Xiaoyi Zhou

Purdue University CIT581

Malware Forensics

Lab 14: Malware Network Signature

Due November 25, 2014

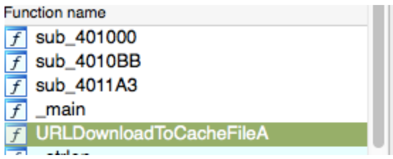
Instructor: Samuel Liles

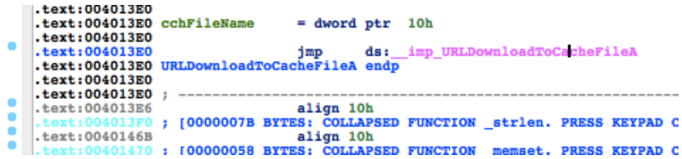
Abstract

This lab focuses on network signature. The malwares didn't use complicated to achieve the malicious goal. The key is to recognize the network beacon and find the important information from it.

Lab14-01

Steps of Processes



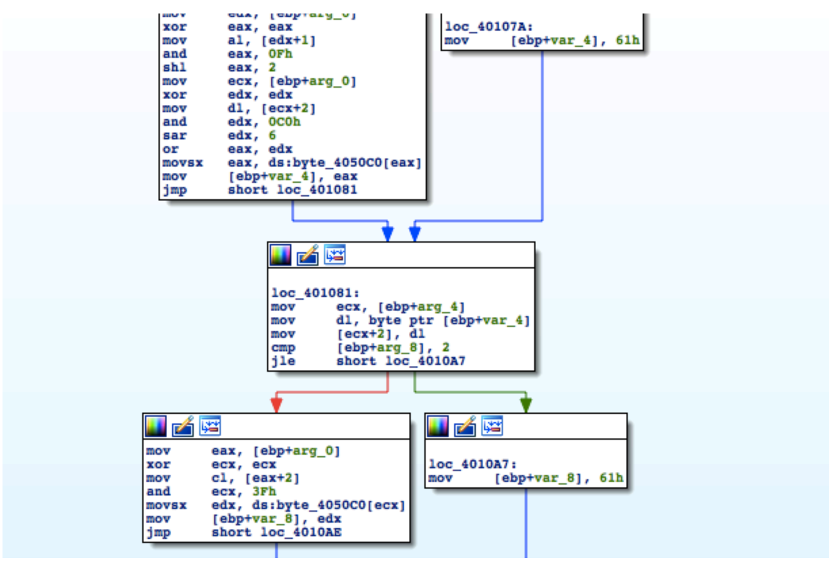


Put this malware into IDA Pro. The first thing I noticed is URLDownloadToCacheFile which is clearly an network-based indicator. Except this function, I didn't see any function related to network. Therefore we should pay attention to this function. URLDownloadToCacheFile downloads data to the Internet cache and returns the file name. Check the cross-reference and navigate to the location where the function is called. The main function calls sub_4011A3; and sub_4011A3 calls URLDownloadToCacheFile. So we can analyze sub_4011A3 first and rename it as Mal_Download. Double click on this function. The function first called sprintf to format the URL string and passed the URL to URLDownloadToCacheFile. The URL is http://www.practicalmalwareanalysis.com/%s/%c.png. The application name is LPSTR which is the destination file. Right now I believe both the file name and string/character are passed from main function. The program will download file from the specific URL to the file. If the operation success, the return value stored in var_41C is 0. Then the program will jumps to loc_401221. At loc_401221, the program calls CreateProcess which takes the application name (file name) as parameter. So the purpose of loc_401221 is to simply launch the target file.

Now we should go back to main function to determine where the string and file name come from. The first function call at this program is GetCurrentHwProfile. This function is used to retrieves information about the current hardware profile for the local computer. The return value is the retrieved information.The information is separated by semi-column at 0x40131F. The formatted string is stored at buffer. Next, the program calls GetUserNameA and then stored the username at local register. We can see another sprintf function call after few lines. The function takes username and GUID information as parameters. Therefore the result should be "GUID information-username". In other words, the result is cc: cc: cc: cc: cc: cc - username.

```
                        ; CODE XREF: _main+CE↑j
lea     edx, [ebp+Buffer]
push    edx
lea     eax, [ebp+var_10098]
push    eax
push    offset aSS      ; "%s-%s"
lea     ecx, [ebp+var_10160]
push    ecx             ; char *
call    _sprintf
add     esp, 10h
push    7FFFh           ; size_t
push    0               ; int
lea     edx, [ebp+var_10000]
push    edx             ; void *
call    _memset
add     esp, 0Ch
lea     eax, [ebp+var_10000]
push    eax             ; int
lea     ecx, [ebp+var_10160]
push    ecx             ; char *
call    StringFormat
add     esp, 8
```

The string is passed to next function call 0x4010BB. The parameter used at Mal_Download is the result of the last function call. The last function call is sub_4010BB. We can rename it as StringFormat. Double click on this function, we can see some cryptography indicator. There are two loops at StringFormat. The first loop divides the result string into 3bytes chunks. The second loops format the chunks into 4bytes loop, which is typically base64 cryptography. The base64 standard string should be located between the two loops. The only significant function between them is sub_401000. So we can rename it as Base64. In the Base64 function, for each chunk the program will use the standard base64 string to encode the string. But the strange thing is I didn't see '=' at the end of the standard string. We should put a question mark and solve it later. Now we still focus on Base64 function. After the string is encoded, the program pads the string with letter 'a' rather than '='. See the following picture. The standard string is stored at byte_4050C0. The padding character is 0x61 which is a in ASCII.

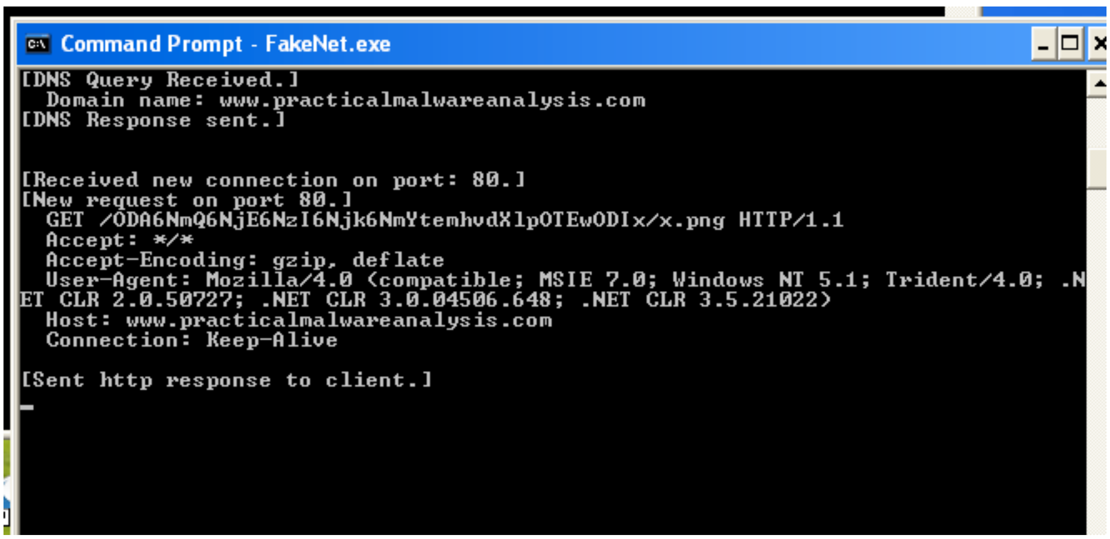


Therefore we can guess that the output string of the program is always followed by letter 'a'. We can open Netcat or FakeNet to verify our guess. The result is a little bit different from what I just analyzed. The encoded string should be ended with letter 'a' because the padding character is lowercase letter 'a', 0x61 in hex digit. However in the FakeNet enviornment, I ran the malware and see the beacon is ended by x/x. I will give the string a question mark here. Now we should decrypt the ciphertext by using base64.

Ciphertext: ODA6NmQ6NjE6NzI6Njk6NmYtemhvdXlpOTEwODIx

Plaintext: 80:6d:61:72:69:6f-zhouyi910821

At this plaintext, zhouyi910821 is my name + birthday, which is always used as username account. I also use it as my account for virtual machine. 80:6d:61:72:69:6f are my GUID.



Now we have already decrypted the beacon. We should try to determine the network signature.

Issue or Problem

I think for this malware the most difficult step is to find the network beacon. Once we find it, the analysis work will be easier. But I am still confused the program padded the beacon string with lowercase 'a'. Why the beacon shows lowercase 'x'?

Conclusion

This malware will download malicious file from the specific URL and then execute the code. It got access to www.practicalmalwareanalysis.com. The downloaded file is always padded with letter 'a'. It also retrieved the GUID and encodes the GUID string by non-standard base64. The ciphertext then is used to initialize the malicious filename.

Reviewed Questions

1. Which networking libraries does the malware use, and what are their advantages?

Since the malware called URLDownloadToCacheFile, I think the malware uses COM network library.

2. What source elements are used to construct the networking beacon, and what conditions would cause the beacon to change?

The beacon is constructed by GUID and username. As long as I am still in the system, the beacon won't change. But it will change if someone else login and I log out first.

3. Why might the information embedded in the networking beacon be of interest to the attacker?

The GUID and username. The attacker wants to know who is logging on this machine.

4. Does the malware use standard Base64 encoding? If not, how is the encoding unusual?

I don't think it is standard because the padding character is letter x instead of '='.

5. What is the overall purpose of this malware?

Spy on user. Download other malicious code and then run it.

6. What elements of the malware's communication may be effectively detected using a network signature?

Check the string format of GUID and username. 80:6d:61:72:69:6f-zhouyi910821. The semi colon and dash characters are the good signatures. Also, the last character of the downloaded file name will always be followed by lowercase letter 'x'.

7. What mistakes might analysts make in trying to develop a signature for this malware?

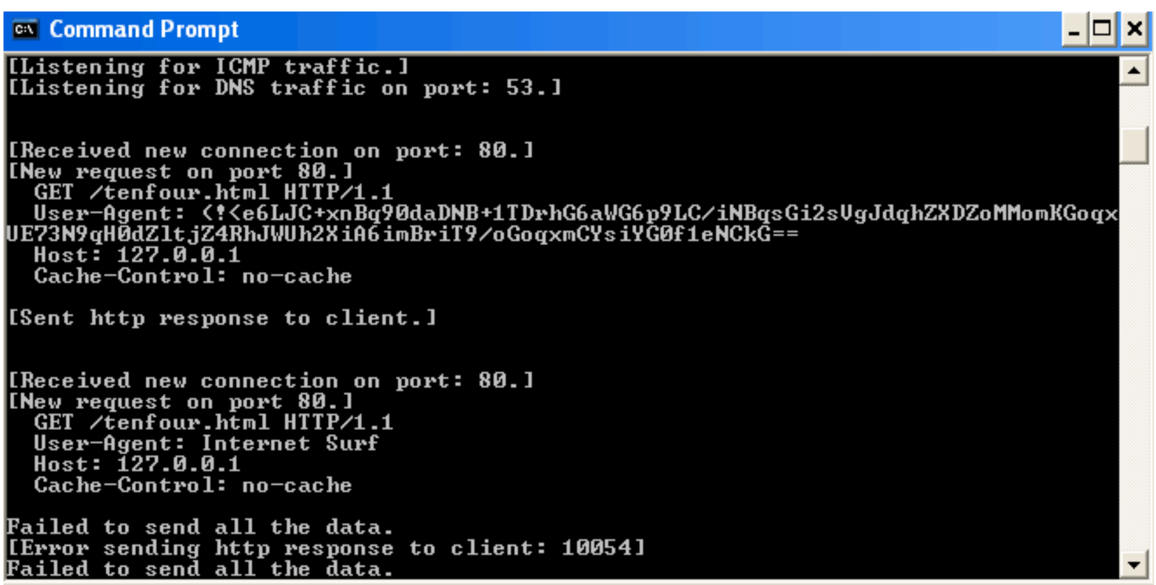
I think the most confused one is the file name. From analyzing the malware IDA Pro, clearly the file name is ended with letter 'a', however the beacon showed the file name is ended with 'x'. Moreover the URI could be various due to the different host and OS.

8. What set of signatures would detect this malware (and future variants)?

I think Snort rule is more effective for detecting the malware behavior. Base64 and the filename are both good signature. So we can generate the network signature as following: alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:" Base64 and png"; urilen:>32; uricontent:".png"; pcre:"/ \ /[A-Z0-9a-z+\ /]{24,}([A-Z0-9a-z+\ /])\ / \1\.png/"; sid:20000001; rev:1;)

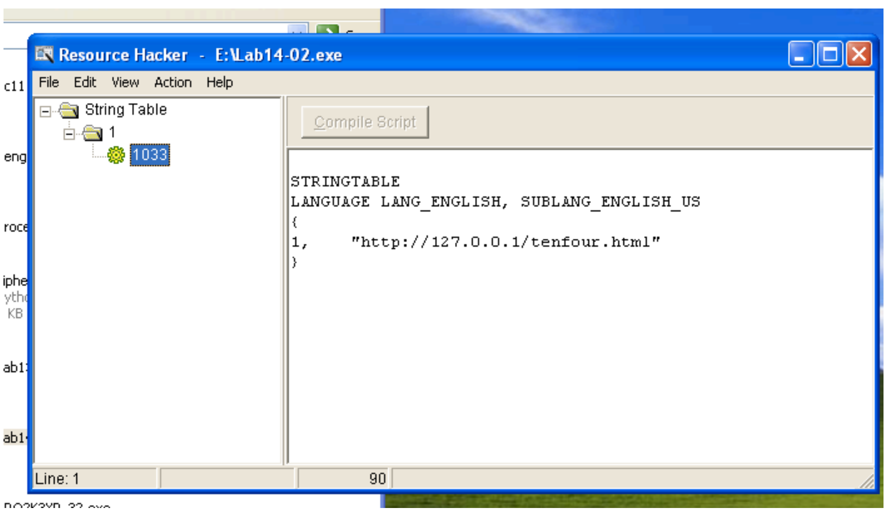
Lab14-02

Steps of Processes



I run the malware before I use IDA Pro analysis. And I found that the malware delete itself after I launch it. If we want to perform statistic or dynamic analysis, we should be careful. We can guess some details of the malware from the information given by FakeNet. First I notice two network beacon which means the malware could possibility create two threads. The first beacon use 64base encryption because I see the padding character '='. Therefore we should pay attention to whether the cipher is standard or customized. Cache-Control: no-cache seems like a command from network function call. We can look up it by Google. This command indicates cached information should not be used and instead requests should be forwarded to the origin server. The program got access to tenfour.html. We should also pay attention to this URL.

Put the malware into IDA Pro and then analysis the details. At the WinMain function, the program calls LoadString. This function implies that the program contains a resource file because the function loads a string resource from the executable file and copy the string to buffer. We use resource hacker to see the hidden string. The string is an URL used for InternetOpenURL later.



Then, the program calls GetCurrentProcess and DuplicateHandle. The duplicated object is cmd.exe. Therefore the current process will start a command prompt when it is running. At 0x40137D, the program creates a thread with offset StartAddress as lpStartAddress. Here the start address is a pointer to another pieces of code. Double click on it and check the details. We should notice the detail that the process information is stored at register used for thread later. The thread will receive input or send out via cmd.exe.

```
push    ebp                 ; lpApplicationName
call    ds:CreateProcessA
mov     esi, ds:CloseHandle
test    eax, eax
jz      short loc_401366
mov     ecx, [esp+1A8h+ProcessInformation.hThread]
mov     edi, [esp+1A8h+ProcessInformation.hProcess]
push    ecx                 ; hObject
call    esi ; CloseHandle
jmp     short loc_40136A
---------------------------------------------------------------------------

                            ; CODE XREF: WinMain(x,x,x,x)+191↑j
mov     edi, [esp+1A8h+var_190]

                            ; CODE XREF: WinMain(x,x,x,x)+1A4↑j
lea     edx, [esp+1A8h+ThreadId]
lea     eax, [esp+1A8h+ThreadAttributes]
push    edx                 ; lpThreadId
push    ebp                 ; dwCreationFlags
push    ebx                 ; lpParameter
push    offset StartAddress ; lpStartAddress
mov     [ebx+8], edi
mov     edi, ds:CreateThread
push    ebp                 ; dwStackSize
push    eax                 ; lpThreadAttributes
mov     [esp+1C0h+ThreadAttributes.nLength], 0Ch
mov     [esp+1C0h+ThreadAttributes.lpSecurityDescriptor], ebp
mov     [esp+1C0h+ThreadAttributes.bInheritHandle], ebp
call    edi ; CreateThread
cmp     eax, ebp
mov     [ebx+0Ch], eax
```

At the code area of StartAddress, the program calls PeekNamedPipe that copies data from a named or anonymous pipe into a buffer without removing it from the pipe. If the data is available, the program will read the data and encrypt the data by base64 at sub_401000. We can rename sub_401000 as base64. The base64 string is stored at byte_403010. However it is not a standard base64. The string is XYZlabcd3fghijko12e456789ABCDEFGHIJKL+/MNOPQRSTUVmn0pqrstuvwxyz and padded with '='. The ciphertext is modified again at next function call sub_401750. The ciphertext is concatenated with "(!<". We see the string at initial beacon as user-agent string. The following string is the plaintext. The beacon will be different if the host changes or the location of the malware changes. To decrypt the string, we can use python script or online custom base64 decoder. Remeber to get rid of '(!<' and 'KG' because the first is plaintext user-agent. The second one is not utf-8 character. It also not important.

```
cipher.py    untitled

import string
import base64
customized = 'WXYZlabcd3fghijko12e456789ABCDEFGHIJKL+/MNOPQRSTUVmn0pqrstuvwxyz'
default = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
ciphertext = 'e6LJC+xnBq90daDNB+1TDrhG6aWG6p9LC/iNBqsGi2sVgJdqhZXDZoMMomKGoqxUE73N9qH0dZltjZ4RhJWUh2XiA6imBriT9/oGoqxmCYsiYG0f1eNC=='
plaintext=""
loc=-1
str1 = ""
for c in ciphertext:
    if(c in customized):
        loc=customized.find(str(c))
        if(loc!=-1):
            str1 += default[loc]
    elif (c == '='):
        str1 += '='
plaintext = base64.decodestring(str1)
print plaintext


Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

E:\
[Finished in 0.0s]
```

The user-agent is determined. The program calls InternetOpen and InternetOpenUrl to open and URL. So far we just have little information about the URL. The URL is the one extracted from resource section. http://127.0.0.1/tenfour.html. The purpose of the first thread is reading the data from the pipe and sending the data to a remote shell via a specific user-agent and URL.

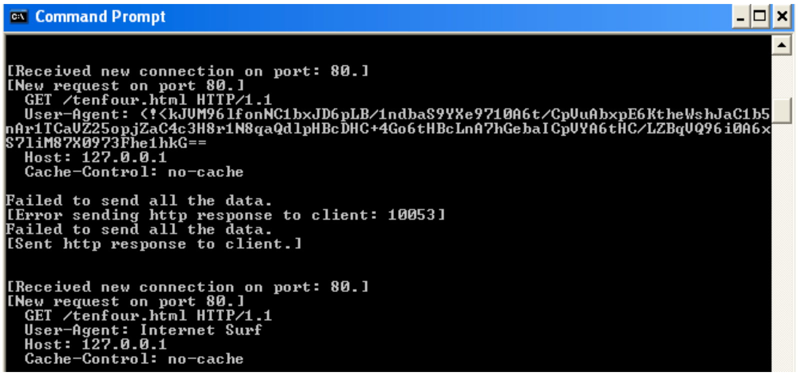
Now we can check the second thread. The lpStartAddress is sub_4015C0. It is a pointer to a piece of malicious code. The program calls InternetOpen at sub_401800. The User-Agent is statistic Internet Surf. The URL is same with the first one. The purpose of the malware is to write data. If the content is exit, then the malware will exit itself. The comparison is initialized by strnicmp. After the second thread is terminated, the program calls sub_401880 where the malware deletes itself by executing the delete command. The command is concatenated by different short string by lstrcat. See IDA Pro labels the strings. /c del [filename] > nul. In this case the filename should be the malware file name.

```
jz      loc_4019D0
lea     ecx, [esp+358h+String1]
push    offset String2  ; "/c del "
push    ecx             ; lpString1
call    ds:lstrcpyA
mov     esi, ds:lstrcatA
lea     edx, [esp+358h+Filename]
lea     eax, [esp+358h+String1]
push    edx             ; lpString2
push    eax             ; lpString1
call    esi ; lstrcatA
lea     ecx, [esp+358h+String1]
push    offset aNul     ; " > nul"
push    ecx             ; lpString1
call    esi ; lstrcatA
mov     [esp+358h+pExecInfo.hwnd], edi
lea     edx, [esp+358h+Buffer]
lea     eax, [esp+358h+String1]
mov     [esp+358h+pExecInfo.lpDirectory], edi
mov     [esp+358h+pExecInfo.nShow], edi
mov     edi, ds:GetCurrentProcess
push    100h            ; dwPriorityClass
mov     [esp+35Ch+pExecInfo.cbSize], 3Ch
mov     [esp+35Ch+pExecInfo.lpVerb], offset aOpen ; "Open"
mov     [esp+35Ch+pExecInfo.lpFile], edx
mov     [esp+35Ch+pExecInfo.lpParameters], eax
mov     [esp+35Ch+pExecInfo.fMask], 40h
call    edi ; GetCurrentProcess
mov     esi, ds:SetPriorityClass
```

Therefore the malware will not delete itself immediately. There will be a few seconds postponed after it launched.

Issue or Problem

I didn't meet any difficult problems here. The only tricky part is to identify the two different beacons and different purposes of the two threads. Actually I saw three different beacons when I launched this malware. I really have no idea how did the following long beacon come out. I try to decrypt the long beacon by base64. But the plaintext is non-readable. I am trying to figure it out.



Conclusion

The malware creates two threads in order to perform reading and writing data, respectively. The encrypted User-Agent used for reading data could be various according to different host and OS. The user-agent used for writing data is statistic. The URL is same for both. In a word, the malware intends to build a command shell and send information to a remote hacker. After it launched, the malware deletes itself.

Reviewed Questions

1. What are the advantages or disadvantages of coding malware to use direct IP addresses?

IP address is more difficult to handle than domain names, which is advantage and disadvantage as well. Because both adversary and defender need to take effort to resolve the statistic IP address.

2. Which networking libraries does this malware use? What are the advantages or disadvantages of using these libraries?

The program calls InternetOpen and InternetOpenURl which are belong to WinINet API library. WinINet() operates at a higher level. It understands the protocol (HTTP, FTP etc) underlying the transfer. However the attacker has to provide explicit user-agent.

3. What is the source of the URL that the malware uses for beaconing? What advantages does this source offer?

The content stored at resource section is similar to macro definition. The hacker can change URL or other command by modifying the resource file instead of modifying the malware.

4. Which aspect of the HTTP protocol does the malware leverage to achieve its objectives?

User-Agent. The first thread would encrypt the host information and concatenated it with string as user-agent.

5. What kind of information is communicated in the malware's initial beacon?

The encrypted host information.

6. What are some disadvantages in the design of this malware's communication channels?

7. Is the malware's encoding scheme standard?

No, it is not standard. The cipher key is WXYZlabcd3fghijko12e456789ABCDEFGHIJKL+/MNOPQRSTUVmn0pqrstuvwxyz.

8. How is communication terminated?

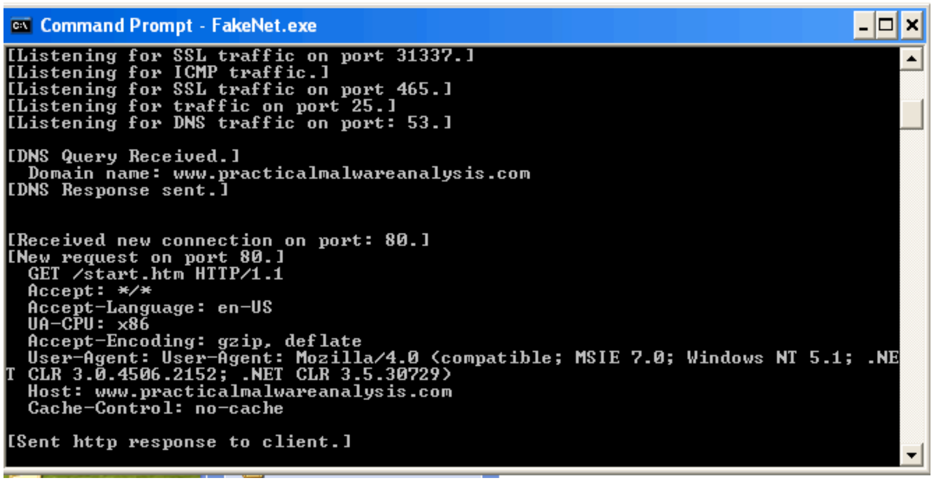
The threads terminated itself after the malware read and write data. And then the malware delete itself.

9. What is the purpose of this malware, and what role might it play in the attacker's arsenal?

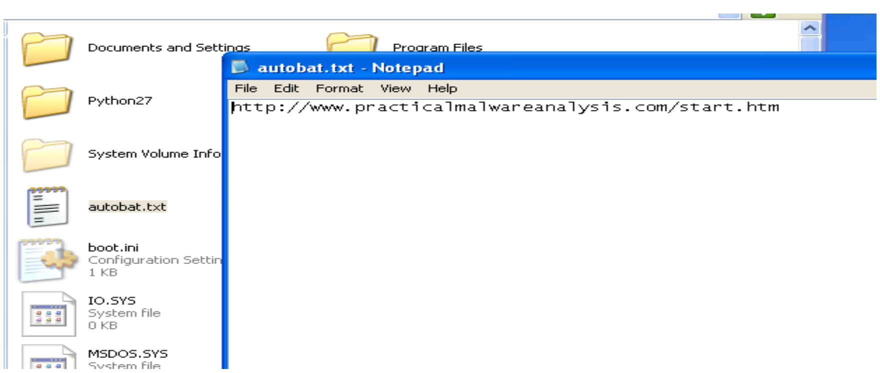
The malware delete itself. So it is not the major technique used by attacker. I think it is a simple backdoor that starts a command shell prompt to the attacker.

Lab14-03

Steps of Processes



We can analyze this malware as the same way we analyzed lab14-02.exe. From the information given by FakeNet, we noticed that the malware try to get access to www.practicalmalwareanalysis.com. So far I haven't seen any clear encryption indicator. The 'accept encoding' is really interesting. We should pay attention to it. Also the malware intended to send http response to client. I guess this malware might be a backdoor and a downloader as well because I put the malware into VirusTotal. One of the virus signatures is a downloader. Put the malware into IDA Pro for further analysis. The first function call is sub_401457. This function creates a file at C:\autobat.exe. The file pointer is stored at register. The file doesn't exist at begging. So the program will take the URL as parameter and then call sub_401372. We can check the C:\autobat.exe right now. I rename it the file as txt format and open it. The content of the file is the URL. Therefore I can assume that sub_401372 is used for writing the URL to the text file for future network function.



Double click on sub_401372. The function takes the http://www.practicalmalwareanalysis.com as parameter. It calls CreateFile and WriteFile.

Clearly the purpose of this function is to write URL to C:\autobat.exe. Now the file has already existed, the program will go back to the sub_401457 again because there is a loop at sub_401457. The loop won't stop until the file exists. If the file exists, the program will jump to loc_4014EA where the program will read the file and return the pointer to the read data. We can rename sub_401457 as ReadURL.

```
768 loc_401768:                             ; CODE XREF: WinMain(
768                 cmp     [ebp+var_208], 0
76F                 jz      short loc_4017A3
771                 push    200h            ; size_t
776                 push    0               ; int
778                 lea     eax, [ebp+szUrl]
77E                 push    eax             ; void *
77F                 call    _memset
784                 add     esp, 0Ch
787                 push    200h            ; int
78C                 lea     ecx, [ebp+szUrl]
792                 push    ecx             ; lpBuffer
793                 call    ReadURL
798                 add     esp, 8
79B                 neg     eax
79D                 sbb     eax, eax
79F                 inc     eax
7A0                 mov     [ebp+var_4], eax
7A3
7A3 loc_4017A3:                             ; CODE XREF: WinMain(
7A3                 cmp     [ebp+var_4], 0
7A7                 jnz     short loc_4017DC
7A9                 lea     edx, [ebp+var_204]
7AF                 push    edx             ; char *
7B0                 lea     eax, [ebp+szUrl]
7B6                 push    eax             ; lpszUrl
7B7                 call    sub_4011F3
7BC                 add     esp, 8
7BF                 test    eax, eax
7C1                 jz      short loc_4017DC
```

The next function call at main function is sub_4011F3. The program formatted some strings for InternetOpen and InternetOpenUrl. At 0x4012C7, the program calls InternetReadFile. The return value will be the data retrieved from the URL. Assuming the data is a string, the program uses strstr to check if the string contains '<no'. If so, then the program will call sub_401000 otherwise the program will call InternetReadFile again. In a word, there is a loop. The loop will not stop until the retrieved string contains '<no'. If the string has '<no', the program cam go to sub_401000. The function takes three parameters: URL, '<no', and general register. IDA Pro has labeled them as char. We double click on it. This function is used to search the certain piece of string. If we right click on the hex digits after comparison operands, we will learn that the certain piece of string is '>niorspct'. I think the order has a problem.

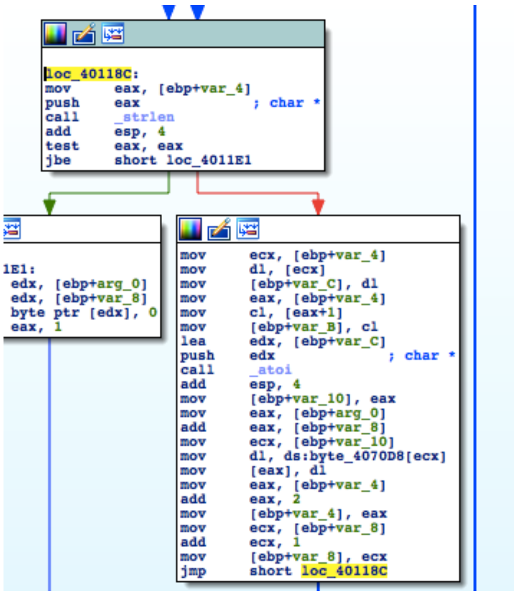
The program calls strcpy and strrchr to search for '/' and '96'. So the return value of the function should be the data between <noscript>.....<96>. Here I don't quite understand why the function takes URL as parameter and search for '/'. Assuming we have already had the return value, the return value is passed to sub_401684. If we don't have the return value, the program will keep executing sub_401000 because the loop can only stop because the program can find the certain piece of string.

At sub_401684, we can notice a jump table which means there is a switch condition. There are five cases including the default one. The return value is stored at eax.

From the future analysis, we can guess that the return value is a character. The program subtracts the character with letter 'd' (0x64). The result will be the switch condition.

```
CE                 mov     [ebp+var_14], eax
D1                 cmp     [ebp+var_14], 0Fh ; switch 16 cases
D5                 ja      short loc_401723 ; jumptable 004016E2 default case
D7                 mov     edx, [ebp+var_14]
DA                 xor     ecx, ecx
DC                 mov     cl, ds:byte_40173E[edx]
E2                 jmp     ds:off_40172A[ecx*4] ; switch jump
E9 ; ---------------------------------------------------------------------------
E9
E9 loc_4016E9:                             ; CODE XREF: sub_401684+5E↑j
E9                                         ; DATA XREF: .text:off_40172A↓o
E9                 mov     eax, [ebp+var_C] ; jumptable 004016E2 case 0
EC                 push    eax             ; char *
ED                 call    sub_401565
F2                 add     esp, 4
F5                 jmp     short loc_401723 ; jumptable 004016E2 default case
F7 ; ---------------------------------------------------------------------------
F7
F7 loc_4016F7:                             ; CODE XREF: sub_401684+5E↑j
F7                                         ; DATA XREF: .text:off_40172A↓o
F7                 mov     [ebp+var_4], 1  ; jumptable 004016E2 case 10
FE                 jmp     short loc_401723 ; jumptable 004016E2 default case
00 ; ---------------------------------------------------------------------------
00
00 loc_401700:                             ; CODE XREF: sub_401684+5E↑j
00                                         ; DATA XREF: .text:off_40172A↓o
00                 mov     ecx, [ebp+var_C] ; jumptable 004016E2 case 15
03                 push    ecx             ; char *
04                 call    sub_401613
09                 add     esp, 4
0C                 jmp     short loc_401723 ; jumptable 004016E2 default case
0E ; ---------------------------------------------------------------------------
0E
0E loc_40170E:                             ; CODE XREF: sub_401684+5E↑j
0E                                         ; DATA XREF: .text:off_40172A↓o
0E                 mov     edx, [ebp+var_C] ; jumptable 004016E2 case 14
11                 push    edx             ; char *
12                 call    sub_401651
```

If the switch condition is 0, the program calls sub_401565. At sub_401565, the program calls URLDownloadToCacheFile, which indicates that the parameter passed by the former function call should be an URL. And the only one URL we have seen at this malware is www.practicalmalwareanalysis.com. So we can see how did the program decrypt the ciphertext into plaintext URL at sub_401147. I think the decryption function is clearer at graphic mode. ecx is the counter. It is set to 0 at beginning. Each time the program finishes a loop, the counter is increased by one. eax is also a counter. But it is for the ciphertext. Therefore we can assume that the ciphertext is a string which is converted to integer by atoi. The converted integer could be divided into many chunks. For each chunk, the two decimal numbers can represent a letter according to the cipher key.The key is /abcdefghijklmnopqrstuvwxyz0123456789:. The cipher is simply substituting the integer by character with same index.



If the switch result is 10, then the program dose nothing but exit to the main function.

If the switch result is 15, the user can input character to control how many seconds the program will sleep. It would be either 20seconds or 10seconds.

If the switch result is 14, the program will start over again, which means it will starting creating autobat.exe and retrieved data from the URL.

Issue or Problem

The data retrieved from the website is a string. The very beginning of the string is supposed to be <noscript>. However I didn't notice any code that indicates to reorganize the order of letters. Besides the retrieved string is between <noscript>http://www.practicalmalwareanalysis.com. I didn't see any concatenation for <noscript> and the URL.

Conclusion

This malware download data from http://www.practicalmalwareanalysis.com. It will execute different operation due to the retrieved content. Overall this malware will not cause too much damage to host machine.

Reviewed Questions

1. What hard-coded elements are used in the initial beacon? What elements, if any, would make a good signature?

The user-agent header could be a good signature. Accept, Accept Language, UA-CPU, Accept encoding are hard-coded elements.

2. What elements of the initial beacon may not be conducive to a long-lasting signature?

The URL might be different according to the C:\autobat.exe. If the content at the file is different, the network beacon will show different URL.

3. How does the malware obtain commands? What example from the chapter used a similar methodology? What are the advantages of this technique?

By retrieved the information from the URL. Then the malware narrowed down the range for the specific command. The advantages is that the URL is totally legitimate so the defender will not be suspicious about it.

4. When the malware receives input, what checks are performed on the input to determine whether it is a valid command? How does the attacker hide the list of commands the malware is searching for?

The malware change the order of <noscript> so that it will not draw attention. To find the specific command, the malware searched for <no, and check the letters in different order.

5. What type of encoding is used for command arguments? How is it different from Base64, and what advantages or disadvantages does it offer?

It has nothing to do with Base64. Totally different. It is just a customized cipher that substitutes number by letter with same index. The advantage and disadvantage is that the cipher is easy to crack.

6. What commands are available to this malware?

download, sleep, exit, and redirect.

7. What is the purpose of this malware?

This malware download data from http://www.practicalmalwareanalysis.com. It will execute different operation due to the retrieved content. Overall this malware will not cause too much damage to host machine.

8. This chapter introduced the idea of targeting different areas of code with independent signatures (where possible) in order to add resiliency to network indicators. What are some distinct areas of code or configuration data that can be targeted by network signatures?

Areas of code that show the hard-coded beacon information; decrypt the domain name of the URL; initial the command.

9. What set of signatures should be used for this malware?

The malware retrieved the information from the URL and search for the certain string. It uses '<no' to find the beginning of the string. It uses '96' as the end of the string. So we can generate the first signature.

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:" Noscript tag and 96"; content:"<noscript>"; content:"96"; distance:0; within:512; sid:20001400; rev:1;)

Then the malware uses customized decryption to get the URL. The encryption function is associated with the redirect command. So we can generate the second signature.

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"Redirect Command"; content:"/http://www "; pcre:"/ \/s[^\ /]{0,15}\ /[0-9]{2,20}/"; sid:20001401; rev:1;)