

**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

1. **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。

1. **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。

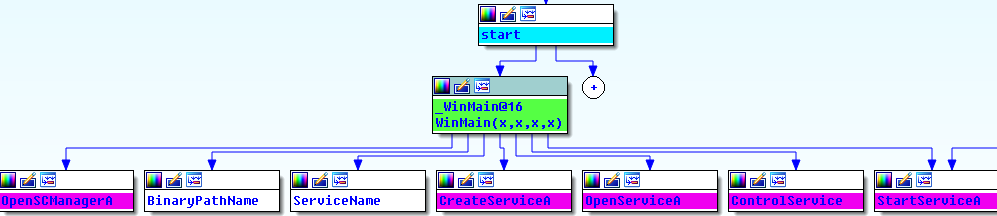
恶意代码静态分析：首先，通过将恶意代码加载到Windbg中，可以进行静态分析。这包括检查恶意代码的文件结构、导入的库、字符串和常量等信息，有助于了解恶意代码的基本行为和功能。

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。

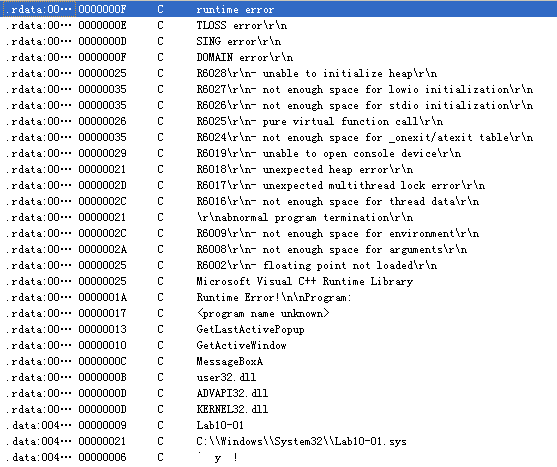
1. **实验过程**



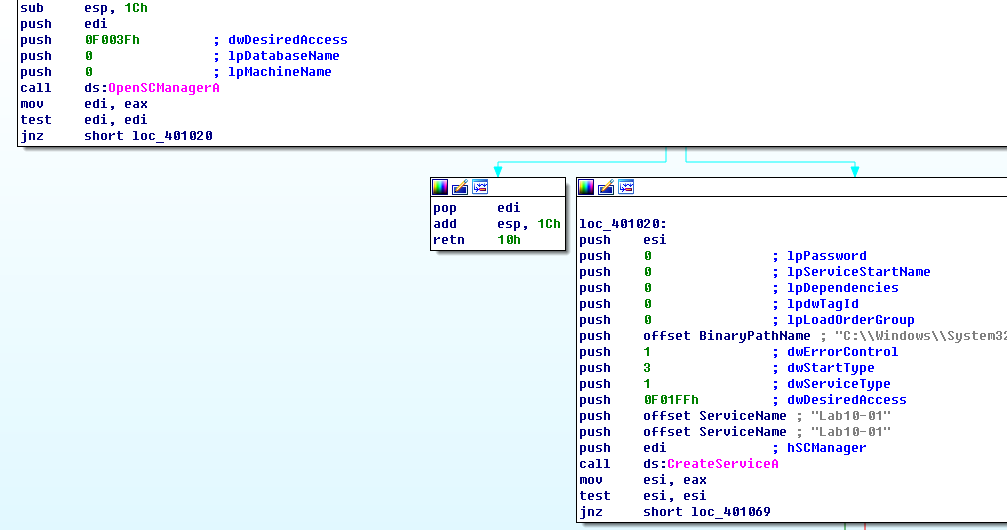
首先在ida pro当中打开相应的页面，在相关的页面下面我们可以看到这个进程的大致内容和运行逻辑。

反汇编代码如下：

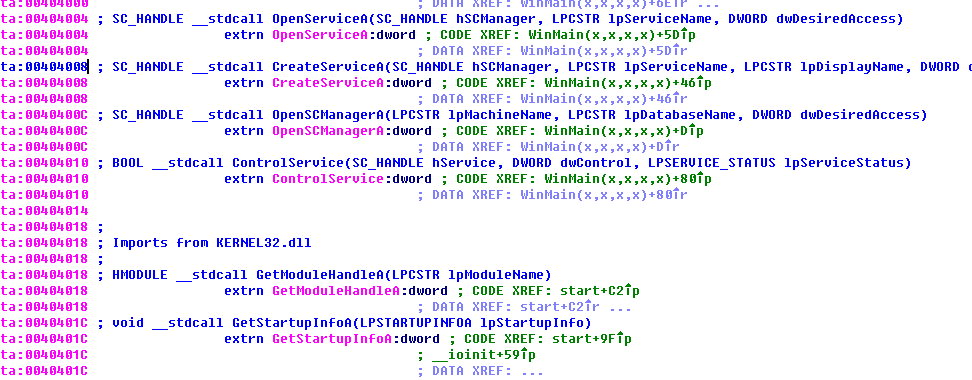


运行后查看：  


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。



可以看到出现了敏感文件路径。



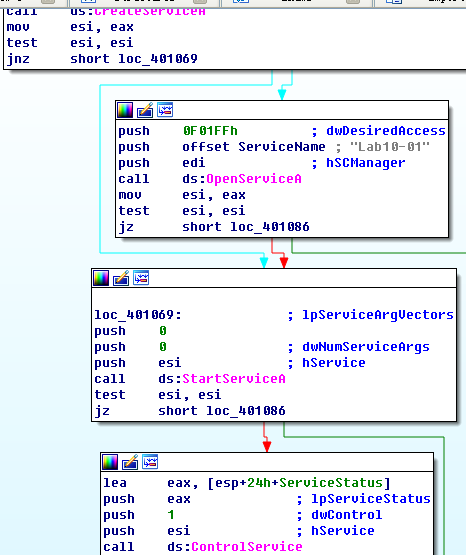
OpenSCManger:在指定及其上创建与服务控制管理程序的联系，并打开指定的数据库，返回的是一个服务管理器的句柄。

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。



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

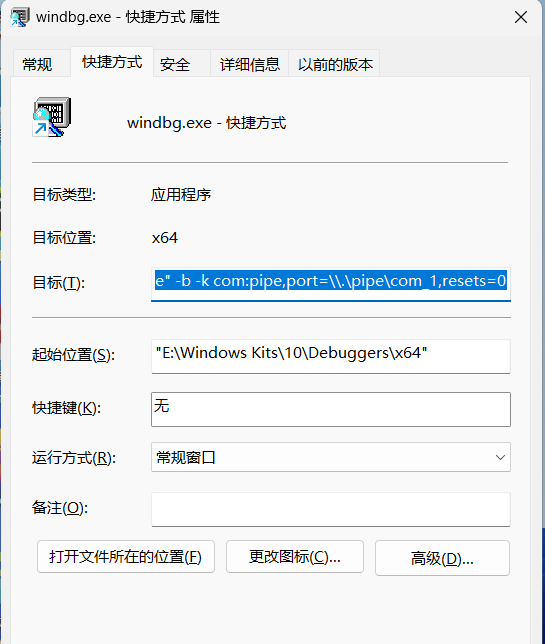
dwControl，要发送的控制码，此处为1，表示CONTROL \_SERVICE\_STOP,将会卸载驱动并调用驱动卸载的函数。==》此次实验就是要在运行过程中，打断点，看看这个程序究竟安装了什么驱动！因为它会自删除驱动。。。IpServiceStatus，返回值，指向存储服务最新状态的结构体Service，返回信息来自SCM中最近的服务状态报告。

使用windbg调试内核。首先是windbg的配置：

1.对串行端口的配置

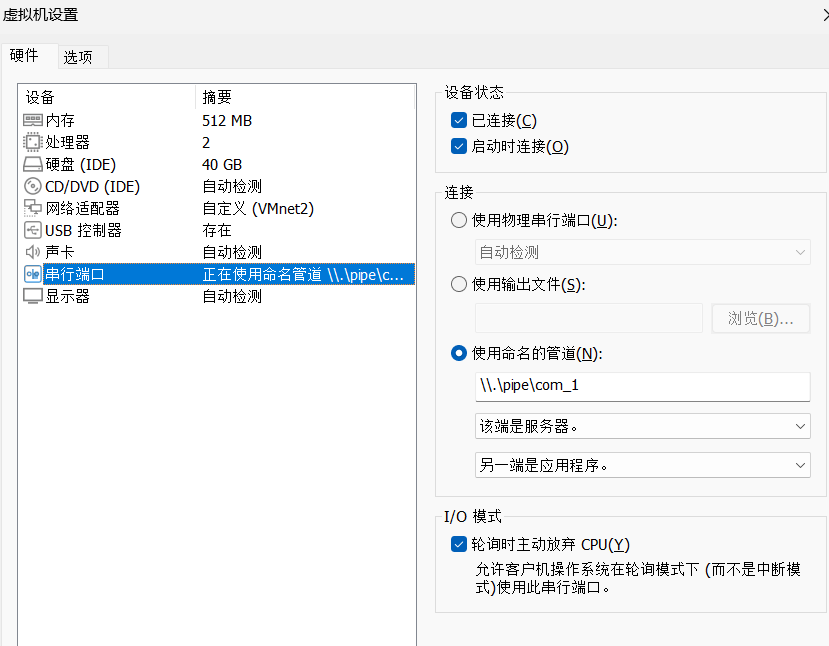
我们在相关网站下载windbg之后，将exe文件创建快捷方式，之后拖到桌面上，打开其属性在目标里面将其串行端口更改。

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



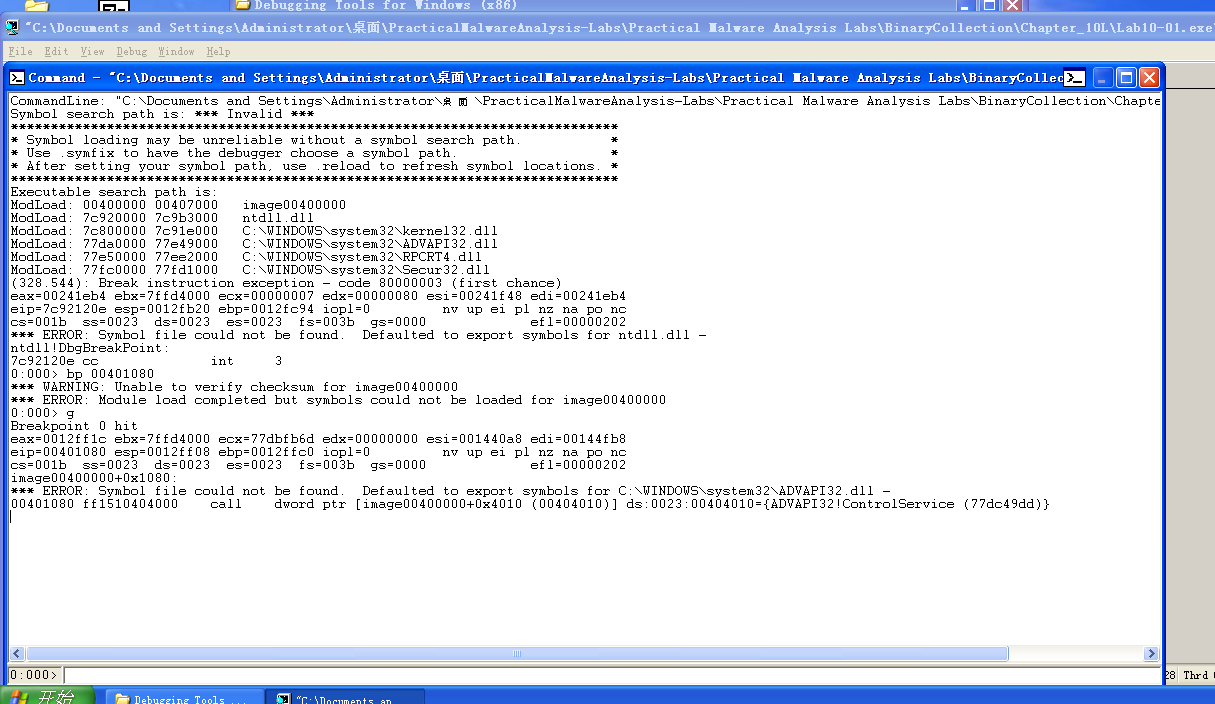
首先在虚拟机当中设置断点，之后我们使用命令g运行，得到相应的输出。

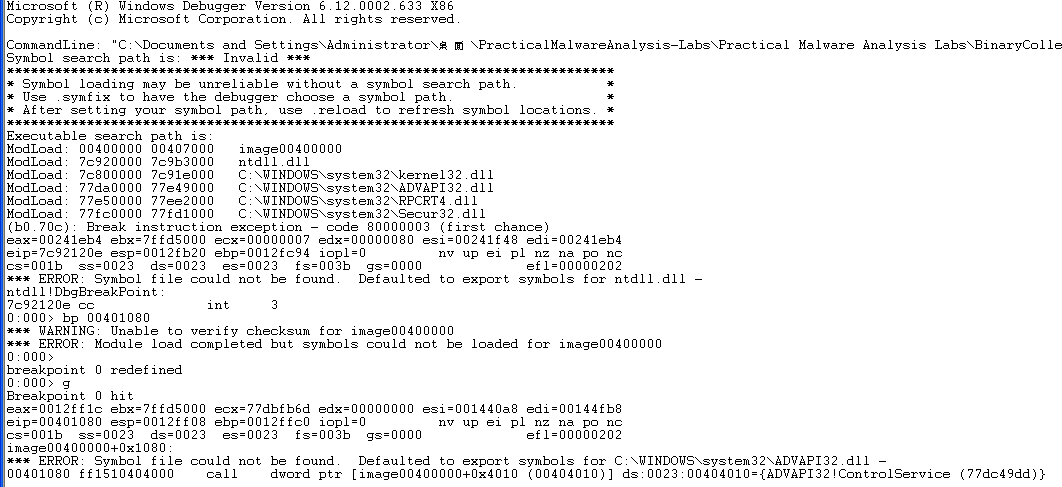
可以看到程序在这个点进行了调用dword进行了相应的输出。



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。

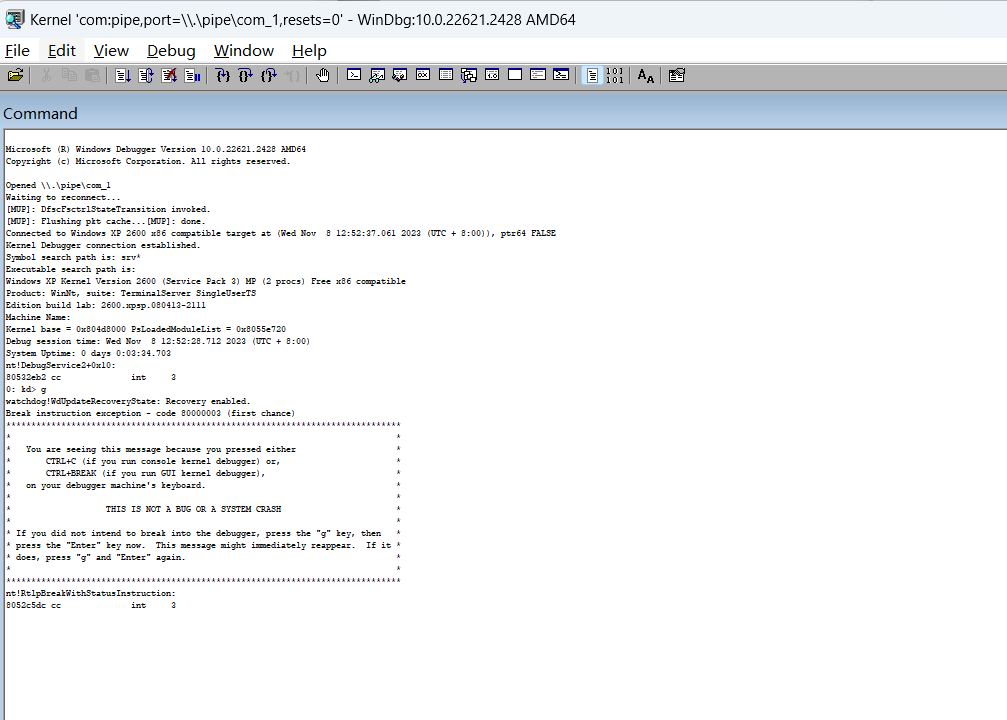




在虚拟机中使用windbg加载lob 10-01.exe

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”

1. 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

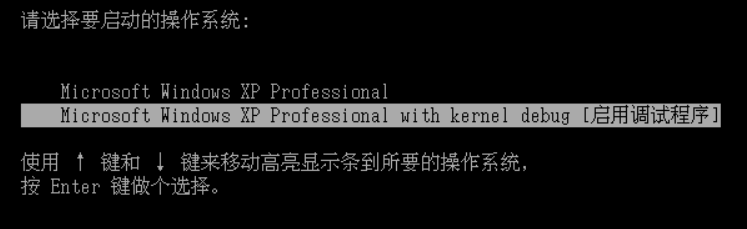
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS

[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

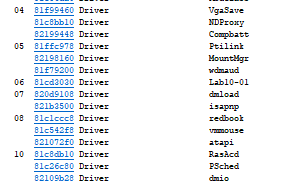
重启之后，可以看到我们选择调试模式进入winxp



然后windbg 点击kernel debug：

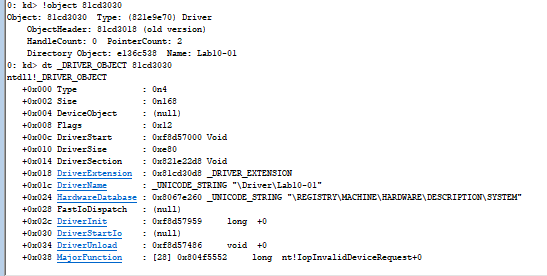
之后在虚拟机当中下断点运行之后在主机下断点停下，按g执行，这样程序就能够停在内核执行并且不删除的状态。

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。

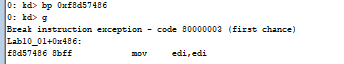


可以看到Lab10-01.exe是在81cd3030这里运行的。

使用dt \_DRIVER\_OBJECT 地址 来解析地址的数据结构

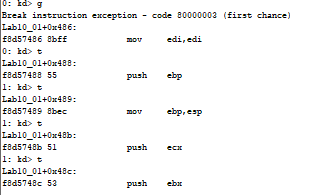


重点观察DriverUnload函数，地址为0xf8d57486，使用bp指令在此加断点,并使用g指令恢复内核的执行。

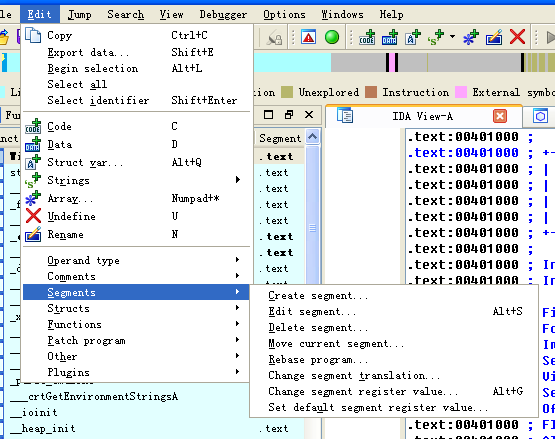


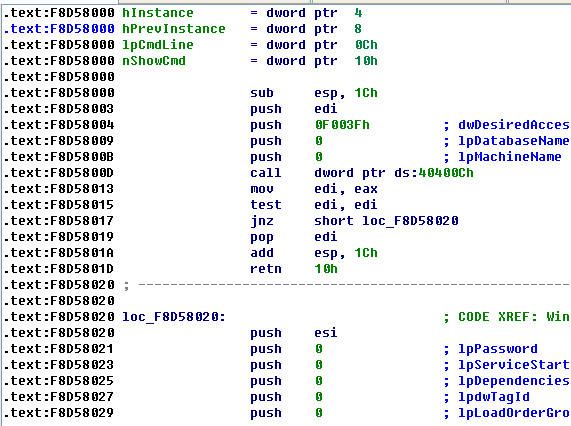
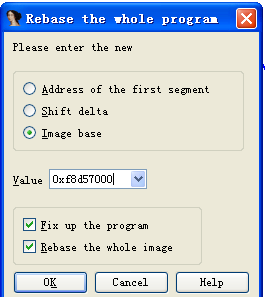
通过按t单步执行下一条指令

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。



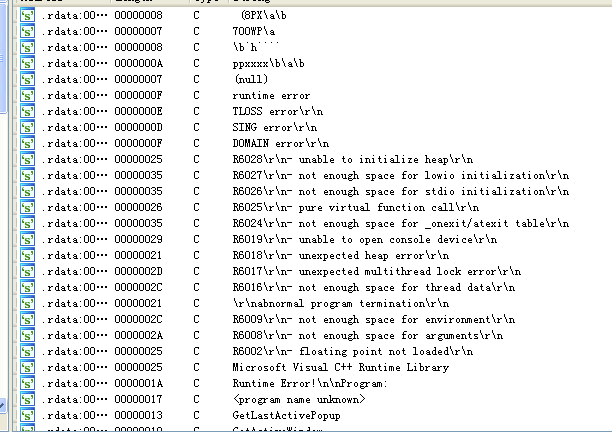
在ida中driver的默认地址的sys文件是从0x00010000开始的，所以函数卸载代码对应的地址为0x00010468。另外一个方法则是重新设置ida默认的基地址



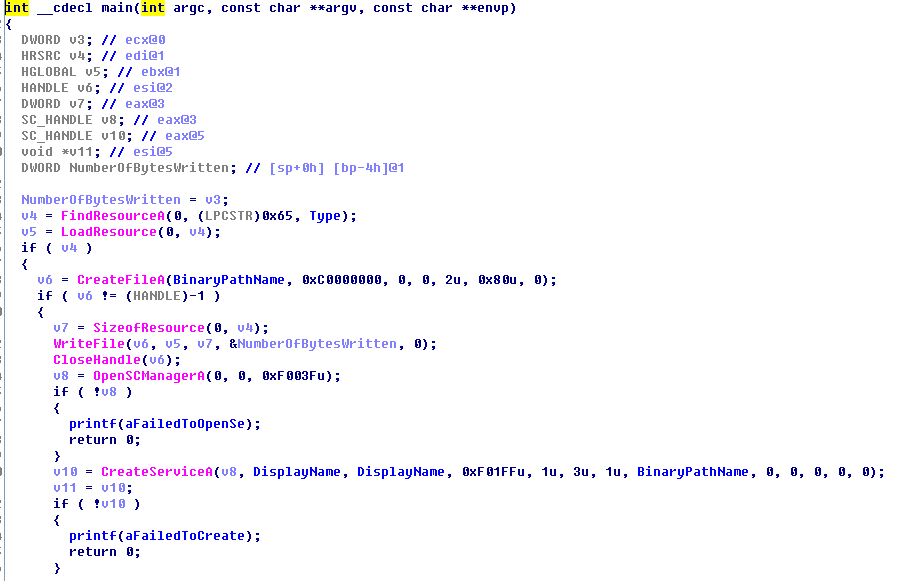


Lab10-02.exe

首先依旧是查看字符串，寻找writefile的相关操作。



接下来查看相应的反汇编代码：

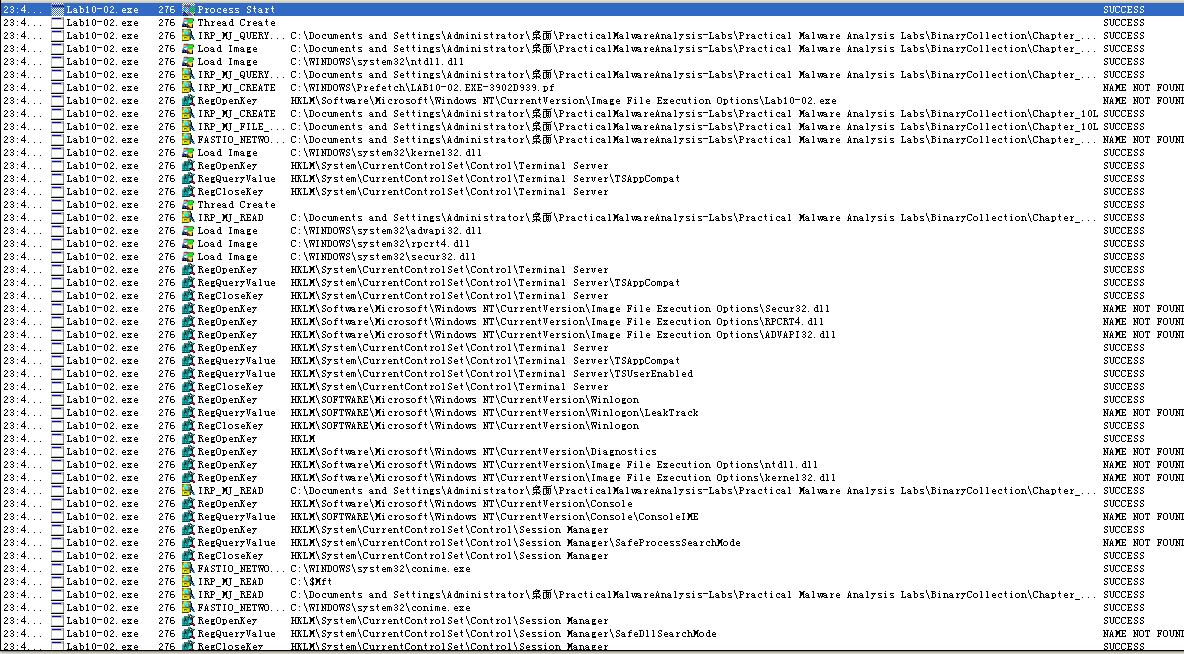


基本上可以确定是在利用资源文件创建服务，服务是一个sys驱动。

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

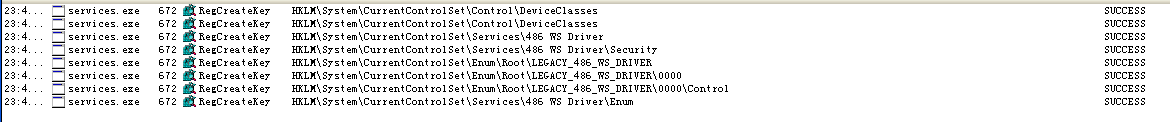


这两个函数是KERNEL32.DLL的导入函数，不注意看还是难发现的，然后我们知道了这个代码会操作自己的资源节，那我们就去检查一下这个程序的资源节

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。

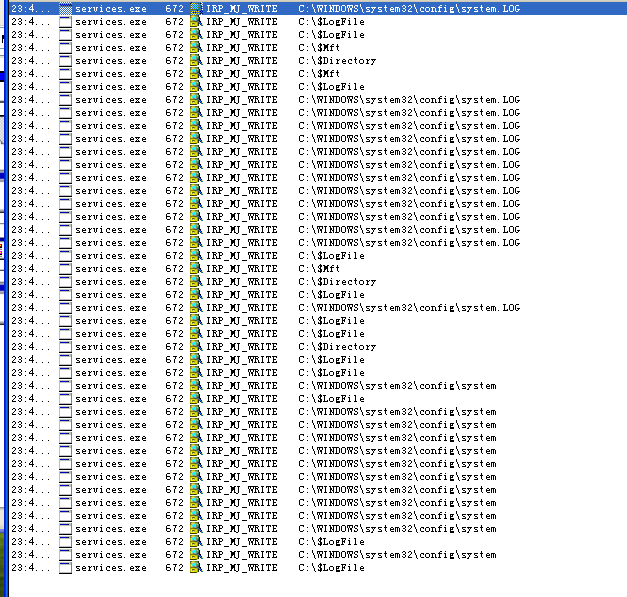
发现其中尤其需要注意的是出现了486\_WS\_Driver

在process monitor当中筛选可以查找到相应的位置里有这个进程。

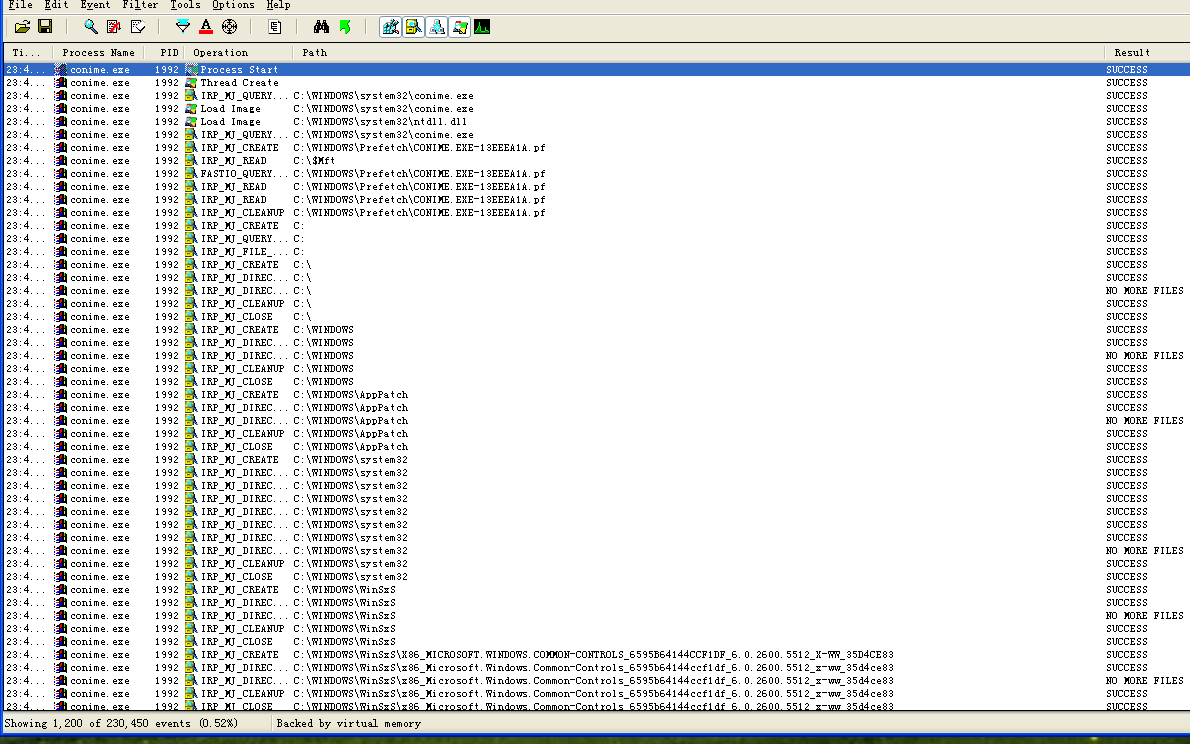


这里我们发现了一个叫services.exe的代码，执行了RegCreateKey，而且路径也和我们Regshot的结果相同.然后我们缩小搜索范围，搜索这个名叫services.exe的程序做了哪些其他事，记住此时这个程序的PID为672。

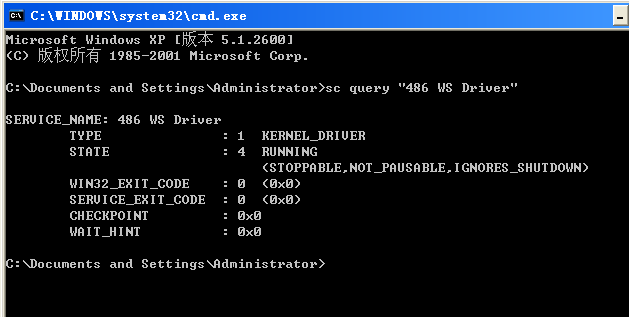
设置筛选条件为WriteFile之后，就会发现这个文件一共写了三个文件，一个是system.LOG，一个是system，还有一个是SysEvent.Evt，然后我们试试查找Lab10-02.exe这个进程名字，恶意行为分析本身就很繁琐。



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

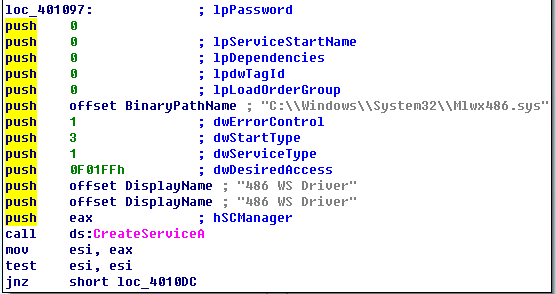


然后这个动态分析大概就只能分析出来这些东西了，书中说，如果你去找那个Mlwx486.sys，你是找不到的，但是我们可以找到这个 conime.exe文件 ==》这是本实验的核心，这个恶意软件的精髓就是隐藏这个Mlwx486.sys文件，如何做到的？就是利用内核的钩子，在系统使用NtQueryDirectoryFile遍历文件的时候，隐藏了Mlwx486.sys！

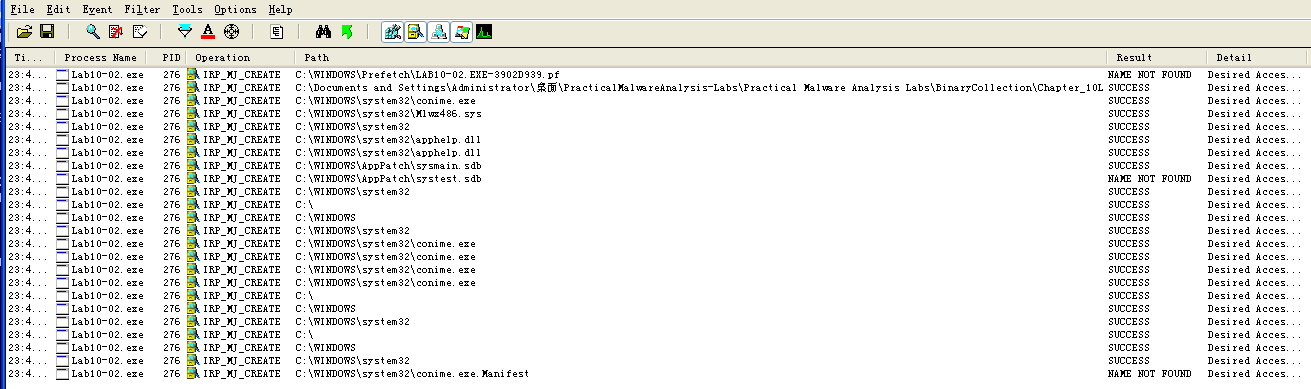


可以很明显的看出来这个是内核的驱动(KERNEL\_DRIVER)

作者是如何知道这个内核驱动的名字是486 WS Driver呢？



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)



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？

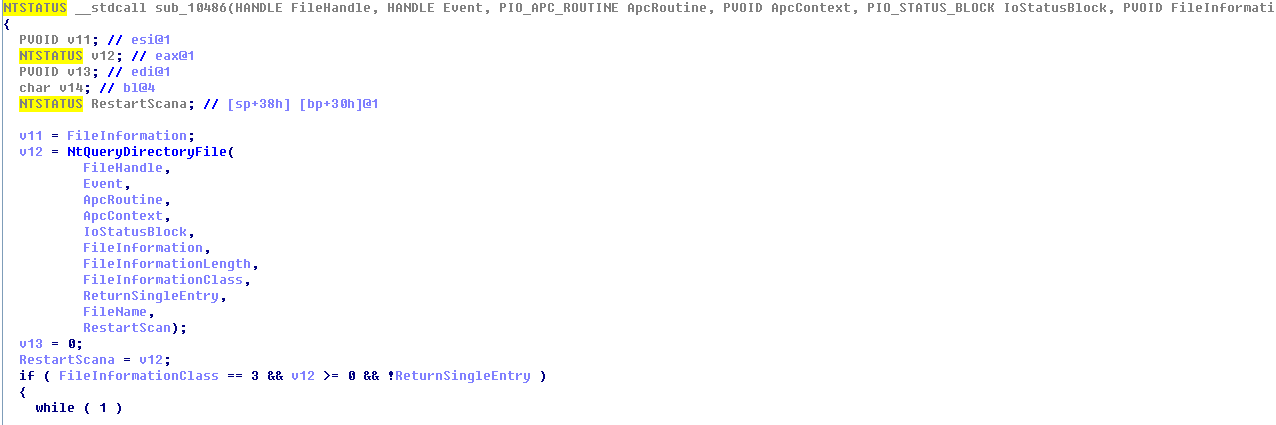
解答： 现在我们就要连接内核调试器来操作了

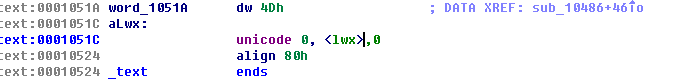
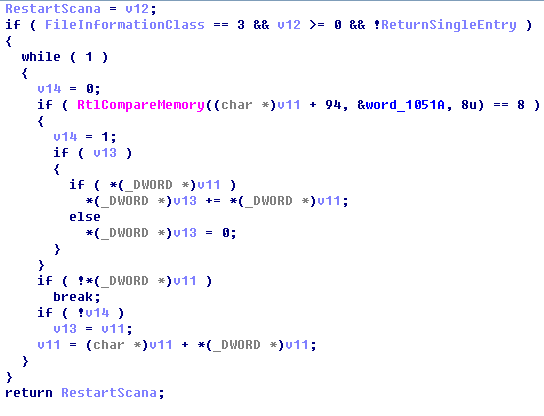
WinDbg里面运行命令

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：



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）

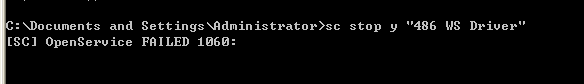
计算机通过计算之后，认为第三个结构体存在00000180这个地址上，就去00000180上取数据，从而跳过了第三个结构体，所以这个通过改变offset在不改变数据结构的前提之下，达到了隐藏文件的目的，也只会有天才才会想的出来了

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

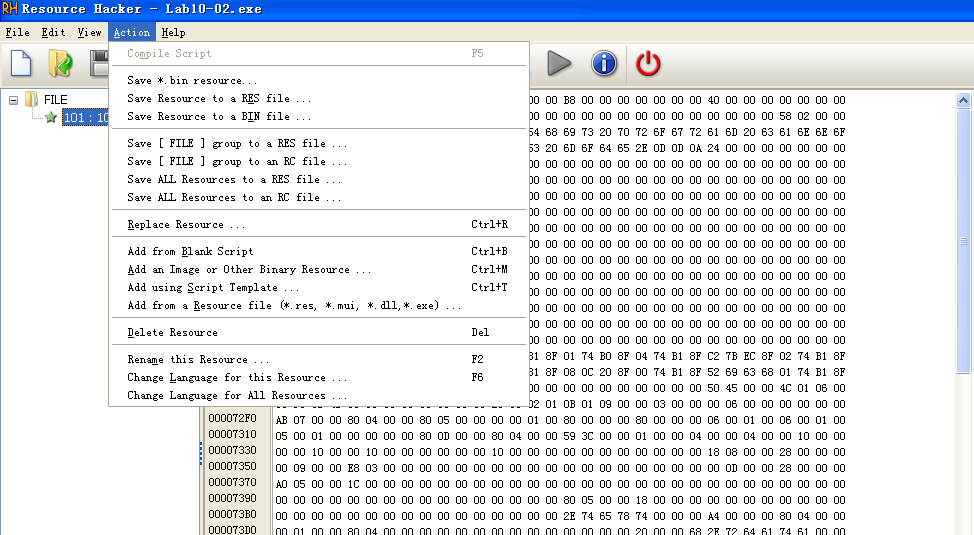
3.这个程序做了些什么？

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



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。

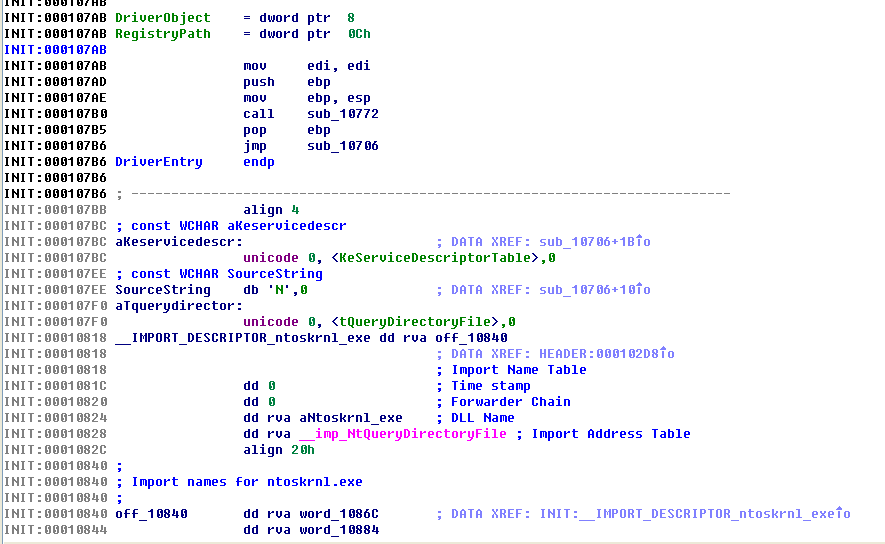


在ida当中打开相应的程序

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；



我们进入DriverEntry这个例程



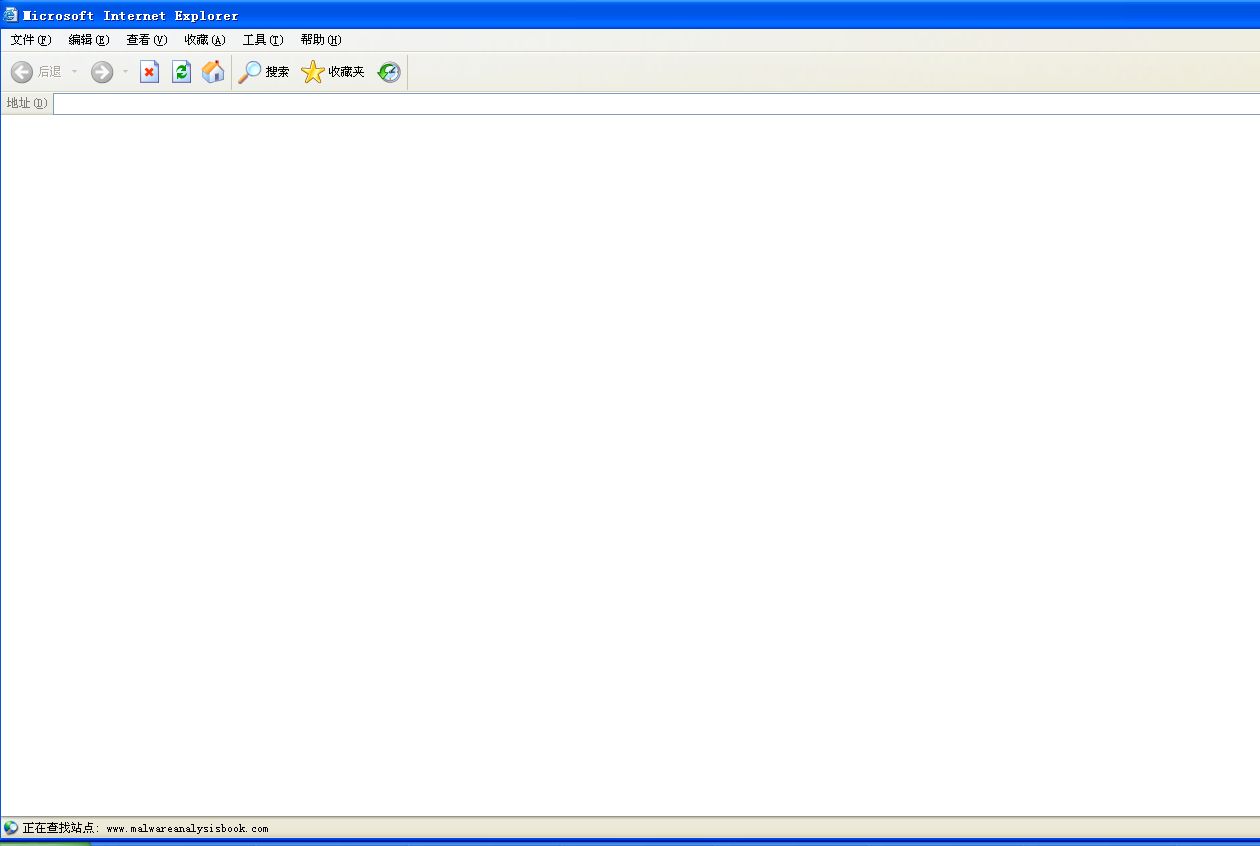
这里就不详细分析这个代码了，书上说是RtlInitUnicodeString以参数KeServiceDescirptorTable和NtQueryDircetoryFile做入参，然后用MmGetSystemRoutineAddress这个函数来查找这个两个地址的偏移量，接下来他把地址做了一个替换。

**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。

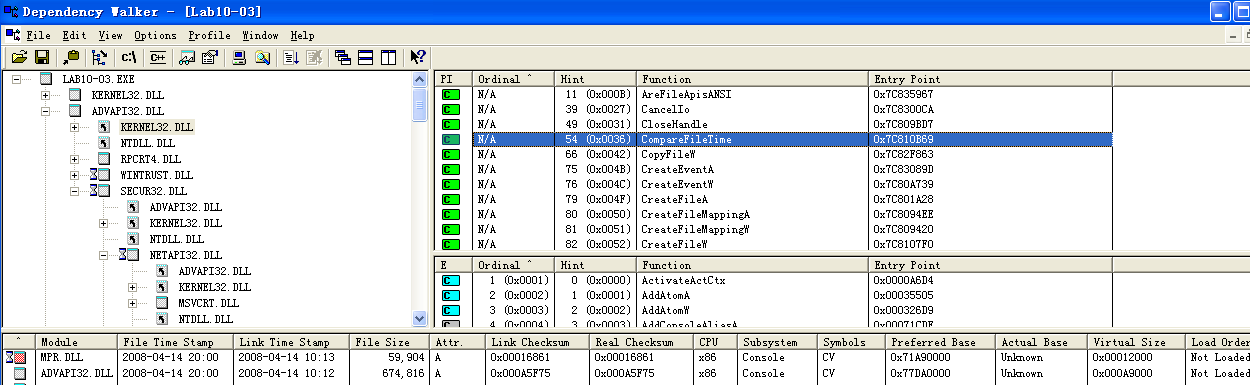
把文件放到那个目录之后，点击执行就会跳出这个IE，然后不断的跳IE出来



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

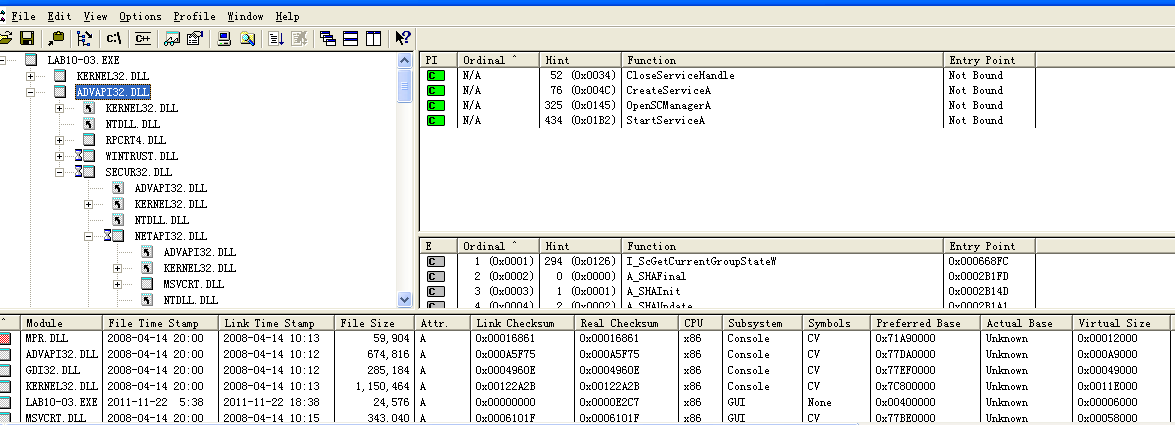


然后我们查看这个ADVAPI32.DLL这个导入库

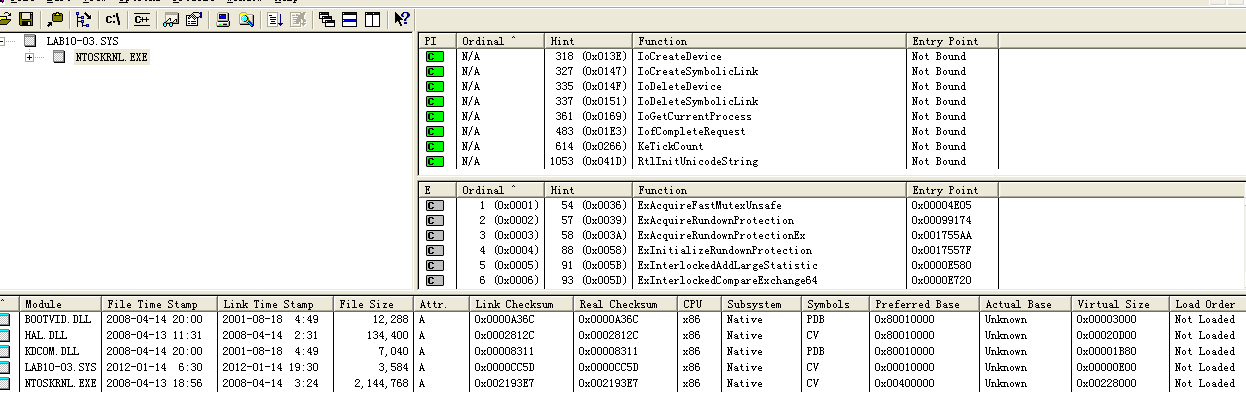
这个有一个比较让人感兴趣的导入函数就是OpenSCManagerA，还有StartServiceA以及CreateServiceA这三个导出函数，说明这个代码会创建一个服务在系统中

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

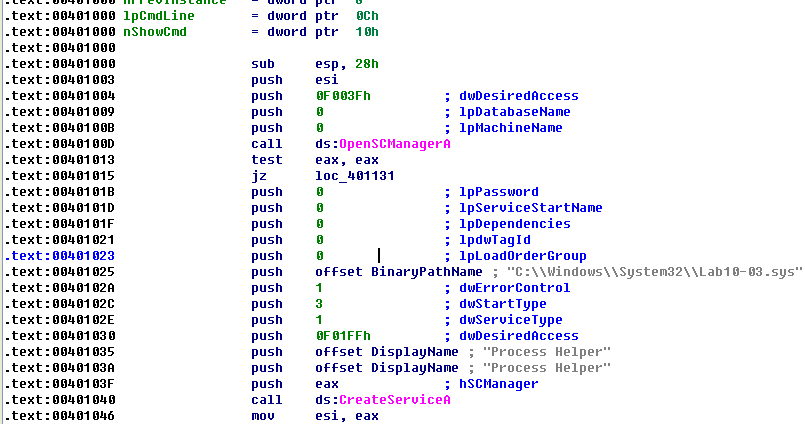


这个病毒最特别的地方在于，如果我们打算用任务管理器关闭这个进程的时候，会发现，根本没有这个进程，包括使用Process Explorer里面也是没有列出来



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



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。

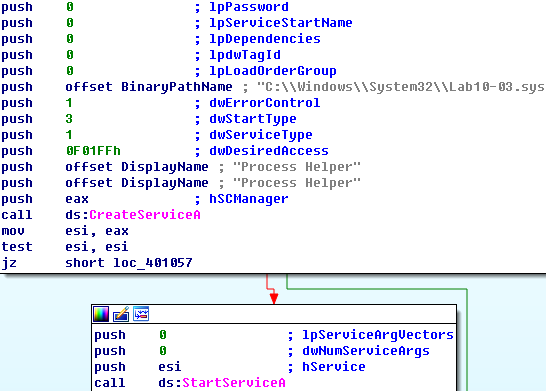
将登录令牌分配给进程。

允许该过程执行。

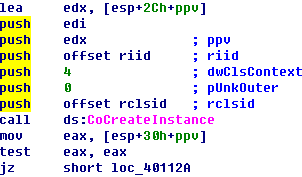
下一个参数是dwServiceType，他的值为1，意义就是表明这是个驱动服务

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



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



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

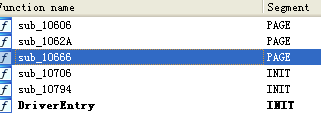
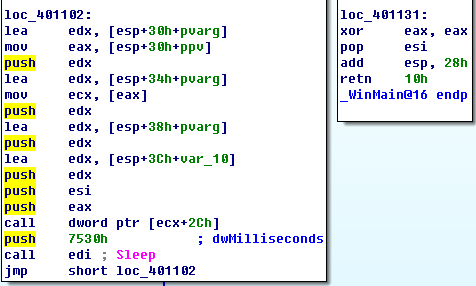
创建与指定的CLSID关联的类的单个未初始化对象。

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

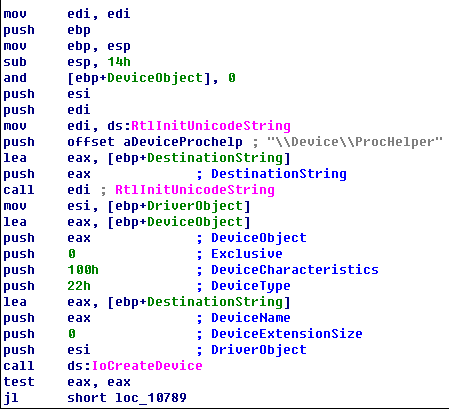


最后在这里调用了Sleep函数休眠了0x7530h毫秒，然后就是一直循环这个代码块，直到你关机为止



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



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

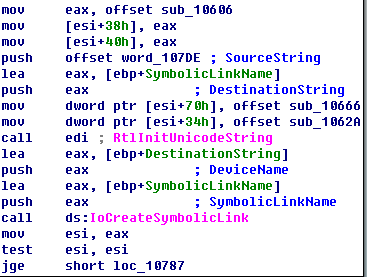
这个函数的标准定义如下，根据代码中的标识，我们可以得出以下结论

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



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,

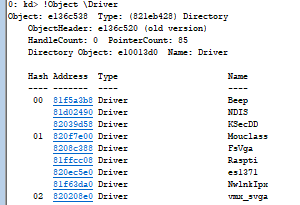
\_In\_ PUNICODE\_STRING DeviceName);

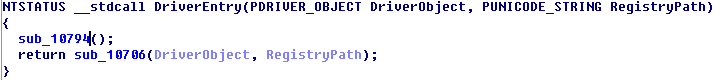
这个IoCreateSymbolicLink创建了一个符号链接供用户态的应用程序访问这个设备



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





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}

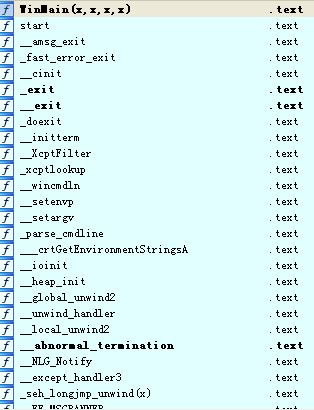


IDA python的编写：

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

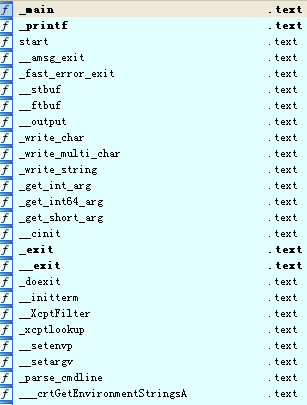
'''

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)

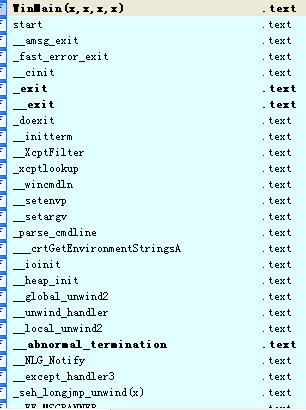
if \_\_name\_\_ == '\_\_main\_\_':

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)

1. **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。**