恶意代码分析与防治课程实验报告

实验八：windbg 使用与分析

学　　院　网络空间安全学院

专　　业　信息安全

学　　号　2111033

姓　　名　艾明旭

班　　级　信息安全一班

一、实验目的

使用 Windbg 分析恶意代码的目的是为了深入了解和研究恶意代码的行为、功能和潜在威胁，以及为保护系统和网络安全提供支持。具体目的包括：

识别恶意行为：通过分析恶意代码，可以确定它在系统中的具体行为，例如文件的创建、注册表的修改、网络通信等。这有助于提醒安全团队防范和检测类似行为，加强系统安全措施。

漏洞分析：恶意代码常常利用软件漏洞来入侵系统。通过 Windbg 的调试功能，可以深入分析恶意代码如何利用漏洞，了解攻击者的技术手段和攻击路径，帮助软件开发者修补漏洞以增强系统安全性。反制策略：分析恶意代码的目的还包括寻找阻止、检测和清除恶意代码的有效策略。通过分析恶意代码的执行过程和技术，可以制定相应的安全策略和对策，提高系统的安全性和抵御能力。收集情报：研究恶意代码有助于收集关于攻击者、攻击组织和攻击活动的情报。通过对恶意代码的分析，可以获取攻击者的行为模式、工具和攻击方法，为网络安全团队提供更深入的情报支持。对抗恶意软件：最后，通过分析恶意代码，可以揭示恶意软件的设计原理和技术手段，为开发反恶意软件工具提供依据。这有助于改进防火墙、杀毒软件等安全产品，提高对抗恶意软件的能力。综上所述，使用 Windbg 分析恶意代码的目的是为了加深对恶意代码的理解，强化系统安全，寻找防御和对抗策略，并为网络安全领域的进一步研究和发展提供信息和情报支持。

二、实验原理

Windbg 是一种强大的调试工具，常用于恶意代码程序分析。它的原理可以简述如下：

调试器功能：Windbg 是微软的用户模式和内核模式调试器，可以在 Windows 操作系统中跟踪和调试程序的执行过程。

调试环境：使用 Windbg 时，恶意代码通常在虚拟机或实验环境中运行。这样可以隔离恶意代码对真实系统的影响，并提供更安全的分析环境。

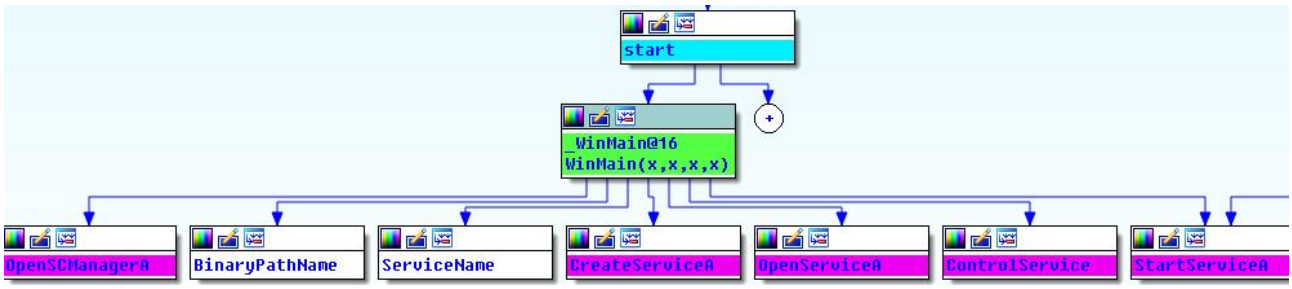
恶意代码静态分析：首先，通过将恶意代码加载到 Windbg 中，可以进行静态分析。这包括检查恶意代码的文件结构、导入的库、字符串和常量等信息，有助于了解恶意代码的基本行为和功能。

动态分析：接下来，通过以调试模式运行恶意代码，可以在 Windbg 中监控其执行过程。可以设置断点、观察寄存器和内存状态，跟踪代码执行路径，以及分析恶意代码的行为和可能的漏洞。

调试命令和扩展：Windbg 提供了丰富的调试命令和扩展，用于深入分析恶意代码。可以使用这些命令来查找漏洞、识别恶意行为、追踪函数调用堆栈等。

总之，Windbg 通过提供强大的调试功能和灵活的分析手段，支持恶意代码的静态和动态分析，帮助安全研究人员深入了解恶意代码的运行机制和潜在威胁。

三、实验过程



首先在 ida pro 当中打开相应的页面，在相关的页面下面我们可以看到这个进程的大致内容和运行逻辑。

反汇编代码如下：

```
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
  int result; // eax@1
  int v5; // edi@1
  SC_HANDLE v6; // esi@2
  struct _SERVICE_STATUS ServiceStatus; // [sp+4h] [bp-1Ch]@5

  result = (int)OpenSCManagerA(0, 0, 0xF003Fu);
  v5 = result;
  if ( result )
  {
    v6 = CreateServiceA(
           (SC_HANDLE)result,
           ServiceName,
           ServiceName,
           0xF01FFu,
           1u,
           3u,
           1u,
           BinaryPathName,
           0,
           0,
           0,
           0,
           0);
    if ( v6 || (v6 = OpenServiceA((SC_HANDLE)v5, ServiceName, 0xF01FFu)) != 0 )
    {
      StartServiceA(v6, 0, 0);
      if ( v6 )
        ControlService(v6, 1u, &ServiceStatus);
    }
    result = 0;
  }
  return result;
```

运行后查看：

```
.rdata:00··· 0000000F    C    runtime error
.rdata:00··· 0000000E    C    TLOSS error\r\n
.rdata:00··· 0000000D    C    SING error\r\n
.rdata:00··· 0000000F    C    DOMAIN error\r\n
.rdata:00··· 00000025    C    R6028\r\n- unable to initialize heap\r\n
.rdata:00··· 00000035    C    R6027\r\n- not enough space for lowio initialization\r\n
.rdata:00··· 00000035    C    R6026\r\n- not enough space for stdio initialization\r\n
.rdata:00··· 00000026    C    R6025\r\n- pure virtual function call\r\n
.rdata:00··· 00000035    C    R6024\r\n- not enough space for _onexit/atexit table\r\n
.rdata:00··· 00000029    C    R6019\r\n- unable to open console device\r\n
.rdata:00··· 00000021    C    R6018\r\n- unexpected heap error\r\n
.rdata:00··· 0000002D    C    R6017\r\n- unexpected multithread lock error\r\n
.rdata:00··· 0000002C    C    R6016\r\n- not enough space for thread data\r\n
.rdata:00··· 00000021    C    \r\nabnormal program termination\r\n
.rdata:00··· 0000002C    C    R6009\r\n- not enough space for environment\r\n
.rdata:00··· 0000002A    C    R6008\r\n- not enough space for arguments\r\n
.rdata:00··· 00000025    C    R6002\r\n- floating point not loaded\r\n
.rdata:00··· 00000025    C    Microsoft Visual C++ Runtime Library
.rdata:00··· 0000001A    C    Runtime Error!\n\nProgram:
.rdata:00··· 00000017    C    <program name unknown>
.rdata:00··· 00000013    C    GetLastActivePopup
.rdata:00··· 00000010    C    GetActiveWindow
.rdata:00··· 0000000C    C    MessageBoxA
.rdata:00··· 0000000B    C    user32.dll
.rdata:00··· 0000000D    C    ADVAPI32.dll
.rdata:00··· 0000000D    C    KERNEL32.dll
.data:004··· 00000009    C    Lab10-01
.data:004··· 00000021    C    C:\\Windows\\System32\\Lab10-01.sys
.data:004··· 00000006    C      y  !
```

服务的交互，是由 services.exe 完成的！services.exe 是微软 Windows 操作系统的一部分。用于管理启动和停止服务。该进程也会处理在计算机启动和关机时运行的服务。这个程序对你系统的正常运行是非常重要的。终止进程后会重启。 先反汇编看下，可以知道是在创建服务并启动！然后再看 strings，基本判断是通过驱动方式来加载恶意代码。

```
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
```

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

```
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
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
```

可以看到出现了敏感文件路径。

```
ta:00404000                                     ; DATA XREF: WinMain(x,x,x,x)+6E↑r ...
ta:00404004 ; SC_HANDLE __stdcall OpenServiceA(SC_HANDLE hSCManager, LPCSTR lpServiceName, DWORD dwDesiredAccess)
ta:00404004                 extrn OpenServiceA:dword ; CODE XREF: WinMain(x,x,x,x)+5D↑p
ta:00404004                                     ; DATA XREF: WinMain(x,x,x,x)+5D↑r
ta:00404008 ; SC_HANDLE __stdcall CreateServiceA(SC_HANDLE hSCManager, LPCSTR lpServiceName, LPCSTR lpDisplayName, DWORD d
ta:00404008                 extrn CreateServiceA:dword ; CODE XREF: WinMain(x,x,x,x)+46↑p
ta:00404008                                     ; DATA XREF: WinMain(x,x,x,x)+46↑r
ta:0040400C ; SC_HANDLE __stdcall OpenSCManagerA(LPCSTR lpMachineName, LPCSTR lpDatabaseName, DWORD dwDesiredAccess)
ta:0040400C                 extrn OpenSCManagerA:dword ; CODE XREF: WinMain(x,x,x,x)+D↑p
ta:0040400C                                     ; DATA XREF: WinMain(x,x,x,x)+D↑r
ta:00404010 ; BOOL __stdcall ControlService(SC_HANDLE hService, DWORD dwControl, LPSERVICE_STATUS lpServiceStatus)
ta:00404010                 extrn ControlService:dword ; CODE XREF: WinMain(x,x,x,x)+80↑p
ta:00404010                                     ; DATA XREF: WinMain(x,x,x,x)+80↑r
ta:00404014
ta:00404018 ;
ta:00404018 ; Imports from KERNEL32.dll
ta:00404018 ;
ta:00404018 ; HMODULE __stdcall GetModuleHandleA(LPCSTR lpModuleName)
ta:00404018                 extrn GetModuleHandleA:dword ; CODE XREF: start+C2↑p
ta:00404018                                     ; DATA XREF: start+C2↑r ...
ta:0040401C ; void __stdcall GetStartupInfoA(LPSTARTUPINFOA lpStartupInfo)
ta:0040401C                 extrn GetStartupInfoA:dword ; CODE XREF: start+9F↑p
ta:0040401C                                     ; __ioinit+59↑p
ta:0040401C                                     ; DATA XREF: ...
```

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 为创建的服务名称，此处为 lab10-01。

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）.dwStartType 为服务启动类型，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。

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

lea     eax, [esp+24h+ServiceStatus]
push    eax                 ; lpServiceStatus
push    1                   ; dwControl
push    esi                 ; hService
call    ds:ControlService
```

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

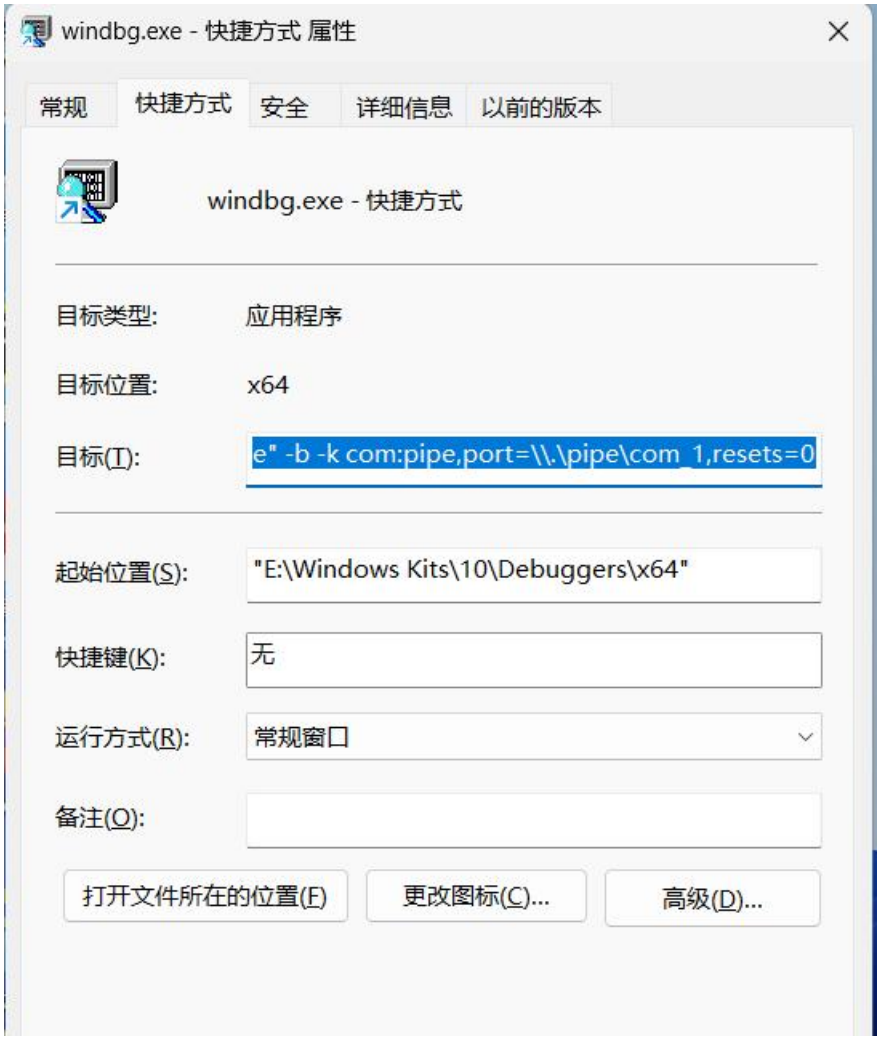
dwControl，要发送的控制码，此处为 1，表示 CONTROL _SERVICE_STOP,将会卸载驱动并调用驱动卸载的函数。==》此次实验就是要在运行过程中，打断点，看看这个程序究竟安装了什么驱动！因为它会自删除驱动。。。IpServiceStatus，返回值，指向存储服务最新状态的结构体 Service，返回信息来自 SCM 中最近的服务状态报告。

使用 windbg 调试内核。首先是 windbg 的配置：

1.对串行端口的配置

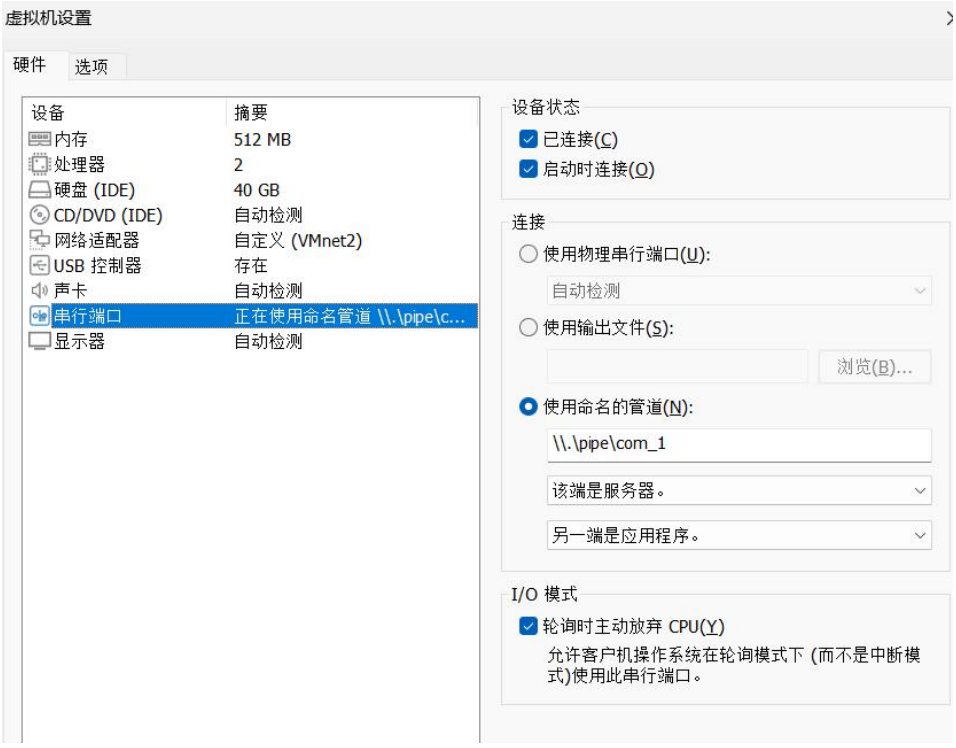
我们在相关网站下载 windbg 之后，将 exe 文件创建快捷方式，之后拖到桌面上，打开其属性在目标里面将其串行端口更改。

2.之后我们配置虚拟机当中的项目，将虚拟机当中的项目给到相应的位置，之后打开虚拟机的设置，将打印机删除，之后新建一个串行端口，并命名为\\.\pip\com_1



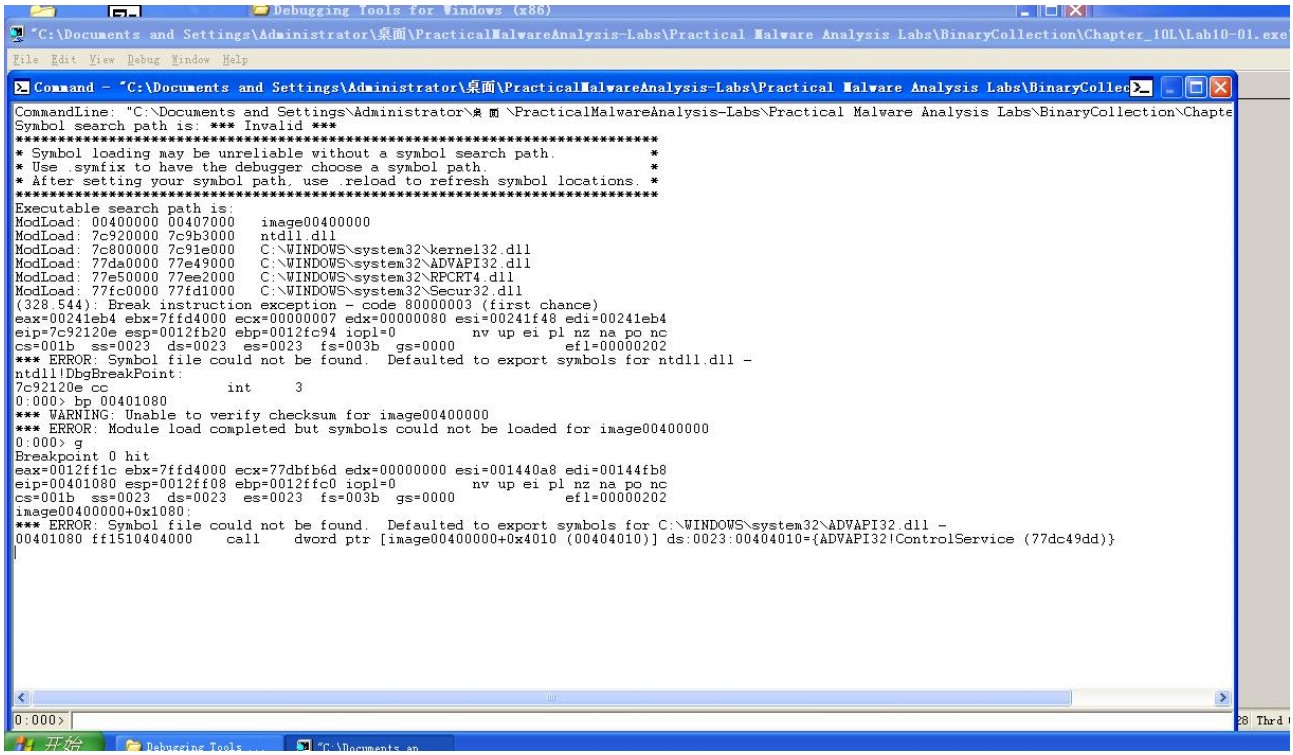
首先在虚拟机当中设置断点，之后我们使用命令 g 运行，得到相应的输出。

可以看到程序在这个点进行了调用 dword 进行了相应的输出。



之后在主机当中选择运行 windbg，之后打开虚拟机，就可以看到虚拟机可以进入调试模式，选择调试模式之后虚拟机会停止运行，这时候我们可以选择在主机的 windbg 端口按 g，就能操控虚拟机了。

如果我们需要在主机当中下断点，可以用 ctrl+break 或者，直接选择 debug->break，就可以将虚拟机停下来，在主机当中下断点进行分析。



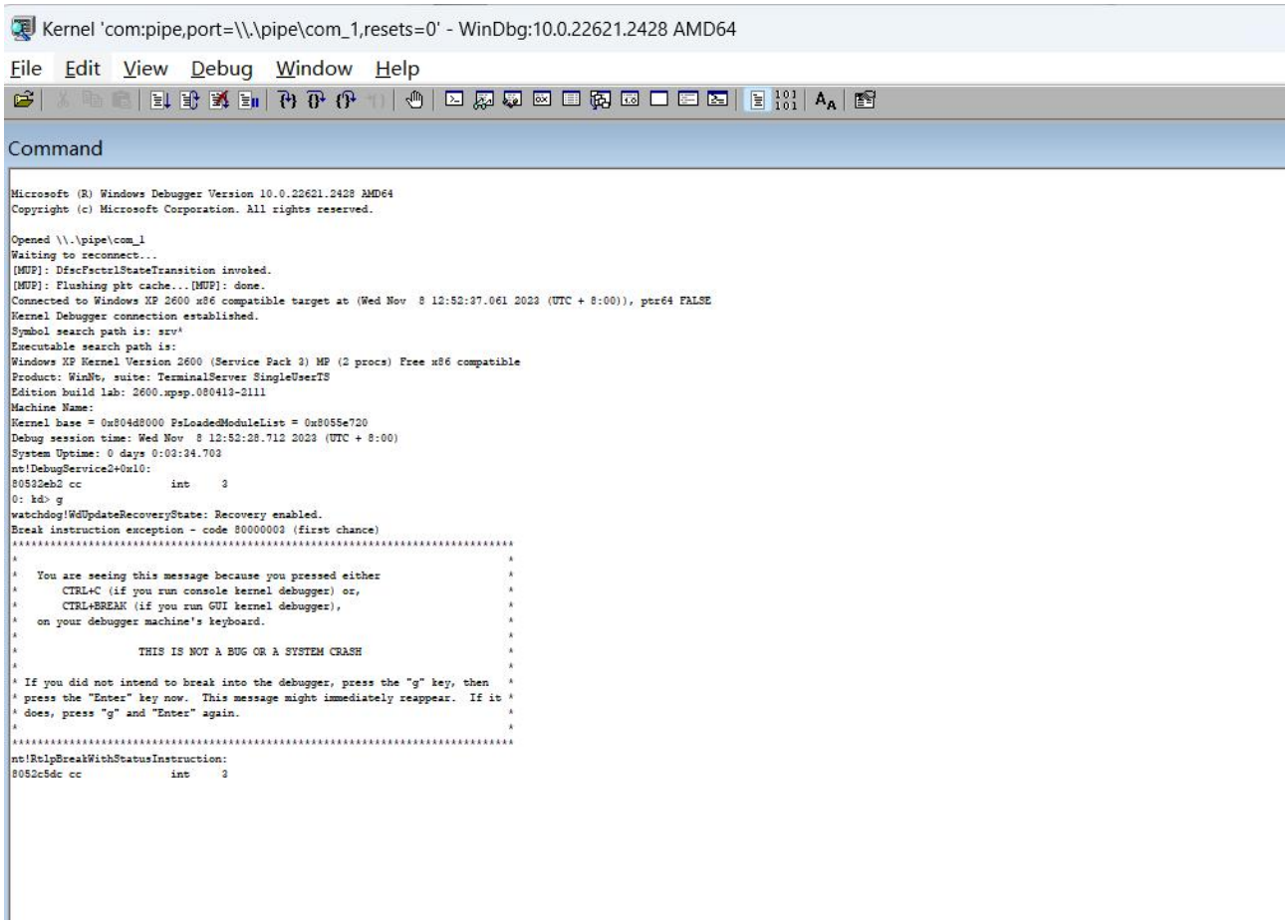

```
Microsoft (R) Windows Debugger Version 6.12.0002.633 X86
Copyright (c) Microsoft Corporation. All rights reserved.

CommandLine: "C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryColle
Symbol search path is: *** Invalid ***
*************************************************************************
* Symbol loading may be unreliable without a symbol search path.        *
* Use .symfix to have the debugger choose a symbol path.                *
* After setting your symbol path, use .reload to refresh symbol locations. *
*************************************************************************
Executable search path is:
ModLoad: 00400000 00407000   image00400000
ModLoad: 7c920000 7c9b3000   ntdll.dll
ModLoad: 7c800000 7c91e000   C:\WINDOWS\system32\kernel32.dll
ModLoad: 77da0000 77e49000   C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e50000 77ee2000   C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77fc0000 77fd1000   C:\WINDOWS\system32\Secur32.dll
(b0.70c): Break instruction exception - code 80000003 (first chance)
eax=00241eb4 ebx=7ffd5000 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>
breakpoint 0 redefined
0:000> g
Breakpoint 0 hit
eax=0012ff1c ebx=7ffd5000 ecx=77dbfb6d edx=00000000 esi=001440a8 edi=00144fb8
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)}
```

在虚拟机中使用 windbg 加载 lob 10-01.exe

在之前使用 ida 得到的 controlservice 地址进行断点，bp 00401080

最终得到 00401080 的数据段是 call dword ptr



使用 win7 宿主机的 winbg 进行调试,以查看此时内核中的驱动加载情况！

winxp 做一些准备工作，修改 boot.ini 文件：

我们在桌面上找到“我的电脑”然后右键单击，选择“属性”，进入系统属性页面之后，我们在第一行选择“高级”然后在下方选择“设置”

2、第二步，我们在"启动和故障恢复"页面中点击“编辑”就能对 Boot.ini 文件进行编辑了，或者如果在虚拟机当中无法显示 boot.ini 的位置，就可以将相关的内容在桌面上编辑好，拖动到相应的文件夹当中，也可以成功的使用。

[boot loader]

timeout=30

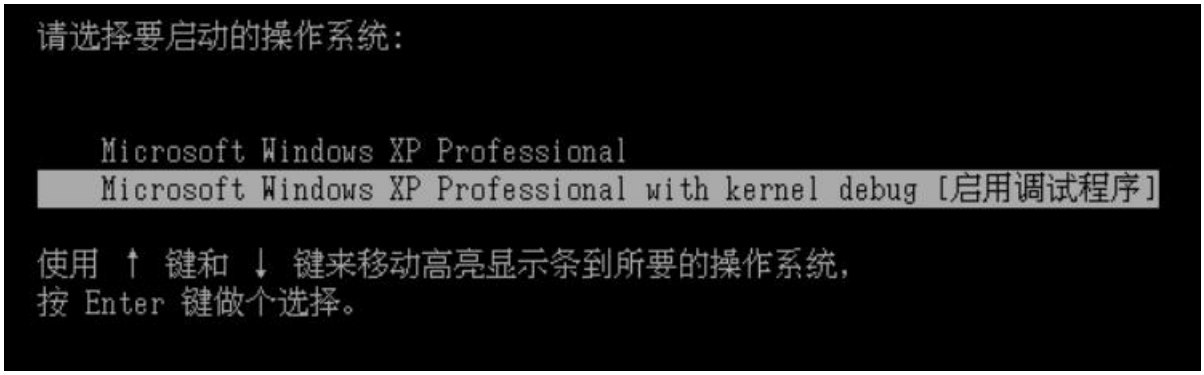
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS

[operating systems]

multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /noexecute=optin /fastdetect

multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional with kernel debug" /noexecute=optin /fastdetect /debug /debugport=COM1 /baudrate=115200

重启之后，可以看到我们选择调试模式进入 winxp



然后 windbg 点击 kernel debug：

之后在虚拟机当中下断点运行之后在主机下断点停下，按 g 执行，这样程序就能够停在内核执行并且不删除的状态。

接下来我们在主机当中下断点将进程停下来，我们可以输入！Object \Driver 这个命令，查看这个进程。

```
 04  81f99460 Driver                    VgaSave
     81c8bb10 Driver                    NDProxy
     82199448 Driver                    Compbatt
 05  81ffc978 Driver                    Ptilink
     82198160 Driver                    MountMgr
     81f79200 Driver                    wdmaud
 06  81cd3030 Driver                    Lab10-01
 07  820d9108 Driver                    dmload
     821b3500 Driver                    isapnp
 08  81c1ccc8 Driver                    redbook
     81c542f8 Driver                    vmmouse
     821072f0 Driver                    atapi
 10  81c8db10 Driver                    RasAcd
     81c26c80 Driver                    PSched
     82109b28 Driver                    dmio
```

可以看到 Lab10-01.exe 是在 81cd3030 这里运行的。

使用 dt _DRIVER_OBJECT 地址 来解析地址的数据结构

```
0: kd> !object 81cd3030
Object: 81cd3030  Type: (821e9e70) Driver
    ObjectHeader: 81cd3018 (old version)
    HandleCount: 0  PointerCount: 2
    Directory Object: e136c538  Name: Lab10-01
0: kd> dt _DRIVER_OBJECT 81cd3030
ntdll!_DRIVER_OBJECT
   +0x000 Type             : 0n4
   +0x002 Size             : 0n168
   +0x004 DeviceObject     : (null)
   +0x008 Flags            : 0x12
   +0x00c DriverStart      : 0xf8d57000 Void
   +0x010 DriverSize       : 0xe80
   +0x014 DriverSection    : 0x821e22d8 Void
   +0x018 DriverExtension  : 0x81cd30d8 _DRIVER_EXTENSION
   +0x01c DriverName       : _UNICODE_STRING "\Driver\Lab10-01"
   +0x024 HardwareDatabase : 0x8067e260 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
   +0x028 FastIoDispatch   : (null)
   +0x02c DriverInit       : 0xf8d57959     long  +0
   +0x030 DriverStartIo    : (null)
   +0x034 DriverUnload     : 0xf8d57486     void  +0
   +0x038 MajorFunction    : [28] 0x804f5552     long  nt!IopInvalidDeviceRequest+0
```

重点观察 DriverUnload 函数，地址为 0xf8d57486，使用 bp 指令在此加断点,并使用 g 指令恢复内核的执行。

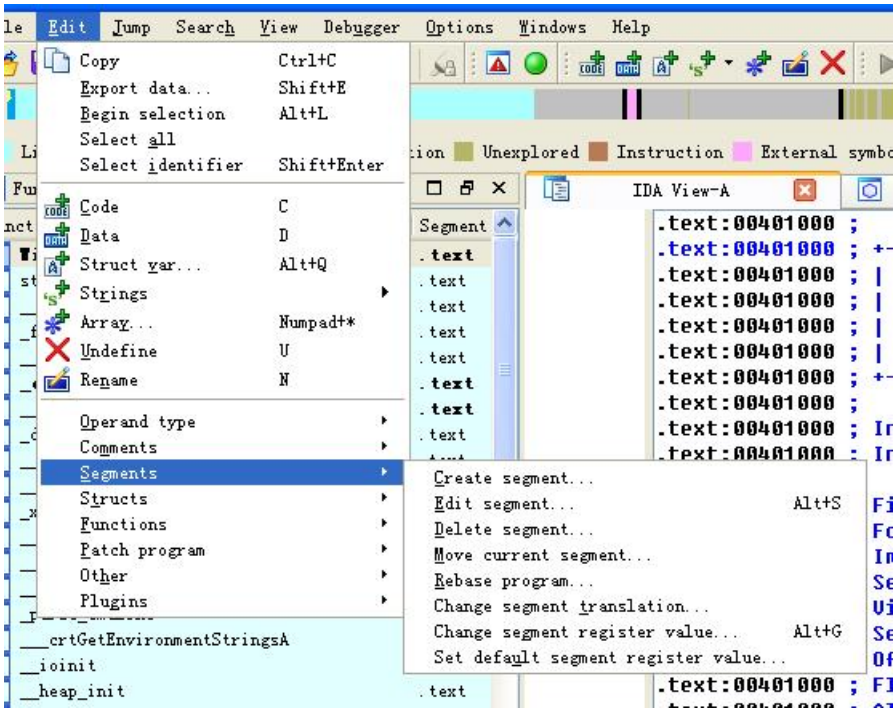
```
0: kd> bp 0xf8d57486
0: kd> g
Break instruction exception - code 80000003 (first chance)
Lab10_01+0x486:
f8d57486 8bff            mov     edi,edi
```

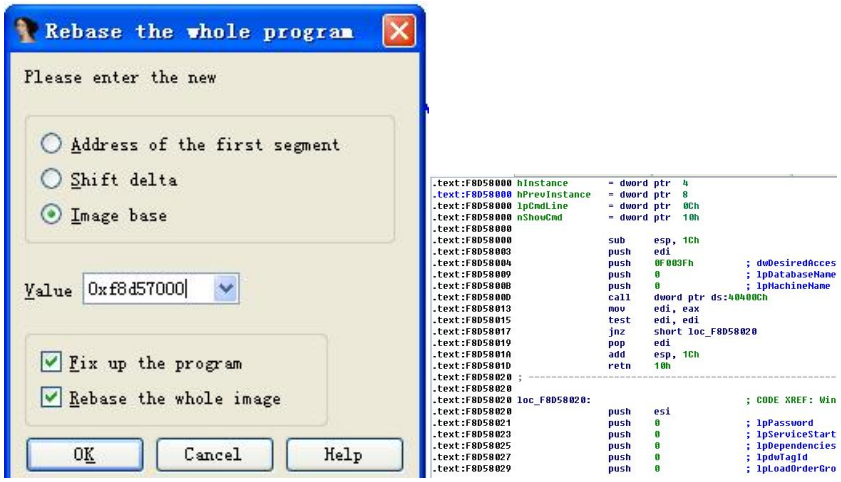
通过按 t 单步执行下一条指令

可以使用 ida 进行分析。从前面得知 DriverStart 的地址和 DriverUpload 的地址，从而得到偏移量 0x486。

```
0: kd> g
Break instruction exception - code 80000003 (first chance)
Lab10_01+0x486:
f8d57486 8bff            mov     edi,edi
0: kd> t
Lab10_01+0x488:
f8d57488 55              push    ebp
1: kd> t
Lab10_01+0x489:
f8d57489 8bec            mov     ebp,esp
1: kd> t
Lab10_01+0x48b:
f8d5748b 51              push    ecx
1: kd> t
Lab10_01+0x48c:
f8d5748c 53              push    ebx
```

在 ida 中 driver 的默认地址的 sys 文件是从 0x00010000 开始的，所以函数卸载代码对应的地址为 0x00010468。另外一个方法则是重新设置 ida 默认的基地址





Lab10-02.exe

首先依旧是查看字符串，寻找 writefile 的相关操作。

```
.rdata:00··· 00000008    C     (8PX\a\b
.rdata:00··· 00000007    C    700WP\a
.rdata:00··· 00000008    C    \b`h`
.rdata:00··· 0000000A    C    ppxxxx\b\a\b
.rdata:00··· 00000007    C    (null)
.rdata:00··· 0000000F    C    runtime error
.rdata:00··· 0000000E    C    TLOSS error\r\n
.rdata:00··· 0000000D    C    SING error\r\n
.rdata:00··· 0000000F    C    DOMAIN error\r\n
.rdata:00··· 00000025    C    R6028\r\n- unable to initialize heap\r\n
.rdata:00··· 00000035    C    R6027\r\n- not enough space for lowio initialization\r\n
.rdata:00··· 00000035    C    R6026\r\n- not enough space for stdio initialization\r\n
.rdata:00··· 00000026    C    R6025\r\n- pure virtual function call\r\n
.rdata:00··· 00000035    C    R6024\r\n- not enough space for _onexit/atexit table\r\n
.rdata:00··· 00000029    C    R6019\r\n- unable to open console device\r\n
.rdata:00··· 00000021    C    R6018\r\n- unexpected heap error\r\n
.rdata:00··· 0000002D    C    R6017\r\n- unexpected multithread lock error\r\n
.rdata:00··· 0000002C    C    R6016\r\n- not enough space for thread data\r\n
.rdata:00··· 00000021    C    \r\nabnormal program termination\r\n
.rdata:00··· 0000002C    C    R6009\r\n- not enough space for environment\r\n
.rdata:00··· 0000002A    C    R6008\r\n- not enough space for arguments\r\n
.rdata:00··· 00000025    C    R6002\r\n- floating point not loaded\r\n
.rdata:00··· 00000025    C    Microsoft Visual C++ Runtime Library
.rdata:00··· 0000001A    C    Runtime Error!\n\nProgram:
.rdata:00··· 00000017    C    <program name unknown>
.rdata:00··· 00000013    C    GetLastActivePopup
```

接下来查看相应的反汇编代码：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  DWORD v3; // ecx@0
  HRSRC v4; // edi@1
  HGLOBAL v5; // ebx@1
  HANDLE v6; // esi@2
  DWORD v7; // eax@3
  SC_HANDLE v8; // eax@3
  SC_HANDLE v10; // eax@5
  void *v11; // esi@5
  DWORD NumberOfBytesWritten; // [sp+0h] [bp-4h]@1

  NumberOfBytesWritten = v3;
  v4 = FindResourceA(0, (LPCSTR)0x65, Type);
  v5 = LoadResource(0, v4);
  if ( v4 )
  {
    v6 = CreateFileA(BinaryPathName, 0xC0000000, 0, 0, 2u, 0x80u, 0);
    if ( v6 != (HANDLE)-1 )
    {
      v7 = SizeofResource(0, v4);
      WriteFile(v6, v5, v7, &NumberOfBytesWritten, 0);
      CloseHandle(v6);
      v8 = OpenSCManagerA(0, 0, 0xF003Fu);
      if ( !v8 )
      {
        printf(aFailedToOpenSe);
        return 0;
      }
      v10 = CreateServiceA(v8, DisplayName, DisplayName, 0xF01FFu, 1u, 3u, 1u, BinaryPathName, 0, 0, 0, 0, 0);
      v11 = v10;
      if ( !v10 )
      {
        printf(aFailedToCreate);
        return 0;
      }
```

基本上可以确定是在利用资源文件创建服务，服务是一个 sys 驱动。

在 process monitor 下监控其运行

这里有几个我们已经见过好几次的函数，OpenSCManagerA 是用来打开服务管理器的函数，StartServiceA 是用来启动一个服务的函数，CreateServiceA 是创建一个服务的，说明这个代码会在宿主计算机上创建一个服务来运行代码。

书中还说了这两个函数 LoadResource 和 SizeOfResource，说明这个代码对 Lab10-02.exe 的资源节做了一些操作，我们找找这两个函数

23:4...	Lab10-02.exe	276	Process Start		SUCCESS
23:4...	Lab10-02.exe	276	Thread Create		SUCCESS
23:4...	Lab10-02.exe	276	IRP_MJ_QUERY...	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_...	SUCCESS
23:4...	Lab10-02.exe	276	Load Image	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_...	SUCCESS
23:4...	Lab10-02.exe	276	Load Image	C:\WINDOWS\system32\ntdll.dll	SUCCESS
23:4...	Lab10-02.exe	276	IRP_MJ_QUERY...	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_...	SUCCESS
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\Prefetch\LAB10-02.EXE-3902D939.pf	NAME NOT FOUNI
23:4...	Lab10-02.exe	276	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Lab10-02.exe	NAME NOT FOUNI
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_10L	SUCCESS
23:4...	Lab10-02.exe	276	IRP_MJ_FILE_...	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_10L	SUCCESS
23:4...	Lab10-02.exe	276	FASTIO_NETWO...	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_...	NAME NOT FOUNI
23:4...	Lab10-02.exe	276	Load Image	C:\WINDOWS\system32\kernel32.dll	SUCCESS
23:4...	Lab10-02.exe	276	RegOpenKey	HKLM\System\CurrentControlSet\Control\Terminal Server	SUCCESS
23:4...	Lab10-02.exe	276	RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSAppCompat	SUCCESS
23:4...	Lab10-02.exe	276	RegCloseKey	HKLM\System\CurrentControlSet\Control\Terminal Server	SUCCESS
23:4...	Lab10-02.exe	276	Thread Create		SUCCESS
23:4...	Lab10-02.exe	276	IRP_MJ_READ	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_...	SUCCESS
23:4...	Lab10-02.exe	276	Load Image	C:\WINDOWS\system32\advapi32.dll	SUCCESS
23:4...	Lab10-02.exe	276	Load Image	C:\WINDOWS\system32\rpcrt4.dll	SUCCESS
23:4...	Lab10-02.exe	276	Load Image	C:\WINDOWS\system32\secur32.dll	SUCCESS
23:4...	Lab10-02.exe	276	RegOpenKey	HKLM\System\CurrentControlSet\Control\Terminal Server	SUCCESS
23:4...	Lab10-02.exe	276	RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSAppCompat	SUCCESS
23:4...	Lab10-02.exe	276	RegCloseKey	HKLM\System\CurrentControlSet\Control\Terminal Server	SUCCESS
23:4...	Lab10-02.exe	276	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Secur32.dll	NAME NOT FOUNI
23:4...	Lab10-02.exe	276	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\RPCRT4.dll	NAME NOT FOUNI
23:4...	Lab10-02.exe	276	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\ADVAPI32.dll	NAME NOT FOUNI
23:4...	Lab10-02.exe	276	RegOpenKey	HKLM\System\CurrentControlSet\Control\Terminal Server	SUCCESS
23:4...	Lab10-02.exe	276	RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSAppCompat	SUCCESS
23:4...	Lab10-02.exe	276	RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSUserEnabled	SUCCESS
23:4...	Lab10-02.exe	276	RegCloseKey	HKLM\System\CurrentControlSet\Control\Terminal Server	SUCCESS
23:4...	Lab10-02.exe	276	RegOpenKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon	SUCCESS
23:4...	Lab10-02.exe	276	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\LeakTrack	NAME NOT FOUNI
23:4...	Lab10-02.exe	276	RegCloseKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon	SUCCESS
23:4...	Lab10-02.exe	276	RegOpenKey	HKLM	SUCCESS
23:4...	Lab10-02.exe	276	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Diagnostics	NAME NOT FOUNI
23:4...	Lab10-02.exe	276	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\ntdll.dll	NAME NOT FOUNI
23:4...	Lab10-02.exe	276	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\kernel32.dll	NAME NOT FOUNI
23:4...	Lab10-02.exe	276	IRP_MJ_READ	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_...	SUCCESS
23:4...	Lab10-02.exe	276	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Console	SUCCESS
23:4...	Lab10-02.exe	276	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Console\ConsoleIME	NAME NOT FOUNI
23:4...	Lab10-02.exe	276	RegOpenKey	HKLM\System\CurrentControlSet\Control\Session Manager	SUCCESS
23:4...	Lab10-02.exe	276	RegQueryValue	HKLM\System\CurrentControlSet\Control\Session Manager\SafeProcessSearchMode	NAME NOT FOUNI
23:4...	Lab10-02.exe	276	RegCloseKey	HKLM\System\CurrentControlSet\Control\Session Manager	SUCCESS
23:4...	Lab10-02.exe	276	FASTIO_NETWO...	C:\WINDOWS\system32\conime.exe	SUCCESS
23:4...	Lab10-02.exe	276	IRP_MJ_READ	C:\$Mft	SUCCESS
23:4...	Lab10-02.exe	276	IRP_MJ_READ	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_...	SUCCESS
23:4...	Lab10-02.exe	276	FASTIO_NETWO...	C:\WINDOWS\system32\conime.exe	SUCCESS
23:4...	Lab10-02.exe	276	RegOpenKey	HKLM\System\CurrentControlSet\Control\Session Manager	SUCCESS
23:4...	Lab10-02.exe	276	RegQueryValue	HKLM\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode	NAME NOT FOUNI
23:4...	Lab10-02.exe	276	RegCloseKey	HKLM\System\CurrentControlSet\Control\Session Manager	SUCCESS

这两个函数是 KERNEL32.DLL 的导入函数，不注意看还是难发现的，然后我们知道了这个代码会操作自己的资源节，那我们就去检查一下这个程序的资源节

进行 regshot 动态分析，这里我们可以发现的是，这里出现了几个键的改变。

发现其中尤其需要注意的是出现了 486_WS_Driver

在 process monitor 当中筛选可以查找到相应的位置里有这个进程。

23:4...	services.exe	672	RegCreateKey	HKLM\System\CurrentControlSet\Control\DeviceClasses	SUCCESS
23:4...	services.exe	672	RegCreateKey	HKLM\System\CurrentControlSet\Control\DeviceClasses	SUCCESS
23:4...	services.exe	672	RegCreateKey	HKLM\System\CurrentControlSet\Services\486 WS Driver	SUCCESS
23:4...	services.exe	672	RegCreateKey	HKLM\System\CurrentControlSet\Services\486 WS Driver\Security	SUCCESS
23:4...	services.exe	672	RegCreateKey	HKLM\System\CurrentControlSet\Enum\Root\LEGACY_486_WS_DRIVER	SUCCESS
23:4...	services.exe	672	RegCreateKey	HKLM\System\CurrentControlSet\Enum\Root\LEGACY_486_WS_DRIVER\0000	SUCCESS
23:4...	services.exe	672	RegCreateKey	HKLM\System\CurrentControlSet\Enum\Root\LEGACY_486_WS_DRIVER\0000\Control	SUCCESS
23:4...	services.exe	672	RegCreateKey	HKLM\System\CurrentControlSet\Services\486 WS Driver\Enum	SUCCESS

这里我们发现了一个叫 services.exe 的代码，执行了 RegCreateKey，而且路径也和我们 Regshot 的结果相同.然后我们缩小搜索范围，搜索这个名叫 services.exe 的程序做了哪些其他事，记住此时这个程序的 PID 为 672。

设置筛选条件为 WriteFile 之后，就会发现这个文件一共写了三个文件，一个是 system.LOG，一个是 system，还有一个是 SysEvent.Evt，然后我们试试查找 Lab10-02.exe 这个进程名字，恶意行为分析本身就很繁琐。

23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system.LOG
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$LogFile
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$LogFile
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$Mft
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$Directory
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$Mft
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$LogFile
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system.LOG
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system.LOG
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system.LOG
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system.LOG
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system.LOG
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system.LOG
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system.LOG
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system.LOG
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system.LOG
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system.LOG
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system.LOG
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system.LOG
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$LogFile
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$Mft
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$Directory
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$LogFile
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system.LOG
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$LogFile
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$LogFile
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$Directory
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$LogFile
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$LogFile
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$LogFile
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$LogFile
23:4...	services.exe	672	IRP_MJ_WRITE	C:\WINDOWS\system32\config\system
23:4...	services.exe	672	IRP_MJ_WRITE	C:\$LogFile

又查看了 Lab10.02.exe 的操作，发现其中有对于 conime.exe 的操作，于是我们查找 conime.exe

File Edit Event Filter Tools Options Help

Ti...	Process Name	PID	Operation	Path	Result
23:4...	conime.exe	1992	Process Start		SUCCESS
23:4...	conime.exe	1992	Thread Create		SUCCESS
23:4...	conime.exe	1992	IRP_MJ_QUERY...	C:\WINDOWS\system32\conime.exe	SUCCESS
23:4...	conime.exe	1992	Load Image	C:\WINDOWS\system32\conime.exe	SUCCESS
23:4...	conime.exe	1992	Load Image	C:\WINDOWS\system32\ntdll.dll	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_QUERY...	C:\WINDOWS\system32\conime.exe	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_CREATE	C:\WINDOWS\Prefetch\CONIME.EXE-13EEEA1A.pf	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_READ	C:\$Mft	SUCCESS
23:4...	conime.exe	1992	FASTIO_QUERY...	C:\WINDOWS\Prefetch\CONIME.EXE-13EEEA1A.pf	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_READ	C:\WINDOWS\Prefetch\CONIME.EXE-13EEEA1A.pf	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_READ	C:\WINDOWS\Prefetch\CONIME.EXE-13EEEA1A.pf	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_CLEANUP	C:\WINDOWS\Prefetch\CONIME.EXE-13EEEA1A.pf	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_CREATE	C:	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_QUERY...	C:	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_FILE_...	C:	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_CREATE	C:\	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\	NO MORE FILES
23:4...	conime.exe	1992	IRP_MJ_CLEANUP	C:\	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_CLOSE	C:\	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_CREATE	C:\WINDOWS	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\WINDOWS	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\WINDOWS	NO MORE FILES
23:4...	conime.exe	1992	IRP_MJ_CLEANUP	C:\WINDOWS	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_CLOSE	C:\WINDOWS	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_CREATE	C:\WINDOWS\AppPatch	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\WINDOWS\AppPatch	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\WINDOWS\AppPatch	NO MORE FILES
23:4...	conime.exe	1992	IRP_MJ_CLEANUP	C:\WINDOWS\AppPatch	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_CLOSE	C:\WINDOWS\AppPatch	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_CREATE	C:\WINDOWS\system32	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\WINDOWS\system32	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\WINDOWS\system32	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\WINDOWS\system32	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\WINDOWS\system32	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\WINDOWS\system32	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\WINDOWS\system32	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\WINDOWS\system32	NO MORE FILES
23:4...	conime.exe	1992	IRP_MJ_CLEANUP	C:\WINDOWS\system32	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_CLOSE	C:\WINDOWS\system32	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_CREATE	C:\WINDOWS\WinSxS	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\WINDOWS\WinSxS	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\WINDOWS\WinSxS	NO MORE FILES
23:4...	conime.exe	1992	IRP_MJ_CLEANUP	C:\WINDOWS\WinSxS	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_CLOSE	C:\WINDOWS\WinSxS	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_CREATE	C:\WINDOWS\WinSxS\X86_MICROSOFT.WINDOWS.COMMON-CONTROLS_6595B64144CCF1DF_6.0.2600.5512_X-WW_35D4CE83	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_DIREC...	C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83	NO MORE FILES
23:4...	conime.exe	1992	IRP_MJ_CLEANUP	C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83	SUCCESS
23:4...	conime.exe	1992	IRP_MJ_CLOSE	C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83	SUCCESS

Showing 1,200 of 230,450 events (0.52%) Backed by virtual memory

然后这个动态分析大概就只能分析出来这些东西了，书中说，如果你去找那个 Mlwx486.sys，你是找不到的，但是我们可以找到这个 conime.exe 文件 ==》这是本实验的核心，这个恶意软件的精髓就是隐藏这个 Mlwx486.sys 文件，如何做到的？就是利用内核的钩子，在系统使用 NtQueryDirectoryFile 遍历文件的时候，隐藏了 Mlwx486.sys！

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>sc query "486 WS Driver"

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

C:\Documents and Settings\Administrator>
```

可以很明显的看出来这个是内核的驱动(KERNEL_DRIVER)

作者是如何知道这个内核驱动的名字是 486 WS Driver 呢？

```
loc_401097:                 ; lpPassword
push    0
push    0                   ; lpServiceStartName
push    0                   ; lpDependencies
push    0                   ; lpdwTagId
push    0                   ; lpLoadOrderGroup
push    offset BinaryPathName ; "C:\\Windows\\System32\\Mlwx486.sys"
push    1                   ; dwErrorControl
push    3                   ; dwStartType
push    1                   ; dwServiceType
push    0F01FFh             ; dwDesiredAccess
push    offset DisplayName ; "486 WS Driver"
push    offset DisplayName ; "486 WS Driver"
push    eax                 ; hSCManager
call    ds:CreateServiceA
mov     esi, eax
test    esi, esi
jnz     short loc_4010DC
```

我们可以从注册表中可以找到这个字符串的位置，然后也可以从从 CreateServiceA 中看出(书中说)

File Edit Event Filter Tools Options Help

Ti...	Process Name	PID	Operation	Path	Result	Detail
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\Prefetch\LAB10-02.EXE-3902D939.pf	NAME NOT FOUND	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\Documents and Settings\Administrator\桌面\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_10L	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\system32\conime.exe	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\system32\Mlwx486.sys	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\system32	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\system32\apphelp.dll	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\system32\apphelp.dll	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\AppPatch\sysmain.sdb	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\AppPatch\systest.sdb	NAME NOT FOUND	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\system32	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\system32	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\system32\conime.exe	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\system32\conime.exe	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\system32\conime.exe	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\system32\conime.exe	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\system32	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\system32	SUCCESS	Desired Acces...
23:4...	Lab10-02.exe	276	IRP_MJ_CREATE	C:\WINDOWS\system32\conime.exe.Manifest	NAME NOT FOUND	Desired Acces...

所以这个问题的答案就是创建了 conime.exe，还有 apphelp.dll，sysmain.sdb，systest.sdb，最后当然还有那个 sys 驱动 Mlwx486.sys

2.这个程序有内核组件吗？

解答： 现在我们就要连接内核调试器来操作了

WinDbg 里面运行命令

Lm

然后仔细找就可以找到这个驱动

下一步是要把虚拟机恢复成 Rootkit 安装之前的状态来查找这个位置上原来的函数是什么

这个函数原来的位置是 nt!NtQueryDirectoryFile，然后接下来我们运行这个病毒，开始继续分析这个病毒，现在我们已经运行了病毒，找到那个函数的位置！为了搞清楚这个函数在做啥，我们导出资源文件里面的 PE，IDA 反编译：

```
NTSTATUS __stdcall sub_10486(HANDLE FileHandle, HANDLE Event, PIO_APC_ROUTINE ApcRoutine, PVOID ApcContext, PIO_STATUS_BLOCK IoStatusBlock, PVOID FileInformati
{
  PVOID v11; // esi@1
  NTSTATUS v12; // eax@1
  PVOID v13; // edi@1
  char v14; // bl@4
  NTSTATUS RestartScana; // [sp+38h] [bp+30h]@1

  v11 = FileInformation;
  v12 = NtQueryDirectoryFile(
          FileHandle,
          Event,
          ApcRoutine,
          ApcContext,
          IoStatusBlock,
          FileInformation,
          FileInformationLength,
          FileInformationClass,
          ReturnSingleEntry,
          FileName,
          RestartScan);
  v13 = 0;
  RestartScana = v12;
  if ( FileInformationClass == 3 && v12 >= 0 && !ReturnSingleEntry )
  {
    while ( 1 )
```

```
RestartScana = v12;
if ( FileInformationClass == 3 && v12 >= 0 && !ReturnSingleEntry )
{
  while ( 1 )
  {
    v14 = 0;
    if ( RtlCompareMemory((char *)v11 + 94, &word_1051A, 8u) == 8 )
    {
      v14 = 1;
      if ( v13 )
      {
        if ( *(_DWORD *)v11 )
          *(_DWORD *)v13 += *(_DWORD *)v11;
        else
          *(_DWORD *)v13 = 0;
      }
    }
    if ( !*(_DWORD *)v11 )
      break;
    if ( !v14 )
      v13 = v11;
    v11 = (char *)v11 + *(_DWORD *)v11;
  }
}
return RestartScana;
```

```
.text:0001051A word_1051A       dw 4Dh                  ; DATA XREF: sub_10486+46↑o
.text:0001051C aLwx:
.text:0001051C                  unicode 0, <lwx>,0
.text:00010524                  align 80h
.text:00010524 _text            ends
```

然后计算机通过上面那个公式计算真实地址

本来第三个结构体的地址是 000000c0，但是经过这么一个通过改变 offset 之后，计算机计算之后，得出的地址就变成 00000180（根据上面那个计算公式）

计算机通过计算之后，认为第三个结构体存在 00000180 这个地址上，就去 00000180 上取数据，从而跳过了第三个结构体，所以这个通过改变 offset 在不改变数据结构的前提之下，达到了隐藏文件的目的，也只会有天才才会想的出来了

第二问的答案就是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

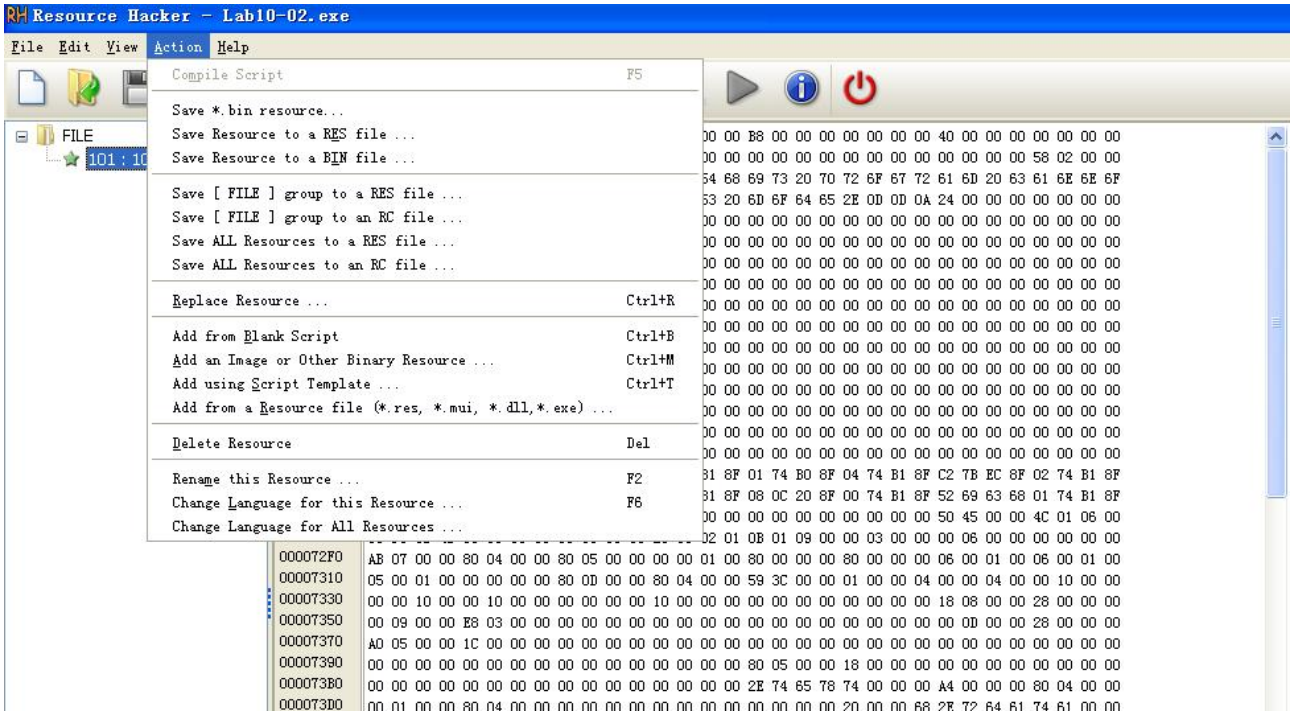
3.这个程序做了些什么？

解答：通过上面的分析，可以得出，这是用来隐藏文件的 RootKit，它使用 SSDT 来挂钩覆盖 NtQueryDirectoryFile 函数，通过自定义一些操作，来隐藏文件

书中给了我们三个方案，第一中通过 cmd 关闭的行为并未能验证成功

```
C:\Documents and Settings\Administrator>sc stop y "486 WS Driver"
[SC] OpenService FAILED 1060:
```

第二种方案我们可以用 Resource hacker 进行捕获，捕获之后我们可以获得一个文件，将其放入 ida 之中进行分析，可以看到相关的内容。



在 ida 当中打开相应的程序

这就很显然了，创建一个叫做 Process Helper 的服务，加载 C:\\Windows\\System32\\Lab10-03.sys 的内核驱动；

```
.text:00010514
.text:00010514                 jmp     ds:__imp_NtQueryDirectoryFile
.text:00010514 NtQueryDirectoryFile endp
.text:00010514
.text:00010514 ; ---------------------------------------------------------------------------
.text:0001051A word_1051A      dw 4Dh                  ; DATA XREF: sub_10486+46↑o
.text:0001051C aLwx:
.text:0001051C                 unicode 0, <lwx>,0
.text:00010524                 align 80h
.text:00010524 _text           ends
.text:00010524
.idata:00010580 ; Section 2. (virtual address 00000580)
.idata:00010580 ; Virtual size                  : 000000A6 (    166.)
.idata:00010580 ; Section size in file          : 00000100 (    256.)
.idata:00010580 ; Offset to raw data for section: 00000580
.idata:00010580 ; Flags 48000040: Data Not pageable Readable
.idata:00010580 ; Alignment     : default
.idata:00010580 ;
.idata:00010580 ; Imports from ntoskrnl.exe
.idata:00010580 ;
.idata:00010580 ; ===========================================================================
.idata:00010580
.idata:00010580 ; Segment type: Externs
.idata:00010580 ; _idata
.idata:00010580 ; NTSTATUS __stdcall NtQueryDirectoryFile(HANDLE FileHandle, HANDLE Event, PIO_APC_ROUTINE ApcRoutine, PVOID ApcContext,
.idata:00010580                 extrn __imp_NtQueryDirectoryFile:dword
.idata:00010580                                         ; DATA XREF: HEADER:00010288↑o
```

我们进入 DriverEntry 这个例程

```
INIT:000107AB DriverObject    = dword ptr  8
INIT:000107AB RegistryPath    = dword ptr  0Ch
INIT:000107AB
INIT:000107AB                 mov     edi, edi
INIT:000107AD                 push    ebp
INIT:000107AE                 mov     ebp, esp
INIT:000107B0                 call    sub_10772
INIT:000107B5                 pop     ebp
INIT:000107B6                 jmp     sub_10706
INIT:000107B6 DriverEntry     endp
INIT:000107B6
INIT:000107B6 ; ---------------------------------------------------------------------------
INIT:000107BB                 align 4
INIT:000107BC ; const WCHAR aKeservicedescr
INIT:000107BC aKeservicedescr:                        ; DATA XREF: sub_10706+1B↑o
INIT:000107BC                 unicode 0, <KeServiceDescriptorTable>,0
INIT:000107EE ; const WCHAR SourceString
INIT:000107EE SourceString    db 'N',0                ; DATA XREF: sub_10706+10↑o
INIT:000107F0 aTquerydirector:
INIT:000107F0                 unicode 0, <tQueryDirectoryFile>,0
INIT:00010818 __IMPORT_DESCRIPTOR_ntoskrnl_exe dd rva off_10840
INIT:00010818                                         ; DATA XREF: HEADER:000102D8↑o
INIT:00010818                                         ; Import Name Table
INIT:0001081C                 dd 0                    ; Time stamp
INIT:00010820                 dd 0                    ; Forwarder Chain
INIT:00010824                 dd rva aNtoskrnl_exe    ; DLL Name
INIT:00010828                 dd rva __imp_NtQueryDirectoryFile ; Import Address Table
INIT:0001082C                 align 20h
INIT:00010840 ;
INIT:00010840 ; Import names for ntoskrnl.exe
INIT:00010840 ;
INIT:00010840 off_10840       dd rva word_1086C       ; DATA XREF: INIT:__IMPORT_DESCRIPTOR_ntoskrnl_exe↑o
INIT:00010844                 dd rva word_10884
```

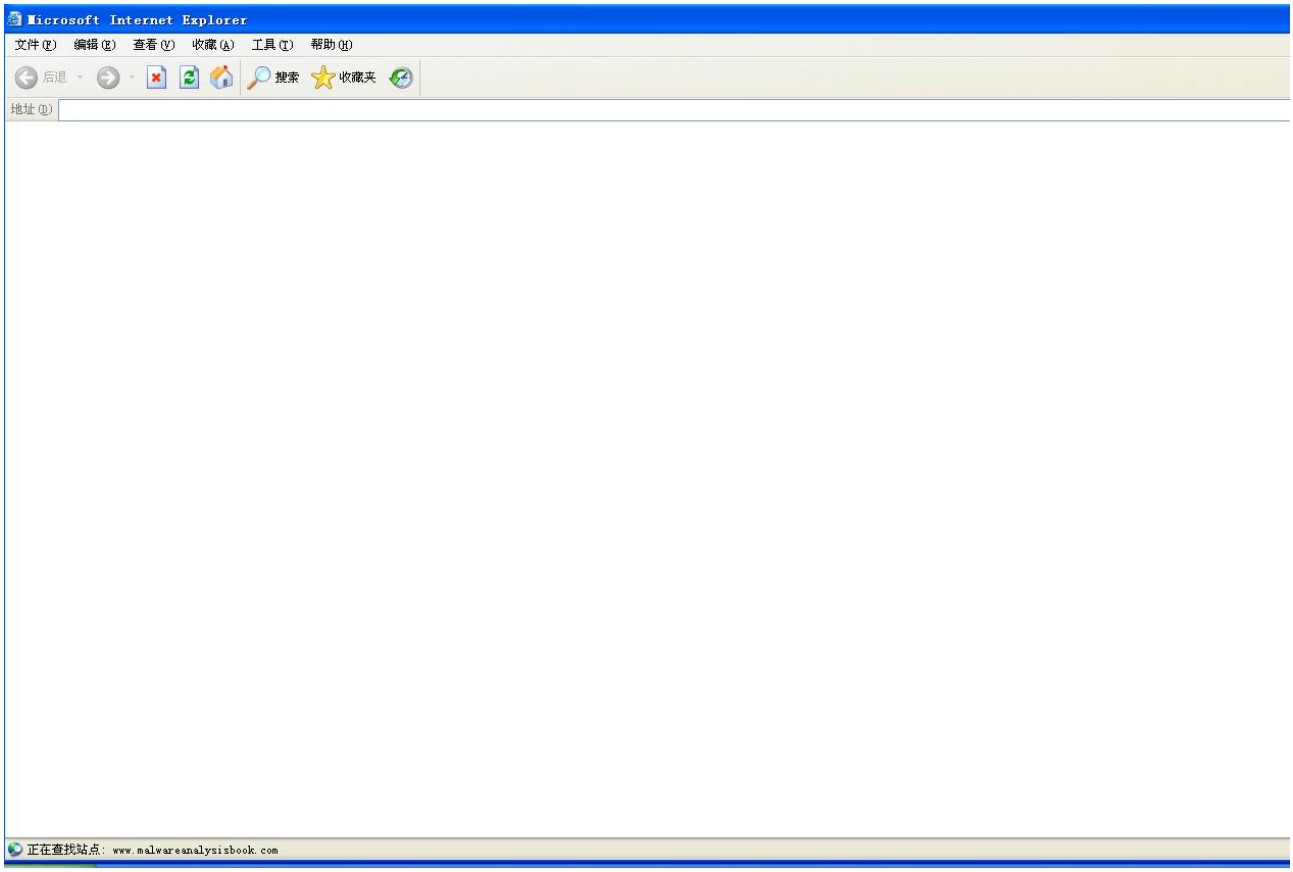
这里就不详细分析这个代码了，书上说是 RtlInitUnicodeString 以参数 KeServiceDescirptorTable 和 NtQueryDircetoryFile 做入参，然后用 MmGetSystemRoutineAddress 这个函数来查找这个两个地址的偏移量，接下来他把地址做了一个替换。

Lab10-03.exe

1.这个程序做了些什么？

解答：书上说本次实验包括一个驱动程序和一个可执行文件，我们把两者全部放到 C:\Windows\System32 目录下面，我们试试，接下来运行可执行文件，就可以发现程序想要打开目标网站的操作。

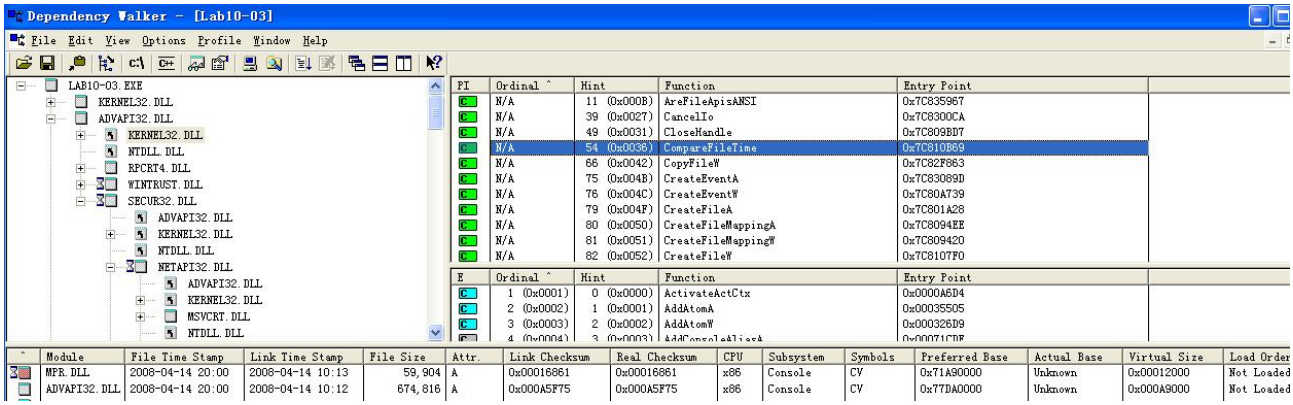
把文件放到那个目录之后，点击执行就会跳出这个 IE，然后不断的跳 IE 出来



然后我们开始分析，先是安装书上的开始静态分析

我们先分析的是 exe 文件

在 KERNEL32.DLL 里面我们发现这么几个有意思的函数 CreateFile 和 WriteFile

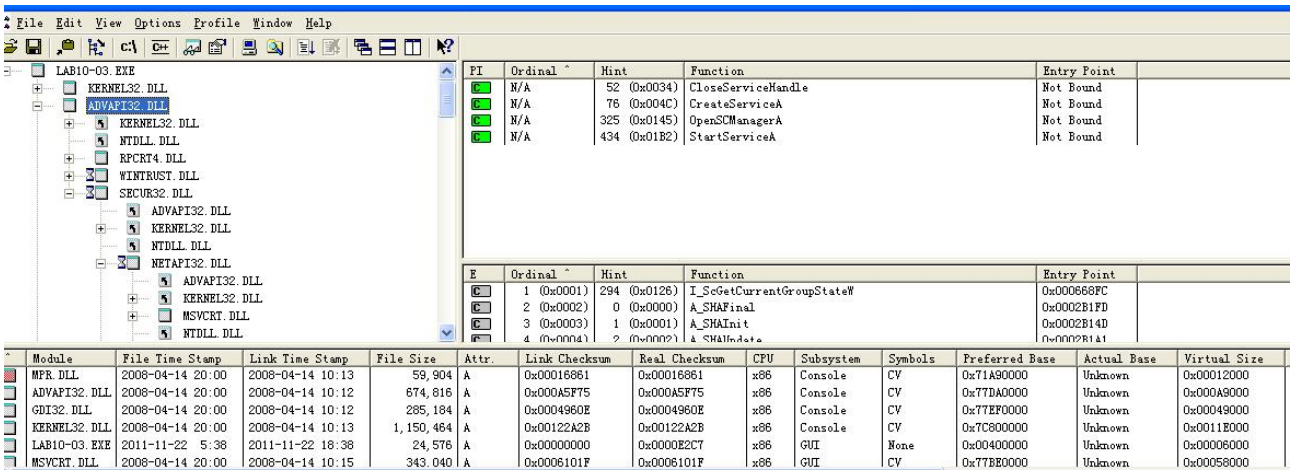


然后我们查看这个 ADVAPI32.DLL 这个导入库

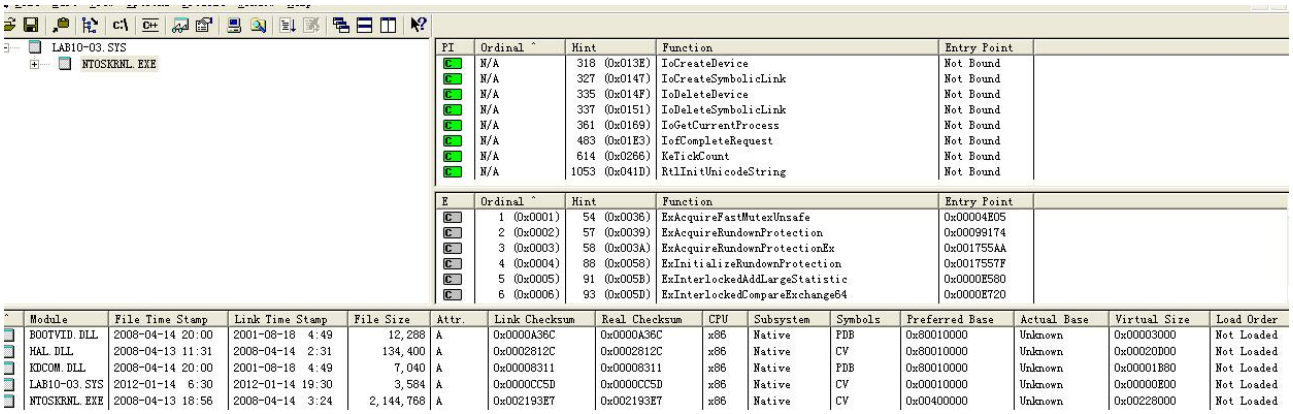
这个有一个比较让人感兴趣的导入函数就是 OpenSCManagerA，还有 StartServiceA 以及 CreateServiceA 这三个导出函数，说明这个代码会创建一个服务在系统中

然后我们开始分析 sys 文件的导出函数有哪些

我们可以看到如下的一些函数，书上说，从 IoGetCurrentProcess 这个例程可以看出，这个驱动或者在修改正在运行的进程，或者需要关于进程的信息



这个病毒最特别的地方在于，如果我们打算用任务管理器关闭这个进程的时候，会发现，根本没有这个进程，包括使用 Process Explorer 里面也是没有列出来



第一个调用的函数就 OpenSCManagerA，这是一个用于打开服务控制的函数，说明程序打算在这里操作服务，这里我们就不着重分析各种跳转了

如果上面的 OpenSCManagerA 调用成功，就会开始执行下面这些代码

```
.text:00401000 lpCmdLine       = dword ptr  0Ch
.text:00401000 nShowCmd        = dword ptr  10h
.text:00401000
.text:00401000                 sub     esp, 28h
.text:00401003                 push    esi
.text:00401004                 push    0F003Fh         ; dwDesiredAccess
.text:00401009                 push    0               ; lpDatabaseName
.text:0040100B                 push    0               ; lpMachineName
.text:0040100D                 call    ds:OpenSCManagerA
.text:00401013                 test    eax, eax
.text:00401015                 jz      loc_401131
.text:0040101B                 push    0               ; lpPassword
.text:0040101D                 push    0               ; lpServiceStartName
.text:0040101F                 push    0               ; lpDependencies
.text:00401021                 push    0               ; lpdwTagId
.text:00401023                 push    0               ; lpLoadOrderGroup
.text:00401025                 push    offset BinaryPathName ; "C:\\Windows\\System32\\Lab10-03.sys"
.text:0040102A                 push    1               ; dwErrorControl
.text:0040102C                 push    3               ; dwStartType
.text:0040102E                 push    1               ; dwServiceType
.text:00401030                 push    0F01FFh         ; dwDesiredAccess
.text:00401035                 push    offset DisplayName ; "Process Helper"
.text:0040103A                 push    offset DisplayName ; "Process Helper"
.text:0040103F                 push    eax             ; hSCManager
.text:00401040                 call    ds:CreateServiceA
.text:00401046                 mov     esi, eax
```

我们可以看到一个字符串变量被压入了栈中，C:\\Windows\\System32\\Lab10-03.sys，这就是我们那个驱动文件，然后这里我们忽略入参为 0 的参数，主要看不为 0 的参数

其中的一个是 BinaryPathName，它的值是我们那个驱动文件的路径，这是用于指明服务的二进制文件的位置的参数，然后从上往下的下一个入参是 dwErrorControl 这个参数，这个参数是用于错误控制的，对于我们来说，没有太多的意义。我们注意到这里有个 dwStartType 这个参数，值为 3

这个的意义就是

用户可以使用“服务”控制面板实用程序启动服务。 用户可以在“开始参数”字段中为服务指定参数。 服务控制程序可以启动服务并使用 StartService 函数指定其参数。

服务启动时，SCM 执行以下步骤：

检索存储在数据库中的帐户信息。

登录服务帐户。

加载用户配置文件。

在暂停状态下创建服务。

将登录令牌分配给进程。

允许该过程执行。

下一个参数是 dwServiceType，他的值为 1，意义就是表明这是个驱动服务

最后的 lpServiceName 和 lpDisplayName 说明的是这个服务的名字是 Process Helper

如果调用 CreateServiceA 成功的话，下面执行 StartService，一旦执行这个之后，恶意驱动 Lab10-03.sys 就会被加载到内核中

```
push    0                ; lpPassword
push    0                ; lpServiceStartName
push    0                ; lpDependencies
push    0                ; lpdwTagId
push    0                ; lpLoadOrderGroup
push    offset BinaryPathName ; "C:\\Windows\\System32\\Lab10-03.sys
push    1                ; dwErrorControl
push    3                ; dwStartType
push    1                ; dwServiceType
push    0F01FFh          ; dwDesiredAccess
push    offset DisplayName ; "Process Helper"
push    offset DisplayName ; "Process Helper"
push    eax              ; hSCManager
call    ds:CreateServiceA
mov     esi, eax
test    esi, esi
jz      short loc_401057
```

```
push    0                ; lpServiceArgVectors
push    0                ; dwNumServiceArgs
push    esi              ; hService
call    ds:StartServiceA
```

这里创建了一个文件在\\.\ProcHelper 并作为一个句柄打开，还是如果一切顺利的话，会执行下面这个代码

```
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
```

```
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
```

这里有个新函数叫 DeviceIoControl 这个东西，这个函数的用途如下

将控制代码直接发送到指定的设备驱动程序，导致相应的设备执行相应的操作。

这里我们需要分析一个这个 DeviceIoControl 的各种用途，按照书上的说法，这里 DeviceIoControl 的参数 lpInBuffer 和 lpOutBuffer 被设置为了 Null 也就是 0 很不寻常，这意味着这个请求没有发送任何的信息到内核驱动中(lpInBuffer = 0)，并且内核驱动的反馈也是没有的(lpOutBuffer = 0)，然后还有个古怪的地方就是 dwIoControlCode 的值是 abcdedf01，这个值有点太人工了

然后下一个函数调用就是这个 CoCreateInstance，这个函数在 MSDN 中的解释就是

创建与指定的 CLSID 关联的类的单个未初始化对象。

当您只想在本地系统上创建一个对象时调用 CoCreateInstance。 要在远程系统上创建单个对象，请调用 CoCreateInstanceEx 函数。 要基于单个 CLSID 创建多个对象，请调用 CoGetClassObject 函数。

这是用于一个用于创建 COM 对象的

```
lea     eax, [esp+30h+pvarg]
push    eax                 ; pvarg
call    ds:VariantInit
push    offset psz          ; "http://www.malwareanalysisbook.com/ad.h"...
mov     [esp+34h+var_10], 3
mov     [esp+34h+var_8], 1
call    ds:SysAllocString
mov     edi, ds:Sleep
mov     esi, eax
```

```
loc_40112A:
call    ds:OleUninitialize
pop     edi
```

最后在这里调用了 Sleep 函数休眠了 0x7530h 毫秒，然后就是一直循环这个代码块，直到你关机为止

```
loc_401102:
lea     edx, [esp+30h+pvarg]
mov     eax, [esp+30h+ppv]
push    edx
lea     edx, [esp+34h+pvarg]
mov     ecx, [eax]
push    edx
lea     edx, [esp+38h+pvarg]
push    edx
lea     edx, [esp+3Ch+var_10]
push    edx
push    esi
push    eax
call    dword ptr [ecx+2Ch]
push    7530h               ; dwMilliseconds
call    edi ; Sleep
jmp     short loc_401102
```

```
loc_401131:
xor     eax, eax
pop     esi
add     esp, 28h
retn    10h
_WinMain@16 endp
```

Function name	Segment
sub_10606	PAGE
sub_1062A	PAGE
sub_10666	PAGE
sub_10706	INIT
sub_10794	INIT
DriverEntry	INIT

接下来我们分析 sys 文件，初始打开会报错，接下来就只能观察到如下的几个函数。不过不影响

我们直接进最后那个函数调用

```
mov     edi, edi
push    ebp
mov     ebp, esp
sub     esp, 14h
and     [ebp+DeviceObject], 0
push    esi
push    edi
mov     edi, ds:RtlInitUnicodeString
push    offset aDeviceProchelp ; "\\Device\\ProcHelper"
lea     eax, [ebp+DestinationString]
push    eax             ; DestinationString
call    edi ; RtlInitUnicodeString
mov     esi, [ebp+DriverObject]
lea     eax, [ebp+DeviceObject]
push    eax             ; DeviceObject
push    0               ; Exclusive
push    100h            ; DeviceCharacteristics
push    22h             ; DeviceType
lea     eax, [ebp+DestinationString]
push    eax             ; DeviceName
push    0               ; DeviceExtensionSize
push    esi             ; DriverObject
call    ds:IoCreateDevice
test    eax, eax
jl      short loc_10789
```

第一个函数调用是 RtlInitUnicodeString，初始化一个统计的 Unicode 字符串。

这个函数的标准定义如下，根据代码中的标识，我们可以得出以下结论

VOID WINAPI RtlInitUnicodeString(

Inout PUNICODE_STRING DestinationString = eax,

_In_opt_ PCWSTR SourceString = \\Device\\ProcHelper);

其中，DestinationString 是计数的 Unicode 字符串被初始化的缓冲区。如果未指定 SourceString，则长度初始化为零，而 SourceString 是可选指针，用于初始化已计数字符串的以空字符结尾的 Unicode 字符串

```
mov     eax, offset sub_10606
mov     [esi+38h], eax
mov     [esi+40h], eax
push    offset word_107DE ; SourceString
lea     eax, [ebp+SymbolicLinkName]
push    eax             ; DestinationString
mov     dword ptr [esi+70h], offset sub_10666
mov     dword ptr [esi+34h], offset sub_1062A
call    edi ; RtlInitUnicodeString
lea     eax, [ebp+DestinationString]
push    eax             ; DeviceName
lea     eax, [ebp+SymbolicLinkName]
push    eax             ; SymbolicLinkName
call    ds:IoCreateSymbolicLink
mov     esi, eax
test    esi, esi
jge     short loc_10787
```

RtlInitUnicodeString 这个函数有两个入参，我们往上找两个 push 的代码，倒数第一个是 eax，倒数第二个是 word_107DE，现在我们来看看这个倒数第二个是什么东西

这个 ProcHelper 就是这个驱动的名字，注意到这点我们继续

这里初始化了一个字符串，然后下面就是一个调用 IoCreateSymbolicLink

IoCreateSymbolicLink 例程在设备对象名称和设备的用户可见名称之间建立符号链接

定义如下

NTSTATUS IoCreateSymbolicLink(

_In_ PUNICODE_STRING SymbolicLinkName,

_In_ PUNICODE_STRING DeviceName);

这个 IoCreateSymbolicLink 创建了一个符号链接供用户态的应用程序访问这个设备

```
push    [ebp+DeviceObject] ; DeviceObject
call    ds:IoDeleteDevice
```

最后一个调用是 IoDeleteDevice 这个函数，这个函数删除驱动之后就退出了

下一步我们开始连上 WinDbg 来进行内核的分析

```
0: kd> !Object \Driver
Object: e136c538  Type: (821eb428) Directory
    ObjectHeader: e136c520 (old version)
    HandleCount: 0  PointerCount: 85
    Directory Object: e10013d0  Name: Driver

    Hash Address  Type                    Name
    ---- -------  ----                    ----
     00  81f5a3b8 Driver                  Beep
         81d02490 Driver                  NDIS
         82039d58 Driver                  KSecDD
     01  820f7e00 Driver                  Mouclass
         8208c388 Driver                  FsVga
         81ffcc08 Driver                  Raspti
         820ec5e0 Driver                  es1371
         81f63da0 Driver                  NwlnkIpx
     02  820208e0 Driver                  vmx_svga
```

```
NTSTATUS __stdcall DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
{
  sub_10794();
  return sub_10706(DriverObject, RegistryPath);
}
```

和上面一个实验类似！修改链表节点方式来实现隐藏。

2.一旦程序运行，你怎样停止它？

解答：书上的说法是重启，也只有这种把法了

3.它的内核组件做了什么操作？

解答： 修改了进程链接表的结构，隐藏了自己的 LIST_ENTRY，通过那个偏移为 0xe 的函数，这个函数我们现在还不知道怎么知道把偏移量和函数名对应起来，因为我们也看了 wdm.h，根本找不到这个函数，可执行文件调用了 DeviceIoControl 之后，驱动把进程隐藏

YARA 规则的编写

根据字符串，编写以下的 yara 规则

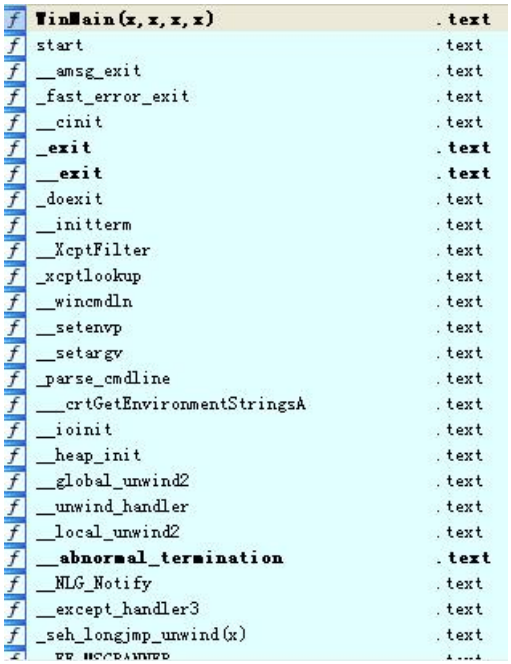
```
import "pe"
rule UrlRequest {
        strings:
                $http = "http"
                $com = /[a-zA-Z0-9_]*.com/
        condition:
                $http or $com}
rule EXE {
        strings:
                $exe = /[a-zA-Z0-9_]*.exe/
        condition:
                $exe}
```

```
rule Regedit {
	strings:
		$system = "Registry"
		$software = "SOFTWARE"
	condition:
		$system or $software }
rule DriverFile {
	strings:
		$name = ".sys"
	condition:
		$name}
rule Device {
	strings:
		$name = "Device"
	condition:
		$name}
rule Service {
	strings:
		$create = "CreateService"
		$start = "StartService"
	condition:
		$create or $start }
rule ResourceFile {
	strings:
		$name = ".rsrc"
	condition:
		$name}
```

```
C:\Users\53653_000>C:\Users\53653_000\Desktop\yara64.exe C:\Users\53653_000\Desk
top\lab10.yar "C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Pra
ctical Malware Analysis Labs\BinaryCollection\Chapter_10L"
EXE C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malw
are Analysis Labs\BinaryCollection\Chapter_10L\Lab10-01.sys
Regedit C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical
Malware Analysis Labs\BinaryCollection\Chapter_10L\Lab10-01.sys
ResourceFile C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Pract
ical Malware Analysis Labs\BinaryCollection\Chapter_10L\Lab10-01.sys
EXE C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malw
are Analysis Labs\BinaryCollection\Chapter_10L\Lab10-03.sys
Device C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical M
alware Analysis Labs\BinaryCollection\Chapter_10L\Lab10-03.sys
ResourceFile C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Pract
ical Malware Analysis Labs\BinaryCollection\Chapter_10L\Lab10-03.sys
EXE C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical Malw
are Analysis Labs\BinaryCollection\Chapter_10L\Lab10-02.exe
DriverFile C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practic
al Malware Analysis Labs\BinaryCollection\Chapter_10L\Lab10-02.exe
Service C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical
Malware Analysis Labs\BinaryCollection\Chapter_10L\Lab10-02.exe
DriverFile C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practic
al Malware Analysis Labs\BinaryCollection\Chapter_10L\Lab10-03.exe
Device C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical M
alware Analysis Labs\BinaryCollection\Chapter_10L\Lab10-03.exe
Service C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical
Malware Analysis Labs\BinaryCollection\Chapter_10L\Lab10-03.exe
DriverFile C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practic
al Malware Analysis Labs\BinaryCollection\Chapter_10L\Lab10-01.exe
Service C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Practical
Malware Analysis Labs\BinaryCollection\Chapter_10L\Lab10-01.exe
ResourceFile C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Pract
ical Malware Analysis Labs\BinaryCollection\Chapter_10L\Lab10-02.exe
ResourceFile C:\Users\53653_000\Desktop\duku\PracticalMalwareAnalysis-Labs\Pract
ical Malware Analysis Labs\BinaryCollection\Chapter_10L\Lab10-01.exe
```

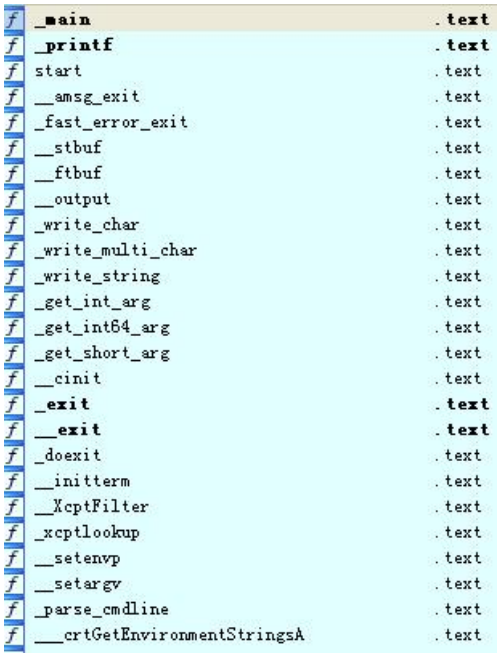
IDA python 的编写：
根据函数名称，编写相应的 ida python 脚本
Lab10-01.exe

```
from idaapi import *
# 设置颜色
def judgeAduit(addr):
    '''
    not safe function handler
    '''
    MakeComm(addr,"### AUDIT HERE ###")
    SetColor(addr,CIC_ITEM,0x0000ff)    #set backgroud to red
    pass
# 函数标识
def flagCalls(danger_funcs):
    '''
    not safe function finder
    '''
    count = 0
    for func in danger_funcs:
        faddr = LocByName( func )
        if faddr != BADADDR:
            # Grab the cross-references to this address
            cross_refs = CodeRefsTo( faddr, 0 )
            for addr in cross_refs:
                count += 1
                Message("%s[%d] calls 0x%08x\n"%(func,count,addr))
                judgeAduit(addr)
if __name__ == '__main__':
    '''
    handle all not safe functions
    '''
    print "--------------------------------"
    # 列表存储需要识别的函数
    danger_funcs = ["WinMain(x,x,x,x)","RtlUnwind","__alloca_probe","_strncpy","___sbh_heap_init"]
    flagCalls(danger_funcs)
```



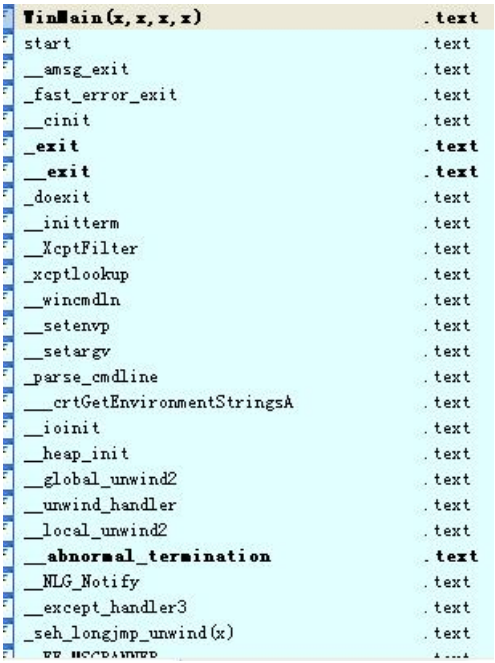
Lab10-02.exe

```
from idaapi import *
# 设置颜色
def judgeAduit(addr):
    '''
    not safe function handler
    '''
    MakeComm(addr,"### AUDIT HERE ###")
    SetColor(addr,CIC_ITEM,0x0000ff)    #set backgroud to red
    pass
# 函数标识
def flagCalls(danger_funcs):
    '''
    not safe function finder
    '''
    count = 0
    for func in danger_funcs:
        faddr = LocByName( func )
        if faddr != BADADDR:
            # Grab the cross-references to this address
            cross_refs = CodeRefsTo( faddr, 0 )
            for addr in cross_refs:
                count += 1
                Message("%s[%d] calls 0x%08x\n"%(func,count,addr))
```



```
                    judgeAduit(addr)
if __name__ == '__main__':
        handle all not safe functions
            # 列表存储需要识别的函数
    danger_funcs = ["___crtGetStringTypeA","__get_osfhandle","__alloca_probe","___crtMessageBoxA","__abnormal_termination"]
    flagCalls(danger_funcs)
```

Lab10-03.exe




```
from idaapi import *
# 设置颜色
def judgeAduit(addr):
    not safe function handler
    MakeComm(addr,"### AUDIT HERE ###")
    SetColor(addr,CIC_ITEM,0x0000ff)    #set backgroud to red
    pass
# 函数标识
def flagCalls(danger_funcs):
    not safe function finder
    count = 0
    for func in danger_funcs:
        faddr = LocByName( func )
        if faddr != BADADDR:
            # Grab the cross-references to this address
            cross_refs = CodeRefsTo( faddr, 0 )
            for addr in cross_refs:
                count += 1
                Message("%s[%d] calls 0x%08x\n"%(func,count,addr))
                judgeAduit(addr)
if __name__ == '__main__':
    handle all not safe functions
    # 列表存储需要识别的函数
    danger_funcs = ["__alloca_probe","___sbh_find_block","___sbh_alloc_new_region","___crtLCMapStringA","___crtGetEnvironmentStringsA"]
    flagCalls(danger_funcs)
```

四、实验结论及心得体会

本次实验，我研究了关于 windbg 内核调试的功能和实现方法，这里需要我们不断地在两个内核当中去寻找相关的操作，同时用到了我们学习到的操作系统知识，对这两门课的联系也更加深刻。我们可以通过 windbg 观察到内核的许多操作，是程序分析不止局限于 ida 当中的静态汇编，动态对于汇编的思路也更加的明确。

本次实验当中我们还学到了 ida 重定位等知识，这些在我们之前不了解内核运行的时候并不熟悉，但是在程序与 ida 协同操作运行之后，就可以发现有一些内核上的特殊操作，值得我们去注意。

这一次的实验是恶意代码与防治分析的`Lab10`实验，对理论课上讲的`IDA Python`编写技术有了一定的了解，也对`IDA Pro`的使用比如说交叉引用、语句跳转、反汇编分析等更加的熟练。

在本次实验中，也对所检测程序编写了相应的 yara 规则，对于 yara 规则的编写也更加的熟练。