恶意代码分析与防治技术实验报告

Lab10

学号：2011937　　姓名：姜志凯　　专业：信息安全

一、 实验环境

- Windows10
- Windows xp

二、 实验工具

WinDbg

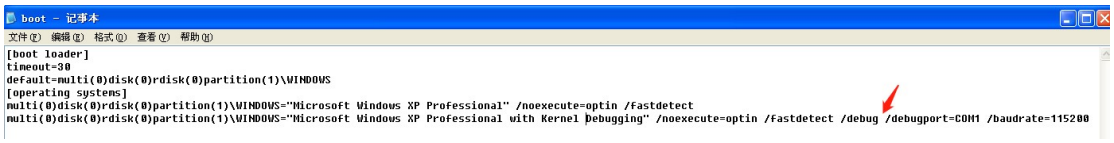
通过修改虚拟机的 boot.ini 文件，增加一个开机入口选项，使能内核调试。，然后

三、 实验内容

配置内核调试环境：

虚拟机：

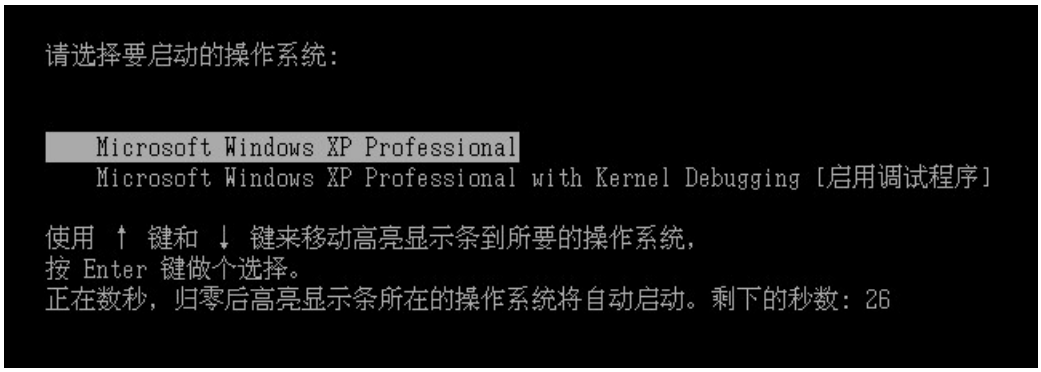
1、xp 虚拟机上：修改 boot.ini 文件，增加开机入口选项，使能内核调试；



这个开机选项只是增加了内核调试功能，OS 也可以正常运行，下次开机时有 30s 的时间来选择这个选项；

2、虚拟机添加一个串行端口，使用命名的管道，命名为\\.\pipe\com_1，选择管道一端是服务器，另一端是应用程序，创建一个与主机的虚拟连接，连接主机上的内核调试器，即 WinDbg；

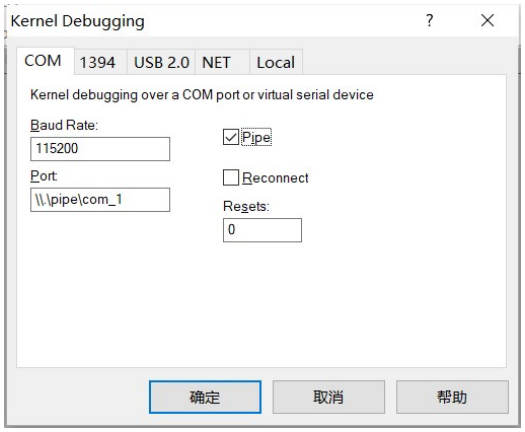
此时，虚拟机配置完毕，启动，选择使能内核调试模式。

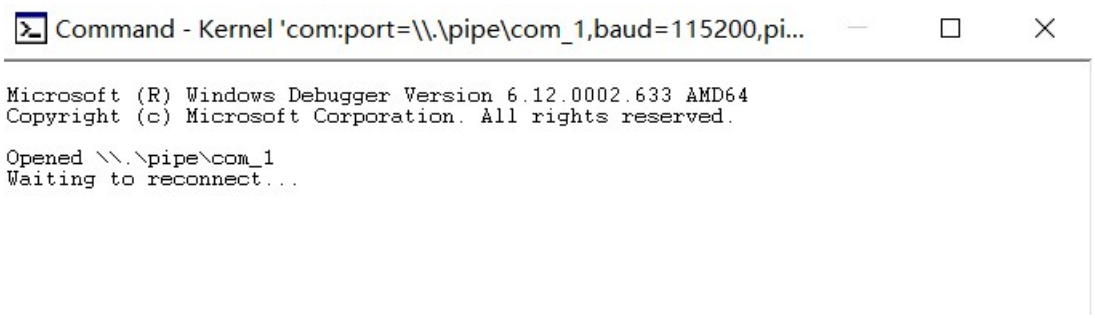


主机：

使用 WinDbg 连接虚拟机，开始内核调试。

安装 WinDbg，打开文件选择 Kernel Debuging，填入端口即可完成连接，开始内核调试：





Lab 10-1

This lab includes both a driver and an executable. You can run the executable from anywhere, but in order for the program to work properly, the driver must be placed in the *C:\Windows\System32* directory where it was originally found on the victim computer. The executable is *Lab10-01.exe*, and the driver is *Lab10-01.sys*.

Questions

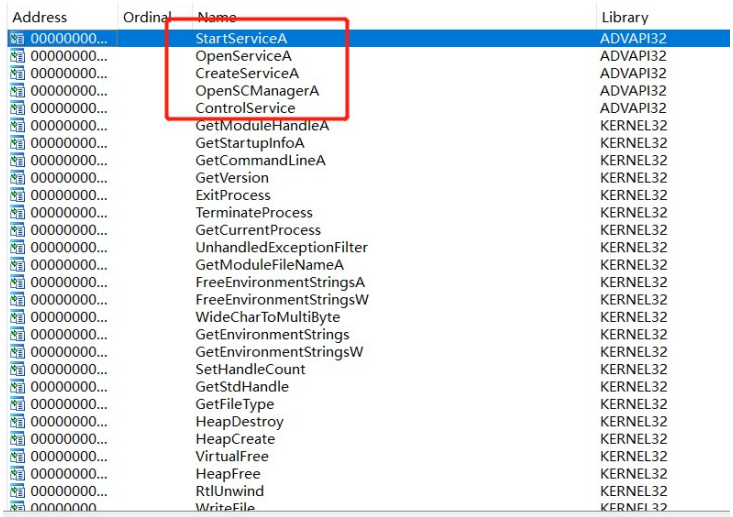
1. Does this program make any direct changes to the registry? (Use procmon to check.)
2. The user-space program calls the `ControlService` function. Can you set a breakpoint with WinDbg to see what is executed in the kernel as a result of the call to `ControlService`?
3. What does this program do?

1、这个程序是否对注册表进行直接的改变？（用 procmon 监测）

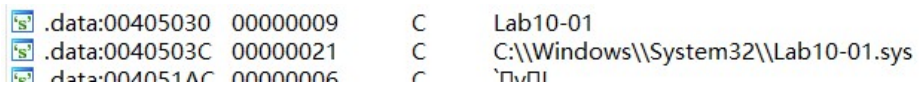
首先静态分析：

IDA Pro

查看注册表，可以看到一些创建、开启服务相关的函数

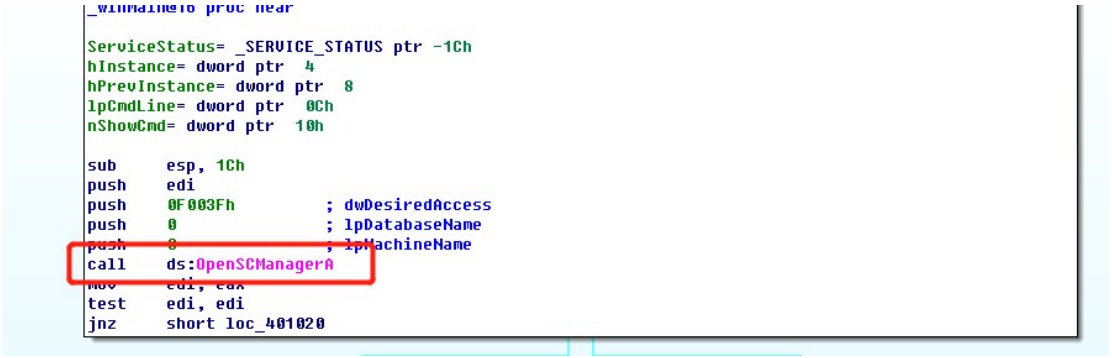


查看字符串

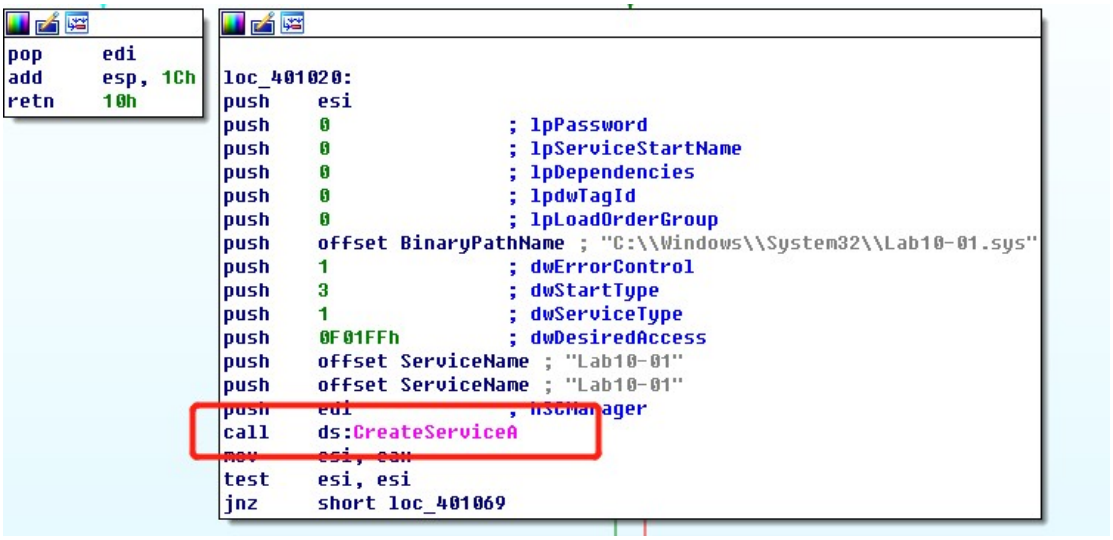


发现敏感路径

深入分析：



OpenSCManger:在指定及其上创建与服务控制管理程序的联系，并打开指定的数据库，返回的是一个服务管理器的句柄。



CreateService:创建一个服务对象，并将它添加到指定的服务控制管理程序的数据库中。

Service 为创建的服务名，在这里为 Lab10-01。

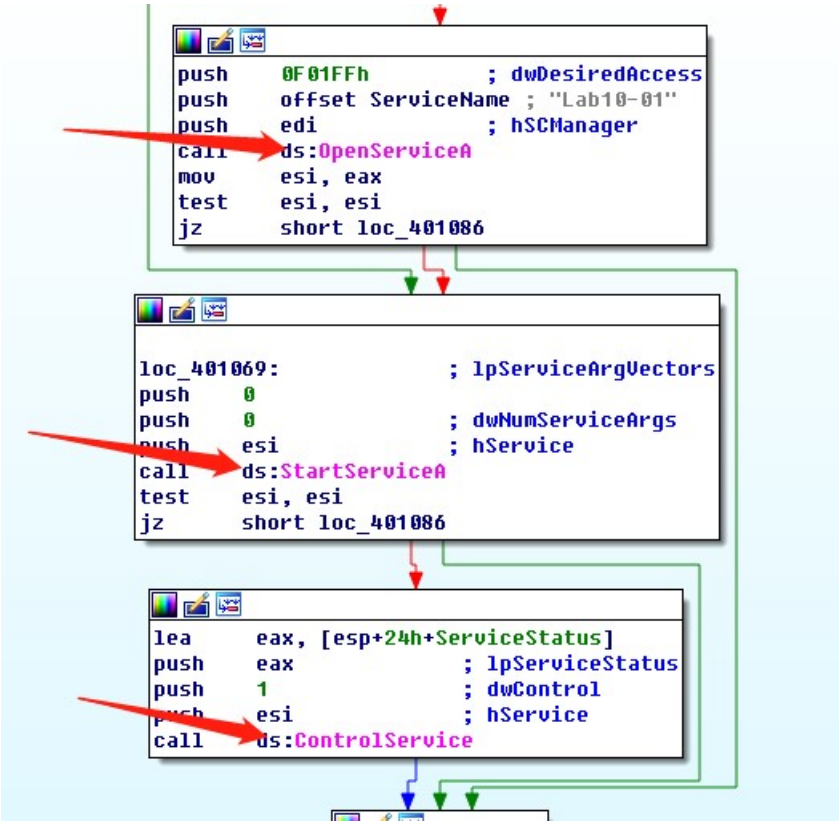
其中，

dwServiceType 为服务类型，1 表示此服务为驱动服务（此文件会加载到内核中去）；3 表示此服务会自动启动 ；

dwErrorControl 表示严重性错误，以及采取的行动，如果这项服务无法启动，1 表示启动程序在事件日志中记录，但继续启动操作；

BinaryPathName 表示服务二进制文件的完全限定路径；

dwDesiredAccess 为访问权限，0xF01FF 表示除此表中的所有访问权限外，还包括 STANDARD_RIGHTS_REQUIRED。



如果服务存在导致服务创建失败，则使用 OpenService 打开同名服务。如果打开成功，使用 StartService 开启服务。

ControlService: hservice，OpenService 或 CreateService 返回的服务句柄。

dwControl，要发送的控制码，此处为 1，表示 CONTROL _SERVICE_STOP,将会卸载驱动并调用驱动卸载的函数。IpServiceStatus，返回值，指向存储服务最新状态的结构体 Service，返回信息来自 SCM 中最近的服务状态报告。

现在来查看具体服务，应该在 Lab10-01.sys 中，分析之：

查看导入表：

Address	Ordinal	Name	Library
00000000...		RtlCreateRegistryKey	ntoskrnl
00000000...		KeTickCount	ntoskrnl
00000000...		RtlWriteRegistryValue	ntoskrnl

与注册表的相关的操作函数；

查看 Strings

Address	Length	Type	String
INIT:000109E2	0000000D	C	ntoskrnl.exe
.reloc:00010E19	0000000E	C	9!939:9?9H9O9

只看到了 ntoskenl.exe，包含大量内核二进制代码。

使用 Strings 工具查看：

```
EnableFirewall
\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFirewall\StandardProfile
\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFirewall\DomainProfile
\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFirewall
\Registry\Machine\SOFTWARE\Policies\Microsoft
RSDS
c:\winddk\7600.16385.1\src\general\regwriter\wdm\sys\objfre_wxp_x86\i386\sioctl.pdb
N@
RtlWriteRegistryValue
```

大量与防火墙注册表相关。

使用 process monitor 监测

发现只有一个对注册表的写操作,还只是写了一个随机种子。

2、用户态的程序调用了 ControlService 函数，你是否能够使用 WinDbg 设置一个断点，以此来观察由于 ControlService 的调用导致内核执行了怎样的操作？

继续静态分析 sys 程序，看服务到底干什么：

```
; NTSTATUS __stdcall DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
public DriverEntry
DriverEntry proc near

DriverObject= dword ptr  8
RegistryPath= dword ptr  0Ch

mov     edi, edi
push    ebp
mov     ebp, esp
call    sub_10920
pop     ebp
jmp     sub_10906
DriverEntry endp
```

该驱动调用了两个函数，驱动的入口为 sub_10906，查看：

```
; Attributes: bp-based frame

sub_10906 proc near

arg_0= dword ptr  8

mov     edi, edi
push    ebp
mov     ebp, esp
mov     eax, [ebp+arg_0]
mov     dword ptr [eax+34h], offset sub_10486
xor     eax, eax
pop     ebp
retn    8
sub_10906 endp
```

查看

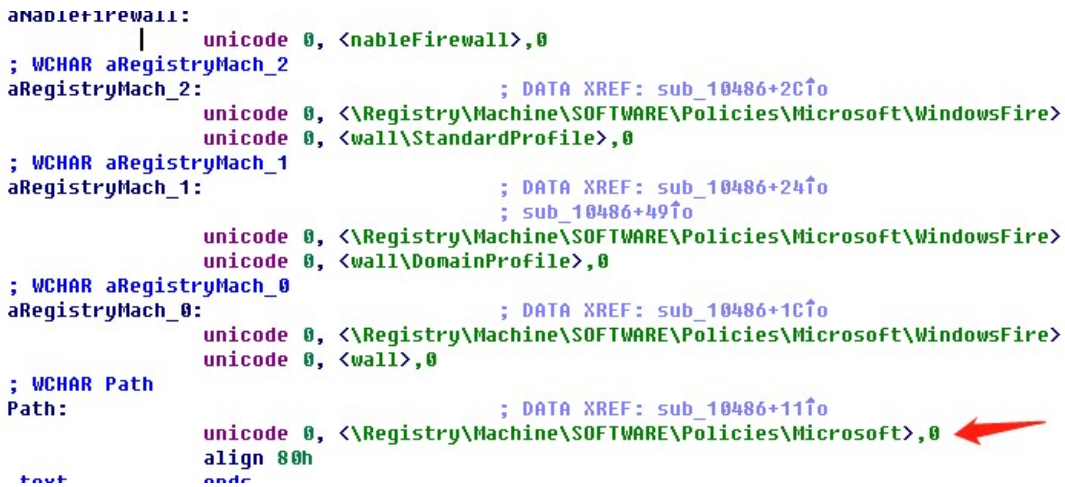
```
mov     esi, ds:RtlCreateRegistryKey
push    edi
xor     edi, edi
push    offset Path     ; "\\Registry\\Machine\\SOFTWARE\\Policies"...
push    edi             ; RelativeTo
mov     [ebp+ValueData], edi
call    esi ; RtlCreateRegistryKey
push    offset aRegistryMach_0 ; "\\Registry\\Machine\\SOFTWARE\\Policies"...
push    edi             ; RelativeTo
call    esi ; RtlCreateRegistryKey
push    offset aRegistryMach_1 ; "\\Registry\\Machine\\SOFTWARE\\Policies"...
push    edi             ; RelativeTo
call    esi ; RtlCreateRegistryKey
mov     ebx, offset aRegistryMach_2 ; "\\Registry\\Machine\\SOFTWARE\\Policies"...
push    ebx             ; Path
push    edi             ; RelativeTo
call    esi ; RtlCreateRegistryKey
mov     esi, ds:RtlWriteRegistryValue
push    4               ; ValueLength
lea     eax, [ebp+ValueData]
push    eax             ; ValueData
push    4               ; ValueType
mov     edi, offset ValueName
push    edi             ; ValueName
push    offset aRegistryMach_1 ; "\\Registry\\Machine\\SOFTWARE\\Policies"...
push    0               ; RelativeTo
call    esi ; RtlWriteRegistryValue
push    4               ; ValueLength
lea     eax, [ebp+ValueData]
push    eax             ; ValueData
push    4               ; ValueType
push    edi             ; ValueName
push    ebx             ; Path
push    0               ; RelativeTo
call    esi ; RtlWriteRegistryValue
```

大量对注册表的操作：

RtlCreaterRegistrykey:通过一个给定的注册表相对路径和值创建指定的键。

RtlWriteRegistryValue:将提供的数据以指定的值名称写入指定的相对路径。

而路径的参数，指向的是防火墙的注册表，所以是对防火墙的修改，



```
aEnableFirewall:
                unicode 0, <nableFirewall>,0
; WCHAR aRegistryMach_2
aRegistryMach_2:                        ; DATA XREF: sub_10486+2C↑o
                unicode 0, <\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFire>
                unicode 0, <wall\StandardProfile>,0
; WCHAR aRegistryMach_1
aRegistryMach_1:                        ; DATA XREF: sub_10486+24↑o
                                        ; sub_10486+49↑o
                unicode 0, <\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFire>
                unicode 0, <wall\DomainProfile>,0
; WCHAR aRegistryMach_0
aRegistryMach_0:                        ; DATA XREF: sub_10486+1C↑o
                unicode 0, <\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFire>
                unicode 0, <wall>,0
; WCHAR Path
Path:                                   ; DATA XREF: sub_10486+11↑o
                unicode 0, <\Registry\Machine\SOFTWARE\Policies\Microsoft>,0
                align 80h
```

基本可以确定，关闭了防火墙。

接下来，用 WinDbg 设置一个断点，观察由于 ControlService 的调用内核执行的操作：

```
.text:0040107F     |            push    esi             ; hService
.text:00401080                  call    ds:ControlService
.text:00401086
```

可知地址为 00401080，打断点：bp 00401080

```
ModLoad: 77da0000 77e49000   C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e50000 77ee2000   C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77fc0000 77fd1000   C:\WINDOWS\system32\Secur32.dll
(650.540): Break instruction exception - code 80000003 (first chance)
eax=00241eb4 ebx=7ffda000 ecx=00000007 edx=00000080 esi=00241f48 edi=00241eb4
eip=7c92120e esp=0012fb20 ebp=0012fc94 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for ntdll.dll -
ntdll!DbgBreakPoint:
7c92120e cc              int     3
0:000> bp 00401080
*** WARNING: Unable to verify checksum for image00400000
*** ERROR: Module load completed but symbols could not be loaded for image00400000
0:000> g
Breakpoint 0 hit
eax=0012ff1c ebx=7ffda000 ecx=77dbfb6d edx=00000000 esi=00144008 edi=00144f18
eip=00401080 esp=0012ff08 ebp=0012ffc0 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
image00400000+0x1080:
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\WINDOWS\system32\ADVAPI32.dll -
00401080 ff1510404000    call    dword ptr [image00400000+0x4010 (00404010)] ds:0023:00404010={ADVAPI32!ControlService (77dc49dd)}
```

输入命令!drvobj 查看服务 Lab10-01

```
0: kd> !drvobj Lab10-01
Driver object (88d7af38) is for:
DBGHELP: c:/mysymbol*http://msdl.microsoft.com/download/symbols is not a valid store
*** Unable to resolve unqualified symbol in Bp expression 'Lab10-01!DriverEntry'.
 \Driver\Lab10-01
```

没找到，可能是用户不可访问，查看所有驱动列表，命令：!Object \Driver

```
0: kd> !object \drvier
Object drvier not found
0: kd> !Object \Driver
Object: e12a6b50  Type: (897efe70) Directory
    ObjectHeader: e12a6b38 (old version)
    HandleCount: 0  PointerCount: 83
    Directory Object: e1001220  Name: Driver

    Hash Address  Type                      Name
    ---- -------  ----                      ----
     00  89752658 Driver                    NDIS
         89312910 Driver                    KSecDD
     01  895fa490 Driver                    FsVga
         892a3d38 Driver                    Mouclass
         892a9578 Driver                    Raspti
         892a5ce0 Driver                    es1371
     02  89312bc0 Driver                    vmx_svga
     03  896f8d10 Driver                    Fips
         89643828 Driver                    Kbdclass
         89822cd8 Driver                    nvrd32
     04  892aaba8 Driver                    VgaSave
         89642f38 Driver                    NDProxy
         896464b8 Driver                    Compbatt
     05  892c44b8 Driver                    Ptilink
         892c5e18 Driver                    MountMgr
         [illegible] Driver                 wdmaud
     06  88d7af38 Driver                    Lab10-01
     07  [illegible] Driver                 dmload
         89756610 Driver                    isapnp
     08  895ee030 Driver                    redbook
```

使用 dt _DRIVER_OBJECT 地址来查看驱动的数据结构：

```
0: kd> dt _DRIVER_OBJECT 88d7af39
ntdll!_DRIVER_OBJECT
   +0x000 Type             : 0n-22528
   +0x002 Size             : 0n0
   +0x004 DeviceObject     : 0x12000000 _DEVICE_OBJECT
   +0x008 Flags            : 0
   +0x00c DriverStart      : 0x80f7a700 Void
   +0x010 DriverSize       : 0x1000000e
   +0x014 DriverSection    : 0xe0896aca Void
   +0x018 DriverExtension  : 0x2088d7af _DRIVER_EXTENSION
   +0x01c DriverName       : _UNICODE_STRING "--- memory read error at address 0x10e200ce ---"
   +0x024 HardwareDatabase : 0x008069c2 _UNICODE_STRING
   +0x028 FastIoDispatch   : 0x59000000 _FAST_IO_DISPATCH
   +0x02c DriverInit       : 0x00f7a709     long  +f7a709
   +0x030 DriverStartIo    : 0x86000000     void  +ffffffff86000000
   +0x034 DriverUnload     : 0x29f7a704     void  +29f7a704
   +0x038 MajorFunction    : [28] 0x29804fa7     long  +29804fa7
```

看到 DriverUnload 函数，为卸载函数，打断点，看看之后的变化：

bp 0x29f7a704

g

虚拟机中继续执行，主机上运行到断点处：

```
   +0x034 DriverUnload     : 0xf7a76486     v
   +0x038 MajorFunction    : [28] 0x804fa729
0: kd> bp  0xf7a76486
0: kd> g
Breakpoint 1 hit
Lab10_01+0x486:
f7a76486 8bff            mov     edi,edi
```

t 命令，单步执行：

```
Breakpoint 1 hit
Lab10_01+0x486:
f7a76486 8bff            mov     edi,edi
0: kd> p
Lab10_01+0x488:
f7a76488 55              push    ebp
0: kd> t
Lab10_01+0x489:
f7a76489 8bec            mov     ebp,esp
0: kd> t
Lab10_01+0x48b:
f7a7648b 51              push    ecx
1: kd> p
Lab10_01+0x48c:
f7a7648c 53              push    ebx
```

单步运行发现四次调用 RtlCreateRegistryKey,两次调用 RtlWriteRegistryValue

r 查看各个寄存器的值，他们就是各个函数的参数，其中 eax 是要写入的值，edi 是要被写的项名字，ebx 是绝对路径

```
kd> r
eax=ba507d5c ebx=ba77e50c ecx=00000008 edx=80500f8d esi=805e90a8 edi=ba77e4ee
eip=ba77e4e4 esp=ba507d38 ebp=ba507d60 iopl=0         nv up ei pl zr na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000246
```

```
kd> db ba77e50c L100
ba77e50c  5c 00 52 00 65 00 67 00-69 00 73 00 74 00 72 00  \.R.e.g.i.s.t.r.
ba77e51c  79 00 5c 00 4d 00 61 00-63 00 68 00 69 00 6e 00  y.\.M.a.c.h.i.n.
ba77e52c  65 00 5c 00 53 00 4f 00-46 00 54 00 57 00 41 00  e.\.S.O.F.T.W.A.
ba77e53c  52 00 45 00 5c 00 50 00-6f 00 6c 00 69 00 63 00  R.E.\.P.o.l.i.c.
ba77e54c  69 00 65 00 73 00 5c 00-4d 00 69 00 63 00 72 00  i.e.s.\.M.i.c.r.
ba77e55c  6f 00 73 00 6f 00 66 00-74 00 5c 00 57 00 69 00  o.s.o.f.t.\.W.i.
ba77e56c  6e 00 64 00 6f 00 77 00-73 00 46 00 69 00 72 00  n.d.o.w.s.F.i.r.
ba77e57c  65 00 77 00 61 00 6c 00-6c 00 5c 00 53 00 74 00  e.w.a.l.l.\.S.t.
ba77e58c  61 00 6e 00 64 00 61 00-72 00 64 00 50 00 72 00  a.n.d.a.r.d.P.r.
```

edi 是 Enablefirewall；eax 是 0；ebx 是

\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFirewall\StandardProfile

所以这个程序就是把 Enablefirewall 修改为 0，即关闭防火墙。

这里看着比较麻烦，回到 IDA 中看，地址偏移为 0x486，而 Driver 的基址为 0x00010000，

所以跳到从 0x00010486 开始查看，可以得知，完全一样，与静态分析一致。

3、程序做啥了

通过创建服务来加载驱动，驱动代码会创建并修改注册表键值来关闭防火墙。

Lab 10-2

The file for this lab is *Lab10-02.exe*.

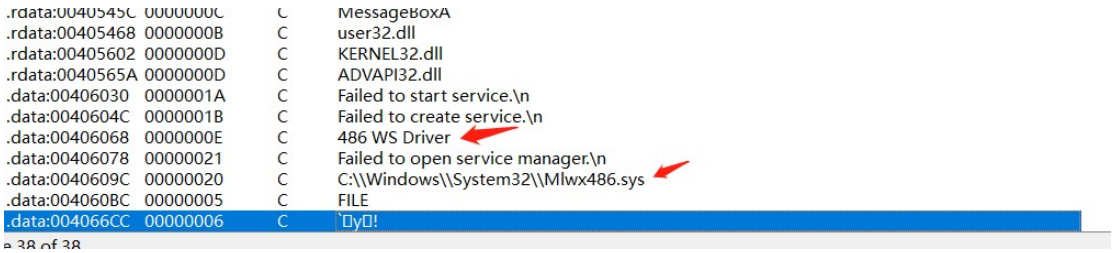
Questions

1. Does this program create any files? If so, what are they?
2. Does this program have a kernel component?
3. What does this program do?

1、这个程序创建文件了吗？是什么？

静态分析：

查看字符串：

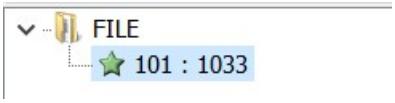


发现一个驱动，和一个.sys 文件，猜测可能是创建的

导入表：

Address	Ordinal	Name	Library
00000000...		CreateServiceA	ADVAPI32
00000000...		StartServiceA	ADVAPI32
00000000...		CloseServiceHandle	ADVAPI32
00000000...		OpenSCManagerA	ADVAPI32
00000000...		CreateFileA	KERNEL32
00000000...		SizeofResource	KERNEL32
00000000...		WriteFile	KERNEL32
00000000...		FindResourceA	KERNEL32
00000000...		LoadResource	KERNEL32
00000000...		CloseHandle	KERNEL32
00000000		GetCommandLineA	KERNEL32

创建服务、创建文件，写文件，加载资源啥的



用 Resource Hacker 查看资源，果然发现了，资源名为 file

深入分析：

```
.text:00401004                push    offset Type     ; "FILE"
.text:00401009                push    65h             ; lpName
.text:0040100B                push    0               ; hModule
.text:0040100D                call    ds:FindResourceA
.text:00401013                mov     edi, eax
.text:00401015                push    edi             ; hResInfo
.text:00401016                push    0               ; hModule
.text:00401018                call    ds:LoadResource
.text:0040101E                test    edi, edi
.text:00401020                mov     ebx, eax
.text:00401022                jz      loc_4010FF
.text:00401028                push    0               ; hTemplateFile
.text:0040102A                push    80h             ; dwFlagsAndAttributes
.text:0040102F                push    2               ; dwCreationDisposition
.text:00401031                push    0               ; lpSecurityAttributes
.text:00401033                push    0               ; dwShareMode
.text:00401035                push    0C0000000h      ; dwDesiredAccess
.text:0040103A                push    offset BinaryPathName ; "C:\\Windows\\System32\\Mlwx486.sys"
.text:0040103F                call    ds:CreateFileA
.text:00401045                mov     esi, eax
.text:00401047                cmp     esi, 0FFFFFFFFh
.text:0040104A                jz      loc_4010FF
.text:00401050                lea     eax, [esp+10h+NumberOfBytesWritten]
.text:00401054                push    0               ; lpOverlapped
.text:00401056                push    eax             ; lpNumberOfBytesWritten
.text:00401057                push    edi             ; hResInfo
.text:00401058                push    0               ; hModule
.text:0040105A                call    ds:SizeofResource
.text:00401060                push    eax             ; nNumberOfBytesToWrite
.text:00401061                push    ebx             ; lpBuffer
.text:00401062                push    esi             ; hFile
.text:00401063                call    ds:WriteFile
.text:00401069                push    esi             ; hObject
```

先是 FindResourceA 找到名为 file 的资源传回资源块句柄，然后 LoadResource 加载这个资源，返回被装载的资源句柄，，然后 CreateFileA 创建 C:\\Windows\\System32\\Mlwx486.sys 这个文件，然后 SizeofResource 查看资源大小，然后 WriteFile 把资源写到创建的那个文件里面，之后关闭句柄；

```
.text:00401075                push    0               ; lpDatabaseName
.text:00401077                push    0               ; lpMachineName
.text:00401079                call    ds:OpenSCManagerA
.text:0040107F                test    eax, eax
.text:00401081                jnz     short loc_401097
.text:00401083                push    offset aFailedToOpenSe ; "Failed to open service manager.\n"
.text:00401088                call    _printf
.text:0040108D                add     esp, 4
.text:00401090                xor     eax, eax
```

OpenSCManagerA 打开服务管理器,失败输出相应信息然后结束,否则继续；

```
.text:0040109D                push    0               ; lpdwTagId
.text:0040109F                push    0               ; lpLoadOrderGroup
.text:004010A1                push    offset BinaryPathName ; "C:\\Windows\\System32\\Mlwx486.sys"
.text:004010A6                push    1               ; dwErrorControl
.text:004010A8                push    3               ; dwStartType
.text:004010AA                push    1               ; dwServiceType
.text:004010AC                push    0F01FFh         ; dwDesiredAccess
.text:004010B1                push    offset DisplayName ; "486 WS Driver"
.text:004010B6                push    offset DisplayName ; "486 WS Driver"
.text:004010BB                push    eax             ; hSCManager
.text:004010BC                call    ds:CreateServiceA
.text:004010C2                mov     esi, eax
.text:004010C4                test    esi, esi
.text:004010C6                jnz     short loc_4010DC
.text:004010C8                push    offset aFailedToCreate ; "Failed to create service.\n"
.text:004010CD                call    _printf
.text:004010D2                add     esp, 4
```

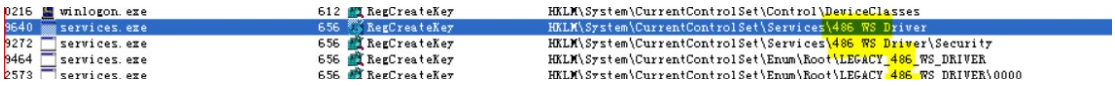
然后 CreateServiceA 创建 486 WS Driver 驱动服务,两个参数需要注意,一个 dwstarttype 是 3 说明服务管理器启动,dwservicetype 是 1 说明是驱动服务程序,同样是失败打印信息结束,成功继续；

```
.text:004010DC loc_4010DC:                            ; CODE XREF: _main+C6↑j
.text:004010DC                push    0               ; lpServiceArgVectors
.text:004010DE                push    0               ; dwNumServiceArgs
.text:004010E0                push    esi             ; hService
.text:004010E1                call    ds:StartServiceA
.text:004010E7                test    eax, eax
.text:004010E9                jnz     short loc_4010F8
.text:004010EB                push    offset aFailedToStartS ; "Failed to start service.\n"
.text:004010F0                call    _printf
.text:004010F5                add     esp, 4
.text:004010F8
.text:004010F8 loc_4010F8:                            ; CODE XREF: _main+E9↑j
.text:004010F8                push    esi             ; hSCObject
.text:004010F9                call    ds:CloseServiceHandle
.text:004010FF
.text:004010FF loc_4010FF:                            ; CODE XREF: _main+22↑j
.text:004010FF                                        ; _main+4A↑j
.text:004010FF                pop     edi
.text:00401100                pop     esi
```

然后开启服务，失败打印错误信息，成功继续。

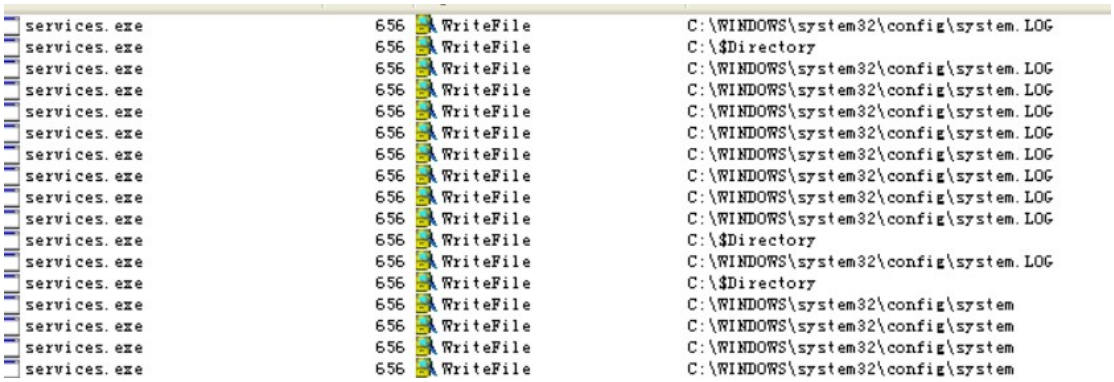
动态分析：

用 procomn 检测：

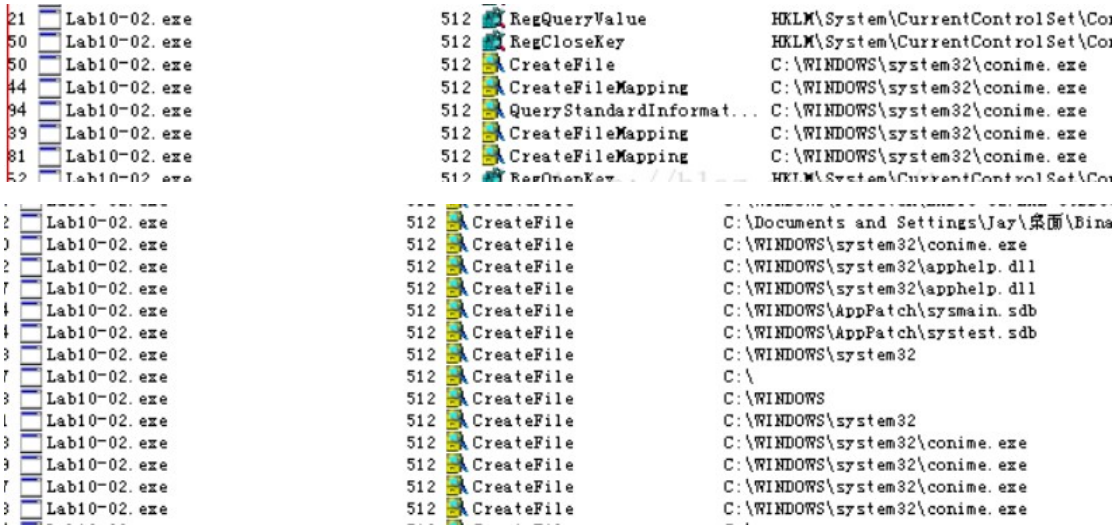


发现创建了 services.exe 的代码，执行了 RegCreateKey

然后搜索 services.exe



可以得知这个是写文件的，再查 Lab10-02.exe 进程：

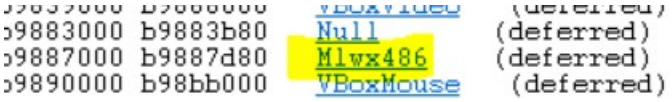


发现还创建了 conime.exe、apphelp.dll、sysmain.sdb、systest.sdb 文件，还有之前的 Mlwx486.sys。

2、这个程序有内核组件吗？

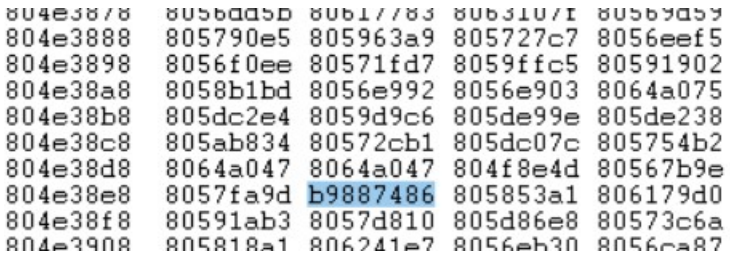
内核调试：

命令 lm，可以找到 Mlwx486.sys 驱动。



查看 SSDT 的修改项

dd dwo(KeServiceDescriptorTable) L100



有一个跳转很多的内存地址，查看是什么函数：

```
kd> u b9887486
*** ERROR: Module load completed but symbols
Mlwx486+0x486:
b9887486 8bff            mov     edi,edi
b9887488 55              push    ebp
b9887489 8bec            mov     ebp,esp
```

下一步把虚拟机恢复成 roobit 安装之前的样子，看看这个位置是啥：

```
04e38a8  8058b1bd 8056e992 8056e903 8064a075
04e38b8  805dc2e4 8059d9c6 805de99e 805de238
04e38c8  805ab834 80572cb1 805dc07c 805754b2
04e38d8  8064a047 8064a047 804f8e4d 80567b9e
04e38e8  8057fa9d 80573111 805853a1 806179d0
04e38f8  80591ab3 8057d810 805d86e8 80573c6a
04e3908  805818a1 806241e7 8056eb30 8056ca87
```

是 80573111，查看这个函数：

```
kd> u 80573111
nt!NtQueryDirectoryFile:
80573111 8bff            mov     edi,edi
80573113 55              push    ebp
80573114 8bec            mov     ebp,esp
80573116 8d452c          lea     eax,[ebp+2Ch]
80573119 50              push    eax
8057311a 8d4528          lea     eax,[ebp+28h]
8057311d 50              push    eax
```

在 f7a75486 这打断点，单步调试，得到程序：

mov edi, edi

push ebp

mov ebp, esp

push esi

mov esi, dword ptr [ebp+1Ch]

push edi

push dword ptr [ebp+30h]

push dword ptr [ebp+2Ch]

push dword ptr [ebp+28h]

push dword ptr [ebp+24h]

push dword ptr [ebp+20h]

push esi

push dword ptr [ebp+18h]

push dword ptr [ebp+14h]

push dword ptr [ebp+10h]

```
push dword ptr [ebp+0Ch]
push dword ptr [ebp+8]
call Miwx486+0x514(f7a75514)
mov    esi, dword ptr [ebp+1Ch]
push edi                              = b95a5d64
/* Note: ebp = b95a5d30 */
push dword ptr [ebp+30h] = [b95a5d60] = 0
push dword ptr [ebp+2Ch] = [b95a5d5c] = 80 e1 70
push dword ptr [ebp+28h] = [b95a5d58] = 1
push dword ptr [ebp+24h] = [b95a5d54] = 3
push dword ptr [ebp+20h] = [b95a5d50] = 268
push esi = 0070e198
push dword ptr [ebp+18h] = [b95a5d48] = 68 e1 70
push dword ptr [ebp+14h] = [b95a5d44] = 0
push dword ptr [ebp+10h] = [b95a5d40] = 0
push dword ptr [ebp+0Ch] = [b95a5d3c] = 0
push dword ptr [ebp+8]      = [b95a5d38] = 464
call Miwx486+0x514(f7ab7514)
```

NtQueryDirectoryFile 的定义：

```
NTSTATUS ZwQueryDirectoryFile(
  _In_         HANDLE                 FileHandle = 464,
  _In_opt_ HANDLE                     Event = 0,
  _In_opt_ PIO_APC_ROUTINE            ApcRoutine = 0,
  _In_opt_ PVOID                      ApcContext = 0,
  _Out_       PIO_STATUS_BLOCK        IoStatusBlock = 68 e1 70,
  _Out_       PVOID                   FileInformation = 0070e198,
  _In_         ULONG                  Length = 268,
  _In_         FILE_INFORMATION_CLASS FileInformationClass = 3,
  _In_         BOOLEAN                ReturnSingleEntry = 1,
  _In_opt_ PUNICODE_STRING            FileName = 80 e1 70,
```

_In_ BOOLEAN RestartScan = 0

);

第八个入参 FileInformationClass 的值为 3，然后我们按 t 来进入这个函数

```
kd> p
Mlwx486+0x4af:
f7ab74af e860000000      call    Mlwx486+0x514 (f7ab7514)
kd> t
Mlwx486+0x514:
f7ab7514 ff258075abf7    jmp     dword ptr [Mlwx486+0x580 (f7ab7580)]
```

跳转到保存在地址 f7ab7580 上那个地址，继续

```
kd> p
nt!NtQueryDirectoryFile:
80573111 8bff            mov     edi,edi
```

这个函数标注为 nt!NtQueryDirectoryFile，就是那个被替换的函数的本身

```
kd> p
Mlwx486+0x4b6:
f7ab74b6 837d2403        cmp     dword ptr [ebp+24h],3
```

[ebp+24h]，是 FileInformationClass 的值，从这里开始比较这个 FileInformationClass 的值

之后把 eax 赋值为 0，然后跳转：

cmp dword ptr [ebp+24h], 3

mov dword ptr [ebp+30h], eax

jne Mlwx486+0x505 (f7ab7505)

这里由于 zf 的值为 1，所以不跳转；

如果 FileInformationClass 的值是 3 的话，继续执行：

太长了，就这样分析，可以得到全部的内核代码（好恶心）：

```
/* 栈初始化开始 */
mov    edi, edi
push ebp
mov    ebp, esp
push esi
/* 栈初始化结束 */
mov    esi, dword ptr [ebp+1Ch]
push edi
```

```
push dword ptr [ebp+30h] // RestartScan
push dword ptr [ebp+2Ch] // FileName
push dword ptr [ebp+28h] // ReturnSingleEntry
push dword ptr [ebp+24h] // FileInformationClass
push dword ptr [ebp+20h] // Length
push esi                 // FileInformation
push dword ptr [ebp+18h] // IoStatusBlock
push dword ptr [ebp+14h] // ApcContext
push dword ptr [ebp+10h] // PacRoutine
push dword ptr [ebp+0Ch] // Event
push dword ptr [ebp+8]   // FileHandle
call Mlwx486+0x514       //-> jmp dword ptr [Mlwx486+0x580 (f7ab2580)] -> nt!NtQueryDirectoryFile
xor  edi, edi
cmp  dword ptr [ebp+24h], 3   // FileInformationClass = 3
mov  dword ptr [ebp+30h], eax // RestartScan, 0
jne  Mlwx486+0x505            // if [ebp+24h] != 3 -> jmp and ret 2Ch
test eax, eax                 // eax is NtQueryDirectoryFile return value(success return 0)
jl   Mlwx486+0x505            // if eax < 0 -> jmp and ret 2Ch
cmp  byte ptr [ebp+28h], 0    // ReturnSingleEntry = 1
jne  Mlwx486+0x505            // if [ebp+28h] != 0 -> jmp and ret 2Ch
push ebx                      //-> p(f7ab2486)
push 8                        // function Mlwx486+0x4ca here
push offset Mlwx486+0x51a     //-> 'Mlwx'
lea  eax, [esi+5Eh]
push eax
xor  bl, bl
call dword ptr [Mlwx486+0x590] // standard windows nt function RtlCompareMemory
cmp  eax, 8                   // eax = 0
```

```
jne    Mlwx486+0x4f4
        |_ mov    eax , dword ptr [esi] // [esi] = 0
            test eax, eax
            je     Mlwx486+0x504              // if eax == 0 -> jmp and ret 2Ch
            test bl, bl                       // bl always equal 0
            jne    Mlwx486+0x500              // if bl != 0 -> jmp back to 'push 8'
            mov    edi, esi
            add    esi, eax
            jmp    Mlwx486+0x4ca              // jmp back to 'push 8'
            pop    ebx
            mov    eax, dword ptr [ebp+30h]
            pop    edi
            pop    esi
            pop    ebp
            ret    2Ch
inc    bl
test edi, edi
je     Mlwx486+0x4f4
mov    eax, dword ptr [esi]
test eax, eax
jne    Mlwx486+0x4f2
and    dword ptr [edi], eax
jmp    Mlwx486+0x4f4
add    dword ptr [edi], eax
mov    eax, dword ptr [esi]
test eax, eax
je     Mlwx486+0x504
test bl, bl
jne    Mlwx486+0x500
```

```
mov    edi, esi
add    esi, eax
jmp    Mlwx486+0x4ca
pop    ebx
mov    eax dword ptr [ebp+30h]
pop    edi
pop    esi
pop    ebp
ret    2Ch
```

代码的大概意思：

先是生成 KeServiceDescriptorTable NtQueryDirectoryFile 的 Unicode 字符串，

然后 MmGetSystemRoutineAddress,分别得到他们的地址，然后在系统服务描述符表里面寻找 NtQueryDirectoryFile,找到了用函数 sub_10486 的地址覆盖，而 sub_10486 函数的作用：

先是调用原 NtQueryDirectoryFile 函数,然后是三个判断 FileInformationClass == 3 && 返回值>= 0 && 0==ReturnSingleEntry,满足这三个条件才继续，然后 RtlCompareMemory 比较每个文件的文件名的头四个字符是不是 Mlwx,是的话才继续操作,否则继续比较，最后将指向 Mlwx 的 FILE_BOTH_DIR_INFORMATION 结构的上个 FILE_BOTH_DIR_INFORMATION 指向原本 Mlwx 的 FILE_BOTH_DIR_INFORMATION 指向的下一个 FILE_BOTH_DIR_INFORMATION 结构来达到隐藏 Mlwx 文件的目的。

所以，这个程序拥有一个内核模块，存储在程序的资源节上，执行的时候释放 sys 文件，然后这个 sys 文件就会加载到内核中执行

3、程序目的

从上述分析可知，这个程序是用来隐藏文件的，就是隐藏那个创建的.sys 文件。

Lab 10-3

This lab includes a driver and an executable. You can run the executable from anywhere, but in order for the program to work properly, the driver must be placed in the *C:\Windows\System32* directory where it was originally found on the victim computer. The executable is *Lab10-03.exe*, and the driver is *Lab10-03.sys*.

Questions

1. What does this program do?
2. Once this program is running, how do you stop it?
3. What does the kernel component do?

静态分析：

查看字符串：

Address	Length	Type	String
.rdata:004040FC	0000000F	C	runtime error
.rdata:00404110	0000000E	C	TLOSS error\r\n
.rdata:00404120	0000000D	C	SING error\r\n
.rdata:00404130	0000000F	C	DOMAIN error\r\n
.rdata:00404140	00000025	C	R6028\r\n- unable to initialize heap\r\n
.rdata:00404168	00000035	C	R6027\r\n- not enough space for lowio initialization\r\n
.rdata:004041A0	00000035	C	R6026\r\n- not enough space for stdio initialization\r\n
.rdata:004041D8	00000026	C	R6025\r\n- pure virtual function call\r\n
.rdata:00404200	00000035	C	R6024\r\n- not enough space for _onexit/atexit table\r\n
.rdata:00404238	00000029	C	R6019\r\n- unable to open console device\r\n
.rdata:00404264	00000021	C	R6018\r\n- unexpected heap error\r\n
.rdata:00404288	0000002D	C	R6017\r\n- unexpected multithread lock error\r\n
.rdata:004042B8	0000002C	C	R6016\r\n- not enough space for thread data\r\n
.rdata:004042E4	00000021	C	\r\nabnormal program termination\r\n
.rdata:00404308	0000002C	C	R6009\r\n- not enough space for environment\r\n
.rdata:00404334	0000002A	C	R6008\r\n- not enough space for arguments\r\n
.rdata:00404360	00000025	C	R6002\r\n- floating point not loaded\r\n
.rdata:00404388	00000025	C	Microsoft Visual C++ Runtime Library
.rdata:004043B4	0000001A	C	Runtime Error!\n\nProgram:
.rdata:004043D4	00000017	C	<program name unknown>
.rdata:004043EC	00000013	C	GetLastActivePopup
.rdata:00404400	00000010	C	GetActiveWindow
.rdata:00404410	0000000C	C	MessageBoxA
.rdata:0040441C	0000000B	C	user32.dll
.rdata:004045B0	0000000D	C	KERNEL32.dll
.rdata:00404608	0000000D	C	ADVAPI32.dll
.rdata:0040464C	0000000A	C	ole32.dll

有服务相关的信息和 COM 组件，之前做过一个广告弹窗的，可能一样。

Address	Ordinal	Name	Library
00000000...		CloseServiceHandle	ADVAPI32
00000000...		OpenSCManagerA	ADVAPI32
00000000...		CreateServiceA	ADVAPI32
00000000...		StartServiceA	ADVAPI32
00000000...		CreateFileA	KERNEL32
00000000...		DeviceIoControl	KERNEL32
00000000...		Sleep	KERNEL32
00000000...		GetStringTypeA	KERNEL32
00000000...		LCMapStringW	KERNEL32
00000000...		LCMapStringA	KERNEL32
00000000...		MultiByteToWideChar	KERNEL32
00000000...		LoadLibraryA	KERNEL32
00000000...		GetProcAddress	KERNEL32
00000000...		GetModuleHandleA	KERNEL32
00000000...		GetStartupInfoA	KERNEL32
00000000...		GetCommandLineA	KERNEL32
00000000...		GetVersion	KERNEL32
00000000...		ExitProcess	KERNEL32
00000000...		TerminateProcess	KERNEL32
00000000...		GetCurrentProcess	KERNEL32
00000000...		UnhandledExceptionFilter	KERNEL32
00000000...		GetModuleFileNameA	KERNEL32

导入表也是如此。

分析主程序：

打开服务管理器，然后创建 Process Helper 服务,并启动

```
push    esi             ; hService
call    ds:StartServiceA
```

```
loc_401057:             ; hSCObject
push    esi
call    ds:CloseServiceHandle
push    0               ; hTemplateFile
push    80h             ; dwFlagsAndAttributes
push    2               ; dwCreationDisposition
push    0               ; lpSecurityAttributes
push    0               ; dwShareMode
push    0C0000000h      ; dwDesiredAccess
push    offset FileName ; "\\\\.\\ProcHelper"
call    ds:CreateFileA
cmp     eax, 0FFFFFFFFh
jnz     short loc_40108C
```

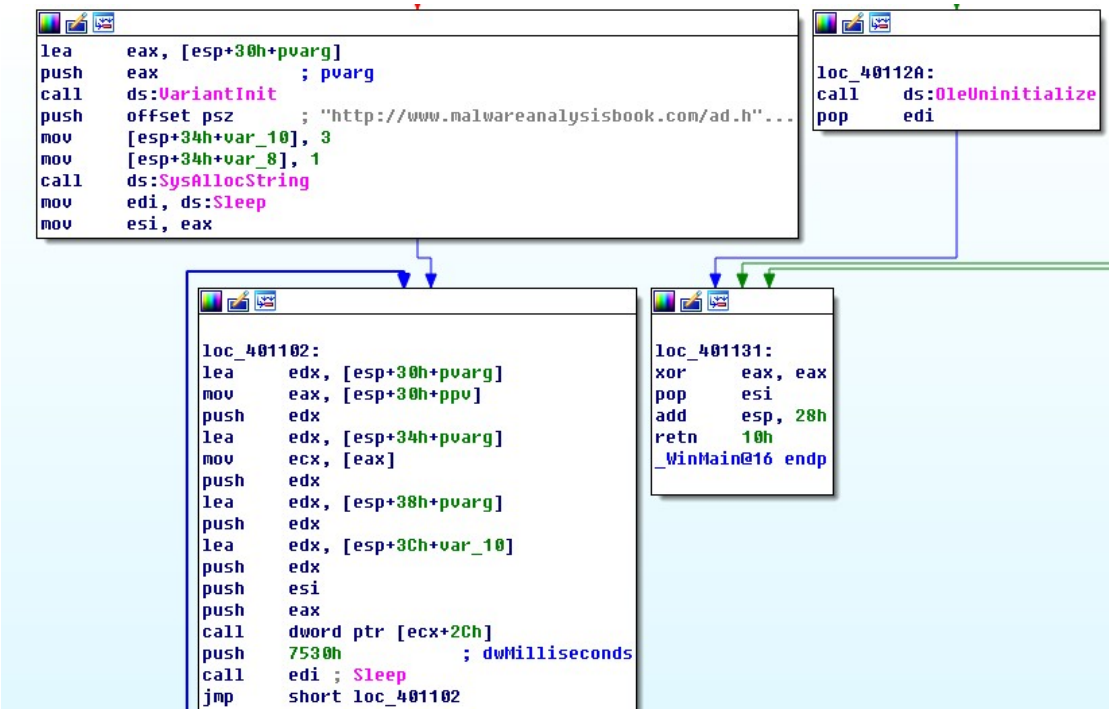
```
loc_401057:             ; hSCObject
push    esi
call    ds:CloseServiceHandle
push    0               ; hTemplateFile
push    80h             ; dwFlagsAndAttributes
push    2               ; dwCreationDisposition
push    0               ; lpSecurityAttributes
push    0               ; dwShareMode
push    0C0000000h      ; dwDesiredAccess
push    offset FileName ; "\\\\.\\ProcHelper"
call    ds:CreateFileA
```

关闭服务句柄,得到\\.\ProcHelper 这个设备句柄

```
push    0               ; nOutBufferSize
push    0               ; lpOutBuffer
push    0               ; nInBufferSize
push    0               ; lpInBuffer
push    0ABCDEF01h      ; dwIoControlCode
push    eax             ; hDevice
call    ds:DeviceIoControl
push    0               ; pvReserved
call    ds:OleInitialize
test    eax, eax
jl      short loc_401131
```

```
lea     edx, [esp+2Ch+ppv]
push    edi
push    edx             ; ppv
push    offset riid     ; riid
push    4               ; dwClsContext
push    0               ; pUnkOuter
push    offset rclsid   ; rclsid
call    ds:CoCreateInstance
mov     eax, [esp+30h+ppv]
test    eax, eax
jz      short loc_40112A
```

然后发送请求到内核驱动



最后就是一个广告，跟之前的一样

接下来对内核分析，看看内核收到句柄后如何操作

```
INIT:000107DE word_107DE      dw 5Ch                  ; DATA XREF: sub_10706+4B↑o
INIT:000107E0 aDosdevicesPr_0:
INIT:000107E0                 unicode 0, <DosDevices\ProcHelper>,0
INIT:0001080C ; const WCHAR aDeviceProchelp
INIT:0001080C aDeviceProchelp:                        ; DATA XREF: sub_10706+14↑o
INIT:0001080C                 unicode 0, <\Device\ProcHelper>,0
INIT:00010832                 align 4
INIT:00010834 __IMPORT_DESCRIPTOR_ntoskrnl_exe dd rva off_1085C
INIT:00010834                                         ; DATA XREF: HEADER:000102C8↑o
INIT:00010834                                         ; Import Name Table
INIT:00010838                 dd 0                    ; Time stamp
INIT:0001083C                 dd 0                    ; Forwarder Chain
INIT:00010840                 dd rva aNtoskrnl_exe    ; DLL Name
INIT:00010844                 dd rva IofCompleteRequest ; Import Address Table
INIT:00010848                 dd 5 dup(0)
INIT:0001085C ;
INIT:0001085C ; Import names for ntoskrnl.exe
INIT:0001085C ;
INIT:0001085C off_1085C       dd rva word_10880       ; DATA XREF: INIT:__IMPORT_DESCRIPTOR_ntoskrnl_exe↑o
INIT:00010860                 dd rva word_10896
INIT:00010864                 dd rva word_108A8
INIT:00010868                 dd rva word_108C0
INIT:0001086C                 dd rva word_108D8
```

创建名为\Device\ProcHelper 的设备，然后创建一个\DosDevices\ProcHelper 的符号链接供用户访问，到这我真不会了 qwq，理解都不能理解，对不起，查看网上资源：

查看 sub_10706 函数：

```
INIT:00010744                   jl      short loc_10789
INIT:00010746                   mov     eax, offset sub_10606
INIT:0001074B                   mov     [esi+38h], eax
INIT:0001074E                   mov     [esi+40h], eax
INIT:00010751                   push    offset word_107DE ; SourceString
INIT:00010756                   lea     eax, [ebp+SymbolicLinkName]
INIT:00010759                   push    eax             ; DestinationString
INIT:0001075A                   mov     dword ptr [esi+70h], offset sub_10666
INIT:00010761                   mov     dword ptr [esi+34h], offset sub_1062A
INIT:00010768                   call    edi ; RtlInitUnicodeString
INIT:0001076A                   lea     eax, [ebp+DestinationString]
INIT:0001076D                   push    eax             ; DeviceName
INIT:0001076E                   lea     eax, [ebp+SymbolicLinkName]
INIT:00010771                   push    eax             ; SymbolicLinkName
INIT:00010772                   call    ds:IoCreateSymbolicLink
INIT:00010778                   mov     esi, eax
INIT:0001077A                   test    esi, esi
INIT:0001077C                   jge     short loc_10787
```

这里修改了主函数表偏移 0(38h),2(40h-38h=8,8/4=2),e(70h-38h=38h,56/4=14=eh)

这三处按照课本上说法是 create close 和 deviceIocontrol,他们分别被 sub_10606 和 sub10666 替代,其中 10606 为 IofcompleteRequest 就返回。

而 sub'_10666 处的函数修改了程序 peb：

```
PAGE:00010666
PAGE:00010666                   mov     edi, edi
PAGE:00010668                   push    ebp
PAGE:00010669                   mov     ebp, esp
PAGE:0001066B                   call    ds:IoGetCurrentProcess
PAGE:00010671                   mov     ecx, [eax+8Ch]
PAGE:00010677                   add     eax, 88h
PAGE:0001067C                   mov     edx, [eax]
PAGE:0001067E                   mov     [ecx], edx
PAGE:00010680                   mov     ecx, [eax]
PAGE:00010682                   mov     eax, [eax+4]
PAGE:00010685                   mov     [ecx+4], eax
PAGE:00010688                   mov     ecx, [ebp+Irp]  ; Irp
PAGE:0001068B                   and     dword ptr [ecx+18h], 0
PAGE:0001068F                   and     dword ptr [ecx+1Ch], 0
PAGE:00010693                   xor     dl, dl          ; PriorityBoost
PAGE:00010695                   call    ds:IofCompleteRequest
PAGE:0001069B                   xor     eax, eax
PAGE:0001069D                   pop     ebp
PAGE:0001069E                   retn    8
PAGE:0001069E sub_10666         endp
```

假设一个结构体：

LIST{

LIST *BLINK(指向前一项相当于 88h 处)

LIST *FLINK(指向后一项相当于 8ch 处)

}*LINK 自身相当于初始 eax

现在 ecx=*FLINk,edx=*Blink,现在对[ecx]赋值,这里[ecx]指的是 FLINK 指向的那个 LIST 结构体的第一项也就是下一个结构体的*BLINk 现在他被赋值为 edx 也就是当前结构体指向的前一个结构体,然后后面三个赋值跟这个差不多,只不过是,前一个结构体第二个项被赋值为当前结构体指向的下一个结构体。也就是隐藏了当前进程。

所以这个程序会隐藏自身进程的广告弹窗。

1、程序做了什么？

隐藏自身进程的广告弹窗

2、一旦程序运行，怎么阻止？

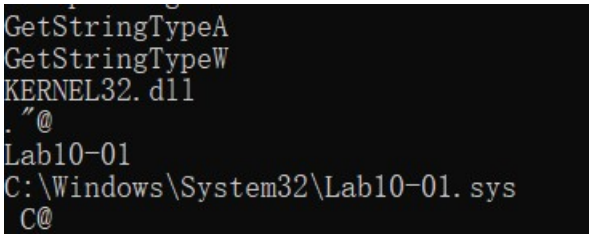
整个分析过程未发现可以认为控制的程序出口，书上说，重启

3、内核组件做了什么？

修改了进程链接表的结构，隐藏了自己的 LIST_ENTRY，就是隐藏了广告进程。

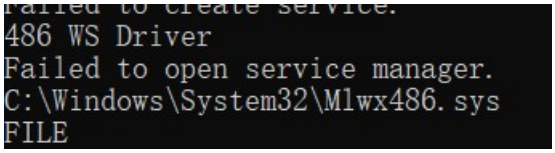
Yara 规则：

Lab10-01.exe




```
rule Lab10_1_feature{
    meta:
        description = "Lab10-01.exe’s features"
    strings:
        $s1 = "C:\\Windows\\System32\\Lab10-01.sys" fullword ascii
        $s2 = "Lab10-01" fullword ascii
        $s3 = "runtime error" fullword ascii
    condition:
        $s1 and $s2 and $s3
}
```

Lab10-02.exe



```
rule Lab10_2_feature{
    meta:
        description = "Lab10-02.exe’s features"
    strings:
        $s1 = "C:\\Windows\\System32\\Mlwx486.sys" fullword ascii
        $s2 = "486 WS Driver" fullword ascii
        $s3 = "Failed to open service manager." fullword ascii
    condition:
        $s1 and $s2 and $s3
}
```

Lab10-03.exe

```
GetStringTypeW
http://www.malwareanalysisbook.com/ad.html
\\.\ProcHelper
Process Helper
C:\Windows\System32\Lab10-03.sys
```

```
rule Lab10_3_feature{
    meta:
        description = "Lab10-03.exe’s features"
    strings:
        $s1 = "Process Helper" fullword ascii
        $s2 = "\\\\.\\ProcHelper" fullword ascii
        $s3 = "http://www.malwareanalysisbook.com/ad.html" fullword ascii
    condition:
        $s1 and $s2 and $s3
}
```

结果：

```
C:\Users\PC\Desktop\恶意代码\实验\yara64>yara64 Lab10_rules.txt C:\Users\PC\Desktop\恶意代码\实验\上机实验样本\Chapter_10L\Lab10-01.exe
Lab10_1_feature C:\Users\PC\Desktop\恶意代码\实验\上机实验样本\Chapter_10L\Lab10-01.exe

C:\Users\PC\Desktop\恶意代码\实验\yara64>yara64 Lab10_rules.txt C:\Users\PC\Desktop\恶意代码\实验\上机实验样本\Chapter_10L\Lab10-02.exe
Lab10_2_feature C:\Users\PC\Desktop\恶意代码\实验\上机实验样本\Chapter_10L\Lab10-02.exe

C:\Users\PC\Desktop\恶意代码\实验\yara64>yara64 Lab10_rules.txt C:\Users\PC\Desktop\恶意代码\实验\上机实验样本\Chapter_10L\Lab10-03.exe
```

IDA Python 脚本：

本次没有用到 IDA Python，但可以分享一下之前用的：

1、快速定位到 init_array 函数：

```
def goInitarray(self):
    # _get_modules 是 idc 提供的接口
    for module in idc._get_modules():
        # 遍历所有 module，找到 linker
        module_name = module.name
        if 'linker' in module_name:
            print 'linker address is ' + str(hex(module.base + 0x2464))
            # 0x2464 是 Android 某个版本的 init_array 的偏移地址,
            # jumpto 可以直接跳转到目标地址
            idc.jumpto(module.base + 0x2464)
            # 在 init_array 上下个断点
            idc.add_bpt(module.base + 0x2464, 1)
            # makecode 更不用说了，相当于 C
            idaapi.auto_make_code(module.base + 0x2464)
```

2、保存日志、函数名字

即保存某些寄存器的值或者某个函数名之类的，方便快速回到调试之前

```
# 通过起始地址，终止地址，以及偏移地址去保存日志
def saveDebugMessage(self):
    # create file first
    # 用个轻量级的存储 shelve
    f = shelve.open(self.id)
    # 保存日志的起始地址
    addr_start = int(self.address_start, 16)
    # 保存日志的终止地址
    addr_end = int(self.address_end, 16)
    log_dict = {}
    log_dict_list = []
    for num in range(addr_start, addr_end):
```

```
            # 获取我们当前地址的日志
            com = idc.GetCommentEx(num, True)
            if com != None:
                #获取函数名
                fun_name = idc.GetFunctionName(num)
                print fun_name
                if fun_name != None and not 'sub' in fun_name:
                    log_dict = {'offset': str(num - addr_start), 'msg': str(com), 'funtion_name':
str(fun_name)}
                else:
                    log_dict = {'offset': str(num - addr_start), 'msg': str(com)}
                log_dict_list.append(log_dict)
                pass
        print(log_dict_list)
        # 保存日志
        f['info'] = log_dict_list
        f.close()
    # 通过起始地址即可，会自动判断长度，并且获取偏移地址去设置日志
def loadDebugMessage(self):
        f = shelve.open(self.id)
        data = f['info']
        addr_start = int(self.address_start, 16)
        for num in range(0, len(data)):
            offset = data[num]['offset']
            msg = data[num]['msg']
            fun_name = data[num]['funtion_name']
            idc.MakeRptCmt(addr_start + int(offset), msg)
            if fun_name is not None and fun_name != '':
                idc.SetFunctionCmt(addr_start + int(offset), fun_name, False)
```

四、 实验心得

本次实验，初次尝试 WinDbg 内核调试，还不是很流畅，很多东西是借鉴网上的，但是算是一个新的世界。看到了针对感染内核的恶意代码的分析过程，尝试用虚拟机和本机，配合来调试内核运行，很新颖，很有意思，就是太难了，但能力得到了提升。

复习了 yara 规则的编写和 IDA Python 脚本的运行。