

CNIT 58100 CFM: CYBERFORENSICS OF MALWARE – LAB 13

Ibrahim Waziri Jr

PhD in Information Security (CERIAS)

Lab 13

Instructor: **Associate Prof Sam Liles**

Purdue University

2014

Abstract

This lab covers the skills discussed in chapter 13 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 13 labs. The lab uses 3 different files which are: *Lab13-01.exe*, *Lab13-02.exe* and *Lab13-03.exe*. These files are malwares are therefore could be harmful if used for non-training purposes.

Lab 13-1

We start by running some basic and dynamic analysis, and then making reference to the analysis to answer the questions:

Values added:2
 HKU\S-1-5-21-3280545418-1106663961-1194358563-500\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_EHACNGU-P:\Qbphzra
 HKU\S-1-5-21-3280545418-1106663961-1194358563-500\Software\Microsoft\Windows\Shell\NoRoam\MUICache\{C:\Documents and Settings\Administrator\Desktop\Chapter_13\Lab13-01.exe: "Lab13-01"

Fig 1: Analysis using Regshot



Fig 2: Exports using IDAPro

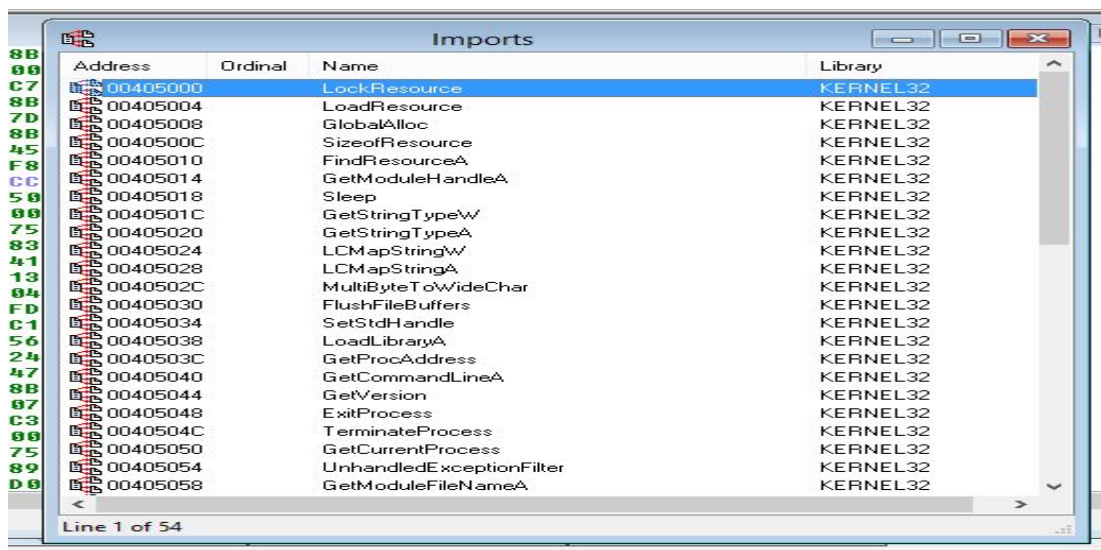


Fig 3: Imports from IDAPro

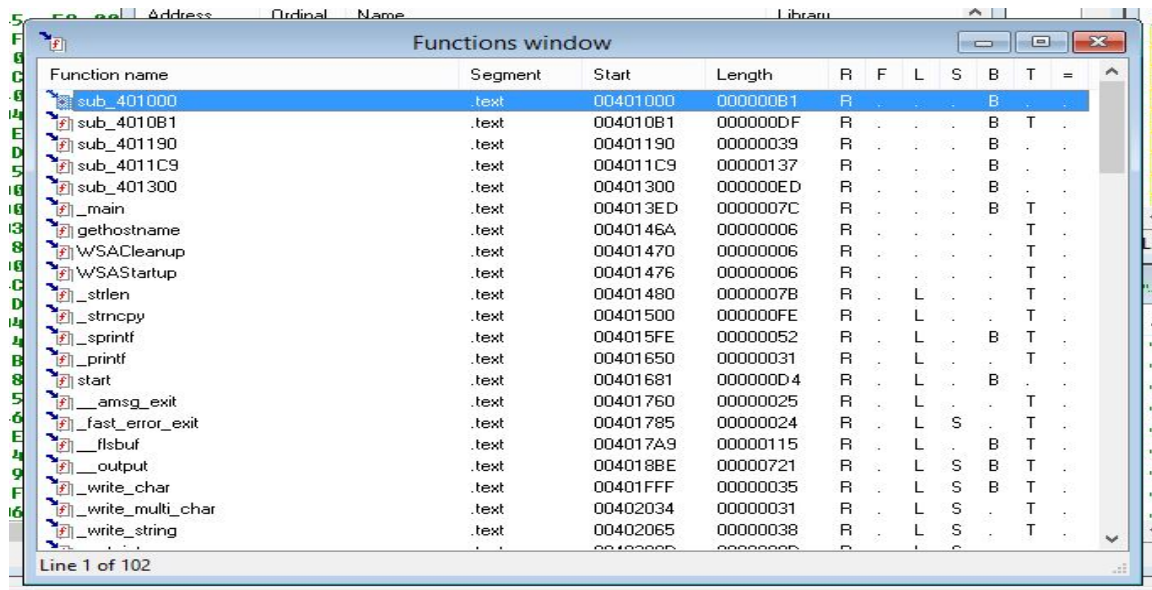


Fig 4: Imports from IDAPro

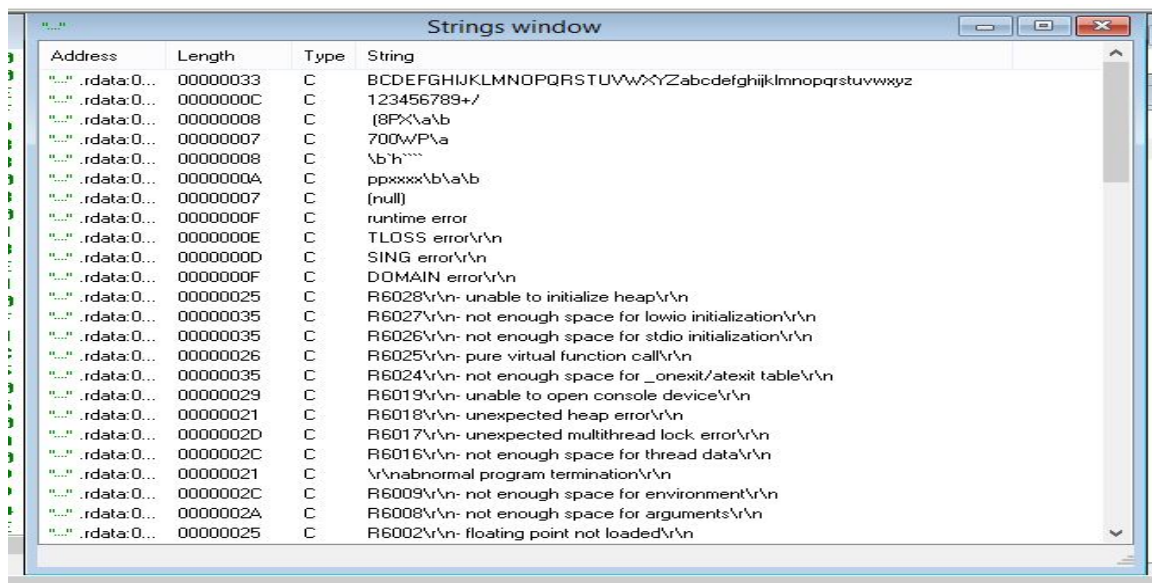
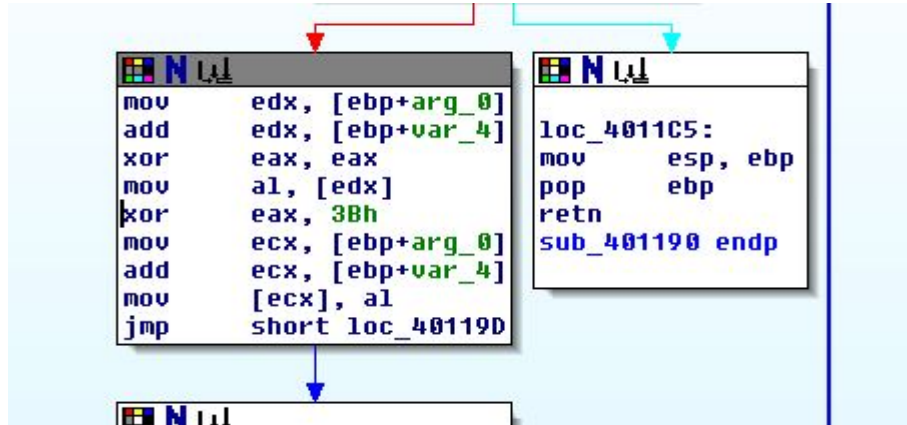
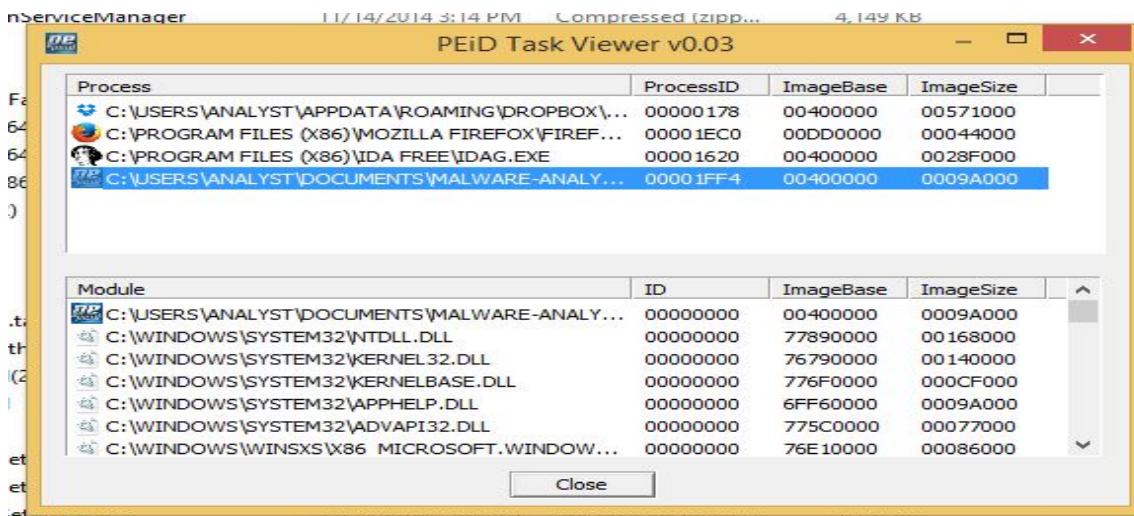


Fig 5: Strings from IDAPro

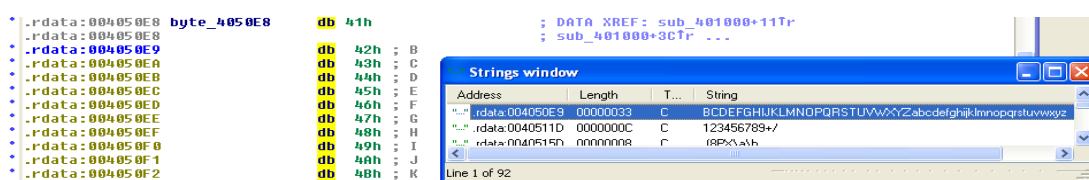
1. With reference to Fig 5, the output strings which elements might be encoded are www.practicalmalwareanalysis.com and the other is a GET request path.
2. From figure below: we can see that xor instruction at 004011B8 leads to a single-byte XOR-encoding loop in sub_4011B8.



- Identifying the content affected by xorEncode. The function sub_401300 is the only one that calls xorEncode. Tracing its code blocks that precede the call to xorEncode as shown in Figure above we see the single-byte XOR encoding uses the byte 0x38h. The raw data resource with index 101 is an XOR-encoded buffer that decodes to www.practicalmalwareanalysis.com
- An attempt to use all stated plugins was not responsive, however we tried to run PEiD and the result we got is shown in the figure below:



However we can see the string shown in the figure below, and that is a base 64 encoding system.



5. We couldn't find the type of encoding system for the portion of the network traffic, but considering base 64 is the standard encoding system we assume it is.
6. At the call function of 0x0040122A and 0x004010B1 as shown in the analysis.

Lab 13-2

1. Running the malware creates large random files in its current directory were created, and the files seems to be of large size approx. 4MB as shown below

Lab13-01.til	11/24/2014 7:28 PM	TIL File	1 KB
<input checked="" type="checkbox"/> Lab13-02	11/14/2011 3:47 PM	Application	32 KB
Lab13-03	11/17/2011 6:04 PM	Application	76 KB
temp05a0c19e	11/24/2014 8:58 PM	File	4,001 KB
<input type="checkbox"/> temp05a0e9c7	11/24/2014 8:59 PM	File	4,001 KB
temp05a01f14	11/24/2014 8:58 PM	File	4,001 KB
temp05a1b322	11/24/2014 8:59 PM	File	4,001 KB
temp05a1db4b	11/24/2014 9:00 PM	File	4,001 KB
temp05a2a581	11/24/2014 9:00 PM	File	4,001 KB
temp05a2cdd9	11/24/2014 9:01 PM	File	4,001 KB
temp05a13a77	11/24/2014 8:59 PM	File	4,001 KB
temp05a18ae9	11/24/2014 8:59 PM	File	4,001 KB
temp05a22c69	11/24/2014 9:00 PM	File	4,001 KB
temp05a27d19	11/24/2014 9:00 PM	File	4,001 KB
temp05a047ab	11/24/2014 8:58 PM	File	4,001 KB
temp05a098d8	11/24/2014 8:58 PM	File	4,001 KB
temp05a162a1	11/24/2014 8:59 PM	File	4,001 KB
temp05a203c3	11/24/2014 9:00 PM	File	4,001 KB
temp05a254c1	11/24/2014 9:00 PM	File	4,001 KB
temp05a1124e	11/24/2014 8:59 PM	File	4,001 KB
temp05a07022	11/24/2014 8:58 PM	File	4,001 KB
temp059d9666	11/24/2014 8:55 PM	File	4,001 KB
temp059dbe9f	11/24/2014 8:55 PM	File	4,001 KB
temp059de6e7	11/24/2014 8:55 PM	File	4,001 KB

2. From the analysis it seems like the malware uses the xor and move function to hide its identity. We can see that in the figure below:

```

var_44= dword ptr -44h
arg_0= dword ptr 8
arg_4= dword ptr 0Ch

push    ebp
mov     esp, esp
sub     esp, 44h
push    44h
push    0
lea     eax, [ebp+var_44]
push    eax
call    _memset
add     esp, 0Ch
mov     ecx, [ebp+arg_4]
mov     edx, [ebp+arg_0]
push    edx
mov     eax, [ebp+arg_0]
push    eax
lea     ecx, [ebp+var_44]
push    ecx
call    sub_401739
add     esp, 10h
mov     esp, ebp
pop     ebp
retn
sub_40181F endp

```


3. With reference to the figure below: It appears that the best prospect for finding the encoding function will be the call to WriteFile.

```
; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use32
assume cs:_text
;org 401000h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing

; Attributes: bp-based frame

; int __cdecl sub_401000(LPCVOID lpBuffer,DWORD nNumberOfBytesToWrite,LPCSTR lpFileName)
sub_401000 proc near

hObject= dword ptr -10h
NumberOfBytesWritten= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4
lpBuffer= dword ptr 8
nNumberOfBytesToWrite= dword ptr 0Ch
lpFileName= dword ptr 10h

push    ebp
mov     ebp, esp
```

4. Reference to the above figure, we can see that the encoding function is sub_40181F.
5. Screen capture is the source content because in the string list as shown in figure below, we can see files like GetDesktopWindow. Such string files indicate that screen capture is the source content.

[" .rdata:0...	0000000D	C	KERNEL32.dll
[" .rdata:0...	0000000A	C	ReleaseDC
[" .rdata:0...	00000006	C	GetDC
[" .rdata:0...	00000011	C	GetDesktopWindow
[" .rdata:0...	00000011	C	GetSystemMetrics
[" .rdata:0...	0000000B	C	USER32.dll
[" .rdata:0...	0000000D	C	DeleteObject
[" .rdata:0...	00000009	C	DeleteDC
[" .rdata:0...	0000000A	C	GetDIBits
[" .rdata:0...	0000000B	C	GetObjectA
[" .rdata:0...	00000007	C	BitBlt
[" .rdata:0...	0000000D	C	SelectObject

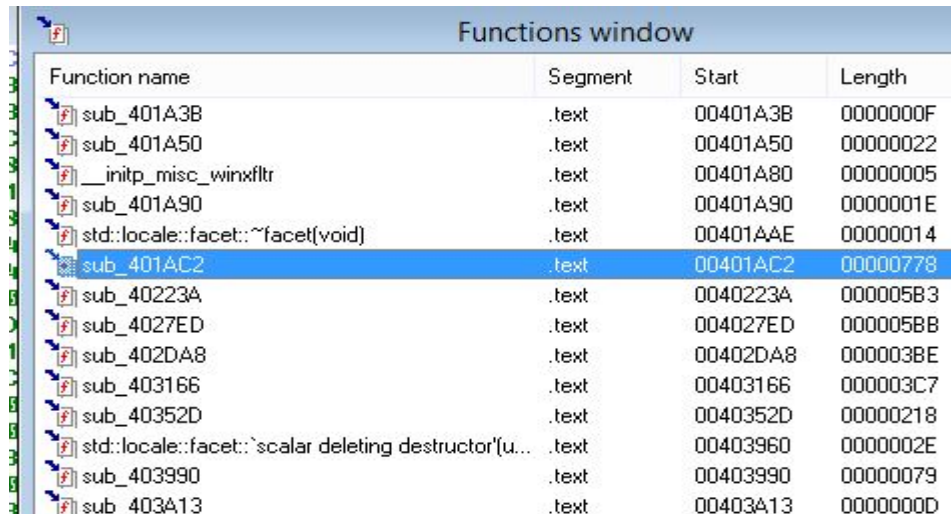
6. A breakdown of the algorithm is a bit hard to comprehend.

Lab 13-3

1. Dynamic analysis brings some gibberish string, which looks encoded. Decoding the strings to make comparison with the strings found from basic analysis. As shown in the figure below, the strings obtained from dynamic analysis using IDAPro does not make any sense. Further analysis will be conducted using OllyDbg.

Address	Length	Type	String
"..." rdata:0...	00000006	C	~=zG=d
"..." rdata:0...	00000008	C	\'D\'\'*T~*
"..." rdata:0...	00000009	C	2dV2:tN:\n
"..." rdata:0...	00000005	C	\$Hl\$\\
"..." rdata:0...	00000005	C	7nY7m
"..." rdata:0...	00000009	C	x%Jo%.\r.
"..." rdata:0...	00000005	C	p> B>
"..." rdata:0...	00000006	C	a5j_5w
"..." rdata:0...	00000005	C	U(Px{
"..." rdata:0...	00000006	C	ggV}++
"..." rdata:0...	0000000C	C	Lj&&Z66~A??
"..." rdata:0...	00000005	C	h\\44Q
"..." rdata:0...	00000006	C	bS11*?
"..." rdata:0...	00000005	C	\\a\\a\$6
"..." rdata:0...	00000006	C	Xt,,4.
"..." rdata:0...	00000006	C	RRvM,;
"..." rdata:0...	00000006	C	MMfU33
"..." rdata:0...	00000007	C	PPxD<<%
"..." rdata:0...	00000006	C	Bc!! 0
"..." rdata:0...	00000006	C	~~zG==
"..." rdata:0...	00000009	C	DR\'\'*T~*;
"..." rdata:0...	00000008	C	dV22tN::

2. Using static analysis, we see six different xor instruction functions that are associate with encoding as shown in the figure below: We decided to view the functions of Lab13-03.exe using IDAPro. Because The functions are not organized in order and there is no way to organize it in IDAPro, we decided to only show a screen shot of one function and list the other functions, which are: 0x00401AC2, 0x0040223A, 0x004027ED, 0x00402DA8, 0x00403166, 0x00403990.



Function name	Segment	Start	Length
sub_401A3B	.text	00401A3B	0000000F
sub_401A50	.text	00401A50	00000022
__initp_misc_winxfltr	.text	00401A80	00000005
sub_401A90	.text	00401A90	0000001E
std::locale::facet::~facet(void)	.text	00401AAE	00000014
sub_401AC2	.text	00401AC2	00000778
sub_40223A	.text	0040223A	000005B3
sub_4027ED	.text	004027ED	000005B8
sub_402DA8	.text	00402DA8	000003BE
sub_403166	.text	00403166	000003C7
sub_40352D	.text	0040352D	00000218
std::locale::facet::~scalar deleting destructor(u...	.text	00403960	0000002E
sub_403990	.text	00403990	00000079
sub_403A13	.text	00403A13	0000000D

- Using IDA Pro Entropy Plugin we identify a custom base64 indexing string as shown in Figure below, however we cant identify any association with xor instruction. The string contains alphabetic and numeric characters along with some symbols as shown below



- From the string shown above we can tell that the encoding technique used by this malware is a custom Base64 cipher. And the malware uses AES
- The key for the custom Base64 cipher is the index string shown in the figure above, which is:
DEFKHIJKLMNOPQRSTUVWXYZABcdefghijklmnopqrstuvwxyzab0123456789+/
Yes, for the cryptographic encryption algorithm. The key is sufficient enough to decode the Base64 encrypted information.
- The malware establishes a reverse command shell with the incoming commands decoded using custom Base64 cipher and the outgoing command-shell responses encrypted with AES.

Conclusion

These labs show us how malware uses encoding techniques to mask its malicious activities. It exposes us to understand these techniques in order to fully understand malware activities