Xiaoyi Zhou
Purdue University CIT581
Malware Forensics
Lab 13: Data Coding
Due November 11, 2014
Instructor: Samuel Liles

Abstract

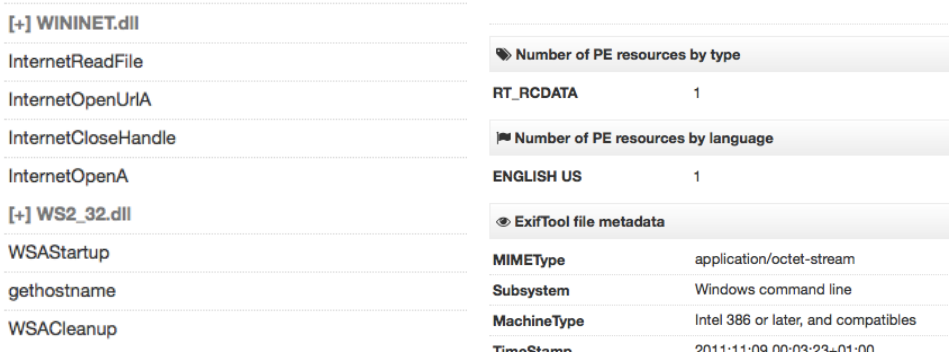
This lab focuses on how to decrypt the encrypted data. However the encryption cipher is difficult to write by an algorithm or program. Sometime even though the plugins don't work, we can still detect and determine the cipher for example when the malware uses XOR or base64 (standard or customized). At Lab13-3, the malware uses AES which is a little bit difficult to detect. But if the user is familiar with the schema of AES cipher, it is still possible to determine.

The decryption process is difficult if the user only relies on algorithm. OllyDbg provides an effective way to decrypt cipher especially when the cipher is XOR because double XOR means plaintext. I don't think the same method could be applied to other ciphers except XOR. For example we cannot base64 or complicated DES/AES. This Lab help review the previous cryptography knowledge that I almost forgot.

Lab13-01

Steps of Process

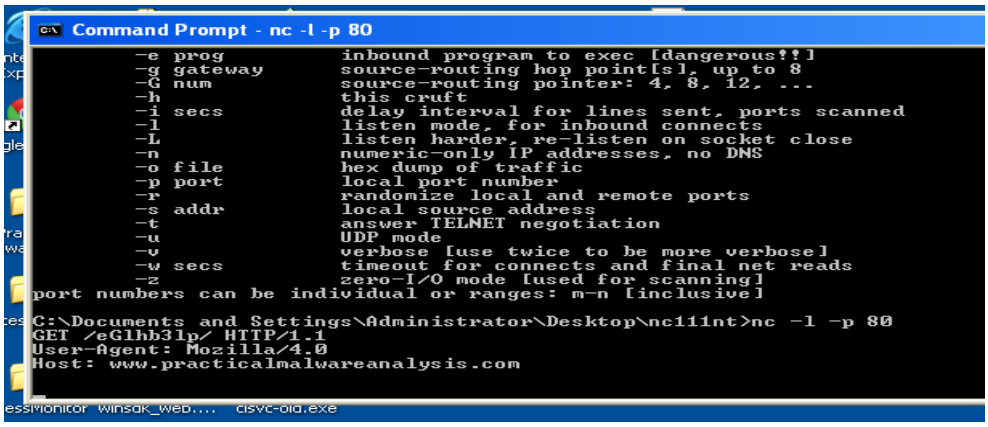
This malware contains encrypted information because VirusTotal has detected Malware-Cryptor.InstallCore.1. First we examine the imports and exports. This malware intends to intercept with Internet because we can notice a lot of Internet indicator such as gethostname, WSAStartup, InternetReadFile and etc. Besides the Internet indicator, we also noticed FindResourceFile and LoadResourceFile which means the malware has a PEfile at resource section. To guess the purpose of the malware, I use string and made a conclusion that this malware is not powerful because I didn't see a lot of memory mapping neither loading (unloading) driver operations. So we can try to run this malware and see the details.



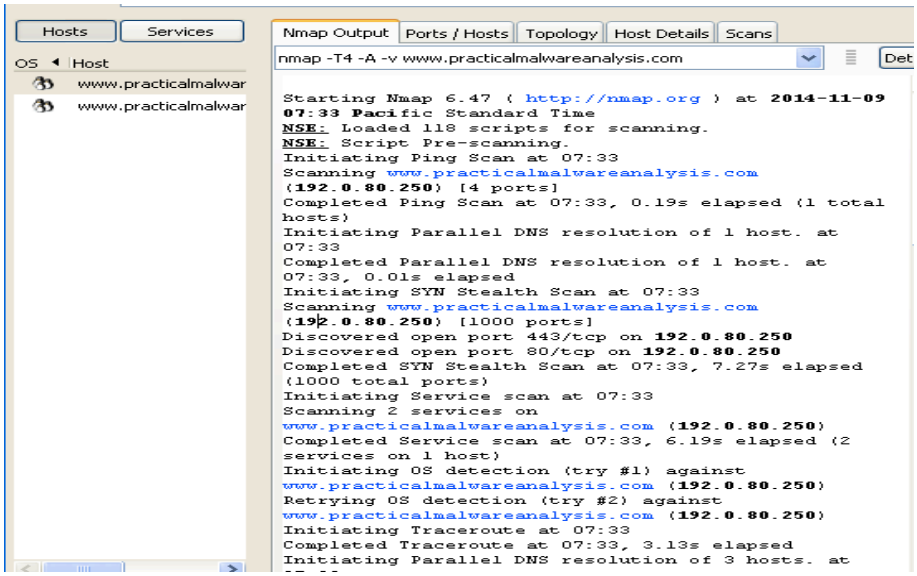
[+] WININET.dll
InternetReadFile
InternetOpenUrlA
InternetCloseHandle
InternetOpenA
[+] WS2_32.dll
WSAStartup
gethostname
WSACleanup

Number of PE resources by type	
RT_RCDATA	1
Number of PE resources by language	
ENGLISH US	1
ExifTool file metadata	
MIMEType	application/octet-stream
Subsystem	Windows command line
MachineType	Intel 386 or later, and compatibles
TimeStamp	2011:11:09 00:03:23+01:00

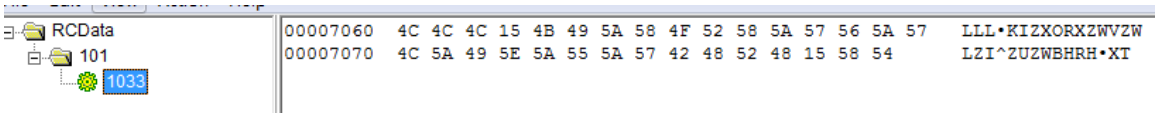
Since I already realized that the malware intercepts with Internet, I use apateDNS, ZenMap, and netcat to listen on port 80. The result if very interesting because I didn't notice the host address and the beacon after GET command at strings. Therefore the host name and beacon string might be encrypted. I noticed user-agent at string so it is a plaintext.



We put the malware into IDA Pro.



The goal is to find any cipher indicator and decrypt it. The first function at main function is sub_401300. This function is right before WSAStartup. The return value of sub_401300 is lpWSAData that contain the operations of window socket implementation. Therefore we can guess the purpose of sub_401300 is to find the domain address and resolve Internet command. I rename sub_401300 as WSA_Struct. Double click on it and I found that the first purpose is to free the PEfile at recourse section. The program calls FindResourceA, SizeofResource, LoadResource and LockResource to find the resource and pass the section pointer to eax. The pointer then is passed to edx and being a parameter at sub_401190. At sub_401190, I noticed a key operation "XOR" with cipher key 0x3B. It could be the cipher that decrypts the information at resource section. I rename sub_401190 as Cipher. The loop at Cipher starts at 0x40119D and ends at 0x4011C3. eax is counter that increases by one each time the cipher is implemented. If the number of counter is greater than the length of ciphertext, then the loop will break. To decrypt the information, first we need to extract the PEfile from resource section. Second we decrypt it at WinHex.



```
00007010  31 3B 3B 3B 23 3B 3B BB  3B 3B 3B 3B 3B 3B 3B 3B  1;;;#;;»;;;;;;;;
00007020  3B 3B 3B 3B 3B 3B 3A 3B  5E 3B 3B 3B 0B 3B 3B BB  ;;;;;;;:^;;; ;;»
00007030  3B 3B 3B 3B 3B 3B 3B 3B  3B 3B 3B 3B 3B 3B 3A 3B  ;;;;;;;;;;;;;;:;
00007040  32 3F 3B 3B 73 3B 3B 3B  63 BB 3B 3B 1B 3B 3B 3B  2?;;s;;;c»;; ;;;
00007050  3B 3B 3B 3B 3B 3B 3B 3B  77 77 77 2E 70 72 61 63  ;;;;;;;;www.prac
00007060  74 69 63 61 6C 6D 61 6C  77 61 72 65 61 6E 61 6C  ticalmalwareanal
00007070  79 73 69 73 2E 63 6F 6D  3B 3B 3B 3B 3B 3B 3B 3B  ysis.com;;;;;;;;
```

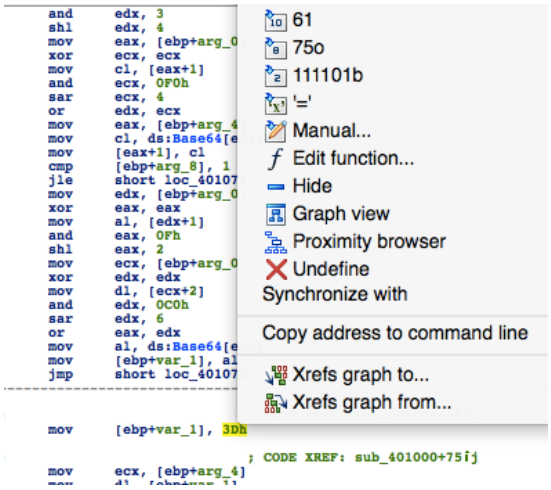
The above result is the plaintext. Also, I notice the ciphertext at strings.

After the program get the host address, it will sleep for few seconds and call sub_4011C9. Most of Internet indicators are at sub_4011C9. Therefore I guess the purpose of this function is to use Internet command to send request from Internet. I rename it as Internet_cmd. Double click on it and I can notice the user-agent string Mozilla/4.0. The user-agent string is shown as plaintext like I mentioned before. The next important function call is gethostname that takes length and name pointer as parameter. Here the name pointer is stored at ebp which is the parameter passed from Internet_cmd. At loc_40142E, we can notice that Internet_cmd only takes one parameter [ebp+var_19C]. This is a pointer to return value of ealier WSA_Struct( ).The return valye is the host address www.practicalmalwareanalysis.Therefore gethostname is used to find www.practicalmalwareanalysis.com. Then the program calls strcnpy to copy the first 12 characters of host name. (0xC is 12 in decimal). The result will be stored at destination parameter [ebp+var_18]. Later [ebp+var_18] is passed to sub_4010B1. There are two loops at this function. The first loop is start from 1 to 4. The second loop that is within the first loop starts from 1 to 3. So we can assume the program is using base64 cipher to encrypt data from 3bytes into 4bytes. Besides, there is a good base64 indicator at strings. The next goal is to find the base64table and the padding indicator as well.

```
;=@n@
@m@
tiW
Yt<
%\i@
Xi@
 ^][
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
EEE
 (8PX
?00WP
`h````
ppxxxx
(null)
(null)
runtime error
TLOSS error
SING error
```

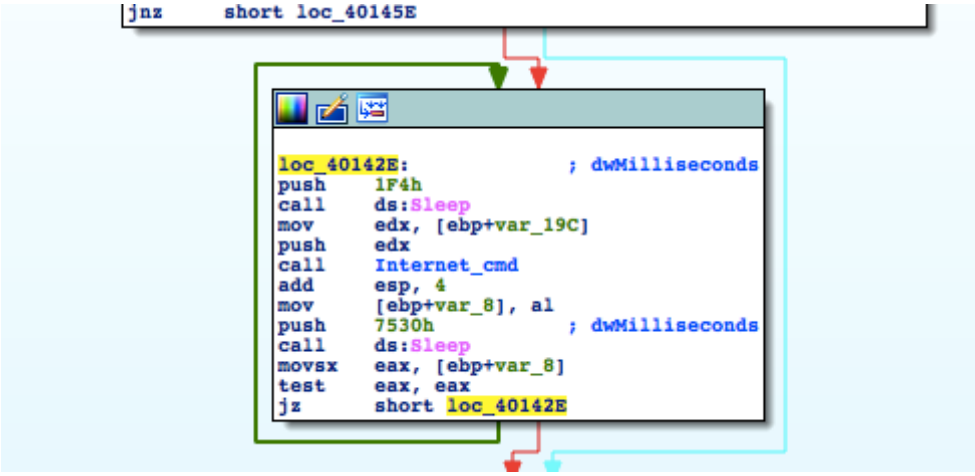
The IDA Pro I use doesn't label the base64 index and padding indicator. So we can find it manually. The base64table is located at function sub_401000. I rename it as

base64Cipher. Double click on it and we can notice that base64 appears many times and the padding indicator is shown as hex digit. However I have a few questions concerning this code area. (See issues or problems section)



The function will return the ciphertext of hostname to 0040122F. Then the ciphertext is passed to sprintf as a parameter. The outcome of sprintf should be an URL because the string format is http://%s/%s/. The program opens the URL and read the data from the URL. The return data is stored at a pointer parameter ebp+var_8. If the data is not zero, then the program jumps to loc_4012EA. At loc_4012EA, the first character of data is compare with 0x6F which is letter "o". If the first letter is "o", the program will return number 1 and exit otherwise it will return number 0 without terminating. Back to main function where Internet_cmd( ) is called. We found out that there is a loop.

If the first character of data is not "o", the program will keep running because there isn't any alternative condition for break. The program will take a sleep and open the URL and read data again until the first character is "o". That is the reason why when I launch the malware, the command prompt didn't exit until I close it.



Issues or Problems

The first issue I met is that all the plunge-in doesn't work at my IDA Pro. After I download those plugins and put them into IDA\Plugin folder, those plugins cannot show up at edit/plugin when IDA Pro starts. The second issue I met is that I listen to port 80 and get the beacon eG1hb31p instead of the string listed at book. Since the string is based on hostname, so I will say it is not a big issue.

Conclusion

Compared with the next two labs, this lab is not that difficult. This malware is an identifier showing that it is running and reading data from the specific hostname. If the data starts with letter "o", the malware will return number 0 and stop. If the data doesn't start with "o", the malware will keep running within the loop until it meets the condition. The host URL is encrypted by XOR with key 0x3B. The first 12bytes of hostname is encrypted by base64 as beacon.

Reviewed Questions

1. Compare the strings in the malware (from the output of the strings command) with the information available via dynamic analysis. Based on this comparison, which elements might be encoded?

host RUL is encrypted because I didn't see www.practicalmalwareanalysis.com at strings. GET string *beacon* is encrypted.

2. Use IDA Pro to look for potential encoding by searching for the string xor . What type of encoding do you find?

XOR cipher appears in three different functions. The useful one is located at 004011B8. Except this one, XOR operations are used to clean the registers.

3. What is the key used for encoding and what content does it encode?

ciphertext: LLL·KIZXORXZWVZW LZI^ZUZWBHRH·XT

key: 0x3B

plaintext: www.practicalmalwareanalysis.com

4. Use the static tools FindCrypt2, Krypto ANALyzer (KANAL), and the IDA Entropy Plugin to identify any other encoding mechanisms. What do you find?

The plugin doesn't work at the version of IDA Pro that I use. But from analyzing IDA Pro assembly code, I know that the cipher is XOR and base64.

5. What type of encoding is used for a portion of the network traffic sent by the malware?

Base64 is used to generate the beacon string used by GET.

6. Where is the Base64 function in the disassembly?

0x004010B1

7. What is the maximum length of the Base64-encoded data that is sent?
What is encoded?

The program copies the first 12bytes of hostname. Therefore the maximum length should be 12. Base64 encode the data from 3bytes into 4bytes. Therefore the encoded data should have 16bytes as maximum length.

8. In this malware, would you ever see the padding characters (= or ==) in the Base64-encoded data?

I saw the padding character at loc_401077. The character is shown as hex digit 0x3D.

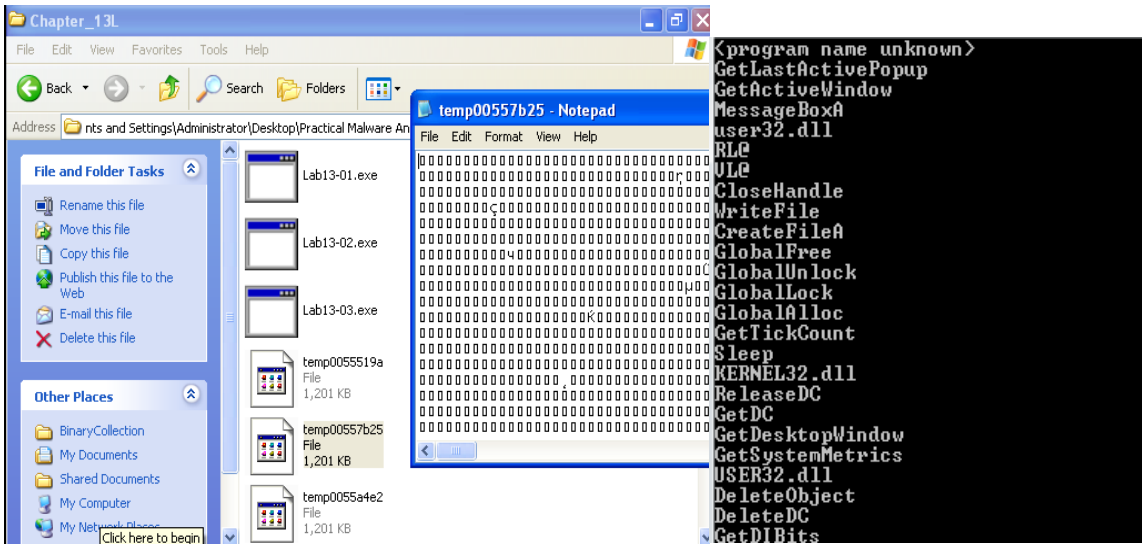
9. What does this malware do?

This malware encrypts the hostname and reads data from the host URL. It will not stop until it receives specific data string starting with letter "o".

Lab13-02

Steps of Processes

First I check the string to guess the purpose of the malware. The string shows some interesting entries for example WriteFile, GetDesktopWindows, Temp%08x, GDI32.dll, and etc. The malware has intercepted with GDI32.dll which contains functions for the Windows GDI (Graphics Device Interface). This module of the Windows GDI assists windows in creating simple 2–dimensional objects. The malware might capture the screenshot and retrieve information from the screenshot and save the encrypted information to temp file. So we can run the malware and see the behavior. After the malware is launched, it creates many temp files at current directory. The data stored at temp files are not readable.



There isn't any network-based indicator since I have already set apateDNS and found nothing. Now we can analyze the malware at IDA Pro. The main function has only one function call sub_401851. We double click on it and see there a lot of function calls inside of it. So we will analyze the functions one by one.

sub_401070 has many functions but it is not difficult to determine. GetSystemMetrics is used to retrieves the specified system metric or system configuration setting. GetDesktopWindow is used to get the windows screenshot. GetDC function retrieves a handle to a device context (DC) for the client area of a specified window or for the entire screen. The BitBlt function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context. The GetObject function retrieves information for the specified graphics object. So far we can almost assure that sub_401070 is used to get the screenshot and transfer the pixels of picture to bits. Therefore we can rename the function as GetPicBit.

sub_40181F also contains many function calls. The most interesting operation is "XOR". The function takes as pixel bytes and output pointer as parameters. Therefore we can pretty sure that this function is the cipher, and the cipher schema is XOR. Because of the complexity of encryption, we will go back to this function later. Rename it as XORCipher. After the cipher schema, the program calls GetTickCount that gives a few seconds snap between the malware creates new file at temp path. Morevoer, the return value of GetTickCount will be the name of the created file. That's why we see the different random file names at temp path.

sub_401000 writes the encrypted data into files. It takes the encrypted bytes, file name, file path as parameters. It calls CreateFile and WriteFile. This function is easily to analyze. Rename it as WriteCiphertext. Then the program calls GlobalUnlock/GlobalFree and it goes back to sleep before it terminates.

So far the major problem is the encryption schema. Since all the plugins cannot work here, I can assume that XOR is the only cipher. We search XOR and see the results. The result makes matter only if it combines with number or register storing value. If XOR operation is followed by register+register, then we can ignore it. Therefore the only two left results are sub_401739 and sub_40128D. The first one is in sub_40181F which is renamed as XORCipher. If we click on the second one and trace back according to the cross-reference. We can also find out that sub_40128D is within XORCipher. Therefore we should analyze XORCipher.

Address	Function	Instruction	
.text:00401040	WriteCiphertext	xor	eax, eax
.text:004012D6	sub_40128D	xor	eax, [ebp+var_10]
.text:0040171F		xor	eax, [esi+edx*4]
.text:0040176F	sub_401739	xor	edx, [ecx]
.text:0040177A	sub_401739	xor	edx, ecx
.text:00401785	sub_401739	xor	edx, ecx
.text:00401795	sub_401739	xor	eax, [edx+8]
.text:004017A1	sub_401739	xor	eax, edx
.text:004017AC	sub_401739	xor	eax, edx
.text:004017BD	sub_401739	xor	ecx, [eax+10h]
.text:004017C9	sub_401739	xor	ecx, eax
.text:004017D4	sub_401739	xor	ecx, eax
.text:004017E5	sub_401739	xor	edx, [ecx+18h]
.text:004017F1	sub_401739	xor	edx, ecx
.text:004017FC	sub_401739	xor	edx, ecx
.text:0040191E	_main	xor	eax, eax
.text:0040311A		xor	dh, [eax]
.text:0040311E		xor	[eax], dh
.text:00403688		xor	ecx, ecx
.text:004036A5		xor	edx, edx

XORCipher(sub_40181F) -> sub_4017390(XOR) ->sub_4012DD ->jump to loc_40142D -> sub_40128D(XOR). According to this chain, we can clearly see that the second XOR is embedded into the first XOR cipher. At code area of sub_4017390 and sub_40128D, we can notice many bit shift (left or right) operations because of the cipher. The algorithm is too complicated to convert to program. Therefore we should use Immdbg or Ollydbg to decrypt it. The logic is not difficult to understand due to the features of XOR. We set a breakpoint right before the encryption. Then we change the content of plaintext. Here the plaintext is the pixels of screenshot. We need to change it to the previous ciphertext. Here the ciphertext is the encrypted pixels of screenshot which is not readable data. The malware encrypt the ciphertext, which is equivalent to decrypt the ciphertext because it is XOR cipher. If we use it twice you will get the original information back.

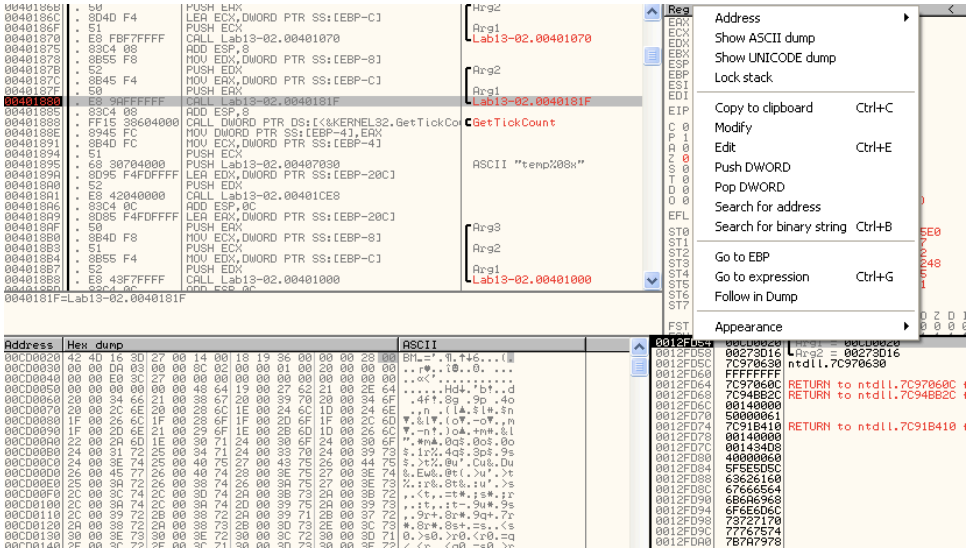
For example: 1010(plaintext) ^ 1110 (key)=100(Ciphertext)

100 ^ 1110 = 1010 (the original information)

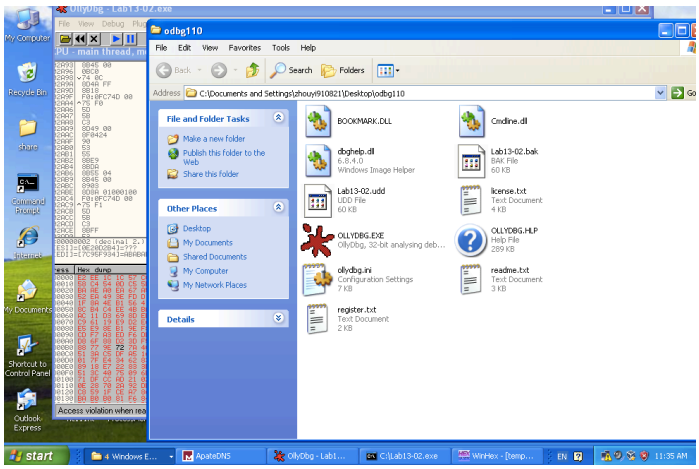
We didn't detour the malware program. The important thing is to change the content at buffer from plaintext to previous ciphertext. Then the decrypted image should be saved at the new file with different name.

To see the plaintext, the screenshot picture, we can figure out the cipher algorithm and run it reversely. Or we can run OllyDbg to retrieve the plaintext. The first breakpoint I pick is 0x401880(XORCipher). This is the only option for the first breakpoint because we need to change the value of plaintext which is going to be passed into XORCipher( ). XORCipher takes two parameters, length and plaintext; the second breakpoint should be

set after WriteFile function is called. Therefore I set it at 0x4018CE. Now we can run OllyDbg and see the malware's behavior. The program stops at the first breakpoint. IDA Pro helps us label the content at register. arg2 is number of bytes to write; arg1 is plaintext. Therefore we choose the value at arg1 and show the hex digits. See the lower panel at left. We need to modify the ASCII at that area (plaintext to ciphertext). We paste ciphertext from tempXXXX file and resume the program.

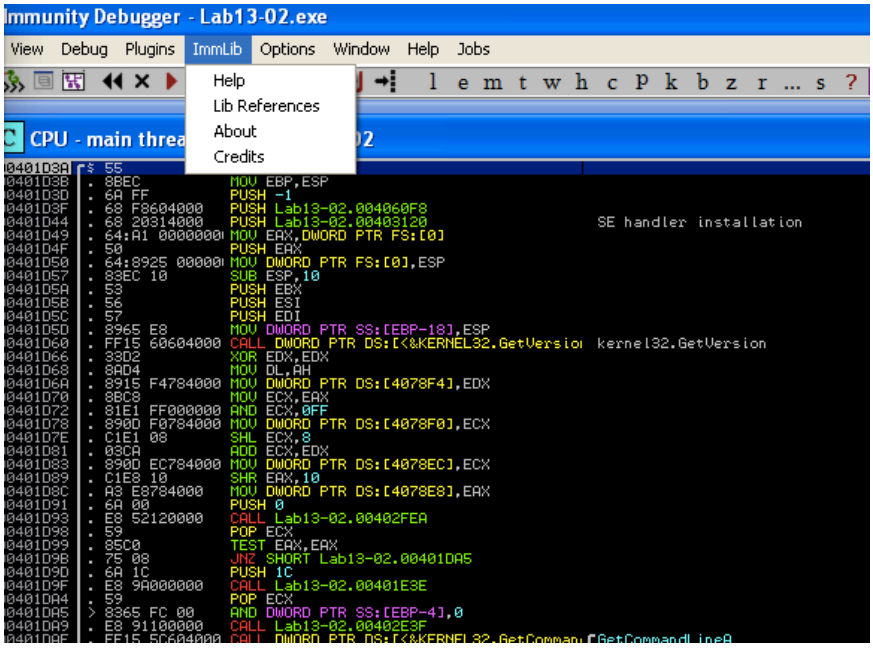


The data is red after binary paste. Then we can run the program. After the second breakpoint is hit, the malware create a new file at local path. Before I start OllyDbg, I deleted all the ciphertext files except one for future use. Therefore right now there are only two files at current path. One is the ciphertext and the other one should be plaintext with name tempXXXXXX. To see the screenshot, rename the new created file with extension png or bmp. Open it.



Issues or Problems

First I try to use ImmDbg python plugin to decrypt it. However I cannot find where the python script is ran. The python script has already been placed into PyCommand folder. But I can't see the run script command at ImmDbg library.



Conclusion

This malware is not that harmful. It capture the current windows and encrypt the pixels then save the ciphertext into a file formatting by "temp"+"ticketcount". The cipher schema is XOR. To decrypt it, we should use same cipher schema to encrypt it twice. The malware will stop if we close it manually.

Reviewed Questions

1. Using dynamic analysis, determine what this malware creates.

A lot of files containing ciphertexts of the screenshots at current path.

2. Use static techniques such as a "XOR" search, FindCrypt2, KANAL, and the

IDA Entropy Plugin to look for potential encoding. What do you find?

The only cipher schema used in this malware is XOR. I found many entries of XOR but only two of them matter.

3. Based on your answer to question 1, which imported function would be a good prospect for finding the encoding functions?

We should pay attention to the code area between GetDesktopWindow/ BitBlt and WriteFile.

4. Where is the encoding function in the disassembly?

The encoding function is sub_40181F.

5. Trace from the encoding function to the source of the encoded content. What is the content?

The plaintext is the screenshot of the current window.

6. Can you find the algorithm used for encoding? If not, how can you decode the content?

The algorithm is complicated. Therefore a good way to decrypt is to XOR twice, which means we use OllyDbg to change the content at buffer from plaintext to ciphertext.

7. Using instrumentation: can you recover the original source of one of the encoded files?

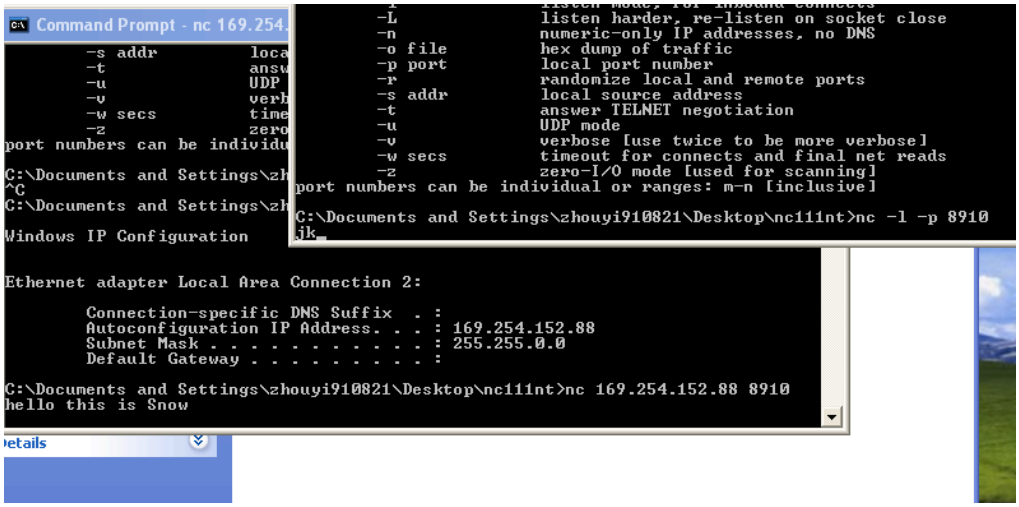
Yes. XOR the content twice and check the new created file.

Lab13-03

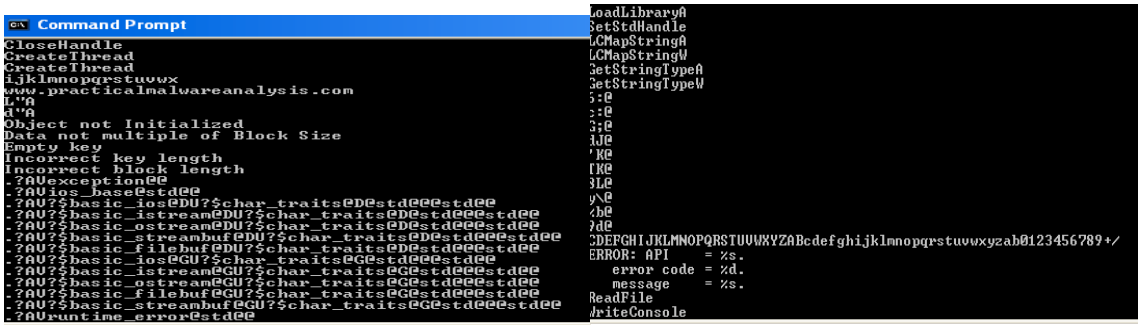
Steps of Process

First we use basic static and dynamic analysis to examine the malware. We noticed that a few functions intercept with Internet function. For example Gethistbyname, WSASocketA, and connect. Therefore we should set Wireshark and Netcat and

ApateDNS to see the malware's behavior related to network function. From the Wireshark, the malware is using port 8910. So I listen on the port 8910 and wait for the result. But nothing shows up. Therefore I started to send some message via socket and I found that the message I send has been encrypted. Moreover the malware intend to get access to www.practicalmalwareanalysis.com. We can figure it out by ApateDNS. The domain name is not encrypted because we can see the name at string. String can give us a lot of entries in this lab.



This is one entry "ijklmnopqrsuvwx" seems like random generated string or cipher key. So far we don't know the cipher schema. We can guess the string is cipher key. And we also noticed "empty key", "incorrect key length", "incorrect block length". I think the malware divide the plaintext into block and encrypt the information by block. This block cipher schema is normally used by DES and AES. I also found an interesting string similar to base64. But in this case I don't think it is standard base64. It is customized base64 cipher schema. Right now we can assume the malware has used two different ciphers. Under the string, we saw some information related to network. When we close the socket at Netcat, we can see those strings. Error API, error code, message. Right now the first goal is to determine the cipher.



We can open the malware at IDA Pro. The main function is not complicated. We can notice the string ijklmnopqrstuvwx immediately after we open IDA Pro. The string is a parameter used by sub_401AC2. So I don't think it is just a random string. It could be the cipher key. Now we can rename sub_401AC2 as Mal_Cipher. We can deterimine the cipher schema later. The return value of Mal_Cipher should be a pointer to WSAstruct so that WSAStartup can initial Winsock properly. Later the program calls WSASocket to create a socket. Type value is one so that the program starts a socket stream that provides sequenced, reliable, two-way, connection-based byte streams with an OOB data transmission mechanism. Then the program keeps calling Internet-related functions like gethostname, htons, connect and sub_4015B7.

Double click on sub_4015B7, we can notice that the program calls CreatePipe for the incoming and outgoing message communication. Then the program opens process "cmd.exe" in order to create a thread in this process later. The threads at cmd.exe are created at 0x401807 and 0x40184B, respectively. One of them is outgoing communication thread; the other one is incoming communication thread. efore the program calls CreateThread at 0x401807, the program first calls 0x401082 by passing the pointer StartAddress. At 0x401082, the block cipher encrypts the plaintext (discuss later). At another CreateThread function call 0x4084B, the program calls sub_40132B where base64 happens. Therefore we can assure that the program encrypt the incoming message and outgoing message with block cipher and base64, respectively. But I cannot figure out which one is corresponding to incoming message; which one is corresponding to outgoing message. There is supposed to have an explicit command or process handle indicating the output or input. But I didn't find it.

Search the key word XOR to determine the cipher schema. We can see many entries. If XOR is followed by register + register, then it doesn't make matter. Except the

register cleaning operation, there are five main entries. The first one is located at Mal_Cipher (sub_401AC2); the second one is located at sub_40223A; the third one is located at sub_4027ED. The fourth one is located at sub_402DA8; the fifth one is located at sub_403166.

Now we double click on Mal_Cipher that takes the key as parameter. At the code area of Mal_Cipher, we can see many hints like "Empty key" and "Incorrect block length". The hints indicate that the malware use block cipher. It could possibly be AES\DES. The first comparison is between cipher key and NULL. The cipher key is stored at arg_0. The parameter "ijklmnopqrstuvwx" is passed from main function. If the key is not empty then the program will compare the length of key and block. If all of them are correct, the length of key and block should both be 16 (0x10) in decimal, then the program will keep going and implementing cipher. I cannot determine which block cipher the malware uses. I think it is AES because the block length is 16*8 bits that is 128bits. DES will not use this length. Therefore the main purpose of Mal_Cipher is to set and initialize block cipher in order to implement encryption.

Next we navigate to sub_40223A where the second XOR appears.

```
sar     edx, 10h
and     edx, 0FFh
mov     eax, ds:dword_40CB08[ecx*4]
xor     eax, ds:dword_40CF08[edx*4]
mov     ecx, [ebp+var_18]
sar     ecx, 8
and     ecx, 0FFh
xor     eax, ds:dword_40D308[ecx*4]
mov     edx, [ebp+var_8]
and     edx, 0FFh
xor     eax, ds:dword_40D708[edx*4]
mov     ecx, [ebp+var_2C]
xor     eax, [ecx+4]
mov     [ebp+var_20], eax
mov     edx, [ebp+var_10]
sar     edx, 18h
and     edx, 0FFh
mov     eax, [ebp+var_18]
sar     eax, 10h
and     eax, 0FFh
mov     ecx, ds:dword_40CB08[edx*4]
xor     ecx, ds:dword_40CF08[eax*4]
mov     edx, [ebp+var_8]
sar     edx, 8
```

We can see a lot of operation involving dword_40CB08 and dword_40CF08. Both of them are from same data block if we double click on them. Therefore it might be the encryption or decryption process of block cipher. The same theory could also be applied to 0x402DA8. However if we double click on 0x4027ED, we can see that there are some operations related to block cipher. But the data are from different data set compared to dword_40CB08 and dword_40CF08. Unfortunately I failed to figure out how AES works at this malware. But I assume that the purpose of 0x4027ED is opposite to 0x402DA8/0x40223A. They are encryption and decryption function, respectively.

IDA Pro didn't give us block cipher hint at 0x401383 but I think there should be

an encryption process between ReadFile and WriteFile. If we navigate to function call sub_40352D between ReadFile and WriteFile, the function take byte_412EF8 as parameter. Earlier the Mal_Cipher function call also takes it as parameter. Therefore byte_412EF8 might be an indicator to block cipher. Double click on sub_40352D, we csn see more clear hints. The offset "off_412244" has labeled as "Data not multiple of Block Size ", which already indicated that the function is related to block cipher.

Besides block cipher, there is another cipher. The key is located at 0x40105E and stored at pointer register byte_4120A4[ecx]. To determine how it works, we navigate to loc_4010BD because there is a loop. The encryption procedure of base64 has to involve a loop because it divides data into a certain number of blocks. In this case, the loop of base64 is located among 0x4010BD and 0x401119. Between them, the program sets edx as counter. Each time the counter is increased by one and later compared with number 4, which suggest that the plaintext will be divided into 4-bytes chunk. Another question is why the program uses base64. We check the cross reference of sub_401082 where the base64 happens. This function is called at 0x40154E right before WriteFile. The purpose of base64 is similar to the previous block cipher we talked about while they are in different locations.

Issues or Problems

I cannot verify my conclusion except the base64 cipher. I think the malware uses AES but I am still not sure. Moreover the decryption procedure won't work without the python plugin.

Conclusion

This malware is the most difficult one at the lab13 because it use two different cipher AES and customized base64. The malware starts a command shell. All the incoming messages are encrypted by Base64; All the outgoing messages are encrypted by AES.

Reviewed Questions

1. Compare the output of strings with the information available via dynamic analysis. Based on this comparison, which elements might be encoded?

The incoming and outgoing messages are encrypted.

2. Use static analysis to look for potential encoding by searching for the string XOR. What type of encoding do you find?

block cipher could be DES/AES. I think it is AES.

3. Use static tools like FindCrypt2, KANAL, and the IDA Entropy Plugin to identify any other encoding mechanisms. How do these findings compare with the XOR findings?

AES and base64. AES is combined with XOR.

4. Which two encoding techniques are used in this malware?

Customized base64 and block cipher:

5. For each encoding technique, what is the key?

Customized base64:
DEFGHIJKLMNOPQRSTUVWXYZABcdefghijklmnopqrstuvwxyzab0123456789+/
block cipher: ijklmnopqrstuvwx

6. For the cryptographic encryption algorithm, is the key sufficient? What else must be known?

For 64base the information is enough. But for AES, we need to know the initial vector. And how does IV combine with key.

7. What does this malware do?

This malware build a remote shell. It encrypts the incoming message by base64; encrypts outgoing message by AES.

8. Create code to decrypt some of the content produced during dynamic analysis. What is this content?