

**恶意代码防治课程实验报告**

**实验八：R77技术分析**

****

学 院 网络空间安全

专 业 信息安全

学 号 2212790

姓 名 贾程皓

班 级 0975

1. **实验目的**

在使用R77的基础上，撰写技术分析，描述使用过程中看到的行为如何技术实现。

1. **实验过程**

**2.1 系统环境**

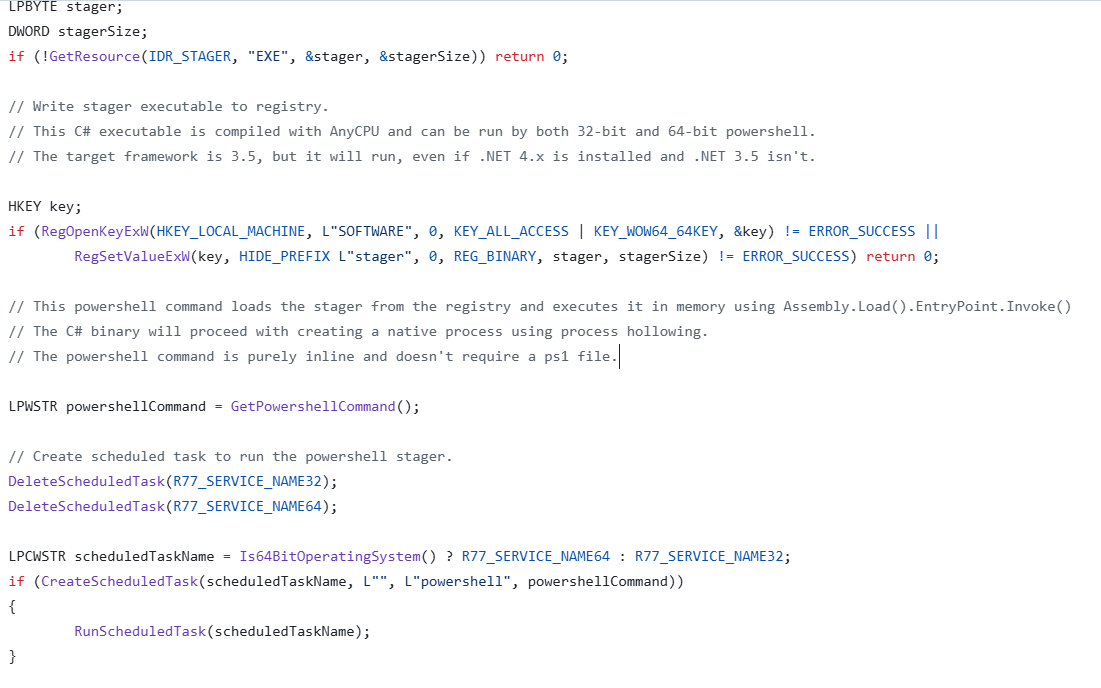
本实验使用Windwos11 专业版下的Sandbox。

**2.2 r77 Header**

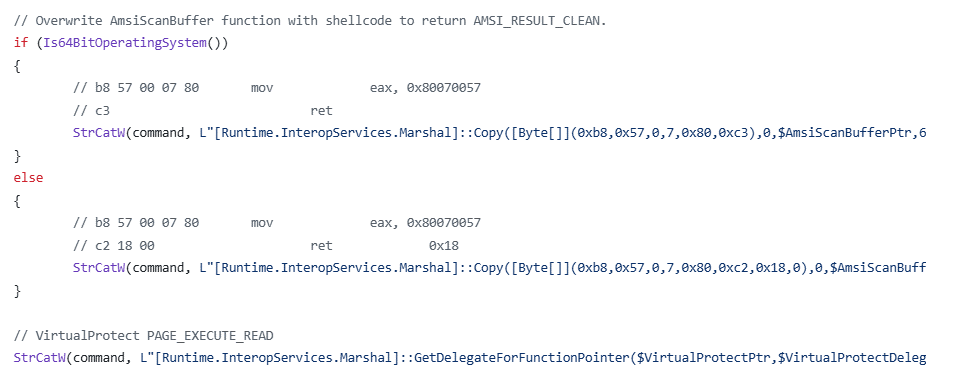
加载到内存中的进程在r77 Console.exe中是怎样判断是否被感染的？

在r77的文档中解释到，恶意代码采用了“r77 header”，即将**Dos中从0x00000040开始的两个字节进行修改**。其中被感染的进程在运行时，会修改它的“R77\_SIGNATURE = 0x7277”；r77服务进程（例如install.exe）在编译时，即修改Dos头部为“R77\_SERVICE\_SIGNATURE = 0x7273”；r77 Console等辅助进程在编译时，也会修改Dos首部为“R77\_HELPER\_SIGNATURE = 0x7268”。

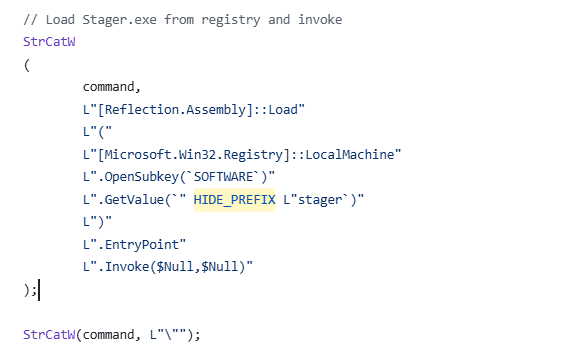
**2.3 Install**



其中，GetResource()函数将资源文件中的**IDR\_STAGER（即stager.exe）写入指针stager里**。RegOpenKey()和RegSetValue()函数共同将注册表下Local\_Machine/Software/r77stager里写入stager.exe的二进制数据。**GetPowershellCommand()函数得到接下来运行的shell指令，需要重点介绍。**



首先会通过创建委托，**通过汇编级的覆写AmsiScanBuffer来绕过Windows操作系统的AMSI反恶意软件扫描接口。**

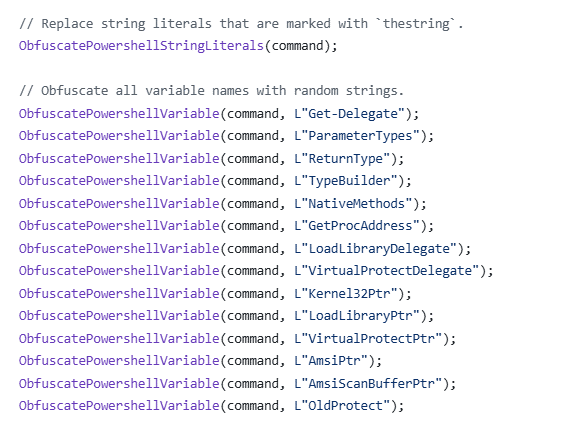


**之后会通过StrCatW函数，将stager相关命令行指令进行拼接，**从注册表中取出并执行。这里拼接好的指令是

|  |
| --- |
| [Reflection.Assembly]::load \\  ([Microsoft.Win32.Registry]::LocalMachine.OpenSubKey(software).getValue(r77stager)) \\  .EntryPoint .invoke(null,null) |

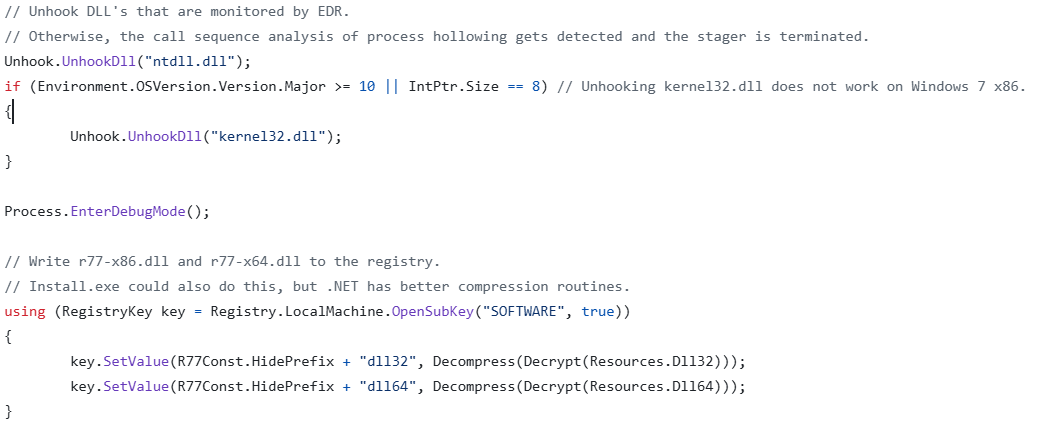
 加载后的程序集通过 **.EntryPoint 获取其入口方法**。

 **.Invoke($Null, $Null) 调用入口方法**。



最后将之前命令行中使用过的**字符串进行混淆**。因此，汇编级的简单字符串分析不能追踪到原始shell命令。

**2.4 Stager**

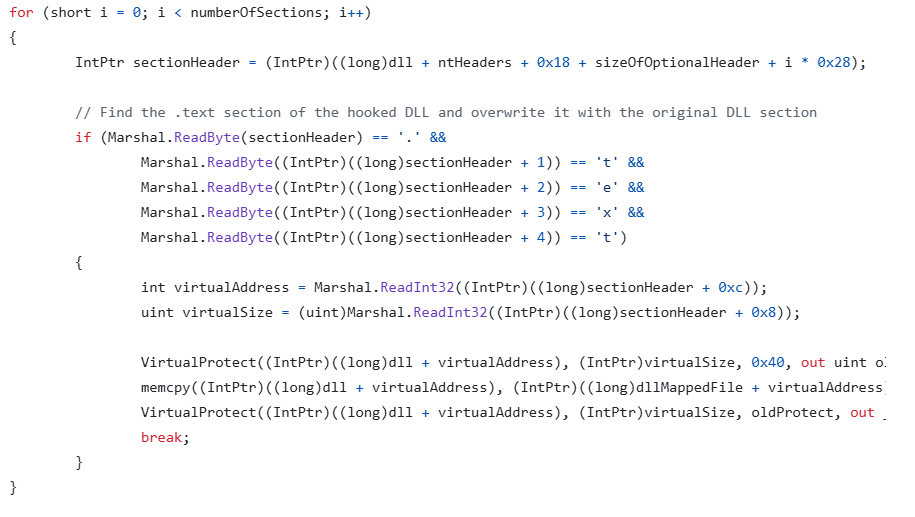


其中**，UnhookDll是用于解除目标dll的钩子的函数，主要通过找到目标dll的.text(代码部分)进行复原，**从而解除原来由于杀毒软件等给dll中附上的钩子。

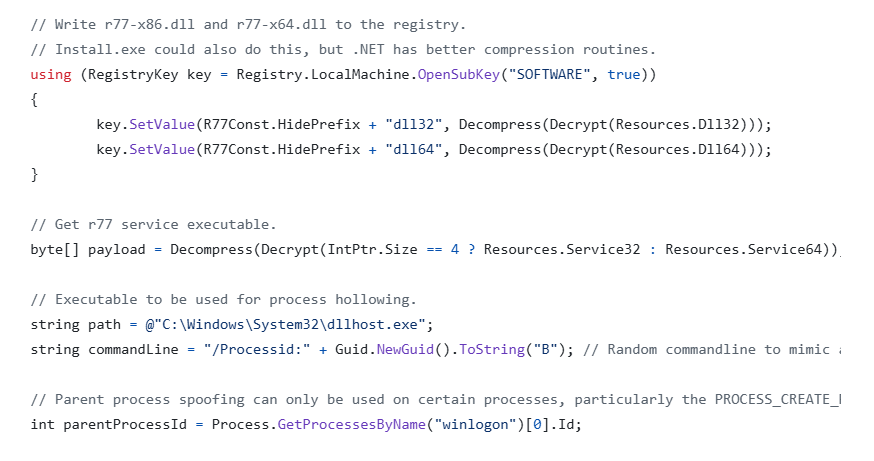
那么纯净的、没有被挂过钩子的dll文件怎么获取呢？



**这里打开了c:\windows\system32下的dll文件，应该是默认这里的dll文件不会被杀毒软件等进行更改。**



之后，修改当前进程的虚拟地址中的数据。其中，**VirtualProtect函数用于更改当前进程地址空间中某个区域的内存保护属性，这里是将(long)dll + virtualAddress处的保护属性进行了修改，并在下面进行复制；**修改完毕后再将保护属性恢复。

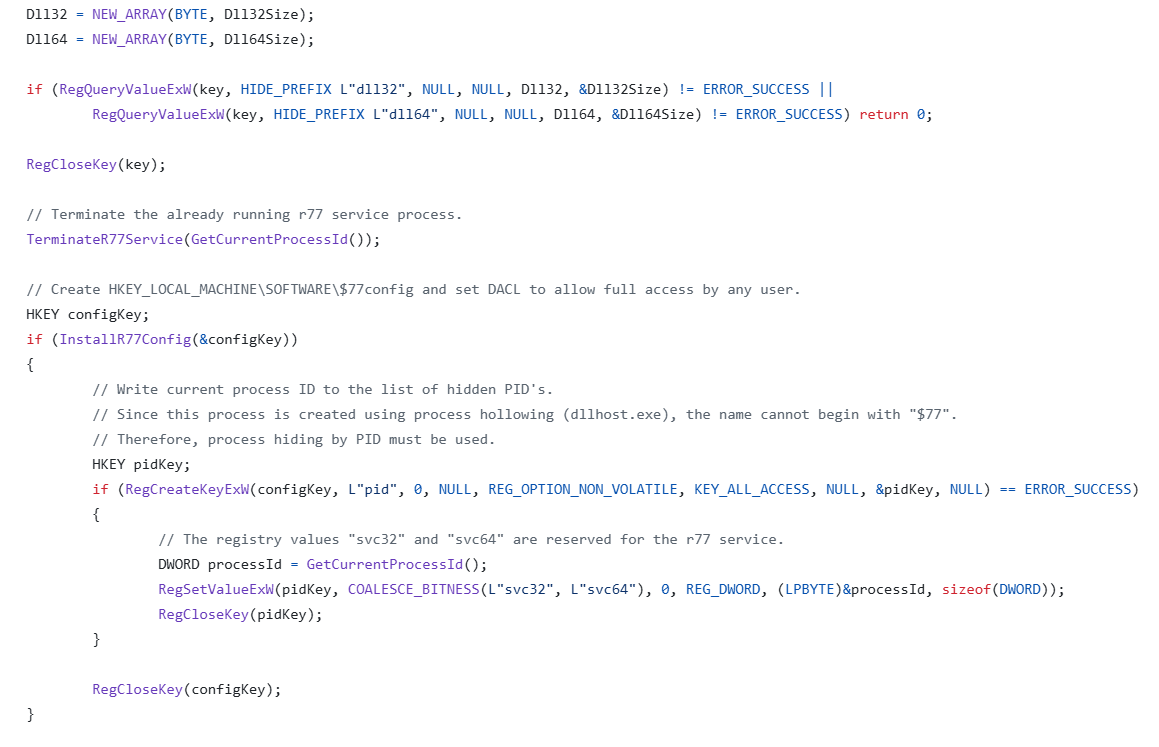


随后，**将r77-x86.dll和r77-x64.dll写入到\HKEY\_LOCAL\_MACHINE\**

**SOFTWARE\下的注册表中。**

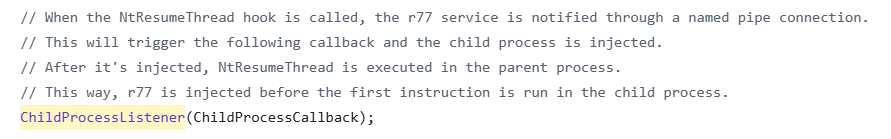
**之后，通过内存指针找到service.exe，再启动一个dllhost.exe程序作为winlogon的子进程，最后，通过进程镂空技术，使用service.exe替换winlogon，**从而实现service.exe进程的创建。

**2.5 Service**



先**读取Stager运行时写入到注册表的\HKEY\_LOCAL\_MACHINE\SOFTWARE**

**\$77dll的值**，也就是r77 x64/r77x86.dll文件的内容。





**再实现一个类似全局注入的逻辑**——创建一个监控进程，并提供一个回调函数。这个监控进程是尝试连接指定名称的管道pipe，当连接成功的时候读取获取到的内容（这个内容其实就是一个pid），然后调用提供的回调函数操作这个pid；这个回调函数是一个远程进程注入函数，通过指定的目的pid，向对应的进程注入第二步获取的r77x64/32.dll。

在R77的说明文档中，介绍了注入dll后的操作。

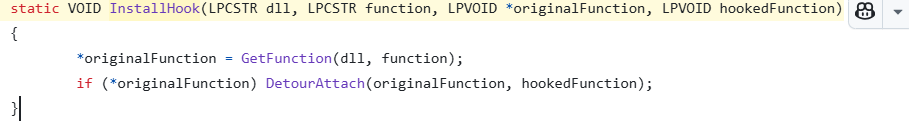
**一是挂钩NtResumeThread()函数。这个函数的前缀来自我们课上学过动态链接库的Ntdll.dll，它在进程由挂起态转为运行态时被调用。由于Windows系统在创建完大部分新进程后，都会首先将它置为挂起态，**因此，可以保证大多数进程可以被感染。

二是每隔100ms的时间间隔检查所有新创建的进程，并将没感染的进程重新感染，这样可以感染services.exe等进程创建的进程。

**2.6 R77**

这里就是上面在Stager写入注册表，servcie从注册表中读出的r77x32/x64.dll的实现。





这里通过微软研究部门开发的**开源Detours库来进行函数的重定向。被重定向的函数如下。**

**2.6.1 NtQuerySystemInformation**

**该函数用于计算正在运行的进程占用的CPU总量，检索CPU使用情况。**

**2.6.2 NtResumeThread**

**该函数用于注入创建的子进程，当新进程仍处于挂起状态时调用。注入完成后，才会实际调用此函数。（文档中提到，Hook CreateProcess不是一个好选择，因为它会在一次调用中创建并启动进程，并且可能和多个其他高级API有关）。**

**2.6.3 NtQueryDirectoryFile**

**该函数用于其他软件枚举查看文件、目录、连接点和命名管道时对特定文件进行隐藏。**

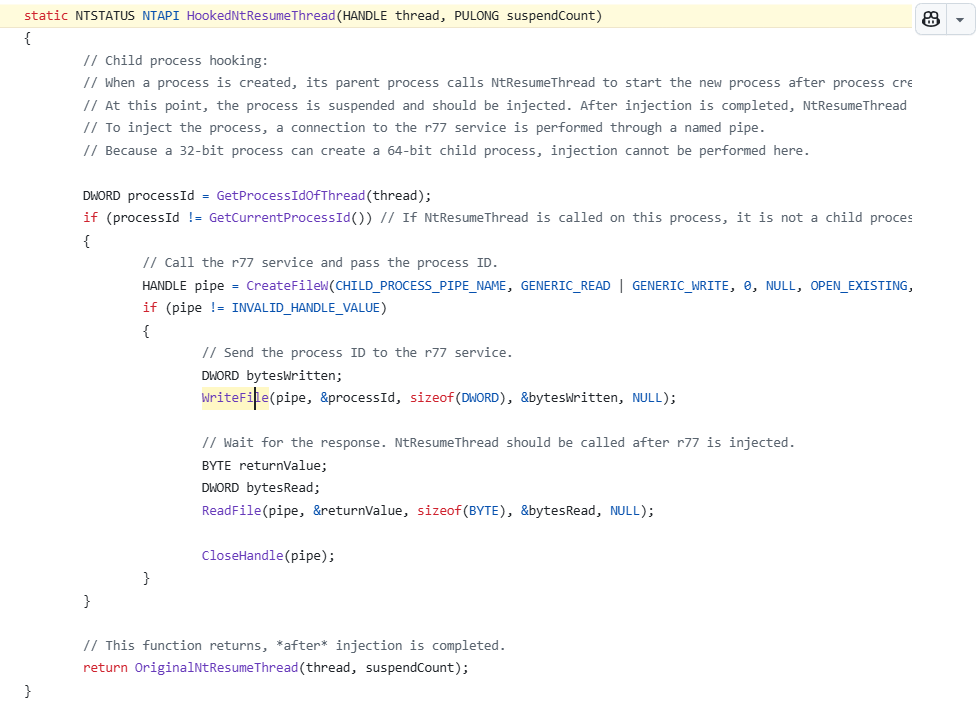
**2.6.4 NtEnumerateKey、NtEnumerateValueKey、NtEnumerateServiceGroup**

该函数用于枚举注册表键（值、服务）。调用者指定一个键的索引以检索该键。为了隐藏注册表键，必须调整索引。因此，需要重新枚举键，以便找到正确的“新”索引。

**2.6.5 NtDeviceIoControlFile**

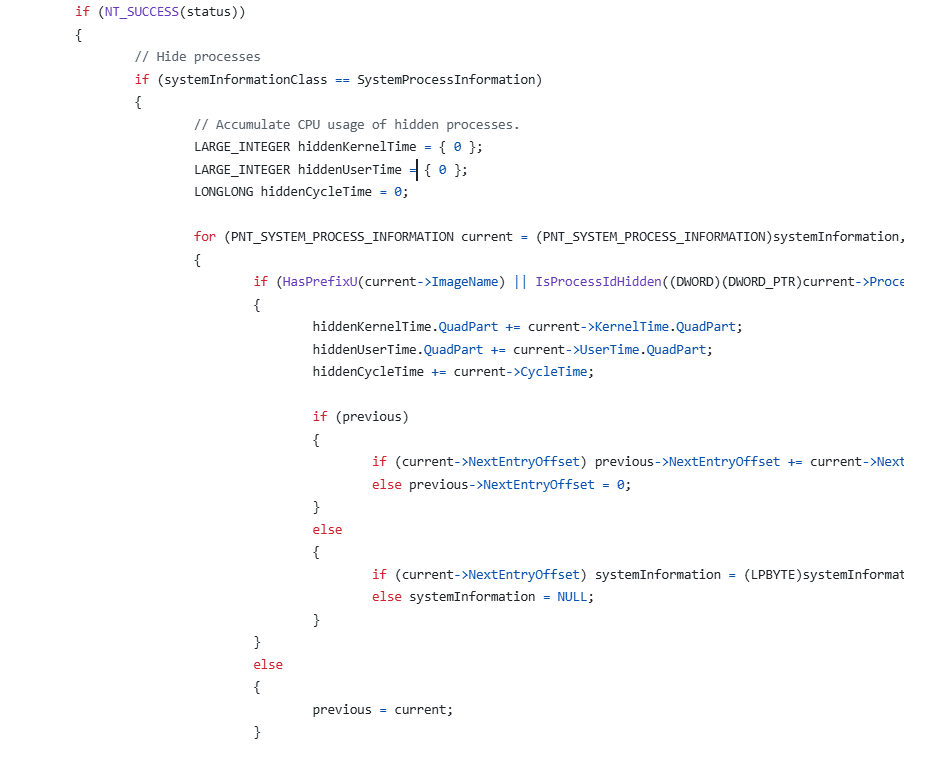
该函数用于通过IOCTL（与内核中的设备驱动程序交互的接口）访问驱动程序，请求调用方获取所有 TCP 和 UDP 连接的列表。

**当r77运行后，首先感染已经启动的所有进程。接下来创建的进程就通过上面挂钩的函数进行持续感染。**

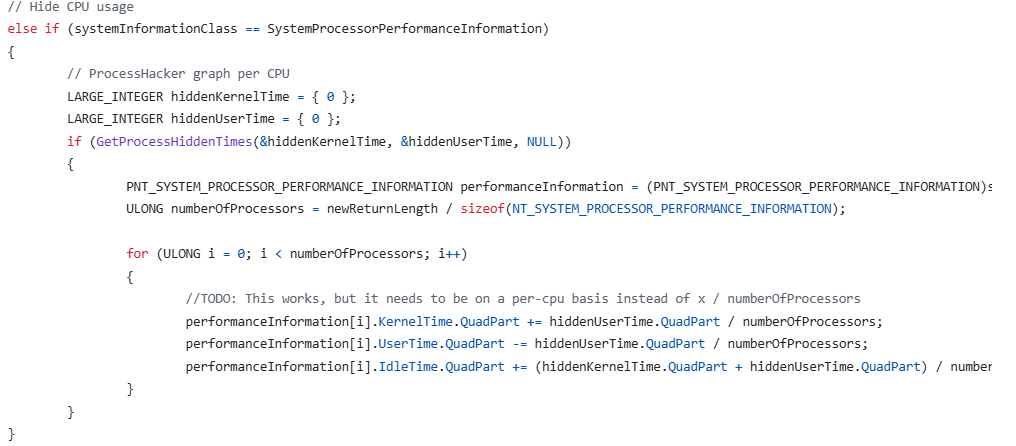


**进入挂钩后的NtResumeThread()看看，确实调用了WriteFile()函数，但是可惜没看见说明文档中提到的具体对Dos头修改的值。**

**2.6 对文件、注册表、TCP端口的隐藏**



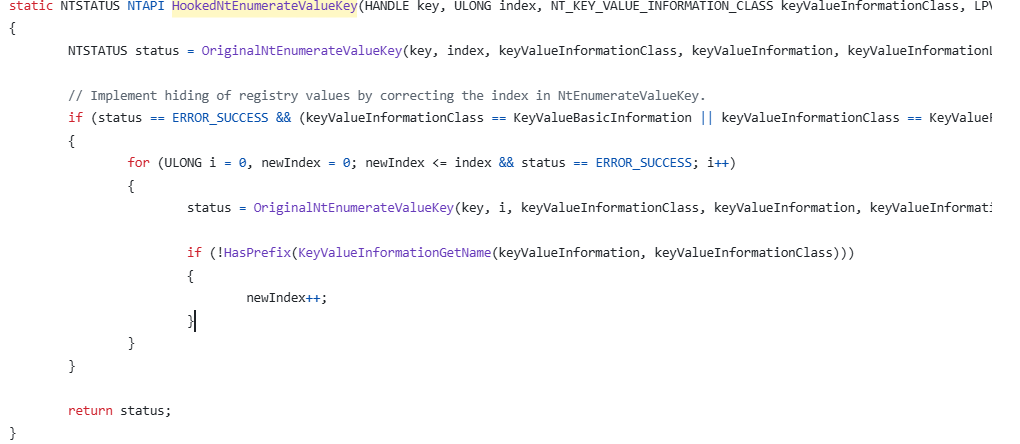
先介绍一下这里的数据结构。PNT\_SYSTEM\_PROCESS\_INFORMATION是用于描述系统中的进程和线程信息的数据结构，里面包含**LARGE\_INTEGER类型的KernelTime和UserTime**。而LARGE\_INTEGER又包含QuadPart成员，用于记录64位大整数的数值。

****

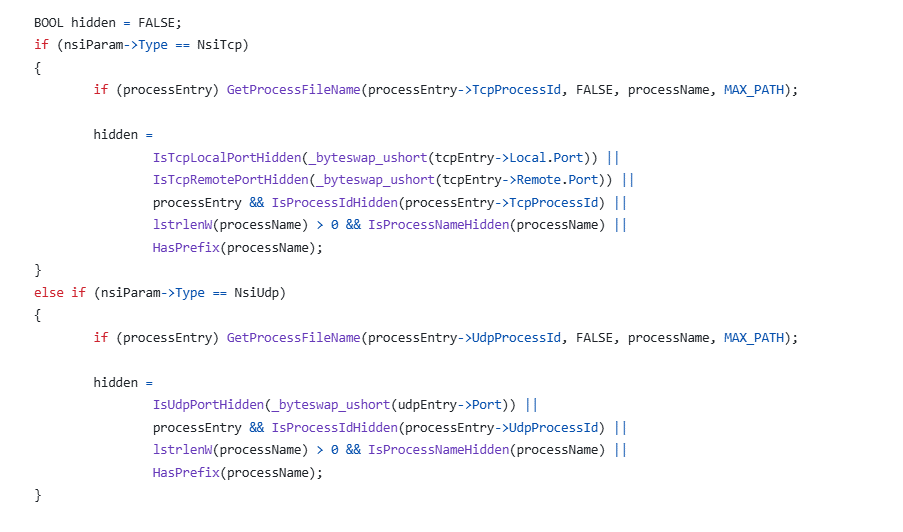
这里计算CPU时间的逻辑很简单。分别计算固定时间内Kernel和User态下CPU运行时间即可，用运行时间占总测试时间的比例作为CPU占用率。

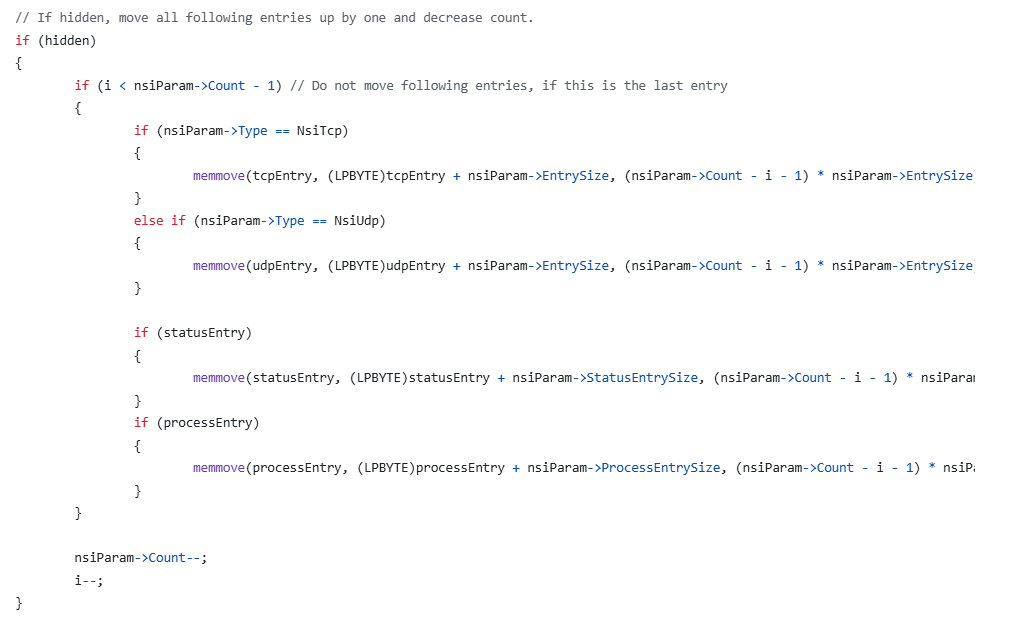
****

这里对文件的显示也很简单，在显示File前先判断文件名是否有前缀、文件路径是否满足、是否由已经隐藏的进程创建的。如果是，隐藏该文件。

****

对注册表键值对的显示时，分析键的路径决定是否隐藏。





对TCP和UDP判断时，依次分析端口、创建它的进程是否在隐藏范围内，如果在，将检索到的TCP、UDP隐藏。

1. **实验结论及心得体会**

其实这是上一次R77作业的延续。上一次作业中我已经在源代码程度上对R77的实现做了一定程度的分析。完成这次作业时，我又重新阅读了R77的技术文档，把之前分析时没提到的部分又增加进来了。同时，对源代码，尤其是开始的Install和最后的Detours过程进行了更加深入的逻辑与函数调用层面上的分析。

在完善作业的过程中发现了许多之前忽略了的东西，例如Detours机制挂钩的函数都在挂钩后各自做了些什么，进程的隐藏是通过什么方式实现的等。再看一遍侧重点更加深入、分析过程更加流畅，新的收获很大。