2024-1-14

北京理工大学计算机学院

计算机病毒防治

实验七 WinDbg实验报告

目录

[实验准备 2](#_Toc156217486)

[一、 实验环境和所用到的软件 2](#_Toc156217487)

[二、 基础知识 2](#_Toc156217488)

[Lab 10-1 4](#_Toc156217489)

[一、 使用procmon去检查程序是否会对注册表造成直接改变 4](#_Toc156217490)

[二、 用户空间程序调用了ControlService函数。使用WinDbg设置断点去看调用了ControlService函数后内核执行了什么 5](#_Toc156217491)

[三、 这个程序做什么 15](#_Toc156217492)

[Lab 10-2 15](#_Toc156217493)

[一、 这个程序创建了什么文件 15](#_Toc156217494)

[二、 这个程序是否有一个内核组件 15](#_Toc156217495)

[三、 这个程序做什么 17](#_Toc156217496)

[Lab 10-3 21](#_Toc156217497)

[一、 这个程序做什么 21](#_Toc156217498)

[二、 程序运行中如何停止它 24](#_Toc156217499)

[三、 内核组件做什么 24](#_Toc156217500)

[实验总结 27](#_Toc156217501)

# 实验准备

1. 实验环境和所用到的软件

* 虚拟机为Windows 10 x86，宿主机为Windows 11 x64
* WinDbg、Procmon、IDA Pro等

1. 基础知识
2. WinDbg命令：

* dx + 内存地址
  1. da：以ASCII码方式显示
  2. du：以Unicode方式显示
  3. dd：以32位双字方式显示
* ex + 内存地址 + 要写入的数据
  1. ea：以ASCII码方式写入
  2. eu：以Unicode方式写入
  3. ed：以32位双字方式写入
* 常用的运算符：+、-、\*、/
* dwo：显示地址指向的值
  1. du dwo (esp+4)：以宽字符方式显示函数的第一个参数
* bg：设置断点
  1. bg GetProcAddress "da dwo (esp+8);g"：在GetProcAddress处设置断点，打印出第二个参数，然后继续运行
* g：继续运行
* lm：列出加载到进程中的所有模块，包括用户态的exe、dll和内核态的驱动
  1. 获取内存映象
  2. lm m nt：列出匹配nt的模块，显示SSDT
  3. 在内核模式下都是驱动
* u 模块名!符号名：反汇编
* bu 模块名!导出函数：一旦模块被加载就在导出函数处加一个断点
* $iment()：找到模块的入口点
  1. bu $iment(驱动名)：在驱动代码运行之前在驱动的入口点加一个断点
* x nt!\*CreateProcess\*：使用通配符搜索函数或符号，展现所有的导出函数或内在函数
* ln 地址：列出最相近的符号

1. 用来指明函数调用进度
2. 第一行展现出相近匹配
3. 最后一行展现出精确匹配

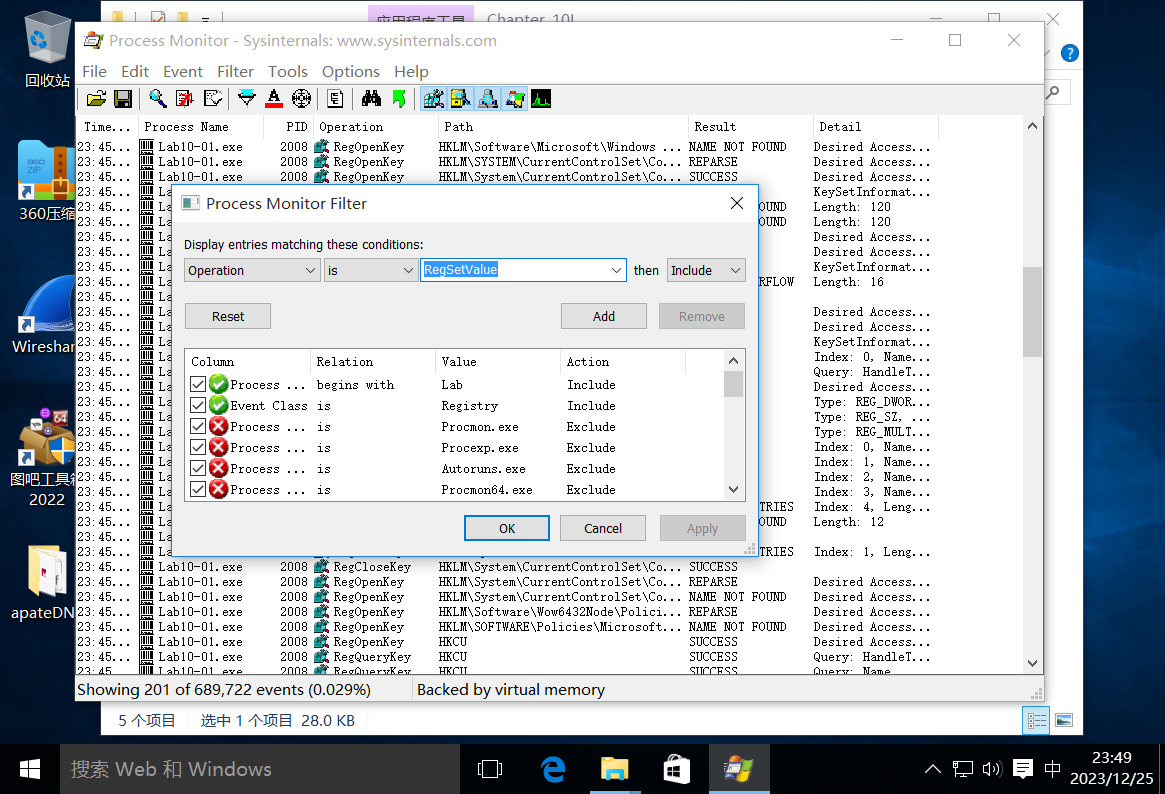
* dt 模块名!符号名：展示类型信息
* dt 模块名!符号名 地址：展示在地址处数据的结构体
* !idt：显示IDT
* !drvobj 驱动名：显示驱动对象
* !devogj：驱动对象
* !devhandles：列出使用驱动的应用程序

# Lab 10-1

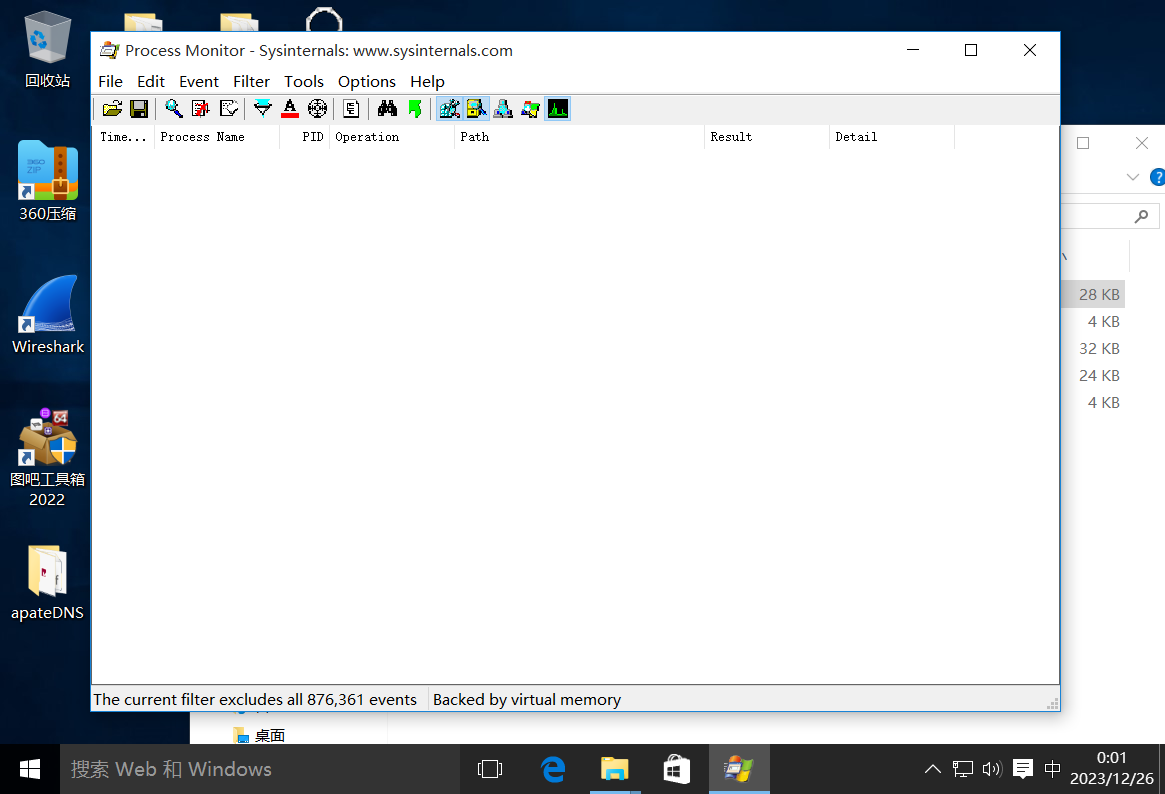
1. 使用procmon去检查程序是否会对注册表造成直接改变

首先将Lab10-01.sys复制到C:\Windows\System32目录下。

打开procmon并清除相关事件。之后打开Lab10-01.exe，然后过滤进程名Lab10-01.exe、操作RegSetValue。



这里我没有找到任何关于该程序直接修改注册表的信息。



所以使用procmon不能检测到注册表的修改。

1. 用户空间程序调用了ControlService函数。使用WinDbg设置断点去看调用了ControlService函数后内核执行了什么

使用IDA Pro打开Lab10-01.exe，可以看到在主函数里只有5个函数调用，为OpenSCManagerA、CreateServiceA、OpenServiceA、StartServiceA、ControlServiceA。

图示

中度可信度描述已自动生成

这个程序调用CreateServiceA创建了名为“Lab10-01”的服务，且会使用位于C:\Windows\System32\Lab10-01.sys的代码，且服务类型为3，表示这个文件将被加载到内核中。如果服务已经存在则会创建失败，此时直接调用OpenServiceA函数即可。之后调用StartServiceA去启动服务。

图形用户界面, 文本

中度可信度描述已自动生成

图形用户界面, 应用程序

描述已自动生成

最后，这个程序调用ControlService来停止服务，卸载驱动并调用驱动的卸载函数。

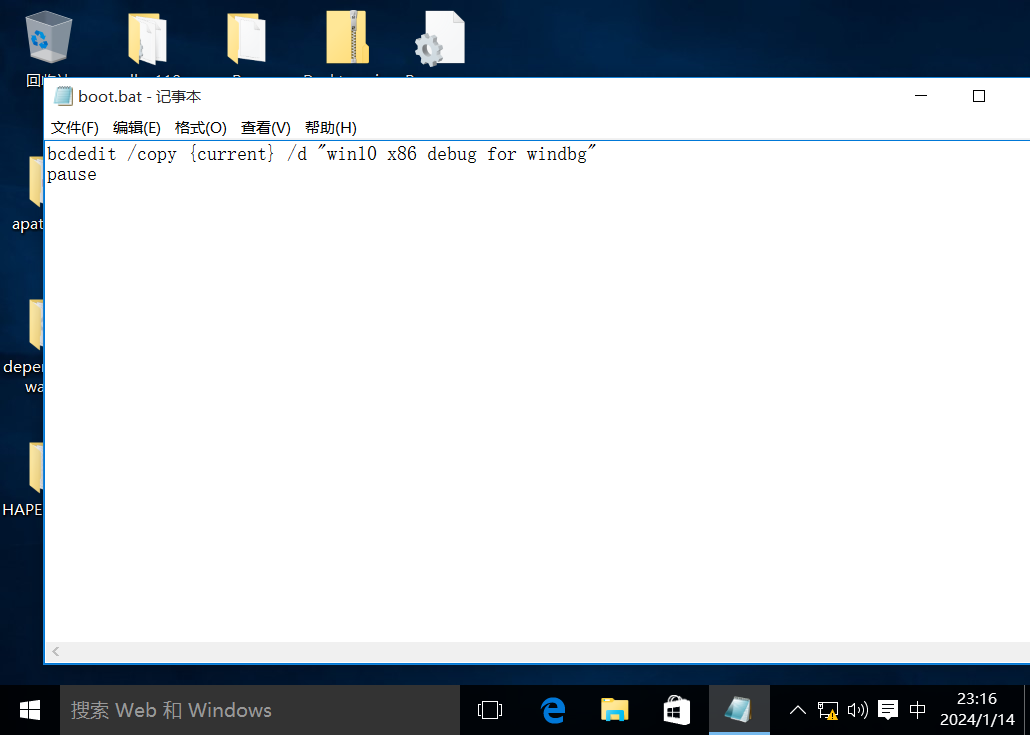
从上述分析可以看出，用户态的程序加载Lab10-10.sys然后卸载它。接着，我们使用WinDbg来检查Lab10-01.sys。

接下来配置内核调试环境。首先添加一个串行端口，用于虚拟机和主机进行通信。

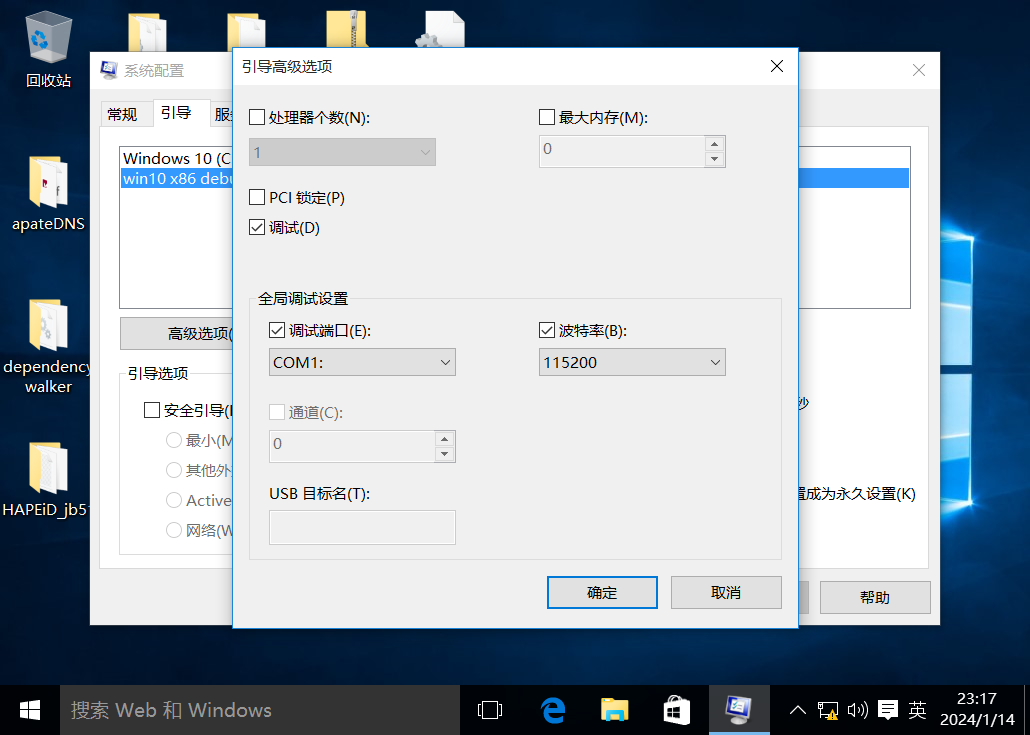
图形用户界面, 应用程序

描述已自动生成

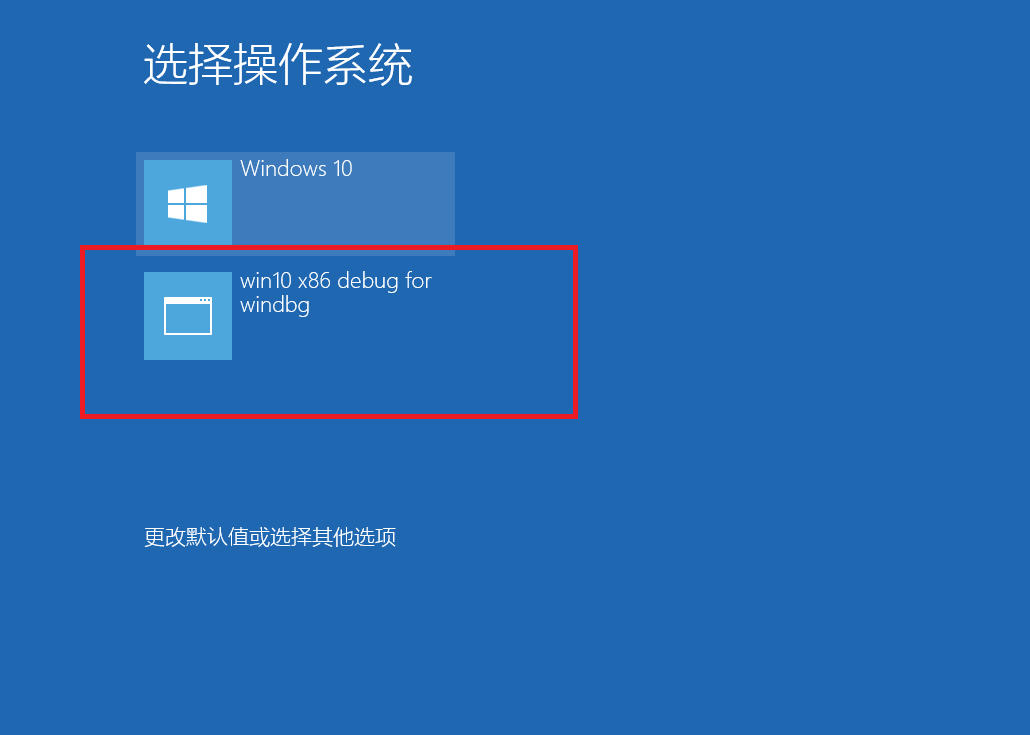
编辑虚拟机中的脚本，并以管理员身份运行，从而在启动项中添加内核调试选项。



使用运行打开msconfig，并进行如下配置。



然后重新启动虚拟机，选中win10 x86 debug for windbg引导项。

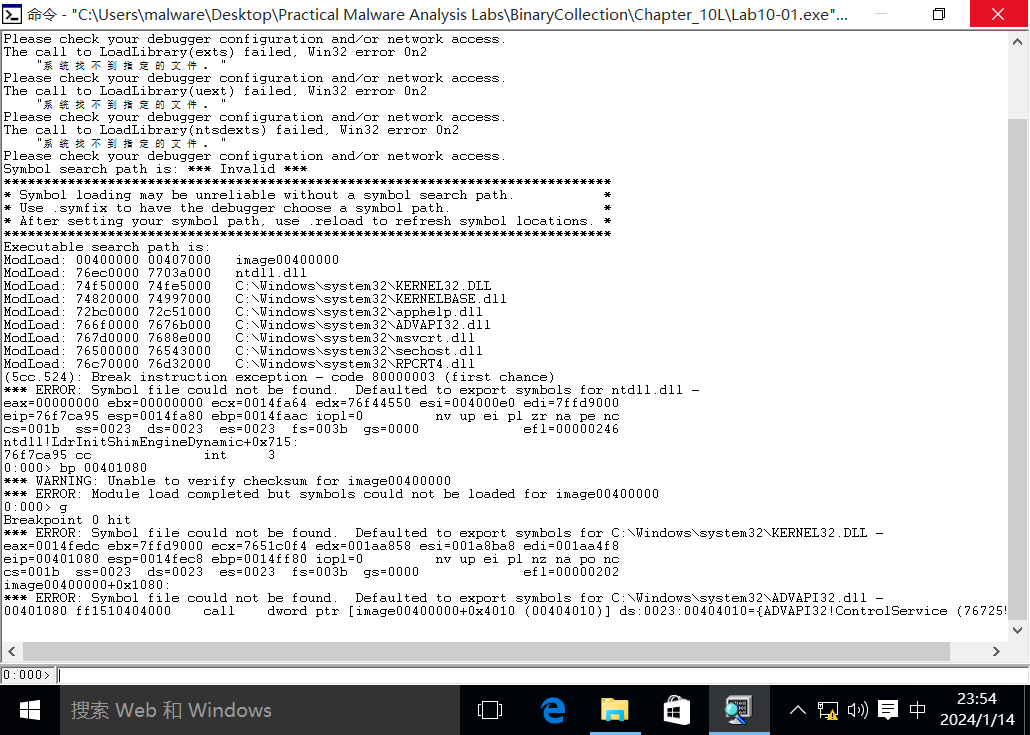


使用WinDbg打开Lab10-01.exe，这里注意一定要使用管理员方式打开WinDbg，然后输入命令“bp 00401080”在调用ControlService处添加断点。0x401080是调用ControlService的地址。

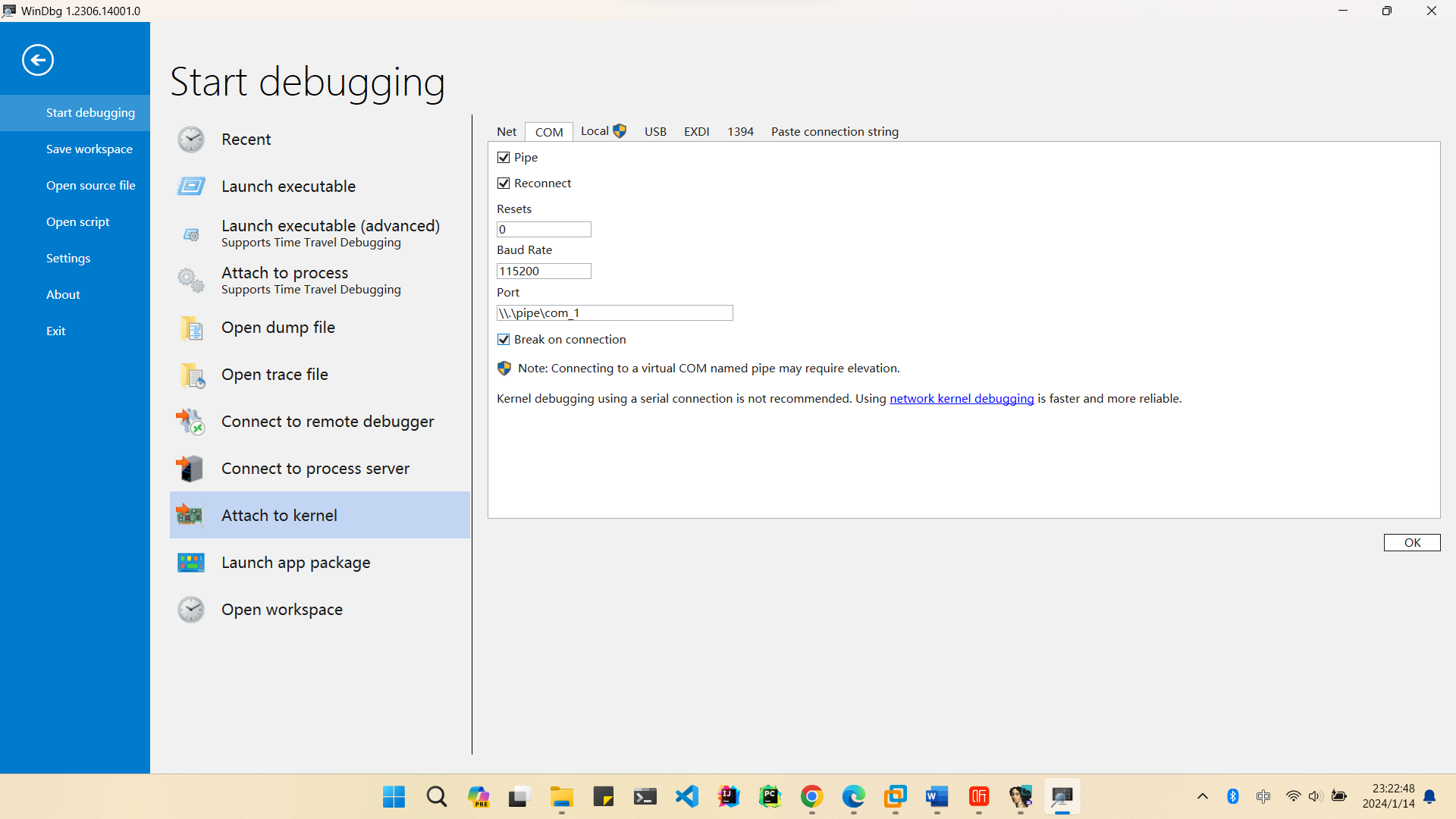
文本

描述已自动生成

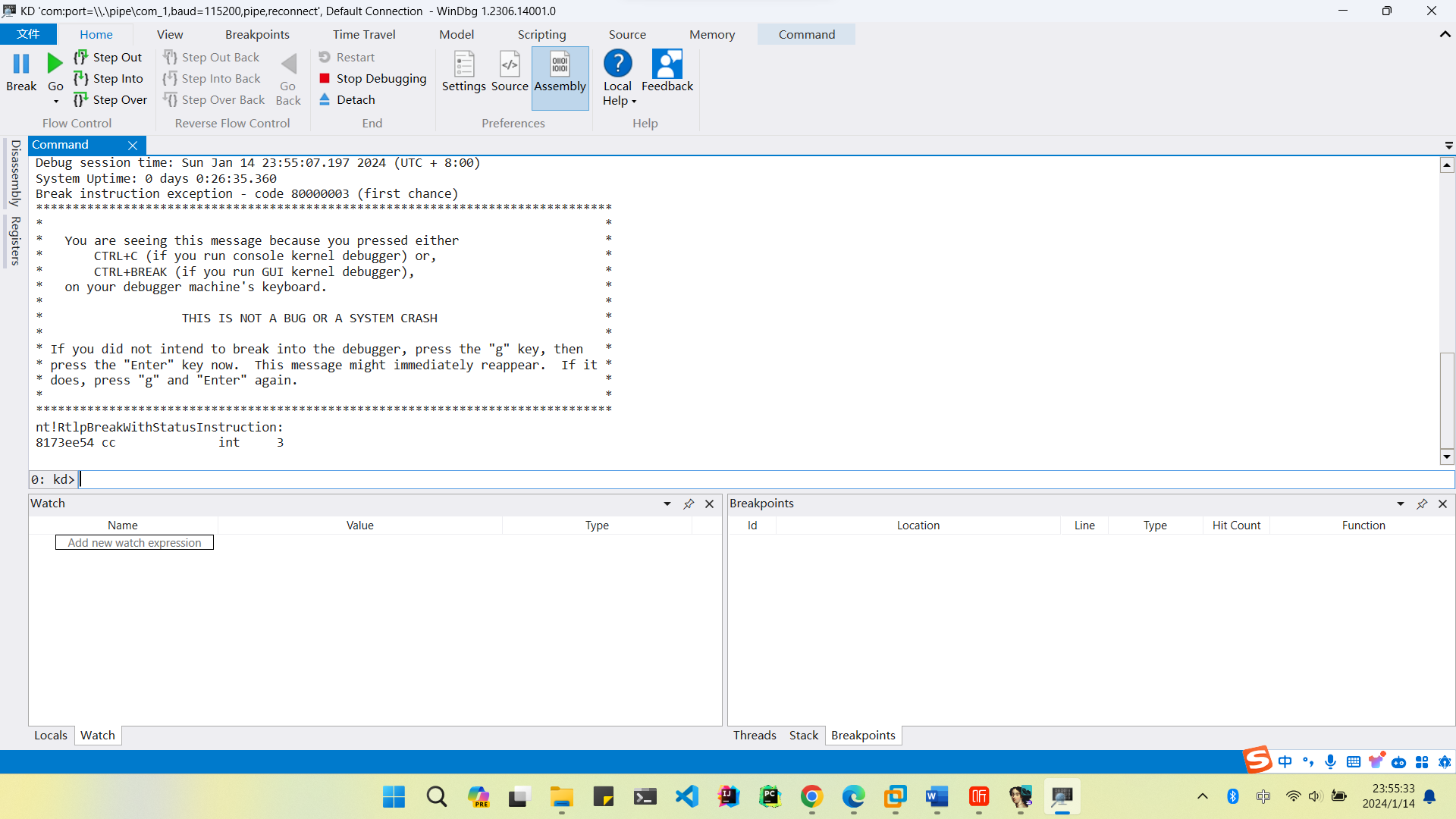
然后输入命令“g”让程序在断点处停下来。此时WinDbg的输出如下：



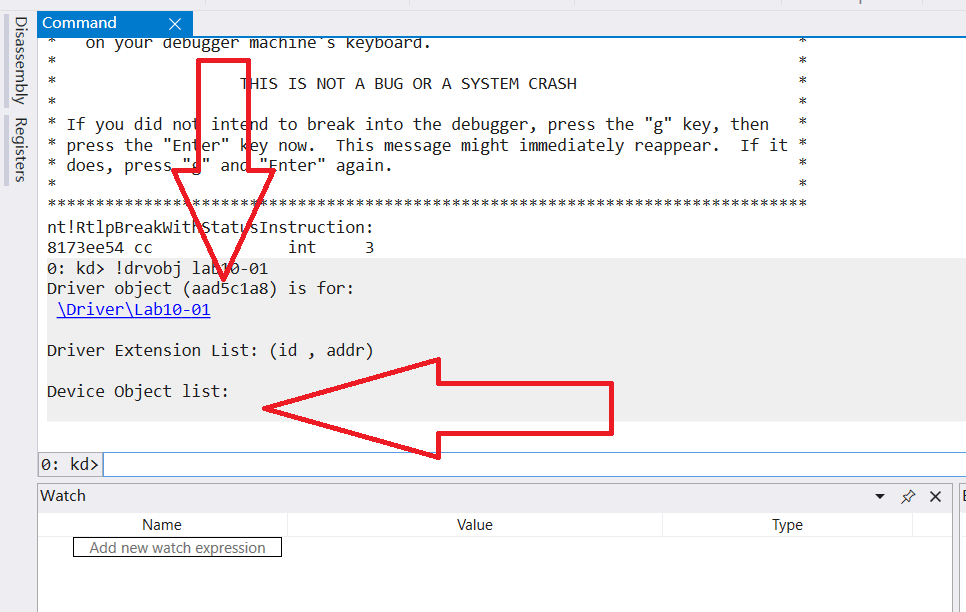
之后在宿主机中启动WinDbg，并连接虚拟机。配置如下：



连接成功后显示如下：



输入命令“!drvobj lab10-01”来显示服务lab10-01的驱动对象。从中我们可以看到驱动对象的地址为0xaad5c1a8。由于设备对象列表为空，所以这个驱动没有可以在用户空间应用程序访问的设备。



接着使用命令“dt \_DRIVER\_OBJECT aad5c1a8”来查看驱动对象的具体内容。

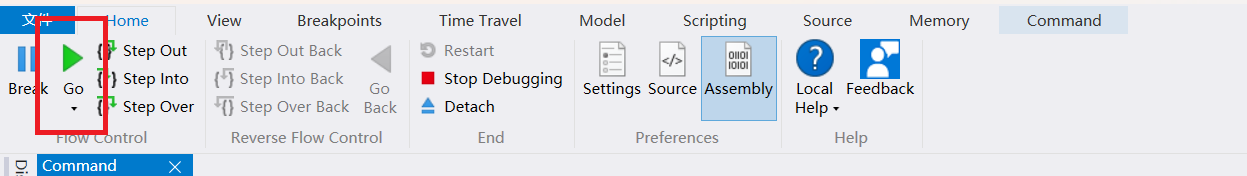
图形用户界面, 应用程序

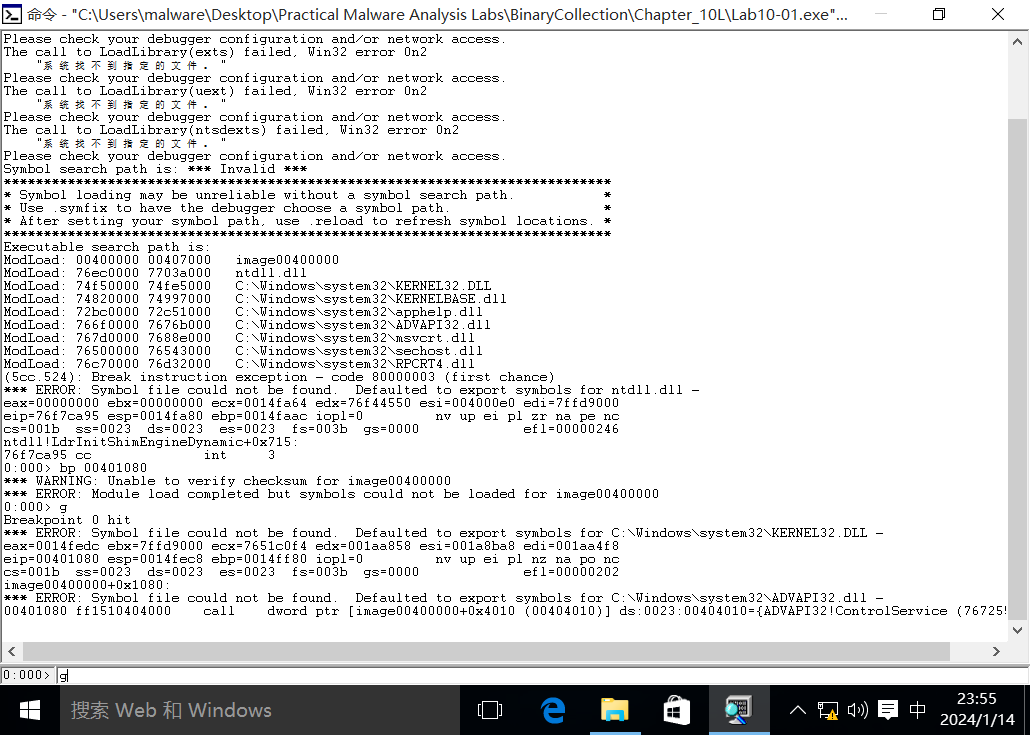
描述已自动生成

从中我们可以看到驱动卸载时调用的函数DriverUnload，其地址为0x9d5d0486。接着输入命令“bp 0x9d5d0486”来在卸载函数处设置一个断点。设置断点后，按下继续运行按钮。回到虚拟机后输入命令“g”来卸载驱动并调用卸载驱动函数。

图片包含 图形用户界面

描述已自动生成





这时我们返回宿主机，发现断点已经被命中了。

图形用户界面, 文本, 应用程序

描述已自动生成

我们已经知道了驱动卸载函数的地址，使用IDA Pro来分析函数更方便，但是在WinDbg中的地址有一个偏移。从之前驱动对象的具体内容中的DriverStart，我们可以得到驱动加载的地址为0x9d5d0000。使用IDA Pro打开Lab10-01.sys，并设置其加载地址。

图形用户界面, 文本, 应用程序

描述已自动生成

然后我们定位到0x9d5d0486，可以看到如下内容：

图形用户界面, 文本, 应用程序

描述已自动生成

从上面可以看出，这个函数调用了RtlCreateRegistryKey函数来创建注册表项：、\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFirewall\StandardProfile、\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFirewall\DomainProfile，然后调用RtlWriteRegistryValue函数来修改刚才创建的注册表项值为0。

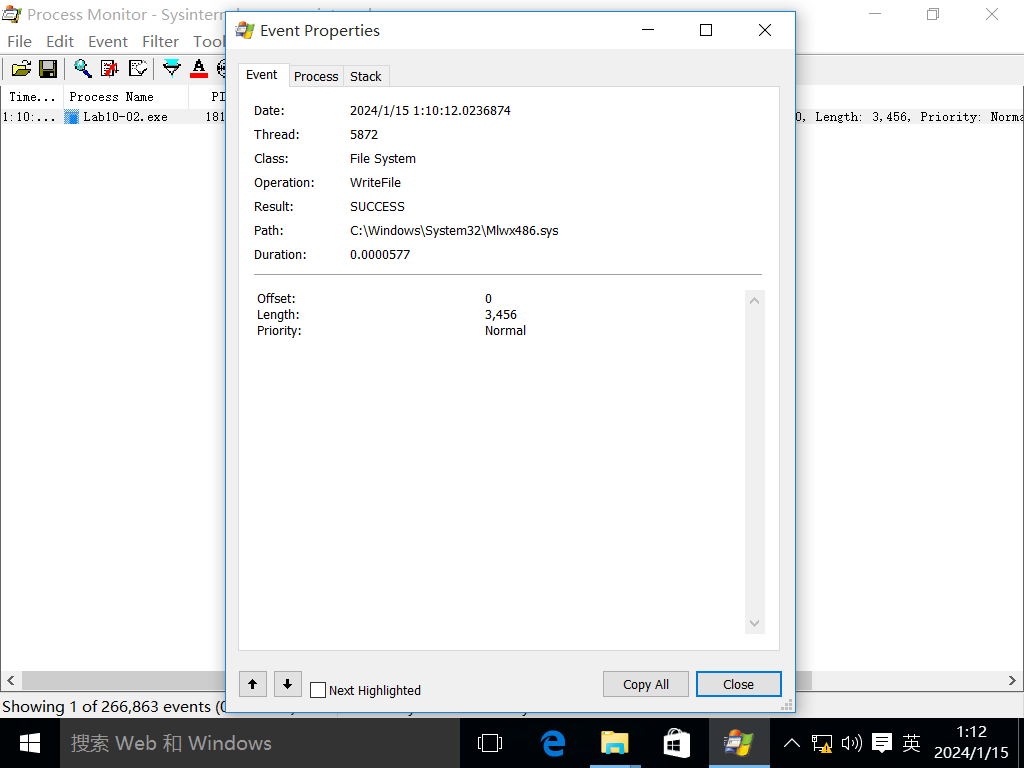
1. 这个程序做什么

综合以上分析，这个程序创建了一个服务，这个服务加载了一个驱动Lab10-01.sys，然后在卸载驱动时创建注册表项并修改，从而禁用防火墙。

# Lab 10-2

1. 这个程序创建了什么文件

使用Procmon来检测Lab10-02.exe运行后的事件，并设置过滤条件：操作为WriteFile。我们可以发现这个程序创建了Mlwx486.sys文件。



1. 这个程序是否有一个内核组件

接着使用IDA Pro来打开Lab10-02.exe，来分析该文件的功能。可以看到该恶意程序首先调用FindResourceA函数和LoadResource函数来加载资源节中的内容。之后调用CreateFileA函数和WriteFile函数来将资源节中的内容写入C:\\Windows\\System32\\Mlwx486.sys文件中。

图形用户界面, 文本

描述已自动生成

之后，该恶意软件调用CreateServiceA函数来创建名为“486 WS Driver”的服务，该服务加载了C:\\Windows\\System32\\Mlwx486.sys。最后通过调用StartServiceA函数来启动该服务。

图形用户界面, 应用程序

描述已自动生成

我们使用命令来内核驱动是否被加载，可以看到内核驱动已经被加载了。

文本

描述已自动生成

综上所述，这个程序有一个内核组件，它存储在文件的资源节中，然后写入磁盘中，最后将其加载到内核中作为一个服务。

1. 这个程序做什么

接下来使用宿主机的WinDbg来连接虚拟机。输入命令“lm”来列出加载到进程中的所有模块。输入命令“dd dwo(KeServiceDescriptorTable) L100”来查看SSDT，可以看到有一个地址0x9d3c0486和其他地址有较大的差异。

表格

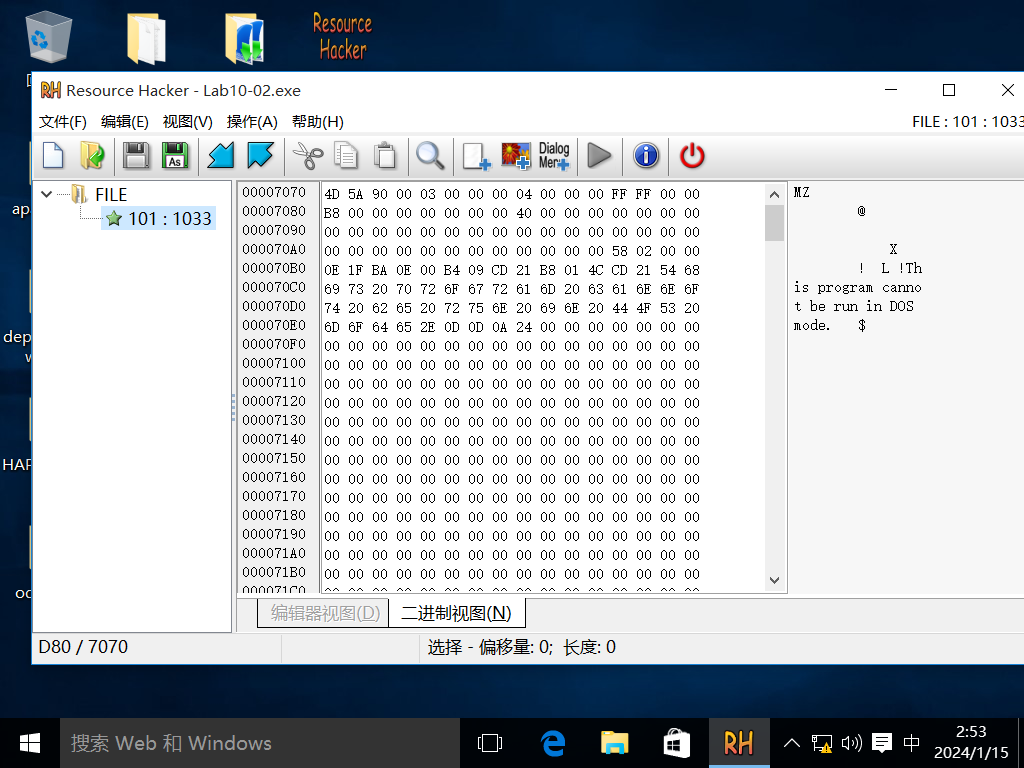
描述已自动生成

被替换之前的函数为NtQueryDirectoryFile，它用来返回有关给定文件句柄指定的目录中文件的各种信息。

表格

中度可信度描述已自动生成

使用Resource Hacker来提取资源节中的内容，然后使用IDA Pro打开进行分析。



文本

中度可信度描述已自动生成

在IDA Pro中搜索NtQueryDirectoryFile的交叉引用，可以看到sub\_10486函数中调用了该函数，而且该函数的地址与SSDT中的地址也相符合。

文本

描述已自动生成

在sub\_10486中，我们可以看到NtQueryDirectoryFile函数被调用，这使得hook替换之前的功能仍然可以正常使用。接着，函数调用了RtlCompareMemory函数，该函数比较两个内存块的前8字节，一个内存块存放“Mlwx”（这里是Unicode编码，每个字符占据两个字节），另一个内存块指向文件名。

图形用户界面, 应用程序

描述已自动生成

这里借用课本上的图来描述隐藏文件的原理。

图表, 表格

中度可信度描述已自动生成

综上，该程序是一个rootkit。它修改SSDT的指向NtQueryDirectoryFile的hook，从而实现隐藏以“Mlwx”开头的文件的功能。

# Lab 10-3

1. 这个程序做什么

首先将Lab10-03.sys复制到C:\Windows\System32文件夹下。

然后打开procmon来检测程序运行时的事件。

在尝试运行该Lab10-03.exe时，我发现每次运行都会蓝屏，所以只能转为静态分析。



使用IDA Pro打开Lab10-03.exe，首先发现该恶意程序创建了名为“Process Helper”的服务，服务类型为驱动程序服务，二进制文件路径为“C:\\Windows\\System32\\Lab10-03.sys”。创建服务成功后启动服务。

图形用户界面, 应用程序

描述已自动生成

接着，恶意程序通过调用CreateFileA来创建一个设备文件“[\\.\ProcHelper](file:///\\.\ProcHelper)”，之后调用DeviceIoControl来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。它将控制码0xABCDEF01发送给驱动设备，但是命令缓存区和返回缓存区为空。

图片包含 日程表

描述已自动生成

最后恶意程序通过COM库来操作浏览器，该部分内容和Lab07-02.exe相似。

图形用户界面, 应用程序

描述已自动生成

综上，该程序创建服务，加载驱动，并每隔30s打开浏览器弹出广告。

1. 程序运行中如何停止它

由于该恶意程序不在运行进程列表中，所以无法通过结束进程来停止它。我们尝试重新启动操作系统，发现弹窗消失。

1. 内核组件做什么

使用IDA Pro打开Lab10-03.sys。首先驱动程序调用IoCreateDevice来创建一个名为“\Device\ProcHelper”的驱动设备。

文本

描述已自动生成

然后驱动程序调用IoCreateSymbolicLink在设备对象名称和设备的用户可见名称之间设置符号链接，它将符号链接命令为“\DosDevices\ProcHelper”供用户空间程序使用。

图形用户界面, 文本, 应用程序

描述已自动生成

从课本得知DriverInit函数的偏移为0x7cd，DriverUnload函数的偏移为0x62a，主函数MajorFunction的偏移为0x606。使用IDA Pro定位到相应位置，我们可以得到DriverUnload函数即为sub\_1062A函数。该函数的主要功能是删除符号链接和驱动设备。

**文本

低可信度描述已自动生成**

接下来查看处理DeviceIoControl的函数sub\_10666。首先函数调用IoGetCurrentProcess函数来返回指向当前进程EPROCESS结构体的指针。

图形用户界面, 文本, 应用程序

描述已自动生成

EPROCESS 中0x88偏移处的字段为\_LIST\_ENTRY，LIST\_ENTRY是一个双向链表，将所有的活动进程都连接在一起。0x88处指向下一个进程的EPROCESS，0x8c处指向上一个进程的EPROCESS。

这里用课本上的图来说明以上代码的作用。

图示

描述已自动生成

所以，该内核组件响应DeviceIoControl请求，该请求中修改了进程链表，从而隐藏当前进程。

# 实验总结

本次实验让我初步掌握了WinDbg的使用方法，同时我意识到WinDbg是一款强大的工具，需要我课后抽时间学习一下它的用法。而且新版的WinDbg界面更加美观，美中不足就是只支持x64系统，只能在宿主机上使用。同时IDA Pro仍然在这章发挥了巨大的作用，我对IDA Pro的喜爱程度更加深厚了。最后，由于实验环境的原因，部分程序无法正常运行，导致我只能转向静态分析了。