Xiaoyi Zhou

Purdue University CIT581

Malware Forensics

Lab 11: Malware Behavior

Due October 22, 2014

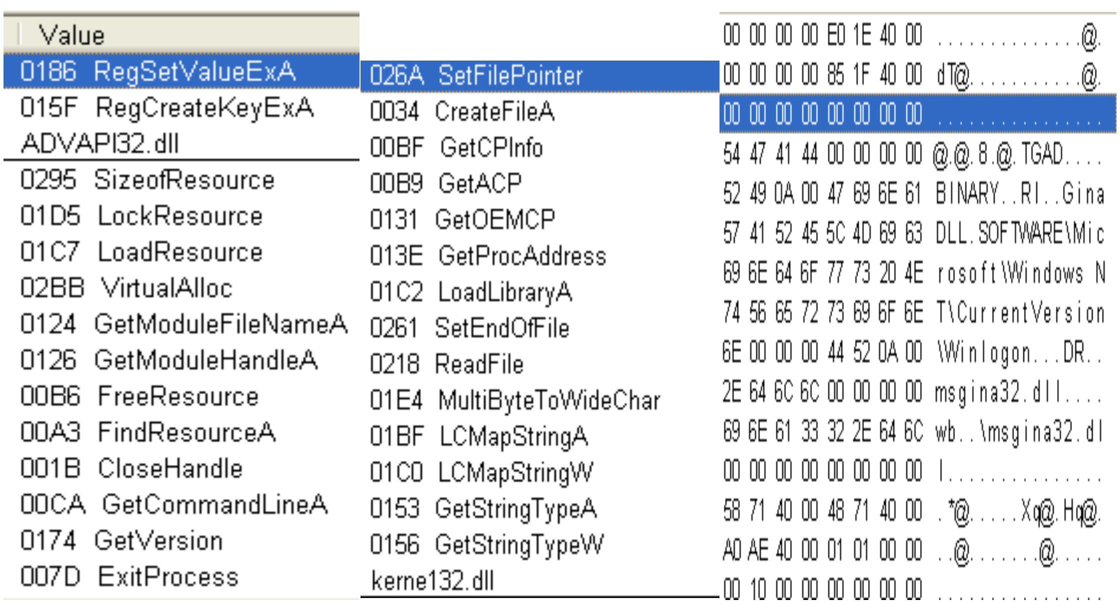
Instructor: Samuel Liles

Abstract

Lab11-01

Steps of Processes

I will not run this executable file before analyzing the exports and strings. First I go through the file by PEview and then ckeck the exports functions and data section. At the exports section, I can see some functions related to modify registry, such as RegSetValue and RegCreateKey. Also some functions related to create file, like CreateFile and ReadFile, which means this malware create new file and also change the registry key. At the left, there are FreeResource and FindResource which both indicate that the malware search and extract the hidden embedded resource. It could be PE file or DLL file. A good way to extract the file is using resource hacker, while in this case we also can run the malware on virtual machine and see what's going on. The next step is to check data section. I found some interesting strings for example Winlogon, msgina32.dll and BINARY. Normally, the file stored credential information is msgina.dll instead of msgina32.dll. But the new dll file is stored at path \winlogon. Therefore the msgina32.dll might be a malicious file that steal credential information and pass the information to msgina.dll to logon. At the book page 235, it clearly points out how the malicious dll steal information by standing between Winlogon and msgina.dll.



Before running this malware, I open it at IDA Pro and check my above assumption. The main function has two functions call. First function call sub_401080( ) takes LpModuleName as paramater. But the value is 0 so we don't have to pay too much attention on it because the function will return the real resource's name. Double click on the function and I can see a lot of memory mapping. The crucial location is loc_4010B8 where FindResource( ) is called. And the parameters are the resource attributes passed by hModule, LpType and LpName. After FindResource( ), a serials of function calls related to resource are called. Those operations are aimed to check the memory space for the resource and then load the resource. At the location 0x401166, I saw the following information which indicated the memory space for the dll file is prepared the loading process is also prepared. So msgina32.dll is written. That's the main purpose of sub_401080( ).

```
.text:00401161                 push    offset aWb      ; "wb"
.text:00401166                 push    offset aMsgina32_dll_0 ; "msgina32.dll"
.text:0040116B                 call    fopen
```

Now, skip a lot of memory mapping process and directly go to function sub_40100( ), here I can see many functions related to registry. The first function call is located at 0x401021 named RegCreateKeyEx( ). This function is designed to create/open a specific registry key. In this case, the specific key is "SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" which mainly loads the credential information of user. When the specific is found and opened, the program call RegSetValueEx( ) to change the value of the key. In this case, the content at GinaDLL is set to the pathof msgina32.dll. Third party is always found at this place. I assume that Gina will be implemented at msgina32.dll. But to see if my assumption is true or not, I need to check msgina32.dll. The file should have many exports that are required by Gina.

Name	Type	Data
forceunlocklogon	REG_DWORD	0x00000000 (0)
GinaDLL	REG_SZ	C:\Documents and Settings\Administrator\Desktop\Practical Malware ...
HibernationPrevi...	REG_DWORD	0x00000001 (1)
LegalNoticeCaption	REG_SZ	
LegalNoticeText	REG_SZ	

So, the executable file is only used to extract the malicious dll file and set value of GinaDLL. Now we can launch the malware and analyze the msgina32.dll file. I launch the malware and see a new created file after one second. Scan msgina32.dll to see the exports. There are many exports starting with Wlx. The book mentions that Clearly, if you find that you are analyzing a DLL with many export functions that begin with the string Wlx , you have a good indicator that you are examining a GINA interceptor. So, msgina32.dll should be a GINA interceptor.

WlxActivateUserShell

WlxDisconnectNotify

WlxDisplayLockedNotice

WlxDisplaySASNotice

WlxDisplayStatusMessage

WlxGetConsoleSwitchCredentials

WlxGetStatusMessage

WlxInitialize

WlxIsLockOk

WlxIsLogoffOk

WlxLoggedOnSAS

WlxLoggedOutSAS

WlxLogoff

WlxNegotiate

WlxNetworkProviderLoad

WlxReconnectNotify

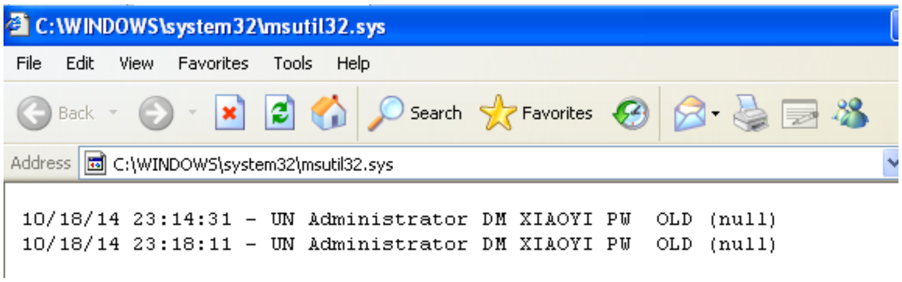
(exports from msgina32.dll)

Then open IDA Pro to see details. Code reference show that DllMain Entry point is being examined. When the system starts or terminates a process or thread, it calls the entry-point function for each loaded DLL using the first thread of the process. Therefore, codes at DllMain Entry point are important because it might help the malware achieve persistence. DllMain Entry has three parameter, hinstDLL, fdwReason and lpvReserved. Here the program compares the value of fdwReason with number 1. fdwReason has possible four value. Value 1 means DLL_PROCESS_ATTACH the DLL is being loaded into the virtual address space of the current process as a result of the process starting up or as a result of a call to LoadLibrary. In another words, no matter which process is running in the system when the machine starts, msgina32.dll will be attached to the process and the comparison result of fdwReason and 1 is 0; the program jumps to 0x 1000105F. That's the way the program achieve persistence. After 0x1000105F, the program calls GetSystemDirectory( ) and lstrcat( ) to grab the path of msgina.dll and combine the path with the name of msgina together; save the result at edx and pass it as parameter to LoadLibrary. This function loads the specified module into the address space of the calling process. The return value will be the handle to a module. In this case, this function will return a handle to msgina.dll and save the handle at hModule.

The DLLMain function ends at the saving of hModule. But the most malicious codes haven't been analyzed. Main function help the program achieve persistence and also pass data to real msgina.dll through exports functions. The book mentions that Most of these exports simply call through to the real functions in msgina.dll. In the case of GINA interception, all but the WlxLoggedOutSAS export call through to the real functions. Winlogon calls this function when it receives a secure attention sequence (SAS) event while no user is logged on. Double click on WlxLoggedOutSAS, the program takes WlxLoggedOutSAS as parameter and immediately call sub_10001000 where GetProcAddress( ) is called. So the main purpose sub_10001000 is aimed to retrieve the address of WlxLoggedOutSAS. Since sub_1000100 is going to be used many times in future, I rename it as Address_Retrieve( ). After the address is retrieved, the program starts memory mapping from 0x100014AC until 0x00014FC where sub_10001570 is called. The function takes eax, edx, ecx, and "UN %s DM %s PW %s OLD %s" as parameters so that the credential information were stored in general registry but passed to another function call right now. This is exactly place where credential information is stolen. Rename the function as "Cred_Lost ( )" and double click on it. The function contains a lot of operation related to file. Firstly it calls vsnwprintf( ) to write the string according to the specific format UN %s DM %s PW %s OLD %s. Then, it calls wfopen, wstrtime and wstrdate to open msutil32.sys and set data/time. Last, it calls fwprintf to write the string according to format "%s %s - %s". Therefore, all the credential information is stolen and stored at msutil32.sys. The malware will return the stolen value back to WinLogon so that the malicious process cannot be detected.

If we are not sure the path of msutil32.sys, we can apply explore function at machine. The path of msutil32.sys is at C:\Windows\System32. Remeber that msutil32.sys will not be created until WlxLoggedOutSAS is implemented. So, to see the stolen credential, log off the system and log in again. Now we can see msutil32. sys is created. The created date and time are same with log off- log in time.

This is not a driver because I can open and see the detail content at TextEditor. Since I haven't set any password to my virtual machine, so the content here is just like the following screenshot.



Besides WlxLoggedOutSAS, most of exports has same format like following:

push offset aWlxnegotiate_0 ; "Wlx[] [] [] [] [] []"; the name of exports.

call sub_10001000; get the process address.

jmp eax

```
                public WlxScreenSaverNotify
WlxScreenSaverNotify proc near          ; DATA XREF: .rdata:off_10002348↓o
                push    offset aWlxscreensav_0 ; "WlxScreenSaverNotify"
                call    Address_Retrieve
                jmp     eax
WlxScreenSaverNotify endp

; ---------------------------------------------------------------------------
                align 10h
; Exported entry  48. WlxShutdown

; =============== S U B R O U T I N E =======================================

                public WlxShutdown
WlxShutdown     proc near               ; DATA XREF: .rdata:off_10002348↓o
                push    offset aWlxshutdown_0 ; "WlxShutdown"
                call    Address_Retrieve
                jmp     eax
WlxShutdown     endp

; ---------------------------------------------------------------------------
                align 10h
; Exported entry  49. WlxStartApplication

; =============== S U B R O U T I N E =======================================

                public WlxStartApplication
WlxStartApplication proc near           ; DATA XREF: .rdata:off_10002348↓o
                push    offset aWlxstartappl_0 ; "WlxStartApplication"
                call    Address_Retrieve
                jmp     eax
WlxStartApplication endp
```

Check functions starting with Wlx at MSDN, the explanations are similar at the first sentence: The WlxLoggedOnSAS function must be implemented by a replacement GINA DLL. Winlogon calls this function when it receives a secure attention sequence (SAS). Therefore, no matter if the export functions are called, the return value will be passed to Winlogon and continue the logon process.

Issues or Problems

The first issue I met is when I analyzed the msgina32.dll file, at WlxLoggedOutSAS export code area, there is one line code:

call "??2@YAPAXI@Z" ; operator new(uint); I don't understand why the function appears at random characters. And what is the purpose of this function call? The second issue I met is also at WlxLoggedOutSAS export code area. Right before sub_10001570, there should be four parameters (user name, domin name, new password, old password) stored in four different general registers. But I can only count three parameters.

Conclusion

This malware steals credential information such as username, domain name, new password, and old password. It achieves persistence by applying trojanized system binaries technique. The stolen information is stored at msutil32.sys under C:\Windows\System32. It also creates msgina32.dll as GINA interception when the executable file is launched. Most of malicious word is implemented at msgina32.dll.

Review Questions

1. What does the malware drop to disk?

The malware drop msgina32.dll at the location where the malware is launched. In my case, the path is C:\Documents and Setting\Desktop\Practical Malware Analysis\Chapter11

2. How does the malware achieve persistence?

Using DLL_PROCESS _ATTACH at DLLMain Entry Point to load the dll into every process when the system starts. Also it changes the registry where the original GinaDll stored. Once the system boots/reboots, the malicious msgina32.dll is loaded.

3. How does the malware steal user credentials?

Create GINA interception which is msgina32.dll. It steals and stores all the credential information related to system authentication process.

4. What does the malware do with stolen credentials?

It creates a file msutil32.sys under C:\Windows\System32. Then it stores the credential information at this file. It is not a driver.

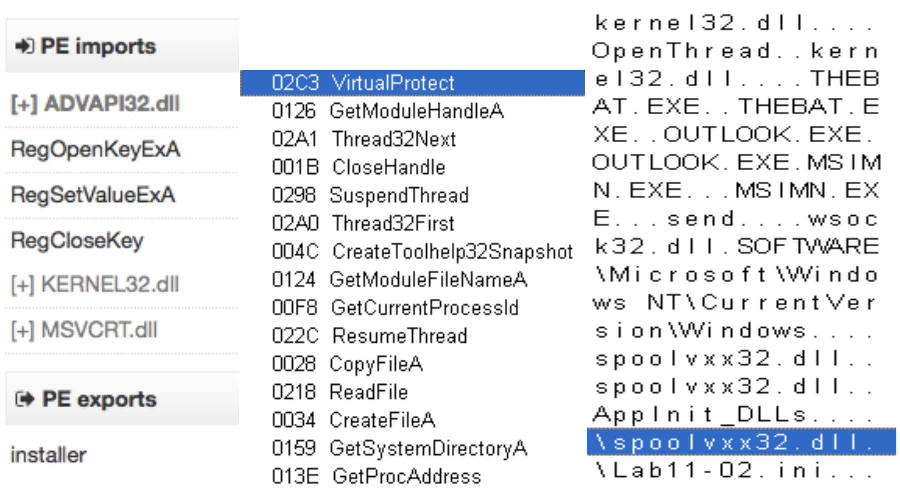
5. How can you use this malware to get user credentials from your test environment?

Run the malware; Reboot the computer; log off and then log in again; open msutil32.sys at notepad and see the credential information.

Lab11-02

Steps of Processes

First I check the import function and export function of this malware. The malware has only one export but many interesting imports. The only one export is installer which help install the malware by rundll32.exe. As for imports, it indicates that the malware will change registry value; copy content to a new file; get the current process address and load DLL file with it; takes a snapshot of the specified processes; the string contains a lot of useful information; outlook.exe and msimn.exe and thebat.exe seemingly indicate the malware intercepts with email process; send( ) and wsock32.dll indicate the malware use Internet to intercept with email function; \Microsoft\Window NT\CurrentVersion\Windows should be path where we give a high attention; spoolvx32.dll might be created by this malware; Applnit_DLLS might be the way that helps the malware achieve persistence;



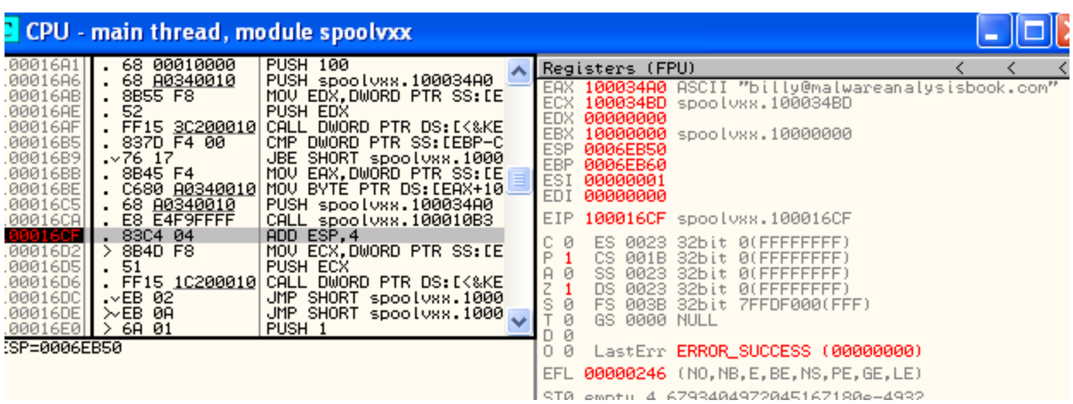
Lab11-02.ini also intercept with this malware. I try to open it but the content is not readable: "CHMMXaL@MV@SD@O@MXRHRCNNJBNL", this is exactly what the text editor shows to me except the double quotation. So I open IDA Pro to examine the malware before installing it.

Sometimes export function is more important than DLLMain because the installer can indicate what does the program intend to do when it is installed. Actually I check DLLMain function first and I just figured little information. So I switch to see the export installer. The program gets access to SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows and set the value to AppInit_DLLs. Our book mention this technique that helps the malware achieve persistence. Therefore after the installing the malware, value of APPInit_Dlls is not user32.dll anymore. It will be spoolvxx32.dll as mentioned at IDA Pro location 0x100015B. And malware also checks to see in which process the DLL is running before executing their payload. This check is often performed in DllMain of the malicious DLL. Since spoolvxx32.dll is attached to the current process which should have intercepted with user32.dll, spoolvxx32.dll must be generated. Few lines later, I saw sub_1000105B and CopyFile() for spoolvxx32.dll. sub_1000105B must contain the path of spoolvxx32.dll. So double click on it and see GetSystemDirectory( ). This function indicates the path C:\Windows\System32. This is the place where the new dll file can be found later. The old file is ExistingFileName which is presumably Lab11-02.dll. We can compare those two files. That is what will happen after installing the malware.

Now check the DLLMain function. First the program compares fdwreason with PROCESS_ATTACH at 0x1000161E. Right after the comparison success, it jumps to 10001629 otherwise it terminates. The malicious code has to wait for being executed until DLL is loaded to a process. Recall page 248 at our book, inline hooking overwrites the API function code contained in the imported DLLs, so it must wait until the DLL is loaded to begin executing. IAT hooking simply modifies the pointers, but inline hooking changes the actual function code. A malicious rootkit performing inline hooking will often replace the start of the code with a jump that takes the execution to malicious code inserted by the rootkit. So this malware might perform inline rootkit function, which will be determined later.

The program call sub_1000105B( ) again to retrieve the system root path and then combine and path with Lab11-02.ini by strncat. The file address is stored at destination lpBuffer 100034A0. Then the program finds the path and read file.

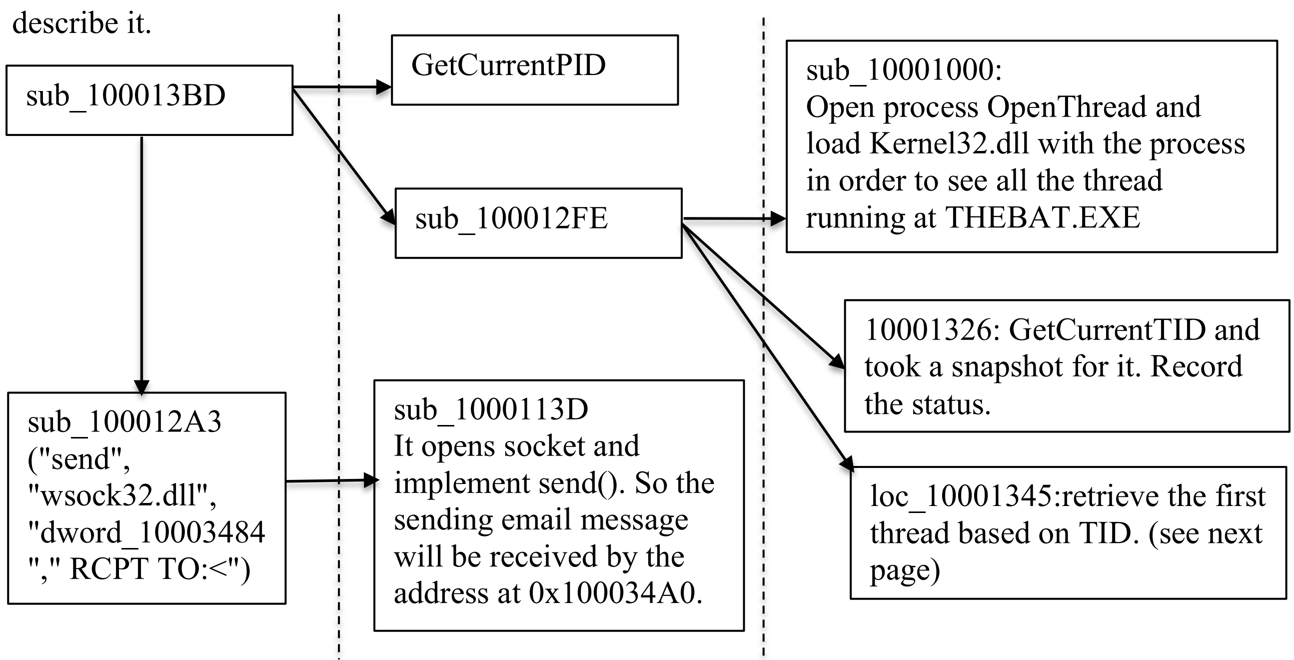
The next function call sub_100010B3 is located at 0x100016CA. This function takes the address of Lab11-02.ini as parameter. I mentioned above that Lab11-02.ini is not readable. Therefore, this function could be encryption or decryption function. I double click this function but cannot figure out what does the function do. It is a loop meanwhile doing some multiplication and arithmetic shift. I open OnlyDbg to check the encryption/decryption result. Set a breakpoint at 0x100016CF after the sub_100010B3 is done. So we can directly see the result which turns out that Lab11-02.ini contain an email address. billy@malwareanalysisbook.com.

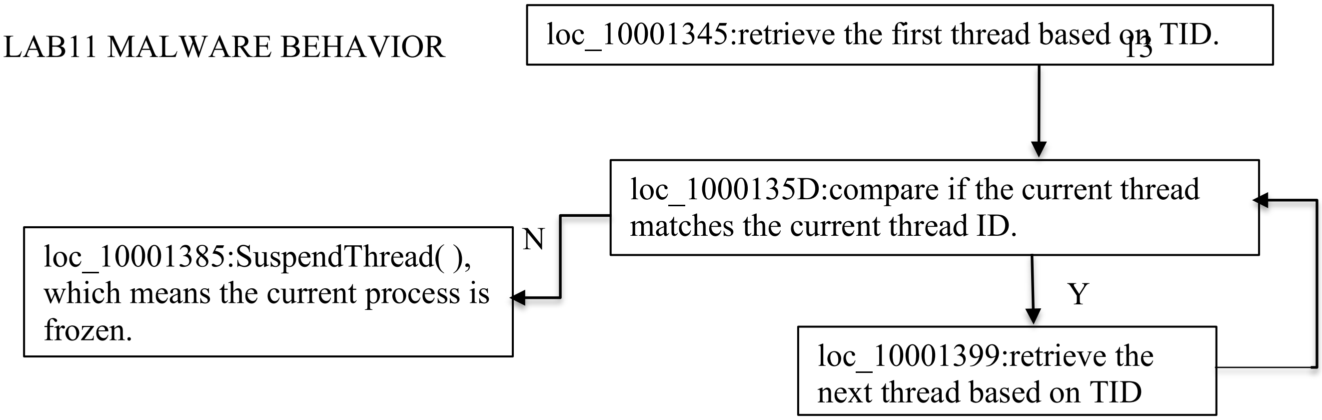


At 0x100016D6, the program called CloseHandle( ) and sub_100014B6 where most of malicious codes are stored. We rename the following process as inline hooking process. The process jumps to location 0x10001075 where GetModuleFileName( ) is called. This function is used to gain the qualified path for the file that is loaded by the current process. Buf1 is used to store the address of the current process module name. For example the current running process is thebat.exe. So, Buf1 is pointing to [thebat.exe]. If the current name of the process is not equal to 0, then the program jumps to loc_100014EC. Rename sub_10001075 and loc_100014EC before keep going.

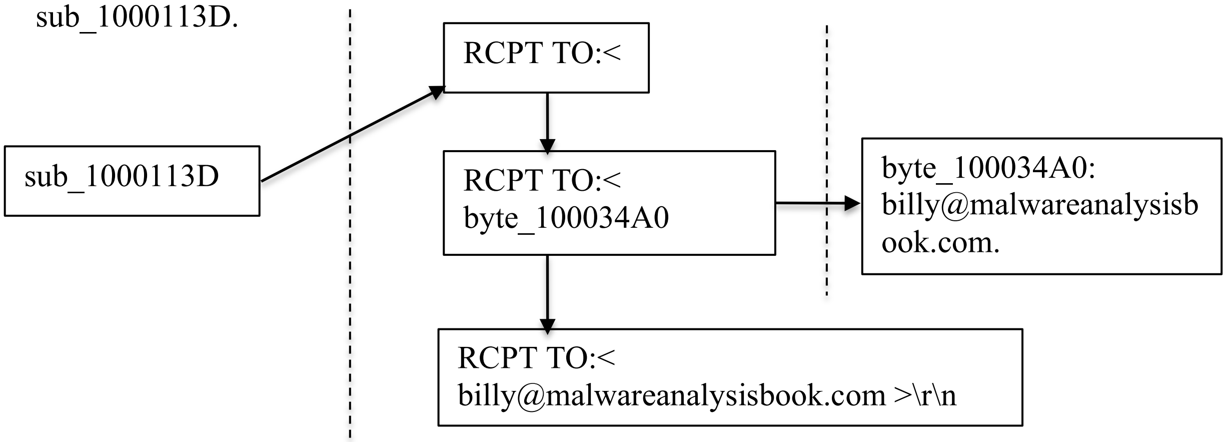
```
push    ebp
mov     ebp, esp
push    ecx
cmp     [ebp+arg_0], 0
jz      loc_10001587
lea     eax, [ebp+Buf1]
push    eax             ; int
push    0               ; hModule
call    GetProcessName
add     esp, 8
mov     ecx, [ebp+Buf1]
push    ecx             ; Str
call    sub_10001104
add     esp, 4
mov     [ebp+Buf1], eax
cmp     [ebp+Buf1], 0
jnz     short Inline_Hooking
jmp     loc_10001587
```

At Inline_Hooking part, sub_1000102D( ) takes Buf1 as parameter and calls toupper( ). It converts thebat.exe to THEBAT.EXE and return the uppercase string (we can rename sub_1000102D). The uppercase string is passed to memcpy( ) few times because the program is trying to compare the process name with the three targets process: THEBAT.EXE, OUTLOOK.EXE, and MSIMN.EXE. This malware is not designed for all processes. The specific target is email client. If current running process is one of the three, inline-hooking program will call loc_10001561. Since 1001561( ) contains too many functions to analyze, using C syntax or flow chart is a more convenient way to describe it.

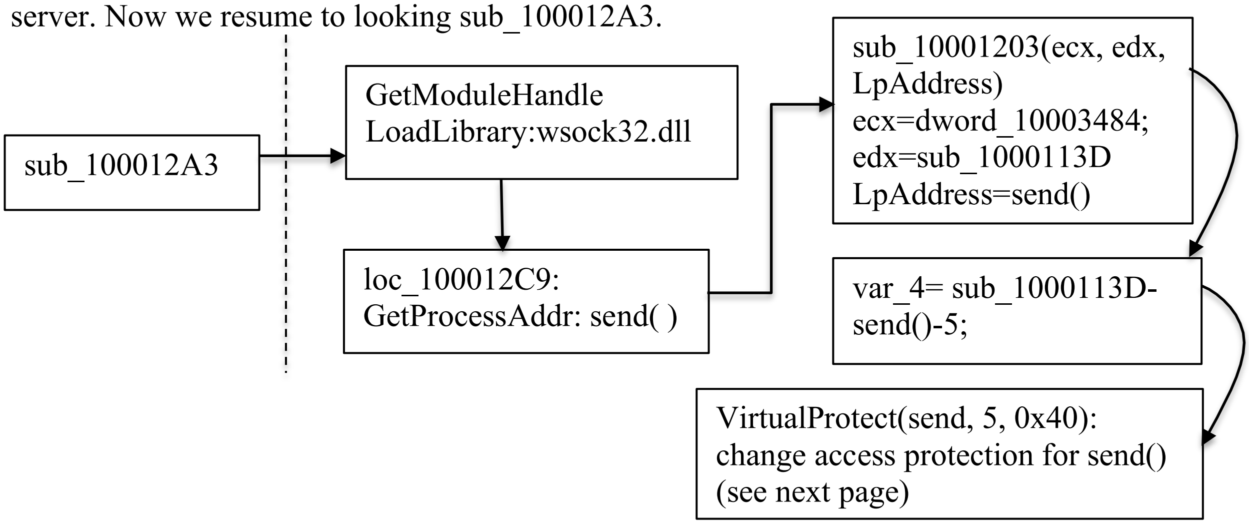


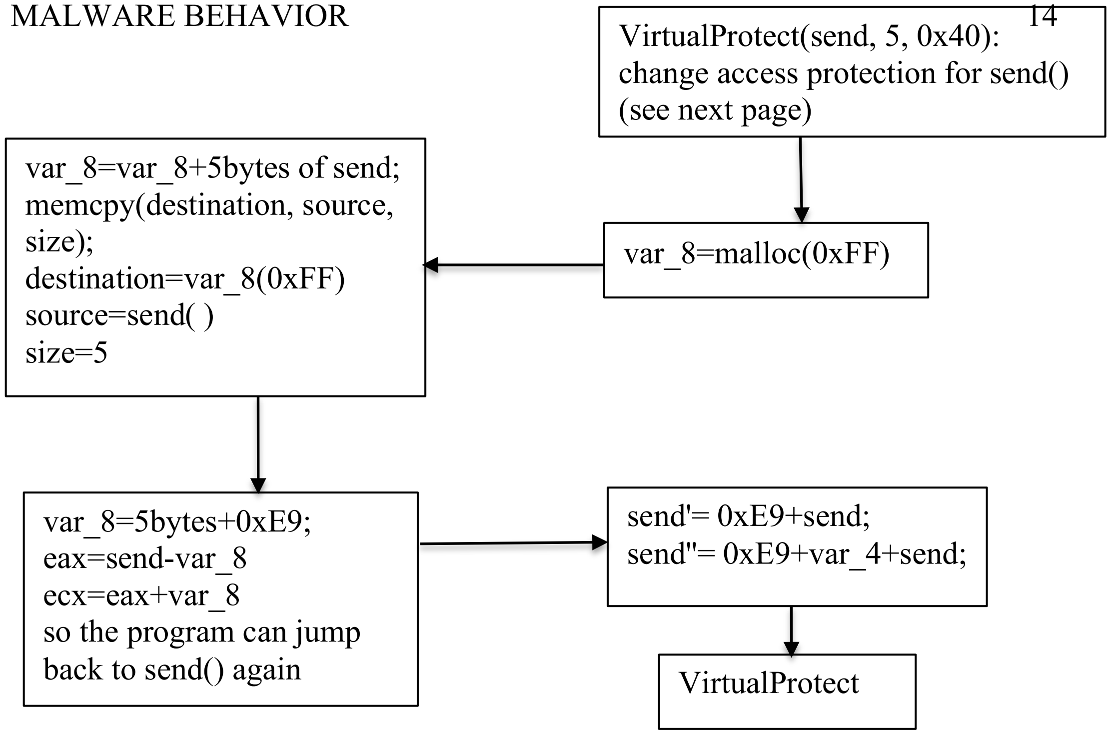


At sub_100012A3. dword_10003483 doesn't take any functions, neither important data. So it is regarded as variable. The malicious operations and email address is stored at sub_1000113D.



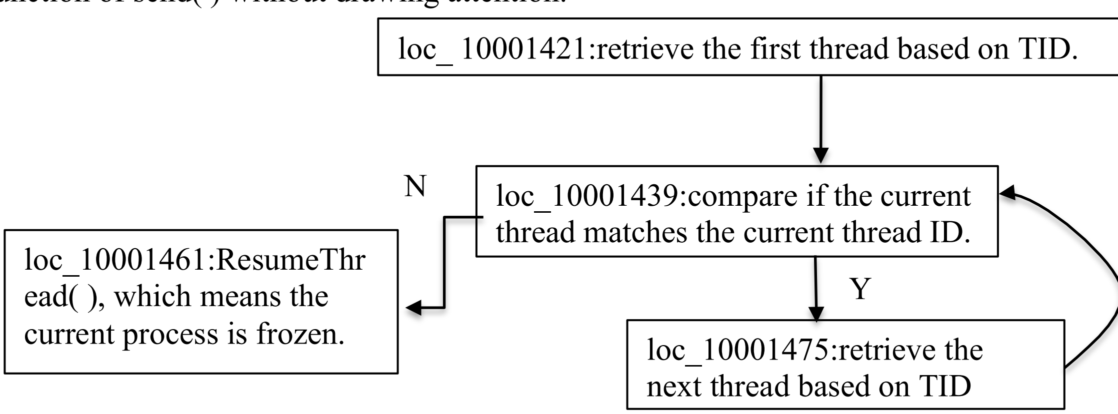
To see the client address stored at 0x100034A0. We double click on the address and check the code reference. The address is first used at DLLMain. When the program is trying to get access to Lab11-02.ini and set C:\Windows\System32\Lab11-02.ini as destination. So the content stored at 0x10034A0 should be the email address at Lab11-02. Each time user sends email via THEBAT.EXE or OUTLOOK.EXE or MSIMN.EXE, the email message will also be delivered to billy@malwareanalysisbook.com no matter if the user intends to send email to it. So the malware is kind of eavesdropping on user's email server. Now we resume to looking sub_100012A3.





The purpose of sub_100012A3 is to embed the malicious code to send( ). Operand code for jump is 0xE9 in assembly. The jump command is added to the beginning of send. And jump location is added right after jump command. If we denote var_4 as offset between send and sub_100012A3, the offset value must be added later in order to give location for "jmp". The content at address of send( ) should be jmp, sub_10001203( ), jmp, send.

Before email server performs function of send( ), it firstly jump to sub_1000113D where the malware embeds receipt RCPT TO:< billy@malwareanalysisbook.com >\r\n. After the embedding process is done, the program jumps back and keep performing function of send( ) without drawing attention.



The last function call at inline-hooking area is sub_10001499( ). The flow chart for sub_10001499 is similar to sub_100013BD that helps freeze current threads. It first get process ID and load kernel32.dll to OpenThread process. Then, it starts unfreeze the frozen threads. The different part is listed on the above charts. Opposite to sub_100013BD, sub_10001499 helps to unfreeze the current running thread. So the process keep running functionally and no one will notice malicious code has already been implemented.

After the analysis part is done, we can set Outlook server and launch the malware.

Issues or Problems

Still confused about the operations at sub_10001203( ). The program rewrites the first five bytes at address of send( ). But it doesn't have to be five. There should be a function check how many bytes need to rewrite in order to inject the malicious codes. Except that I think the codes are not difficult to understand.

Conclusion

This malware plunges itself to email process at the beginning function of send( ). It is an inline hooking function that eavesdropping on email content. The malicious client is billy@malwareanalysisbook.com. I wonder if we rewrite the content in Lab11-02.ini and change it to encrypted version of personal email address, maybe we are able to perform email eavesdropping function. But, unfortunately, the encryption/decryption schema is still unknown.

Reviewed Questions

1. What are the exports for this DLL malware?

installer

2. What happens after you attempt to install this malware using rundll32.exe?

The malware gets access to Lab11-02.ini, creates spoolvxx.dll which is identical to Lab11-02.dll. It also changes the registry value from user32.dll to spoolvxx.dll.

3. Where must Lab11-02.ini reside in order for the malware to install properly?

System directory. In my case it is C:\WINDOWS\System32

4. How is this malware installed for persistence?

By using AppInit_DLLs technique. Changing user32.dll to spoolvxx.dll. So every process needs user32.dll is infected.

5. What user-space rootkit technique does this malware employ?

It is inline hooking function that hides the malicious code before send( ) function call.

6. What does the hooking code do?

It adds billy@malwareanalysisbook.com as receipt to every sending email message.

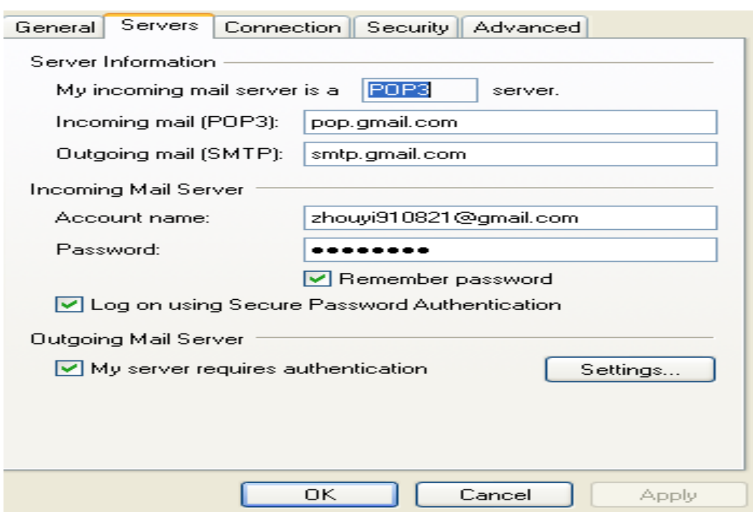
7. Which process does this malware attack and why?

The malware only pay attention to email function. for example THEBAT.EXE, OUTLOOK.EXE and MSMIN.EXE.

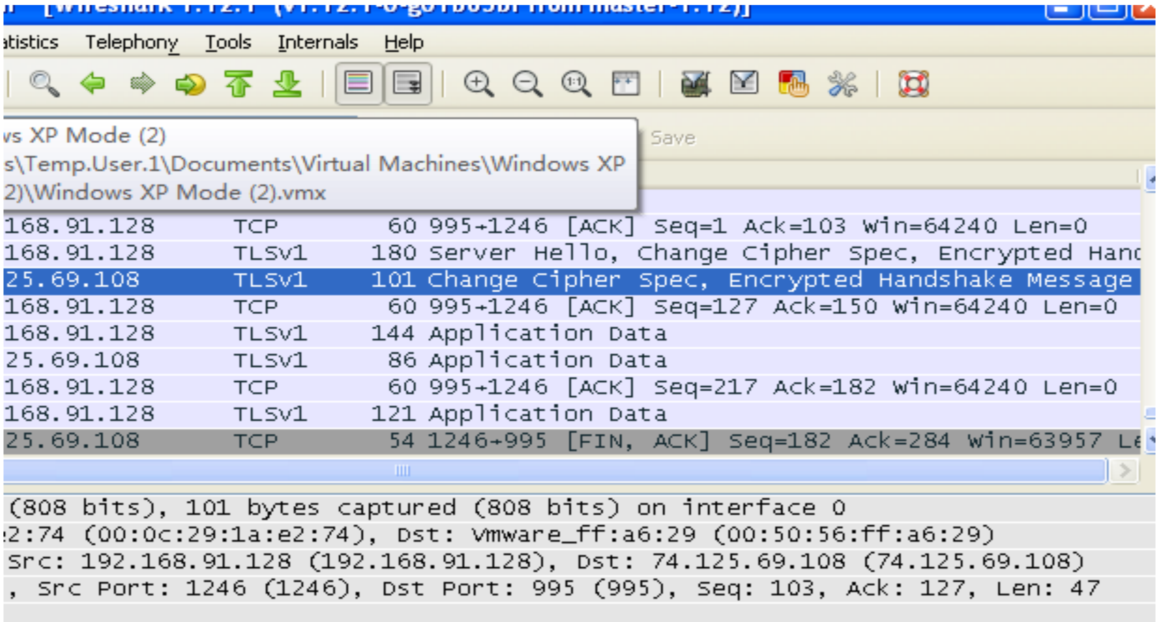
8. What is the significance of the .ini file?

The malicious email account: billy@malwareanalysisbook.com

9. How can you dynamically capture this malware's activity with Wireshark?



The malware should be attached to process of Outlook Express. So I set up Outlook Express with my Gmail account. Then start Wireshark to capture Internet activity. I use smtp4dev.exe to start a fake mail server which email address is "rob@rnwood.co.uk." The fake email server is retrieved from https://smtp4dev.codeplex.com. Then I send email from Outlook Express and check the capture information on Wireshark as following. We can see a serials actives and details. If we click on follow TCP stream, there will be more detail like how does Outlook express resolve the email address.

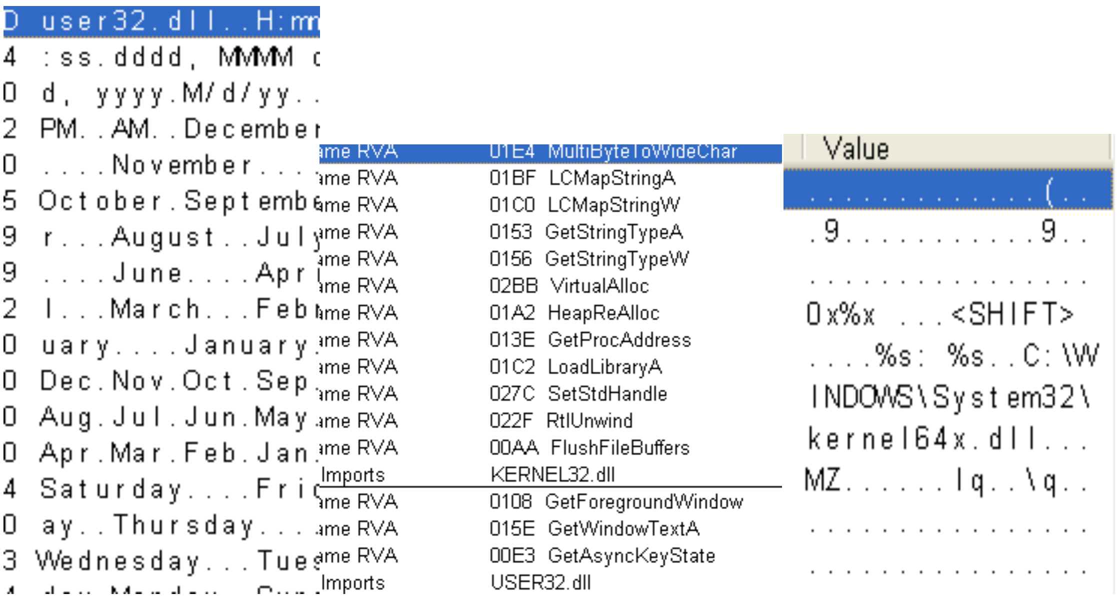


But I haven't found two receipts address at detailed information. I can only see Outlook Express was trying to resolve email address of my own account.

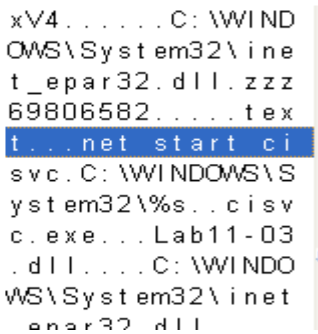
Lab11-03

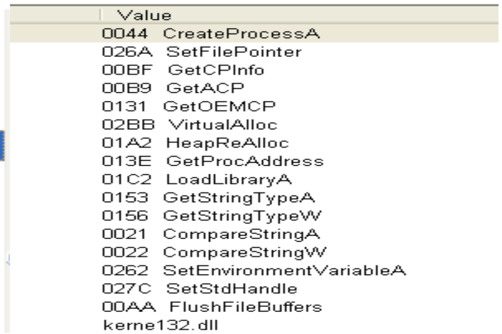
Steps of Processes

First I start analyzing lab11-03 by PEview and String. I think the dll file might be more malicious so I start with dll file. It has an interesting export named zzz69806582, which means I need to pay attention to this export function when I put into IDA Pro. It has imports function like GetWindowText, GetAsyncKeyState, GetForegroundWindow which is located at user32.dll. It also has CreateFile, WriteFile and VirtualAlloc, which indicate the malware might create a file to store the stolen credential information. Therefore I assume it is a key logger. And it can achieve persistency. It has string C:\WINDOWS\System32\Kernel64x.dll which might be the target attacked by malware or store information I typed. When I navigate to the path, I didn't see Kernel64x.dll. I guess it won't show up until the program is running.

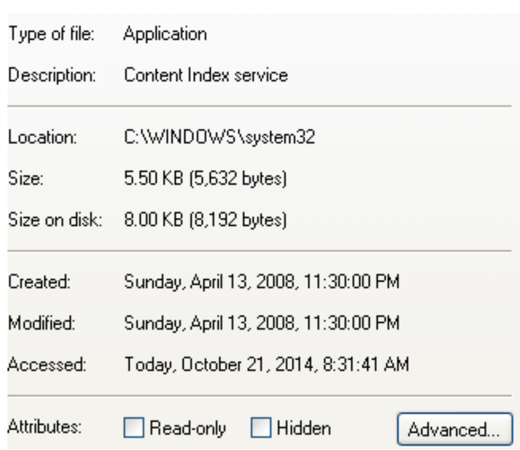


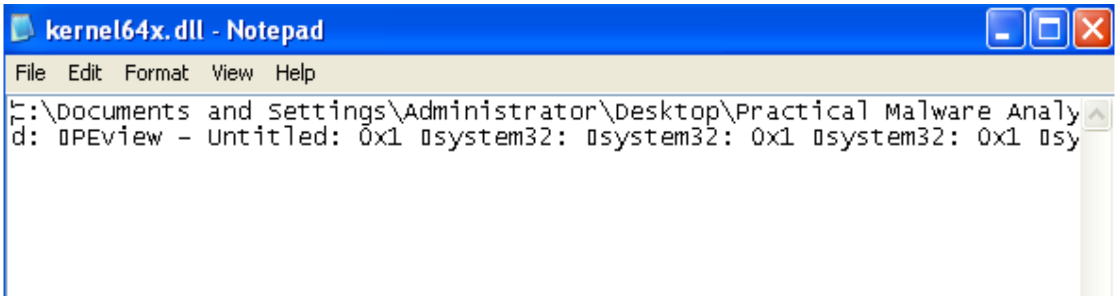
Analyze Lab11-03.exe with statistic tools. It doesn't have export functions but a bunch of import functions GetProcAddress, LoadLibrary and CreateProcess. The executable file might load the malicious dll to current running process. It also creates KERNE123.dll on purpose to be alike with KERNEL32.dll. Check the string at PEview, I notice net start cisvc. Therefore, the executable file will start a service named "cisvc". We will check it by using What's Run. And it also intercepts with Lab11-03.dll and C:WINDOWS\System32\inet_epar32.dll.



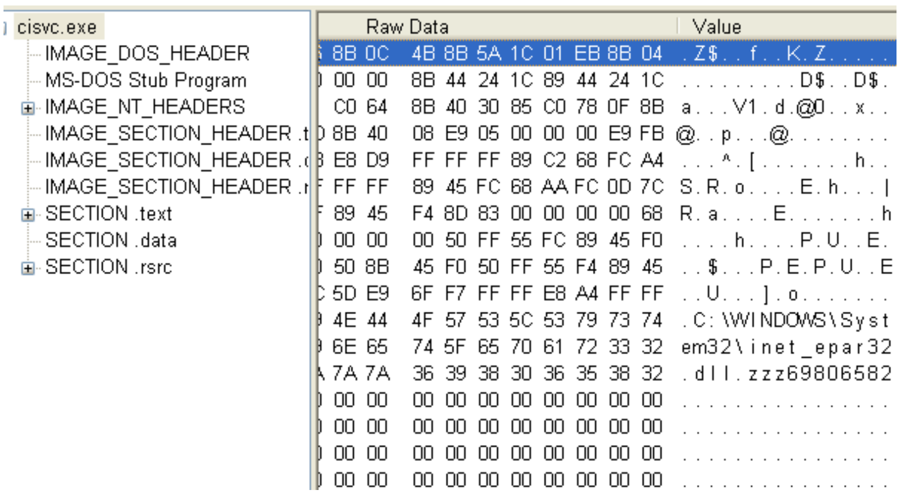


Put the two files into IDA Pro and start detailed analysis. First I analyze the executable file. At the main function, the program calls Copyfile( ) with parameters NewFileName C:\WINDOWS\System32\inet_epar32.dll and ExistingFileName Lab11-03.dll. It creates a new file and copy Lab11-03.dll to it. After the program is running, there is supposed to be a new created file inet_epar32.dll that is identical to Lab11-03.dll. The next function call is sprintf with parameter cisvc.exe and C:\WINDOWS\System32\%s. It formats the string C:\WINDOWS\SYstem32\cisvc.exe and passes it as parameter to sub_401070. Then, the last function call at main function is system (net start cisvc). Now double click on sub_401070 and analyze the details. It create cisvc.exe file by CreateFile 0x401095. Actually cisve.exe is already exists and it is used for monitoring the index server. Therefore, I think the purpose should be modifying the file and mapping it into memory. Because when I ran the malware and check the properties of cisvc.exe, I can see it was accessed a few seconds ago. Also, I copy the original cisvc.exe file so that I can make a comparison later. Moreover, kernel64x.dll is created. Open it with notepad and the contents are hex operands that represent different operations.





Open the infected version of cisvc.exe at PEview and I noticed the program is attached to exports function of inet_epar32.dll. So, cisvc.exe will first load inet_epar32.dll and call export function. I assume the keylogger malicious codes are in export function.



The analyzing process to Lab11-03.exe is not done yet. So I go back to IDA Pro. In order to map cisvc.exe into memory, the program calls GetFileSize( ), CreateFileMapping( ), MapViewOfFile( ) and UnmapViewOfFile( ). MapViewOfFile( ) is aimed to maps a view of a file mapping into the address space of the current running process. UnmapViewOfFile performs opposite function to MapViewOfFile. Those two functions are combined together to modify the file, which replaces WriteFile function call. After mapping view of the file, the program calls sub_401000 that contains IsBadReadPtr at 0x401013 to make sure that current running process has read access to the specified range of memory. If the return value is 0, then access successfully and the program will start serials of memory mapping operations starting from loc_40112F until the end of sub_401070. Among the processes of memory mapping, one location has different codes with others that write and read memory. At loc_409030, the program shows operations at data section, which is unusual. Double click on the location, and I notice this is the place where the malicious DLL is injected.

```
.data:00409030 ; ---------------------------------------------------------------------------
.data:00409030
.data:00409030 loc_409030:                             ; DATA XREF: sub_401070+19Dîr
.data:00409030                                         ; sub_401070+1FFîw ...
.data:00409030                 push    ebp
.data:00409031
.data:00409031 loc_409031:                             ; DATA XREF: sub_401070+1ADîr
.data:00409031                                         ; sub_401070+1BDîr
.data:00409031                 mov     ebp, esp
.data:00409033
.data:00409033 loc_409033:                             ; DATA XREF: sub_401070+1CDîr
.data:00409033                 sub     esp, 40h
.data:00409039                 jmp     loc_409134
.data:0040903E
.data:0040903E ; =============== S U B R O U T I N E =======================================
```

esp is subtracted from 0x40 and then the jump location is injected immediately. Double click on the jump location and see the following injection command.

```
.data:00409139                 db  43h ; C
.data:0040913A                 db  3Ah ; :
.data:0040913B                 db  5Ch ; \
.data:0040913C                 db  57h ; W
.data:0040913D                 db  49h ; I
.data:0040913E                 db  4Eh ; N
.data:0040913F                 db  44h ; D
.data:00409140                 db  4Fh ; O
.data:00409141                 db  57h ; W
.data:00409142                 db  53h ; S
.data:00409143                 db  5Ch ; \
.data:00409144                 db  53h ; S
.data:00409145                 db  79h ; y
.data:00409146                 db  73h ; s
.data:00409147                 db  74h ; t
.data:00409148                 db  65h ; e
.data:00409149                 db  6Dh ; m
.data:0040914A                 db  33h ; 3
.data:0040914B                 db  32h ; 2
.data:0040914C                 db  5Ch ; \
.data:0040914D                 db  69h ; i
.data:0040914E                 db  6Eh ; n
.data:0040914F                 db  65h ; e
.data:00409150                 db  74h ; t
.data:00409151                 db  5Fh ; _
.data:00409152                 db  65h ; e
.data:00409153                 db  70h ; p
.data:00409154                 db  61h ; a
.data:00409155                 db  72h ; r
.data:00409156                 db  33h ; 3
.data:00409157                 db  32h ; 2
.data:00409158                 db  2Eh ; .
.data:00409159                 db  64h ; d
.data:0040915A                 db  6Ch ; l
.data:0040915B                 db  6Ch ; l
.data:0040915C                 db    0
.data:0040915D                 db  7Ah ; z
.data:0040915E                 db  7Ah ; z
.data:0040915F                 db  7Ah ; z
.data:00409160                 db  36h ; 6
.data:00409161                 db  39h ; 9
.data:00409162                 db  38h ; 8
.data:00409163                 db  30h ; 0
```

Recall what we analyze about the malicious version of cisvc.exe. It has string C:\WINDOWS\System32\inet_epar32.dll zzz69806582. Therefore, when cisvc.exe is running, it will load inet_epar32.dll that is same as Lab11-03.dll and call export functions at zzz69806582.
The injection command is stored at data section, which implies that the malware performs a shellcode injection attack.

So far we can conclude that the executable file performs file mapping function and shellcode injection that load keylogger DLL file to the current running process. To see how does keylogger work, we still need to analyze lab11-03.dll, or inet_epar32.dll.

Check DLLMain entry point first because the malware might achieve persistency by performing trojanized binaries at entry point. In this case, the malware makes use of trojanized binaries to achieve persistency. From 0x10001574 to 0x1000157A, it compares fdwReason with PROCESS_ATTACH. Therefore, each current running program needs cisvc.exe will make a jump to the 0x40 where inet_epar32.dll is loaded at cisvc.exe. Now double click on export function zzz69806582 at 0x10001540. The only function call here is CreateThread( ) with six parameters. Only one parameter lpStartAddress takes value. It is a pointer pointing to the address of StartAddress function. So I double click on it. At code area of StartAddress, the program open a mutex named "MZ" at 0x10001481. If the program is running right now, the program will not create another piece otherwise it will create a mutex named "MZ" at 0x100014A6.

```
loc_100014BD:                           ; CODE XREF: StartAddress+A9↑j
                push    0               ; hTemplateFile
                push    80h             ; dwFlagsAndAttributes
                push    4               ; dwCreationDisposition
                push    0               ; lpSecurityAttributes
                push    1               ; dwShareMode
                push    0C0000000h      ; dwDesiredAccess
                push    offset FileName ; "C:\\WINDOWS\\System32\\kernel64x.dll"
                call    ds:CreateFileA
                mov     [ebp+hFile], eax
                cmp     [ebp+hFile], 0
                jnz     short loc_100014EB
                jmp     short loc_10001530
; ---------------------------------------------------------------------------

loc_100014EB:                           ; CODE XREF: StartAddress+D7↑j
                push    2               ; dwMoveMethod
                push    0               ; lpDistanceToMoveHigh
                push    0               ; lDistanceToMove
                mov     eax, [ebp+hFile]
                push    eax             ; hFile
                call    ds:SetFilePointer
                mov     ecx, [ebp+hFile]
                mov     [ebp+var_4], ecx
                lea     edx, [ebp+var_810]
                push    edx
                call    sub_10001380
                add     esp, 4
                mov     eax, [ebp+hFile]
                push    eax             ; hObject
```

After the mutex is created, we can see three main function calls here. The first one is CreateFile. The program creates Kernel64x.dll under C:\WINDOWS\System32 and passes the address of the file to next function call SetFilePointer. The pointer of kernel64x.dll is passed to third function call sub_10001380. According to above analysis, kernel64x.dll is used to store the stolen key status. Therefore the pointer and sub_10001380 is supposed to write file and perform keylogger function. Double click on sub_10001380, it immediately calls another function sub_10001030 where the malware record the key status. I rename sub_10001030 as keylogger and rename the only one function call within it as Current_Window. The Current_Window function takes a lot of parameter in order to call GetForegroundWindow and GetWindowText. The former one is used to identify the foreground window—the one that has focus—which tells the keylogger which application is being used for keyboard entry. The later one is used to copy the text of the specified window's title bar. The results will be returned to Current_Window function so that the program can keep going to check the key status.

The recording process starts at 0x10001127. GetAsyncKeyState( ) has been called many times. Each time GetAsyncKeyState( ) is called, the program will check if "SHIFT" button is pressed or not at loc_10001180. Consider it as a loop function. If all keys have been checked, the program will format the pressed key as operands in hex digits "0x%x" format. Then it returns the result back to keylogger( ) and format the result as "%s: %s\n". Finally the program calls Writefile( ) to record key status, operations and current window into kernel64x.dll. After that, the program can check next window focused by users.

Each time keylogger is implemented the program will sleep for 10 milliseconds and starts over again. Check the black line as following. The loop is clearly indicated.

```
.text:1000138A loc_1000138A:                           ; CODE XREF: sub_1(
.text:1000138A                 mov     eax, [ebp+arg_0]
.text:1000138D                 push    eax
.text:1000138E                 call    keylogger
.text:10001393                 add     esp, 4
.text:10001396                 test    eax, eax
.text:10001398                 jnz     short loc_10001404
.text:1000139A                 mov     ecx, [ebp+arg_0]
.text:1000139D                 cmp     dword ptr [ecx], 0
.text:100013A0                 jz      short loc_100013FA
.text:100013A2                 mov     edx, [ebp+arg_0]
.text:100013A5                 add     edx, 408h
.text:100013AB                 push    edx
.text:100013AC                 mov     eax, [ebp+arg_0]
.text:100013AF                 add     eax, 4
.text:100013B2                 push    eax
.text:100013B3                 push    offset aSS      ; "%s: %s\n"
.text:100013B8                 lea     ecx, [ebp+Buffer]
.text:100013BE                 push    ecx             ; char *
.text:100013BF                 call    _sprintf
.text:100013C4                 add     esp, 10h
.text:100013C7                 push    0               ; lpOverlapped
.text:100013C9                 lea     edx, [ebp+NumberOfBytesWritten]
.text:100013CF                 push    edx             ; lpNumberOfBytesWr
.text:100013D0                 lea     edi, [ebp+Buffer]
.text:100013D6                 or      ecx, 0FFFFFFFFh
.text:100013D9                 xor     eax, eax
.text:100013DB                 repne scasb
.text:100013DD                 not     ecx
.text:100013DF                 add     ecx, 0FFFFFFFFh
.text:100013E2                 push    ecx             ; nNumberOfBytesToW
.text:100013E3                 lea     eax, [ebp+Buffer]
.text:100013E9                 push    eax             ; lpBuffer
.text:100013EA                 mov     ecx, [ebp+arg_0]
.text:100013ED                 mov     edx, [ecx+80Ch]
.text:100013F3                 push    edx             ; hFile
.text:100013F4                 call    ds:WriteFile
.text:100013FA
.text:100013FA loc_100013FA:                           ; CODE XREF: sub_1(
.text:100013FA                 push    0Ah             ; dwMilliseconds
.text:100013FC                 call    ds:Sleep
.text:10001402                 jmp     short loc_1000138A
00001393 10001393: sub_10001380+13
```

Issues of Problems

Conclusion

Reviewed Questions

1. What interesting analysis leads can you discover using basic static analysis?

C:\WINDOWS\System32\inet_epar32.dll

C:\WINDOWS\System32\kernel64x.dll

zzz69806582

C:\WINDOWS\System32\Lab11-03.dll
net start csivc
import functions like GetAsyncKeyState

2. What happens when you run this malware?

A command line window prompted and displayed that service starts, which is too quick to take screenshot. Moreover, kernel64x.dll is created under system directory; mutex MZ is created; cisvc.exe is modified; inet_epar32.dll is created under system directory.

3. How does Lab11-03.exe persistently install Lab11-03.dll?

Trojanized Binaries at the entry point of Lab11-03.dll so that the shellcode could be injected.

4. Which Windows system file does the malware infect?

It infects cisvc.exe and each program requiring cisvc.exe.

5. What does Lab11-03.dll do?

It performs keylogger function to record the keyboard status and retrieve which program is being used by user.

6. Where does the malware store the data it collects?

At new created file name kernel64x.dll at C:\WINDOWS\System32.