Xiaoyi Zhou

Purdue University CIT581

Malware Forensics

Lab 5: IDA Pro

Due September 26, 2014

Instructor: Samuel Liles

**Abstract**

This lab is designed for letting us be familiar with IDA Pro. Since it is a rewritten version, I found that I become more familiar with how does IDA Pro analyze malware. Lab5.dll cannot be run at OllyDbg and part of codes cannot be displayed at IDA Pro free version. But there is always be a way to figure out the answer.

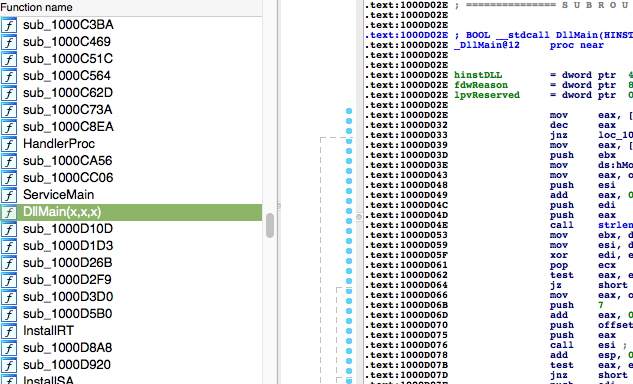
From what I analyze, the malware will intercept with network and TCP protocol. It might execute DNS poison attack. Moreover, it will start a remote shell and start conversation with hacker.

**Steps of Process with Reviewed Questions**

**1. What is the address of DllMain?**

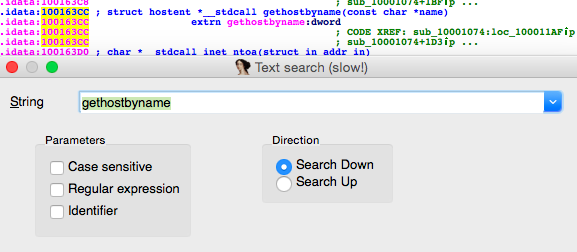
Using search text tab to search DllMain. There are a lot of entries containing DllMain, but the real DllMain( )function is located at .text:1000D02E

Or we don't have to use search tab. See the left side function name bar. Scroll censor down or up until we see DllMain(x, x, x), which is easier.



**2. Use the Imports window to browse to gethostbyname . Where is the import located?**

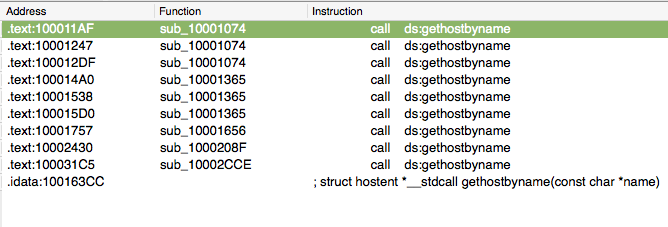
I try to use same method that finds DllMain at question one. However, gethostbyname seemingly doesn't appear at left side function bar. Therefore I use traditional method that opened searching box and type gethostbyname( ).



gethostbyname( )function is located at .idada:100163CC. Just directly search the name of the function at impart window and we will get the answer.

**3. How many functions call gethostbyname?**

Open searching box and type gethostbyname( ); Count the locations that contain gethostname( ) except the one at data section because it is import of gethostbyname( ).



In this case the last entry idata:100163CC should be excluded.

First time call .text:100011AF

Second time call .text: 10001247

Third time call .text :100012DF

Fourth time call .text:1000014A0

Fifth time call .text: 10001538

Sixth time call .text:100015D0

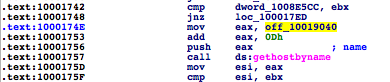
Seventh time call .text:10001757

Eighth time call .text:20002430

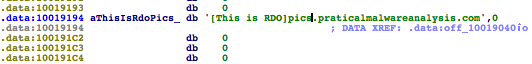
Ninth time call .text:10031C5

**4. Focusing on the call to gethostbyname located at 0x10001757; can you figure out which DNS request will be made?**

The function call gethostbyname only takes on parameter. The register after push should be the parameter location. In this case, eax is a string pointer at 0x1000174E. The string is located at offset 0x100019404.

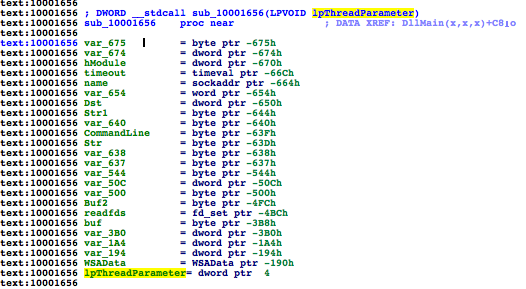
****

Double click on off\_10019404, I can see the string value stored here. "[This is RDO]pics.practicalmalwareanalysis.com" is stored at off\_1009040 and then moved to eax. Actually it should drop the double quotation when we analyze it. eax is added by 0xD which is 13 in decimal. Therefore, the pointer pointing to eax should be move right 13bytes. So I wipe 13 characters from [This is RDO]pics.practicalmalwareanalysis.com. The result will be pics.practicalmalwareanalysis.com.

****

Put the censor at the address 0x00174E three lines above the gethostbyname, the 0x10019040 located the DNS address of pics.praticalmalwareanalysis.com

**5. How many local variables has IDA Pro recognized for the subroutine at 0x10001656?**



Use censor or searching box to find ox10001656. Here the subroutine just takes one parameter. If we don't which one is considered as parameter at the left side list, we can check the function and see how many parameters are in brackets. Moreover, local variable is mapped with offset while parameter is pointed by a pointer. Except the parameters, the rest listed on left hand are local variable. We can count them as 23 local variables. I think using IDA Pro demo version is better than free version because demo version can give me more information.

**6. How many parameters has IDA Pro recognized for the subroutine at 0x10001656?**

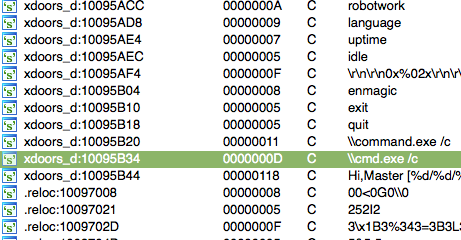
There is only one parameter like the following displayed.

lpThreadAddress =dword ptr 4.

In IDA Pro free version, the parameter is denoted as arg\_0. Although arg\_0 is more like a parameter, lpThreadAddress can indicate what is stored at this parameter.

**7. Use the Strings window to locate the string \cmd.exe /c in the disassembly. Where is it located?**

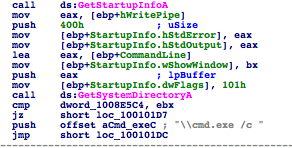
According to this question, we should open string view at IDA pro. But when I open string view, I found there are too many strings. It is a little bit time consuming to find a specific string at string view.



So I think a more convenient way is to type the key words at searching box and we can find that the string is located at 0x10095B34 and took as parameter at 0x100101D0.

Mac:Users:xiaoyizhou:Desktop:Screen Shot 2014-10-21 at 10.41.32 PM.png

**8. What is happening in the area of code that references \cmd.exe /c?**



In the area of the code, there is getsystemdirectoryA()function, and reference involving starting information and showing window function.

For example "Encrypt Magic number For This Remote Shell Session".

At the area of address 10095B44 (about 10 lines), the IDA pro listed the malicious conversation that is obviously revoking the host. Therefore, I think the malware starts a shell remotely and try to talk with the host.

**9. In the same area, at 0x100101C8, it looks like dword\_1008E5C4 is a global variable that helps decide which path to take. How does the malware set dword\_1008E5C4 ? (Hint: Use dword\_1008E5C4 ’s cross-references.)**

Navigate to the address and double click on dword\_1008E5C4 because the current location cannot give us further useful information related to dword\_1008E5C4.

Mac:Users:xiaoyizhou:Desktop:Screen Shot 2014-10-22 at 11.01.04 AM.png

There are data cross-reference and a function reference. We need to choose the data reference and see how it is set by the malware. We navigate the cross-reference of the key words, at the address of 0x1008E5C4 where we found the important lines:

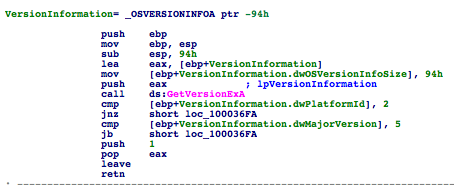
call sub\_10003695

mov dword\_1008E5C4, eax

The data in eax is stored in dword\_1008E5C4, so we should figure out what is in eax and what does the function call do. Then we navigate to the address 0x10001678, here we find the two important lines:

call sub\_10003695

mov dword\_1008E5C4, eax



Put censor at sub\_10003695, the function shows the OSVERSIONINFO as result.

see the code: VersionInformation=\_OSVERSIONINFOA ptr -94h

Therefore, the global variable dword\_10008E5C4 is used to store the OS information.

**10. A few hundred lines into the subroutine at 0x1000FF58, a series of comparisons use memcmp to compare strings. What happens if the string comparison to robotwork is successful (when memcmp returns 0)?**

When I first navigate to the address 0x1000FF58, I didn't have any clue so that we directly research key words "robotwork". Then we go to the address containing robotwork which is 0x10001044C. Around that code area, we can find that:

call memcmp ;the result is 0 indicating success.

add esp, 0ch

test eax, eax ;the condition for the comparison. The value of ZF.

jnz short loc\_100010468 ;since the memcmp returns value 0, ZF value is 0. This code will not be executed. We can ignore it so far.

push [ebp+s] ;this is variable used by sub\_100052A2 ().

call sub\_100052A2

Now, we should navigate to the function sub\_100052A2(). We noticed that it is a subroutine of sub\_1000FF58. In the function sub\_100052A2(), the malware will get the information of registry value.

SOFTWARE\Microsoft\Windows\\CurrentVersion

WorkTime

Robot\_WorkTime.

The function will return the three information as return value to sub\_1000FF58(). Now we have to go back to 0x1000FF58 again.

There are a lot of functions related to socket and system and malicious conversation at question 8. Such as GetTickCount() and GetCurrentDirectoryA(). Under GetTickCount(), we also can find closesocket(). Thus, I assume the function sub\_1000FF68 will also start a shell. And it will display the value of work time, robot work time and windows current version at the shell window, which aims to create or fake the system.

**11. What does the export PSLIST do?**

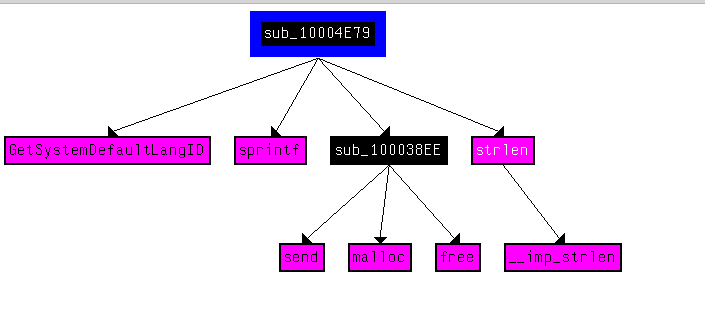
It seems like the function sub\_100036C3 at PSLIST wants to get the current OS version information and return the current version.

Then, comparing the current version with the version needed by the malware. If the two versions are same, the function sub\_10006518() will be executed. In this function, we see the function CreateToolhelp32SnapShot() with the processID. When the function succeeds, it returns an open handle to the specified snapshot. In other words, this function takes a snapshot of the processes, heaps, modules, and threads used by the process.

Now the malware gets all the information of the process.

After the function CreateToolhelp32SnapShot(), the malware called the function sub\_1000620C, I assume the malware DLL file took the process as the host exe to run.

**12. Use the graph mode to graph the cross-references from sub\_10004E79 .Which API functions could be called by entering this function? Based on the API functions alone, what could you rename this function?**



sub\_100004E79 has the function calls (in called order):

GetSystemDefaultLangID()

sprintf()

strlen()

I assume this function is collecting the system language information.

Inside of the sub\_100004E79, there is a function called sub\_100038EE that has the API calls as following:

malloc()

send()

free()

Since the parameter in sub\_100038EE is (socket s, int, int), I assume this function will send the system language information gained by the former function call to a remote shell, or just send it to the network.

By right click sub\_10004E79, we can rename it like SystemLanguageInfo.

**13. How many Windows API functions does DllMain call directly? How many at a depth of 2?**

Depth of 2 means a function has one more function call inside of it. Navigate to the location by pressing G. Using graphic version is easier to answer this question.

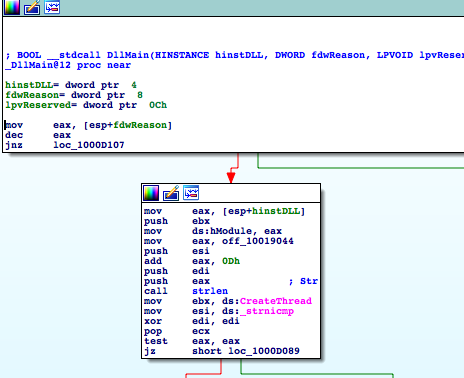
Directly called function as following:

strlen()

strnicmp()

Createthread()

strncpy()



Inside the DllMain(), the functions such as strlen(), sub\_10001074, sub\_10001365, sub\_10001656 has more than 1 depth.

strlen() contains imp\_strlen().

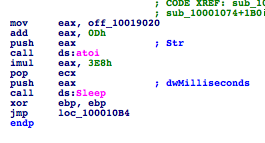
sub\_10001074 contains strchr(), inet\_addr(), gethostbyname(), memcpy(), strcpy(), atoi(), and sleep().

sub\_10001365 contains almost the same API function calls.

sub\_10001656 contains sleep(), WSAStartup(), GetProcAddress(), closesocket(), socket(), imp\_printf(), WSAGetLastError(), memcmp(), ntohs(), connect(), memset(), send(), recv(), select(), createthread(), closehandle(), and exitthread().



**14. At 0x10001358, there is a call to Sleep (an API function that takes one parameter containing the number of milliseconds to sleep). Looking backward through the code, how long will the program sleep if this code executes?**



Mac:Users:xiaoyizhou:Desktop:Screen Shot 2014-10-22 at 10.32.31 AM.png

Look backward through the code.

mov eax, off\_10019020 ; eax here is a pointer. eax = '[This is CIT]30', we can double click the offset, and double click on unk\_100192AC.

add eax, 0Dh ; eax=30 (string here)

push eax

call ds:atoi ; eax=30 (integer here)

imul eax, 3E8h ; eax=30000 (milliseconds)

Therefore the process will sleep for 30 seconds.

At IDA Pro free version, the string is '[This is CIT]', without 30. But IDA Pro demo version will show full string. Here eax is a pointer register and it is added to 0xD (13 in decimal) so it moves left to right by 13bytes. 'This is CIT' got dropped. Function call atoi( ) convert string 30 to decimal 30. That is how I get the number 30.

**15. At 0x10001701 is a call to socket. What are the three parameters?**

Use censor or press G to navigate to 0x10001701. A good way to check parameter is to find the register after operand push but not always. In this case, before the function call of socket(), there are three parameters like the following:

push 6

push 1

push 2

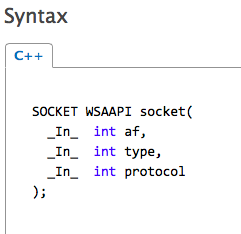
So the answer should be 6, 1, and 2.

Mac:Users:xiaoyizhou:Desktop:Screen Shot 2014-10-22 at 10.20.17 AM.png

**16. Using the MSDN page for socket and the named symbolic constants functionality in IDA Pro, can you make the parameters more meaningful?**

**What are the parameters after you apply changes?**

A part of following answers is retrieved from MSDN page of socket function. http://msdn.microsoft.com/en-us/library/windows/desktop/ms740506(v=vs.85).aspx



The three parameters are (6, 1, 2). According to the MSDN pages for socket, the three parameters indicates the address family as af, address type as type, protocol as protocol

We click each number and choose use standard symbolic constant. However, each number can represent a lot of type names. We have to go through the type names and find the one we need.

af: The address family specification. Possible values for the address family are defined in the Winsock2.h header file.

when af=2, af=AF\_INET=The Internet Protocol version 4 (IPv4) address family.

type: The type specification for the new socket. Possible values for the socket type are defined in the Winsock2.h header file.

when type=1, type=SOCKET\_STREAM=A socket type that provides sequenced, reliable, two-way, connection-based byte streams with an OOB data transmission mechanism.

when protocol is 6, protocol=IPPROTO\_TCP=The Transmission Control Protocol (TCP).

On most x86 machines, function arguments are pushed on the stack from right side to left side. Therefore, 6 is protocol, 1 is address type, and 2 is address family. We can note 6 as IPROTO\_TCP, 1 as SOCKET\_STREAM, and 2 as AF\_INET.

Mac:Users:xiaoyizhou:Desktop:Screen Shot 2014-10-22 at 10.31.13 AM.png

**17. Search for usage of the in instruction (opcode 0xED ). This instruction is used with a magic string VMXh to perform VMware detection. Is that in use in this malware? Using the cross-references to the function that executes the in instruction, is there further evidence of VMware detection?**

Yes, we the string VMXh is used at this malware. We use search sequence function to find 0xED, and double click the function "in" at the address 0x100061DB. To get the value stored in eax, we moved the censor to upper code.

mov eax, 564D5868h ;VMXh

mov ebx, 0

mov exc, 0Ah

mov edx, 5658h ; VM

in eax, dx

cmp ebx, 564D5868h ;VMXh. If two values are equal, Z is set to 1 otherwise 0.

......

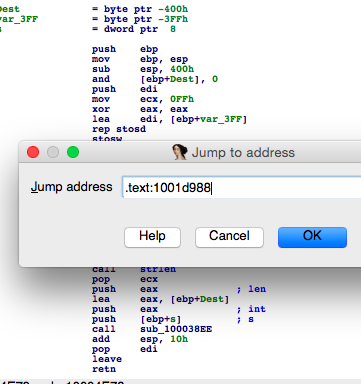
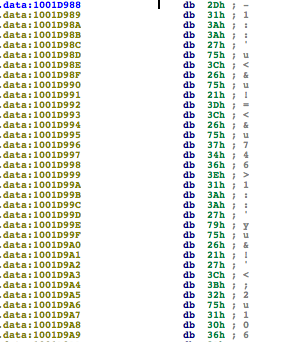
......

jmp short loc\_100061F6; jump to the address 0x100061F6

In this code area, the value stored at eax is 564D5868h whose ASCII code is VMXh by right click on it. For the further evidence, double click on the code cross reference, and we found the operand like that:

mov [esp+8+var\_8], offset aFoundVirtualMa: "Found Virtual Machine, Install Cancel." which could be result of detecting virtual machine.

**18. Jump your cursor to 0x1001D988. What do you find?**



If we want to jump to a specific location, we can press G and put the address on the message box. I found there isn't any operation but the data 0x2D.

**19. If you have the IDA Python plug-in installed (included with the commercial version of IDA Pro), run Lab05-01.py, an IDA Pro Python script provided with the malware for this book. (Make sure the cursor is at 0x1001D988.) What happens after you run the script?**

We saw the python script like the following:

sea= ScreenEA() /\* current location of the censor at 0x2D \*/

for i range (0x00, 0x50) /\*for loop starts at 0, end at 80 \*/

b=byte(sea+i) /\*move the plaintext to the next one \*/

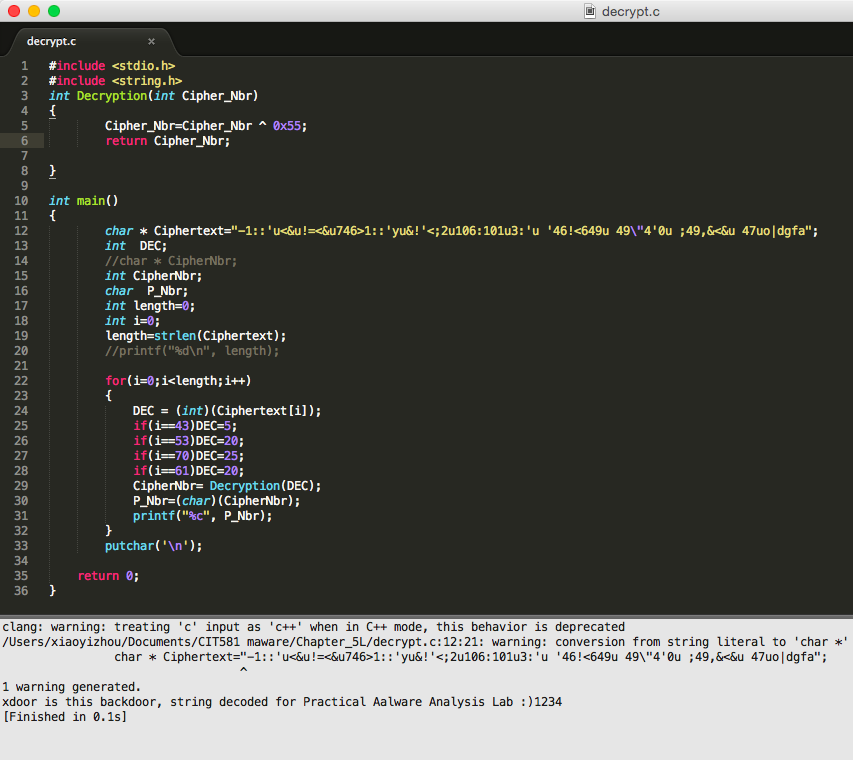
decode\_byte=b^0x55 /\*encrypt the data with XOR 0x55, 80 in decimal \*/

Patcgbyte(sea+i, decode\_byte)

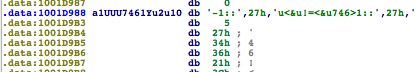
This is an encryption function.

Data from 0x2D to 0x61 has been encrypted. After we run the lab5-01.py, we can see the plaintext has been encrypted into readable character. See the ciphertext by press A. The ciphertext is xdoor is this backdoor, string decoded for Practical Malware Analysis Lab :)1234.

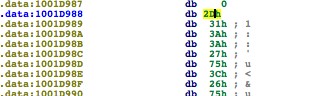
Actually we don't have to run the python script if our IDA version doesn't have the python plug in. In IDA Pro demo version, we don't have it. Therefore we can write a program to decrypt this string. See next page. However, we should manipulate and change some characters because 0x5, 0x14, 0x18 and 0x19 are special characters at ASCII table. So we need to pass the decimal value to cipher number variable.



**20. With the cursor in the same location, how do you turn this data into a single ASCII string?**

****

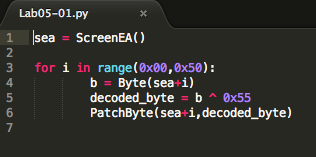
If we press A on keyboard, the data area at 0x2D will be like this. Press D on keyboard again, there will be a massage box showing that convert between data and ASCII, just click ok, and the data will be converted to hex digit again.

****

We can press A and D respectively to convert between hex data and ASCII code.

The string is xdoor is this backdoor, string decoded for Practical Malware Analysis Lab :)1234.

**21. Open the script with a text editor. How does it work?**



See the decryption program at question 19. This is a python script which could be easily converted to C/C++. The script will perform an encryption function.

The plaintext is the unreadable data from 0x2D to 0x61.

The key is 0x55.

The cipher is XOR

The ciphertext is xdoor is this backdoor, string decoded for Practical Malware Analysis Lab :)1234.

**Issues or Problems**

I try to install IDA Pro python plugin but it didn't work. And sometimes demo version is better then free version.

**Conclusion**

I noticed that this malware is able to detect if it is running on virtual machine, which is interesting and seems like sneaky. I would like to see this kind of malware in future.