

实验8 使用WinDbg调试内核级恶意代码

1 实验目的

掌握内核级程序动态调试技术，能够使用工具初步分析恶意代码。

2 实验前提

- 请安装虚拟机 winXPenSP3 调试环境。
- 安装WinDbg
 - windows 7/8/10 下，安装winsdk时，单独选择debugger
 - 或可以试用 microsoft store中的windows debugger preview

本实验包括一个驱动程序和一个可执行文件。

- 可执行文件是Lab10-l.exe
- 驱动程序是Lab10-0l.sys。
- 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。

3 实验内容

- 安装、配置WinDbg内核调试环境
使用WinDbg等工具分析：
 - Lab10-01.exe和Lab10-01.sys
 - Lab10-02.exe
 - Lab10-03.exe

4 实验步骤

4.1 安装、配置WinDbg内核调试环境

WinDbg(常被读作“Windbag”)是微软提供的一个免费调试器。

虽然在恶意代码分析中 WinDbg 不如 OllyDbg 调试器那么流行，但是它有很多独特的优点，比如它支持内核调试。

4.1.1 内核调试方法概述

内核调试比起用户模式调试来说更加复杂，因为进行内核调试时，操作系统将被冻结，这种情况下不可能运行调试器。因此，调试内核的常用方法是使用VMware。

与用户态调试不同，内核调试需要一些初始化设置。

- 首先需要设置虚拟操作系统并开启内核调试；
- 然后配置VMware使虚拟机与宿主系统之间有一条虚拟化的串口；
- 同时还应该配置宿主操作系统的 WinDbg。

4.1.2 虚拟机（被调试者）的设置

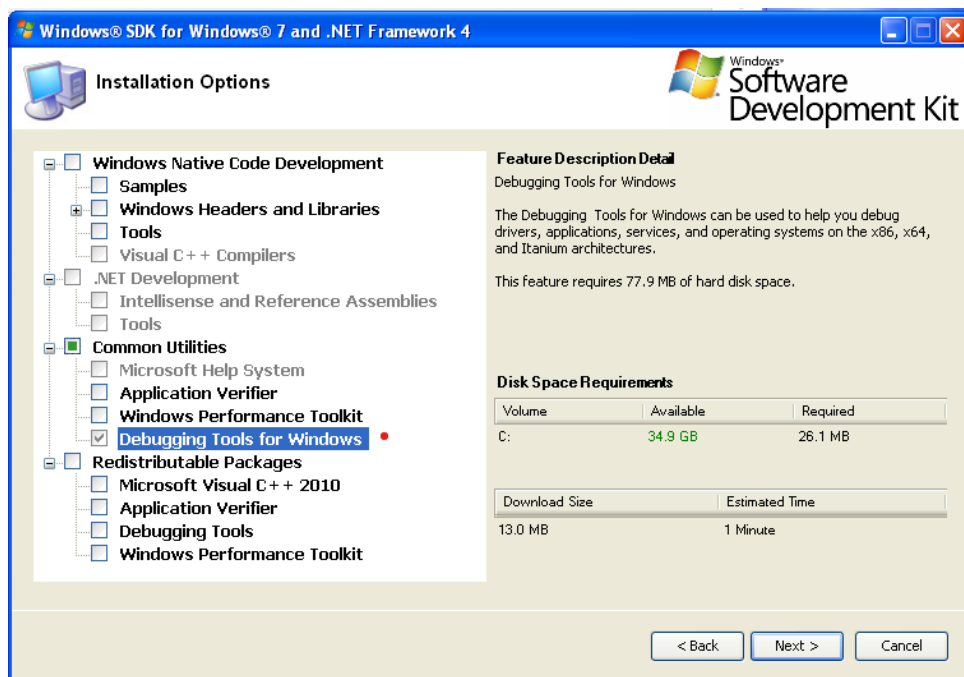
虚拟机内安装WinDbg

虚拟机内部用的WinDbg，可以用于调试用户态的程序，功能与OllyDbg相同。

在虚拟机中，打开微软SDK下载网址：<https://www.microsoft.com/en-us/download/details.aspx?id=8279>，下载Windows SDK 7。

Windows SDK 7 适用于 Windows xp 和 windows 7 等产品。

下载完毕后点击安装，在安装过程中选中 debugger，其他的安装选项可以不选。



其他按默认完成。

设置虚拟机（以windowxp为例）

在进行以下配置前请重新安装 **vmware tools**，安装完毕，重启系统后再进行如下操作。

虚拟操作系统的设置是编辑 **C:\boot.ini** (Windows XP)下请确保文件夹选项设置为显示隐藏文件)，该文件在系统中通常是隐藏的。建议在编辑**boot.ini**文件之前，为你的虚拟操作系统做一个快照，如果配置文件错误或者损坏了**boot.ini**，你可以使用快照还原系统。

下列代码清单，是设置内核调试功能之后，**boot.ini**文件的内容：

```
[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="WinXP Pro with Kernel Debugging"
/noexecute=optin /fastdetect /debug /debugport=com1 /baudrate=115200
```

说明：上述两条 **multi(0)** 开头的语句必须在一行中，不可换行。

当再次开机运行你的虚拟操作系统时，系统会提供一个开启内核调试的选项让你选择。

另外，系统会给你30秒的时间，决定是否以调试模式启动系统。如果想要连接内核调试器 (即 WinDbg)，你需要在每次开机时选择调试版本启动项。

注意:操作系统以调试模式启动，并不意味着需要连接调试器，在没有连接调试器的情况下，系统也会正常运行。

为虚拟机增加串口设备

下一步，需要设置VMware,在虚拟操作系统和宿主操作系统之间创建一个虚拟连接。为此，我们在VMware上添加一个新的设备来使用宿主系统中的一个命名管道上的串口。

下面是关闭虚拟机后，添加设备的步骤:

- 1.单击**VMWare Settings**，然后会弹出**VMware**设置对话框。
- 2.在**VMware**设置对话框中，单击右下角的**Add**按钮，在弹出的设备类型选择窗口中选择**Serial Port**，然后单击下一步。
- 3.在请求串口类型的对话框中，选择**Output to Named Pipe**，然后单击下一步。
- 4.在接下来的窗口中，输入 **\\.\pipe\com_1** 对管道进行命名，然后选择：
 - **This end is the server** 该端是服务器
 - **The other end is an application** 另一端是应用程序
- 5.选择轮询时主动放弃CPU。

当完成串口的添加后，虚拟机的设置对话框应该与下图所示的串口设备的配置类似。



完成虚拟机的配置后启动虚拟机。

4.1.3 宿主机（调试器）端安装、设置

宿主机Windbg安装

Windows 10中可用的Windbg可以从两方面获取：

- Microsoft Store 中搜索 WinDbg Preview，然后点击安装。
- 从 <https://developer.microsoft.com/zh-cn/windows/downloads/sdk-archive> 寻找合适的SDK安装。

推荐第一种方法，较为简单。Windbg preview是新版本的工具，界面设计更人性化。

宿主机Windbg设置

在宿主操作系统中，使用下列步骤使WinDbg连接虚拟机并开始调试内核。

若安装了 WinDbg preview

- 1.启动WinDbg
- 2.打开File——Settings——Debugging settings，键入如下内容：

```
srv*c:\localsymbols*http://msdl.microsoft.com/download/symbols
```

- 3.选择File —— Attach Kernel，之后单击COM标签，然后输入文件名和先前在oot.ini文件中设置的波特率，本例中我们设置为115200；选中Pipe复选框后确定。

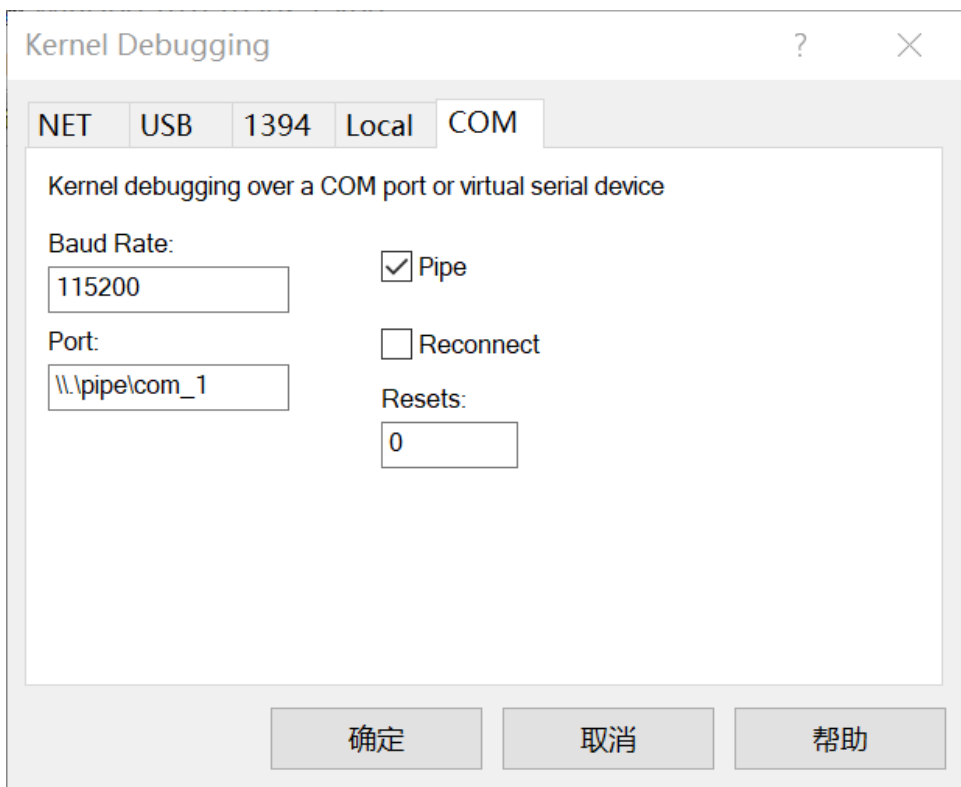
若安装了 WinDbg

- 1.启动WinDbg
- 2.打开菜单File——Symbol Search Path，键入如下内容：

```
srv*c:\localsymbols*http://msdl.microsoft.com/download/symbols
```

- 3.选择File--Kernel Debug，之后单击COM标签，然后输入文件名和先前在boot.ini文件中设置的波特率，本例中我们设置为115200；选中Pipe复选框后确定。

设置窗口如下图所示：



如果虚拟操作系统处于运行状态，调试器会在数秒内连接到虚拟机操作系统。如果虚拟操作系统没有运行，调试器将处于等待，直到虚拟操作系统启动，启动过程中调试器将连接到被调试系统。

调试器连接后，为了更加全面地获取到调试过程发生的事件，建议在调试过程中启用详细信息输出功能。启用详细信息输出功能后，每当驱动程序被加载和卸载时，你将会得到通知。这些信息在某些情况下可以帮助你识别恶意驱动的加载。

4.2 分析Lab10-01.exe和Lab10-01.sys

1.这个程序是否直接修改了注册表(使用procmon来检查)?

提示：使用procmon监视这个程序，你会看到唯一写注册表的地方是键值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。事实上，如果我们使用regshot，会发现这个程序还是修改了注册表的一些键值，这些修改是从内核直接修改的，不能被procmon记录。

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？

提示：要设一个断点来查看内核发生了什么，你必须使用一个运行在虚拟机中的WinDbg实例打开可执行文件10-01.exe。宿主机中的windbg用于调试内核（观察10-01.sys）。当Lab10-01.exe在虚拟机中被新停后，使用!drvobj命令获得驱动设备的句柄，它包含一个卸载函数的指针。接下来，在驱动的卸载函数上设置一个断点。重启lab 10-01.exe之后，断点将会被触发。

在试图使用WinDbg分析这个驱动之前，我们可以用IDA Pro打开这个驱动，来检查它的DriverEntry函数。这个函数是驱动的入口点，但是它不是DriverEntry函数。编译器在DriverEntry的周围插入封装代码。真正的DriverEntry函数位于sub_10906处。

```
; Attributes: bp-based frame

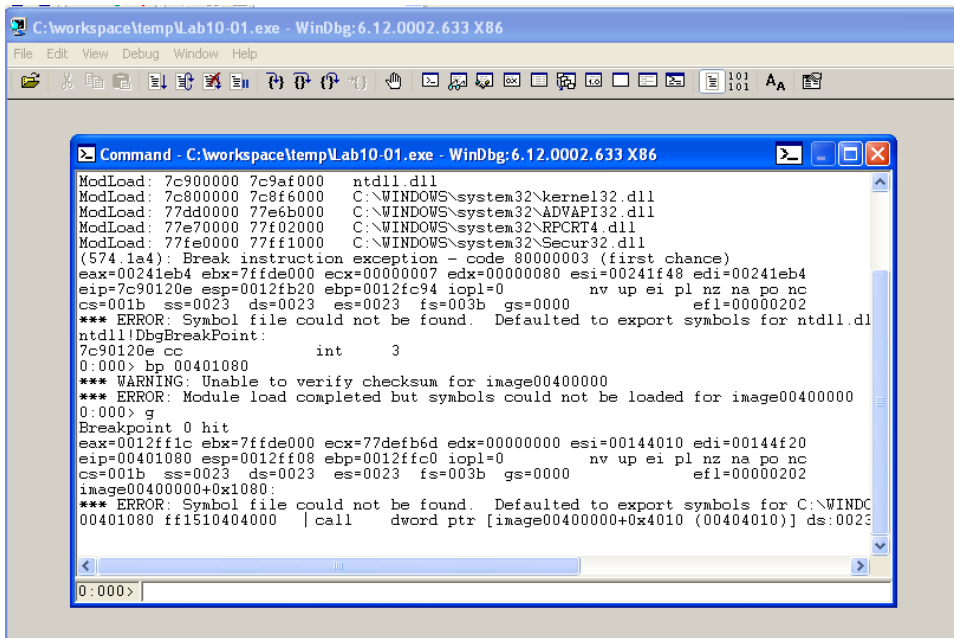
; NTSTATUS __stdcall DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
public DriverEntry
DriverEntry proc near

DriverObject= dword ptr 8
RegistryPath= dword ptr 0Ch

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

DriverEntry函数的主体部分似乎将一个偏移量移入到一个内存位置。除此之外，它没有进行任何函数调用，也没有与系统进行交互。

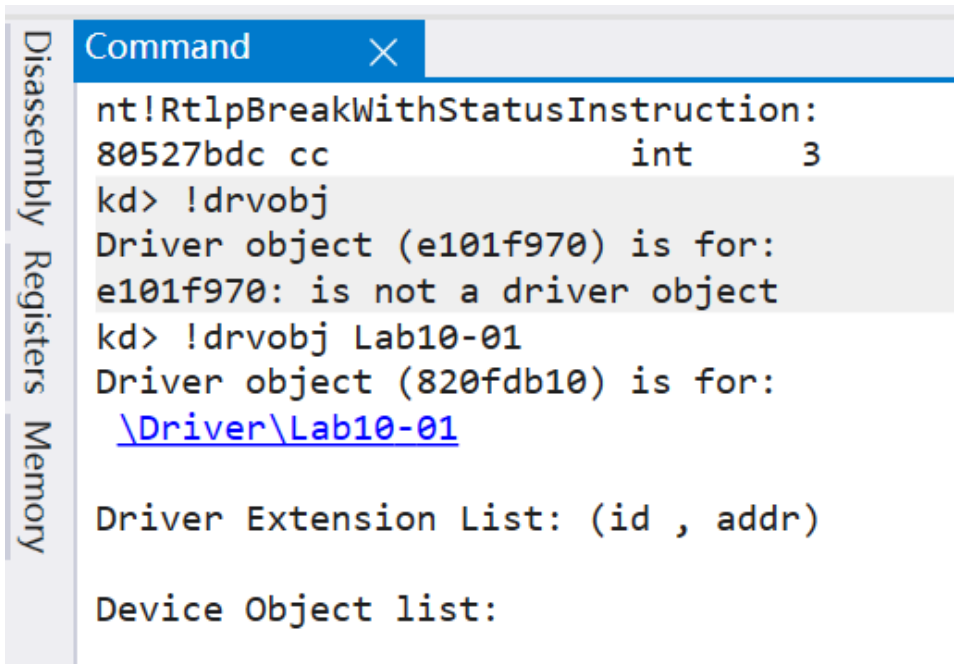
为了在Lab10-01.sys载入内存后，使用WinDbg分析它，在虚拟机中，我们将可执行程序载入到Windbg中。使用下列命令，我们在驱动加载和卸载之间设置一个断点（在ControlService调用上，因为这个函数调用时它的第二个参数是1，查微软文档可知这是要SERVICE_CONTROL_STOP，之后sys就会被卸载，所以在此断点。）：bp 00401080。然后，我们启动程序直到断点命中。



```
Command - C:\workspace\temp\Lab10-01.exe - WinDbg:6.12.0002.633 X86
ModLoad: 7c900000 7c9af000 ntdll.dll
ModLoad: 7c800000 7c8f6000 C:\WINDOWS\system32\kernel32.dll
ModLoad: 77dd0000 77e6b000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e70000 77f02000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77fe0000 77ff1000 C:\WINDOWS\system32\Secur32.dll
(574.1a4): Break instruction exception - code 80000003 (first chance)
eax=00241eb4 ebx=7ffde000 ecx=00000007 edx=00000080 esi=00241f48 edi=00241eb4
eip=7c90120e esp=0012fb20 ebp=0012fc94 iopl=0         nv up ei pl zr 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:
7c90120e cc          int     3
0:000> bp 00401080
*** WARNING: Unable to verify checksum for image00400000
*** ERROR: Module load completed but symbols could not be loaded for image00400000
0:000> g
Breakpoint 0 hit
eax=0012f1c ebx=7ffde000 ecx=77defb6d edx=00000000 esi=00144010 edi=00144f20
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:
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\WINDC
00401080 ff1510404000 | call     dword ptr [image00400000+0x4010 (00404010)] ds:0023
0:000>
```

一旦程序在断点处暂停，我们就跳出虚拟机，以便连接内核调试器，并且获取关于Lab10-01.sys的信息。我们打开宿主机中的WinDbg，选择File-Kernel Debug,设置管道为 \\.\pipe\com_1，波特率 (baud rate)为115200，将宿主机上运行的WinDbg实例与虚拟机中的内核连接上。我们知道要分析的服务叫做Lab10-01,所以运行：`!drvobj`，来获取对象。

命令 `!drvobj` 输出给我们提供驱动对象的地址。因为在设备对象列表中没有设备列出，所以我们明白这个驱动没有提供用户空间中应用程序访问的设备。



```
Command
nt!RtlpBreakWithStatusInstruction:
80527bdc cc          int     3
kd> !drvobj
Driver object (e101f970) is for:
e101f970: is not a driver object
kd> !drvobj Lab10-01
Driver object (820fdb10) is for:
   \Driver\Lab10-01

Driver Extension List: (id , addr)

Device Object list:
```

注意:为了解决任何难以定位的服务名，你可以使用 `!object \Driver` 命令获取当前内核中的驱动对象列表。一旦获得了驱动对象地址，你就可以使用 `dt` 命令查看它。

```
Command X
kd> !object \Driver
Object: e101f970 Type: (825ed348) Directory
ObjectHeader: e101f958 (old version)
HandleCount: 0 PointerCount: 86
Directory Object: e1001138 Name: Driver

Hash Address Type Name
----
00 821d3550 Driver Beep
824e7298 Driver NDIS
824e7030 Driver KSecDD
01 82191f38 Driver FsVga
825e2d40 Driver Raspti
824fd030 Driver es1371
823f7bc0 Driver Mouclass
02 824053c0 Driver vmx_svga
03 82458da0 Driver Fips
8207ccd0 Driver Kbdclass
04 824af030 Driver VgaSave
824ae6e8 Driver NDPProxy
8253a218 Driver Compbatt
05 823982c0 Driver Ptilink
8259d6f8 Driver MountMgr
820f3648 Driver wdmaud
06 820fdb10 Driver Lab10-01
07 825e0888 Driver dmload
```

执行命令 `dt _Driver_OBJECT 820fdb10`

```
kd> dt _Driver_OBJECT 820fdb10
nt!_DRIVER_OBJECT
+0x000 Type : 0n4
+0x002 Size : 0n168
+0x004 DeviceObject : (null)
+0x008 Flags : 0x12
+0x00c DriverStart : 0xf8c62000 Void
+0x010 DriverSize : 0xe80
+0x014 DriverSection : 0x81e77108 Void
+0x018 DriverExtension : 0x820fdbb8 _DRIVER_EXTENSION
+0x01c DriverName : _UNICODE_STRING "\Driver\Lab10-01"
+0x024 HardwareDatabase : 0x80670ae0 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
+0x028 FastIoDispatch : (null)
+0x02c DriverInit : 0xf8c62959 long +0
+0x030 DriverStartIo : (null)
+0x034 DriverUnload : 0xf8c62486 void +0
+0x038 MajorFunction : [28] 0x804f354a long nt!IoInvalidDeviceRequest+0
```

我们尝试确定驱动对象卸载时调用的函数：偏移量0x034的信息DriverUnload。然后，我们使用如下命令设置一个断点：`bp 0xf8c62486`。

设置断点后，恢复内核运行。回到虚拟机内，看运行可执行程序的windbg，恢复它的运行。命中断点后，虚拟机卡死。此时回到宿主机中的windbg。单步调试代码，发现程序3次调用了RtlCreateRegistryKey函数，创建了一些注册表键，然后调用了2次RtlWriteRegistryValue函数，在这两个地方设置了EnableFirewall值为0。

如果0xf8d4e486处的卸载函数很长或者很复杂，那么使用WinDbg很难分析它。多数情况下，我们确定了函数的位置，使IDA Pro比较容易分析，因为IDA Pro在分析函数功能做的很好。然

后，WinDbg中的函数位置与IDA Pro中的函数位置不同，所以为了在IDA pro中查看，必须进行手动计算。我们必须使用lm命令，计算函数从windbg加载文件开始处的偏移量。

例如：文件被加载windbg中的 0xf8d4e000。若卸载函数数位于。我们从0xf8d4e486减去 0xf8d4e000得到偏移量 0x486。然后在IDA Pro中跳到卸载函数。例如，若IDA Pro加载基地址是 0x00100000，那么我们在IDA Pro中地址0x00100486处找到卸载函数。>接下来我们使用静态分析和IDA Pro来验证我们在WinDbg中发现的东西。

另外一种方法，在IDA Pro中，我们可以通过选择Edit——Segments——Rebase Program来改变基地址，从0x00100000改为0xf7c47000。

```
; Attributes: bp-based frame
sub_10486 proc near
ValueData= dword ptr -4
mov     edi, edi
push    ebp
mov     ebp, esp
push    ecx
push    ebx
push    esi
mov     esi, ds:RtlCreateRegistryKey
push    edi
xor     edi, edi
push    offset Path      ; "\\Registry\\Machine\\SOFTWARE\\Policies\\Mic"...
push    edi              ; RelativeTo
mov     [ebp+ValueData], edi
call    esi ; RtlCreateRegistryKey
push    offset aRegistryMach_0 ; "\\Registry\\Machine\\SOFTWARE\\Policies\\Mic"...
push    edi              ; RelativeTo
call    esi ; RtlCreateRegistryKey
push    offset aRegistryMach_1 ; "\\Registry\\Machine\\SOFTWARE\\Policies\\Mic"...
push    edi              ; RelativeTo
call    esi ; RtlCreateRegistryKey
...
```

3.这个程序做了些什么？

提示：这个程序创建一个服务来加载驱动。然后，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，设置这些键值将禁用防火墙。

4.3 分析Lab10-02.exe

- 1.这个程序创建文件了吗？它创建了什么文件？
- 2.这个程序有内核组件吗？
- 3.这个程序做了些什么？

4.4 分析Lab10-03.exe

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Lab10-3.exe,驱动程序是Lab10-03.sys。

1.这个程序做了些什么？

2.一旦程序运行，你怎样停止它？

3.它的内核组件做了什么操作？