

CNIT 58100 CFM: CYBERFORENSICS OF MALWARE – LAB 14

Ibrahim Waziri Jr

PhD in Information Security (CERIAS)

Lab 14

Instructor: **Associate Prof Sam Liles**

Purdue University

2014

Abstract

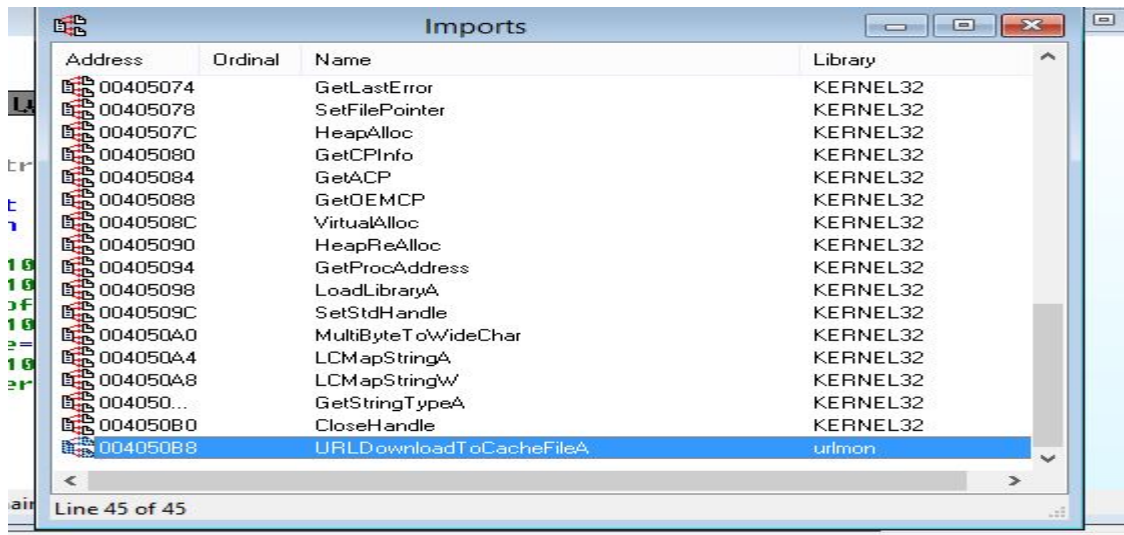
This lab covers the skills discussed in chapter 14 of the text. The practice covered in these labs is all based on malware analysis. The malware files used are provided as an extension of the text for practical purposes.

Each of the labs consists of multiple questions that require short answers. Depending on the question, certain special tools might be required to fully analyze the malware and find answers to the question.

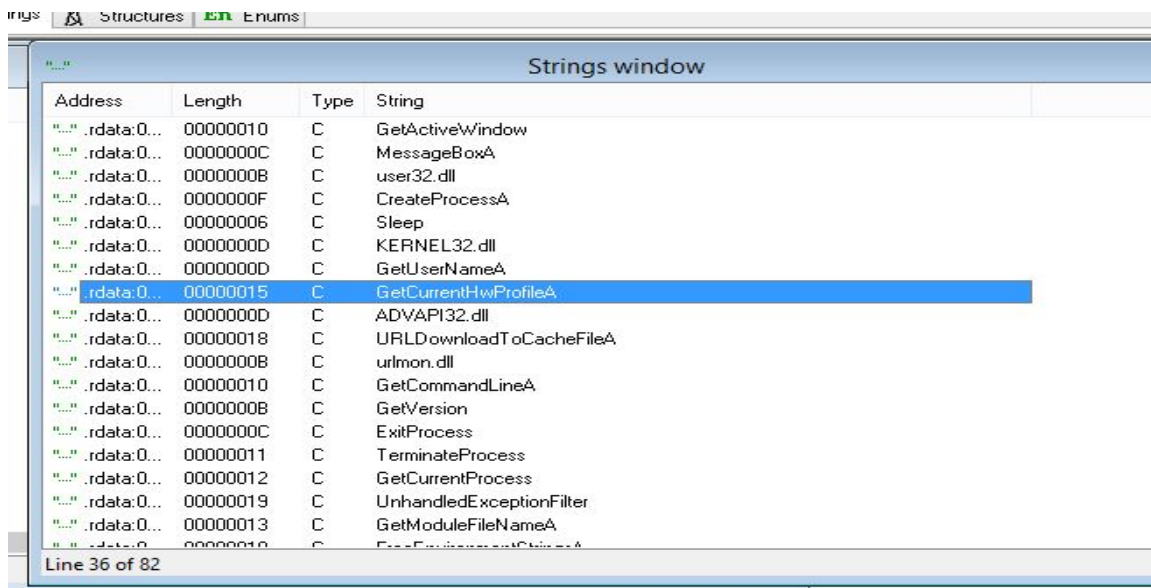
This paper provides answers to Chapter 14 labs. The lab uses 3 different files which are: *Lab14-01.exe*, *Lab14-02.exe* and *Lab14-03.exe*. These files are malwares are therefore could be harmful if used for non-training purposes.

Lab 14-1

- Using IDAPro we can see from the functions that the program contains the URLDownloadToCacheFile string/function, this is a networking library that uses the COM interface as shown below: When programs use COM interfaces, most of its HTTP content request comes from within the Operating System itself. Therefore the advantage of this is that cannot effectively be targeted using network signatures.



- To determine how the beacon is constructed, we first examine the strings. Within the strings we see GetCurrentHwProfileA as shown below:



```

push    ecx, [ebp+HwProfileInfo]
call    ds:GetCurrentHwProfileA
movsx   edx, [ebp+HwProfileInfo.szHwProfileGuid+24h]
push    edx
movsx   eax, [ebp+HwProfileInfo.szHwProfileGuid+23h]
push    eax
movsx   ecx, [ebp+HwProfileInfo.szHwProfileGuid+22h]
push    ecx
movsx   edx, [ebp+HwProfileInfo.szHwProfileGuid+21h]
push    edx
movsx   eax, [ebp+HwProfileInfo.szHwProfileGuid+20h]
push    eax
movsx   ecx, [ebp+HwProfileInfo.szHwProfileGuid+1Fh]
push    ecx
movsx   edx, [ebp+HwProfileInfo.szHwProfileGuid+1Eh]
push    edx
movsx   eax, [ebp+HwProfileInfo.szHwProfileGuid+1Dh]
push    eax
movsx   ecx, [ebp+HwProfileInfo.szHwProfileGuid+1Ch]
push    ecx
movsx   edx, [ebp+HwProfileInfo.szHwProfileGuid+1Bh]
push    edx
movsx   eax, [ebp+HwProfileInfo.szHwProfileGuid+1Ah]
push    eax
movsx   ecx, [ebp+HwProfileInfo.szHwProfileGuid+19h]
push    ecx
push    offset aCCCCCCCCCCC ; "%c%c:%c%c:%c%c:%c%c:%c%c:%c%c"
lea     edx, [ebp+var_10098]
push    edx ; char *

```

00401285: _main

Next we see the strings %c%c%c%c%c%c%c%c as shown below, this might appear gibberish to thorough analysis reveals that the strings is a username.

Address	Offset	Type	String
00401000	00000000	C	GetCurrentHwProfileA
00401001	0000000F	C	GetStringTypeA
00401002	0000000F	C	GetStringTypeW
00401003	00000011	C	FlushFileBuffers
00401004	0000000C	C	CloseHandle
00401005	00000032	C	http://www.practicalmalwareanalysis.com/%s/%c.png
00401006	0000001E	C	%c%c:%c%c:%c%c:%c%c:%c%c:%c%c
00401007	00000006	C	%s-%s

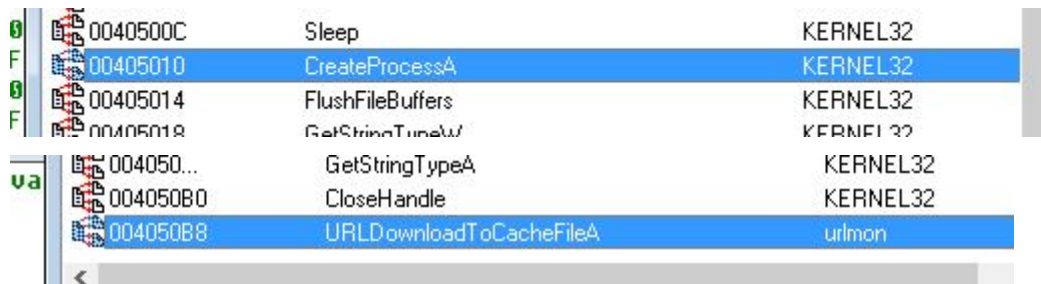
Page 36 of 82

Therefore, from the string GetCurrentHwProfileA, we can tell that the element is a part of the host GUID, and the username can change whenever the login user is changed.

- Because the attacker might want to track the computer running the program and also target any user depending on the login information.
- We can see the malware encoding system below, and it is not a standard Base64 because it doesn't use = sign for its padding

Address	Length	Type	String
00401000	00000033	C	BCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
00401001	0000000C	C	123456789+/-
00401002	00000008	C	{8P%\a\b
00401003	00000007	C	700wP\

5. Two strings CreateProcessA and URLDownloadToCacheFileA shown below tell us that the malware downloads a file and executes it by creating a process.



0040500C	Sleep	KERNEL32
00405010	CreateProcessA	KERNEL32
00405014	FlushFileBuffers	KERNEL32
00405018	GetStringT...	KERNEL32
004050...	GetStringTypeA	KERNEL32
00405080	CloseHandle	KERNEL32
00405088	URLDownloadToCacheFileA	urlmon

6. The element of the malware communication to be targeted and detected using a network signature will be the PNG single file that is being referenced to from the string as shown below: <http://www.practicalmalwareanalysis.com/%s/%c.png>.



data:00...	0000000C	C	CloseHandle
data:00...	00000032	C	http://www.practicalmalwareanalysis.com/%s/%c.png
data:00...	0000001E	C	%c%c:%c%c:%c%c:%c%c:%c%c:%c%c
data:00...	00000006	C	%s-%s

7. Considering that the standard Base64 usually starts with an = sign and this malware uses a instead, an analyst might easily mistakenly the encryption to be a standard Base64. Also considering that the file ends with PNG might be a bit confusing is an analyst doesn't have the knowledge of file formats.

Lab 14-2

- 1- An attacker might find static IP addresses more difficult to manage than domain names. Using DNS allows the attacker to deploy his assests to any computer and dynamically redirects his bots by changing only a DNS address.
- 2- Analyzing the strings, we see 4 different DLL files and those are the networking libraries this malware uses. These DLL files are: KERNEL32.dll, USER32.dll, SHELL32.dll and WININET.dll as shown in the figure below:

CNIT 581: CyberForensics of Malware – Lab 14

..rdata:0...	00000010	C	GetEnvironmentVariableA
..rdata:0...	00000012	C	GetShortPathNameA
..rdata:0...	00000013	C	GetModuleFileNameA
..rdata:0...	0000000D	C	KERNEL32.dll
..rdata:0...	0000000C	C	LoadStringA
..rdata:0...	0000000B	C	USER32.dll
..rdata:0...	0000000F	C	SHChangeNotify
..rdata:0...	00000010	C	ShellExecuteExA
..rdata:0...	0000000C	C	SHELL32.dll
..rdata:0...	00000014	C	InternetCloseHandle
..rdata:0...	00000011	C	InternetOpenUrlA
..rdata:0...	0000000E	C	InternetOpenA
..rdata:0...	00000011	C	InternetReadFile
..rdata:0...	0000000C	C	WININET.dll
..rdata:0...	00000005	C	free
..rdata:0...	00000007	C	"

With a focus on the WININET.dll, one advantage of this library is that elements such as cookies and cache are provided by the operating system.

- Examining the resource section, within the address 0x4011C0 we see a call to LoadStringA as shown below:

```

push    ecx                ; hInstance
call    ds:LoadStringA
mov     esi, ds:CreateEventA
xor     ebp, ebp
push    ebp                ; lpName
push    ebp                ; bInitialState

```

To understand what source of the URL the malware uses for beaconing, we have to follow the call to LoadStringA we see, and the result is shown in the figure below:

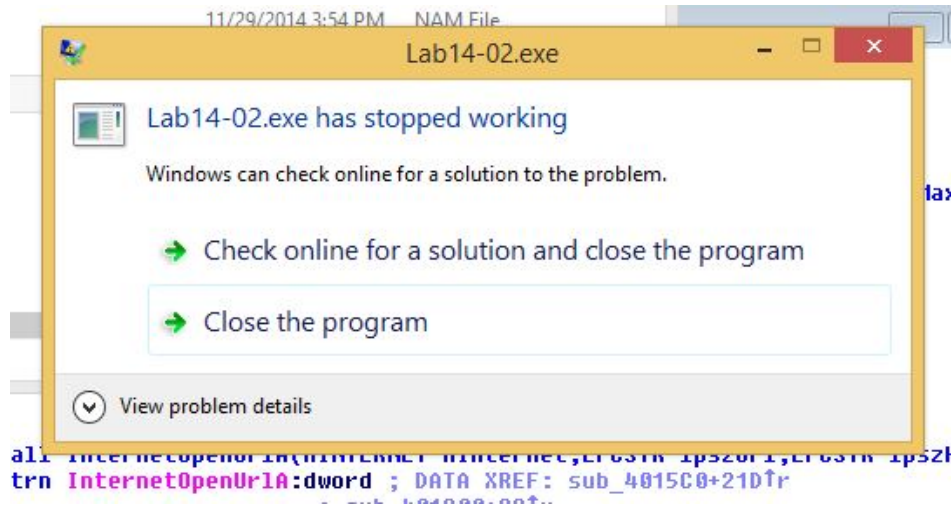
```

..rdata:00402000 ; Imports from USER32.dll
..rdata:0040200C ; Imports from USER32.dll
..rdata:0040200C ; int __stdcall LoadStringA(HINSTANCE hInstance,UINT uID,LPSTR lpBuffer,int nBufferMax)
..rdata:0040200C ; extrn LoadStringA:dword ; DATA XREF: MinMain(x,x,x,x)+217r
..rdata:00402000 ;
..rdata:00402004 ; Imports from WININET.dll
..rdata:00402004 ;
..rdata:00402004 ; BOOL __stdcall InternetCloseHandle(HINTERNET hInternet)
..rdata:00402004 ; extrn InternetCloseHandle:dword
..rdata:00402004 ; ; DATA XREF: sub_4015C0:loc_4017EB7r
..rdata:00402004 ; ; sub_4015C0+2327r ...
..rdata:00402008 ; HINTERNET __stdcall InternetOpenUrlA(HINTERNET hInternet,LPCSTR lpszUrl,LPCSTR lpszHea
..rdata:00402008 ; extrn InternetOpenUrlA:dword ; DATA XREF: sub_4015C0+2107r
..rdata:00402008 ; ; sub_401800+297r
..rdata:0040200C ; HINTERNET __stdcall InternetOpenA(LPCSTR lpszAgent,DWORD dwAccessType,LPCSTR lpszProxy,
..rdata:0040200C ; extrn InternetOpenA:dword ; DATA XREF: sub_4015C0+2047r
..rdata:0040200C ; ; sub_401800+107r

```

From the above we can see the library WININET.dll and there is no singled out URL the malware uses, however the malware seems to have control of certain internet functions like: InternetCloseHandle, InternetOpenUrlA and InternetOpenA as identified in the figure above.

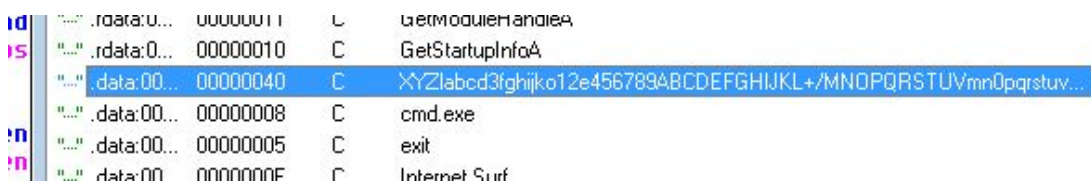
4. From the analysis of question 3 above, it looks like for the malware to achieve its objective, it abuses the HTTP User-Agent which contains the application information.
5. We try to run the malware; however the malware seems to be crashing on its own as shown in the figure below:



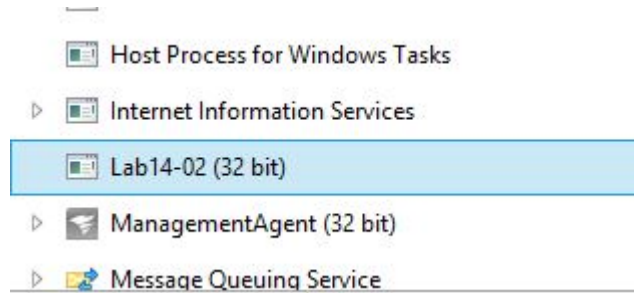
But it appears the malware the information communicated in the malware initial beacon is encoded command-shell prompt because when we try to run the program using the cmd, it appears to run normal as shown below:

```
C:\Users>cd analyst
C:\Users\analyst>cd desktop
C:\Users\analyst\Desktop>cd chapter_14L
C:\Users\analyst\Desktop\Chapter_14L>lab14-02.exe
C:\Users\analyst\Desktop\Chapter_14L>
```

6. The outgoing information appears to be encoded which is a lot of work for the attacker, considering the server dependency can be targeted with signatures.
7. Yes, the malware encoding scheme is a standard Base64 with alpha-numeric characters and symbols as shown below:



8. The communication seems to be terminated only by ending the process using Task Manager as shown in the figure below:

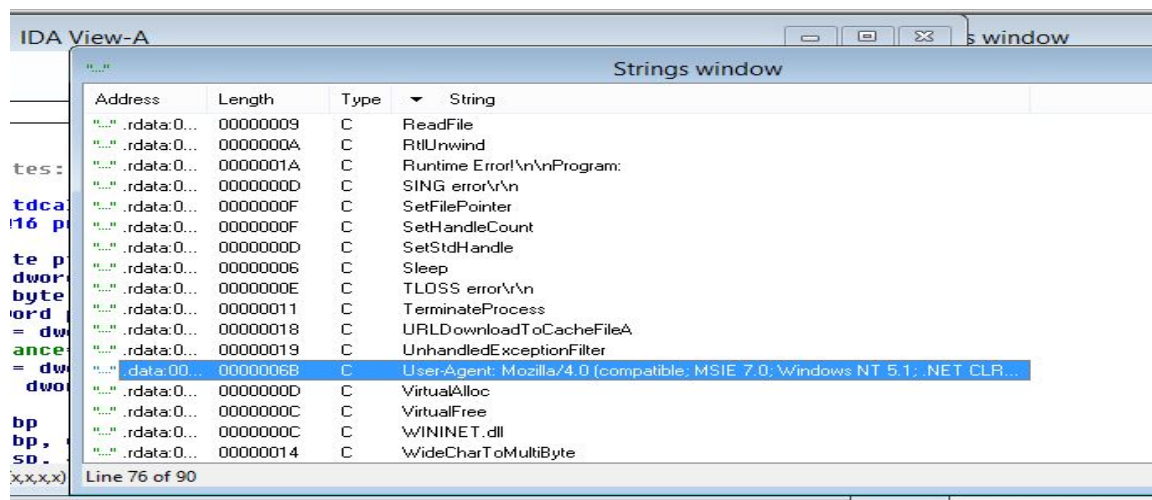


9. Analysis of then imports, strings and functions (with reference to figure above) tells us that the purpose of the malware is to provide a command-shell interface to a remote attacker that wont be detected by common signatures that watch for outbound command-shell activity.

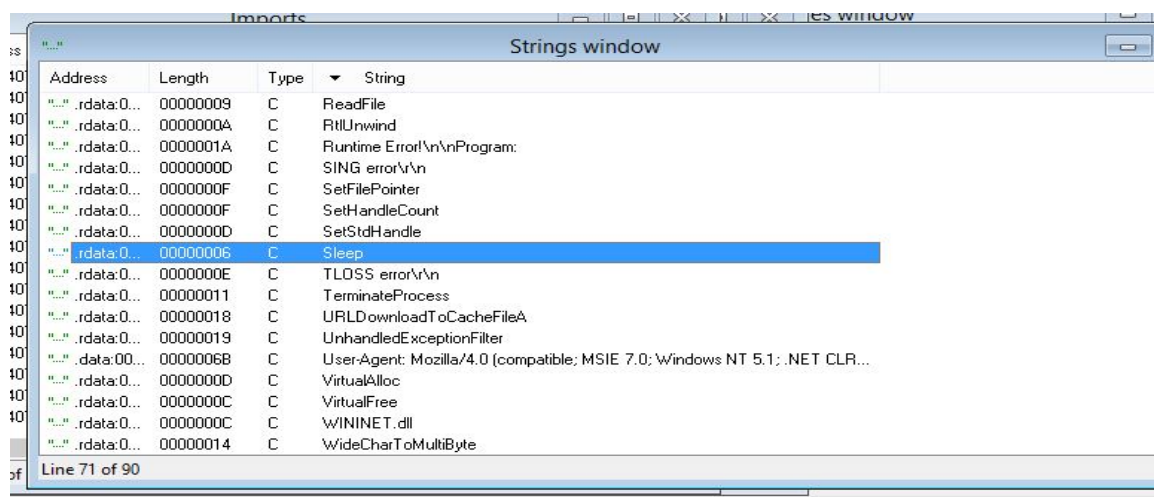
Lab 14-3

This program gave me a lot of problems, running this file even though its stated that it is not harmful, it results to me using a different snapshot virtual machine environment. I actually had to set up twice because it keeps failing and crashing

1. Using OllyDbg to analyze this program, we see that the hard-corded headers include Accept-Encoding and User-Agent. The complete user-agent header will make an effective signature as shown below



2. Both the domain name and path of the URL are hard-coded only where the configuration file is unavailable.
3. For some reason, whenever I try to further any thorough analysis using the program Lab14-03 it keeps failing and crashing my system.
4. Same issue as to what question 3 gives
5. View the imports and functions the malware doesn't seem to have any encoding system.
6. From the strings shown below we can see commands like quit, download, sleep and redirect.



The screenshot shows a 'Strings window' from a debugger. It displays a list of memory addresses, lengths, types, and strings. The 'Sleep' function is highlighted in blue. Other visible strings include 'ReadFile', 'RtlUnwind', 'Runtime Error!\n\nProgram:', 'SING_error!\n\n', 'SetFilePointer', 'SetHandleCount', 'SetStdHandle', 'TLOSS_error!\n\n', 'TerminateProcess', 'URLDownloadToCacheFileA', 'UnhandledExceptionFilter', 'User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR...', 'VirtualAlloc', 'VirtualFree', 'w\NINET.dll', and 'WideCharToMultiByte'.

Address	Length	Type	String
00000009	1	C	ReadFile
0000000A	1	C	RtlUnwind
0000001A	1	C	Runtime Error!\n\nProgram:
0000000D	1	C	SING_error!\n\n
0000000F	1	C	SetFilePointer
0000000F	1	C	SetHandleCount
0000000D	1	C	SetStdHandle
00000006	1	C	Sleep
0000000E	1	C	TLOSS_error!\n\n
00000011	1	C	TerminateProcess
00000018	1	C	URLDownloadToCacheFileA
00000019	1	C	UnhandledExceptionFilter
0000006B	1	C	User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR...
0000000D	1	C	VirtualAlloc
0000000C	1	C	VirtualFree
0000000C	1	C	w\NINET.dll
00000014	1	C	WideCharToMultiByte

7. From the strings shown above, we can tell that the malware is a downloader with a web-based control and flexibility to adjust as a malicious domain.

Conclusion:

Considering malwares uses heavy network connectivity. This lab shows us how to develop effective network-based countermeasures used to respond to threats and to detect or prevent malicious activities.