**恶意代码分析与防治技术实验报告**

**Lab10**

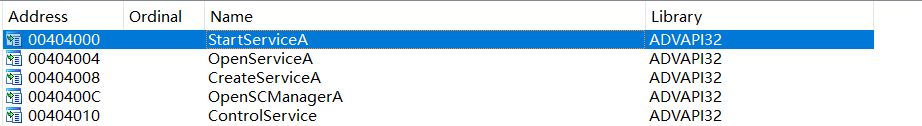
**学号：2013018 姓名：许健 专业：信息安全**

1. **实验内容**

完成课本Lab10的实验内容，编写Yara规则，并尝试IDA Python的自动化分析。

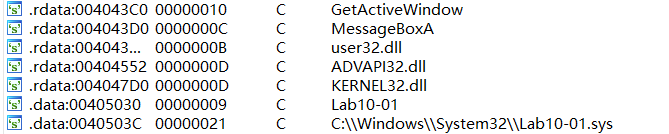
配置实验环境时遇到windbg无法连接虚拟机串行端口以及双机调试环境虚拟机系统卡死的问题，去网上寻求解决方法，使用VMware + win10 + VirtualKD + windbg从零搭建双机内核调试环境。耽误了一些时间，特将报告上传至github repo，还请助教理解。

1. **问题解答**
2. Lab10-1
3. 这个程序是否直接修改了注册表（使用procmon来检查）？

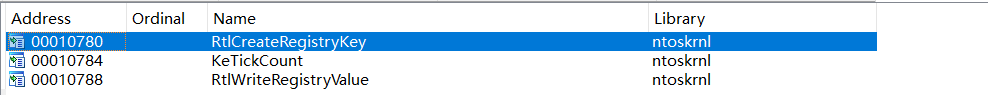


基本静态分析检查可执行文件，从导入函数可以看出该程序创建了一个服务，并且可能启动或者操作这个服务。

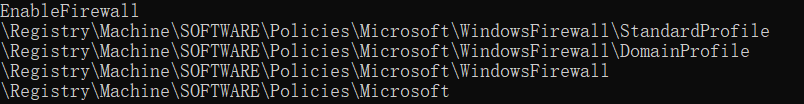
查看字符串C:\\Windows\\System32\\Lab10-01.sys，可能包含这个服务的代码。

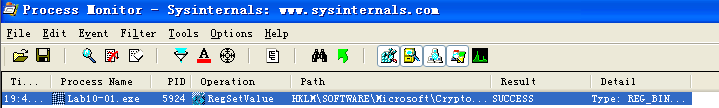


查看驱动程序的导入函数，包含RtlCreateRegistryKey和RtlWriteRegistryValue，说明驱动可能访问了注册表。



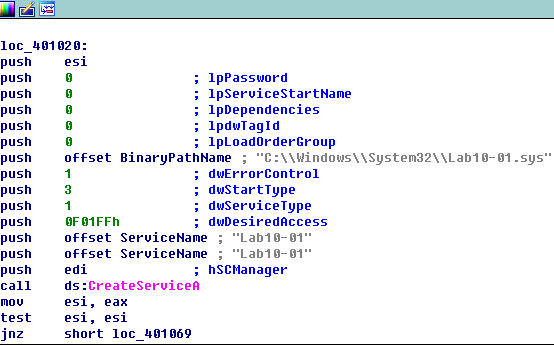
可以看到一些类似注册表键值对，当从内核态访问注册表时，前缀\Registry\Machine等同于用户程序访问的HKEY\_LOCAL\_MACHINE。将EnableFireWall置为0表示禁用Windows XP自带的防火墙。



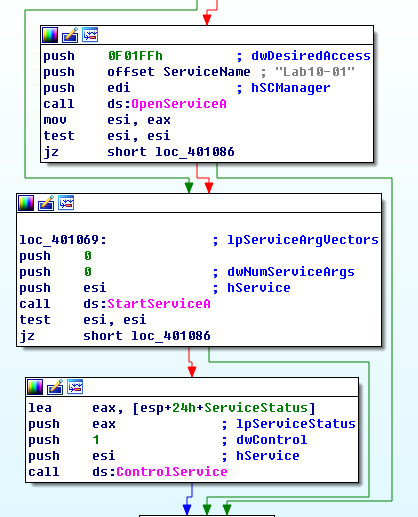
使用procmon可以看到Lab10-01.exe写注册表的调用

RegSetValue写了HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed键值，该键值对一直变化，对于恶意代码分析毫无意义，但是我们无法确保驱动没有秘密地修改注册表。

查看Lab10-01.exe的主函数，首先它调用OpenSCManagerA获取服务管理器的句柄，然后在调用CreateServiceA函数，创建一个名为Lab10-01的服务。服务使用了C:\\Windows\\System32\\Lab10-01.sys中的代码，服务类型为3（Service\_Kernel\_Driver），这意味着这个文件将被加载到内核。（如果创建失败则打开已有服务）

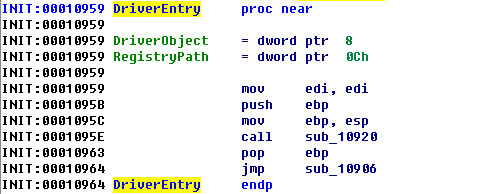


接下来程序调用StartServiceA来启动服务，最后调用ControlService。ControlService的第二个参数是发送控制消息的类型。0x01表示SERVICE\_CONTROL\_STOP，这将会卸载驱动，并调用驱动的卸载函数。

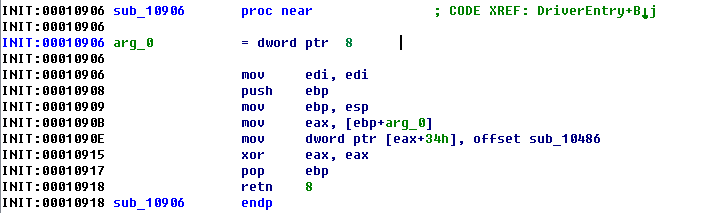


使用IDAPro查看Lab10-01.sys

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。



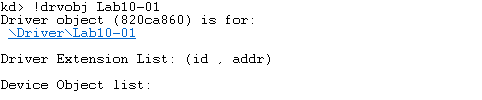
DriverEntry函数的主体部分似乎将一个偏移量移入到一个内存位置



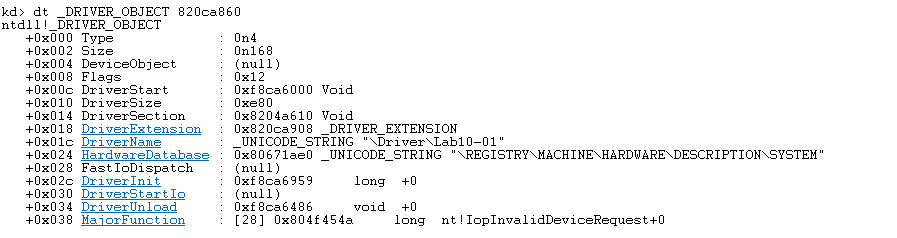
1. 用户态的程序调用了ControlService函数，你是否能够使用Windbg设置一个断点，以此来观察由于ControlService的调用导致内核执行了怎样的操作？

使用Windbg Preview分析Lab10-01.sys，查看当调用ControlService卸载Lab10-01.sys时发生什么。在虚拟机中使用ollydbg运行Lab10-01.exe，在ControlService处打一个断点，在宿主机上使用windbg preview连接虚拟机的内核。

使用!drvobj获取驱动对象，定位Lab10-01的设备对象，命令!drvobj输出给我们提供了驱动对象的地址，因为在输出对象列表中没有设备列出，所以我们知道这个驱动没有供用户空间中应用程序访问的设备。

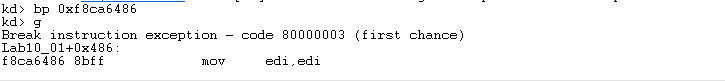


获得驱动对象的地址，可以使用dt命令查看。

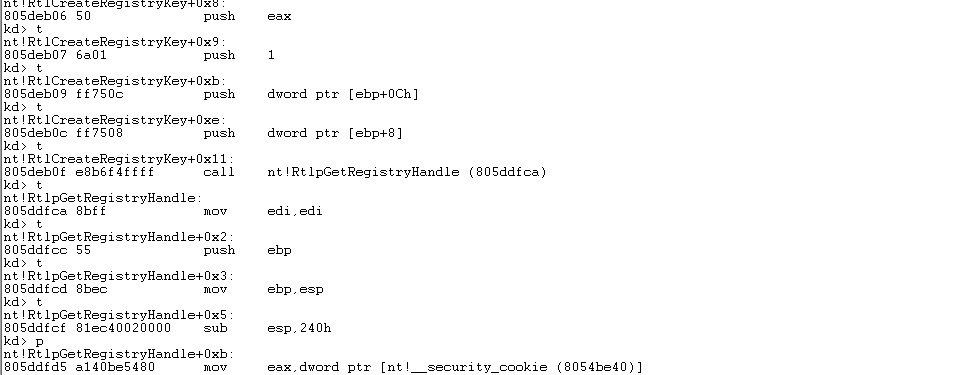


我们尝试确定驱动卸载时调用的函数——偏移量为0x034的信息DriverUpload，并设置一个断点。然后恢复内核运行，回到虚拟机中运行可执行程序的ollydbg，恢复它的运行。由于内核调试器命中断点，虚拟机会卡死。此时回到内核调试器单步调试代码。

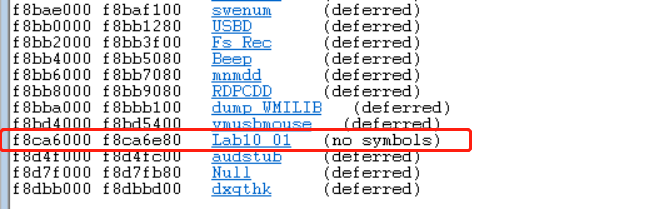
kd> bp 0xf8ca6486



使用windbg单步调试可以看到RtlCreateRegistryKey函数的调用，创建了一些注册表键，但是看的不够明显，可以使用IDA pro分析。



使用lm命令计算函数从Windbg加载文件开始处的偏移量，计算得出偏移量为0x486



使用IDA pro可以看到程序多次调用RtlCreateRegistryKey和RtlWriteRegistryValue，设置EnableFireWall值为0，从内核禁用XP 防火墙很难被安全程序探测。

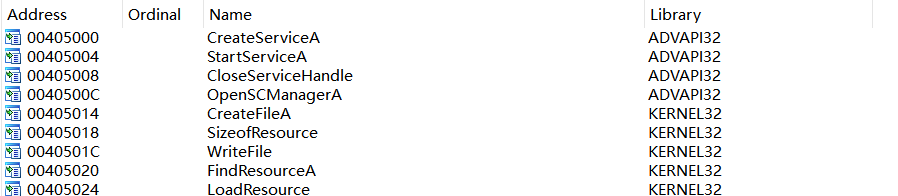


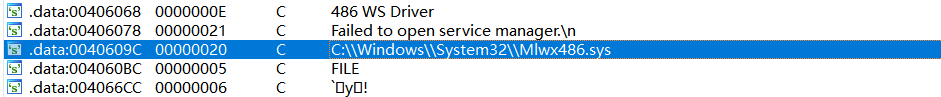
1. 这个程序做了些什么？

创建一个服务加载驱动，设置注册表键值对，关闭Windows XP防火墙。

1. Lab10-2
2. 这个程序创建文件了吗？它创建了什么文件？

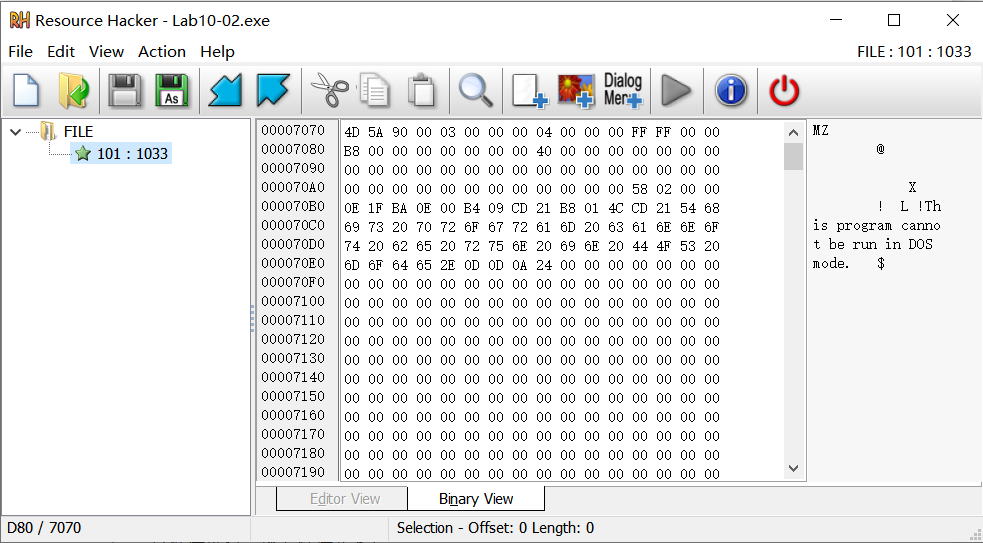
创建了文件C:\Windows\System32\Mlwx486.sys，使用procmon还可以看到其他文件。



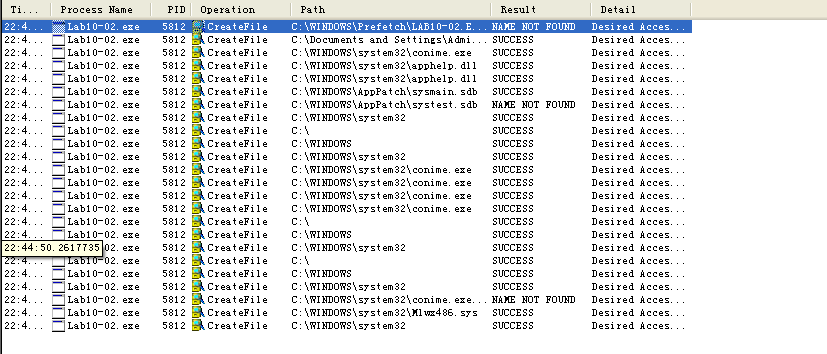


从导入函数推测，该程序会创建并启动一个服务。程序可能会写文件，LoadResource和SizeofResource告诉我们，程序可能会对资源节做处理。

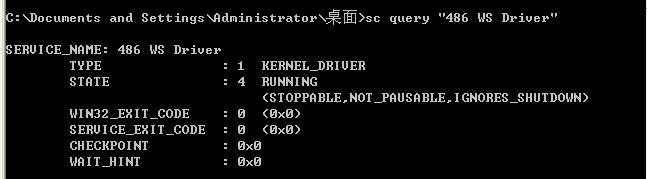
使用Resource Hacker检查资源节，可以看到资源节包含另一个PE头部，可能是将要使用的恶意文件。



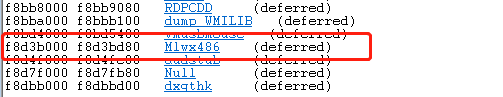
使用Procmon监测并运行程序，程序在C:\Windows\System32目录下创建了文件，并且以这个文件作为一个可执行程序创建了服务。但是我们查看C:\Windows\System32却没有发现任何东西，猜测恶意程序可能是Rootkit



sc命令查看服务状态，服务名称为我们使用IDA Pro看到的字符串。可以看到这个内核驱动仍在运行，在内存中。



使用windbg lm命令查看加载模块，可以确定文件名为Mlwx486.sys的驱动被载入内存中，但是硬盘中没有，说明是一个Rootkit。



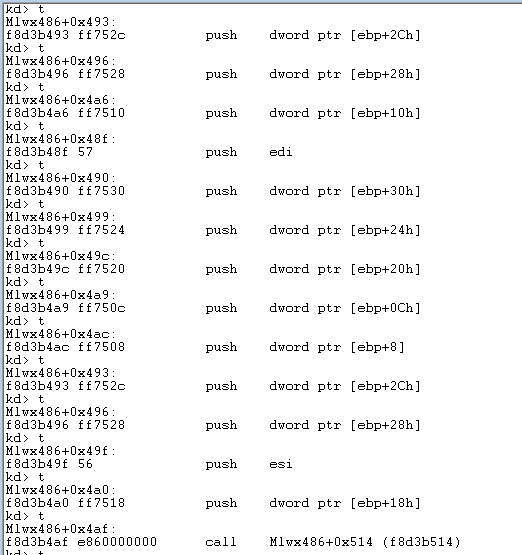
检查SSDT的所有修改项，对比Rootkit安装前后，发现NtQueryDirectoryFile函数被覆盖，该函数用来提取文件和目录信息，Windows资源管理器利用它来显示文件和目录，如果Rootkit挂钩了这个函数，它可以隐藏文件。



、

代替NtQueryDirectoryFile调用的挂钩函数必须与原始函数有相同的接口，查看原始函数文档，NtQueryDirectoryFile函数可以有很多参数，决定返回不同的值。

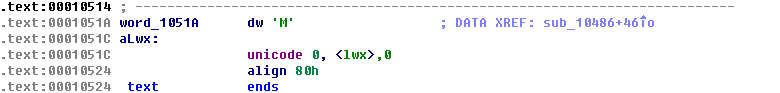
给恶意函数设置断点，查看汇编代码，发现它会先调用原始NtQueryDirectoryFile函数



为了搞清楚恶意代码究竟做了什么，我们需要对资源节中的PE文件反汇编

使用IDA Pro静态分析，发现函数会检查参数，if条件部分应该是隐藏文件，匹配文件是否为Mlws前缀，如果是则在链表中跳过节点（第一个节点指向第三个节点，跳过中间的节点）





或者使用windbg设置条件断点，当且仅当ReturnSingleEntry为0时，断点才会触发。

bp f8d3b486 ".if dwo(esp+24)==0{} .else{gc}"

1. 这个程序有内核组件吗？

这个程序有一个内核模块，该模块被存储在这个文件的资源节，作为一个服务加载到内核。

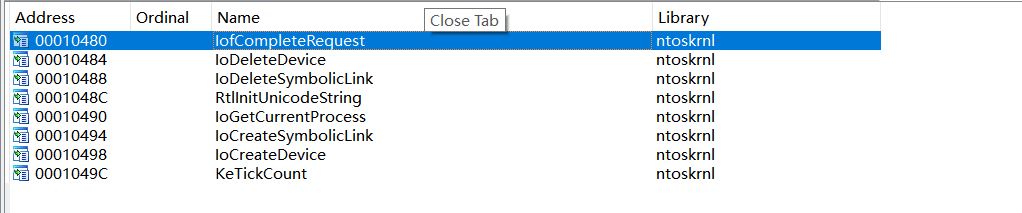
1. 这个程序做了些什么？

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

1. Lab10-3
2. 这个程序做了些什么？

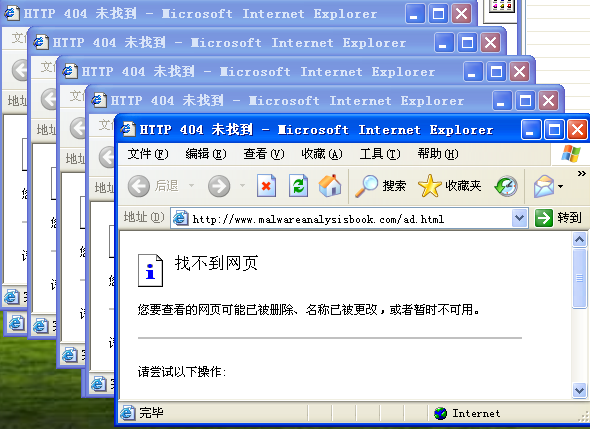
用户态程序加载驱动，每隔一段时间弹出广告。这个驱动通过从系统链表中摘除进程环境块(PEB)来隐藏进程。

对驱动程序进行基础静态分析

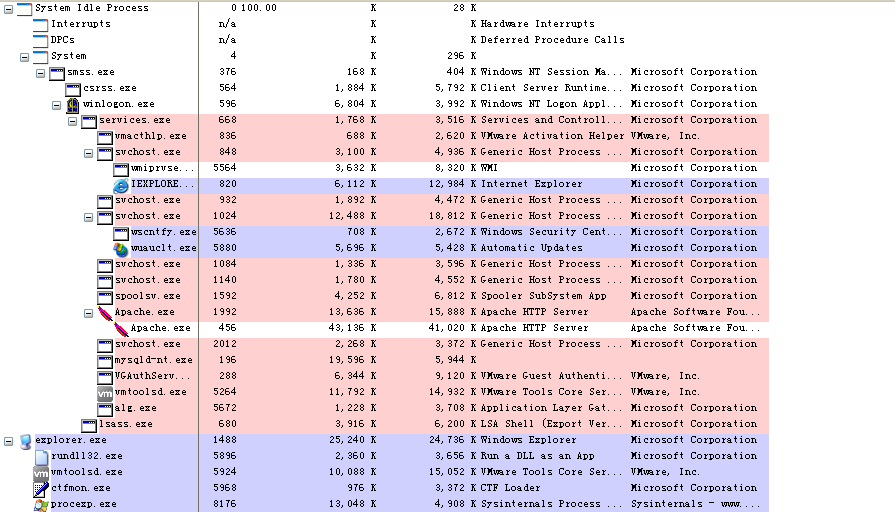


IoGetCurrentProcess调用告诉我们这个驱动或者修改正在运行进程，或者需要关于进程的信息。其它的导入仅仅被驱动用来创建用户态可访问的设备。

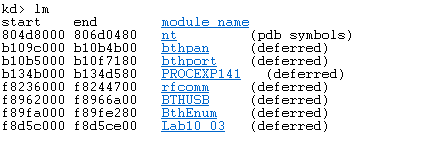
运行该恶意程序，会持续弹出广告界面，而我们在Process Explorer中却没有找到程序



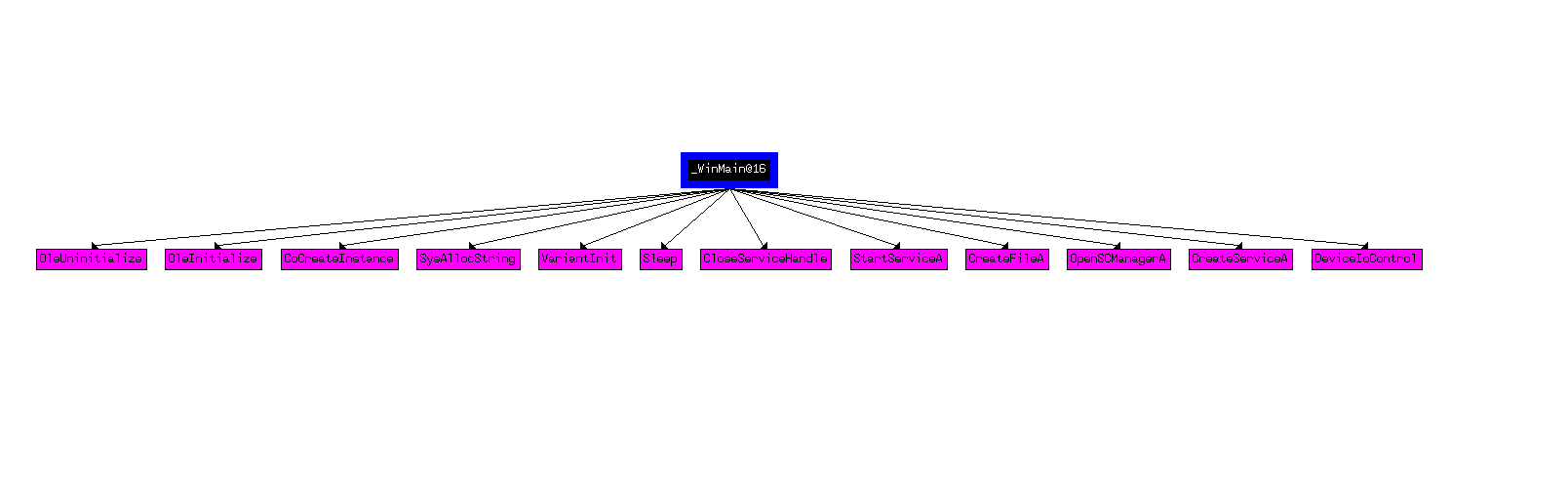
看不到恶意程序，我们无法通过杀死进程的方式结束它



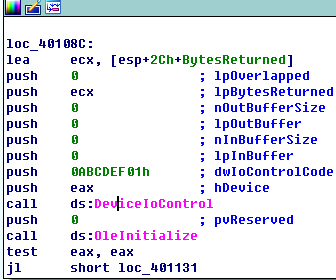
lm可以看到Lab10-03模块



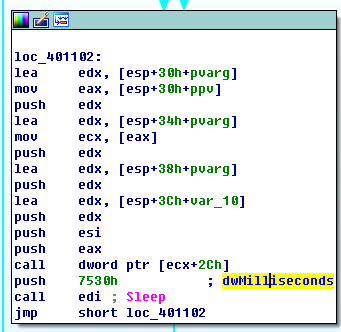
使用IDA Pro静态分析，查看Winmain函数调用的函数，首先它会调用OpenSCManager、DeviceIoControl等函数加载和发送请求到内核驱动。程序创建一个Process Helper服务，加载Lab10-03.sys到内核，并且打开句柄\\.\ProcHelper，这打开一个由ProcHelper驱动创建的内核设备句柄。



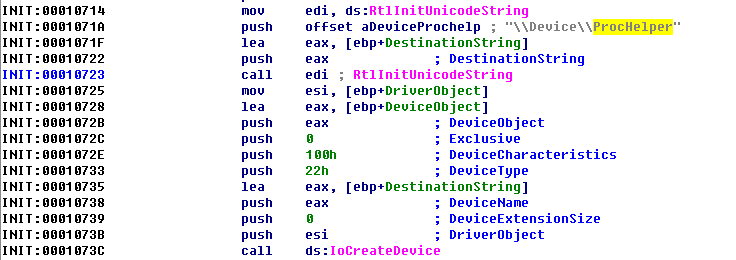
检查DeviceIoControl函数，该调用会发送一个请求到Lab10-03.sys驱动。该函数的输入输出缓冲区被设置为NULL，意味着它没有发送任何消息到内核驱动，且内核驱动没有反馈任何信息。



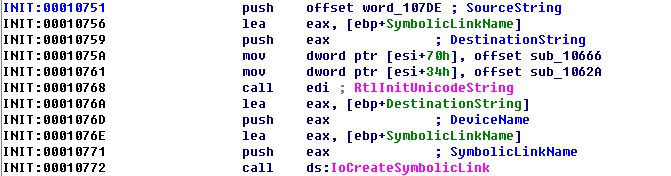
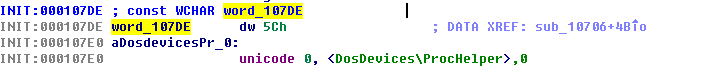
连续运行，在每次调用之间sleep30s，[ecx+2Ch]调用目标暂时不清楚



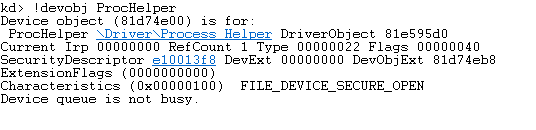
接下来使用IDA Pro分析驱动，调用IoCreateService，创建一个名为\Device\ProHelper的设备



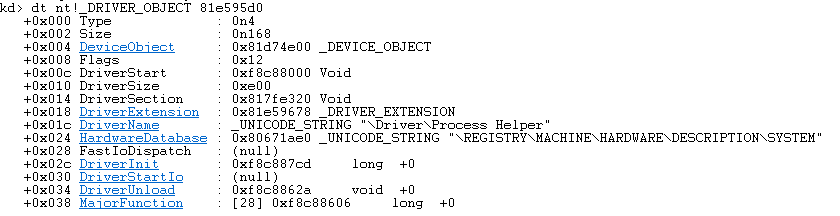
随后函数调用IoCreateSymbolicLink，创建一个名为\DosDevices\ProcHelper的符号链接，来供用户态应用程序访问。



我们尝试使用Windbg查找内存中的驱动



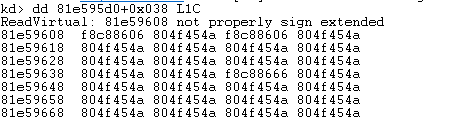
使用nt命令查看标注的驱动对象



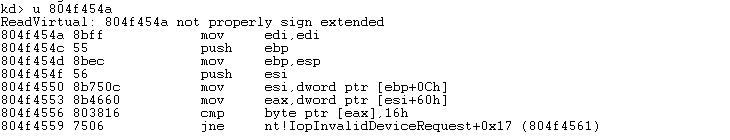
DriverUnload函数当驱动卸载时被调用，它删除了符号链表和创建的设备。



分析主函数表中的函数



0x804f454a处的函数为nt!IopInvalidDeviceRequest，处理驱动无法处理的非法请求。

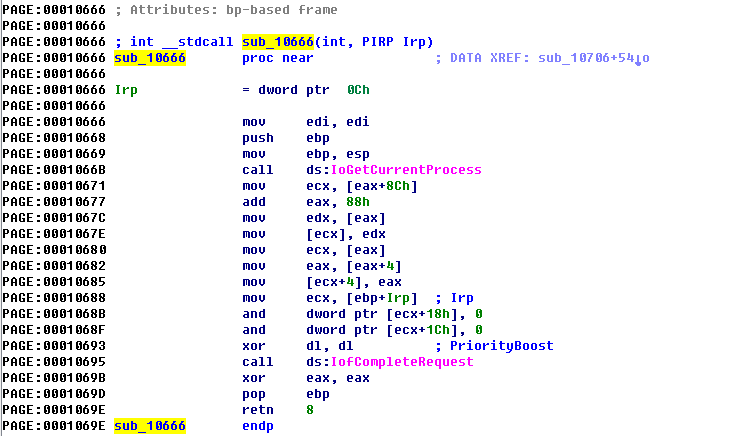


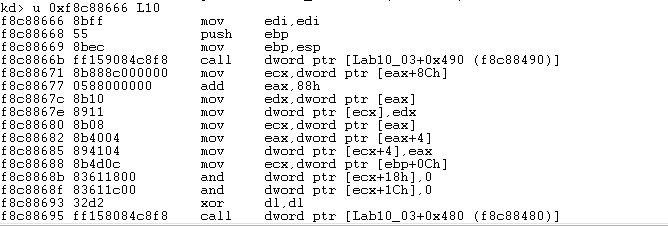
在主函数中还存在不同于0x804f454a的函数，0xf8c88666对应DeviceToControl函数

IDa Pro反汇编sys程序，sub\_10666对应DeviceToControl函数



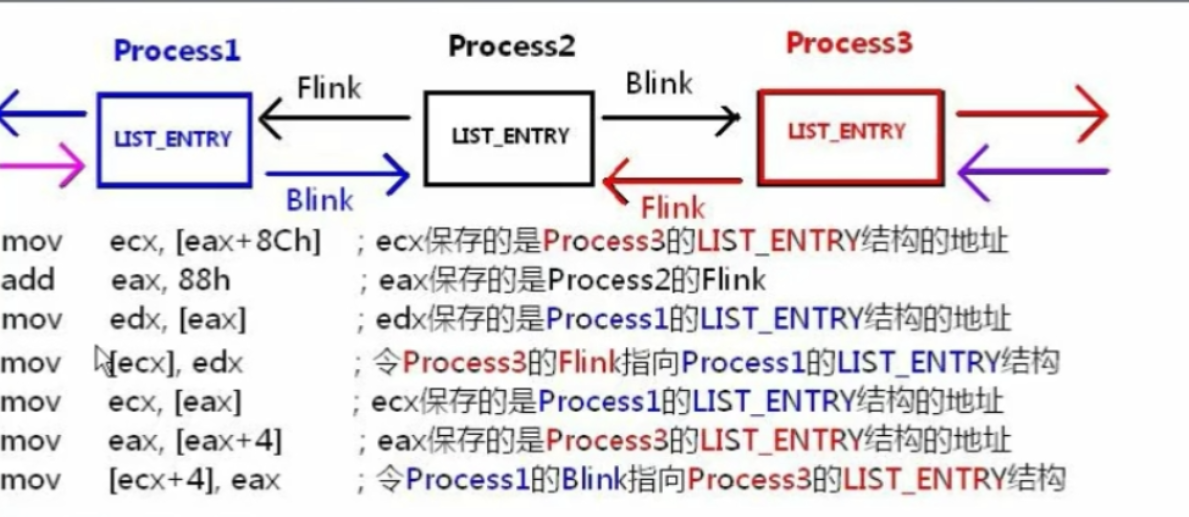
和u 0xf8c88666 L10反汇编得到代码是相同的，这里我们F5反编译分析该函数干的事情





DeviceIoControl操纵了当前进程的PEB，然后对双向链表进行了修改，大致过程如图所示

也就是说恶意病毒通过修改链表节点方式来实现隐藏。



1. 一旦程序运行，你怎样停止它？

重启系统，没有什么好的方法。

1. 它的内核组件做了什么操作？

修改了进程链接表的结构，隐藏了自己的LIST\_ENTRY，可执行文件调用了DeviceIoControl之后，驱动把进程隐藏。

1. **Yara规则**

include ".\includes\IsPE.yar"

include ".\includes\SizeLimit.yar"

rule Lab10\_1

{

    strings:

        $string1 = "C:\\Windows\\System32\\Lab10-01.sys" wide ascii at 0040503C

        $string2 = "Runtime Error!\n\nProgram: " wide ascii at 00404374

        $string3 = "RtlCreateRegistryKey" wide ascii

    condition:

        IsPE and SizeLimit and (all of ($string\*))

}

rule Lab10\_2

{

    strings:

        $string1 = "C:\\Windows\\System32\\Mlwx486.sys" wide ascii at 0040609C

        $string2 = "Runtime Error!\n\nProgram: " wide ascii

        $string3 = "SizeofResource" wide ascii

    condition:

        IsPE and SizeLimit and (all of ($string\*))

}

rule Lab10\_3

{

    strings:

        $string1 = "SizeofResource" wide ascii

        $string2 = "RtlInitUnicodeString" wide ascii

        $string3 = "IoCreateDevice" wide ascii

    condition:

        IsPE and SizeLimit and (all of ($string\*))

}

1. **IDAPython脚本**

编写idapython脚本检测恶意程序的IO操作

一般涉及到内核的恶意程序都会使用DeviceIoControl函数

我们的目的是检测出它的位置以及交叉引用地方

还包括其他IO操作函数如IoGetCurrentProcess、IoCreateDevice等等

from idautils import \*

from time import strftime

INTERESTING\_SUBSTRINGS = [\

("DeviceIoControl", "Windbg"),\

("IoGetCurrentProcess", "Windbg"),\

("IoCreateDevice", "Windbg"),\

("IoCreateSymbolicLink", "Windbg"),\

]

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

print("\n-------------------- Interesting Api Calls ------------------------------")

print("Run at: %s" % (strftime("%Y-%m-%d %H:%M:%S")))

for call\_ea, name in sorted(Names()):

demangled\_name = Demangle(name, INF\_SHORT\_DN)

if demangled\_name != None:

name = demangled\_name

for keyword, note in sorted(INTERESTING\_SUBSTRINGS):

if keyword.lower() in name.lower():

xrefs = sorted([ x for x in CodeRefsTo(call\_ea, 0) if SegName(call\_ea) != ".data" ])

if len(xrefs) > 0:

print("\nCalls To: (%x) %s (%s)" % (call\_ea, name, note))

for xref in xrefs:

print(" [%x] %s" % (xref, GetFunctionName(xref)))