

南開大學

恶意代码分析与防治课程实验报告

实验 8-2: r77 rootkit



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

一、实验目的

首先, SSDT 是一个包含系统服务描述符的内核数据结构, 它存储了操作系统内核中各种系统服务的入口点。这些系统服务包括文件操作、进程管理、网络通信等。

Rootkit 是一种恶意软件, 其目的是隐藏其存在并操纵操作系统以达到非法目的。

Rootkit 通常会修改 SSDT 以劫持系统服务的调用, 以便掩盖和操作其行为。

为了寻找 Rootkit, 我们需要进行内核分析。这包括获取操作系统的内核代码、数据结构以及内核模块的信息。然后, 我们可以通过分析 SSDT 来检测潜在的 Rootkit。

以下是通常用于这种分析的步骤:

获取 SSDT: 我们可以使用特权访问或内核调试工具来获取系统的 SSDT。这些工具使我们能够读取内核内存并提取 SSDT 的内容。

比较标准 SSDT: 我们需要获取操作系统的正常状态下的标准 SSDT。这可以通过从可信源获得的操作系统映像中提取 SSDT 来实现。然后, 我们可以将这个标准 SSDT 与当前系统的 SSDT 进行比较。

检测差异: 对比标准 SSDT 和当前 SSDT, 我们可以找出其中的差异。这些差异可能是 Rootkit 所做的修改。这些修改可能包括将某些系统服务的入口点指向 Rootkit 的代码而不是操作系统的原始代码。

分析修改: 一旦发现差异, 我们可以继续分析 Rootkit 的修改以了解其行为。这可能涉及追踪 Rootkit 修改的入口点, 并分析其替换的代码。

需要注意的是, 这只是一种寻找 Rootkit 并分析内核的基本方法之一。Rootkit 的设计者会使用各种技术来隐藏其存在和修改。在实际情况中, 对抗 Rootkit 需要更加复杂和高级的技术和工具。

二、实验原理

r77-Rootkit 是一款功能强大的无文件 Ring 3 Rootkit, 并且带有完整的安全工具和持久化机制, 可以实现进程、文件和网络连接等操作及任务的隐藏。

r77 能够在所有进程中隐藏下列实体:

文件、目录、连接、命名管道、计划任务;

进程;

CPU 用量;

注册表键&值;

服务;

TCP&UDP 连接;

该工具兼容 32 位和 64 位版本的 Windows 7 以及 Windows 10。

2.1 Windows 的 Detours 机制

Windows 的 Detours 机制是一种软件技术，用于拦截和修改 Win32 函数的调用。这是通过 API 钩子（hooking）实现的，是一种编程技术，允许开发者拦截对动态链接库（DLL）中函数的调用。Detours 通过动态地重写目标函数的机器代码来工作。这种方法通常用于系统监控、程序调试、性能分析等方面。

在 Detours 机制中，当一个程序调用某个 Win32 API 函数时，Detours 可以劫持这个调用，并将它重定向到另一个函数。这个替代函数可以是用户自定义的，也可以是对原函数的扩展或修改。

利用这种技术，开发者可以在不修改原始代码的情况下，动态地更改程序的行为。

例如，Detours 可以用于监视和记录文件操作、网络通信或系统调用。它也被用于创建安全工具，如防病毒软件，这些软件需要监视系统级活动以检测恶意行为。然而，同样的技术也可以被用于恶意软件中，用于隐藏其行为，例如隐藏文件、进程或网络连接，这就是你提到的 R77 程序所做的事情。

2.2 API Hooking

API（应用程序编程接口）Hooking 是一种编程技术，用于改变或增强操作系统或应用程序的功能。在 Windows 系统中，API 函数通常来自各种动态链接库（DLL）。

2.2.1 原理

拦截调用：API Hooking 工作原理是拦截对特定 API 函数的调用。这可以通过多种方式实现，包括修改函数的入口地址（例如，在导入地址表中），或者直接修改函数代码本身（如通过 Inline Hooking）。

重定向调用：当 API 调用被拦截后，它会被重定向到一个自定义函数。这个自定义函数可以在调用原函数之前或之后执行额外的代码，甚至完全替代原函数。

2.2.2 应用

调试和监控：开发人员利用 API Hooking 来监控和记录应用程序的行为，例如文件访问、网络通信等。

安全软件：安全软件（如防病毒程序）使用 API Hooking 来检测和阻止恶意行为。

系统增强：通过 API Hooking，软件可以添加或修改操作系统功能，不需要修改底层代码。

2.2.3 风险

稳定性问题：不正确的 API Hooking 可能导致系统不稳定。

安全隐患：恶意软件也可能利用 API Hooking 来隐藏其行为或损害系统。

2.3 隐藏进程

隐藏进程是一种技术，通常用于防止进程在常规工具（如任务管理器）中被检测到。这在恶意软件和某些类型的系统监控软件中很常见。

2.3.1 实现方法

修改系统结构：通过修改操作系统的内部数据结构（如进程列表）来隐藏进程。

拦截系统调用：使用 API Hooking 或类似技术拦截和修改系统调用，例如拦截列出进程的 API 调用，并从结果中删除特定进程。

内核模式驱动：在内核模式下运行的驱动程序可以直接访问和修改操作系统的核心数据结构，进而隐藏进程。

2.3.2 应用

安全和隐私：某些合法软件可能出于安全或隐私原因隐藏其进程。

恶意软件：病毒、木马和 rootkits 经常隐藏其进程以避免检测。

2.3.3 风险和挑战

安全风险：隐藏进程的技术可以被恶意软件利用，对用户和企业造成严重威胁。
检测难度：隐藏进程的技术提高了安全软件检测和清除这些威胁的难度。

三、实验过程

1. 在计算机网络当中选择相应的 r77 rootkit 安装包并且下载。

r77Rootkit 1.5.0	2023/11/17 11:26	文件夹
r77-rootkit-master	2023/10/3 19:25	文件夹

安装包如下所示：

其中 install.exe

Examples	2023/8/29 3:11	文件夹	
BytecodeApi.dll	2022/10/14 22:16	应用程序扩展	318 KB
BytecodeApi.UI.dll	2022/10/14 22:16	应用程序扩展	77 KB
Helper32.dll	2023/8/29 3:10	应用程序扩展	9 KB
Helper64.dll	2023/8/29 3:10	应用程序扩展	11 KB
Install.exe	2023/8/29 3:10	应用程序	162 KB
Install.shellcode	2023/8/29 3:10	SHELLCODE 文件	163 KB
LICENSE.txt	2023/6/7 4:21	文本文档	2 KB
r77-x64.dll	2023/8/29 3:10	应用程序扩展	143 KB
r77-x86.dll	2023/8/29 3:10	应用程序扩展	108 KB
TestConsole.exe	2023/8/29 3:10	应用程序	263 KB
Uninstall.exe	2023/8/29 3:10	应用程序	13 KB

r77 可以直接使用单独的 “Install.exe” 进行安装，安装工具会将 r77 服务在用户登录之前开启，后台进程会向所有当前正在运行以及后续生成的进程中注入命令。这里需要使用两个进程来分别注入 32 位和 64 位进程，这两个进程都可以使用配置系统和 PID 来进行隐藏。

“Uninstall.exe” 程序负责将 r77 从系统中卸载掉，并解除 Rootkit 跟所有进程的绑定关系。

安装程序为 32 位和 64 位 r77 服务创建两个计划任务。计划任务确实需要存储名为 \$77svc32.job 和 \$77svc64.job 的文件，这是无文件概念的唯一例外。但是，一旦 Rootkit 运行，计划任务也会通过前缀隐藏。

计划任务将使用下列命令开启 “powershell.exe”：

```
[Reflection.Assembly]::Load([Microsoft.Win32.Registry]::LocalMachine.OpenSubkey('SOFTWARE').GetValue('$77stager')).EntryPoint.Invoke($Null,$Null)
```

该命令是内联的，不需要.ps1 脚本。这里，使用 PowerShell 的.NET Framework 功能从注册表加载 C#可执行文件并在内存中执行。由于命令行的最大长度为 260 (MAX_PATH)，因此只有足够的空间执行简单的 Assembly.Load().EntryPoint.Invoke()。

执行的 C#代码为 stager，它将会使用 Process Hollowing 技术创建 r77 服务进程。r77 服务是一个本地可执行文件，分别以 32 位和 64 位架构继续编译。父进程被设置为了 winlogon.exe 以增加欺骗性（模糊性）。另外，这两个进程被 ID 隐藏，在任务管理器中不可见。

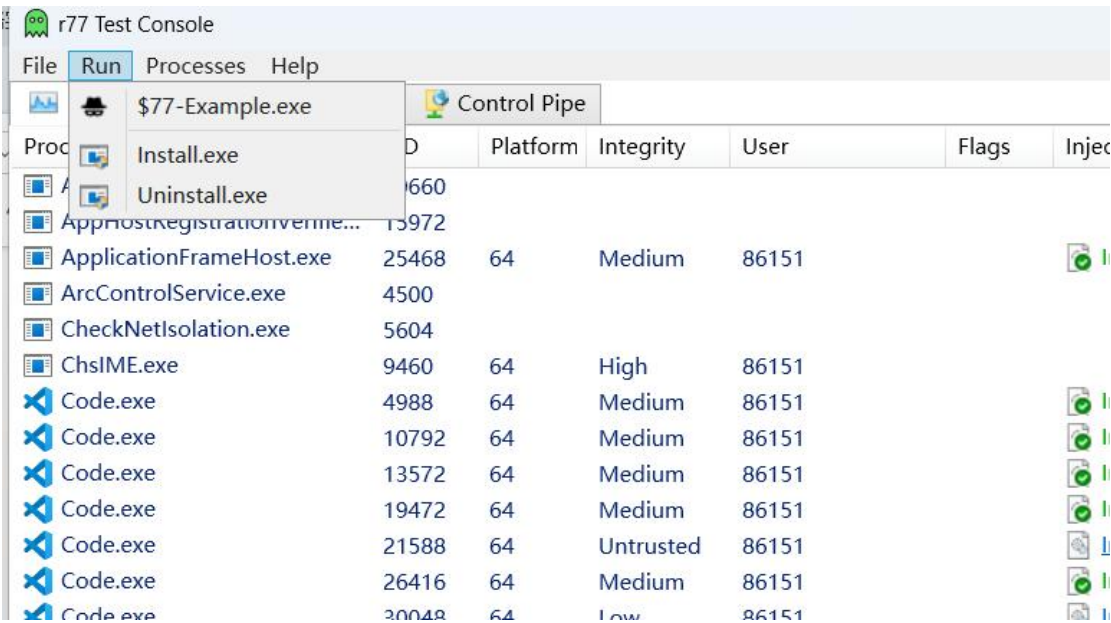
测试控制台可以用来向单独进程注入 r77，或接触进程跟 Rootkit 的绑定关系

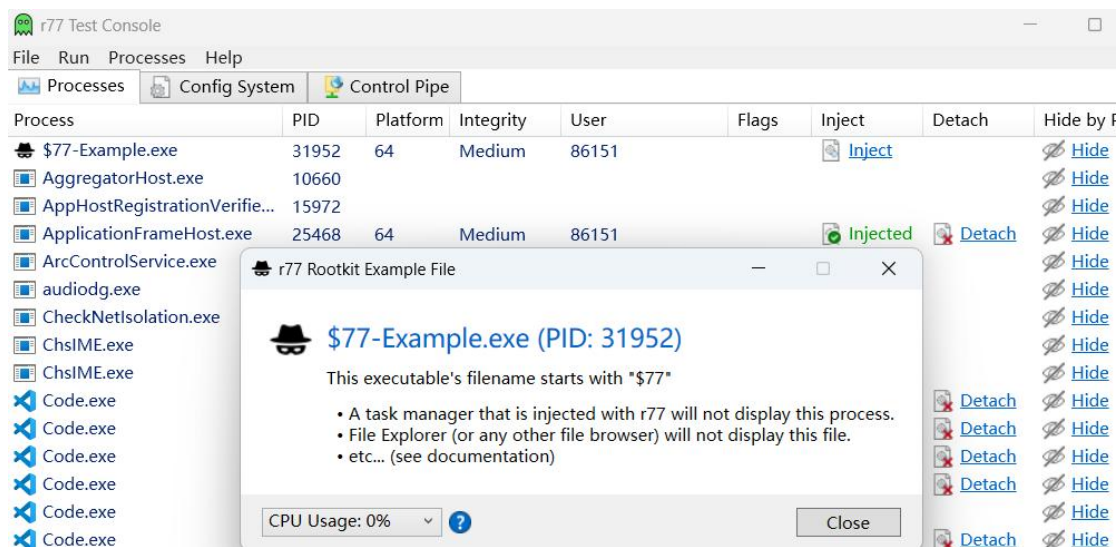
进程隐藏

R77 的进程隐藏功能是一种用于使特定进程在操作系统的常规监视工具（如任务管理器）中不可见的技术。这种隐藏技术主要基于修改操作系统的行为来阻止对特定进程的检测。

示例进程 R77 为我们提供了一个可执行文件 \$77-Example.exe，这个文件以\$77 为开头。

然后我们运行它：





这里我们设置高 cpu 占有率，可以在任务管理器里看到改程序占据大量进程，并且电脑的风扇正在高速运转。

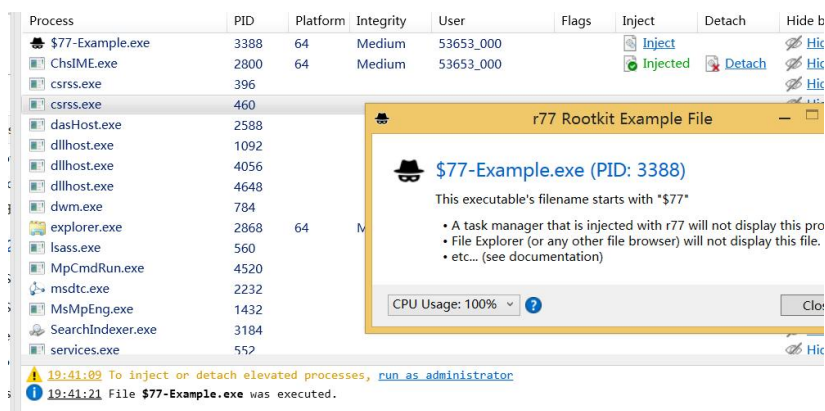
任意进程隐藏：此时运行 Install.exe ，重新打开任务管理器，同时保持这个 exe 运行，发现找不到这个进程了：

名称	状态	CPU	内存	磁盘	网络
应用 (3)					
r77 Test Console		0%	84.6 MB	0 MB/秒	0 Mbps
Windows 资源管理器		0%	41.8 MB	0 MB/秒	0 Mbps
r77Rootkit 1.5.0					
任务管理器		0.7%	11.9 MB	0 MB/秒	0 Mbps

仍然没有找到这个进程，证明其确实被隐藏了。

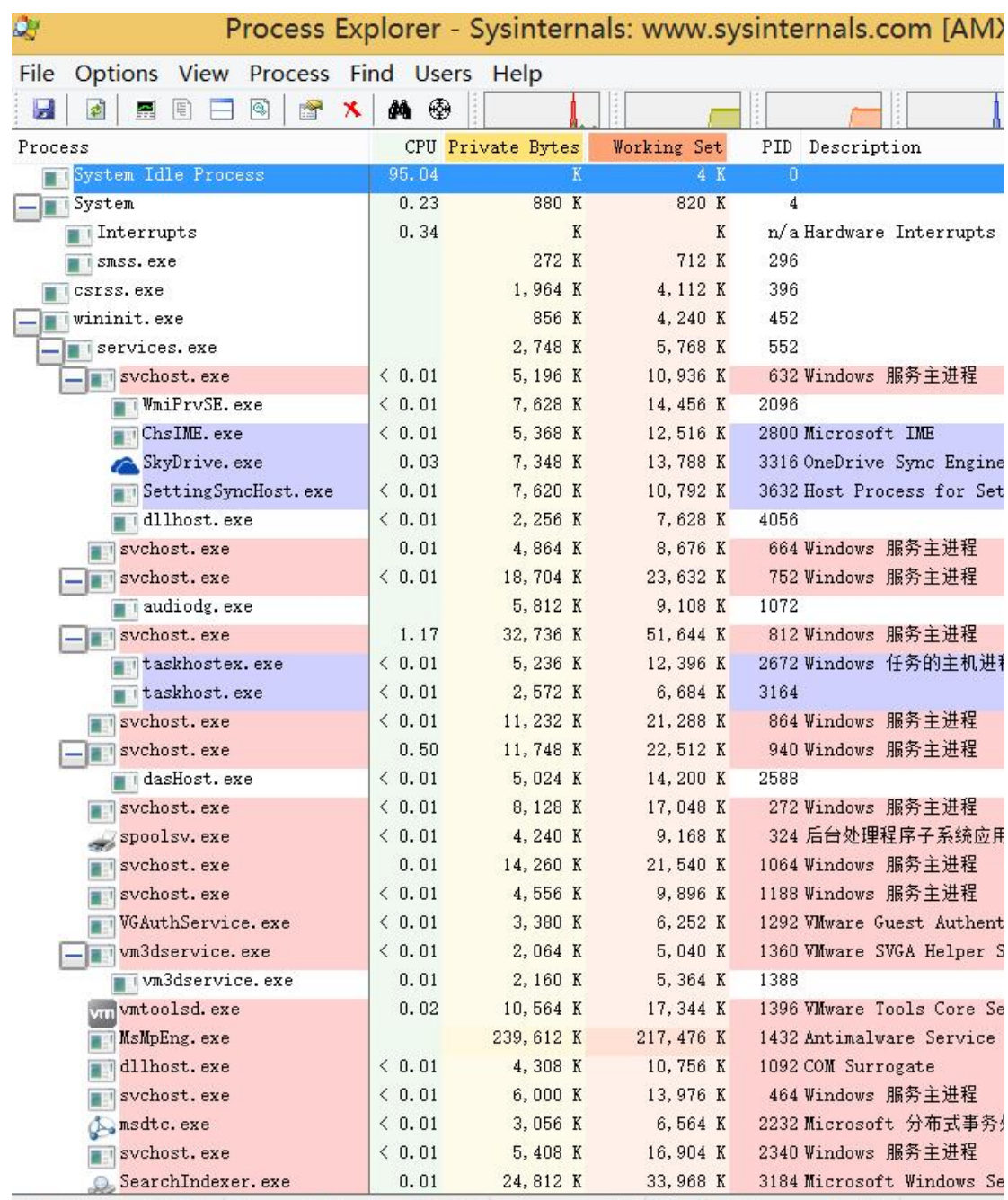
实际上，R77 默认隐藏了\$77 开头的进程，但其他进程也可以进行针对性隐藏。

我们运行 TestConsole.exe ，可以看到进程列表，包括被隐藏的进程：



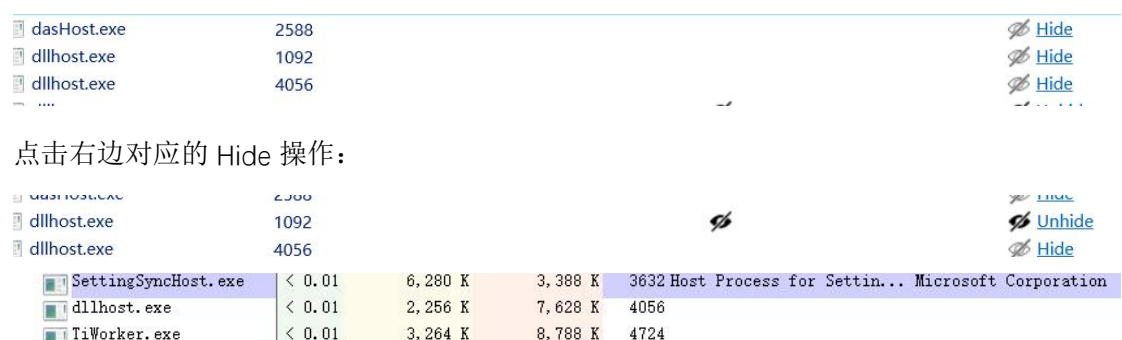
在上图，为了证明这个程序仍然在运行，只是被隐藏了，我将 CPU 占用率调为 100%。可以看到尽管 CPU 占用已经满了，但并没有显示这个进程的存在。

我们打开 Process Explorer 来查看进程：



Process	CPU	Private Bytes	Working Set	PID	Description
System Idle Process	95.04	K	4 K	0	
System	0.23	880 K	820 K	4	
Interrupts	0.34	K	K	n/a	Hardware Interrupts
smss.exe		272 K	712 K	296	
csrss.exe		1,964 K	4,112 K	396	
wininit.exe		856 K	4,240 K	452	
services.exe		2,748 K	5,768 K	552	
svchost.exe	< 0.01	5,196 K	10,936 K	632	Windows 服务主进程
WmiPrvSE.exe	< 0.01	7,628 K	14,456 K	2096	
ChsIME.exe	< 0.01	5,368 K	12,516 K	2800	Microsoft IME
SkyDrive.exe	0.03	7,348 K	13,788 K	3316	OneDrive Sync Engine
SettingSyncHost.exe	< 0.01	7,620 K	10,792 K	3632	Host Process for Set
dllhost.exe	< 0.01	2,256 K	7,628 K	4056	
svchost.exe	0.01	4,864 K	8,676 K	664	Windows 服务主进程
svchost.exe	< 0.01	18,704 K	23,632 K	752	Windows 服务主进程
audiodg.exe		5,812 K	9,108 K	1072	
svchost.exe	1.17	32,736 K	51,644 K	812	Windows 服务主进程
taskhost.exe	< 0.01	5,236 K	12,396 K	2672	Windows 任务的主机进程
taskhost.exe	< 0.01	2,572 K	6,684 K	3164	
svchost.exe	< 0.01	11,232 K	21,288 K	864	Windows 服务主进程
svchost.exe	0.50	11,748 K	22,512 K	940	Windows 服务主进程
dasHost.exe	< 0.01	5,024 K	14,200 K	2588	
svchost.exe	< 0.01	8,128 K	17,048 K	272	Windows 服务主进程
spoolsv.exe	< 0.01	4,240 K	9,168 K	324	后台处理程序子系统应用
svchost.exe	0.01	14,260 K	21,540 K	1064	Windows 服务主进程
svchost.exe	< 0.01	4,556 K	9,896 K	1188	Windows 服务主进程
VGAuthService.exe	< 0.01	3,380 K	6,252 K	1292	VMware Guest Authent
vm3dservice.exe	< 0.01	2,064 K	5,040 K	1360	VMware SVGA Helper S
vm3dservice.exe	0.01	2,160 K	5,364 K	1388	
vmtoolsd.exe	0.02	10,564 K	17,344 K	1396	VMware Tools Core Se
MsMpEng.exe		239,612 K	217,476 K	1432	Antimalware Service
dllhost.exe	< 0.01	4,308 K	10,756 K	1092	COM Surrogate
svchost.exe	< 0.01	6,000 K	13,976 K	464	Windows 服务主进程
msdtc.exe	< 0.01	3,056 K	6,564 K	2232	Microsoft 分布式事务
svchost.exe	< 0.01	5,408 K	16,904 K	2340	Windows 服务主进程
SearchIndexer.exe	0.01	24,812 K	33,968 K	3184	Microsoft Windows Se

随便再找一个任意进程名的进程，选取下面的进程：



dasHost.exe	2588	Hide
dllhost.exe	1092	Hide
dllhost.exe	4056	Hide

SettingSyncHost.exe	< 0.01	6,280 K	3,388 K	3632	Host Process for Settin... Microsoft Corporation
dllhost.exe	< 0.01	2,256 K	7,628 K	4056	
TiWorker.exe	< 0.01	3,264 K	8,788 K	4724	

再次打开任务管理器查看，该进程已经“消失”，同时右边的 Hide 选项也已经变成了 Unhide:

4.3 文件隐藏

R77 的文件隐藏功能是一种技术，用于使特定文件或目录在操作系统的标准文件浏览工具（例如 Windows 资源管理器）中不可见。这通常是通过拦截和修改系统级别的文件系统调用来实现的。

先执行 uninstall:



在一个文件夹里面放入我想要隐藏的文件和文件夹:

然后执行 Install 操作，刷新文件夹:



显示“此文件夹为空”，文件已经被隐藏成功。

回到 lab10-02.exe


这里需要我们寻找 lab10-02.exe 当中提到的 mlwx486.sys

这里详细分析我们如何从 SSDT 当中查找到了相应的函数

查看 SSDT 的修改项

使用 `!m` 命令列举加载的所有模块，可以看到 `MLwx486` 已经被加载到内存中了，但是并没有在硬盘上显示，说明这可能是一个 Rootkit。

10D2E000	10D2E1400	vmusdmouse	(deferred)
f8bf0000	f8bf1100	dump_WMILIB	(deferred)
f8cc4000	f8cc4c00	audstub	(deferred)
f8cf3000	f8cf3d00	dxgthk	(deferred)
f8d24000	f8d24d80	Mlwx486	(deferred)
f8dc3000	f8dc3b80	Null	(deferred)



接下来，使用命令 `dd dwo(KeServiceDescriptorTable) L100` 检查 SSDT 的修改情况，可以看到其中有一条 记录在 `Mlwx486.sys` 的范围内。

其中这个地址可能是我们需要寻找的目标函数所在的地址。

30502d8c	805c2522	805e4a1a	805e47d0	80b0e1b0
30502d9c	8063cc78	805c0346	805eedce	805eaa16
30502dac	805eac02	805aea08	806062dc	8056d0ce
30502dbc	8060db50	8053d02e	80607e68	
30502dcc	80608ac8	f8d24486	805b4de0	805703ca
30502ddc	806063a4	8060d2dc	80570c46	
30502dec	805ccee0	8059b6fc	805c3bfc	805c27c8
30502dfc	805e4afa	80608266	8060f060	8056edda
30502e0c	b1f57300	8061a3d4	8060e93e	805bc04c

使用命令 `u f8d24486` 查看该地址，这是一个 `Mlwx486` 里面自带的函数，并不能提供什么有效信息。

```
*** ERROR: Module load completed but symbols could not be loaded for Mlwx486.sys
Mlwx486+0x468:
f8dd6468 0000      add     byte ptr [eax],al
f8dd646a 0000      add     byte ptr [eax],al
f8dd646c 0000      add     byte ptr [eax],al
f8dd646e 0000      add     byte ptr [eax],al
f8dd6470 0000      add     byte ptr [eax],al
f8dd6472 0000      add     byte ptr [eax],al
f8dd6474 0000      add     byte ptr [eax],al
f8dd6476 0000      add     byte ptr [eax],al
```

于是通过快照将虚拟机恢复到安装 Rootkit 之前的状态，再次使用命令 `dd dwo(KeServiceDescriptorTable) L100`，找到修改前的地址为 `80570074`，于是使用命令 `u 80570074` 查看被覆盖的是哪个函数，是 `NtQueryDirectoryFile`，这是一个提取文件和目录信息的通用函数。Rootkit 可以通过挂钩这个函数来隐藏文件，下面我们开始分析这个函数做了什么。

```

kd> u 80570074
nt!NtQueryDirectoryFile:
80570074 8bff          mov     edi,edi
80570076 55              push    ebp
80570077 8bec          mov     ebp,esp
80570079 8d452c        lea     eax,[ebp+2Ch]
8057007c 50            push    eax
8057007d 8d4528        lea     eax,[ebp+28h]
80570080 50            push    eax
80570081 8d4524        lea     eax,[ebp+24h]

```

我们现在这里设置一个断点 bp f8d24486，然后回到虚拟机中运行程序，命中断点后又回到 WinDbg 开始单步调试。（下面由于重启过虚拟机，改变了地址）

这里不断的使用 p 查找附近的汇编代码，最终找到计算的偏移量。

```

kd> p
Mlwx486+0x490:
f8cd6490 ff7530          push    dword ptr [ebp+30h]
kd> p
Mlwx486+0x493:
f8cd6493 ff752c          push    dword ptr [ebp+2Ch]
kd> p
Mlwx486+0x496:
f8cd6496 ff7528          push    dword ptr [ebp+28h]
kd> p
Mlwx486+0x499:
f8cd6499 ff7524          push    dword ptr [ebp+24h]
kd> p
Mlwx486+0x49c:
f8cd649c ff7520          push    dword ptr [ebp+20h]
kd> p
Mlwx486+0x49f:
f8cd649f 56              push    esi
kd> p
Mlwx486+0x4a0:
f8cd64a0 ff7518          push    dword ptr [ebp+18h]
kd> p
Mlwx486+0x4a3:
f8cd64a3 ff7514          push    dword ptr [ebp+14h]
kd> p
Mlwx486+0x4a6:
f8cd64a6 ff7510          push    dword ptr [ebp+10h]
kd> p
Mlwx486+0x4a9:
f8cd64a9 ff750c          push    dword ptr [ebp+0Ch]
kd>
Mlwx486+0x4ac:
f8cd64ac ff7508          push    dword ptr [ebp+8]
kd> p
Mlwx486+0x4af:
f8cd64af e860000000      call    Mlwx486+0x514 (f8cd6514)

```

到这里，调用了函数 `Mlwx486+0x514`，它的第 8 个参数 `FileInformationClass` 的值是 3，如果不是 3，便会返回 `NtQueryDirectoryFile` 的原始返回值。此外，挂钩函数也会检查 `NtQueryDirectoryFile` 的返回值与第 9 个参数值 `ReturnSingleEntry`。当参数不符合要求时，它的功能会与原始函数一样，只有当参数满足要求时才会修改返回值。接下来我们使用命令 `bp f8cd6486 ".if dwo(esp+0x24)==0 {} .else {gc}"` 在挂钩函数上设置一个条件断点，仅当参数满足 `ReturnSingleEntry` 为 0 时这个断点才被触发。然后使用命令 `dir C:\WINDOWS\system32` 来列出文件夹 `C:\WINDOWS\system32` 这个文件夹下面的所有文件和文件夹，发现这个条件断点被命中了。之后，它调用了 `0xf8d9e590` 处的函数，IDA 将其标注为 `RtlCompareMemory`，查看它的参数，它要将 `eax` 要和字符串 "Mlwx" 进行比较，比较的最大长度为 8。而 `eax` 中存储的是 `[esi+5Eh]`，在前面我们已经看到 `esi` 是 `FileInformation`，而且这个值是为 3，具体意义是 `FileBothDirectoryInformation`，偏移量 `0x5E` 是结构 `FileName` 的起始地址，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"

```
kd> p
Mlwx486+0x4ca:
f8d9e4ca 6a08          push     8
kd> p
Mlwx486+0x4cc:
f8d9e4cc 681ae5d9f8     push     offset Mlwx486+0x51a (f8d9e51a)
kd> p
Mlwx486+0x4d1:
f8d9e4d1 8d465e         lea      eax,[esi+5Eh]
kd> p
Mlwx486+0x4d4:
f8d9e4d4 50            push     eax
kd> p
Mlwx486+0x4d5:
f8d9e4d5 32db         xor      bl,bl
kd> p
Mlwx486+0x4d7:
f8d9e4d7 ff1590e5d9f8   call     dword ptr [Mlwx486+0x590 (f8d9e590)]
kd> p
Mlwx486+0x4dd:
f8d9e4dd 83f808        cmp      eax,8
kd> p
Mlwx486+0x4e0:
f8d9e4e0 7512         jne      Mlwx486+0x4f4 (f8d9e4f4)
```

使用命令 `db esi+5eh` 查看存储在 `esi+5eh` 的内容，这里是 `xs.l`。如果传入的 `FileName` 和 `Mlwx` 不相等，函数直接就退出了。

现在我们要分析它是如何修改 `NtQueryDirectoryFile` 的返回值然后隐藏 `Mlwx486.sys` 文件的。`NtQueryDirectoryFile` 的返回值 `FILE_BOTH_DIR_INFORMATION` 结构是由一系列 `FILE_BOTH_DIR_INFORMATION` 结构串联而成的，如果 `RtlCompareMemory` 返回值是 8 的话，即找到了要隐藏的 "Mlwx" 开头的文件，就会通过两次指令操作把指针往后移动，抹除了 `Mlwx486.sys` 文件的 `FILE_BOTH_DIR_INFORMATION` 结构，达到隐藏文件的目的。

而后再次运行 `kd> dps nt!KiServiceTable | 100`，能够看出服务描述符表已被修改

```

00001dc0 0000cb50 nt!NtSetBootEntryOrder
80501dc0 8060cb50 nt!NtSetBootEntryOrder
80501dc4 8053c02e nt!NtQueryDebugFilterState
80501dc8 80606e68 nt!NtQueryDefaultLocale
80501dcc 80607ac8 nt!NtQueryDefaultUILanguage
80501dd0 baecb486 Mlwx486+0x486
80501dd4 805b3de0 nt!NtQueryDirectoryObject
80501dd8 8056f3ca nt!NtQueryEaFile
80501ddc 806053a4 nt!NtQueryEvent
80501de0 8056c222 nt!NtQueryFullAttributesFile
80501de4 8060c2dc nt!NtQueryInformationAtom
80501de8 8056fc46 nt!NtQueryInformationFile
80501dec 805cbee0 nt!NtQueryInformationJobObject
80501df0 8059a6fc nt!NtQueryInformationPort
80501df4 805c2bfc nt!NtQueryInformationProcess
80501df8 805c17c8 nt!NtQueryInformationThread
80501dfc 805e3afa nt!NtQueryInformationToken
kd>

```

我们连接上 WinDbg 进行内核调试。

首先使用命令!devobj ProcHelper，查看到这个设备存储的位置是 81d22f38。

```

kd> !devobj ProcHelper
Device object (81d40bf8) is for:
ProcHelper*** ERROR: Module load completed but symbols could not be loaded for Lab10-03.sys
\Driver\Process Helper DriverObject 81d22f38
Current Irp 00000000 RefCount 1 Type 00000022 Flags 00000040
Dacl e130ebec DevExt 00000000 DevObjExt 81d40cb0
ExtensionFlags (0000000000)
Device queue is not busy.

```

然后使用命令 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，我们刚刚在 IDA 分析的时候看到过这个函数。

```

kd> dt nt!_DRIVER_OBJECT 81d22f38
+0x000 Type : 0n4
+0x002 Size : 0n168
+0x004 DeviceObject : 0x81d40bf8 _DEVICE_OBJECT
+0x008 Flags : 0x12
+0x00c DriverStart : 0xf8cdd000 Void
+0x010 DriverSize : 0xe00
+0x014 DriverSection : 0x82095a40 Void
+0x018 DriverExtension : 0x81d22fe0 _DRIVER_EXTENSION
+0x01c DriverName : _UNICODE_STRING "\Driver\Process Helper"
+0x024 HardwareDatabase : 0x80671ae0 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM'
+0x028 FastIoDispatch : (null)
+0x02c DriverInit : 0xf8cdd7cd long +0
+0x030 DriverStartIo : (null)
+0x034 DriverUnload : 0xf8cdd62a void +0
+0x038 MajorFunction : [28] 0xf8cdd606 long +0

```

接下来，我们使用命令 dd 81d22f38+0x38 L1c 命令查看主函数表中表项：

```
kd> dd 81d22f38+0x38 L1c
81d22f70 f8cdd606 804f454a f8cdd606 804f454a
81d22f80 804f454a 804f454a 804f454a 804f454a
81d22f90 804f454a 804f454a 804f454a 804f454a
81d22fa0 804f454a 804f454a f8cdd666 804f454a
81d22fb0 804f454a 804f454a 804f454a 804f454a
81d22fc0 804f454a 804f454a 804f454a 804f454a
81d22fd0 804f454a 804f454a 804f454a 804f454a
```

可以看到表中大多数函数都是 804f454a，我们使用命令 `ln 804f454a` 看看这个函数做了什么，可以看到，这个函数被命名为 `IopInvalidDeviceRequest`，从字面意思看是处理驱动无法处理的非法请求的。

```
kd> ln 804f454a
(804f454a) nt!IopInvalidDeviceRequest | (804f4580) nt!IopGetDeviceAttachmentBase
Exact matches:
nt!IopInvalidDeviceRequest = <no type information>
```

此外，主函数表中还有三个不同于 804f454a 的函数，分别是 f8cdd606、f8cddd606 和 f8cdd666，它们在 896587e0+0x38 处的偏移量分别是 0，2 和 0xe，对应着 Create、Close 和 DeviceIoControl。进入这两个（前两个相同）位置查看具体的代码。在 f8cdd606 处，看到它只调用了函数，指向的是 804ef1b0 这个地址，使用指令 `ln 804ef1b0`，看到这个函数是 `IofCompleteRequest`，用于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，意味着告诉操作系统请求这个驱动成功了，但是没做什么其他的操作。

```
kd> u f8cdd606
Lab10_03+0x606:
f8cdd606 8bff          mov     edi,edi
f8cdd608 55            push    ebp
f8cdd609 8bec          mov     ebp,esp
f8cdd60b 8b4d0c          mov     ecx,dword ptr [ebp+0Ch]
f8cdd60e 83611800        and     dword ptr [ecx+18h],0
f8cdd612 83611c00        and     dword ptr [ecx+1Ch],0
f8cdd616 32d2           xor     dl,dl
f8cdd618 ff1580d4cdf8    call    dword ptr [Lab10_03+0x480 (f8cdd480)]
```

在 f8cdd666 处看到该函数在初始化栈操作之后，首先调用了函数 `[Lab10_03+0x490]` 的指针函数，指向的是 804ef608 这个地址，使用指令 `ln 804ef1b0`，看到这个函数是 `nt!IoGetCurrentProcess`，用于返回一个指向当前进程的指针。也就是这个代码会获得一个当前进程的指针，操纵当前进程的 PEB。


```

kd> u f8cdd666 L10
Lab10_03+0x666:
f8cdd666 8bff          mov     edi,edi
f8cdd668 55            push   ebp
f8cdd669 8bec          mov     ebp,esp
f8cdd66b ff1590d4cdf8  call   dword ptr [Lab10_03+0x490 (f8cdd490)]
f8cdd671 8b888c000000  mov     ecx,dword ptr [eax+8Ch]
f8cdd677 0588000000    add     eax,88h
f8cdd67c 8b10          mov     edx,dword ptr [eax]
f8cdd67e 8911          mov     dword ptr [ecx],edx
f8cdd680 8b08          mov     ecx,dword ptr [eax]
f8cdd682 8b4004        mov     eax,dword ptr [eax+4]
f8cdd685 894104        mov     dword ptr [ecx+4],eax
f8cdd688 8b4d0c        mov     ecx,dword ptr [ebp+0Ch]
f8cdd68b 83611800      and     dword ptr [ecx+18h],0
f8cdd68f 83611c00      and     dword ptr [ecx+1Ch],0
f8cdd693 32d2          xor     dl,dl
f8cdd695 ff1580d4cdf8  call   dword ptr [Lab10_03+0x480 (f8cdd480)]

```

之后，这个函数访问偏移量 0x88 处的数据和偏移量 0x8C 处数据。我们使用 dt 命令发现这两处数据是 PEB 结构中的 _LIST_ENTRY。这个结构用于描述双向链接列表中的条目或充当此列表的头部，该程序不仅读取 LIST_ENTRY 结构，还会修改它，通过一系列对指针的操作修改 PEB，使得没有任何的指针将指向他的 LIST_ENTRY 结构，把自己的进程隐藏起来。

```

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
+0x0c4 ObjectTable : Ptr32 _HANDLE_TABLE

```

四、实验结论及心得体会

1. 技术验证：本次实验成功验证了 R77 工具在隐藏进程、文件、注册表项以及 TCP/UDP 连接方面的有效性。通过实际操作，观察到被隐藏的对象在常规工具中不再可见，证实了 R77 在系统级别进行拦截和修改操作的能力。

2. 功能特性：R77 通过拦截系统级调用和修改操

作系统的内部数据结构，实现了对进程、文件和网络活动的隐藏。这一过程不需要修改被隐藏对象的物理状态或数据，显示出高度的隐蔽性和灵活性。

3. 安全和隐私考量：尽管 R77 为系统管理和隐私保护提供了强大工具，但同时也揭示了潜在的安全风险。恶意软件可能利用类似技术进行隐蔽活动，对用户和企业造成威胁。

在 SSDT 对于 rootkit 的研究当中，我们不断的跟踪了其中的一部分环境，并且在环境当中跟随相应的位置找到了 rootkit。

大多数根工具包修改操作系统内核最流行的方法：系统服务描述符表（SSDT）挂钩。

Rootkit 更改了 SSDT 中的值，以便调用 Rootkit 代码，而不是预期的函数。

对于 rootkit 的研究让我们可以寻找程序当中更多的变化，以及 SSDT 的控制原理。