

恶意代码分析与防治技术实验报告

Lab10

学号： 姓名： 专业：

一、 实验环境

1. 已关闭病毒防护的 Windows10
2. VMware + XP

二、 实验工具

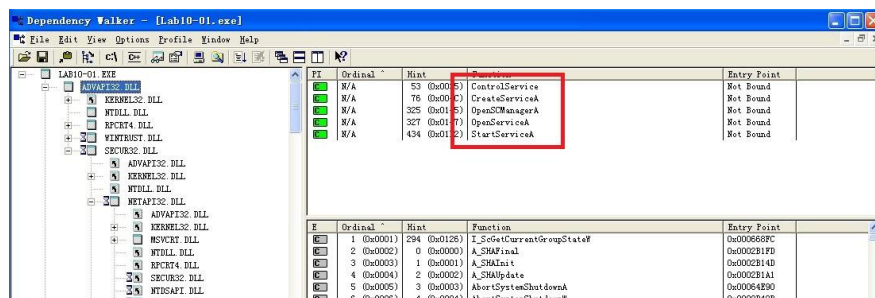
IDA Pro, Strings , Dependency Walker, Resource Hacker, Windbg, Process monitor

三、 Lab10-1

本实验包括一个驱动程序和一个可执行文件。你可以从任意位置运行可执行文件，但为了使程序能够正常运行，必须将驱动程序放到 C:\Windows\System32 目录下，这个目录在受害者计算机中已经存在。可执行文件是 Lab10-01.exe,驱动程序是 Lab 10-01.sys.

具体分析：

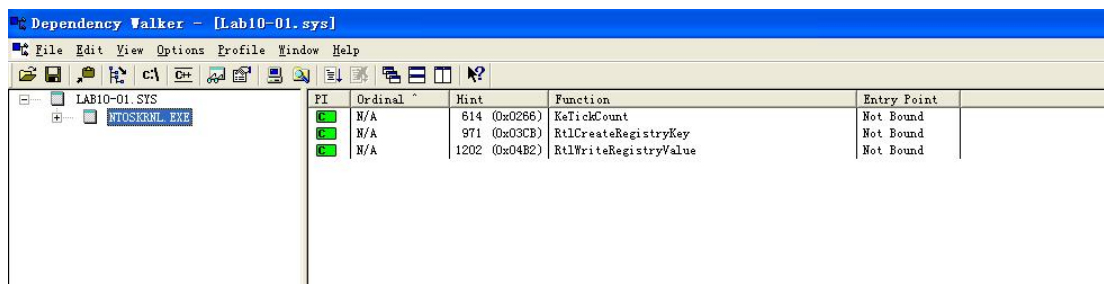
1. 使用 Dependency Walker 进行 Lab10-01.exe 导入函数静态分析，可以观察到下图红框内四个导入函数，表明这个程序创建了一个服务，并且可能启动或者操作了这个服务，除此之外，他与系统进行了很少的交互。



2. 通过 Strings 对 Lab10-01.exe 字符串进行分析，从字符串里可以看到有一个指定文件 C:\Windows\System32\Lab10-01.sys，这暗示 Lab10-01.sys 可能包含这个服务的代码

```
LCMapStringA
LCMapStringW
GetStringTypeA
GetStringTypeW
KERNEL32.dll
"@
Lab10-01
C:\Windows\System32\Lab10-01.sys
C@
xB@
HB@
$B@
A@
(A@
```

3. 使用 Dependency Walker 进行 Lab10-01.sys 导入函数静态分析，可以观察到 RtlCreateRegistryKey 和 RtlWriteRegistryValue，显示驱动可能访问了注册表。



PI	Ordinal	Hint	Function	Entry Point
614	(0x0266)		KeTickCount	Not Bound
971	(0x03CB)		RtlCreateRegistryKey	Not Bound
1202	(0x04B2)		RtlWriteRegistryValue	Not Bound

4. 通过 Strings 对 Lab10-01.sys 字符串进行分析，这些字符串看起来像是注册表键值，然而开头\Regsitry\Machine 却比较怪异，并不像是诸如 HKLM 此类的常见注册表根键。当从内核态访问注册表时，前缀\Registry\Machine 等同于用户态程序访问的 HKEY_LOCAL_MACHINE。

```
WSJ
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
p-
\} ("
c:\winddk\7600.16385.1\src\general\regwriter\wdm\sys\objfre_wxp_x86\siocctl.pdb
```

5. 我们发现 Lab10-01.exe 首先调用了 OpenSCManagerA 获取当前服务管理器的句柄，然后 CreateServiceA 创建一个名为 Lab10-01 的服务，且服务使用了 C:\Windows\System32\Lab10-01.sys 中代码，并且 dwStartType 设置为 3，也就是说这个服务会以内核级运行。

```
ServiceStatus= _SERVICE_STATUS ptr -1Ch
hInstance= dword ptr 4
hPrevInstance= dword ptr 8
lpCmdLine= dword ptr 0Ch
nShowCmd= dword ptr 10h

sub     esp, 1Ch
push    edi
push    0F003Fh      ; dwDesiredAccess
push    0             ; lpDatabaseName
push    0             ; lpMachineName
call    ds:OpenSCManagerA
mov     edi, eax
test    edi, edi
jnz     short loc_401020

loc_401020:
push    esi
push    0             ; lpPassword
push    0             ; lpServiceStartName
push    0             ; lpDependencies
push    0             ; lpdwTagId
push    0             ; lpLoadOrderGroup
push    offset BinaryPathName ; "C:\\Windows\\System32\\Lab10-01.sys"
push    1             ; dwErrorControl
push    3             ; dwStartType
push    1             ; dwServiceType
push    0F01FFh       ; dwDesiredAccess
push    offset ServiceName ; "Lab10-01"
push    offset ServiceName ; "Lab10-01"
push    edi           ; hSCManager
call    ds:CreateServiceA
mov     esi, eax
test    esi, esi
jnz     short loc_401069

pop     edi
add     esp, 1Ch
retn    10h
```

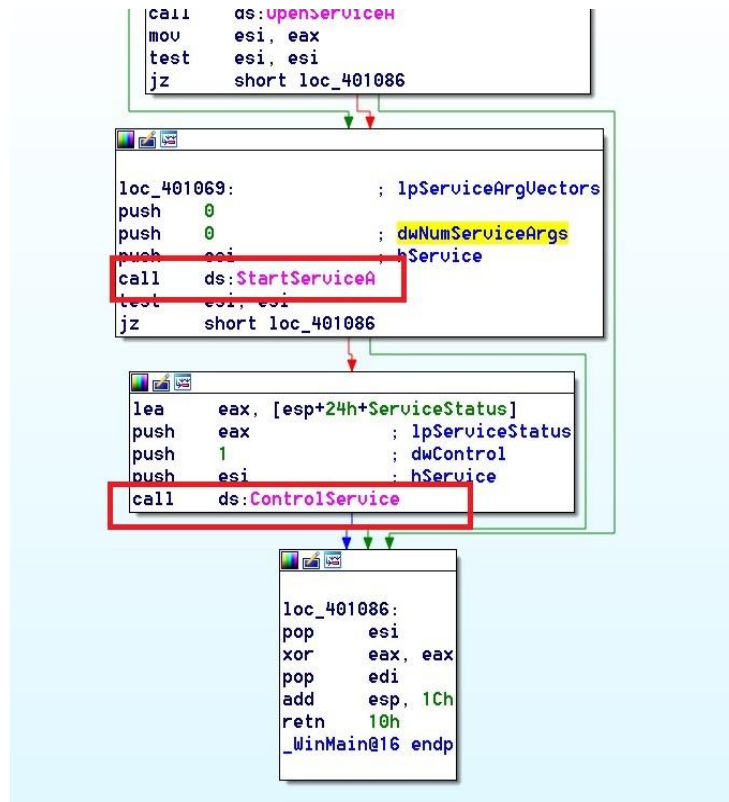
6. 如果 CreateServiceA 调用失败，代码会使用相同的服务名调用 OpenServiceA。如果因为服务已存在而导致调用 CreateServiceA 失败，这将打开 Lab10-01 服务的句柄

```
push    0             ; lpLoadOrderGroup
push    offset BinaryPathName ; "C:\\Windows\\System32\\Lab10-01.sys"
push    1             ; dwErrorControl
push    3             ; dwStartType
push    1             ; dwServiceType
push    0F01FFh       ; dwDesiredAccess
push    offset ServiceName ; "Lab10-01"
push    offset ServiceName ; "Lab10-01"
push    edi           ; hSCManager
call    ds:CreateServiceA
mov     esi, eax
test    esi, esi
jnz     short loc_401069

push    0F01FFh       ; dwDesiredAccess
push    offset ServiceName ; "Lab10-01"
push    edi           ; hSCManager
call    ds:OpenServiceA
mov     esi, eax
test    esi, esi
jz      short loc_401086

loc_401069:
push    0             ; lpServiceArgVectors
push    0             ; dwNumServiceArgs
push    esi           ; hService
call    ds:StartServiceA
test    esi, esi
jz      short loc_401086
```

7. 之后会调用 StartServiceA 启动服务，最后调用 ContorlService，在经过查阅资料以后可以发现 ContorlService 传入的第二个参数是 1，这个参数的意思就是会卸载驱动。



8. 接下来使用 IDA 查看一下 Lab10-01.sys 驱动文件，来检查它的 DriverEntry 函数，当打开驱动文件，移到它的入口点时，可以看到代码如下图所示，在 00010964 位置可以看到一个无条件跳转指令，跳转到了 sub_10906，也就是说这个驱动程序真正的入口点应该是在这个函数的位置。跟踪过去查看一下内容

```

00010959 ; NTSTATUS __stdcall DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
00010959 ; NTSTATUS __stdcall DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
00010959 DriverEntry public DriverEntry
00010959 DriverEntry proc near
00010959
00010959 DriverObject = dword ptr 8
00010959 RegistryPath = dword ptr 0Ch
00010959
00010959 mov     edi, edi
0001095B push    ebp
0001095C mov     ebp, esp
0001095E call    sub_10920
00010963 pop     ebp
00010964 jmp     sub_10906
00010964 DriverEntry endp
00010964

```

9. 但是仅从这里并不能看出来有什么，只看见把一个偏移地址放到了内存中去。

```
INIT:00010906 arg_0 = dword ptr 8
INIT:00010906
INIT:00010906 mov edi,edi
INIT:00010908 push ebp
INIT:00010909 mov ebp,esp
INIT:0001090B mov eax,[ebp+arg_0]
INIT:0001090E mov dword ptr [eax+34h],offset sub_10486
INIT:00010915 xor eax,eax
INIT:00010917 pop ebp
INIT:00010918 retn 8
INIT:00010918 sub_10906 endp
INIT:00010918 ; -----
INIT:0001091B align 10h
INIT:00010920
INIT:00010920 ----- SUBROUTINE -----
```

10. 使用 WinDbg 分析 Lab10-01.sys，来查看当调用 ControlService 卸载 Lab10-01.sys 时会发生什么事，用户空间可执行程序的代码加载 Lab10-01.sys，然后立即卸载它，如果我们再运行可执行程序之前使用内核调试器，因为此使驱动还未在内存中，所以我们还不能检查它，但是如果等待应用程序运行完成，那时候驱动又已经从内存中卸载了。

为了在 Lab10-01.sys 载入内存后，使用 WinDbg 分析它，在虚拟机中我们执行程序载入到 Windbg 中，使用如下命令可以在驱动加载以及卸载之间打上一个断点，在打断点之前使用 lm 命令查看目前的模块内容，然后启动程序直到断点命中，当断点命中之后可以观察到如下信息，然后启动程序直到断点命中时，在 WinDbg 展示了如下信息：

```
*** Error: Module load completed but symbols could not be loaded for image00400000
0:000> lm
start      end             module name
00400000 00407000  image00400000 C (no symbols)
77da0000 77e49000  ADVAPI32 (deferred)
77e50000 77ee3000  RPCRT4 (deferred)
77fc0000 77fd1000  Secur32 (deferred)
7c800000 7c91e000  kernel32 (deferred)
7c920000 7c9b6000  ntdll (export symbols) C:\WINDOWS\system32\ntdll.dll
0:000> bp 00401080
breakpoint 0 redefined
0:000> g
Breakpoint 0 hit
eax=0012ff1c ebx=7ffd7000 ecx=77dbfb8d edx=00000000 esi=00144008 edi=00144f50
eip=00401080 esp=0012ff08 ebp=0012ffc0 iopl=0         nv up ei pl zr na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
image00400000+0x1080.
```

11. 此时，因为 sys 文件已经成功载入内存，所以我们返回 win10 宿主机对 xp 虚拟机进行内核调试。环境配置的时候我这里是需要把端口号注册为 com_2，否则会出现一直连不上虚拟机的情况出现。然后我们就可以通过 !drvobj 命令

来获取驱动对象如下图所示。

可以观察到列表的设备为空，所以可以分析得到这个驱动没有供用户空间中应用程序访问的设备。

```
0: kd> !drvobj lab10-01
Driver object (8a02a460) is for:
  \Driver\Lab10-01

Driver Extension List: (id , addr)

Device Object list:
```

12. 我们获取了驱动对象以及其地址 8a02a460，使用 dt 命令查看驱动对象：可以找到卸载时调用的函数，函数的地址为 0xba672486

```
0: kd> dt _DRIVER_OBJECT 8a02a460
nt!_DRIVER_OBJECT
+0x000 Type : 0n4
+0x002 Size : 0n168
+0x004 DeviceObject : (null)
+0x008 Flags : 0x12
+0x00c DriverStart : 0xba672000 Void
+0x010 DriverSize : 0xe80
+0x014 DriverSection : 0x89efff00 Void
+0x018 DriverExtension : 0x8a02a508 _DRIVER_EXTENSION
+0x01c DriverName : _UNICODE_STRING "\Driver\Lab10-01"
+0x024 HardwareDatabase : 0x8067f260 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
+0x028 FastIoDispatch : (null)
+0x02c DriverInit : 0xba672959 long +0
+0x030 DriverStartIo : (null)
+0x034 DriverUnload : 0xba672486 void +0
+0x038 MajorFunction : [28] 0x804f55ce long nt!IoPInvalidDeviceRequest+0
```

13. 接着，在宿主机设置断点，并且恢复执行，随后在 xp 中同样使用 g 命令恢复执行，非常快程序就执行到了断点，此时 xp 卡死，我们退回到宿主机查看到了一条汇编代码，我们使用单步调试分析后续的指令

```
0: kd> bp 0xba672486
0: kd> g
Break instruction exception - code 80000003 (first chance)
Lab10_01+0x486:
ba672486 8bff          mov     edi,edi
```

14. 可以观察到程序调用了三次 CreateRegistryKey 函数，创建了注册表键然后两次调用 WriteRegistryValue 在两个地方将防火墙的值设置为 0

我们可以不在 windbg 中调试冗长的卸载函数，而是在 IDA pro 中完成分析，使用 lm 指令可以发现文件被加载到 0xba672000，卸载函数被加载到 0xba672486，通过 0xba672486 减去 0xba672000 得到偏移量 0x486，IDA Pro 基址是 0x00100000，然后可以通过在 IDA Pro 中地址为 0x00100486 处找到卸载函数。

```
1: KUD> lm
start end module_name
804d8000 806e6000 nt (pdb symbols) c:\mysymbol\ntkrpamp.pdb\7075F995A48A414F8F7BE9A1E0240F821\ntkrpamp.pdb
b04fb000 b050e000 PROCMON23 (deferred)
b06ae000 b06b8000 PROCEXP152 (deferred)
b082e000 b086ee00 HTTP (deferred)
b09ff000 b0a56600 srv (deferred)
b0c53000 b0c55a00 vmememctl (deferred)
ba4a0000 ba4a6f00 npf (deferred)
ba672000 ba672e80 Lab10_01 (no symbols)
```

问题解答：

1. 这个程序是否直接修改了注册表（使用 procmon 来检查）？

通过使用 procmon 检查的时候设置 Process Name is Lab10-01.exe 过滤后可以观察到下图中信息。

Time	Process	Operation	Path	Result	Details
0:11...	Lab10-01.exe	Load Image	C:\WINDOWS\system32\kernel32.dll	SUCCESS	Image Base: 0...
0:11...	Lab10-01.exe	RegOpenKey	HKLM\System\CurrentControlSet\...	SUCCESS	Desired Acces...
0:11...	Lab10-01.exe	RegQueryValue	HKLM\System\CurrentControlSet\...	SUCCESS	Type: REG_DWO...
0:11...	Lab10-01.exe	RegCloseKey	HKLM\System\CurrentControlSet\...	SUCCESS	
0:11...	Lab10-01.exe	Load Image	C:\WINDOWS\system32\advapi32.dll	SUCCESS	Image Base: 0...
0:11...	Lab10-01.exe	Load Image	C:\WINDOWS\system32\rpcrt4.dll	SUCCESS	Image Base: 0...
0:11...	Lab10-01.exe	Load Image	C:\WINDOWS\system32\secur32.dll	SUCCESS	Image Base: 0...
0:11...	Lab10-01.exe	RegOpenKey	HKLM\System\CurrentControlSet\...	SUCCESS	Desired Acces...
0:11...	Lab10-01.exe	RegQueryValue	HKLM\System\CurrentControlSet\...	SUCCESS	Type: REG_DWO...
0:11...	Lab10-01.exe	RegCloseKey	HKLM\System\CurrentControlSet\...	SUCCESS	
0:11...	Lab10-01.exe	RegOpenKey	HKLM\System\CurrentControlSet\...	SUCCESS	
0:11...	Lab10-01.exe	RegOpenKey	HKLM\Software\Microsoft\Window...	NAME NOT FOUND	Desired Acces...
0:11...	Lab10-01.exe	RegOpenKey	HKLM\Software\Microsoft\Window...	NAME NOT FOUND	Desired Acces...
0:11...	Lab10-01.exe	RegOpenKey	HKLM\Software\Microsoft\Window...	NAME NOT FOUND	Desired Acces...
0:11...	Lab10-01.exe	RegOpenKey	HKLM\System\CurrentControlSet\...	SUCCESS	Desired Acces...
0:11...	Lab10-01.exe	RegQueryValue	HKLM\System\CurrentControlSet\...	SUCCESS	Type: REG_DWO...
0:11...	Lab10-01.exe	RegQueryValue	HKLM\System\CurrentControlSet\...	SUCCESS	Type: REG_DWO...
0:11...	Lab10-01.exe	RegCloseKey	HKLM\System\CurrentControlSet\...	SUCCESS	
0:11...	Lab10-01.exe	RegOpenKey	HKLM\SOFTWARE\Microsoft\Window...	SUCCESS	Desired Acces...
0:11...	Lab10-01.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Window...	NAME NOT FOUND	Length: 144
0:11...	Lab10-01.exe	RegCloseKey	HKLM\SOFTWARE\Microsoft\Window...	SUCCESS	
0:11...	Lab10-01.exe	RegOpenKey	HKLM\Software\Microsoft\Window...	NAME NOT FOUND	Desired Acces...
0:11...	Lab10-01.exe	RegOpenKey	HKLM\Software\Microsoft\Window...	NAME NOT FOUND	Desired Acces...
0:11...	Lab10-01.exe	RegOpenKey	HKLM\Software\Microsoft\Window...	NAME NOT FOUND	Desired Acces...
0:11...	Lab10-01.exe	RegOpenKey	HKLM\Software\Microsoft\Rpc\Pa...	NAME NOT FOUND	Desired Acces...
0:11...	Lab10-01.exe	RegOpenKey	HKLM\Software\Microsoft\Rpc	SUCCESS	Desired Acces...
0:11...	Lab10-01.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Rpc\Ma...	NAME NOT FOUND	Length: 144
0:11...	Lab10-01.exe	RegCloseKey	HKLM\SOFTWARE\Microsoft\Rpc	SUCCESS	
0:11...	Lab10-01.exe	RegOpenKey	HKLM\Software\Microsoft\Window...	NAME NOT FOUND	Desired Acces...
0:11...	Lab10-01.exe	RegOpenKey	HKLM\Software\Policies\Microso...	NAME NOT FOUND	Desired Acces...
0:11...	Lab10-01.exe	RegOpenKey	HKLM\System\CurrentControlSet\...	SUCCESS	Desired Acces...
0:11...	Lab10-01.exe	RegQueryValue	HKLM\System\CurrentControlSet\...	NAME NOT FOUND	Length: 16
0:11...	Lab10-01.exe	RegCloseKey	HKLM\System\CurrentControlSet\...	SUCCESS	
0:11...	Lab10-01.exe	QueryNameInformationFile	C:\Documents and Settings\Admi...	BUFFER OVERFLOW	Name: \D
0:11...	Lab10-01.exe	QueryNameInformationFile	C:\Documents and Settings\Admi...	SUCCESS	Name: \Docume...
0:11...	Lab10-01.exe	RegSetValue	HKLM\SOFTWARE\Microsoft\Crypto...	SUCCESS	Type: REG_BIN...
0:11...	Lab10-01.exe	Thread Exit		SUCCESS	Thread ID: 16...
0:11...	Lab10-01.exe	Process Exit		SUCCESS	Exit Status:...
0:11...	Lab10-01.exe	CloseFile	C:\Documents and Settings\Admi...	SUCCESS	

再进行第二次过滤 Operation is RegSetValue，可以观察到唯一写注册表的调用是通过写键值 HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed 的

RegSetValue 调用。对注册表的一些间接修改通过调用 CreateServiceA 来完成，但这个程序也从内核对注册表进行了直接修改，这些修改却不能被 procmon 探测到。



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？

具体见上述分析

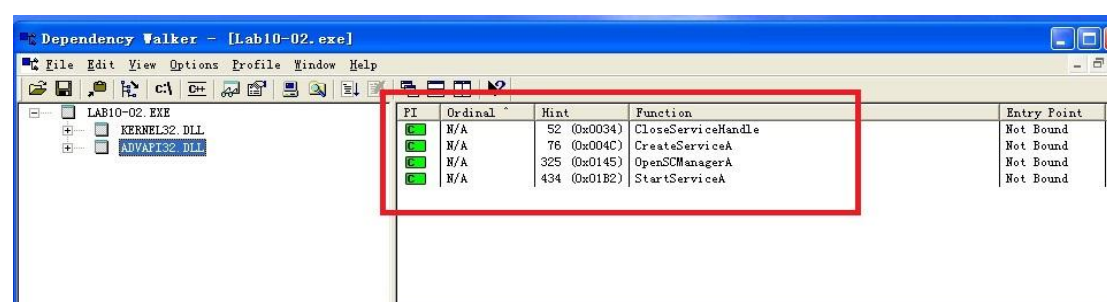
3. 这个程序做了些什么？

这个程序创建一个服务来加载驱动，然后和驱动代码会创建注册表键 \Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFirewall\StandardProfile 和 \Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFirewall\DomainProfile。在 Windows XP 系统中，设置这些键将禁用防火墙。

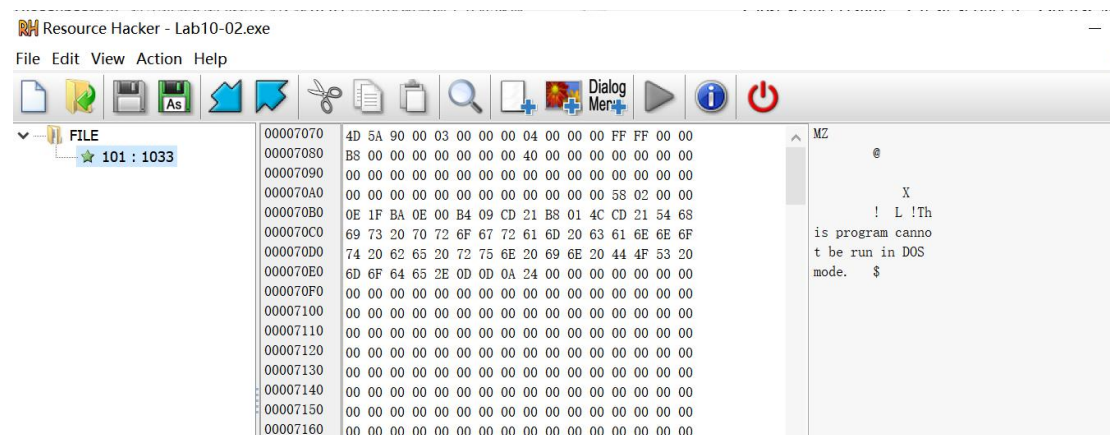
四、 Lab10-2

具体分析：

1. 使用 Dependency Walker 查看 Lab10-02.exe 的导入表，可以发现调用了 CloseServiceHandle、CreateServiceA、OpenSCManagerA 以及 StartServiceA，表示这个程序创建并启动一个服务的，这个程序还调用了 CreatFile 和 WriteFile，表明这个程序将在某个时间点写文件。另外我们还看到了 LoadResource 和 SizeOfResource 调用，表示这个程序对 Lab10-02.exe 的资源节做了某些处理



2. 使用 Resource Hacker 来检查资源节，可以发现资源节中包含了另外一个 PE 头部，这可能是 Lab10-02.exe 将要使用的另外一个恶意文件。



3. 通过 procmon 可以观察到程序在 C:\Windows\System32 目录下创建了一个文件，并且以这个文件作为一个可执行程序创建了服务，这个文件包含右操作系统加载的内核代码。

Lab10-02.exe	6488	QueryDirectory	C:\WINDOWS	NO MORE FILES	
Lab10-02.exe	6488	CloseFile	C:\WINDOWS	SUCCESS	
Lab10-02.exe	6488	CreateFile	C:\WINDOWS\AppPatch	SUCCESS	Desired Acces...
Lab10-02.exe	6488	QueryDirectory	C:\WINDOWS\AppPatch	SUCCESS	0: .., 1: .., ...
Lab10-02.exe	6488	QueryDirectory	C:\WINDOWS\AppPatch	NO MORE FILES	
Lab10-02.exe	6488	CloseFile	C:\WINDOWS\AppPatch	SUCCESS	
Lab10-02.exe	6488	CreateFile	C:\WINDOWS\system32	SUCCESS	Desired Acces...
Lab10-02.exe	6488	QueryDirectory	C:\WINDOWS\system32	SUCCESS	0: .., 1: .., ...
Lab10-02.exe	6488	QueryDirectory	C:\WINDOWS\system32	SUCCESS	0: drmlen.d...

4. 通过 IDA Pro 进行分析，首先，我们看一下这里的 NtQueryDirectoryFile 函数：

```
.text:00010486
.text:00010486      mov     edi, edi
.text:00010488      push   ebp
.text:00010489      mov     ebp, esp
.text:0001048B      push   esi
.text:0001048C      mov     esi, [ebp+FileInformation]
.text:0001048F      push   edi
.text:00010490      push   dword ptr [ebp+RestartScan] ; RestartScan
.text:00010493      push   [ebp+FileName] ; FileName
.text:00010496      push   dword ptr [ebp+ReturnSingleEntry] ; ReturnSingleEntry
.text:00010499      push   [ebp+FileInformationClass] ; FileInformationClass
.text:0001049C      push   [ebp+FileInformationLength] ; FileInformationLength
.text:0001049F      push   esi ; FileInformation
.text:000104A0      push   [ebp+IoStatusBlock] ; IoStatusBlock
.text:000104A3      push   [ebp+ApcContext] ; ApcContext
.text:000104A6      push   [ebp+ApcRoutine] ; ApcRoutine
.text:000104A9      push   [ebp+Event] ; Event
.text:000104AC      push   [ebp+FileHandle] ; FileHandle
.text:000104AF      call    NtQueryDirectoryFile
.text:000104B4      xnr     edi, edi
```

5. 进入 call 函数, 是一个跳转语句, 再双击进去就是正牌的 NtQueryDirectoryFile 了; 很显然, 上面程序中构造的函数, 是想在伪装什么;

```
.text:00010514  
.text:00010514 jmp ds:imp_NtQueryDirectoryFile  
.text:00010514 NtQueryDirectoryFile endp
```

6. 继续往下看, 我们假设它正确跳转且返回了; 继续往下后, 它先是比较了 FileInformationClass 是否等于 3, 我们之前说过的, 它正常来讲应该是等于 3 的; 第二部是检查 NtQueryDirectoryFile() 是否正确执行, 在上面参数都正确构造, 也正确跳转的情况下, 我们可以合理假设它正确的执行且返回了; 再然后就是关于 ReturnSingleEntry 字段了, 关于该字段的描述如下: 如果只返回一个条目, 设置为 TRUE, 否则设置为 FALSE。如果该参数为真, NtQueryDirectoryFile 只返回找到的第一个条目。

```
.text:000104AC push [ebp+FileHandle] ; FileHandle  
.text:000104AF call NtQueryDirectoryFile  
.text:000104B4 xor edi, edi  
.text:000104B6 cmp [ebp+FileInformationClass], 3  
.text:000104BA mov dword ptr [ebp+RestartScan], eax  
.text:000104BD jnz short loc_10505  
.text:000104BF test eax, eax  
.text:000104C1 jl short loc_10505  
.text:000104C3 cmp [ebp+ReturnSingleEntry], 0  
.text:000104C7 jnz short loc_10505  
.text:000104C9 push ebx  
.text:000104CA
```

7. 这里我们要先了解一下 NtQueryDirectoryFile 返回的东西是啥: NtQueryDirectoryFile 例程返回 STATUS_SUCCESS 或适当的错误状态。注意, 可以返回的错误状态值集是特定于文件系统的。NtQueryDirectoryFile 还返回 IoStatusBlock 的 Information 成员中实际写入给定 FileInformation 缓冲区的字节数。其实返回的是 FileInforMation 结构, 而该结构在 FileInformationClass 字段中有定义, 经过继续查询, 我们知道是由一系列的 FILE_BOTH_DIR_INFORMATION 结构组成; 所以, 上述的三个条件都是在检查我们伪装的这个 NtQueryDirectoryFile() 是否正确运行;

8. 继续往下看，很显然这是一个循环，而且在没有找到循环步长的情况下，这是一个死循环；重点是我们要关注一下 RtlCompareMemory()函数比较了什么；

```

.text:000104CA loc_104CA:
.text:000104CA
.text:000104CA
.text:000104CC
.text:000104D1
.text:000104D4
.text:000104D5
.text:000104D7
.text:000104D9
.text:000104E0
.text:000104E2
.text:000104E4
.text:000104E6
.text:000104E8
.text:000104EA
.text:000104EC
.text:000104EE
.text:000104F0
.text:000104F2
.text:000104F2
.text:000104F2 loc_104F2:
.text:000104F4
.text:000104F4 loc_104F4:
.text:000104F4
.text:000104F6
.text:000104F8
.text:000104FA
.text:000104FC
.text:000104FE
.text:00010500
.text:00010500 loc_10500:
.text:00010502
.text:00010504

        push    8
        push    offset word_1051A ; Source2
        lea     eax, [esi+5Eh]
        push    eax ; Source1
        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

        add     [edi], eax ; CODE XREF: sub_10486+66↑j
        mov     eax, [esi] ; CODE XREF: sub_10486+5A↑j
        test    eax, eax ; sub_10486+60↑j ...
        jz      short loc_10504
        test    bl, bl
        jnz     short loc_10508
        mov     edi, esi
        add     esi, eax ; CODE XREF: sub_10486+76↑j
        jmp     short loc_104CA

```

9. 经过向上夙愿，我们找到 FileInformation 字段，我们知道这是一个叫做 FILE_BOTH_DIR_INFORMATION 的结构体，经过计算，我们得到的是 FileName 字段；至于 word_1051A 字段，是内存没定义值，且交叉引用也没有找到赋值语句，所以我们可以合理猜测是一个运行时的值，具体是什么，我们暂时可以画个问号，先看如果比较成功应该会怎么样；

10. 继续往下走，假设返回值是 8，也就是正确返回了，进行下一步；

```

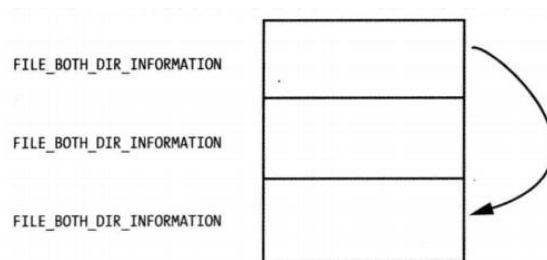
.text:000104D5
.text:000104D7
.text:000104D9
.text:000104DB
.text:000104DD
.text:000104DE
.text:000104E0
.text:000104E2
.text:000104E4
.text:000104E6
.text:000104E8
.text:000104EA
.text:000104EC
.text:000104EE
.text:000104F0
.text:000104F2
.text:000104F4
.text:000104F6
.text:000104F8
.text:000104FA
.text:000104FC
.text:000104FE
.text:00010500
.text:00010502
.text:00010504

        push    8
        push    offset word_1051A ; Source2
        lea     eax, [esi+5Eh]
        push    eax ; Source1
        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

```

11. 总结来说例如下图所示隐藏：每个文件对应一个 FILE_BOTH_DIR_INFORMATION 结构，通常情况下，第一个结构会指向第二

个结构，第二个结构会指向第三个结构，因此隐藏了中间的结构体，这种技巧保证了每个以 `MIwx` 开头的文件会被跳过，并且从目录列表中隐藏。



12. 经过分析我们知道，这个程序构造了一个伪装的 `NtQueryDirectoryFile` 程序，并且将其隐藏；并没有创建设备，也没有向驱动对象添加什么操作函数；

习题解答：

1. 这个程序创建文件了吗？它创建了什么文件？

这个程序创建了文件，可以使用 `procmon` 或其他动态监视工具来查看文件创建，但是因为文件被隐藏，所以在硬盘上看不到它。

2. 这个程序有内核组件吗？

这个程序拥有一个内核模块，这个内核模块被存储在这个文件的资源节中，然后写入硬盘并作为一个服务加载到内核。

3. 这个程序做了些什么？

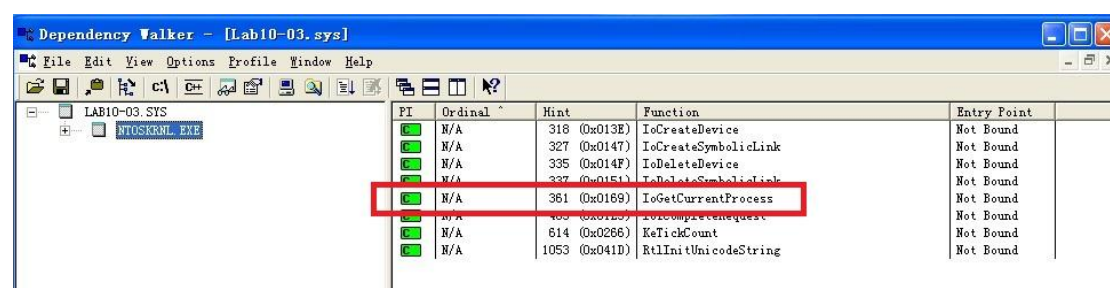
这个程序是一个被设计来隐藏文件的 `Rootkit`，它使用 `SSDT` 挂钩来覆盖 `NtQueryDirectoryFile` 的入口，它会隐藏目录列表中任何以 `MIwx` 开头的文件

五、 Lab7-3

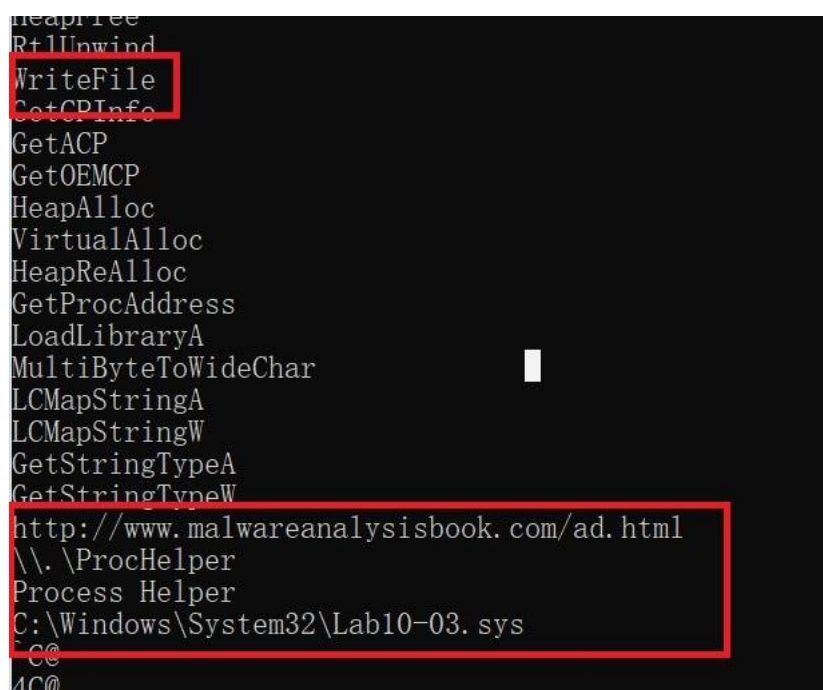
具体分析：

1. 使用 `Dependency Walker` 查看 `Lab10-02.exe` 的导入表，其中 `IoGetCurrentProcess` 导入函数表明这个驱动或者修改正在运行进程或者需要关于

进程的信息。



2. 通过 Strings 查看字符串信息，可以看到这个 exe 文件还具有写文件的能力，并且这个 exe 文件的目的是为了显示一个广告页面而产生的。



3. 我们将驱动文件复制到 C:\Windows\System32，双击可执行程序运行它，我们看到一个弹出广告，并且我们注意到大约 30s 后程序再次弹出广告，并且每隔 30s 执行一次，当打开任务管理器试图终止运行进程时我们看到程序没有被列出，另外 Process Explorer 中也没有被列出。

4. 使用 IDA Pro 加载 exe 文件，在主函数中分析其调用的函数，WinMain 函数的功能大体上可以分为两部分，第一部分是 OpenSCManager、CreateService、StartService、CloseServiceHandle、CreateFile、DeviceIoControl 函数，包括创建


```

.text:0040108C loc_40108C: ; CODE XREF: WinMain(x,x,x,x)+7E↑j
.text:0040108C lea ecx, [esp+2Ch+BytesReturned]
.text:00401090 push 0 ; lpOverlapped
.text:00401092 push ecx ; lpBytesReturned
.text:00401093 push 0 ; nOutBufferSize
.text:00401095 push 0 ; lpOutBuffer
.text:00401097 push 0 ; nInBufferSize
.text:00401099 push 0 ; lpInBuffer
.text:0040109B push 0ABCDEF01h ; dwIoControlCode
.text:004010A0 push eax ; hDevice
.text:004010A1 call ds:DeviceIoControl

```

7. 访问网站并且在两次调用之间 sleep 30 秒

```

.text:004010D5 jz short loc_4010E0
.text:004010D5 lea eax, [esp+30h+pvarg]
.text:004010D9 push eax ; pvarg
.text:004010DA call ds:VariantInit
.text:004010E0 push offset psz ; "http://www.malwareanalysisbook.com/ad.h"
.text:004010E5 mov [esp+34h+uar_10], 3
.text:004010EC mov [esp+34h+uar_8], 1
.text:004010F4 call ds:SysAllocString
.text:004010FA mov edi, ds:Sleep
.text:00401100 mov esi, eax

```

8. 我们用 IDA Pro 打开内核文件，可以看到他调用了 IoCreateDevice 函数，创建了一个\\Device\\ProHelper 的设备

```

INIT:0001071A mov eax, 0
INIT:0001071F push offset aDeviceProchelp ; "\\Device\\ProHelper"
INIT:00010722 lea eax, [ebp+DestinationString]
INIT:00010723 push eax ; DestinationString
INIT:00010725 call edi ; RtlInitUnicodeString
INIT:00010728 mov esi, [ebp+DriverObject]
INIT:0001072B lea eax, [ebp+DeviceObject]
INIT:0001072B push eax ; DeviceObject
INIT:0001072C push 0 ; Exclusive
INIT:0001072E push 100h ; DeviceCharacteristics
INIT:00010733 push 22h ; DeviceType
INIT:00010735 lea eax, [ebp+DestinationString]
INIT:00010738 push eax ; DeviceName
INIT:00010739 push 0 ; DeviceExtensionSize
INIT:0001073B push esi ; DriverObject
INIT:0001073C call ds:IoCreateDevice
INIT:00010742 test eax, eax
INIT:00010744 jl short loc_10789

```

9. 随后函数调用 IoCreateSymbolicLink 时，创建一个名为\\DosDevices\\ProHelper 的符号链接，来供用户态应用程序访问

```

INIT:0001074B mov [esi+30h], 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

```

10. 使用 Windbg 进行内核调试，在虚拟机运行恶意代码将内核驱动加载到内存，通过前面的分析我们能知道设备对象在\\Device\\ProcHelper，所以我们来启动它，为了找到在设备对象中被执行的函数，我们必须找到他的设备对象，还是使用！devobj 来完成

```
nt!RtlpBreakWithStatusInstruction:
8052c708 cc          int     3
0: kd> !devobj ProcHelper
Device object (8a0417c8) is for:
  ProcHelper \Driver\Process Helper DriverObject 89fa2da0
Current Irp 00000000 RefCount 1 Type 00000022 Flags 00000040
SecurityDescriptor e1412b90 DevExt 00000000 DevObjExt 8a041880
ExtensionFlags (000000000)
Characteristics (0x00000100) FILE_DEVICE_SECURE_OPEN
Device queue is not busy.
```

11. DriverObject 包含所有函数的指针，当用户空间的程序访问设备对象时调用这些函数，DriverObject 存储在一个叫做 DRIVER_OBJECT 的数据结构中，我们可以使用 dt 命令查看标注的驱动对象：这段代码包含了几个需要注意的函数指针，可以看到 DriverInit、DriverEntry、DriverUnload 在实验一中被重点分析，当驱动卸载时被调用，可以在 IDA Pro 中看到它删除了符号链表和 DriverEntry 创建的设备

```
0: kd> dt nt!_DRIVER_OBJECT 89fa2da0
+0x000 Type           : 0n4
+0x002 Size           : 0n168
+0x004 DeviceObject   : 0x8a0417c8 _DEVICE_OBJECT
+0x008 Flags          : 0x12
+0x00c DriverStart    : 0xba6b0000 Void
+0x010 DriverSize     : 0xe00
+0x014 DriverSection  : 0x89f90700 Void
+0x018 DriverExtension : 0x89fa2e48 _DRIVER_EXTENSION
+0x01c DriverName     : _UNICODE_STRING "\Driver\Process Helper"
+0x024 HardwareDatabase : 0x8067f260 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
+0x028 FastIoDispatch : (null)
+0x02c DriverInit     : 0xba6b07cd long +ffffffffffba6b07cd
+0x030 DriverStartIo  : (null)
+0x034 DriverUnload   : 0xba6b062a void +ffffffffffba6b062a
+0x038 MajorFunction  : [28] 0xba6b0606 long +ffffffffffba6b0606
```

12. 接下来可以分析主函数表，最具有意义的驱动通常在这完成，我们可以通过 dd 命令查看主函数中表项，表中的每一项表示不同类型的请求，其中大部分都是 804f55ce，这个项表示了驱动不能处理的一个请求类型。


```

0: kd> dd 89fa2da0+0x38 11c
ReadVirtual: 89fa2dd8 not properly sign extended
89fa2dd8  ba6b0606 804f55ce ba6b0606 804f55ce
89fa2de8  804f55ce 804f55ce 804f55ce 804f55ce
89fa2df8  804f55ce 804f55ce 804f55ce 804f55ce
89fa2e08  804f55ce 804f55ce ba6b0666 804f55ce
89fa2e18  804f55ce 804f55ce 804f55ce 804f55ce
89fa2e28  804f55ce 804f55ce 804f55ce 804f55ce
89fa2e38  804f55ce 804f55ce 804f55ce 804f55ce

```

13. 我们可以根据他的表项查找到对应的函数是什么，使用 x /D 命令，并选择以 I 开头的内容可以看到对应地址的内容为 IopInvalidDeviceRequest，这意味他处理驱动无法处理的非法请求

```

0: kd> x /D nt!i*
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

805785f8      nt!IopFreeDCB (_IopFreeDCB@4)
8053f720      nt!IoFreeMapRegisters (_IoFreeMapRegisters@12)
804f069a      nt!IoGetAttachedDeviceReference (_IoGetAttachedDeviceReference@4)
8055bd24      nt!IopUniqueDeviceObjectNumber = <no type information>
804f55ce      nt!IopInvalidDeviceRequest (_IopInvalidDeviceRequest@8)
805762bc      nt!IoQueryFileDosDeviceName (_IoQueryFileDosDeviceName@8)
8059f06a      nt!IoIrpInitialize (_IoIrpInitialize@0)
8058536a      nt!IoInitializeRemoveLockEx (_IoInitializeRemoveLockEx@20)
8058abce      nt!IoReportTargetDeviceChange (_IoReportTargetDeviceChange@8)
80595bdc      nt!IoDestroyDeviceNode (_IoDestroyDeviceNode@4)

```

14. 返回 IDA Pro 查看这个函数：可以看到 DeviceIoControl 操作了当前进程的 PEB，这个函数做的第一件事就是调用函数 IoGetCurrentProcess，这个函数会返回 DeviceIoControl 的 EPROCESS 结构，然后这个函数在接下来访问偏移量为 0x88 处的数据，再然后访问偏移量为 0x8c 处的下一个 DWORD

```

PAGE:00010666 Irp          = dword ptr 0Ch
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:IoCompleteRequest
PAGE:0001069B      xor     eax, eax
PAGE:0001069D      pop     ebp
PAGE:0001069E      retn    8
PAGE:0001069F      add     esp, 10h

```

15. 使用 dt 命令发现存储在 PEB 结构偏移量 0x88 和 0x8c 的 LIST_ENTRY

```
0: kd> dt nt!_EPROCESS
+0x000 Pcb : _KPROCESS
+0x06c ProcessLock : _EX_PUSH_LOCK
+0x070 CreateTime : _LARGE_INTEGER
+0x078 ExitTime : _LARGE_INTEGER
+0x080 RundownProtect : _EX_RUNDOWN_REF
+0x084 UniqueProcessId : Ptr32 Void
+0x088 ActiveProcessLinks : _LIST_ENTRY
+0x090 QuotaUsage : [3] Uint4B
+0x09c QuotaPeak : [3] Uint4B
+0x0a8 CommitCharge : Uint4B
+0x0ac PeakVirtualSize : Uint4B
```

16. LIST_ENTRY 和操作系统课程中使用双向链表很像，他就是一个包含 prev 和 next 的双向链表，sys 呈现首先获取列表中指向下一个结构的指针，然后获取列表中指向前一个的指针，再覆盖下一项的 prev 指针使其指向前一项，在此之前，下一项的 prev 指针指向覆盖项，也就是说，整个链表结构跳过了中间的 LIST_ENTRY 结构

17. 在操作系统进程运行时，每个进程都有一个指针指向它前一个进程和后一个进程的指针，然后在本恶意代码中，他通过修改 LIST_ENTRY 使其跳过 Lab10-03 进程让我们不能通过控制面板或者 process explorer 查看到该进程。

习题解答：

1. 这个程序做了什么？

用户态程序加载驱动，然后每隔 30s 就弹出一个广告，这个驱动通过从系统链表中摘除进程环境块（PEB），来隐藏进程。

2. 一旦程序运行，该怎么停止他？

一旦程序运行，除了重启以外，没有任何一种方法可以轻易停止它。

3. 他的内核组件做了什么？

为了对用户隐藏进程，内核组件负责响应，从进程链表中摘除进程的 DeviceIoControl 请求。

六、 Yara 检测规则编写

1. 核心思想:

通过上面具体分析, 结合该.dll 和功能性函数以及文件大小和 PE 文件特征进行综合编写 Yara 检测规则并进行验证

2. Yara 规则:

rule yara_1_exe{

meta:

description = "匹配 Lab010-01.exe 类型的恶意代码"

author="nzq"

date="2022-11-20"

strings:

\$string1 = "REGWRITERAPP" nocase

\$sys1 = "C:\\Windows\\System32\\Lab10-01.sys"

\$func1 = "GetLastActivePopup"

\$func2 = "OpenServiceA"

\$func3 = "StartServiceA"

\$func4 = "OpenSCManagerA"

\$func5 = "GetCurrentProcess"

\$func6 = "GetOEMCP"

\$string2 = "RegWriterApp Version 1.0"

condition:

filesize< 50KB and //大小

uint16(0) == 0x5A4D and uint16(uint16(0x3C))==0x00004550 and //PE

6 of them //特征字符串或函数或 DLL

}

rule yara_1_sys{

meta:

description = "匹配 Lab010-01.sys 类型的恶意代码"

author="nzq"

date="2022-11-20"

strings:

\$string1 = "\\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\
WindowsFirewall\\DomainProfile"

```

$string2 = "6.1.7600.16385"
$string3 = "c:\\winddk\\7600.16385.1\\src\\general\\regwriter\\wdm\\sys\\
           objfre_wxp_x86\\i386\\sioclt.pdb"
$string4 = "\\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft"
$string5 = "\\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\
           WindowsFirewall\\DomainProfile"

condition:
    filesize< 50KB and //大小
    uint16(0) == 0x5A4D and uint16(uint16(0x3C))==0x00004550 and //PE
    1 of them //特征字符串或函数或 DLL
}

```

rule yara_2{

```

meta:
    description = "匹配 Lab010-02.exe 类型的恶意代码"
    author="nzq"
    date="2022-11-20"

strings:
    $e = "NtQueryDirectoryFile"
    $f = "KeServiceDescriptorTable"
    $g = "c:\\winddk\\7600.16385.1\\src\\general\\rootkit\\wdm\\sys\\
           objfre_wxp_x86\\i386\\sioclt.pdb"
    $h = "C:\\Windows\\System32\\Mlwx486.sys"
    $i = "486 WS Driver"

condition:
    filesize< 50KB and //大小
    uint16(0) == 0x5A4D and uint16(uint16(0x3C))==0x00004550 and //PE
    3 of them //特征字符串或函数或 DLL
}

```

rule yara_3_exe{

```

meta:
    description = "匹配 Lab010-02.exe 类型的恶意代码"
    author="nzq"
    date="2022-11-20"

strings:
    $string1 = "C:\\Windows\\System32\\Lab10-03.sys"
    $string2 = "Process Helper"

```

```

$string3 = "\\ProcHelper"
$http = "http://www.malwareanalysisbook.com/ad.html"
$func1 = "GetOEMCP"
$func2 = "GetCurrentProcess"
$dll1 = "OLEAUT32.dll"
$dll2 = "ole32.dll"

condition:
    filesize< 50KB and //大小
    uint16(0) == 0x5A4D and uint16(uint16(0x3C))==0x00004550 and //PE
    6 of them //特征字符串或函数或 DLL
}

```

rule yara_3_sys{

```

meta:
    description = "匹配 Lab010-03.sys 类型的恶意代码"
    author="nzq"
    date="2022-11-20"
strings:
    $string1 = "ntoskrnl.exe"
    $string2 = "\\DosDevices\\ProcHelper"
    $sys1 = "Lab10-03.sys"
    $func1 = "IoCreateDevice"
    $func2 = "IoCreateSymbolicLink"
    $func3 = "IoGetCurrentProcess"
    $func4 = "RtlInitUnicodeString"
    $func5 = "IoDeleteSymbolicLink"

condition:
    filesize< 50KB and //大小
    uint16(0) == 0x5A4D and uint16(uint16(0x3C))==0x00004550 and //PE
    6 of them //特征字符串或函数或 DLL
}

```

3. 运行结果:

```

D:\Malware\Yara\yara>yara64.exe -r find.yar Lab10
yara_1_exe Lab10\Lab10-01.exe
yara_2 Lab10\Lab10-02.exe
yara_3_exe Lab10\Lab10-03.exe
yara_3_sys Lab10\Lab10-03.sys
yara_1_sys Lab10\Lab10-01.sys
D:\Malware\Yara\yara>_

```

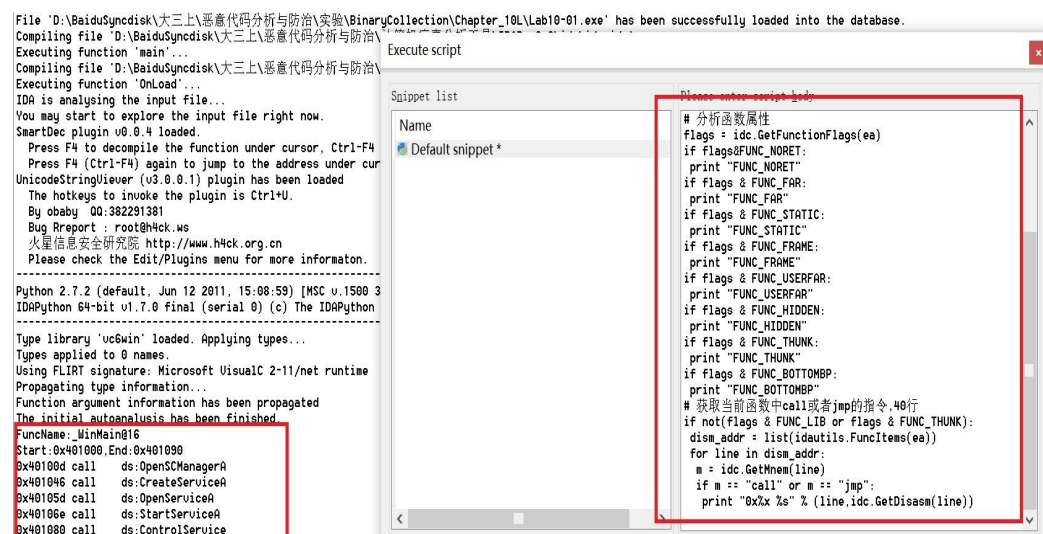
七、 IDA Python 脚本编写

1. 功能：获取光标所在函数的函数名、开始地址和结束地址，分析函数 FUNC_FAR、FUNC_USERFAR、FUNC_LIB（库代码）、FUNC_STATIC（静态函数）、FUNC_FRAME、FUNC_BOTTOMBP、FUNC_HIDDEN 和 FUNC_THUNK 标志，获取当前函数中 jmp 或者 call 指令。

2. 代码：

```
1.  import idutils
2.  ea=idc.ScreenEA()
3.  funcName=idc.GetFunctionName(ea)
4.  func=idaapi.get_func(ea)
5.  # 获取函数名
6.  print("FuncName:%s"%funcName)
7.  # 获取函数开始地址和结束地址
8.  print "Start:0x%x,End:0x%x" % (func.startEA,func.endEA)
9.  # 分析函数属性
10. flags = idc.GetFunctionFlags(ea)
11. if flags&FUNC_NORET:
12.     print "FUNC_NORET"
13. if flags & FUNC_FAR:
14.     print "FUNC_FAR"
15. if flags & FUNC_STATIC:
16.     print "FUNC_STATIC"
17. if flags & FUNC_FRAME:
18.     print "FUNC_FRAME"
19. if flags & FUNC_USERFAR:
20.     print "FUNC_USERFAR"
21. if flags & FUNC_HIDDEN:
22.     print "FUNC_HIDDEN"
23. if flags & FUNC_THUNK:
24.     print "FUNC_THUNK"
25. # 获取当前函数中 call 或者 jmp 的指令
26. if not(flags & FUNC_LIB or flags & FUNC_THUNK):
27.     dism_addr = list(idutils.FuncItems(ea))
28.     for line in dism_addr:
29.         m = idc.GetMnem(line)
30.         if m == "call" or m == "jmp":
31.             print "0x%x %s" % (line,idc.GetDisasm(line))
```

3. 执行结果：



```
File 'D:\BaiduSyncdisk\大三上\恶意代码分析与防治\实验\BinaryCollection\Chapter_10\Lab10-01.exe' has been successfully loaded into the database.
Compiling file 'D:\BaiduSyncdisk\大三上\恶意代码分析与防治\...
Executing function 'main'...
Compiling file 'D:\BaiduSyncdisk\大三上\恶意代码分析与防治\...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
SmartDec plugin v0.0.4 loaded.
Press F4 to decompile the function under cursor, Ctrl-F4
Press F4 (Ctrl-F4) again to jump to the address under cur
UnicodeStringViewer (v3.0.0.1) plugin has been loaded
The hotkeys to invoke the plugin is Ctrl+U.
By obaby QQ:382291381
Bug Report : root@h4ck.ws
火星信息安全研究院 http://www.h4ck.org.cn
Please check the Edit/Plugins menu for more informaton.
-----
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 3
IDAPython 64-bit v1.7.0 final (serial 0) (c) The IDAPython
-----
Type library 'uc6win' loaded. Applying types...
Types applied to 0 names.
Using FLIRT signature: Microsoft VisualC 2-11/net runtime
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
FuncName: WinMain@16
Start:0x401000,End:0x401090
0x40100d call ds:OpenSCManagerA
0x401046 call ds:CreateServiceA
0x40105d call ds:OpenServiceA
0x40106e call ds:StartServiceA
0x401080 call ds:ControlService
```

```
Please enter script code.
# 分析函数属性
flags = idc.GetFunctionFlags(ea)
if flags & FUNC_NORET:
    print "FUNC_NORET"
if flags & FUNC_FAR:
    print "FUNC_FAR"
if flags & FUNC_STATIC:
    print "FUNC_STATIC"
if flags & FUNC_FRAME:
    print "FUNC_FRAME"
if flags & FUNC_USERFAR:
    print "FUNC_USERFAR"
if flags & FUNC_HIDDEN:
    print "FUNC_HIDDEN"
if flags & FUNC_THUNK:
    print "FUNC_THUNK"
if flags & FUNC_BOTTOMBP:
    print "FUNC_BOTTOMBP"
# 获取当前函数中call或者jmp的指令,40行
if not(flags & FUNC_LIB or flags & FUNC_THUNK):
    dism_addr = list(idutils.FuncItems(ea))
    for line in dism_addr:
        m = idc.GetMnem(line)
        if m == "call" or m == "jmp":
            print "0x%x %s" % (line, idc.GetDisasm(line))
```

八、 心得体会

通过此次实验对于恶意代码调试又上了一个台阶，学会了综合基本静态调试（例如 Strings、Dependency Walker 等工具查看字符串和导入函数等信息）+基本动态分析（例如使用 procmon）+IDA Pro 动态调试+Windbg 内核调试