does this import function do?

```
push    ebp
mov     ebp, esp
sub     esp, 1Ch
call    ds:DLL1Print
call    ds:DLL2Print
call    ds:DLL2ReturnJ
mov     [ebp+hObject], eax
push    0                   ; lpOverlapped
lea     eax, [ebp+NumberOfBytesWritten]
push    eax                 ; lpNumberOfBytesWritten
push    17h                 ; nNumberOfBytesToWrite
push    offset aMalwareanalysi ; "malwareanalysisbook.com"
```

```
; Exported entry   1. DLL1Print


; Attributes: bp-based frame

public DLL1Print
DLL1Print proc near
push    ebp
mov     ebp, esp
mov     eax, dword_10008030
push    eax
push    offset aDll1MysteryDat ; "DLL 1 mystery data %d\n"
call    sub_10001038
add     esp, 8
pop     ebp
retn
DLL1Print endp
```

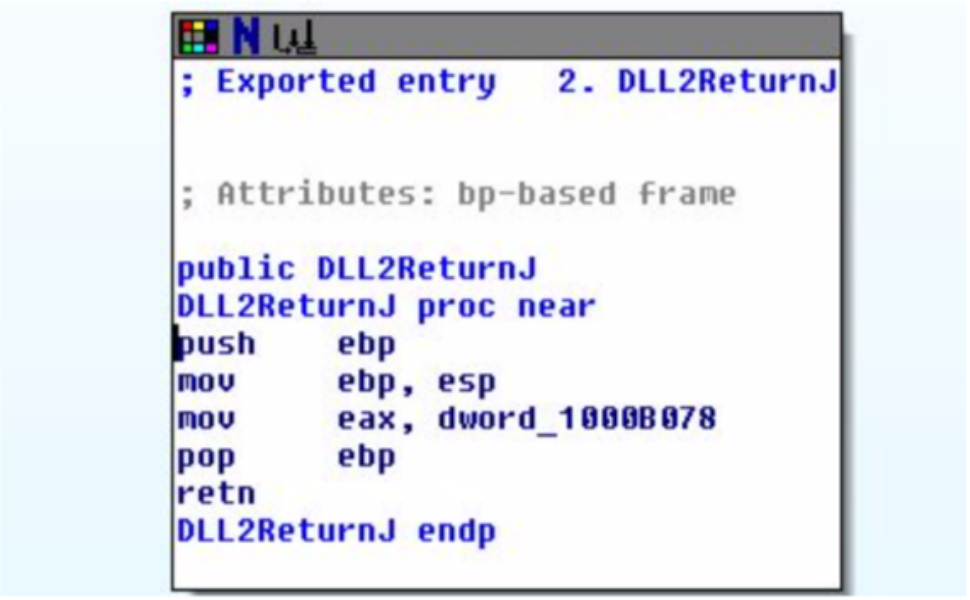
They call the DLL1Print function which pushs the offset “DLL 1 mystery data %d\n” and exafrom dword_10008030 to sub_10001038. And checking the referemce of dword, we find that it seems to be the current process id.

```
; Attributes: bp-based frame

; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
_DllMain@12 proc near

hinstDLL= dword ptr  8
fdwReason= dword ptr  0Ch
lpvReserved= dword ptr  10h

push    ebp
mov     ebp, esp
call    ds:GetCurrentProcessId
mov     dword_10008030, eax
mov     al, 1
pop     ebp
retn    0Ch
_DllMain@12 endp
```

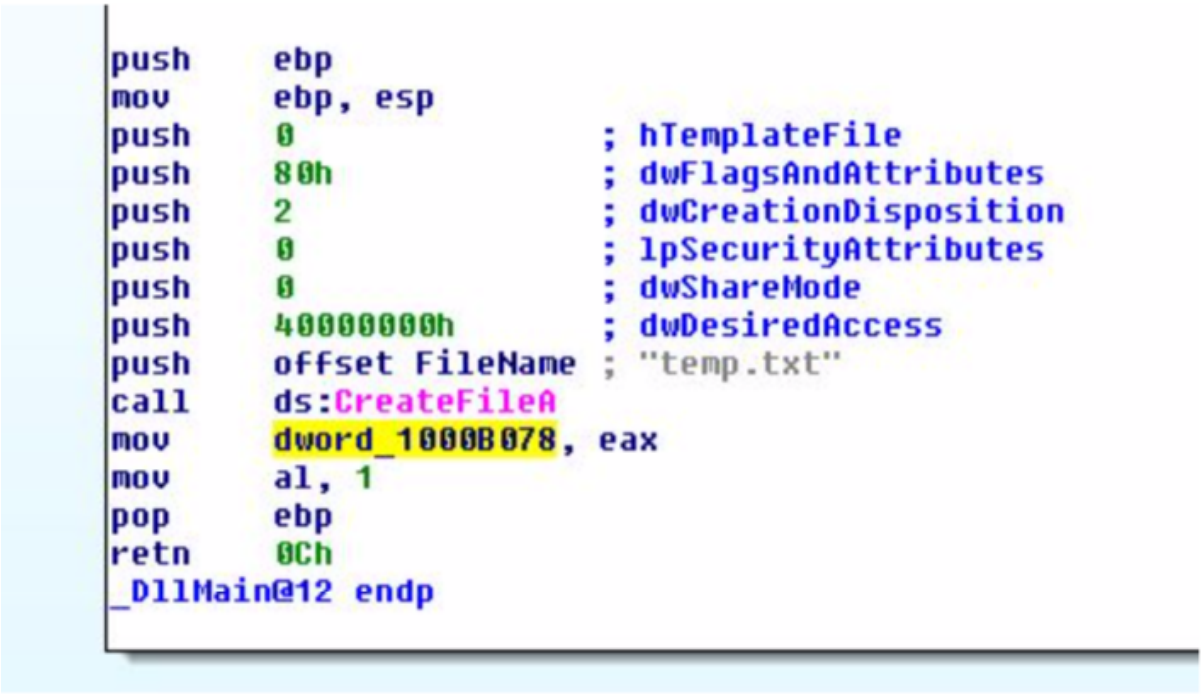
5. When Lab09-03.exe calls WriteFile, what is the filename it writes to?

```
push    ebp
mov     ebp, esp
sub     esp, 1Ch
call    ds:DLL1Print
call    ds:DLL2Print
call    ds:DLL2ReturnJ
mov     [ebp+hObject], eax
push    0               ; lpOverlapped
lea     eax, [ebp+NumberOfBytesWritten]
push    eax             ; lpNumberOfBytesWritten
push    17h             ; nNumberOfBytesToWrite
push    offset aMalwareanalysi ; "malwareanalysisbook.com"
mov     ecx, [ebp+hObject]
push    ecx             ; hFile
call    ds:WriteFile
mov     edx, [ebp+hObject]
push    edx             ; hObject
call    ds:CloseHandle
push    offset LibFileName ; "DLL3.dll"
call    ds:LoadLibraryA
mov     [ebp+hModule], eax
push    offset ProcName ; "DLL3Print"
mov     eax, [ebp+hModule]
push    eax             ; hModule
call    ds:GetProcAddress
```

```
BOOL WINAPI WriteFile(
  _In_        HANDLE       hFile,
  _In_        LPCVOID      lpBuffer,
  _In_        DWORD        nNumberOfBytesToWrite,
  _Out_opt_   LPDWORD      lpNumberOfBytesWritten,
  _Inout_opt_ LPOVERLAPPED lpOverlapped
);
```



```
; Exported entry   2. DLL2ReturnJ


; Attributes: bp-based frame

public DLL2ReturnJ
DLL2ReturnJ proc near
push    ebp
mov     ebp, esp
mov     eax, dword_1000B078
pop     ebp
retn
DLL2ReturnJ endp
```



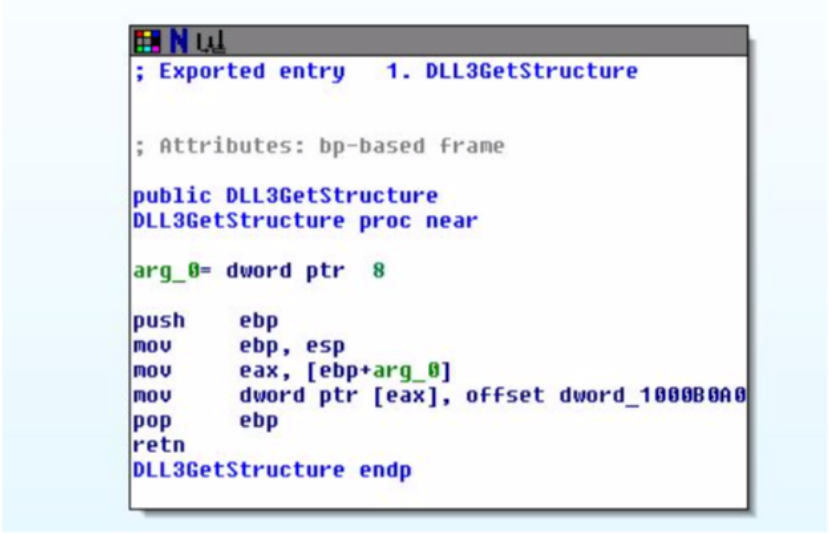
```
push    ebp
mov     ebp, esp
push    0                   ; hTemplateFile
push    80h                 ; dwFlagsAndAttributes
push    2                   ; dwCreationDisposition
push    0                   ; lpSecurityAttributes
push    0                   ; dwShareMode
push    40000000h           ; dwDesiredAccess
push    offset FileName     ; "temp.txt"
call    ds:CreateFileA
mov     dword_1000B078, eax
mov     al, 1
pop     ebp
retn    0Ch
_DllMain@12 endp
```

DLL2ReturnJ. And the name of the file is “temp.txt”.

6. When Lab09-03.exe creates a job using NetScheduleJobAdd, where does it get the data for the second parameter?

```
push    offset LibFileName ; "DLL3.dll"
call    ds:LoadLibraryA
mov     [ebp+hModule], eax
push    offset ProcName ; "DLL3Print"
mov     eax, [ebp+hModule]
push    eax             ; hModule
call    ds:GetProcAddress
mov     [ebp+var_8], eax
call    [ebp+var_8]
push    offset aDll3getstructu ; "DLL3GetStructure"
mov     ecx, [ebp+hModule]
push    ecx             ; hModule
call    ds:GetProcAddress
mov     [ebp+var_10], eax
lea     edx, [ebp+Buffer]
push    edx
call    [ebp+var_10]
add     esp, 4
lea     eax, [ebp+JobId]
push    eax             ; JobId
mov     ecx, [ebp+Buffer]
push    ecx             ; Buffer
push    0               ; Servername
call    NetScheduleJobAdd
push    2710h           ; dwMilliseconds
call    ds:Sleep
xor     eax, eax
mov     esp, ebp
pop     ebp
retn
_main endp
```

The second parameter is Buffer which is [ebp+ Buffer] in this case. Track sdp from LoadLibraryA. It calls the GetProcAddress to get the base address of the function in DLL3.dll.



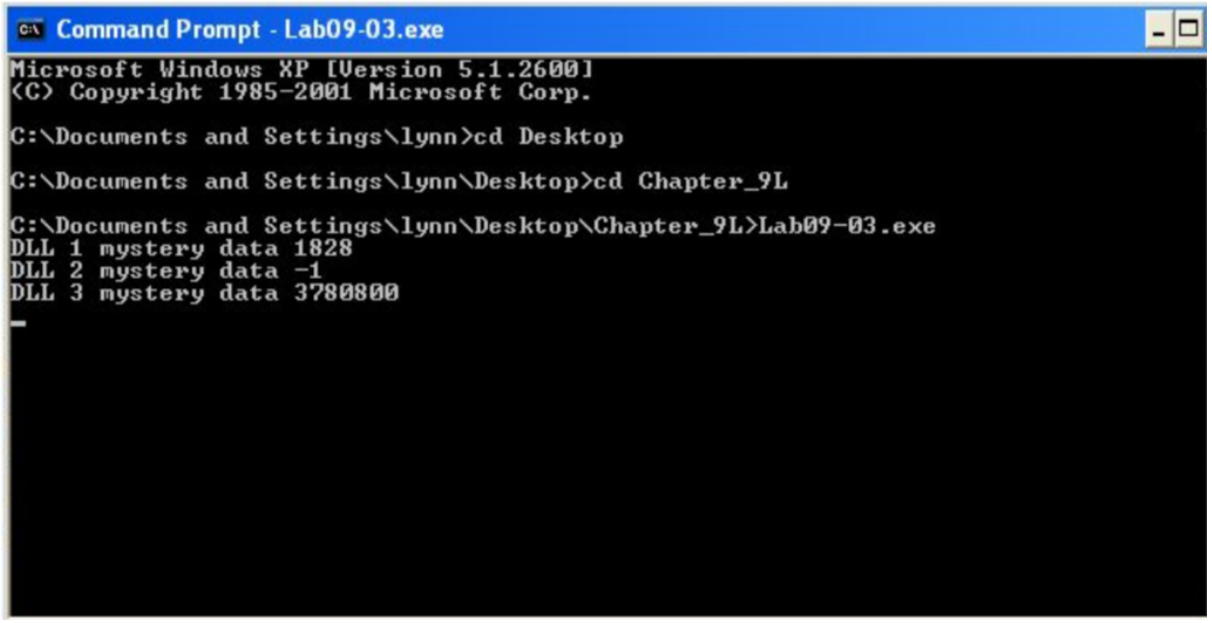
```
; Exported entry   1. DLL3GetStructure


; Attributes: bp-based frame

public DLL3GetStructure
DLL3GetStructure proc near

arg_0= dword ptr  8

push    ebp
mov     ebp, esp
mov     eax, [ebp+arg_0]
mov     dword ptr [eax], offset dword_1000B0A0
pop     ebp
retn
DLL3GetStructure endp
```

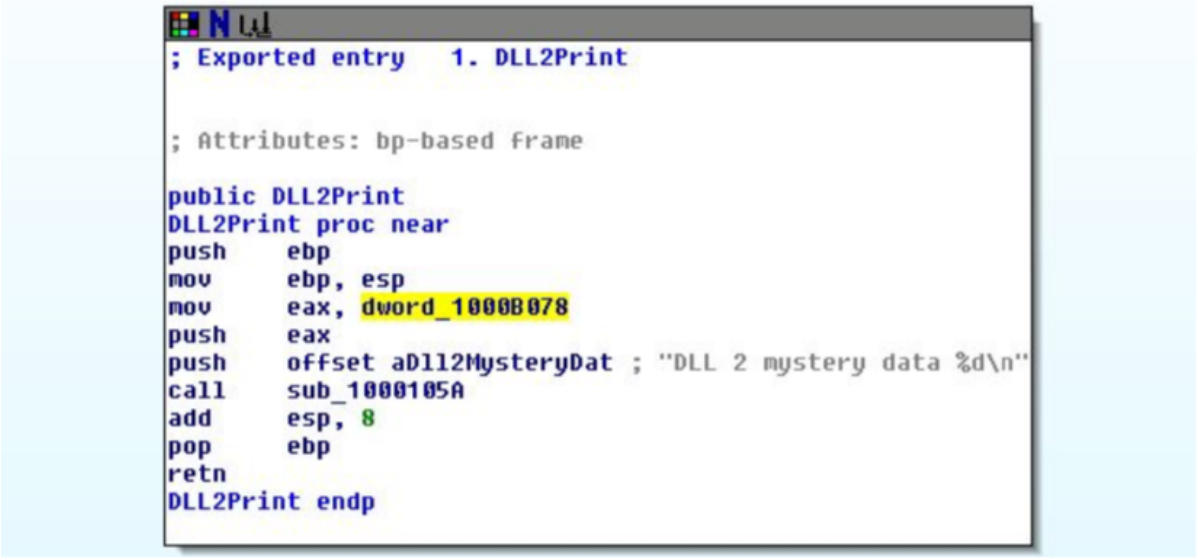
```
lpMultiByteStr= dword ptr -4
hinstDLL= dword ptr  8
fdwReason= dword ptr  0Ch
lpvReserved= dword ptr  10h

push    ebp
mov     ebp, esp
push    ecx
mov     [ebp+lpMultiByteStr], offset aPingWww_malwar ; "ping www.malwareanalysisbook.com"
push    32h             ; cchWideChar
push    offset WideCharStr ; lpWideCharStr
push    0FFFFFFFFh      ; cbMultiByte
mov     eax, [ebp+lpMultiByteStr]
push    eax             ; lpMultiByteStr
push    0               ; dwFlags
push    0               ; CodePage
call    ds:MultiByteToWideChar
mov     dword_1000B0AC, offset WideCharStr
mov     dword_1000B0A0, 36EE80h
mov     dword_1000B0A4, 0
mov     byte_1000B0A8, 7Fh
mov     byte_1000B0A9, 11h
mov     al, 1
mov     esp, ebp
pop     ebp
retn    0Ch
_DllMain@12 endp
```

The data source come from the xref of dword_1000B0A0.

7. While running or debugging the program, you will see that it prints out three pieces of mystery data. What are the following: DLL 1 mystery data 1, DLL 2 mystery data 2, and DLL 3 mystery data 3?

DLL1 may be the current process id.


Command Prompt - Lab09-03.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\lynn>cd Desktop
C:\Documents and Settings\lynn\Desktop>cd Chapter_9L
C:\Documents and Settings\lynn\Desktop\Chapter_9L>Lab09-03.exe
DLL 1 mystery data 1828
DLL 2 mystery data -1
DLL 3 mystery data 3780800



```
; Exported entry   1. DLL2Print


; Attributes: bp-based frame

public DLL2Print
DLL2Print proc near
push    ebp
mov     ebp, esp
mov     eax, dword_1000B078
push    eax
push    offset aDll2MysteryDat ; "DLL 2 mystery data %d\n"
call    sub_1000105A
add     esp, 8
pop     ebp
retn
DLL2Print endp
```

```
push    0                ; lpSecurityAttributes
push    0                ; dwShareMode
push    40000000h        ; dwDesiredAccess
push    offset FileName  ; "temp.txt"
call    ds:CreateFileA
mov     dword_1000B078, eax
mov     al, 1
pop     ebp
retn    0Ch
_DllMain@12 endp
```

```
HANDLE WINAPI CreateFile(
  _In_     LPCTSTR                lpFileName,
  _In_     DWORD                  dwDesiredAccess,
  _In_     DWORD                  dwShareMode,
  _In_opt_ LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  _In_     DWORD                  dwCreationDisposition,
  _In_     DWORD                  dwFlagsAndAttributes,
  _In_opt_ HANDLE                 hTemplateFile
);
```

DLL2 prints the return value of CreateFileA.
DLL3 should be the byte of “ping www.malwareanalysisbook.com”.

```
; Exported entry   2. DLL3Print


; Attributes: bp-based frame

public DLL3Print
DLL3Print proc near
push    ebp
mov     ebp, esp
push    offset WideCharStr
push    offset aDll3MysteryDat ; "DLL 3 mystery data %d\n"
call    sub_10001087
add     esp, 8
pop     ebp
retn
DLL3Print endp
```

```
push    ebp
mov     ebp, esp
push    ecx
mov     [ebp+lpMultiByteStr], offset aPingWww_malwar ; "ping www.malwareanalysisbook.com"
push    32h             ; cchWideChar
push    offset WideCharStr ; lpWideCharStr
push    0FFFFFFFFh      ; cbMultiByte
mov     eax, [ebp+lpMultiByteStr]
push    eax             ; lpMultiByteStr
push    0               ; dwFlags
push    0               ; CodePage
call    ds:MultiByteToWideChar
mov     dword_1000B0AC, offset WideCharStr
mov     dword_1000B0A0, 36EE80h
mov     dword_1000B0A4, 0
mov     byte_1000B0A8, 7Fh
mov     byte_1000B0A9, 11h
mov     al, 1
mov     esp, ebp
pop     ebp
retn    0Ch
_DllMain@12 endp
```

8. How can you load DLL2.dll into IDA Pro so that it matches the load address used by OllyDbg?

Manually load to input the image base of 10000000 to load DLL2.dll into IDA Pro.