

**《恶意代码分析与防治技术》课程实验报告**

**实验十**

****

学 院 网络空间安全学院

专 业 信息安全

学 号 2112060

姓 名 孙蕗

班 级 信息安全1班

1. **实验目的**

理解恶意软件的用户态行为，包括文件创建、注册表操作等，以及对系统的影响。

学习如何使用工具（如IDA Pro、Procmon、Resource Hacker等）来分析和反汇编可执行文件，查看导入函数、资源等信息。

掌握检查内核态组件的方法，了解驱动程序的加载和执行过程。

学习使用调试工具（如Windbg）分析内核驱动的行为，包括对系统结构的修改和对系统进程的隐藏。

1. **实验原理**

恶意软件通过修改系统的注册表、创建文件、创建服务、修改内核、修改代码、加载内核驱动、隐藏进程等方式对系统进行破坏。

使用Regedit等工具查看注册表的变化，使用Procmon监视系统文件和服务的创建，以及对文件的写入等操作。

恶意软件通过等手段影响系统行为。

使用Resource Hacker等工具查看可执行文件的导入函数表、资源节等信息。

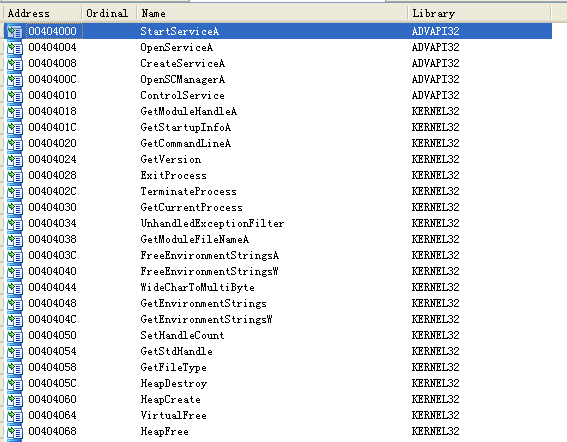
使用Windbg和IDA Pro等工具查看内核驱动的代码，了解其加载过程。

通过分析DeviceIoControl请求，学习内核态驱动对系统进程的隐藏原理。

1. **实验过程**
2. **Lab10-1**

**必须将驱动程序放到C:\Windows\System32目录下,可执行文件是Lab10-1.exe,驱动程序是Lab10-01.sys。**

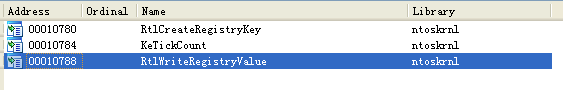
Ida pro查看Lab10-1.exe的导入表





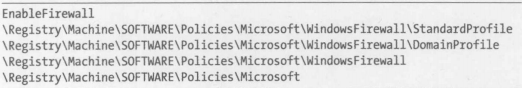
有意义的导入函数有OpenSCManagerA、OpenServiceA、ControlService、StartServiceA以及CreateServiceA。这表明这个程序创建了一个服务，并且可能启动或者操作了这个服务，除此之外，它与系统进行了很少的交互。

Ida pro查看Lab10-1.sys的导入表



它仅仅导入了3个函数。第一个函数是KeTickCount,几乎所有驱动都包含它，可以忽略。剩下的两个函数是RtlCreateRegistryKey和RtlWriteRegistryvalue,这告诉我们驱动可能访问了注册表。

查看Lab10-1.sys的字符串

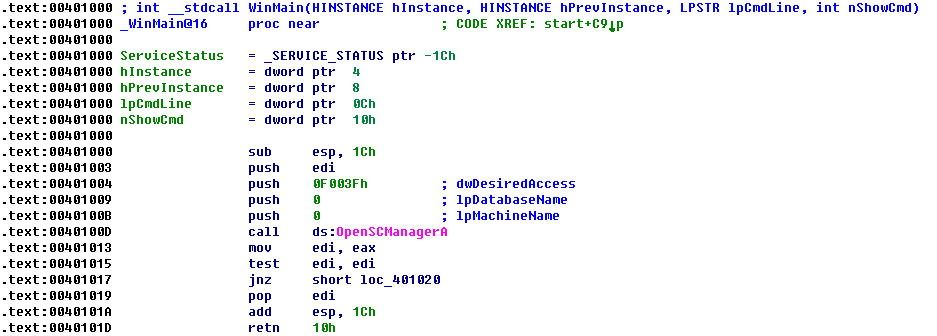


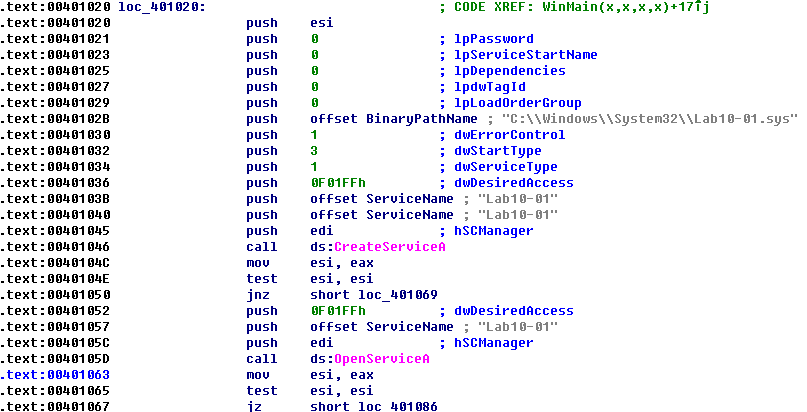
这些字符串看起来很像是注册表键值，然而开头\Registry\Machine却比较怪异，并不像是诸如HKLM此类的常见注册表根键。当从内核态访问注册表时，前缀\Registry\Machine等同于用户态程序访问的HKEY\_LOCAL\_MACHINE。网上搜索一下就可以知道，将EnableFirewall置为0,表示禁用Windows XP自带的防火墙。

双击运行Lab10-01.exe，procmon中显示有一些读注册表的调用，但是仅有一个写注册表的调用：RegSetValue写了HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed键值。这个注册表键值一直改变，对于恶意代码分析毫无意义，但是因为涉及到内核代码，需要确保驱动没有秘密地修改注册表。



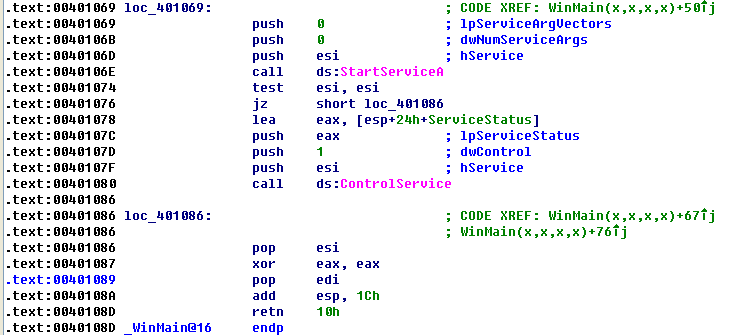
Ida pro查看Lab10-1.exe的main





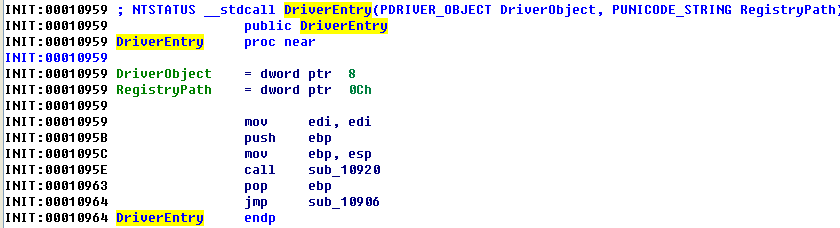
首先，它在0x0040100D调用了OpenSCManagerA获取服务管理器的句柄，然后在0x00401046调用CreateServiceA,创建一个名为Lab10-01的服务。CreateServiceA调用说明服务使用0x00402B的C:\Windows\System32\Lab10-01.sys中的代码，服务类型为3(SERVICE\_KERNEL\_DRIVER),意味着这个文件将被加载到内核。

如果CreateServiceA调用失败，代码会使用相同的服务名调用OpenServiceA,如0x0040100D所示。如果因为服务已经存在而导致调用CreateServiceA失败，这将打开Lab10-01服务的句柄。

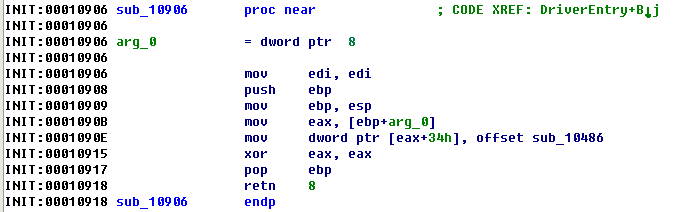


接下来，程序调用StartServiceA来启动服务，如loc\_401069所示。最后，0x00401080调用ControlService.ControlService的第二个参数是发送控制消息的类型，它的值是0x010,SERVICE\_CONTROL\_STOP。这将会卸载驱动，并调用驱动的卸载函数。

Ida pro查看 Lab10-1.sys的DriverEntry函数

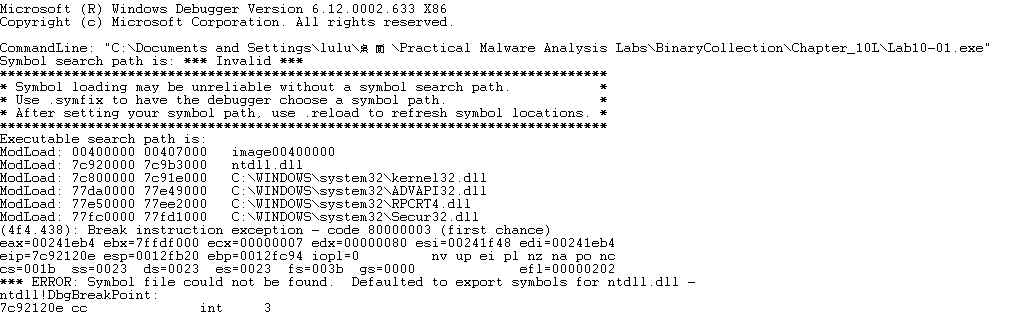


这个函数是驱动的入口点，但是它不是DriverEntry函数。编译器在DriverEntry的周围插入封装代码。真正的DriverEntry函数位于sub\_10906。

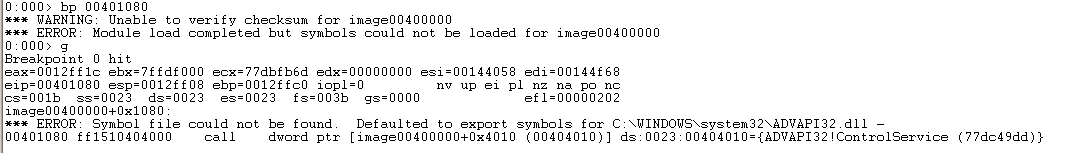


DriverEntry函数的主体部分似乎将一个偏移量移入到一个内存位置，然后除此之外，它没有进行任何函数调用，也没有与系统进行交互。

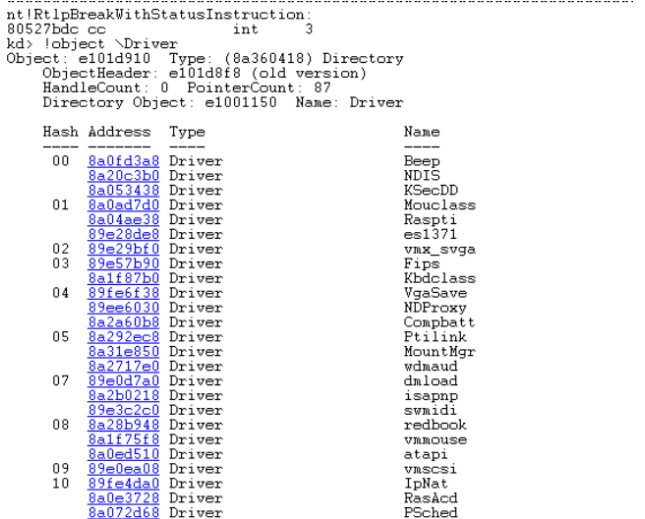
使用WinDbg分析Lab10-1.sys



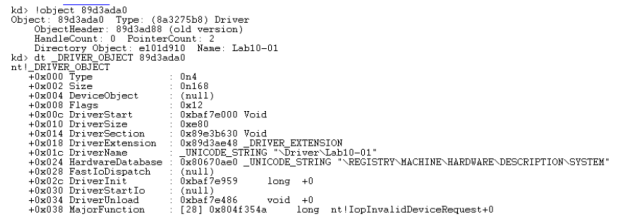
将可执行程序加载到windbg中，结合之前使用ida得到的controlservice地址进行断点。使用bp 00401080，在驱动和卸载之间设置一个断点（在ControlService调用上。启动程序debug->go直到断点命中。



一旦程序在断点处暂停，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。打开Windbg的另一个实例，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。通过使用命令!drvobj,来获取驱动对象。



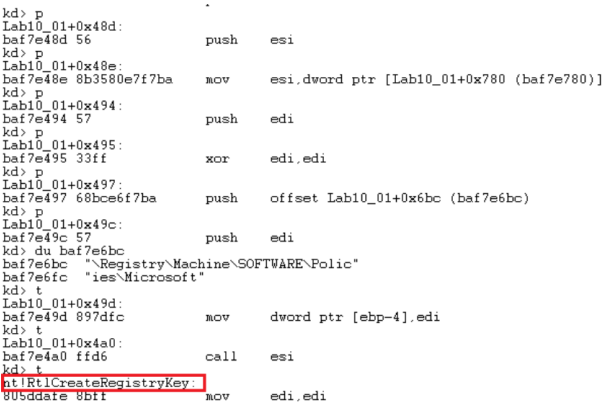
使用dt \_DRIVER\_OBJECT 地址 解析地址的数据结构。



确定驱动卸载时调用的函数——偏移量0x034的信息DriverUnload。

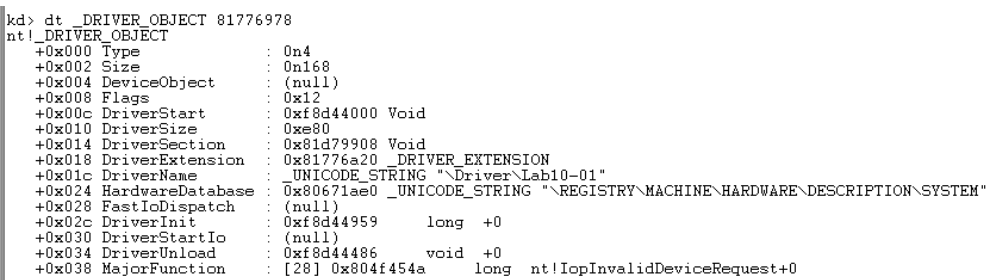
使用命令bp 0xbaf7e486设置一个断点,恢复内核运行。回到虚拟机中运行可执行程序的WinDbg,也恢复它的运行。由于内核调试器命中了断点，整个Guest OS会卡死。

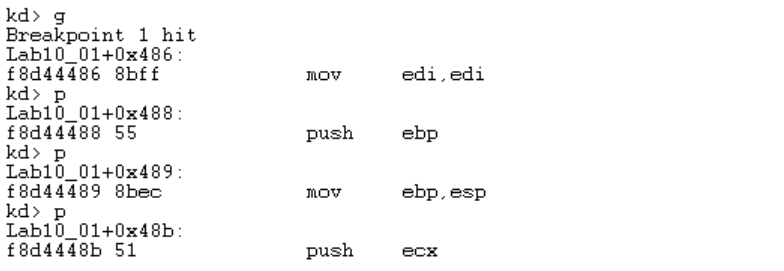
此时，回到内核调试器，g单步调试代码。



看到程序调用了多次RtlCreateRegistryKey函数，创建了一些注册表键，然后调用了多次RtlWriteRegistryValue函数，在两个地方设置EnableFirewall值为0。从内核禁用Windows XP防火墙的这种方法难以被安全程序探测到。

IDA中ControlService的地址为0x401080,使用命令!drvobj lab10-01，来获取驱动对象。使用命令bp 0xf8d44486设置一个断点，在Windbg进行单步调试。





前三行汇编代码是一个函数入口的地方，说明刚刚那个断点断在了一个函数的入口。之后多次调用了RtlCreateRegistryKey函数来创建注册表键，并调用了两次RtlwriteRegistryalue函数将EnableFirewall的值设置为0，禁用Windows XP防火墙。

1. **这个程序是否直接修改了注册表(使用procmon来检查)?**

如果使用procmon监视这个程序，你会看到唯一写注册表的调用是写键值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，但这个程序也从内核对注册表做了直接修改，这些修改却不能被procmon探测到。

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?**

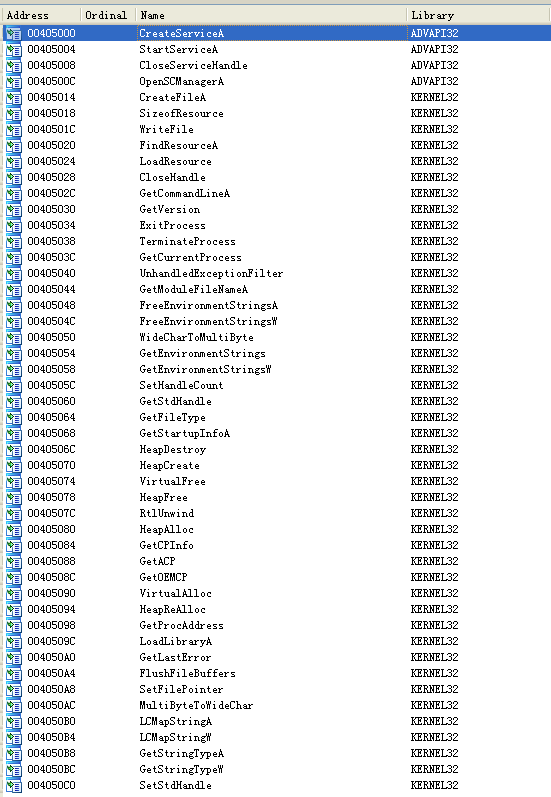
要设置一个断点来查看内核发生了什么，你必须使用一个运行在虚拟机中的WinDbg实例，来打开这个可执行文件，而调试内核使用运行在宿主操作系统中的WinDbg另外一个实例。当Lab10-01.exe在虚拟机中被暂停后，使用!drvobj命令获得驱动设备的句柄，它包含一个卸载函数的指针。接下来，在驱动的卸载函数上设置一个断点。重启Lab10-01.exe之后，断点将会被触发。

1. **这个程序做了些什么?**

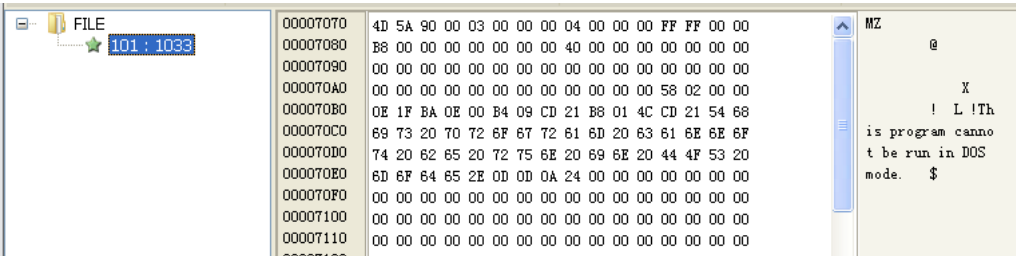
这个程序创建一个服务来加载驱动。然后，驱动代码会创建注册表键\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFirewall\StandardProfile和\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFirewall\DomainProfile。在Windows XP系统中，设置这些键值将禁用防火墙。

1. **Lab10-2**

查看这个可执行文件的导入函数表，看到CloseServiceHandle、CreateServiceA、OpenSCManagerA以及StartServiceA,说明这个程序创建并启动一个服务。程序也调用了CreateFile和WriteFile,说明程序将在某个时间点写文件。LoadResource和SizeOfResource调用，说明这个程序对Lab10-02.exe的资源节做了某些处理。

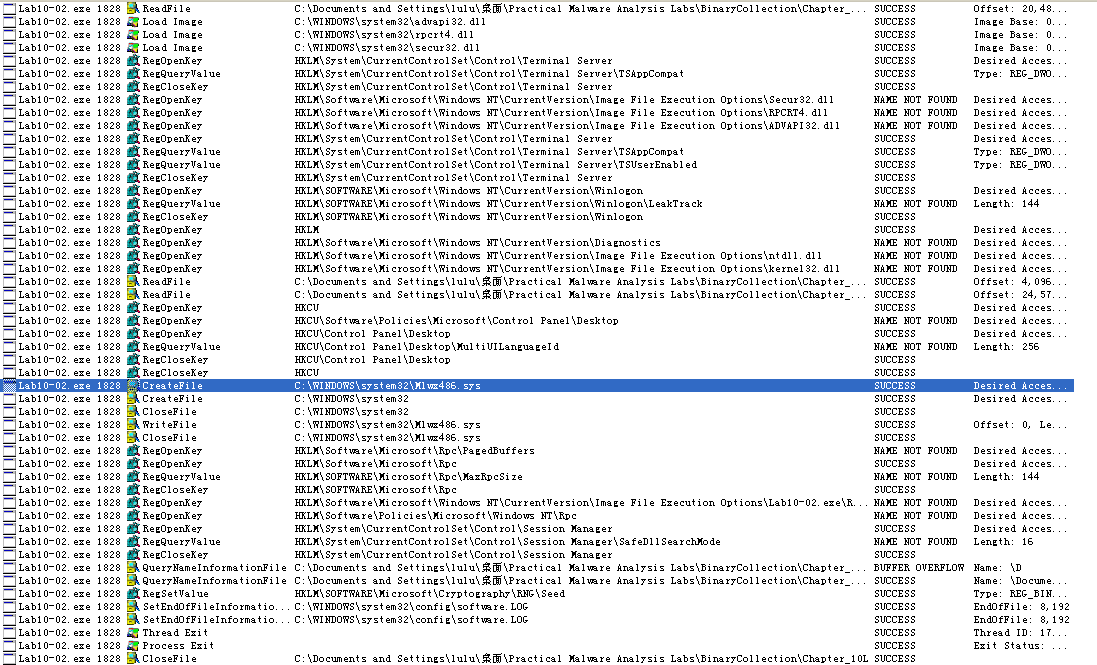


使用Resource Hacker来检查资源节，发现这里有一个FILE，里面包含了另一个PE头。这可能是Lab10-02将要使用的另外一个恶意文件。

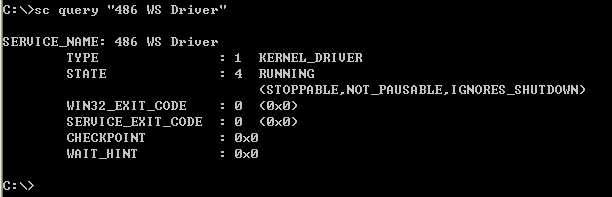


运行程序，发现它创建了一个文件和服务。使用procmon,看到程序在C:\Windows\System32目录下创建了一个文件，并且以这个文件作为一个可执行程序创建了一个名为486 WS Driver服务。这个文件包含了由操作系统加载的内核代码。这个文件一共写了三个文件，一个是system.LOG，一个是system，还有一个是SysEvent.Evt。文件创建了conime.exe、apphelp.dll，sysmain.sdb，systest.sdb，以及Mlwx486.sys。

然而到上述目录下却没有找到程序创建的文件，且在procmon中并没有关于这个文件被删除的操作，推测这个文件被隐藏了，是一个Rootkit。

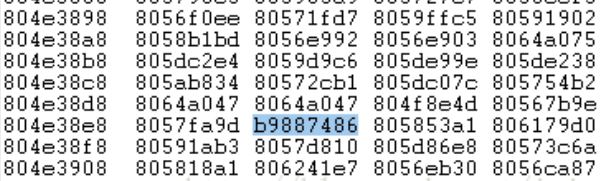


使用sc命令，检查运行内核驱动程序的服务状态

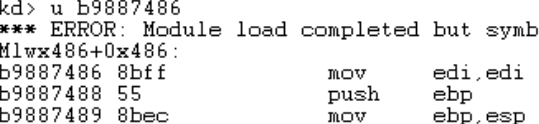


查询名为486 WS Driver的服务,它在CreateServiceA中被指定。看到这个服务仍然运行,这告诉我们内核代码在内存中。奇怪的是驱动仍然运行，但是它没有在硬盘上。

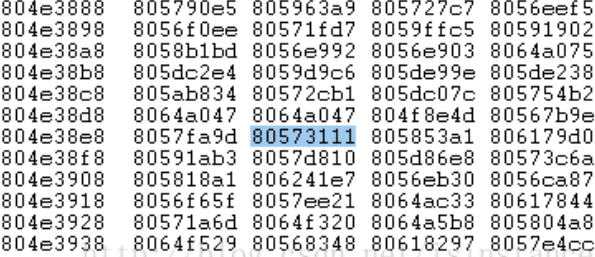
连接虚拟机与内核调试器，然后使用1m命令，查看驱动是否被真正加载。文件名为Mlwx486.sys的驱动被载入到内存，但文件没有在硬盘上显示，推测这个文件可能是一个Rootkit。使用命令dd dwo(KeServiceDescriptorTable) L100检查SSDT的修改情况，发现异常地址b9887486。



使用命令 u b9887486查看此处的内容，发现是一个Mlwx486里面自带的函数。



b9887486处条目所在的内存位置很明显位于ntoskrn1模块的范围之外，位于加载的Mhwx486.sys驱动内。恢复虚拟机到Rootkit安装之前的状态，使用命令dd dwo(KeServiceDescriptorTable) L100以便查看存储在SSDT中的哪个函数被覆盖了。



被覆盖的函数是NtQueryDirectoryFile,它是一个提取文件和目录信息的通用函数，而FindFirstFile和FindNextFile都是调用它来遍历目录结构的。Windows资源管理器也会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，这也解释了不能发现Mlwx486.sys的原因。

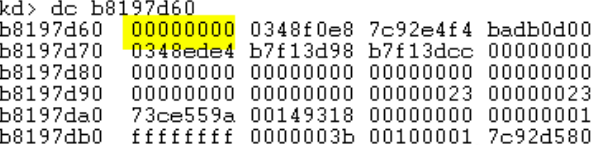
在代替NtQueryDirectoryFile调用的挂钩函数PatchFunction上设置一个断点，发现它做的第一件事是用原始参数调用原始的NtQueryDirectoryFile函数。

使用命令bp f7a75486设置断点，进行单步调试。



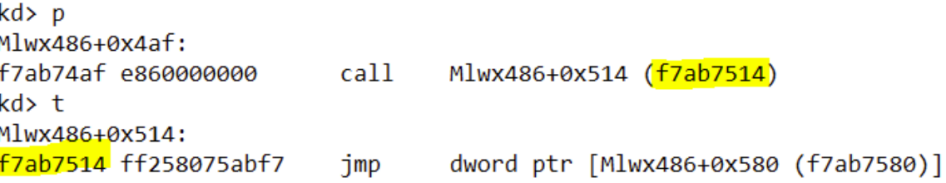
|  |
| --- |
| mov edi, edi  push ebp  mov ebp, esp  push esi  mov esi, dword ptr [ebp+1Ch]  push edi  push dword ptr [ebp+30h]  push dword ptr [ebp+2Ch]  push dword ptr [ebp+28h]  push dword ptr [ebp+24h]  push dword ptr [ebp+20h]  push esi  push dword ptr [ebp+18h]  push dword ptr [ebp+14h]  push dword ptr [ebp+10h]  push dword ptr [ebp+0Ch]  push dword ptr [ebp+8]  call Miwx486+0x514(f7a75514) |

到这里，函数开始调用函数Miwx486+0x514。先计算地址的值，然后查看内存地址上的值是多少。



|  |
| --- |
| mov esi, dword ptr [ebp+1Ch]  push edi = b95a5d64  /\* Note: ebp = b95a5d30 \*/  push dword ptr [ebp+30h] = [b95a5d60] = 0  push dword ptr [ebp+2Ch] = [b95a5d5c] = 80 e1 70  push dword ptr [ebp+28h] = [b95a5d58] = 1  push dword ptr [ebp+24h] = [b95a5d54] = 3  push dword ptr [ebp+20h] = [b95a5d50] = 268  push esi = 0070e198  push dword ptr [ebp+18h] = [b95a5d48] = 68 e1 70  push dword ptr [ebp+14h] = [b95a5d44] = 0  push dword ptr [ebp+10h] = [b95a5d40] = 0  push dword ptr [ebp+0Ch] = [b95a5d3c] = 0  push dword ptr [ebp+8] = [b95a5d38] = 464  call Miwx486+0x514(f7ab7514) |

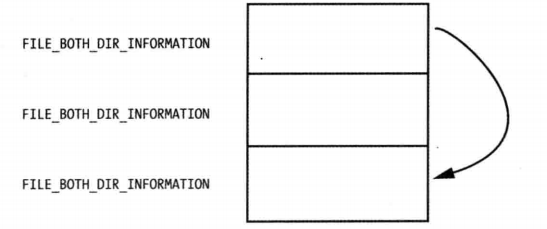
PatchFunction函数检查第8个参数FileInformationClass,如果它是3之外的值，便会返回NtoueryDirectoryFile的原始返回值。另外，它也检查NtQueryDirectoryFile的返回值与第9个参数值ReturnSingleEntry。PatchFunction查找某个参数，如果参数不符合条件，那么它的功能与原始函数一样。如果参数满足了条件，则PatchFunction将会修改返回值。



在PatchFunction上设置一个断点，但是当且仅当ReturnSingleEntry为0时，断点才被触发，断点此处是Rt1CompareMemory函数调用。

RtlCompareMemory的第一个参数是eax,它存储了偏移量esi+5eh,这是文件名的偏移量。在先前的反汇编代码中，esi是FileInformation,包含NtQueryDirectoryFile填充的信息。这是一个FILE\_BOTH\_DIR\_INFORMATION结构，则0x5E偏移量是存储宽字符串文件名的地址，这段代码用于比较每个文件的文件名开头的四个字节是否是"Mlwx" 。

使用命令db esi+5eh查看存储在esi+5eh的内容，如果传入的FileName和Mlwx不相等，函数直接退出。



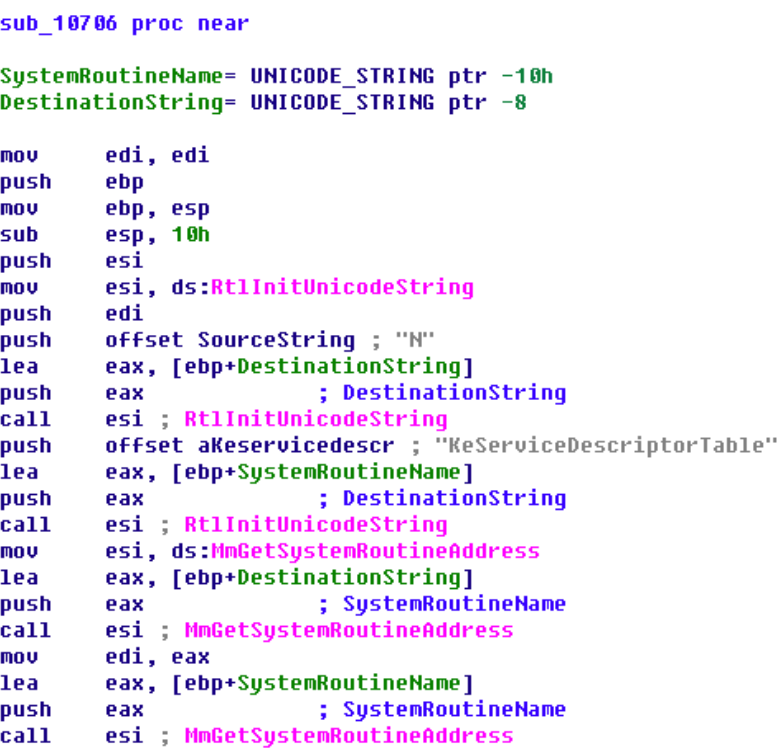
假设第一个结构体的偏移量为00000060，结构体长度为0x60，再假设恶意驱动在NtQueryDirectoryFile返回的第三个结构体里面发现了文件名字和Mlwx匹配，之后通过赋值语句mov eax, dword ptr[esi]获得了Mlwx486.sys这个文件的FILE\_BOTH\_DIR\_INFORMATION结构里面的NextEntryOffset的值，然后将这个这个偏移值乘以2。

程序在第三个结构体发现Mlwx之后，通过add dword ptr [edi], eax偏移值NextEntryOffset的值变成了00000180（c0\*2）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

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。

使用DOS命令copy Mlwx486.sys NewFilename.sys来复制文件，文件NewFilename.sys应该不会被隐藏。

使用IDA打开NewFilename.sys查看DriverEntry例程



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，它在SSDT中查找NtQueryDirectoryFile函数项，并用PatchFunction的地址覆盖这个项。它没有创建设备，也没有向驱动对象添加任何处理函数。

1. **这个程序创建文件了吗?它创建了什么文件?**

这个程序创建了文件C:\WindowslSystem32\Mlhwx486.sys。你可以使用procmon或者其他动态监视工具来查看文件创建，但是，因为文件被隐藏，所以在硬盘上看不到它。

1. **这个程序有内核组件吗?**

这个程序拥有一个内核模块。这个内核模块被存储在这个文件的资源节中，然后写入硬盘并作为一个服务加载到内核。

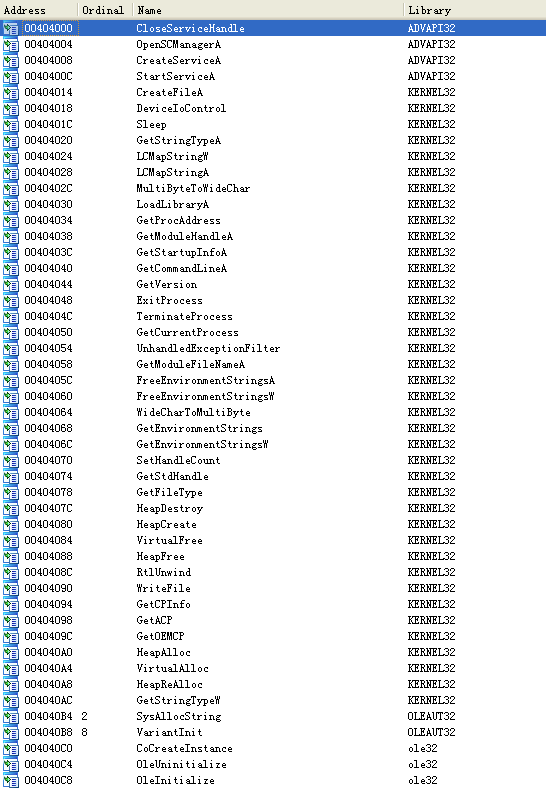
1. **这个程序做了些什么?**

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，它会隐藏目录列表中任何以M/wx(区分大小写)开头的文件。

1. **Lab10-3**

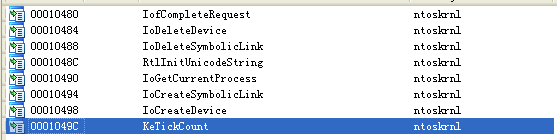
**必须将驱动程序放到C:\Windows\System32目录下,可执行文件是Lab10-3.exe,驱动程序是Lab10-03.sys**

查看Lab10-03.exe的导入表



存在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。

查看Lab10-03.sys的导入表



IoCreateDevice例程创建供驱动程序使用的设备对象

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

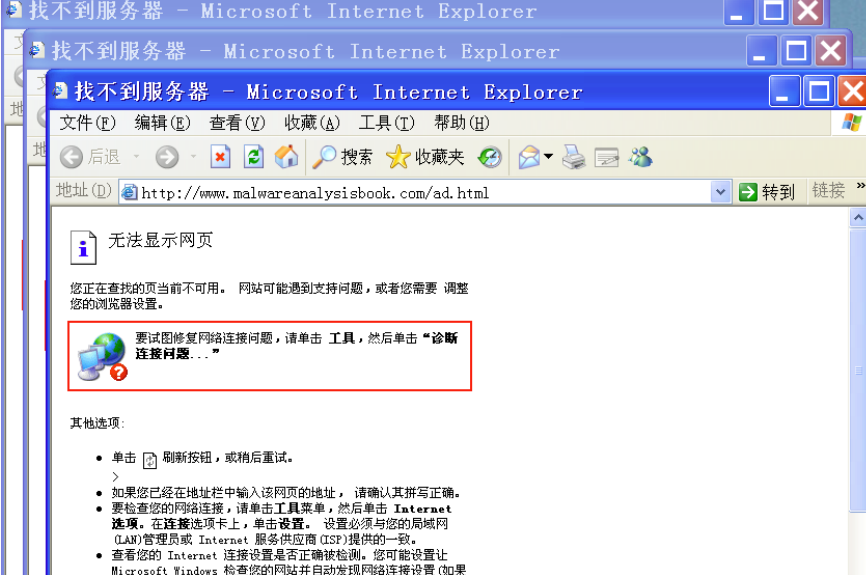
IoGetCurrentProcess例程返回一个指向当前进程的指针

IoCompleteRequest例程指示调用者已完成给定I/O请求的所有处理，并将给定的IRP返回给I/O管理器

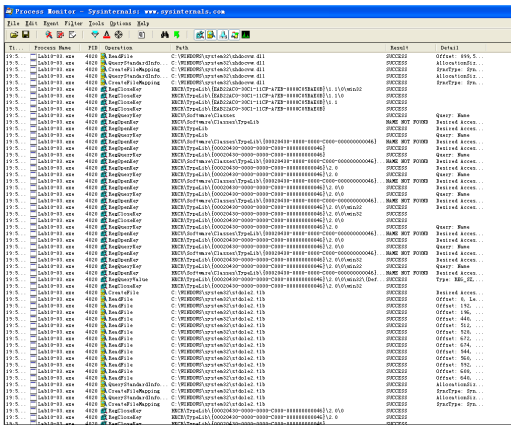
KeTickCount检索自系统启动以来经过的毫秒数，最长为49.7天

RtlInitUnicodeString初始化一个统计的Unicode字符串

其中仅有IoGetCurrentProcess导入函数提供了足够信息(其他的导入仅仅被驱动用来创建用户态可访问的设备)。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。将驱动文件复制到C:\Windows\System32,双击可执行程序运行它，看到一个弹出广告。



查看Procmon的结果，看到很多对注册表、文件的操作。



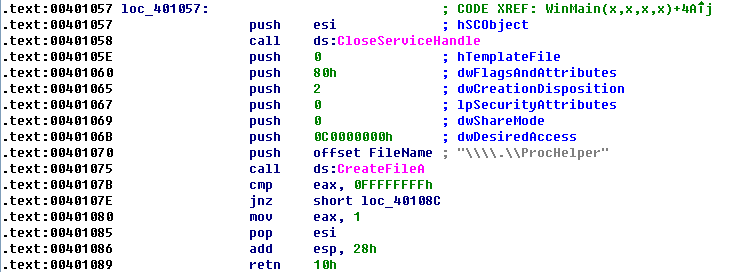
首先，查看服务是否被正确安装，并且验证恶意.sys是否作为服务的一部分。同时，注意到大约30秒后，程序再次弹出广告，每隔30秒执行一次。当打开任务管理器试图终止进程时，看到程序没有被列出，另外在Process Explorer中也没有列出。程序继续打开广告，没有一种简单办法可以停止它。因为没有在进程列表中，所以不能通过杀掉进程的方式，来结束它。

因为程序没有在WinDbg或者OllyDbg进程列表中显示，因此也不能附加一个调试器到进程。此时，唯一的选择是还原到最近快照或是重启系统，希望程序不能持续，如果它不能持续，重启会停止它。

在IDA pro中分析.exe可执行文件，找到WinMain函数。



第一个调用的函数是OpenSCManagerA，如果调用成功，执行0x00401018。字符串变量C:\\Windows\\System32\\Lab10-03.sys被压入栈中。BinaryPathName的值是那个驱动文件的路径，用于指明服务的二进制文件的位置的参数。dwStartType参数的值为3，意思是用户可以使用“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。下一个参数是dwServiceType的值为1，意义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，执行StartService，恶意驱动Lab10-03.sys就会被加载到内核中。



创建了一个文件在\\.\ProcHelper这个地方并作为一个句柄打开。

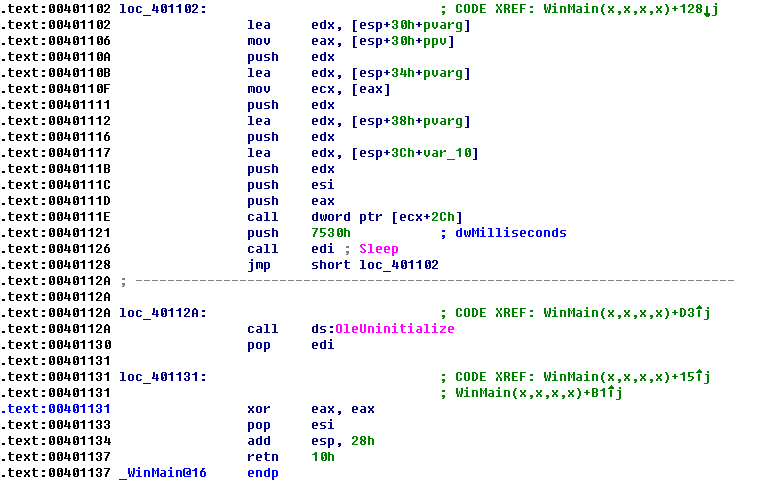


DeviceIoControl函数是将控制代码直接发送到指定的设备驱动程序，导致相应的设备执行相应的操作。这里DeviceIoControl的参数lpInBuffer和lpOutBuffer被设置为了Null很不寻常，意味着这个请求没有发送任何的信息到内核驱动中(lpInBuffer=0)，并且内核驱动的反馈也是没有的(lpOutBuffer=0)。

还有个异常的地方就是dwIoControlCode的值是abcdedf01。

CoCreateInstance创建与指定的CLSID关联的类的单个未初始化对象。

在调用SysAllocString之前，字符串”http://www.malwareanalysisbook.com/ad.html”被压入了栈中,这个字符串就是开始运行时不断打开的链接。

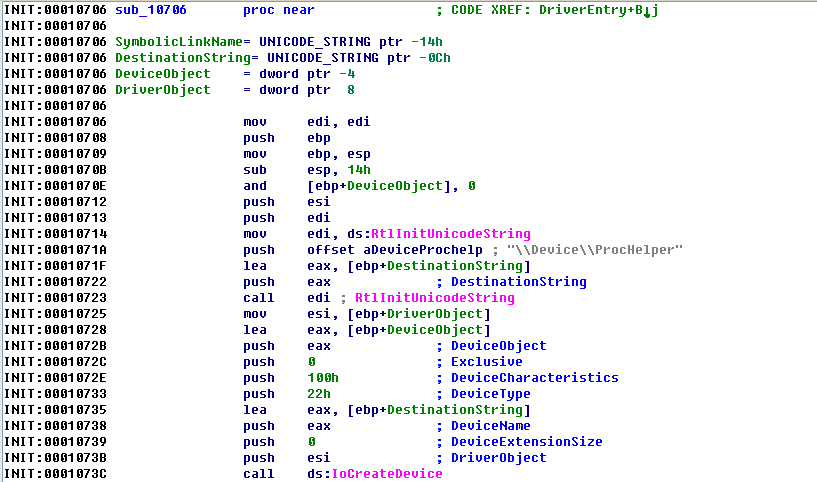


最后调用Sleep函数休眠了0x7530h毫秒，然后就是一直循环这个代码块，直到关机为止。

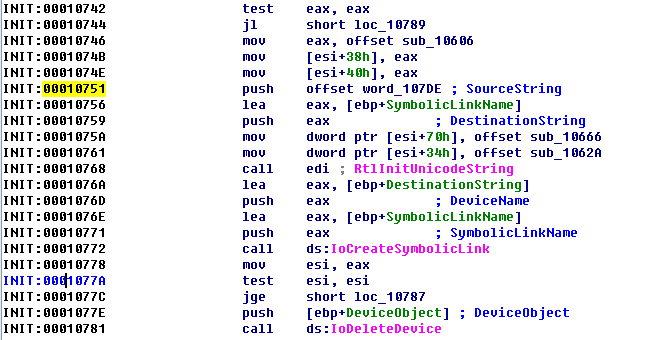
WinMain在逻辑上可以分为两部分：第一部分由OpenSCManager到DeviceIoControl之间的函数调用组成，包含加载和发送请求到内核驱动的函数。第二部分由其余的函数调用组成，这表明是一个COM对象的使用。

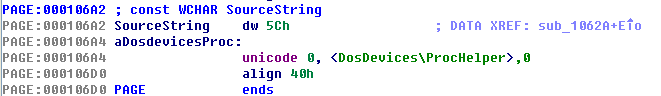
程序创建了一个叫做Process Helper的服务，它负责加载内核驱动：C:\Windows\System32\Lab10-03.sys。然后启动Process Helper服务，加载Lab10-03.sys到内核，并且打开句柄\\.\ProcHelper,这打开一个由ProcHelper驱动创建的内核设备句柄。

在IDA pro中打开.sys内核文件，查看汇编代码



调用了IoCreateDevice，创建一个名为\Device\ProHelper的设备。

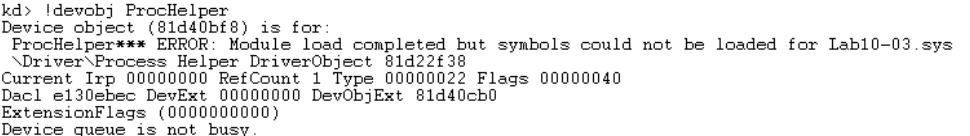




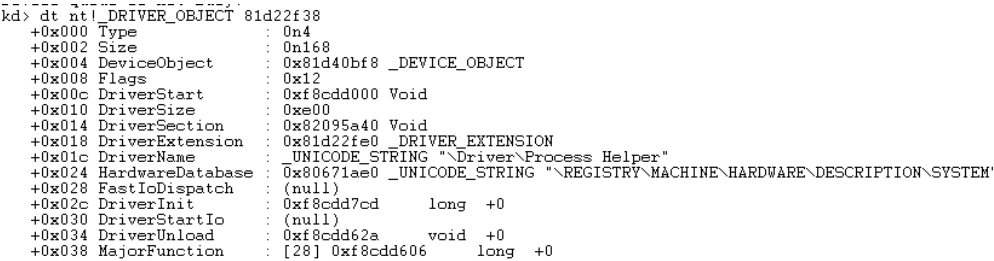
调用IoCreateSymbolicLink时，创建一个名为\DosDevices\ProHelper的符号链接，供用户态应用程序访问。

使用Windbg查找内存中的驱动

使用命令!Devobj ProcHelper查找DriverObject存储的位置。

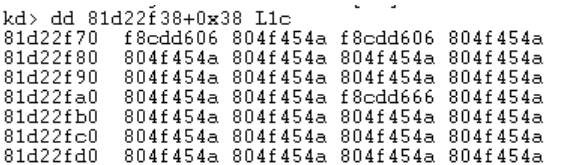


使用命令dt nt!\_DRIVER\_OBJECT 81d22f38查看标注的驱动对象，其中，DriverInit是驱动初始化的操 作地址，DriverUnload是驱动卸载时候的操作地址。在IDA Pro中查看DriverUnload时，会看到它删除了符号链表和DriverEntry创建的设备。



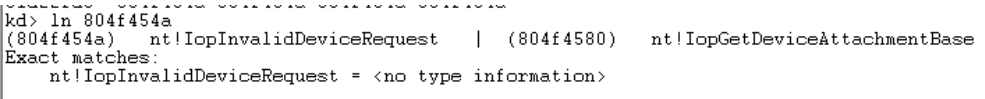
分析主函数表中的函数

使用命令dd 81d22f38+0x38 L1c命令查看主函数表中表项

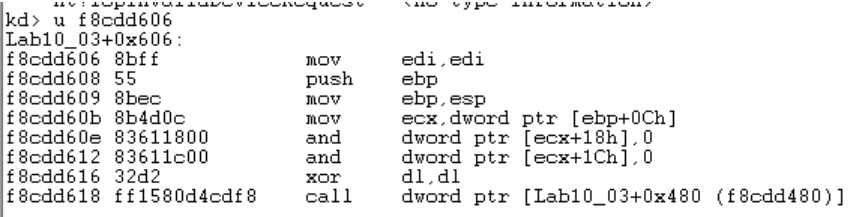


表中大部分项都是0x804f454a表示驱动不能处理的一个请求类型。

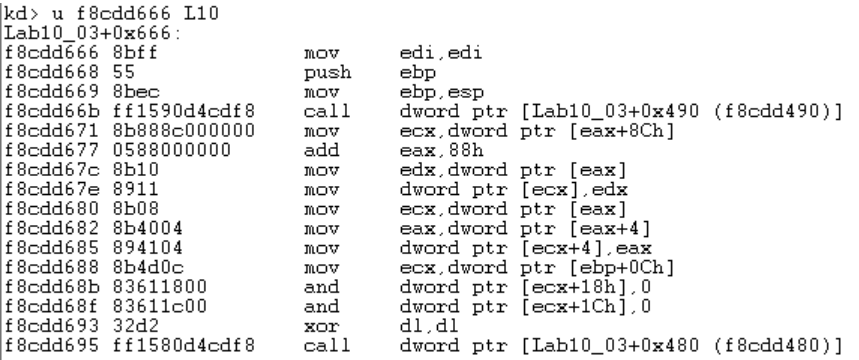
使用命令ln 804f454a，看到这个函数被命名为IopInvalidDeviceRequest，用于处理驱动无法处理的非法请求。



查看wdm.h,发现偏移量0、2、0xe存储Create、Close以及DeviceIoControl函数。主函数表中偏移量为0、2的两项指向同一个函数（0xf8cdd606）。看到这个函数只调用IofCompleteRequest,然后返回。这告诉操作系统请求成功，但是什么也没有做。主函数表中剩下的唯一函数是处理DeviceIoControl的请求，它是最有价值的。



查看DeviceIoControl,看到它操纵了当前进程的PEB。



DeviceIoControl函数做的第一件事情就是调用IoGetCurrentProcess,它返回调用DeviceIoControl进程的EPROCESS结构。然后，这个函数在处访问偏移量0x88处的数据，再然后，访问偏移量0x8C处的下一个DWORD。

使用dt命令发现存储在PEB结构偏移量0x88和0x8C的LIST\_ENTRY。

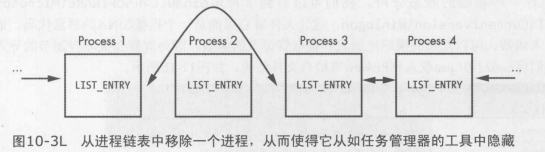
|  |
| --- |
| kd>dt nt!\_EPROCESS  +0x000 Pcb：KPROCESS  +0x06c ProcessLock：EX\_PUSH\_LOCK  +0x070 CreateTime：\_LARGE\_INTEGER  +0x078 ExitTime：\_LARGE\_INTEGER  +0x080 RundownProtect：\_EX\_RUNDOWN\_REF  +0x084 UniqueProcessId：Ptr32 Void  +0x088 ActiveProcessLinks：\_LIST\_ENTRY  +0x090 QuotaUsage：[3]Uint4B  +0x09c QuotaPeak：[3]Uint4B  +0x0a8 CommitCharge：Uint4B  +0x0ac PeakVirtualSize：Uint4B  +0x0b0 VirtualSize：Uint4B  +0x0b4 SessionProcessLinks :\_LIST\_ENTRY  +0x0bc DebugPort：Ptr32 Void  +0x0c0 ExceptionPort：Ptr32 Void  +0x0c40bjectTable：Ptr32 \_HANDLE\_TABLE |

LIST\_ENTRY结构是一个包含两个值的双向链表：第一个是BLINK,它指向列表中的前一项，第二个是FLINK,它指向列表中的下一项。程序不仅读取LIST\_ENTRY结构而且修改这个结构。

|  |
| --- |
| mov eax,[eax+8Ch] ①  add eax,88h  mov edx,[eax] ②  mov [ecx],edx ③  mov ecx,[eax] ④  mov eax,[eax+4] ⑤  mov [ecx+4],eax ⑥ |

①处指令获取列表中指向下一项的指针。②处指令获取列表中指向前一项的指针。③处的指令覆盖下一项的BLINK指针，使其指向前一项⑥。在此之前，下一项的BLINK指针指向当前项。③处的指令覆盖BLINK指针，从而使它跳过当前指针。④、⑤、⑥处的指令执行相同的步骤，除了覆盖列表中前一项的FIINK指针来跳过当前项。

除了修改当前进程的EPROCESS结构之外，上面代码还会修改进程链中的前一个或者后一个进程的EPROCESS结构。这个六条指令通过从加载进程的列表中解除链接，来隐藏当前进程。



当操作系统正常运行时，每个进程都有一个指向它前一个进程或者后一个进程的指针。然后，在图10-3L中，进程2被这个Rootkit隐藏，当OS遍历进程链表时，隐藏进程总是被跳过。

1. **这个程序做了些什么?**

用户态程序加载驱动，然后每隔30秒就弹出一个广告。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。

1. **一旦程序运行，你怎样停止它?**

一旦程序运行，除了重启以外，没有任何一种办法可以轻易停止它。

1. **它的内核组件做了什么操作?**

为了对用户隐藏进程，内核组件负责响应，从进程链接表中摘除进程的DeviceIoControl请求。

1. **Yara规则**

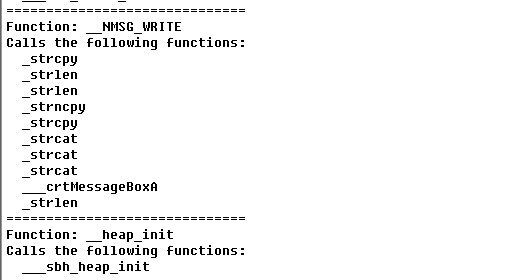
根据特征字符串编写yara如下：

|  |
| --- |
| rule RukeforLab10\_01exe {  meta:  description = "Lab10-01.exe"  strings:  $s1 = "C:\\Windows\\System32\\Lab10-01.sys" fullword ascii  $s2 = "Hello World!" fullword wide  $s3 = "RegWriterApp Version 1.0" fullword wide  $s4 = "REGWRITERAPP" fullword wide  $s5 = "RegWriterApp" fullword wide  $s6 = "System" fullword wide  $s7 = "Copyright (C) 2011" fullword wide  condition:  uint16(0) == 0x5a4d and  uint32(uint32(0x3c))==0x00004550and filesize < 80KB and  all of them  } |
| rule RukeforLab10\_01sys {  meta:  description = "Lab10-01.sys"  strings:  $s1 = "c:\\winddk\\7600.16385.1\\src\\general\\regwriter\\wdm\\sys\\objfre\_wxp\_x86\\i386\\sioctl.pdb" fullword ascii  $s2 = "Lab10-01.sys" fullword wide  $s3 = "Important System Driver" fullword wide  $s4 = "\\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\WindowsFirewall" fullword wide  $s5 = "\\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft" fullword wide  $s6 = "6.1.7600.16385 built by: WinDDK" fullword wide  $s7 = " ABC Corp." fullword wide  condition:  uint16(0) == 0x5a4d and  uint32(uint32(0x3c))==0x00004550and filesize < 10KB and  all of them  } |
| rule RukeforLab10\_02 {  meta:  description = "Lab10-02.exe"  strings:  $s1 = "C:\\Windows\\System32\\Mlwx486.sys" fullword ascii  $s2 = "c:\\winddk\\7600.16385.1\\src\\general\\rootkit\\wdm\\sys\\objfre\_wxp\_x86\\i386\\sioctl.pdb" fullword ascii  $s3 = "SIOCTL.sys" fullword wide  $s4 = "Failed to open service manager." fullword ascii  $s5 = "Failed to start service." fullword ascii  $s6 = "Sample IOCTL Driver" fullword wide  $s7 = "\"WWShtT@" fullword ascii  $s8 = "VWuBhhT@" fullword ascii  $s9 = "486 WS Driver" fullword ascii  $s10 = "6.1.7600.16385 built by: WinDDK" fullword wide  $s11 = "KeServiceDescriptorTable" fullword wide  $s16 = "Failed to create service." fullword ascii  condition:  uint16(0) == 0x5a4d and  uint32(uint32(0x3c))==0x00004550and filesize < 100KB and  6 of them  } |
| rule RukeforLab10\_03exe {  meta:  description = "Lab10-03.exe"  strings:  $s1 = "C:\\Windows\\System32\\Lab10-03.sys" fullword ascii  $s2 = "http://www.malwareanalysisbook.com/ad.html" fullword wide  $s3 = "Process Helper" fullword ascii  $s4 = "\\\\.\\ProcHelper" fullword ascii  condition:  uint16(0) == 0x5a4d and  uint32(uint32(0x3c))==0x00004550and filesize < 70KB and  all of them  } |
| rule RukeforLab10\_03sys {  meta:  description = "Lab10-03.sys"  strings:  $s1 = "c:\\winddk\\7600.16385.1\\src\\general\\rootkitprochide\\wdm\\sys\\objfre\_wxp\_x86\\i386\\sioctl.pdb" fullword ascii  $s2 = "Lab10-03.sys" fullword wide  $s3 = "Important Process Helper" fullword wide  $s4 = "\\DosDevices\\ProcHelper" fullword wide  $s5 = "\\Device\\ProcHelper" fullword wide  $s6 = "6.1.7600.16385 built by: WinDDK" fullword wide  $s7 = " ABC Corp." fullword wide  condition:  uint16(0) == 0x5a4d and  uint32(uint32(0x3c))==0x00004550and filesize < 10KB and  all of them  } |



1. **IDA Python**  
   遍历所有函数并显示每个函数调用的目标函数：

|  |
| --- |
| # -\*- coding: utf-8 -\*-  import idc  import idautils  def analyze\_call\_graph(start\_address):  # 创建函数调用图的字典  call\_graph = {}  # 遍历所有函数  for func\_ea in idautils.Functions():  func\_name = idc.GetFunctionName(func\_ea)  call\_graph[func\_name] = []  # 获取函数信息  func\_start = idc.GetFunctionAttr(func\_ea, idc.FUNCATTR\_START)  func\_end = idc.GetFunctionAttr(func\_ea, idc.FUNCATTR\_END)  # 遍历函数内的指令  for head in idautils.Heads(func\_start, func\_end):  # 获取指令的助记符  mnemonic = idc.GetMnem(head)  # 检查指令是否为"call"  if mnemonic == "call":  target\_address = idc.GetOperandValue(head, 0)  target\_func\_name = idc.GetFunctionName(target\_address)  # 如果目标函数不为空，则将其添加到调用图中  if target\_func\_name:  call\_graph[func\_name].append(target\_func\_name)  # 显示调用图  for func, calls in call\_graph.items():  print("Function: {}".format(func))  if calls:  print("Calls the following functions:")  for called\_func in calls:  print(" {}".format(called\_func))  else:  print("This function does not call other functions.")  print("=" \* 30)  # 在此处替换为要分析的起始地址  start\_address\_to\_analyze = 0x401000 # 替换为实际地址  # 运行分析程序  analyze\_call\_graph(start\_address\_to\_analyze) |



1. **实验结论及心得体会**

本次实验深入了解了恶意软件的行为分析方法，从用户态和内核态两个层面对其进行了全面的分析。通过使用多种工具和调试器，掌握了分析恶意软件的基本技能，包括查看导入函数表、资源、监视系统调用等。同时，了解了恶意软件的一些隐匿手法，如内核态的进程隐藏。这次实验对安全分析和逆向工程能力提升有很大的帮助。