Assignment 4

Justin Kennelly

University Of Arizona

CYBV 454 MALWARE THREATS & ANALYSIS

Code Constructs in ASM

Professor Galde

March 14, 2022

LAB 6-2

* Lab06-02exe : MD5 Checksum Value: c0b54534e188e1392f28d17faff3d454

LAB 6-3

* Lab06-03exe : MD5 Checksum Value: 3f8e2b945deba235fa4888682bd0d640

# LAB 6-2 Static & Dynamic Analysis

Table

Description automatically generated

*Figure 1: Imports used by Lab06-02.exe in PEview*

*Text

Description automatically generated*

*Figure 2: Strings returned from Lab06-02.exe*

*Text

Description automatically generated*

*Figure 3: Running Lab06-02.exe*

*Table

Description automatically generated with medium confidence*

*Figure 4: Monitoring Lab06-02.exe while running through ProcMon*

*Text

Description automatically generated*

*Figure 5: FakeNet Window while running Lab06-02.exe*

*Graphical user interface, application

Description automatically generated*

*Figure 6: Wireshark Data captured by FakeNet while running Lab06-02.exe*

Before analysis of the code constructs of the file Lab06-02.exe I began by doing some basic static and dynamic analysis. I started static analysis by looking at the imports of the file. To do so I opened the file using PEview and navigated to SECTION .rdata>IMPORT Address Table. The file imports two DLL’s one ifs KERNEL32.dll and the other WININET.dll. Immediately I suspect that because the program uses WININET.dll it is using network functionality. Taking a closer look at the functions I can see InternetOpenUrlA, InternetCloseHandle, InternetReadFile, InternetGetConnectedState, and InternetOpenA. This tells me that the program is specifically targeting websites over HTTP or HTTPS with its network functionality. In addition, within KERNEL32.dll I see interesting functions like: GetVersionEX, GetStartupInfo, GetFileType, GetModuleHandle, WriteFile, SetFilePointer, GetCurrentProcess, GetStringType, and HeapCreate (Figure 1). Next, I look at the strings of the program to glean more information. By running the strings command I see \_GLOBAL\_HEAP\_Selected and \_MSVCRT\_HEAP\_SELECT which correlates back to my KERNEL32.dll functionality I saw in PEview. Additionally, I see more strings relating to further imports like User32.dll which could be loaded at runtime since I did not see them in PEview. Finally, strings show me a series of error messages and network related strings like Error 1.1 No Internet, Error 2.3 Fail to get command, Error 2.2: Fail to read file, and Error 2.1: Fail to openUrl, Success: Internet Connection, Internet Explorer 7.5/pma, Success: Parsed Command is %c, and a url to http://www.practicalmalwareanalysis.com/cc.htm (Figure 2). All of which further my idea that this program reaches out to the internet using HTTP protocols.

Moving on to dynamic analysis I decide to run the program within the lab. Before doing so I begin FakeNet because I believe that this program uses networking functionality. Also I start ProcMon and filter for my process name: Lab06-02.exe. Now that I have this setup I start double click the file and begin running. Immediately I see a shell appear on the screen with the statement, “Success: Internet Connection” (Figure 3). I have tricked the program into thinking it established a connection to the internet using FakeNet. Shortly thereafter, I get some data back in my FakeNet window, “Received new connection on port: 80” along with a GET request for /cc.htm HTTP/1.1, the User-Agent: Internet Explorer 7.5/pma, and the Host: www.practicalmalwareanalysis.com (Figure 5). I recognize all of these strings from my static analysis and have now confirmed that the program behaves like I suspected in the regard of reaching out to a website over HTTP. Looking at ProcMon I see that several registry values are set in the process of running this program all within HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings. Some notable values that are set are ProxyBypass, Autodetect, IntranetName, ProxyEnable, ProxyOverride, AutoConfigURL, etc (Figure 4). Also I see the TCP receive operations occur from 127.0.0.1:1039->127.0.0.1:80 (Figure 4). I exit the program and look at the WireShark capture where I look at the TCP stream and confirm once again that this program is reaching out to www.practicalmalwareanalysis.com over HTTP (Figure 6). Once static and dynamic analysis are complete it is time to move onto analyzing the cod constructs in IDA Pro.

# LAB 6-2

## LAB 6-2 Question 1

What operation does the first subroutine called by main perform?

Application

Description automatically generated with low confidence

*Figure 7: Main Function in Lab06-02.exe in IDA Pro*

The first subroutine called in Main is sub\_401000. Jumping to the subroutine 0x401000, I can see that the constructs appear to be an If statement (Figure 7). Two parameters are pushed onto the stack LPDWORD lpdwFlags and DWORD dwReserved and the function InternetConnectedState is called. Once the function is returned the result is compared. If zero, jump to loc\_40102B where the Error is returned, “Error: 1.1: No Internet”. Else, the function InternetConnectedState has returned true and the statement, “Success: Internet Connection” is pushed onto the stack and a call to Sub\_40117F before jumping to loc\_40103A and ending the function call.

## LAB 6-2 Question 2

What is the subroutine located at 0x40117F?

Graphical user interface, text, application

Description automatically generated

*Figure 8: Parameters pushed to call sub\_40117F*

*Table

Description automatically generated with medium confidence*

*Figure 9: The function of sub\_40117F in IDA Pro*

Returning to Main, we find a call to Sub\_40117F down two levels of if statements. The call is present in loc\_40115C. Here we see that before 0x40117F is called two parameters are added to the stack one is a formatted string, “Success: Parsed command is %c” and the other is a byte pointer returned from loc\_401148 (Figure 8). Looking further into Sub\_40117F it appears to be a printf statement. We see that the string is pushed onto the stack as well as the byte pointer used for formatting and then returned (Figure 9).

## LAB 6-2 Question 3

What does the second subroutine called by main do?

Text

Description automatically generated

*Figure 10: Identifying the call to sub\_401040 in IDA Pro*

*Timeline

Description automatically generated*

*Figure 11: Alternate view of the sub\_401040 function showing parameters and calls*

Once the result is returned by Sub\_401000 is returned and is not zero main jumps to loc\_401148. Within loc\_401148 a second subroutine is called: Sub\_401040 (Figure 10). In Sub\_401040 I identify several calls to functions that are related to the import functions of WININET.dll found during static analysis (Figure 1). I change views in IDA Pro to analyze this subroutine by pressing space bar to get a text view. I notice that several parameters are pushed onto the stack: Flags, ProxyBypass, Access type, and the user agent that I had seen several times in static and dynamic analysis: Internet Explorer 7.5/pma. I did a search of the WININET library for the function InternetOpen. What I found this function initializes the use of WININET’s functionality by passing the user-agent, the name of the proxy server, an optional list of host or IP addresses that will not be routed through when accesstype is set to Internet\_Open\_Type\_Proxy. Next, doing the same process by looking up InternetOpenUrlA I find that this function opens a specific resource by using a passed HTTP URL and additional parameters such as headers, header length, flags, the url, the parameter hInternet which handles the current session returned by InternetOpen, and context (Figure 11). In IDA Pro I see that after InternetOpenUrl is called the return hfile is then compared to zero, if it is not zero it jumps to loc\_40109D, else “Error 2.1: Fail to open URL” is printed using printf once again.

## LAB 6-2 Question 4

What type of code construct is used in this subroutine?

A picture containing text

Description automatically generated

*Figure 12: Analysis of the subroutine 0x401040*

*Table

Description automatically generated with low confidence*

*Figure 13: Deeper analysis after jnz to loc\_4010E5 from InternetReadFile*

In loc\_40109D we see that the file is added to EAX as well as NumberofBytesRead and NumberofBytestoRead. A buffer is added to the stack as well as the file then a call is made to the function InternetReadFile (Figure 12). Looking up the function InternetReadFile I learn that this function reads data from a handle opened by the function InternetOpenUrl. The parameters passed are: hfile, lpbuffer, dwNumberofBytesToRead, and lpdwNumberofBytesRead. Hfile is the handle returned by InternetOpenUrl, buffer is a pointer to a buffer that receives all the data, and the last two are self-explanatory. After InternetReadFile is returned it is compared to zero if not zero it jumps to loc\_4010E5, else printf “Error: 2.2: Fail to read file” (Figure 13). In loc\_4010E5 it appears that each byte is read and compared to different bytes. I was able to right-click within IDA Pro and change the bytes: 3ch, 21h, 20h, and 20h to: ‘<’, ‘!’, ‘-’, ‘-’. Together these form ‘<!—' which looks familiar as a comment in HTML. It appears the construct in this subroutine utilizes an array to store the characters in the variable Buffer (Figure 13).

## LAB 6-2 Question 5

Are there any network-based indicators for this program?

Text

Description automatically generated

*Figure 14: Closer look at the strings view of Lab06-02.exe*

*Graphical user interface, text, application

Description automatically generated*

*Figure 15: TCP stream from Lab06-02.exe network traffic*

*Text

Description automatically generated*

*Figure 16: HTTP connection from FakeNet while running Lab06-02.exe*

In the analysis of Lab06-02.exe we have gleaned network-based indicators of compromise. Strings showed us the url <http://www.practicalmalwareanalysis.com/cc.htm> and the user-agent Internet Explorer 7.5/pma (Figure 14). We then analyzed the TCP packets where we were able to clearly see that a GET request is made to the url for /cc.htm with the user-agent specified (Figure 15). Finally we were able to see this request and connection be established in real time using FakeNet (Figure 16). The user-agent, request, and host url are all network-based indicators of compromise in this program.

## LAB 6-2 Question 6

What is the purpose of this malware?

This program starts by checking the system for an active internet connection. If that is successful it will then reach out to the web address www.practicalmalwareanalysis.com and GET /cc.htm. from there it will parse the source of a webpage to search for a comment indicator in the form of <!--. Once this string set is found a character will be returned from the string and if the character is valid will return a message: “Success: Parsed command is %c”. After that it will sleep and then the program will end.

# LAB 6-3 Static & Dynamic Analysis

# Table Description automatically generated

*Figure 17: Imports used by Lab06-03.exe in PEview*

*A picture containing text

Description automatically generated*

*Figure 18: Strings returned from Lab06-03.exe*

*A picture containing text

Description automatically generated*

*Figure 19: Monitoring Lab06-03.exe while running through ProcMon*

*A screenshot of a computer

Description automatically generated*

*Figure 20: FakeNet Window while running Lab06-03.exe*

Like the first file, I began analyzing Lab06-03.exe using static and dynamic analysis techniques. I began by looking at imports in PEview. The structure of the imports is nearly identical to Lab06-02.exe except Lab06-03.exe shows the DLL library ADVAPI32.dll with functions relating to registry manipulation (Figure 17). Looking at strings there are also some similarities and differences as well. We see several R6## strings that seem to relate to error processing in addition to a few new errors not seen in the previous lab such as: Error 3.1: Could not set registry value and Error 3.2: Not a valid command provided. Also we see the string “Malware”, a registry path to: Software\Microsoft\Windows\CurrentVersion\Run, and paths: C:\Temp\cc.exe and C:\Temp (Figure 18).

Next, I moved to dynamic analysis and ran the program in the virtual lab. Since the program showed signs of network functionality I started FakeNet to capture any traffic. Looking at ProcMon I see registry manipulation occur as well as files being created (Figure 19). Looking back at FakeNet I see that I did in fact capture traffic in the form of a GET request on port 80 which is identical to what was seen in Lab06-02.exe (Figure 20).

**LAB 6-3**

## LAB 6-3 Question 1

Compare the calls in main to Lab 6-2 Solutions’s main method. What is the new function called from main?

Timeline

Description automatically generated

*Figure 21: Main function of Lab06-03.exe disassembled in IDA Pro*

*Question 1 Continued..*

The Main of Lab06-03.exe looks nearly the same as Lab06-02.exe with the exception of the subroutine Sub\_401130 being called after the initial setup of calling sub\_401000 to get a connection state then moving down to initiate and call the WININET functions and eventually landing on loc\_40123C which is where Sub\_401130 is called is everything prior is successful.

## LAB 6-3 Question 2

What parameters does this new function take?

Graphical user interface, text, application

Description automatically generated

*Figure 22: A look at the parameters pushed to the stack to call sub\_401130*

*Text

Description automatically generated*

*Figure 23: Parameters used in sub\_401130*

*Graphical user interface, text, application

Description automatically generated*

*Figure 24: Var\_8 used in loc\_401228 & passed to loc\_40123C*

Prior to Sub\_401130 being called we can see that two parameters are pushed to the stack: lpexistingFileName and Char (Figure 22). Looking closer at Sub\_401130 we see the two parameters are argv and var\_8 (Figure 23). Prior to the subroutine being called sub\_401040 is called where the webpage is accessed and the characters are parsed and compared to find the ‘<!--’ string. I conclude that the parsing of the comment string is being carried out to find a specific character to act as a command due to the fact that Sub\_401130 is called with var\_8 after the string “Success: Parsed command is %C” is printed. The character is then passed to Sub\_401130 (Figure 24).

## LAB 6-3 Question 3

What major code construct does this function contain?

A picture containing timeline

Description automatically generated

*Figure 25: Graphical view of Sub\_401130 in IDA Pro*

*Text

Description automatically generated*

*Figure 26: code found in Sub\_401130*

Taking a closer look at Sub\_401130 we see that the function takes many routes dependent on the ja (jump if above outcome). Prior to that instruction we see that EBP is pushed to the stack then ESP is moved to EBP and ESP is subtracted by 8. The EBP is then added to arg\_0 and moved into EAX which is then moved into Var\_8. So, Arg\_0 is moved into Var\_8. Because Arg\_0 is a holder in IDA Pro designating the last parameter added to the stack before the call to Sub\_401130 we know that Arg\_0 is the character obtained from the parsing in the previous function. Once Var\_8 containing the command character is moved to ECX it is subtracted by 61h. Right-clicking on 61h in IDA Pro we discover that this equates to ‘a’. There is a comparison carried out to the character – ‘a’ result to the 4 (Figure 26). This creates a conditional jump. If the condition is met four results can happen: the program will either create a directory, copy a file , Delete a file, open and set a registry key, or sleep for 100000 milliseconds. If the condition is not met an error is return using printf to say, “Error 3.2: Not a valid command provided”. This construct looks like switch statement using the character parsed in a jump table for the decision making on which path the code takes (Figure 25).

## LAB 6-3 Question 4

What can this function do?

Graphical user interface, application

Description automatically generated

*Figure 27: View of the Switch construct showing all possibilities*

The program will follow any of the paths seen in Question 3. It will either create a directory at the path: C:\\Temp, copy a file to C:\\Temp\\cc.exe, Delete a file C:\\Temp\\cc.exe, set a registry value at the path Software\Microsoft\Windows\CurrentVersion\Run\ with the value “Malware”, sleep for 100000 milliseconds, or if nothing else it will display an error.

## LAB 6-3 Question 5

Are there any host-based indicators for this malware?

Graphical user interface, text, application

Description automatically generated

*Figure 28: Closer look at loc\_40118C containing Host-Based Indicators*

The host-based indicators of compromise for this program would be the registry key that is set: \Software\Microsoft\Windows\CurrentVersion\Run\Malware. Additionally the file paths could be used as host-based indicators for the path C:\\Temp and C:\\Temp\\cc.exe.

## LAB 6-3 Question 6

What is the purpose of this malware?

This program starts by checking the system for an active internet connection. If that is successful it will then reach out to the web address www.practicalmalwareanalysis.com and GET /cc.htm. from there it will parse the source of a webpage to search for a comment indicator in the form of <!--. Then it will retrieve the first character and use that as a jump table index to run its next set of commands which will either be to do one of the following: create a directory, copy a file , Delete a file, open and set a registry key, or sleep for 100000 milliseconds, or return using printf to say, “Error 3.2: Not a valid command provided”.