LAB 7

WinDBG Analysis

Under the direction of
Dr. Samuel Liles

Table of Contents

Abstract

This lab is focused on Malware Analysis. The lab is going to use tools and application to do Static/Dynamic analysis of the malware while being isolated from the internet. The Practical Lab 10.1 to Lab 10.3 will be carried out to answer the questions provided.

The Computer Anti-virus was disabled as part of the instructions to enable the download and extract of the files being used. This lab is intended to lay grounds for further labs in the course.

Keywords: Digital Investigation, Forensic Evidence, Malware Analysis.

Lab 7 WinDBG Analysis

Steps of the process

Preparing the LAB

The Computer was rebooted, anti-virus was disabled, and the appropriate files were downloaded. Different Images of VM were installed. Installation of different windows environment such as XP, 7 and 8.1. Programs needed have been downloaded and snapshots of the process have been taken.

LAB 10-1, 10-3

Applications & Tools

The following applications are used to forensically examine the files. The following descriptions have been captured from the developer's website and manuals.

PEiD," is an intuitive application that relies on its user-friendly interface to detect packers, cryptors and compilers found in PE executable files – its detection rate is higher than that of other similar tools since the app packs more than 600 different signatures in PE files" (Gröbert, 2010).

Resource Hacker,"is a freeware utility to view, modify, rename, add, delete and extract resources in 32bit & 64bit Windows executables and resource files (*.res). It incorporates an internal resource script compiler and decompiler and works on all (Win95 - Win7) Windows operating systems" (Johnson, 2011).

PE Explorer"provides powerful tools for disassembly and inspection of unknown binaries, editing the properties of 32-bit executable files and customizing and translating their resources. Use this product to do reverse engineering, analyze the procedures and libraries an executable uses." (Heaventools Software, 2009).

Process Monitor is an advanced monitoring tool for Windows that shows real-time file system, Registry and process/thread activity. It combines the features of two legacy Sysinternals utilities, Filemon and Regmon, and adds an extensive list of enhancements including rich and non-destructive filtering, comprehensive event properties such session IDs and user names, reliable process information, full thread stacks with integrated symbol support for each operation, simultaneous logging to a file, and much more. Its uniquely powerful features will make Process Monitor a core utility in your system troubleshooting and malware hunting toolkit (Russinovich & Cogswell, 2014).

ApateDNS, is a tool for controlling DNS responses though an easy to use GUI. As a phony DNS server, ApateDNS spoofs DNS responses to a user-specified IP address by listening on UDP port 53 on the local machine. It responds to DNS requests with the response set to any IP address you specify. The tool logs and timestamps any DNS request it receives. You may specify a number of non-existent domain (NXDOMAIN) responses to send before returning a valid response. ApateDNS also automatically sets the local DNS to localhost. By default, it will use either the set DNS or default gateway settings as an IP address to use for DNS responses. Upon exiting the tool, it sets back the original local DNS settings (Davis, 2011).

Regshot, is a small, free and open-source registry compare utility that allows you to quickly take a snapshot of your registry and then compare it with a second one - done after doing system changes or installing a new software product. The changes report can be produced in text or HTML format and contains a list of all modifications that have taken place between the two snapshots. In addition, you can also specify folders (with subfolders) to be scanned for changes as well (Regshot Team, 2013).

IDA is the Interactive DisAssembler: the world's smartest and most feature-full disassembler, which many software security specialists are familiar with (Hex-Rays SA, 2014).

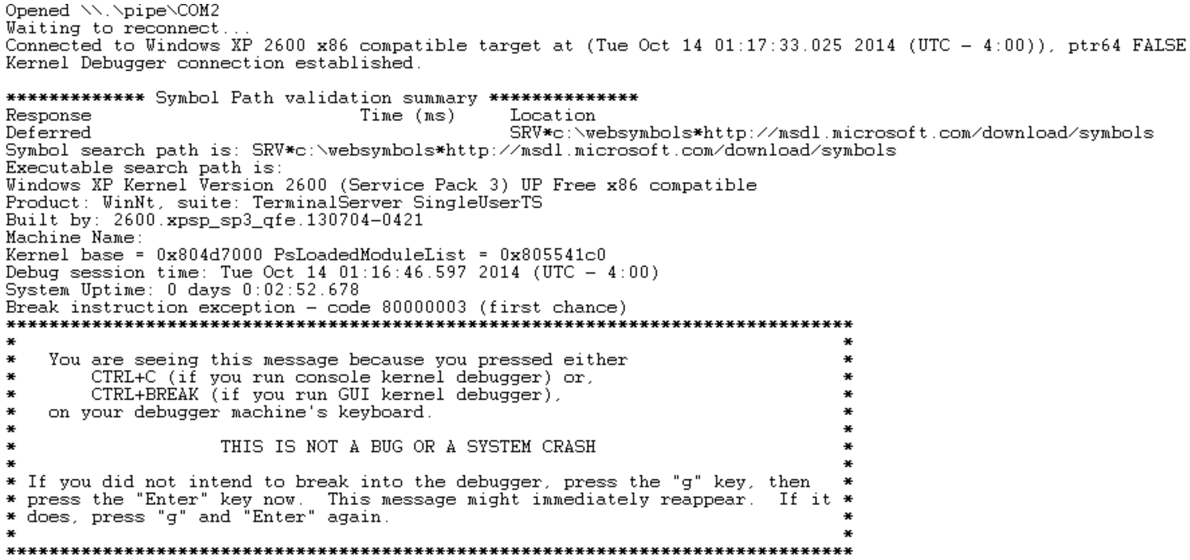
OllyDbg, is a 32-bit assembler level analyzing debugger for Microsoft® Windows®. Emphasis on **binary code analysis** makes it particularly useful in cases where source is unavailable (Yuschuk, 2014).

WinDbg, provides full source-level debugging for the Windows kernel, kernel-mode drivers, and system services, as well as user-mode applications and drivers (Microsoft, 2014).

Show Drivers is the free command-line tool to list Drivers running on your Windows system (SecurityXploded, 2013).

Issues or problems

VM Station cannot run more than one XP mode VM image therefore Kernel Dbg pip needs to be connected to another windows system or a complete XP system image. I will attempt using windows 7 instead. After that I tried Using Windows 7 VM to connect with the XP and Windows 7 license key caused issues so I could not do that. Finally I decided to use the host PC to connect with the VM Kernel and get the info via the PIPE. I had problems connecting. WinDbg was stuck with waiting for connection all the time. I tried different port settings until it worked. At that point I stopped WinDbg to add the Symbol update link SRV*c:\websymbols*http://msdl.microsoft.com/download/symbols. Restarted the connection in order to update it. Then things started working normally.



```
Opened \\.\pipe\COM2
Waiting to reconnect...
Connected to Windows XP 2600 x86 compatible target at (Tue Oct 14 01:17:33.025 2014 (UTC - 4:00)), ptr64 FALSE
Kernel Debugger connection established.

************* Symbol Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       SRV*c:\websymbols*http://msdl.microsoft.com/download/symbols
Symbol search path is: SRV*c:\websymbols*http://msdl.microsoft.com/download/symbols
Executable search path is:
Windows XP Kernel Version 2600 (Service Pack 3) UP Free x86 compatible
Product: WinNt, suite: TerminalServer SingleUserTS
Built by: 2600.xpsp_sp3_qfe.130704-0421
Machine Name:
Kernel base = 0x804d7000 PsLoadedModuleList = 0x805541c0
Debug session time: Tue Oct 14 01:16:46.597 2014 (UTC - 4:00)
System Uptime: 0 days 0:02:52.678
Break instruction exception - code 80000003 (first chance)
*******************************************************************************
*                                                                             *
*   You are seeing this message because you pressed either                    *
*       CTRL+C (if you run console kernel debugger) or,                       *
*       CTRL+BREAK (if you run GUI kernel debugger),                          *
*   on your debugger machine's keyboard.                                      *
*                                                                             *
*                   THIS IS NOT A BUG OR A SYSTEM CRASH                       *
*                                                                             *
* If you did not intend to break into the debugger, press the "g" key, then   *
* press the "Enter" key now.  This message might immediately reappear.  If it *
* does, press "g" and "Enter" again.                                          *
*                                                                             *
*******************************************************************************
```

Conclusions

The Lab identified several programs that helps explore the malwares. The tools showed if the files being used are infected or packed. The tools used also showed the resources on the system that is being utilized such as privilege, CPU usage, Network communication.

Case studies

No Case studies was given with this lab.

Review questions

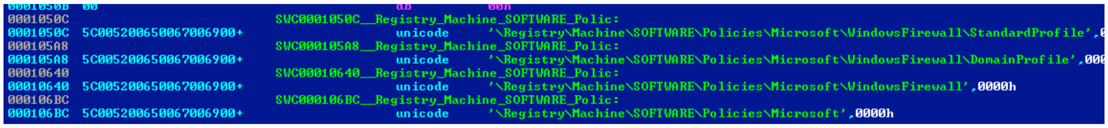
Lab 10-1

<table>
<tr><td>Answers</td><td>Lab10-01. exe, Lab10-01.sys</td></tr>
<tr><td>1</td><td>

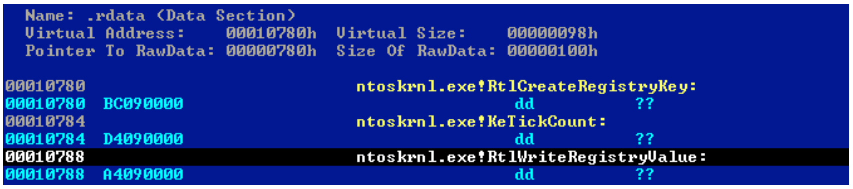
In IDA Pro, with the EXE file we see that createserviceA is one of the functions available which means a service will be created and most likely it will be available in the registry. In order to make sure we check the SYS file and we find using the Strings function in PE Explorer, we can see that Registry_Machine_Software_Polic is a string being used. Furthermore, in the code we can see different locations that are being used. In Fact the following two function:

ntoskrnl.exe!RtlCreateRegistryKey,

ntoskrnl.exe!RtlWriteRegistryValue:

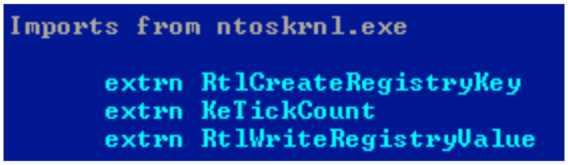


```
0001050B  00                                       db          00h
0001050C                             SWC0001050C__Registry_Machine_SOFTWARE_Polic:
0001050C  5C00520065006700690 0+                   unicode     '\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFirewall\StandardProfile',0
000105A8                             SWC000105A8__Registry_Machine_SOFTWARE_Polic:
000105A8  5C005200650067006900+                    unicode     '\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFirewall\DomainProfile',000
00010640                             SWC00010640__Registry_Machine_SOFTWARE_Polic:
00010640  5C005200650067006900+                    unicode     '\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFirewall',0000h
000106BC                             SWC000106BC__Registry_Machine_SOFTWARE_Polic:
000106BC  5C005200650067006900+                    unicode     '\Registry\Machine\SOFTWARE\Policies\Microsoft',0000h
```



```
  Name: .rdata (Data Section)
  Virtual Address:    00010780h  Virtual Size:    00000098h
  Pointer To RawData: 00000780h  Size Of RawData: 00000100h

00010780                        ntoskrnl.exe!RtlCreateRegistryKey:
00010780  BC090000                          dd            ??
00010784                        ntoskrnl.exe!KeTickCount:
00010784  D4090000                          dd            ??
00010788                        ntoskrnl.exe!RtlWriteRegistryValue:
00010788  A4090000                          dd            ??
```

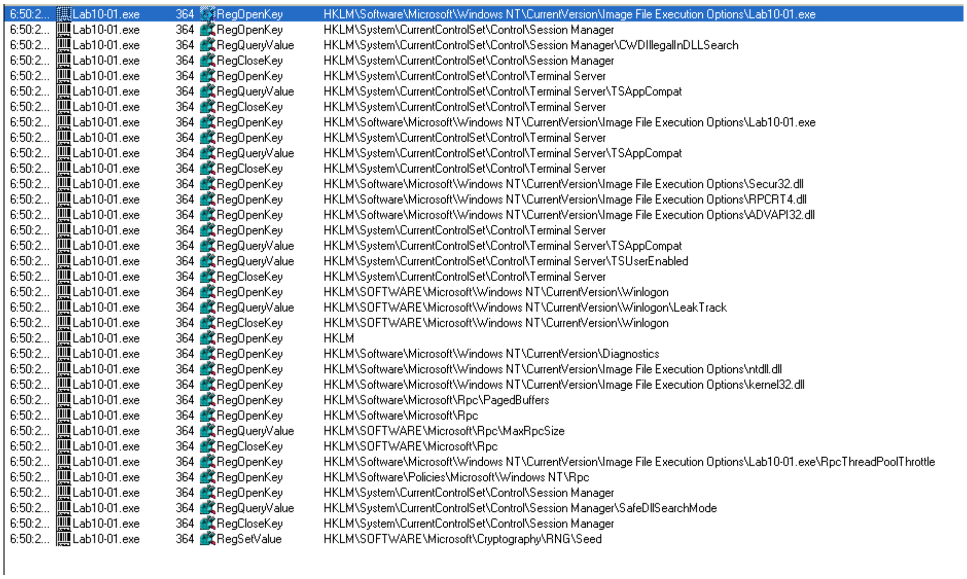
Finally we can see that those registry functions have been imported from ntoskrnl.exe and used in the SYS file.



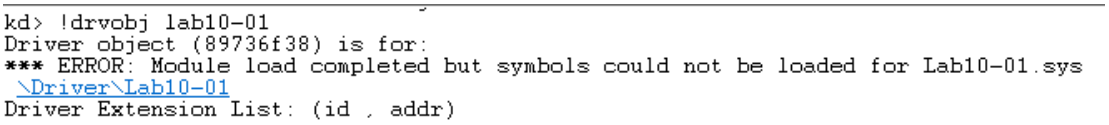
```
Imports from ntoskrnl.exe

        extrn RtlCreateRegistryKey
        extrn KeTickCount
        extrn RtlWriteRegistryValue
```

Procmon will not detect changes made via the kernel however regshot will show changes that has been done regardless of what privilege it was used to creates it. If it is

</td></tr>
</table>

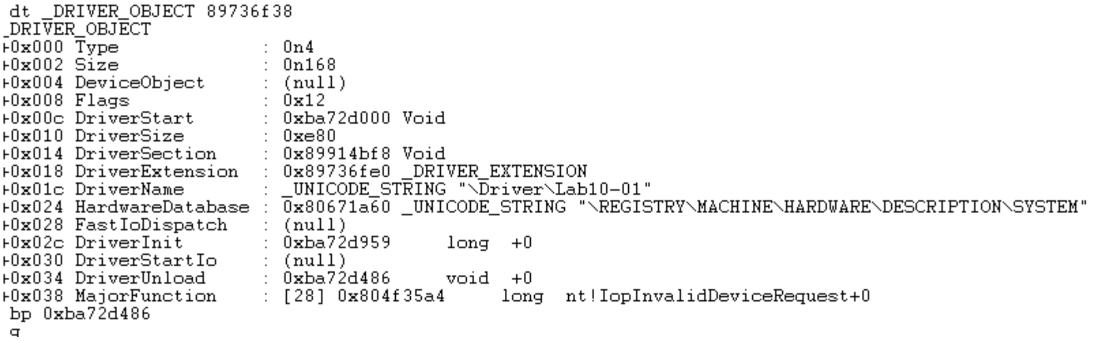
	there it will show up. Using Regshot we find that the following 15 keys have been added HKLM\SOFTWARE\Policies\Microsoft\WindowsFirewall HKLM\SOFTWARE\Policies\Microsoft\WindowsFirewall\DomainProfile HKLM\SOFTWARE\Policies\Microsoft\WindowsFirewall\StandardProfile HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_LAB10-01 HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_LAB10-01\0000 HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_LAB10-01\0000\Control HKLM\SYSTEM\ControlSet001\Services\Lab10-01 HKLM\SYSTEM\ControlSet001\Services\Lab10-01\Security HKLM\SYSTEM\ControlSet001\Services\Lab10-01\Enum HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_LAB10-01 HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_LAB10-01\0000 HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_LAB10-01\0000\Control HKLM\SYSTEM\CurrentControlSet\Services\Lab10-01 HKLM\SYSTEM\CurrentControlSet\Services\Lab10-01\Security HKLM\SYSTEM\CurrentControlSet\Services\Lab10-01\Enum More than 40 values have been added, and 22 have been changed. which means the malware have affected more than 62 records in the registery by simply running it. Now going back to procmon we can confirm our findings since it only detected some of the 62 changes that has been done.  6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Lab10-01.exe 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\System\CurrentControlSet\Control\Session Manager 6:50:2... Lab10-01.exe 364 RegQueryValue HKLM\System\CurrentControlSet\Control\Session Manager\CWDIllegalInDLLSearch 6:50:2... Lab10-01.exe 364 RegCloseKey HKLM\System\CurrentControlSet\Control\Session Manager 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\System\CurrentControlSet\Control\Terminal Server 6:50:2... Lab10-01.exe 364 RegQueryValue HKLM\System\CurrentControlSet\Control\Terminal Server\TSAppCompat 6:50:2... Lab10-01.exe 364 RegCloseKey HKLM\System\CurrentControlSet\Control\Terminal Server 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Lab10-01.exe 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\System\CurrentControlSet\Control\Terminal Server 6:50:2... Lab10-01.exe 364 RegQueryValue HKLM\System\CurrentControlSet\Control\Terminal Server\TSAppCompat 6:50:2... Lab10-01.exe 364 RegCloseKey HKLM\System\CurrentControlSet\Control\Terminal Server 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Secur32.dll 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\RPCRT4.dll 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\ADVAPI32.dll 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\System\CurrentControlSet\Control\Terminal Server 6:50:2... Lab10-01.exe 364 RegQueryValue HKLM\System\CurrentControlSet\Control\Terminal Server\TSAppCompat 6:50:2... Lab10-01.exe 364 RegQueryValue HKLM\System\CurrentControlSet\Control\Terminal Server\TSUserEnabled 6:50:2... Lab10-01.exe 364 RegCloseKey HKLM\System\CurrentControlSet\Control\Terminal Server 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon 6:50:2... Lab10-01.exe 364 RegQueryValue HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\LeakTrack 6:50:2... Lab10-01.exe 364 RegCloseKey HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\Software\Microsoft\Windows NT\CurrentVersion\Diagnostics 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\ntdll.dll 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\kernel32.dll 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\Software\Microsoft\Rpc\PagedBuffers 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\Software\Microsoft\Rpc 6:50:2... Lab10-01.exe 364 RegQueryValue HKLM\SOFTWARE\Microsoft\Rpc\MaxRpcSize 6:50:2... Lab10-01.exe 364 RegCloseKey HKLM\SOFTWARE\Microsoft\Rpc 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Lab10-01.exe\RpcThreadPoolThrottle 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\Software\Policies\Microsoft\Windows NT\Rpc 6:50:2... Lab10-01.exe 364 RegOpenKey HKLM\System\CurrentControlSet\Control\Session Manager 6:50:2... Lab10-01.exe 364 RegQueryValue HKLM\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode 6:50:2... Lab10-01.exe 364 RegCloseKey HKLM\System\CurrentControlSet\Control\Session Manager 6:50:2... Lab10-01.exe 364 RegSetValue HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed
2	In order to break the function we use the Host kernel to control the vm kernel setting points to where the function will be loaded in memory. Using the Disassembly in the

WinDbg. After it stops we go back to the host device and capture its information from memory.



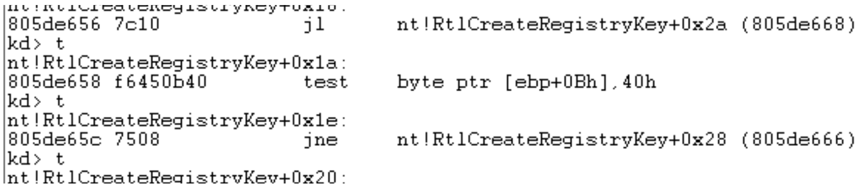
```
kd> !drvobj lab10-01
Driver object (89736f38) is for:
*** ERROR: Module load completed but symbols could not be loaded for Lab10-01.sys
 \Driver\Lab10-01
Driver Extension List: (id , addr)
```

Now this is important because that gives us a way to get it in memory via the 89736f38 value.



```
 dt _DRIVER_OBJECT 89736f38
_DRIVER_OBJECT
+0x000 Type             : 0n4
+0x002 Size             : 0n168
+0x004 DeviceObject     : (null)
+0x008 Flags            : 0x12
+0x00c DriverStart      : 0xba72d000 Void
+0x010 DriverSize       : 0xe80
+0x014 DriverSection    : 0x89914bf8 Void
+0x018 DriverExtension  : 0x89736fe0 _DRIVER_EXTENSION
+0x01c DriverName       : _UNICODE_STRING "\Driver\Lab10-01"
+0x024 HardwareDatabase : 0x80671a60 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
+0x028 FastIoDispatch   : (null)
+0x02c DriverInit       : 0xba72d959     long  +0
+0x030 DriverStartIo    : (null)
+0x034 DriverUnload     : 0xba72d486     void  +0
+0x038 MajorFunction    : [28] 0x804f35a4     long  nt!IopInvalidDeviceRequest+0
 bp 0xba72d486
 g
```

Using that we can get more detailed information about the drive and its settings. This will give us the command line in which the driver will be unloaded and using that as a break point we can step over the code to know more about its functionality. Stepping over the code we find that it creates 3 registry keys and change the values of other two and all are related to firewall settings.



```
nt!RtlCreateRegistryKey+0x18:
805de656 7c10            jl      nt!RtlCreateRegistryKey+0x2a (805de668)
kd> t
nt!RtlCreateRegistryKey+0x1a:
805de658 f6450b40        test    byte ptr [ebp+0Bh],40h
kd> t
nt!RtlCreateRegistryKey+0x1e:
805de65c 7508            jne     nt!RtlCreateRegistryKey+0x28 (805de666)
kd> t
nt!RtlCreateRegistryKey+0x20:
```

3

From the Strings Found in PE Explorer, As well as the Regshot values we see lots of firewall keys from the registry is being changed and added. Which leads me to believe that the file is somehow trying to modify those values in its favor (disabling the firewall) without being detected by the security software since changes are being done via the kernel level.

Lab 10-2

<table>
<tr><td>Answers</td><td>Lab10-02. exe</td></tr>
<tr><td>1</td><td>

Going over the strings using IDA Pro we find the following that is of interest :

C:\Windows\System32\Mlwx486.sys

486 WS Driver

Failed to create service.

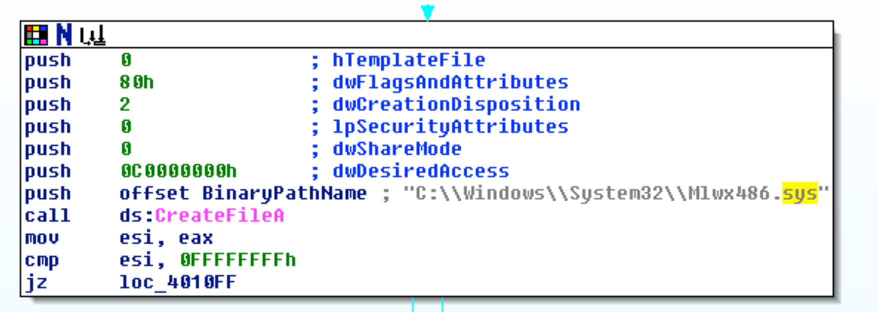
Failed to start service.

KERNEL32.dll

GetCommandLineA

Also, included with the above are several functions that creates, edits, deletes, and moves files and services.

Going over the Disassembly code we can confirm that the Mlwx486 is being created.

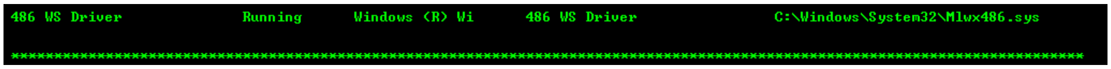


```
push    0                   ; hTemplateFile
push    80h                 ; dwFlagsAndAttributes
push    2                   ; dwCreationDisposition
push    0                   ; lpSecurityAttributes
push    0                   ; dwShareMode
push    0C0000000h          ; dwDesiredAccess
push    offset BinaryPathName ; "C:\\Windows\\System32\\Mlwx486.sys"
call    ds:CreateFileA
mov     esi, eax
cmp     esi, 0FFFFFFFFh
jz      loc_4010FF
```

</td></tr>
<tr><td>2</td><td>

Since we found both the KERNEL32.dll as well "486 WS Driver" we can check if it is running or no by using the Show Drivers command line application. Using the following command : ShowDrivers.exe -t

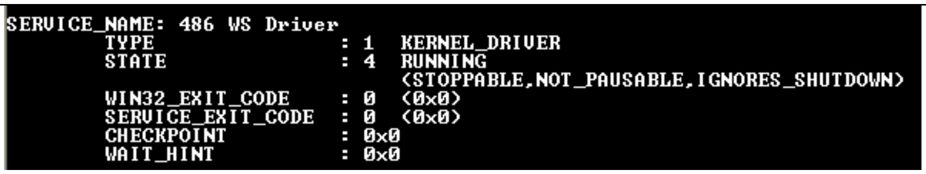
It will display Non Microsoft Drivers from which we find 486 WS Driver Running



```
486 WS Driver            Running      Windows (R) Wi      486 WS Driver              C:\Windows\System32\Mlwx486.sys

**************************************************************************************************************
```

Also when using the system command : sc query "486 WS Driver" we find out that its running as a kernel Driver

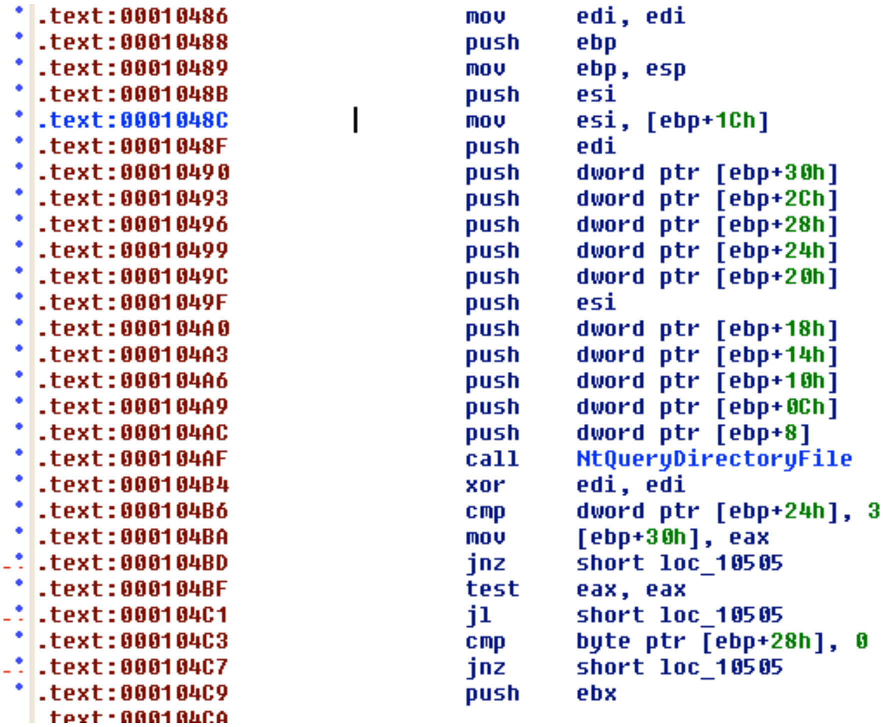
</td></tr>
</table>



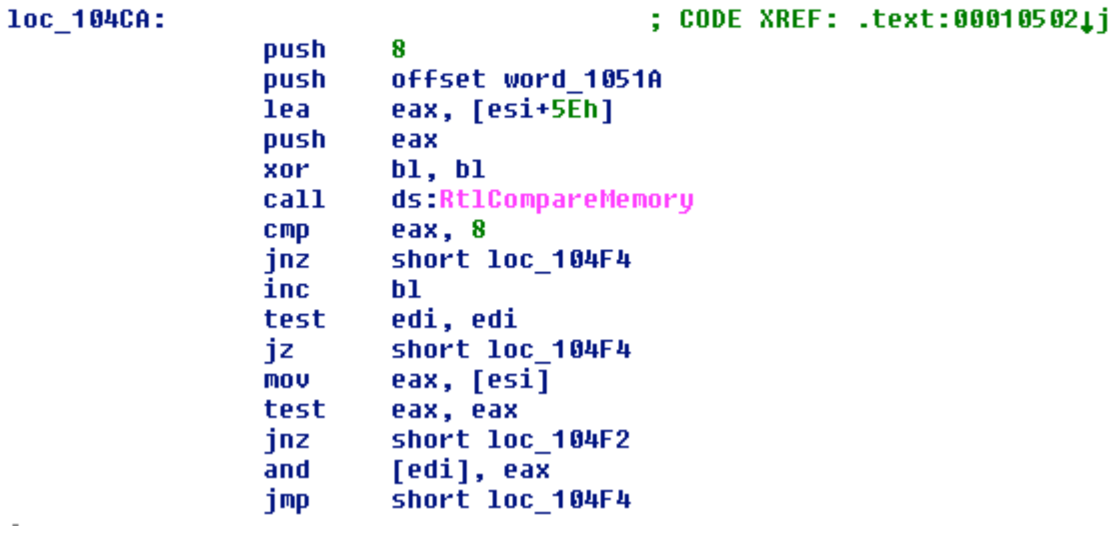
```
SERVICE_NAME: 486 WS Driver
        TYPE               : 1  KERNEL_DRIVER
        STATE              : 4  RUNNING
                                (STOPPABLE,NOT_PAUSABLE,IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

3

The Malware creates a driver that manipulates the system in hiding all files that starts with Mlwx. This is a very nice function since the attacker can use that to temporarily store information or aggregate data before sending it over the network when an opportunity is available. Going into IDA Pro and loading C:\Windows\System32\Mlwx486.sys we find that the file contains one single function that the user went to great lengths to hide it, NtQueryDirectoryFile. Which means that the function is very important to the functionality of the malware. Going over the code of the function we find that it does compare between strings and pointers and based on the value the flow of the program changes. I could narrow it down to Loc_10505 & Loc_104F4 however I could not really find what the change is or how it works to hide the file based on it the first four values Mlwx.



```
.text:00010486                 mov     edi, edi
.text:00010488                 push    ebp
.text:00010489                 mov     ebp, esp
.text:0001048B                 push    esi
.text:0001048C                 mov     esi, [ebp+1Ch]
.text:0001048F                 push    edi
.text:00010490                 push    dword ptr [ebp+30h]
.text:00010493                 push    dword ptr [ebp+2Ch]
.text:00010496                 push    dword ptr [ebp+28h]
.text:00010499                 push    dword ptr [ebp+24h]
.text:0001049C                 push    dword ptr [ebp+20h]
.text:0001049F                 push    esi
.text:000104A0                 push    dword ptr [ebp+18h]
.text:000104A3                 push    dword ptr [ebp+14h]
.text:000104A6                 push    dword ptr [ebp+10h]
.text:000104A9                 push    dword ptr [ebp+0Ch]
.text:000104AC                 push    dword ptr [ebp+8]
.text:000104AF                 call    NtQueryDirectoryFile
.text:000104B4                 xor     edi, edi
.text:000104B6                 cmp     dword ptr [ebp+24h], 3
.text:000104BA                 mov     [ebp+30h], eax
.text:000104BD                 jnz     short loc_10505
.text:000104BF                 test    eax, eax
.text:000104C1                 jl      short loc_10505
.text:000104C3                 cmp     byte ptr [ebp+28h], 0
.text:000104C7                 jnz     short loc_10505
.text:000104C9                 push    ebx
.text:000104CA
```

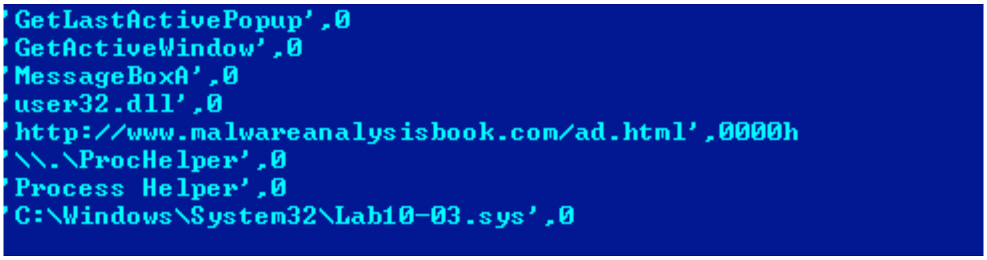


```
loc_104CA:                              ; CODE XREF: .text:00010502↓j
                push    8
                push    offset word_1051A
                lea     eax, [esi+5Eh]
                push    eax
                xor     bl, bl
                call    ds:RtlCompareMemory
                cmp     eax, 8
                jnz     short loc_104F4
                inc     bl
                test    edi, edi
                jz      short loc_104F4
                mov     eax, [esi]
                test    eax, eax
                jnz     short loc_104F2
                and     [edi], eax
                jmp     short loc_104F4
```

Lab 10-3

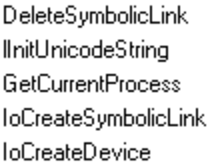
Answers	Lab10-03.exe; Lab10-03.sys

1 In order to know what the program does we statically analyze the malware. Using PE Explorer to view the strings we find the following:



```
'GetLastActivePopup',0
'GetActiveWindow',0
'MessageBoxA',0
'user32.dll',0
'http://www.malwareanalysisbook.com/ad.html',0000h
'\\.\ProcHelper',0
'Process Helper',0
'C:\Windows\System32\Lab10-03.sys',0
```

We can see A link to a website with a webpage titled ad.html, we also see GetLastActivePopup, we also see MessageBoxA, as well as a windows directory file. From that we can safely assume that this malware will Popup windows with a MassageBox periodically with information from the website link. All this is probably tied up to some trick in the system file stored in system32. Moving forward we check the libraries and functions used by the program in the System File. Running IDA Pro we find the following

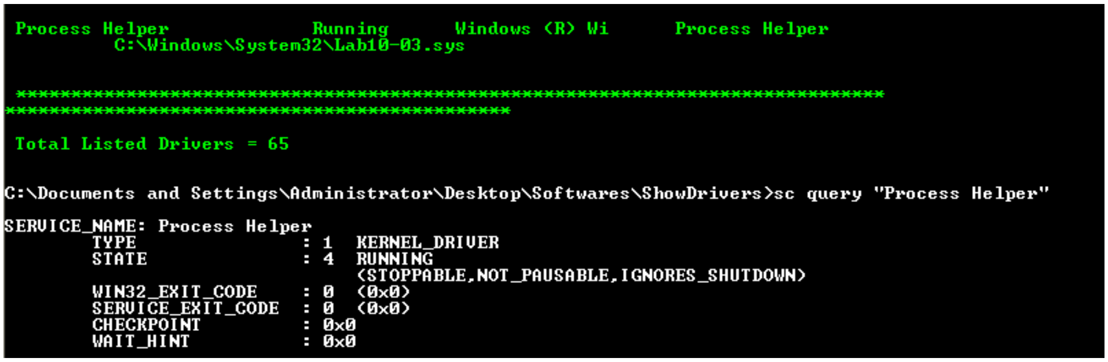


```
DeleteSymbolicLink
IInitUnicodeString
GetCurrentProcess
IoCreateSymbolicLink
IoCreateDevice
```

Reading more about SymbolicLink we learn that it's a simple object pointer that can be used to point to other things including files, objects, and ets. However, for a malware function with Ad intentions it's interesting to find out what did the attacker use it for. Going over the executable in IDA Pro we see that the program creates a service, then creates a file and does something with the IO device, Then attempts to connect to the website, get the information and execute something before going to sleep. The execution is most likely to be the popup AD. Going over the System file shows us again symbolic

<table>
<tr>
<td></td>
<td>
links that are being created, modified and deleted and it is tied up with Inputs and Outputs of the system.

After running the Program we check regshot to find that lots of values have been added under the name Process Helper. Using Show Drivers we find the following Process Helper running as a Driver. After getting that Driver name we use the sc Query to find more information about it. Which showed us that the malware is running on the kernel level loaded in memory.



```
 Process Helper             Running      Windows (R) Wi      Process Helper
          C:\Windows\System32\Lab10-03.sys

 ********************************************************************************
***********************************************

 Total Listed Drivers = 65

C:\Documents and Settings\Administrator\Desktop\Softwares\ShowDrivers>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
```
</td>
</tr>
<tr>
<td>2</td>
<td>Easy way is to stop the service sc stop "Process Helper" Then delete the files and restart the system. Or use an older image of the system to restore. Since its using kernel level Nothing can be done to remove registry values. Any other files and indicators being deleted won’t stop the process running in memory.</td>
</tr>
<tr>
<td>3</td>
<td>It uses the Symbolic links to manipulate the process list and hide itself. As discussed earlier in the first part of this question.</td>
</tr>
</table>

References

Davis, S. (2011, October). *ApateDNS*. Retrieved from https://www.mandiant.com/blog/research-tool-release-apatedns/

Gröbert, F. (2010, 02 07). *PEiD*. Retrieved 02 18, 2014, from https://code.google.com/p/kerckhoffs/downloads/

Heaventools Software. (2009, 10 14). *Heaventools*. Retrieved from http://heaventools.com/download.htm

Hex-Rays SA. (2014, July). *Freeware Download Page*. Retrieved from https://www.hex-rays.com/index.shtml

Johnson, A. (2011, 09 16). *Resource Hacker*. Retrieved from http://www.angusj.com/resourcehacker/

Microsoft. (2014, na.). *Download Center*. Retrieved from http://www.microsoft.com/en-us/download/confirmation.aspx?id=8279

Regshot Team. (2013, August). *Regshot*. Retrieved from http://sourceforge.net/projects/regshot/

Russinovich, M., & Cogswell, B. (2014, March). *Process Monitor v3.1*. Retrieved from http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx

Yuschuk, O. (2014, Feb). *OllyDbg* . Retrieved from http://www.ollydbg.de/