Xiaoyi Zhou
Purdue University CIT581
Malware Forensics
Lab 10: Kernel Debug
Due October 15, 2014
Instructor: Samuel Liles

Abstract

Lab10 is required for analyzing malware by using WinDbg and other dynamic/statistic malware forensic tools. WinDbg is first used at this lab and I found that it is similar to gdb to debug C program. This lab is also required to read and recognized assembly language and understand the structure of link list. After finishing the lab, I gain more experience and knowledge about assembly and stack structure and VMware functions.

This lab contains three malwares. The first malware is designed for disabling the firewall. The second malware is rootkit that hides file. The third malware will keep popping advertisement page. The most difficult one is the second one because the file structure and the function related to NtQueryDirectoryFile are hard to understand. Until I read the book and documentation I didn't know the process that how the rootkit changes the structure of FILE_BOTH_DIR_INFORMATION. But now I learn a lot of knowledge about WinDbg and functions related to file and link list entry.

The most difficult point at second lab is that the driver should be hidden. Therefore, we cannot analyze the file by IDA Pro unless we recover the file. However, in my case, I can see the file, which means it is not hidden anymore. Therefore, I use IDA Pro and WinDbg to analyze it. It is more convenient to use IDA Pro to see the assembly code. But, WinDbg is really important when we want to see the parameter and the stacks. We will see the point at second lab and the first lab.

Moreover, by doing this lab, I am more patient about WinDbg because sometimes WinDbg will have a lot of problems that the book doesn't tell. The good way to solve the problem is to see other documents and give some tries.

Lab10-01

Steps of the Process

To answers analyze this lab, we cannot only run the malware without set up anything. Once the malware is running, the executable file will flash one second at process monitor and then disappear. Therefore, we assume that the malware contain some commands to hide or delete itself. The solution is to find the command and then set a breakpoint at the certain address location. So we go through the executable file by IDA pro.

The main function has four function calls. We can analyze them one by one. The first one is CreateService( ) that contains a list of parameters. Most of parameters don't have value. The important parameters are BinaryPathName, StartType, ServiceType, ServiceName. Binarypath is the qualified path to the service binary file where the malicious code is located at this case. So Lab10-01. sys must be analyzed later. StartType 0x0003 indicates that the service is started by the service control manager. ServiceType means driver service. So, the driver is loaded here at 0x401046 by CreateService( ). Here the service name is Lab10-01. The malware will create a service under this name.

The next function call is OpenService( ). If service Lab10-01 has already existed, the program will go to OpenService( ) to open Lab10-01. Then the program will call StartService( ) and ControlService( ). We pay attention to ControlService( ) because the second question involves this function. Moreover, at ControlService( ), the function has three parameters. Within the three parameters, dwControl is 0x00001 which means *hservice* should stop. In this case the service is Lab10-01. We should mark the location 0x401080 where WinDbg should set a breakpoint later. Basically, the executable file will load driver, create service and implement the malicious content at the driver. Then it will delete the driver. The next step is going to analyze driver. So far we hadn't noticed anything related to registry.

In IDA pro analysis, lab10-01.sys has three subroutines. sub_10920 navigate the program to 0x10932 where KeTickCount( ) is called. sub_10906 is an important subroutine because it loads offset value of malicious content into memory location pointer. (we will know it loads malicious content after we click offset sub_10486). At sub_10486, RtlCreateRegistryKey( ) is called 4 times; RtlWriteRegistryValue() is called

three times. Two of three times, RtlWriteRegistryValue( ) has parameter "0, \Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFire\ DomainProfile", "0, \Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFire\StandardProfile", respectively. The second set of parameter value could be found at ebx at 0x104B7 and 0x104E1. Therefore, I guess the program tried to get access to registry area related to firewall and then disable the firewall. To see the purpose, we still need to check the registry by Registry Editor. See the following steps specific to question 3.

Launch WinDbg and set breakpoint at ControlServie( ) at 0x401080 before the driver is deleted.

Open process monitor and apply filter with PID=540 and Operation=RegSetValue. We can see that the program writes entry into registry as the following: HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed. The value doesn't make any sense because the purpose is to mess up with registry.

We set the breakpoint at 0x401080. However, we cannot just type "bp 00401080" because it won't work. WinDbg is similar to gdb somehow. To check the offset first, by input "lmf" and the start value here is offset. We rebase the address at IDA pro. Fortunately the default rebase value at IDA pro is 0x400000 so we don't have to worry so much about offset. Then we transfer the code at ControlService( ) to assembly at WinDbg by typing u 401080. We saw the relative address is "image00400000+0x1080". Therefore, we should set the breakpoint like "bp image00400000+0x1080". Now we run the program and get the following result:

```
*** ERROR: Module load completed but symbols could not be loaded for
image00400000+0x1080:
00401080 ff1510404000    call    dword ptr [image00400000+0x4010 (00
00401086 5e              pop     esi
00401087 33c0            xor     eax,eax
00401089 5f              pop     edi
0040108a 83c41c          add     esp,1Ch
0040108d c21000          ret     10h
00401090 55              push    ebp
00401091 8bec            mov     ebp,esp
0:000> bp image00400000+0x1080
0:000> bl
 0 e 00401080     0001 (0001)  0:**** image00400000+0x1080
0:000> g
ModLoad: 71a50000 71a8f000   C:\WINDOWS\system32\mswsock.dll
ModLoad: 662b0000 66308000   C:\WINDOWS\system32\hnetcfg.dll
ModLoad: 77f10000 77f59000   C:\WINDOWS\system32\GDI32.dll
ModLoad: 7e410000 7e4a1000   C:\WINDOWS\system32\USER32.dll
ModLoad: 71a90000 71a98000   C:\WINDOWS\System32\wshtcpip.dll
Breakpoint 0 hit
eax=0012ff1c ebx=7ffdc000 ecx=77defb8d edx=00000000 esi=00147320 edi
eip=00401080 esp=0012ff08 ebp=0012ffc0 iopl=0         nv up ei pl nz
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl
image00400000+0x1080:
```

We know that the ControlService unload the driver but the driver contains

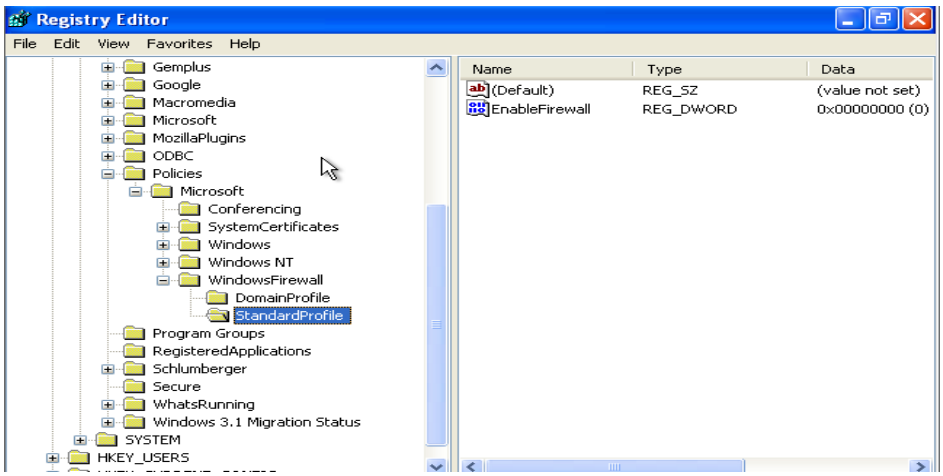
malicious code. Therefore we need to pick another breakpoint right before the driver is gone. So we open another WinDbg and check the driver. The process to set up WinDbg had a few issue. For example, the serials portal didn't match the socket. The solution is to change COM1 to COM2. And the WinDbg always shows reconnecting until we break it or choose resynchronize. Anyway, I type "! drvobj lab10-01" and then my virtual machine got stuck. A good way to get rid of it is type ".crash". We check the driver and get the driver address 8263b418.

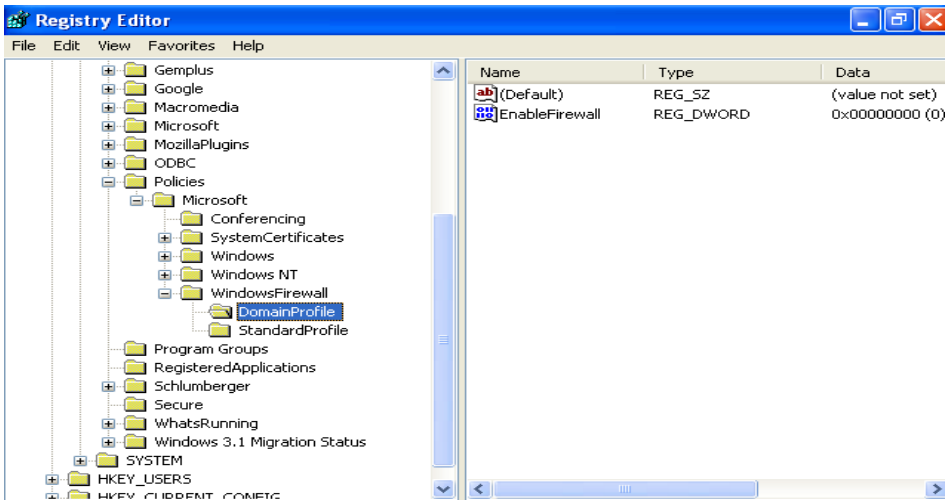
Type command dt _DRIVER_OBJECT 8263b418, we will see a lot of information about the driver. The most important one is where the driver is unloaded. We set the breakpoint at address 0xf7c47486 where it is exactly value we need. When the two breakpoints are set, we keep running the malware meanwhile assuring that the driver will not disappear before it terminates. RtlCreateRegistryKey( ) is called 4 times; RtlWriteRegistryValue() is called three times. \Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFire\ DomainProfile" is set to 0. \Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFire\StandardProfile is also set to 0. (see question 3)

Therefore, after the function call ControlService, the firewall attribute has been modified and the driver is unloaded.

The following step of process is specific to question 3.

Based on above analysis, the malware interacts with two registry paths: "\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFire" and "\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFire \DomainProfile". So we directly go these paths to see the modification by Registry Editor. The value of EnableFirewall at both paths has been set to 0. Therefore, we can make sure that the malware disable the firewall.





The registry path starts with "\Registry\Machine\". But I assume it is the path under the most common registry path HKEY_LOCAL_MACHINE. Then I try to look for the path under HKEY_LOCAL_MACHINE, I found it. We can see the pictures above.

Issue or Problems

The most annoying problem is that when I started the kernel debug, my virtual machine stopped. Therefore, I have to finish the dynamic analyze before starting the kernel debug. And the command "!drvobj lab10-01" didn't work at the first time I applied it. I have to declare the symbol at WinDbg at VMware and then it works. The other problem is that somehow the program can only run once. If the program run twice, the service would step and if we check the service by "sc query" command, I can see it stops so that I can't use debug tools. So I set a mirror point at my VM, I go back to the mirror point every time I ran the malware, which is really time consuming and annoying.

Conclusion

This malware change the firewall attribute at registry from kernel and also write random noise to registry.

Case Study

N/A

Review Questions

1. Does this program make any direct changes to the registry?

Yes. The program writes random nonsense value to registry at path HKLM\SOFTWARE\ Microsoft\Cryptography\RNG\Seed. It also change the value of firewall at registry at the paths \Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFire\ DomainProfile，\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFire\StandardProfile.

2. The user-space program calls the ControlService function. Can you set a breakpoint with WinDbg to see what is executed in the kernel as a result of the call to ControlService ?

At the ControlService, the malicious driver is unloaded so that we cannot find it. Therefore, we should set two breakpoints. The first breakpoint is set at virtual machine Windbg to stop the calling forControlService( ). The second breakpoint is set at host machine to stop the driver from being deleted. After the program hit and breakpoint and keeps running, we can see the code where the driver is loaded and unloaded.

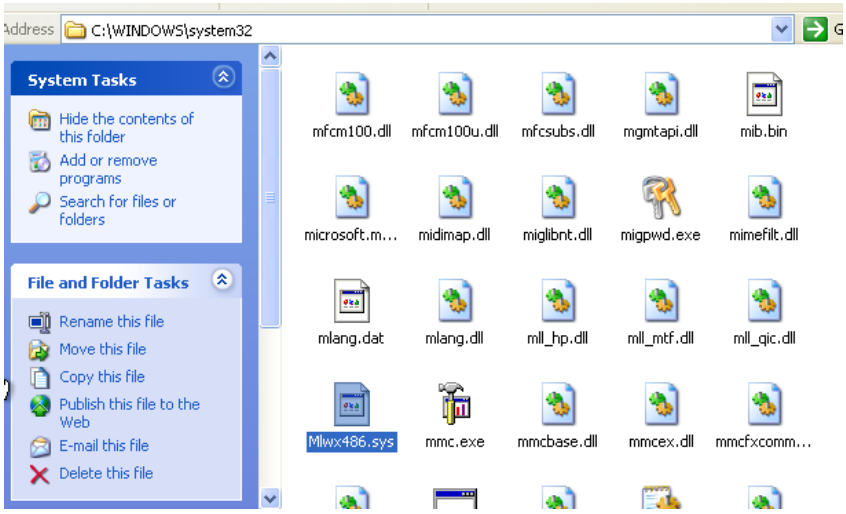
3. What does this program do?

The program creates a service named Lab10-01 to execute the malicious driver under same name. The driver lab10-01 is going to access to registry and change the attribute value of firewall from enable to disable. Then the displaying value is 0x000000. After the modification to firewall attribute, the malware unloads the driver so no one can see it.

Lab10-02

Steps of Processes

Before we run the malware, we need to go through the assembly code and see what it does. The program creates a file named Mlwx486.sys at C:\WINDOWS\system32 By analyzing the program at IDA Pro, we learn that the program tries to get access to the resource section because of FindResource( ) and LoadResource( ). We can use resource hacker to check if the PE header contains another file. But I didn't use it. I assume the PE header contains another malicious content. The program will get access to the malicious content and then implement the malicious function. We can check how does the program get access to the malicious content. At location 0x40103F, the program calls CreateFile( ) with path parameter C:\Windows\System32\Mlwx486.sys, which means the program is going to create a file at this path. We run the malware and then we check the path. Even though Mlwx486.sys is not an executable file, the malware could use service to trig Mlwx486 and then run it like an executable file. But there is one thing weird. Because of the future analysis, I have already known that it is a rootkit (seeing the SSDT), but I can see the file. The malicious content is not hidden. Nevertheless, we can see that the malware create a file at C:\Windows\System32\Mlwx486.sys, then it creates a service named 486 WS Driver at 0x4010BC to run Mlwx486.sys.



To double check the program is running and the driver is loaded, we can use What's run to see the current service. We will see that 486 WS Driver is under Lab10-02.exe with PID 1142.

The program has kernel component that is indicated by Mlwx486.sys and "486 WS Driver". Mlwx486.sys performs a driver function that is loaded at kernel. We can see the driver file which should have been hidden. Therefore, the good way to analyze it is to use IDA pro. The program has four functions. We sort four functions according to the level of importance, sub_10772, sub_10486, sub_10706, and NtQueryDirectoryFile. Here we don't go through sub_10772 because almost every driver file includes this function. The entry point of the driver is 0x107AB. Between start and end point, there are two function calls sub_10772 and sub_10706. We double click on sub_10706. RtlInitUnicodeString( ) is called twice and MmGetSystemRoutineAddress is called twice. The RtlInitUnicodeString routine is obsolete and is exported only to support existing driver binaries. The MmGetSystemRoutineAddress routine returns a pointer to a function specified by SystemRoutineName. Drivers can use this routine to determine if a routine is available.

```
INIT:000107BC aKeservicedescr:                        ; DATA XREF: sub_10706+1Bîo
INIT:000107BC                 unicode 0, <KeServiceDescriptorTable>,0
INIT:000107EE aN              db 'N',0                ; DATA XREF: sub_10706+10îo
INIT:000107F0 aTquerydirector:
INIT:000107F0                 unicode 0, <tQueryDirectoryFile>,0
INIT:00010818 __IMPORT_DESCRIPTOR_ntoskrnl_exe dd rva off_10840
INIT:00010818                                         ; DATA XREF: HEADER:000102D8îo
INIT:00010818                                         ; Import Name Table
INIT:0001081C                 dd 0                    ; Time stamp
INIT:00010820                 dd 0                    ; Forwarder Chain
INIT:00010824                 dd rva aNtoskrnl_exe    ; DLL Name
INIT:00010828                 dd rva __imp_NtQueryDirectoryFile ; Import Address Table
INIT:0001082C                 align 20h
```

In this case, the system routine address should be NtQueryDirectoryFile address. NtQueryDirectoryFile is always used to hide file like a rootkit. The return value is FILE_BOTH_DIR_INFORMATION, which is normally be used at findnextfile( ) in order to enumerate files. The file could be hidden by FileIdBothDirectoryInformation in FileInformationClass. We can try to use the following code to perform file hiding.

```
if ((FileInformationClass==FileBothDirectoryInformation ||
FileInformationClass==FileIdBothDirectoryInformation) && result ==
STATUS_SUCCESS){
if (lstrcmpi(ps,diskName)==0) {
   if (FileInformationClass==FileBothDirectoryInformation) {
PFILE_BOTH_DIR_INFORMATION Prev,info =
(PFILE_BOTH_DIR_INFORMATION)FileInformation;
UINT cnt=0;
```

```
   while (info->NextEntryOffset>0) {
      Prev=info;
      info =(PFILE_BOTH_DIR_INFORMATION)((DWORD)info+(DWORD)info-
>NextEntryOffset);
      UINT bl;
      if (folderName[0]==0) bl =  lstrcmp(info->FileName,nameProcess); else bl =
lstrcmp(info->FileName,folderName);
      if (bl){

         if (info->NextEntryOffset==0){
            Prev->NextEntryOffset=0;
            Prev->FileIndex=info->FileIndex;
            if (cnt==0) return STATUS_NO_SUCH_FILE;
         }
         else {
         Prev->NextEntryOffset = Prev->NextEntryOffset+info->NextEntryOffset;
         Prev->FileIndex = info->FileIndex;
         }
            }

   cnt++;
   }
   }
```

After the file is hidden, the program calls a function RtlCompareMemory( ). This function is used to return the string length. In this case, we know that the string length should be 8 bytes. The first parameter start with 0x4D which is 'M' is ASCII code. We can assume the string is Mlwx. But we cannot make assure until we use WinDbg to check it.

The service 486 WS Driver run the content at driver. When we analyze Mlwx486.sys in IDA Pro, we can see the following code that the program is interacting with SSDT table. Therefore we launch WinDbg and see the content at SSDT table.

```
INIT:00010706                   mov     edi, edi
INIT:00010708                   push    ebp
INIT:00010709                   mov     ebp, esp
INIT:0001070B                   sub     esp, 10h
INIT:0001070E                   push    esi
INIT:0001070F                   mov     esi, ds:RtlInitUnicodeString
INIT:00010715                   push    edi
INIT:00010716                   push    offset aN       ; "N"
INIT:0001071B                   lea     eax, [ebp+var_8]
INIT:0001071E                   push    eax
INIT:0001071F                   call    esi ; RtlInitUnicodeString
INIT:00010721                   push    offset aKeservicedescr ; "KeServiceDescriptorTable"
INIT:00010726                   lea     eax, [ebp+var_10]
INIT:00010729                   push    eax
INIT:0001072A                   call    esi ; RtlInitUnicodeString
INIT:0001072C                   mov     esi, ds:MmGetSystemRoutineAddress
INIT:00010732                   lea     eax, [ebp+var_8]
INIT:00010735                   push    eax
INIT:00010736                   call    esi ; MmGetSystemRoutineAddress
INIT:00010738                   mov     edi, eax
INIT:0001073A                   lea     eax, [ebp+var_10]
INIT:0001073D                   push    eax
INIT:0001073E                   call    esi ; MmGetSystemRoutineAddress
INIT:00010740                   mov     eax, [eax]
INIT:00010742                   xor     ecx, ecx
```

Here is the SSDT table. We saw that there is one distinct entry which means it is a rootkit.

```
80501d6c  8056f41a 8056ddb2 805cba9a 8061b702
80501d7c  8060d940 805ea790 805c1322 805e3a88
80501d8c  805e36ec 8059f7a2 8060b2fe 805b9806
80501d9c  805c15ae 805e3aa6 805e385c 8060d25a
80501dac  8063bd22 805bf3d2 805ede78 805e9aa2
80501dbc  805e9c8e 805ada94 80605386 8056c14e
80501dcc  8060cbfa 8060cbfa 8053c03e 80606f12
80501ddc  80607b72 f8d26486 805b3e6c 8056f44a
80501dec  8060544e 8056c2a2 8060c386 8056fcc6
80501dfc  805cbf6c 8059a77c 805c2c88 805c1854
80501e0c  805e3b86 80607310 8060e10a 8056de5a
80501e1c  8061ba28 8061947e 8060d9e8 805bb0d8
```

f8d26486 is different so we should pay attention to this function. We keep using WinDbg to analyze the function. We will notice that the assembly code at f8d26486 area is just like the codes at IDA Pro from 0x104CC to 0x104F0. The first main difference is the function call for RtlCompareMemory( ) is replaced by a relative address. ptr [Mlwx486+0x590 (f8d26590)] is the location of the function call. The second main difference is the parameter at RtlCompareMemory( ) is replaced by offset [Mlwx486+0x51a (f8d2651a)].

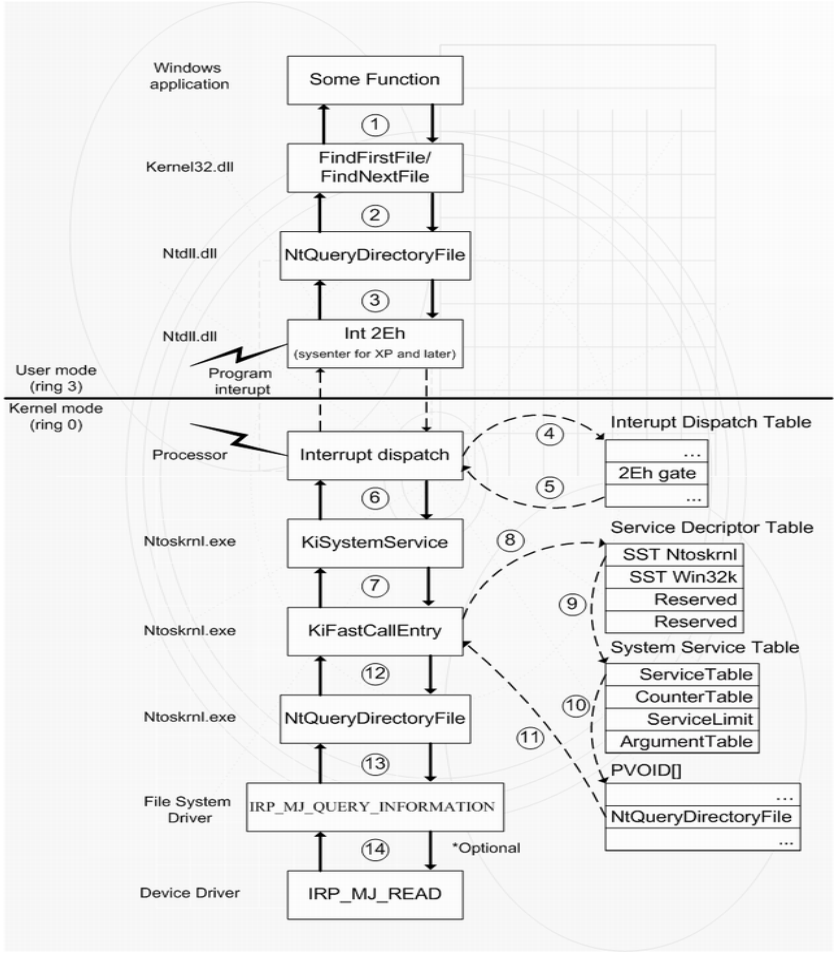
Therefore, we can infer that "f8d26590" is the address of RtlCompareMemory( ) and "f8d2651a" is the address of second parameter. Recall that the function has three parameters. The first parameter is passed by eax. And here the value in eax is loaded by [esi+5e], the second parameter is located at "f8d2651a" and the third parameter is 8. We use db command to see the content of parameters. When we applied the command db esi+5e, db f8d2651a, we see the content at the first parameter is a header file named Installer.h, the content at the second parameter is four characters "Mlwx". Therefore, our former assumption is correct.

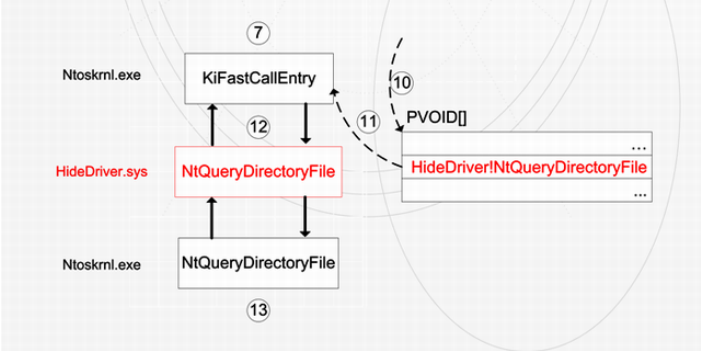
By analyzing the malware, we learn that it is a rootkit which makes use of NtQueryDirectoryFile to hide file. However in my case, I can see the driver file at beginning and I also can see the malicious code related with different function calls.

Especially when I look at the answers at the back of the book, I am totally confused. I wonder if the malware has loophole or the process of changing the structure of FileIdBothDirectoryInformation( ) didn't success. Right now I am still looking for a method to check the structure of files.

Issue or Problems

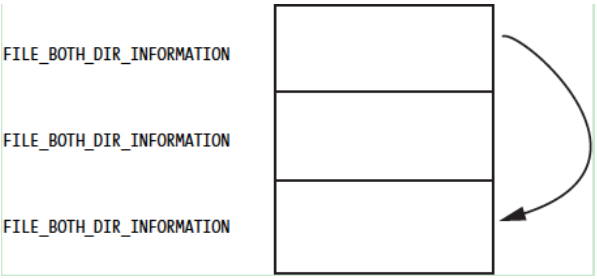
After I look the answers at the back of the book, I realize that there should be process called "recovery the hidden file". But, the file is not hidden how can I recover it? Anyway, I still check the process to recovery the file. I disable the service and reboot my machine. The malicious driver is still at the path C:\Windows\System32\Mlwx486.sys. And I cannot run the malware twice otherwise the service open failed and stopped. Therefore, there is only one chance to analyze it. Just set mirror point at VM in case I screw the one chance. The other problem is related to the function NtQueryDirectoryFile. I used to know that the function is always associated with rootkit to hide file. But I am not sure the specific structure until I saw this:





The first two pictures are retrieved from http://www.codeproject.com/Articles/32744/Driver-to-Hide-Processes-and-Files.

The third picture is from the back of the book.



The documentation related to NtQueryDirectoryFile is not that much. Therefore, a good way to see the function purpose is to debug it step by step.

Conclusion

This malware is a hidden rootkit while it is not hidden on my machine. Theoretically, if the rootkit didn't hide itself, which means the NtQueryDirectoryFile didn't work. But in this case, the function NtQueryDirectoryFile call success. I think my debugging process might have problems.

The rootkit is designed to hide a file start with Mlwx. The only file start with Mlwx is the driver file Mlwx486.sys.

Case Study

N/A

Question Review

1. Does this program create any files? If so, what are they?

Yes. The malware create a file name Mlwx486.sys at the path C:\Windows\System32\Mlwx486.sys.

2. Does this program have a kernel component?

Yes, the program has kernel component that hides the file start with Mlwx. The kernel component is loaded and execute by a service named "486 WS Driver".

3. What does this program do?

The program creates a service and loads the malicious driver that changes the FILE_BOTH_DIR_INFORMATION at NtQueryDirectoryFile in order to hide file.

Lab10-03

Steps of Processes

Before we launch the malware, we analyze it first by IDA pro. The executable file does not contain many codes. The most important function is CoCreateInstance with parameter 0x2C. The value indicate the 12th functions at the header file of CoCreateInstance( ), which is navigation to a specific website. The address of the website is passed by SysAllocString( ). http://www.malwareanalysisbook.com/ad.html. At the location 0x401102, the program starts an infinite loop that it keep launch the website every 0x7530 milliseconds (30 seconds). Therefore, we should set a breakpoint within the infinite loop. We still want the malware to pop advertisement page, so we set a breakpoint at 0x401121.

```
.text:00401106                  mov     eax, [esp+30h+ppv]
.text:0040110A                  push    edx
.text:0040110B                  lea     edx, [esp+34h+pvarg]
.text:0040110F                  mov     ecx, [eax]
.text:00401111                  push    edx
.text:00401112                  lea     edx, [esp+38h+pvarg]
.text:00401116                  push    edx
.text:00401117                  lea     edx, [esp+3Ch+var_10]
.text:0040111B                  push    edx
.text:0040111C                  push    esi
.text:0040111D                  push    eax
.text:0040111E                  call    dword ptr [ecx+2Ch]
.text:00401121                  push    7530h           ; dwMilliseconds
.text:00401126                  call    edi ; Sleep
.text:00401128                  jmp     short loc_401102
.text:0040112A ; ---------------------------------------------------------------
.text:0040112A
```

The reason to set the breakpoint is that we don't want to always see the advertisement

page and we still need to check the service.

```
C:\>sc query "Process Helper"

SERVICE_NAME: Process Helper
        TYPE               : 1  KERNEL_DRIVER
        STATE              : 4  RUNNING
                                (STOPPABLE,NOT_PAUSABLE,IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0

C:\>
```

The malware loaded driver at C:\Windows\System32\Lab10-03.sys and create a service Process Helper to execute the malicious code at driver file. The program pass a few parameters to driver by function call DeviceIoControl( ). But most of parameters don't have value. The parameter IO_ControlCode has value 0x0ABCDEF01. The parameter hdevice has value stored at eax. Here the value in eax is the device named "ProcHelper", which means the operation is performed on ProcHelper. But the lpOutBuffer & OutBufferSize, and lpInBuffer & InBufferSize are all set to 0. A pointer to the input buffer that contains the data required to perform the operation is 0. Then there isn't sending request. A pointer to the output buffer that is to receive the data returned by the operation is also 0. So, there isn't any indicator that the data is received. I will mark this place as a problem.

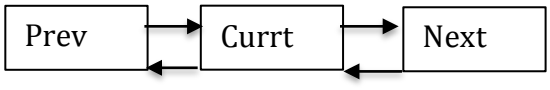
We analyze Lab10-03.sys which is related to the kernel component by IDA Pro. The first subroutine to analyze is sub_10706. The program create a device named ProcHelper by function call IoCreateDevice( ). The device is going to be used for driver in the future. After that, the program creates a symbolic link between two object. At the location sub_10666, we can see the details to create the symbolic link.

The link here must be double link list otherwise the pointer cannot point backward.

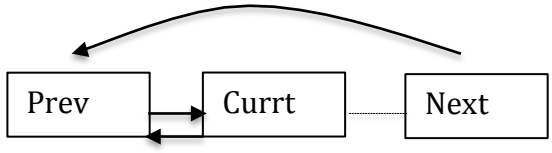
mov ecx, [eax+8Ch]; Assume eax value is the current process. Then the next process address is 0x8C;

add eax, 0x88; subtract by 4 byte. So it gets the previous objects address 0x88;

mov edx, [eax]; edx is 0x88;



mov [ecx], edx; ecx is 0x88 rather than 0x8C;

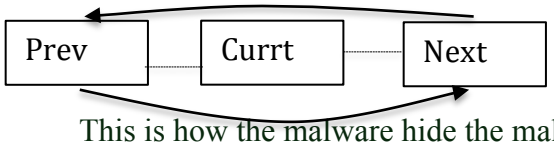


dash line means isolated pointer.

mov ecx, [eax]; ecx is 0x88;

mov eax, [eax+4]; eax is 0x8C;

mov [ecx+4], eax; ecx is 0x88;



so far the current process is isolated.

This is how the malware hide the malicious process from being detected. After the structure of link list is changed, the program will delete the useless pointer at sub_1062A and then delete the device. However, even though the device is deleted, the process is still there.

I don't think the program could stop once it is running. A good way to control it is to set a breakpoint before it launch because when the program is running, we cannot use WindDbg to set a breakpoint or remove the service from command line. I tried both but they just don't work.

Issues or Problems

In this lab, the kernel component sends request to user-level function call. But the InputBuffer and Outputbuffer at the function DeviceIoControl( ) are set to 0. Then how can the kernel part interact with the function call to pop the advertisement?

Conclusion

This malware is an advertisement-popping program that launch the website every 30 seconds. And it cannot stop. So we should set a break point before the malware is running otherwise it will be really annoyed. And also remember that do not open the Process Explorer otherwise it will keep updating.

Review Questions

1. What does this program do?

The malware create a service named Process Helper to load the device named

"ProcHelp". The device sends request to function DeviceIoControl and then interact with kernel. The malware will delete the symbolic link of the current malicious process in order to hide the process. The process is designed to keep popping advertisement page http://www.malwareanalysisbook.com/ad.html every 30 seconds.

2. Once this program is running, how do you stop it?
Reboot the system. "sc delete process helper" is not working at this case.

3: What does the kernel component do?
The kernel component is designed for hiding the current process. The data structure involved here is double link list. See the picture at the Steps of Processes. The device send request and receive operations from DeviceIoControl and then delete the link.