

**《恶意代码分析与防治技术》课程实验报告**

**实验11-2**

****

学 院 网络空间安全学院

专 业 信息安全

学 号 2112060

姓 名 孙蕗

1. **实验要求**

在使用R77的基础上，撰写技术分析，要求描述使用过程中看到的行为如何技术实现。

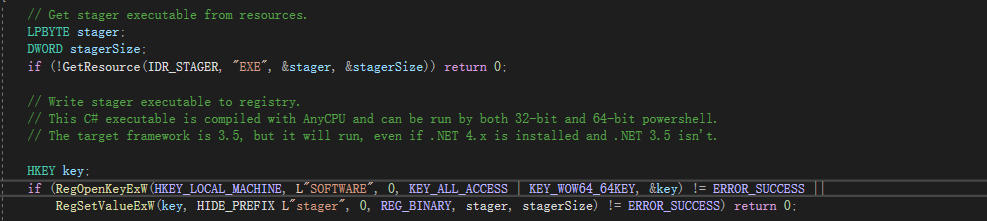
1. **实验过程**
2. **查看install.cpp**

通过将可执行文件写入Windows注册表，创建定时任务以执行一个PowerShell命令（加载和执行stager），并使用各种混淆技术来逃避静态分析。尝试绕过AMSI表明这是为了规避杀毒软件和其他安全机制的检测。

1. Stager提取和注册表写入:

代码首先使用GetResource函数从资源中提取一个名为stager的可执行文件。该stager以字节数组（stager）及其大小的形式获取。

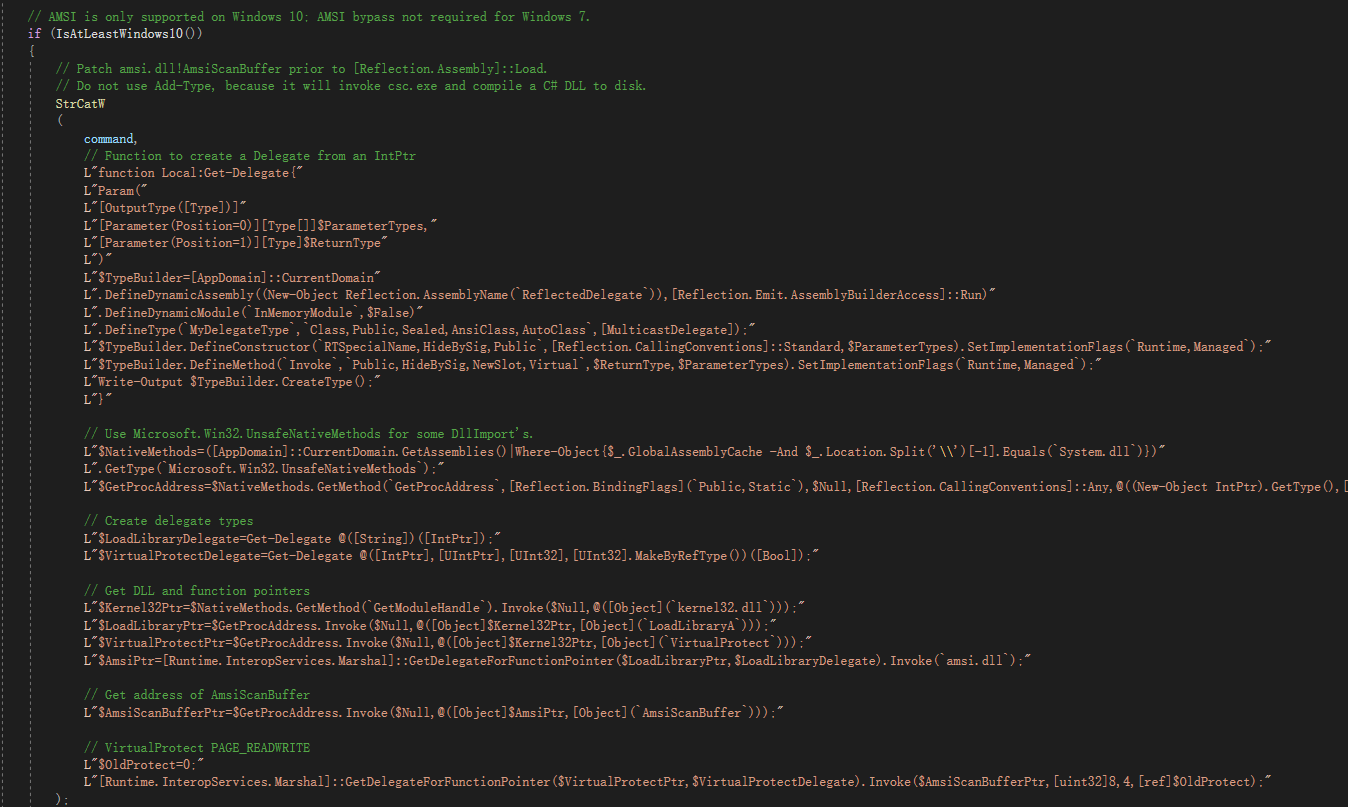
接着，它尝试打开Windows注册表（HKEY\_LOCAL\_MACHINE\SOFTWARE）并将stager可执行文件写入注册表，注册表项的名称前缀为HIDE\_PREFIX。这可能是为了在注册表中隐藏该项。



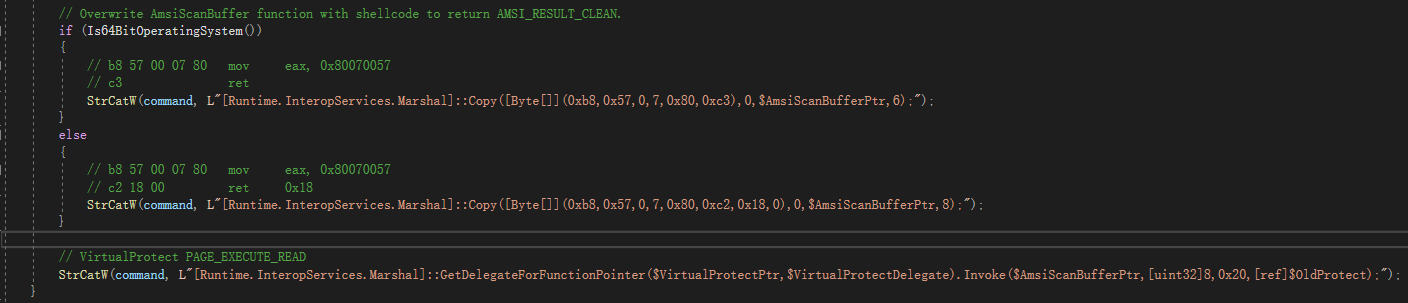
1. 生成Powershell命令:

GetPowershellCommand函数生成一个PowerShell命令，执行以下几个任务：

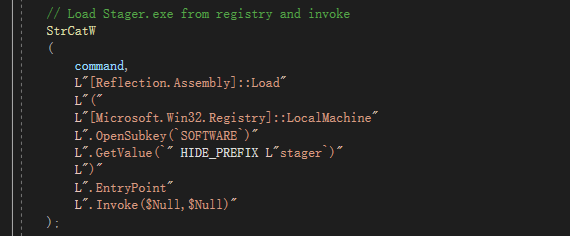
检查系统是否运行Windows 10或更高版本。



如果在Windows 10上，尝试绕过AMSI（Antimalware Scan Interface），通过在内存中修补AmsiScanBuffer函数，使其始终返AMSI\_RESULT\_CLEAN。

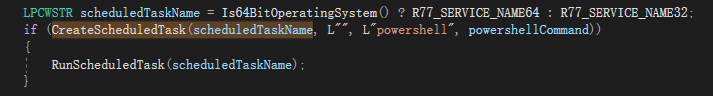


构造PowerShell命令以从注册表中加载并执行stager。



1. 创建定时任务:

代码尝试使用CreateScheduledTask函数创建一个定时任务。此任务配置为运行一个PowerShell命令，可能是前一步中生成的命令，并计划在将来执行。

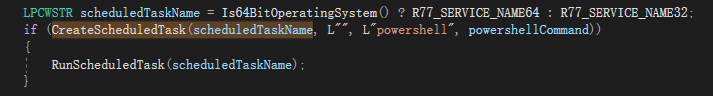


在创建新任务之前，删除了名为R77\_SERVICE\_NAME32和R77\_SERVICE\_NAME64的现有定时任务。



1. 定时任务执行:

创建定时任务后，代码尝试立即运行它，使用的是RunScheduledTask函数。此执行触发执行PowerShell命令，从而加载和执行stager。

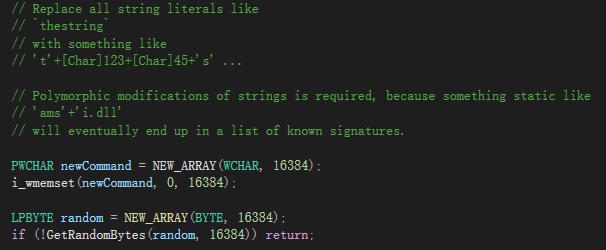


1. 字符串混淆:

代码包含用于混淆字符串文字和PowerShell变量名称的函数。

1. ObfuscatePowershellStringLiterals修改字符串文字，使得字符串在保持原有语义的同时，更难以被静态分析工具检测到。

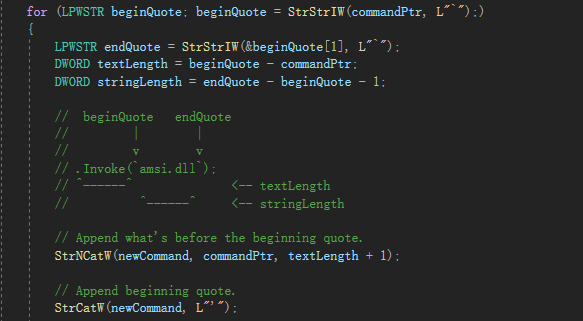
newCommand 用于存储修改后的命令，random 用于生成随机数，增加混淆。使用 i\_wmemset 函数将 newCommand 缓冲区初始化为零。

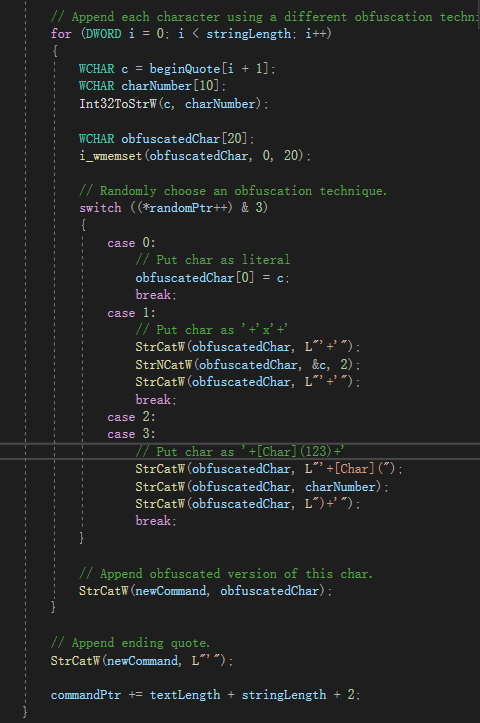


遍历PowerShell命令，查找用反引号（`）括起来的字符串文字。反引号是PowerShell中用于转义字符的特殊符号。

对于每个找到的字符串文字，将其前面的部分追加到 newCommand 中。

用单引号（'）替换开始的反引号，并开始对字符串文字中的每个字符进行混淆。使用了一些随机选择的混淆技术来对PowerShell命令中的字符串文字进行多态化，包括字符作为字面值、字符拼接为字符串、字符表示为 [Char](ASCII值) 等。



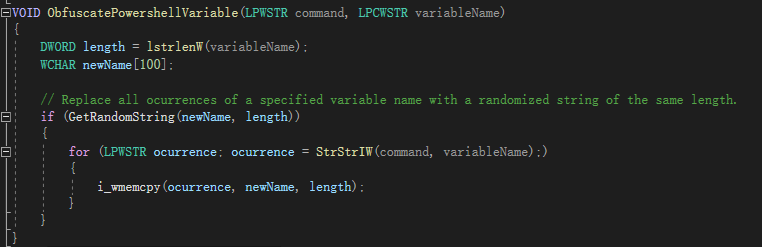


将修改后的命令 newCommand 复制回原始的命令缓冲区 command。

释放分配的内存。



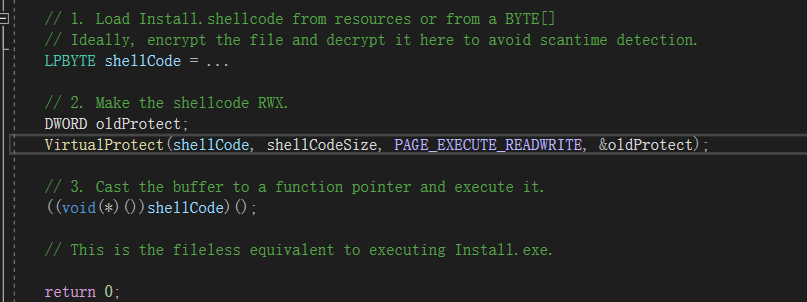
1. ObfuscatePowershellVariable用随机字符串替换指定变量名称的出现。



1. InstallShellCode.cpp

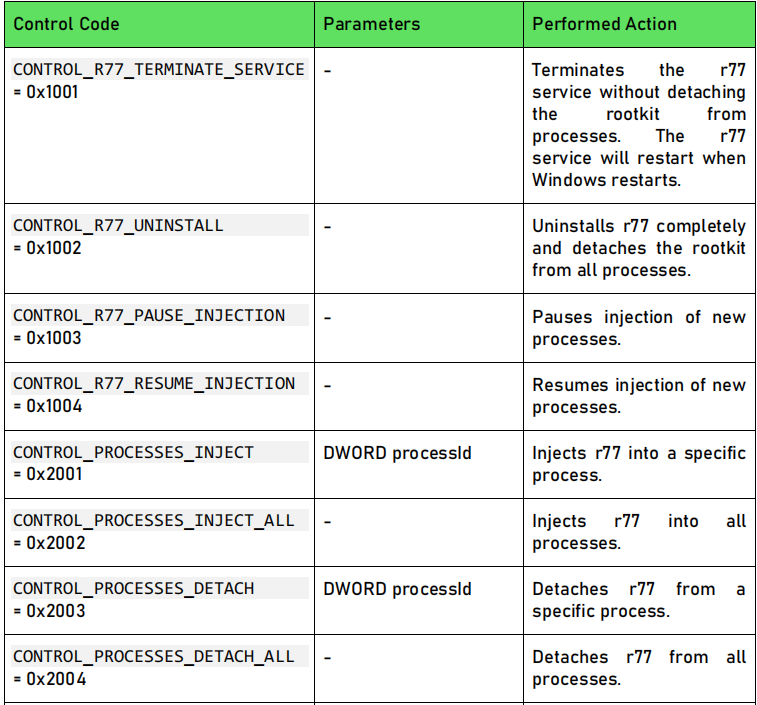
获取 Install.shellcode 的二进制数据，使用 VirtualProtect 函数将 Shellcode 的内存页设置为可读写可执行的,然后将 Shellcode 的地址强制转换为函数指针，然后通过调用这个函数指针来执行 Shellcode。

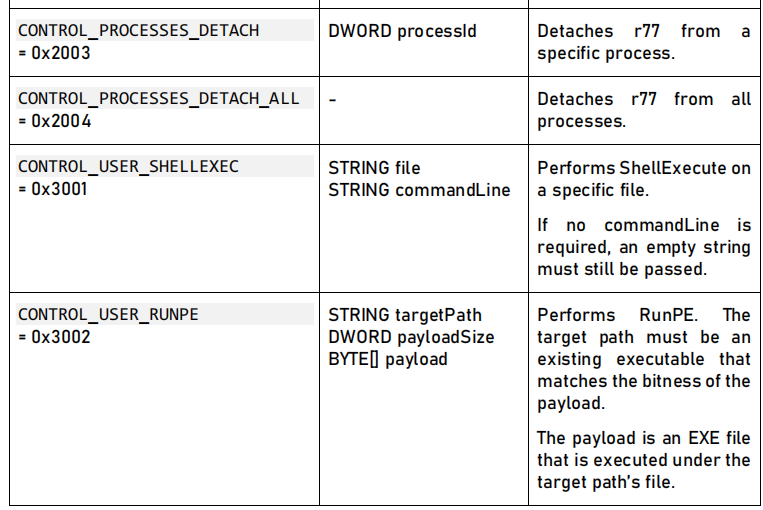
通过加载和执行 Shellcode 的方式来达到一些目标，这可能包括执行系统级任务、绕过安全检测、植入恶意代码等。这种技术常常用于恶意软件攻击中，因为它允许攻击者在内存中执行代码而无需将完整的可执行文件写入磁盘，从而更难以被检测和防范。

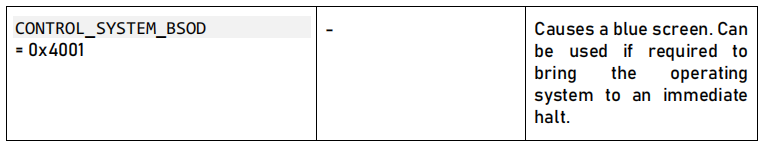


1. 控制

r77服务命令发送和接受主要通过控制管道进行。控制管道是一个命名的管道，其中r77服务接收来自任何进程的命令并执行它们。这样，一个进程（即使具有较低的特权）就可以请求r77来执行某些操作。





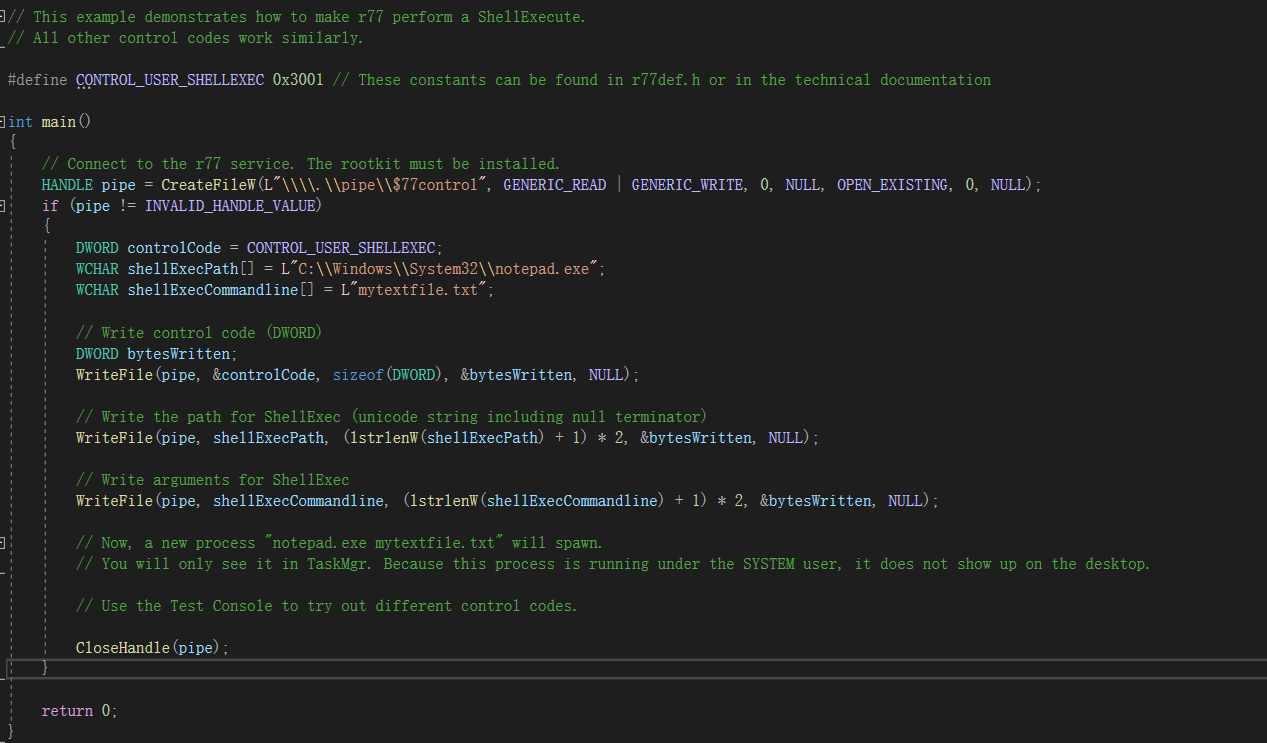


\\.\pipe\$77control

在每一次命名管道的创建前，程序都会自动加上隐藏前缀$77.

在接收到指令时，软件会先对Controlcode进行校验，而后调用相应的命令。

在加载完成之后，程序连接到r77服务，并且写入控制代码和可执行文件的位置。ControlPipe.cpp通过pipe，加载了notepad.exe mytextfile.txt的进程。



1. Rookit DLL反射式DLL注入

r77采用的是反射DLL注入，一旦注入到进程中，对应进程就不会显示被隐藏相关信息。文件被写入远程进程内存，并调用ReflectiveDllMain导出以最终加载DLL并调用DllMain。因此，DLL不会在PEB中列出。

r77中的反射注入思路：

将进程指针地址复制给进程句柄，打开进程(OpenProcess)，创建线程（PROCESS\_CREATE\_THREAD），获取线程信息（PROCESS\_QUERY\_INFORMATION），读写内存(PROCESS\_VM\_OPERATION)；

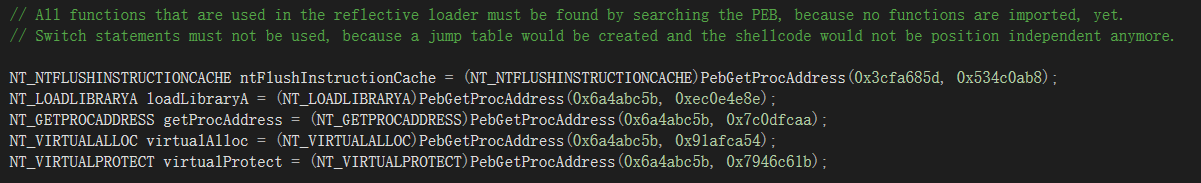
通过NtCreateThreadEx创建线程，将allocatedMemory + entryPoint作为开始地址。

查看ReflectiveDllMain.c

主要用于将 DLL 文件加载到内存中并执行其中的 DllMain 函数。这种加载方式可以绕过传统的 LoadLibrary 和 GetProcAddress 的调用，提高代码的防御性和难以静态分析。

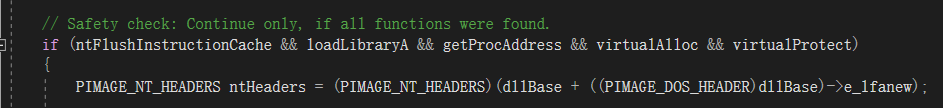
1. 获取重要函数地址：

使用 PebGetProcAddress 函数从 PEB（进程环境块）中获取 ntFlushInstructionCache、loadLibraryA、getProcAddress、virtualAlloc、virtualProtect 函数的地址。



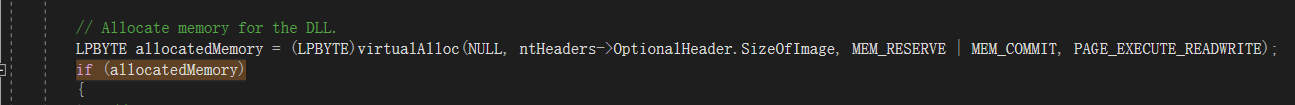
1. 执行安全性检查：

确保获取所有函数的地址成功，否则返回 FALSE。



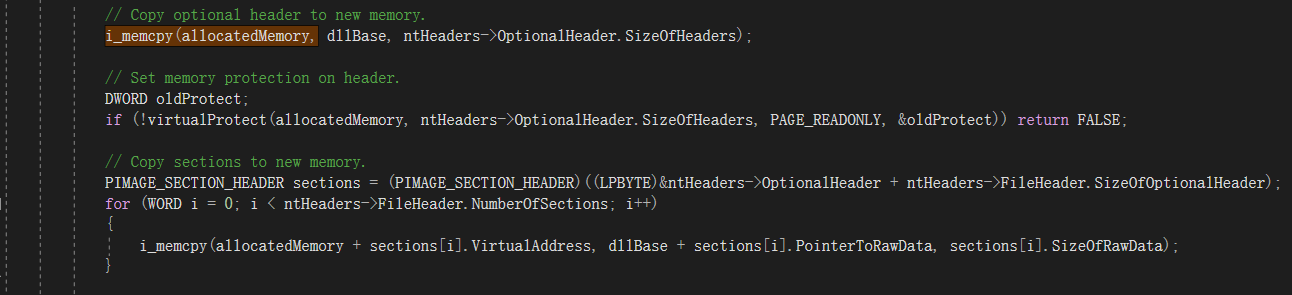
1. 分配内存空间：

使用 virtualAlloc 在进程的地址空间中分配内存，大小为 DLL 文件的大小（ntHeaders->OptionalHeader.SizeOfImage）。



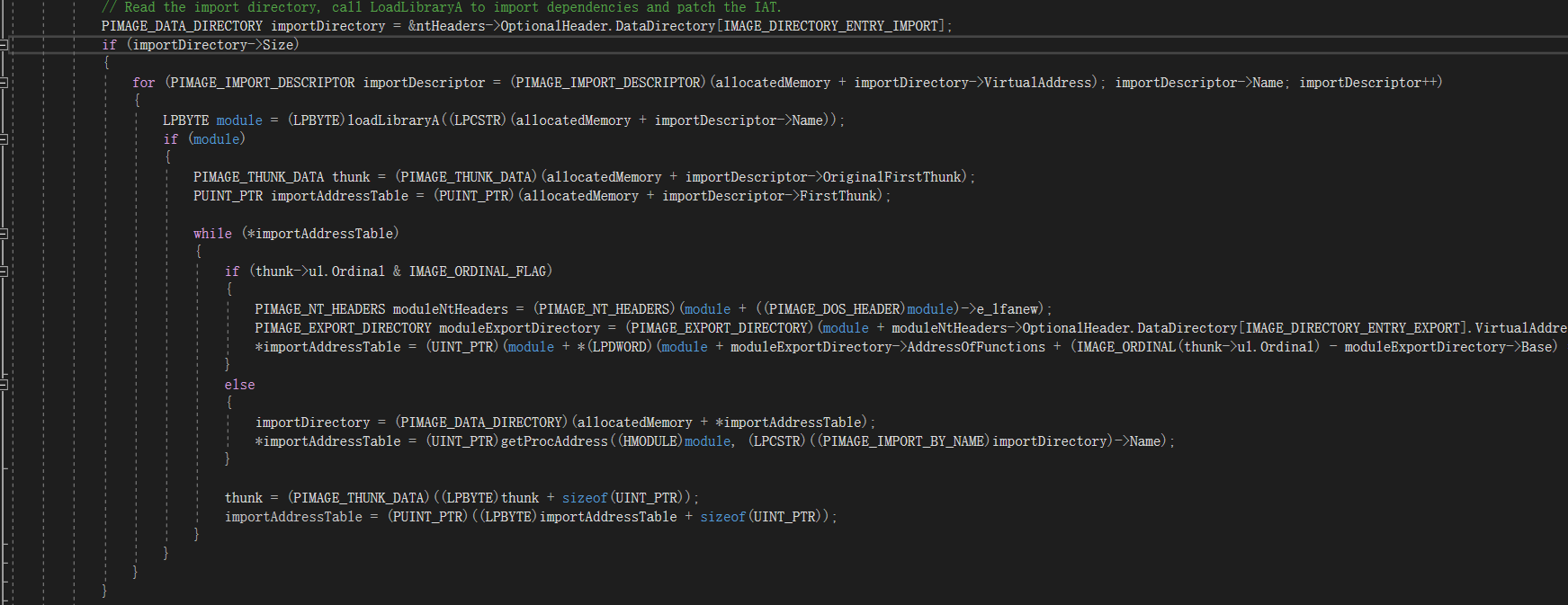
1. 复制 DLL 头和节表：

将 DLL 的头部和各个节（sections）复制到新分配的内存空间。



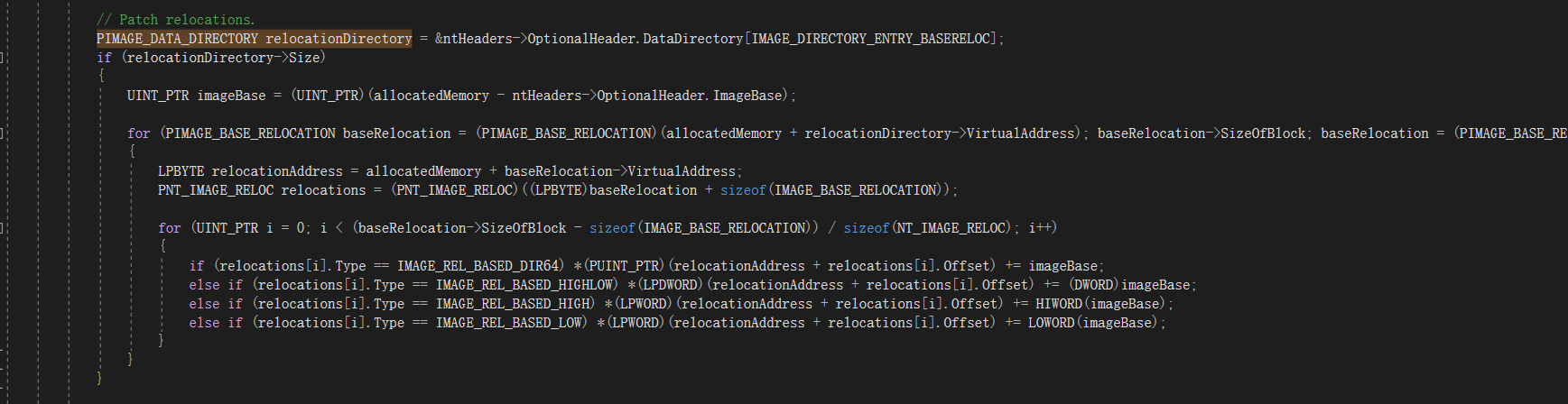
1. 读取导入表（Import Table）：

读取 DLL 的导入表，调用 LoadLibraryA 导入依赖项，并修复导入地址表（IAT）。



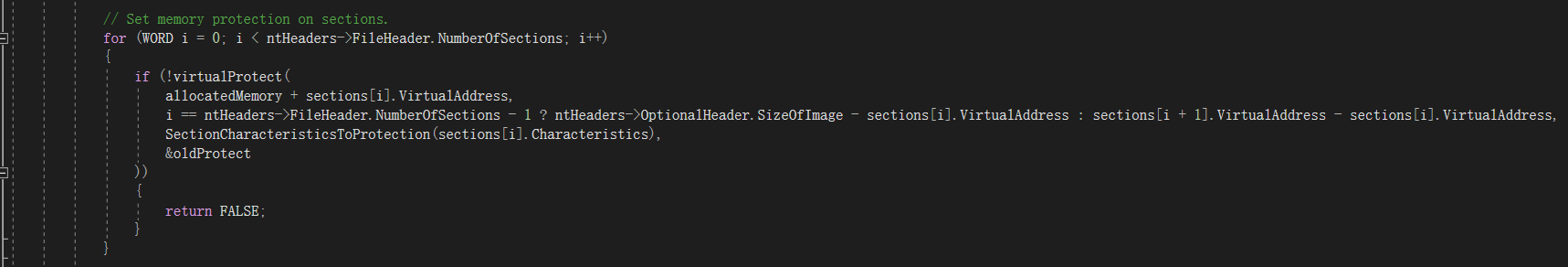
1. 修复重定位表（Relocation Table）：

如果 DLL 包含重定位表，修复其中的地址，使其适应新的加载地址。



1. 设置内存权限：

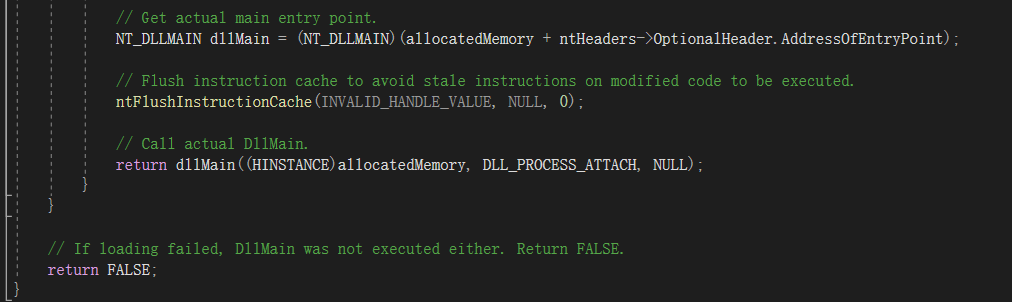
根据节的属性，设置内存的保护权限。



1. 获取实际的入口点（Actual Entry Point）：

获取 DLL 的 DllMain 函数的地址，然后调用它。

1. 清理缓存并调用 DllMain：刷新指令缓存，然后调用 DllMain。



1. PebGetProcAddress 函数：

PebGetProcAddress 函数是一个辅助函数，用于从 PEB 中获取模块和导出函数的地址，以便在不引入导入表的情况下调用这些函数。该函数的实现通过哈希值匹配模块名和函数名。



1. 隐藏

r77Rootkit通过注册表进行操作来进行总体的配置，它在HIDE\_PRIFIX配置了$77，因此以之为开头的文件、进程、计划任务和命名管道都会被隐藏。有关隐藏项目的注册表项位于HKEY\_LOCAL\_MACHINE\SOFTWARE$77config中，此键值的DACL设置为授予任何用户完全访问权限，也是因此，可由任何没有提升权限的进程写入。

r77中的“隐藏”意味着从枚举中删除隐藏的实体。如果用户知道文件名或进程ID，仍然可以直接访问文件、打开进程。这是因为打开文件、进程等等的函数没有被hook，而且它们不会通过返回“未找到错误”来伪装隐藏。主要原因是，r77目前没有其他方法来维持它本身。粒入球如果隐藏的注册表键值完全不可访问，r77也无法从配置系统中读取它自己。

为了防止被读取，可以通过设置足够复杂的文件名来确保r77相关的名称不会被猜解。同样，也可以在编译阶段修改隐藏前缀，但r77不能在一次项目中变更多个不同的前缀。

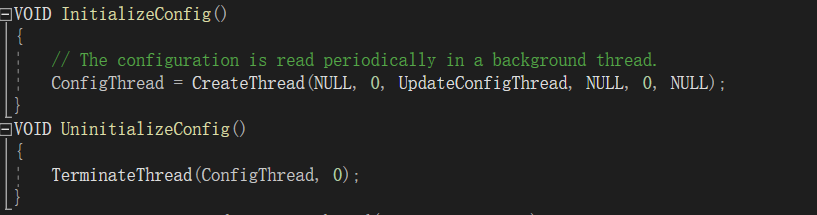
查看config.c

通过配置管理模块，使得某些进程、路径、服务或者端口可以被隐藏，从而增加程序的隐蔽性。隐藏的规则和信息存储在配置结构体 PR77\_CONFIG 中，由后台线程负责加载和更新。

1. 初始化和反初始化配置：

InitializeConfig 函数用于初始化配置，创建一个后台线程（UpdateConfigThread），该线程定期读取配置信息。

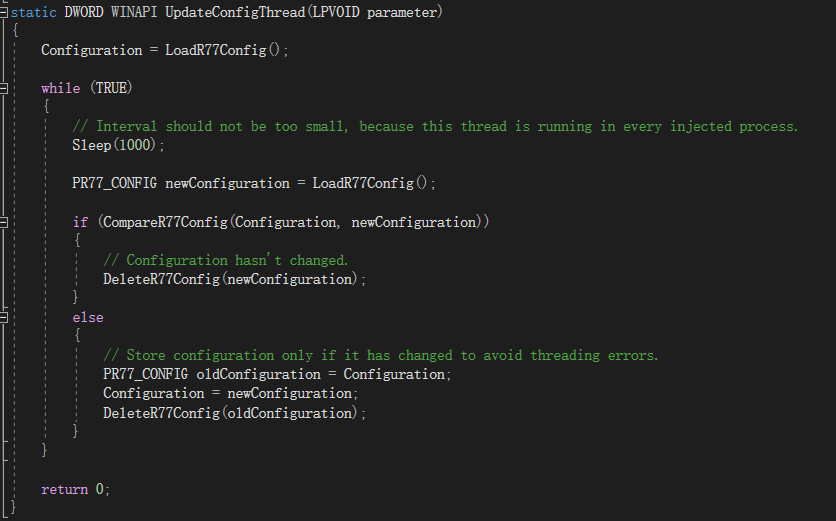
UninitializeConfig 函数用于反初始化配置，终止后台线程。



1. 后台线程（UpdateConfigThread）：

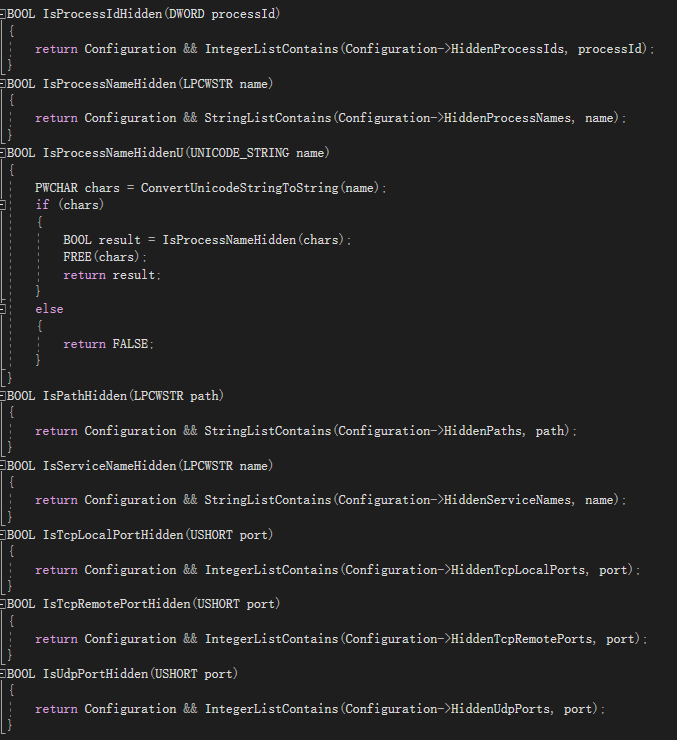
后台线程周期性地（每秒一次）加载新的配置信息，并与当前的配置信息进行比较。

如果配置信息有变化，更新当前配置，并释放旧的配置信息。



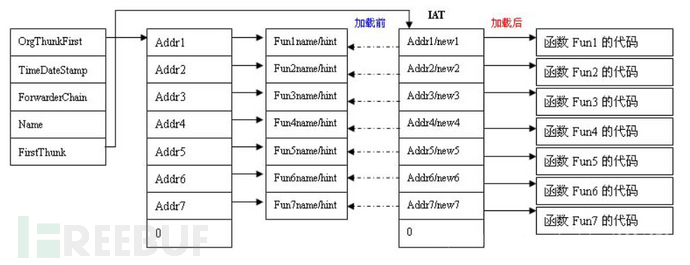
1. 检查进程、路径、服务、端口是否被隐藏：

提供一系列函数用于检查给定的进程 ID、进程名称、路径、服务名称以及 TCP 和 UDP 端口是否在配置中被隐藏。这些函数会检查当前的配置，如果配置中包含对应的信息，则返回 TRUE，表示该信息被隐藏。



1. Hook

Ring3 层的 Hook 基本上可以分为两种大的类型，第一类即是 Windows 消息的 Hook，第二类则是 Windows API 的Hook。每个调用的 API 函数地址都保存在 IAT 表中。 API 函数调用时，每个输入节所指向的 IAT 结构如下图所示。



r77中的hook主要通过调用Detours来实现，Detours是微软提供的一个开发库，使用它可以简单地实现API HOOK的功能。它可以通过为目标函数重写在内存中的代码而达到拦截Win32函数的目的。Detours还可以将任意的DLL或数据片段(称之为有效载荷)注入到任意Win32二进制文件中。

在r77中，detour被用于hook ntdll.dll中的几个函数。此DLL加载到操作系统上的每个进程中。它是所有系统调用的包装器，这使它成为Ring

3中可用的最低层。来自kernel32.dll或其他库和框架的任何WinAPI函数最终都将调用ntdll.dll函数。无法直接hook系统调用。这是Ring3

Rootkit的常见限制。

以下是被hook的函数：

NtQuerySystemInformation：此函数用于枚举正在运行的进程并检索CPU使用情况。

NtResumeThread:当新进程仍处于挂起状态时，此函数被挂起以注入创建的子进程。只有在注入完成后，才实际调用此函数。

NtQueryDirectoryFile:此函数枚举文件、目录、连接和命名管道。

NtQueryDirectoryGileEx:此函数与NtQueryDirectoryFile非常相似，并且也必须hook。实施方式大致相同。dir就使用此函数而不是NtQueryDirectoryFile。

NtEnumerateKey :此函数用于枚举注册表项。调用者指定键的索引以检索它。要隐藏注册表项，必须更正索引。因此，必须再次枚举该键才能找到正确的“新”索引。

NtEnumerationValueKey：此函数用于枚举注册表项。调用者指定键的索引以检索它。要隐藏注册表项，必须更正索引。因此，必须再次枚举该键才能找到正确的“新”索引。

EnumServiceGroupW：此函数用于枚举服务，主要被services.msc调用。

EnumServicesStatusExW：此函数类似于EnumServiceGroupW，主要被Windows 7下的任务管理器和ProcessHacker调用。

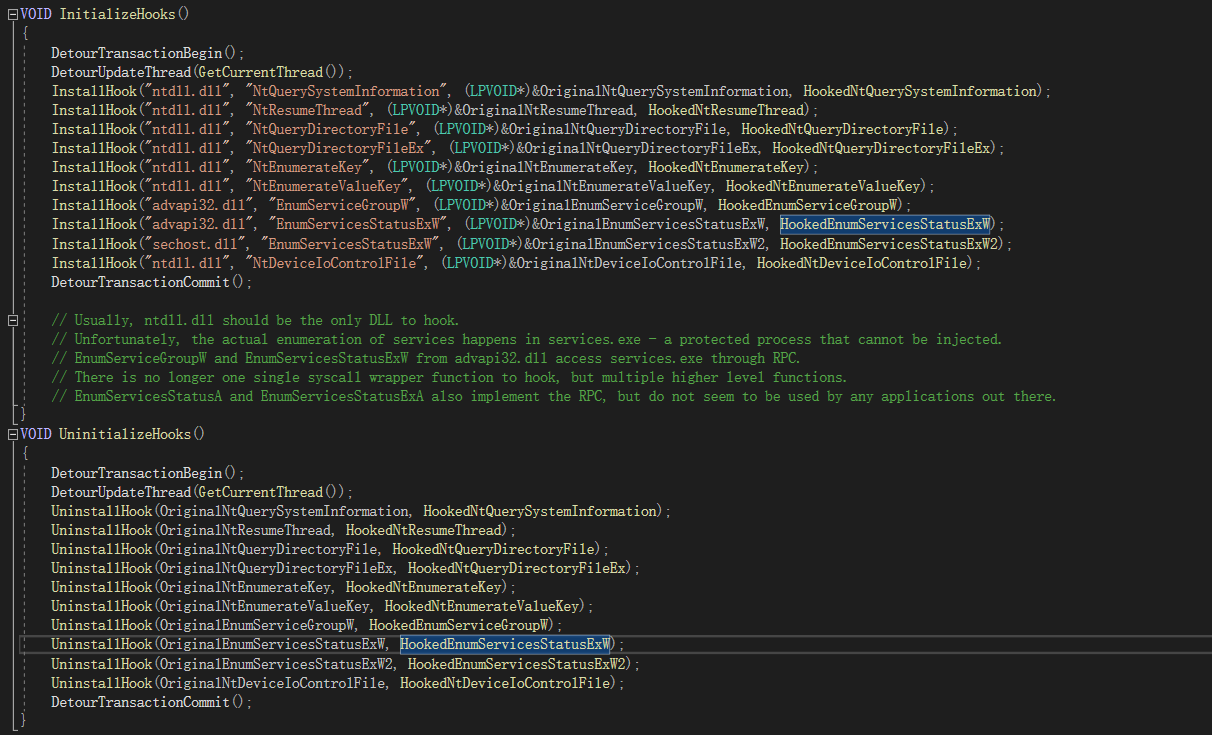
NtDeviceIoControlFile：此功能用于使用IOCTL访问驱动程序。

其中，除了EnumServiceGroupW、EnumServicesStatusExW来自更高级的DLL，advapi32.dll和sechost.dll之外，其他函数都来自ntdll.dll。一般来说，ntdll.dll确实是唯一要被hook的dll。但服务的实际枚举发生在service.exe，这是个无法注入的受保护进程。而来自advapi32.dll的EnumServiceGroupW和EnumServicesStatusExW通过RPC访问service.exe以搜索服务列表。ntdll.dll的钩子不会产生任何影响，因为只有service.exe使用这两个ntdll函数。

查看hook.c文件

1. 初始化和反初始化函数钩取

首先需要对detours进行初始化、需要更新进行detours的线程。InstallHook()函数中，则调用了DetourAttach()进行hook，这个函数的职责是挂接目标API，函数的第一个参数是一个指向将要被挂接函数地址的函数指针，第二个参数是指向实际运行的函数的指针，一般来说是我们定义的替代函数的地址。



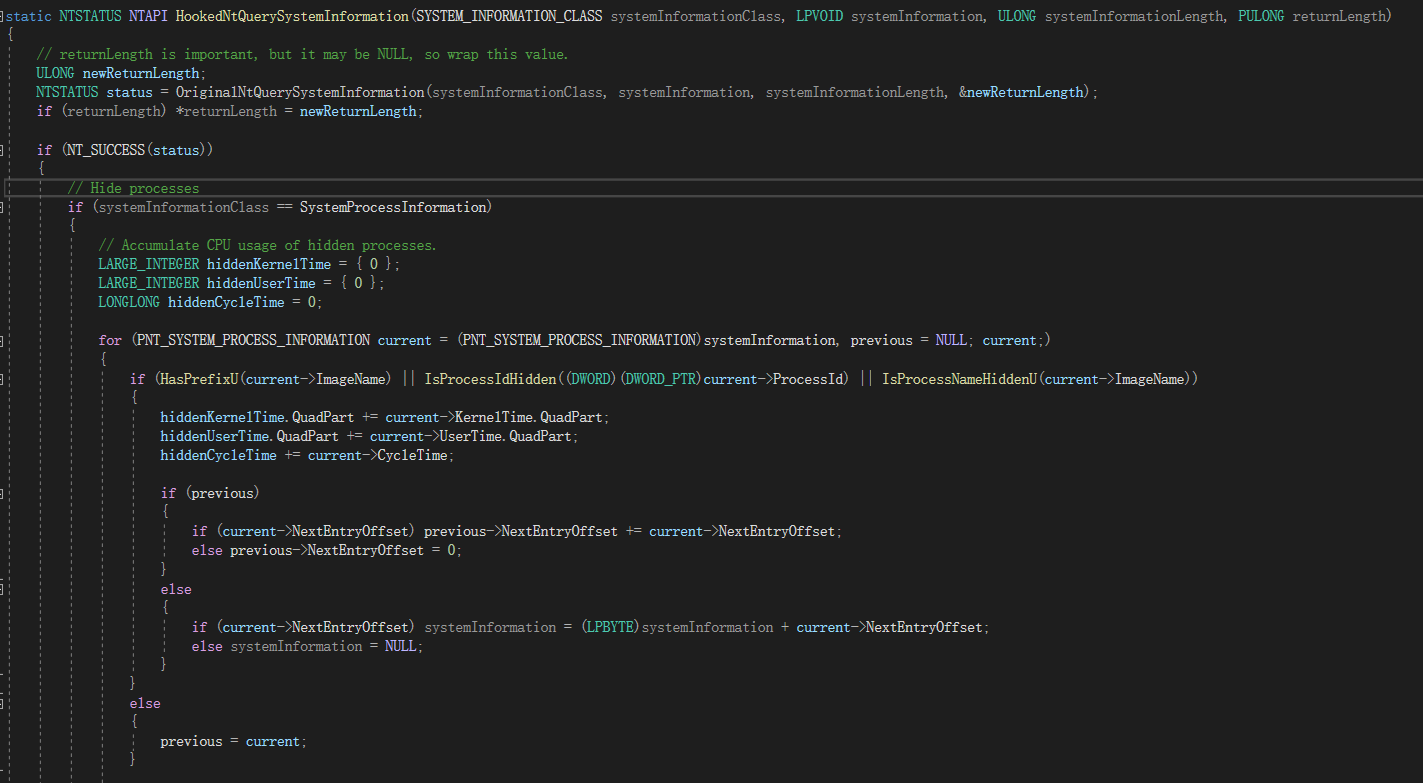


1. 隐藏指定的进程和相应的 CPU 使用情况

通过对 NtQuerySystemInformation 系统调用的钩取，实现了隐藏指定进程和修改 CPU 使用情况的功能。

1. 获取原始系统调用结果
2. 隐藏进程信息

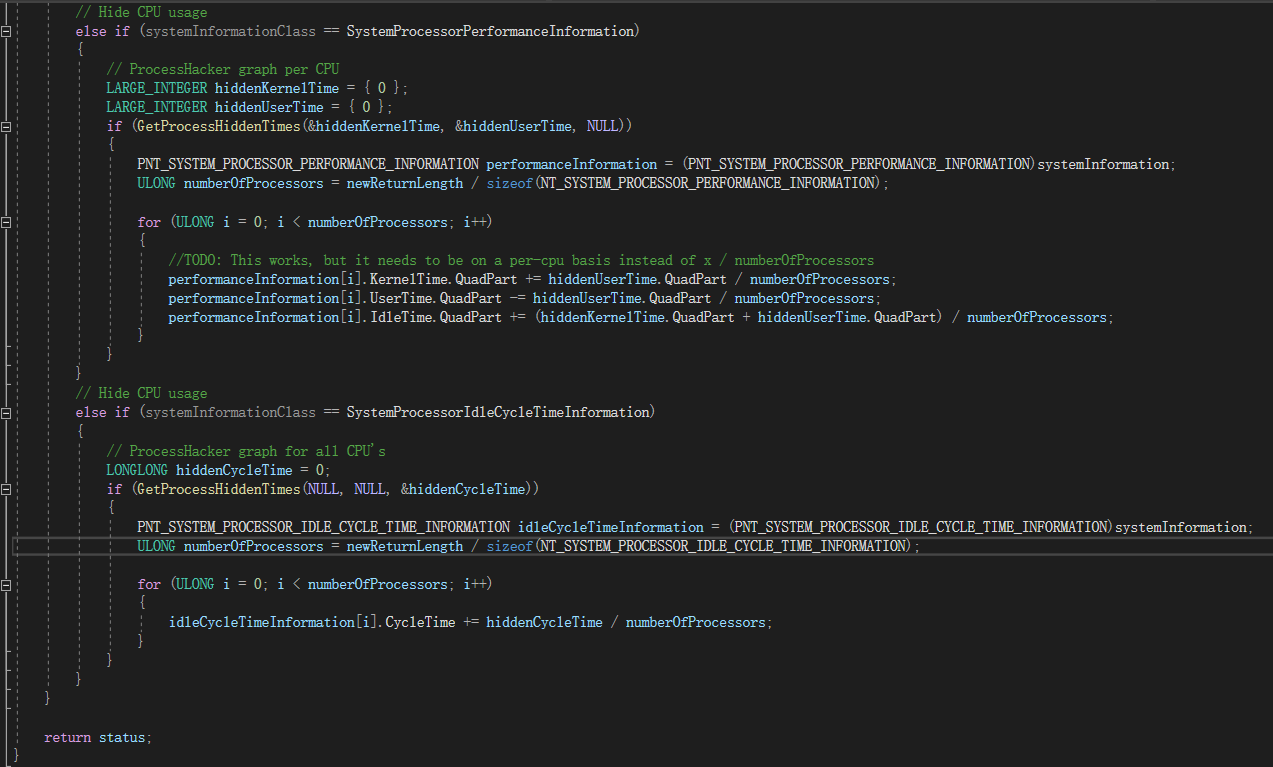
遍历系统进程信息，并根据一些条件判断是否需要隐藏该进程。如果需要隐藏，就将其从列表中删除，并将相应的 CPU 使用情况累积到 System Idle Process 条目中。





1. 隐藏 CPU 使用情况

获取隐藏的 CPU 使用时间或 CPU 周期时间，并将其分配给每个 CPU 条目



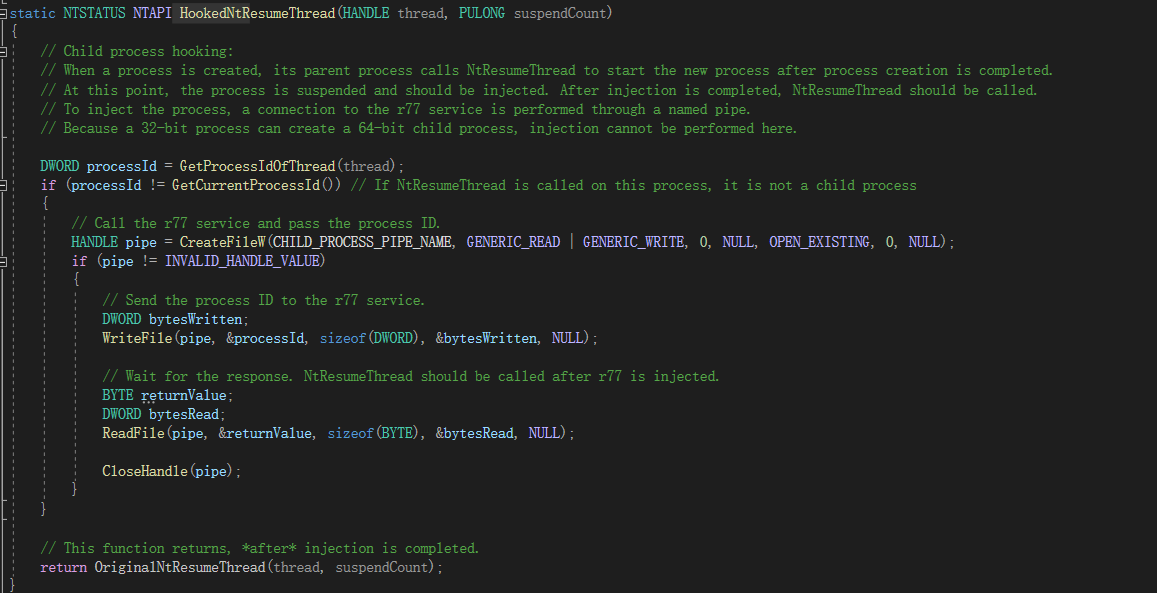
1. HookedNtResumeThread

该函数是 NtResumeThread 系统调用的钩子。主要目的是在恢复线程时，检测是否是子进程的创建过程，并在需要时通过与 r77 服务的通信实现进程注入。

通过获取线程所属进程的 ID，判断当前线程所在的进程是否为当前进程。如果不是，说明 NtResumeThread 被调用在一个子进程上，而不是当前进程。

如果是子进程，它会尝试通过创建一个命名管道与 r77 服务建立通信。然后，它将子进程的进程ID发送给 r77 服务，等待服务的响应。这一过程是为了确保 r77 服务已经完成对子进程的注入，然后再调用 NtResumeThread。

无论是否是子进程，最终都会调用原始的 NtResumeThread 函数，以完成线程的恢复操作。



1. HookedNtQueryDirectoryFile

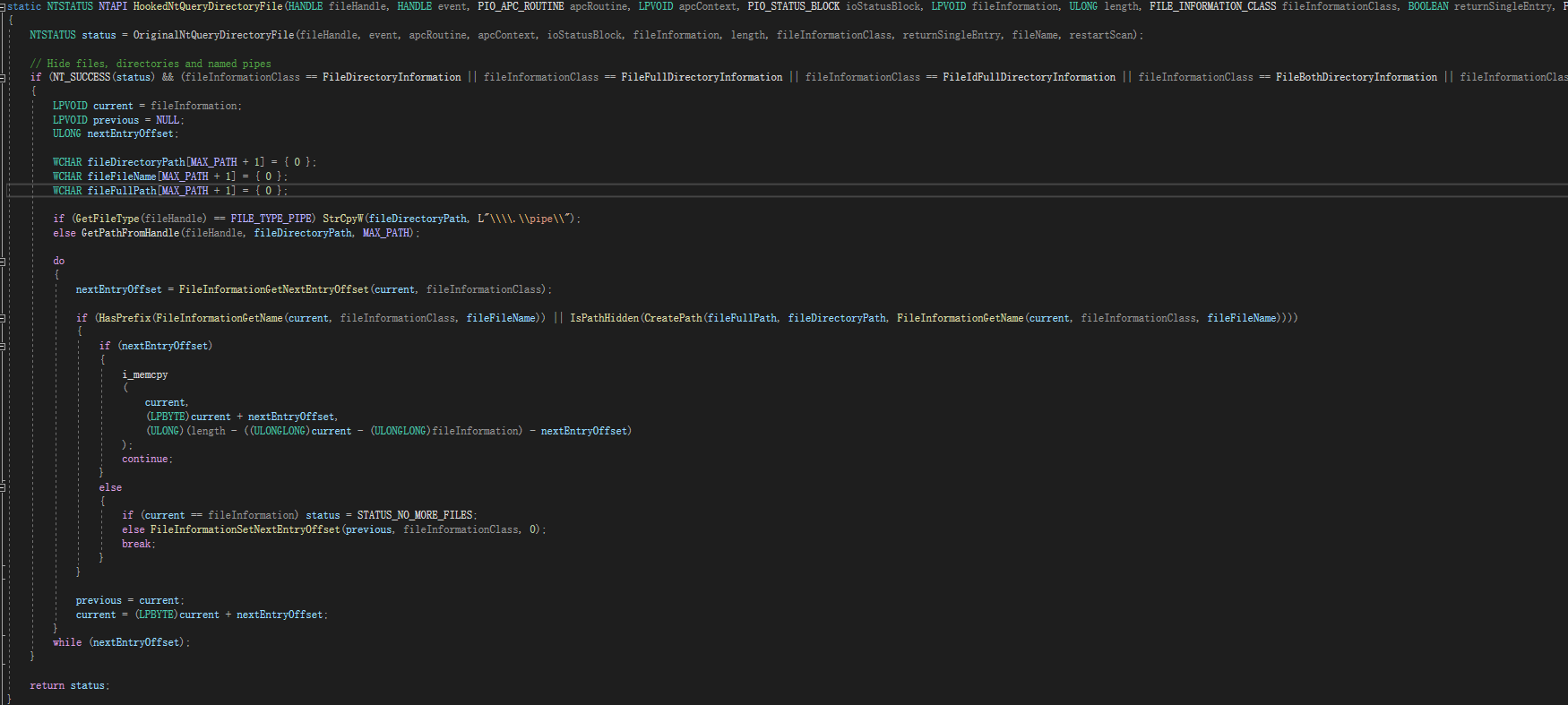
该函数是 NtQueryDirectoryFile 系统调用的钩子。主要目的是在查询目录文件时，过滤隐藏的文件、目录和命名管道，以达到隐藏特定文件的效果。

调用原始的 NtQueryDirectoryFile 函数，执行实际的目录查询操作。

确保 NtQueryDirectoryFile 的调用成功，而且查询的信息类别是文件目录信息之一（如文件信息、完整文件目录信息等）

遍历查询结果，检查每个文件的信息，如果文件需要隐藏，则从查询结果中移除该文件的信息。检查文件是否需要隐藏的逻辑主要通过 HasPrefix 和 IsPathHidden 函数来实现，这些函数可能会根据一些条件决定是否隐藏文件。

返回经过处理的查询结果或者原始的查询结果，视隐藏逻辑是否触发而定。



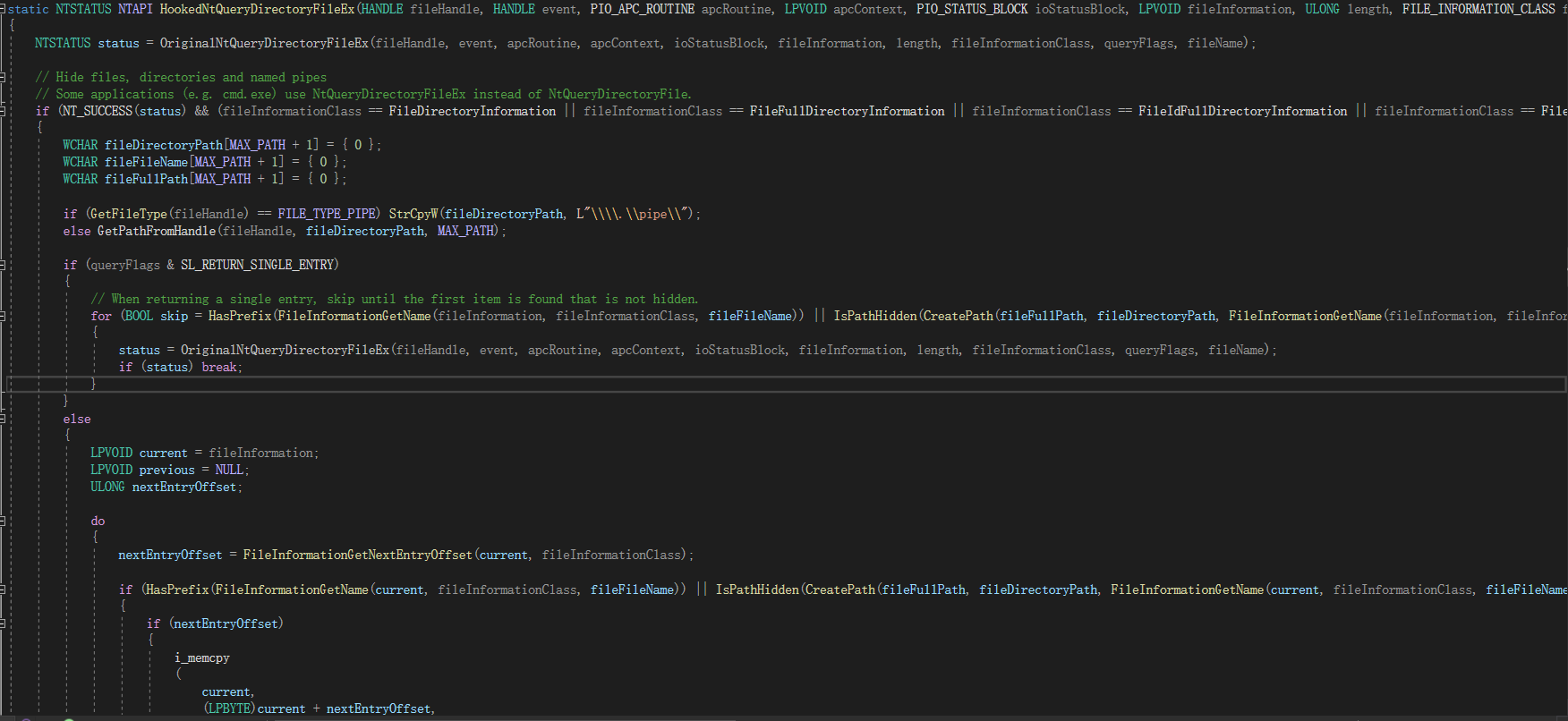
1. HookedNtQueryDirectoryFileEx

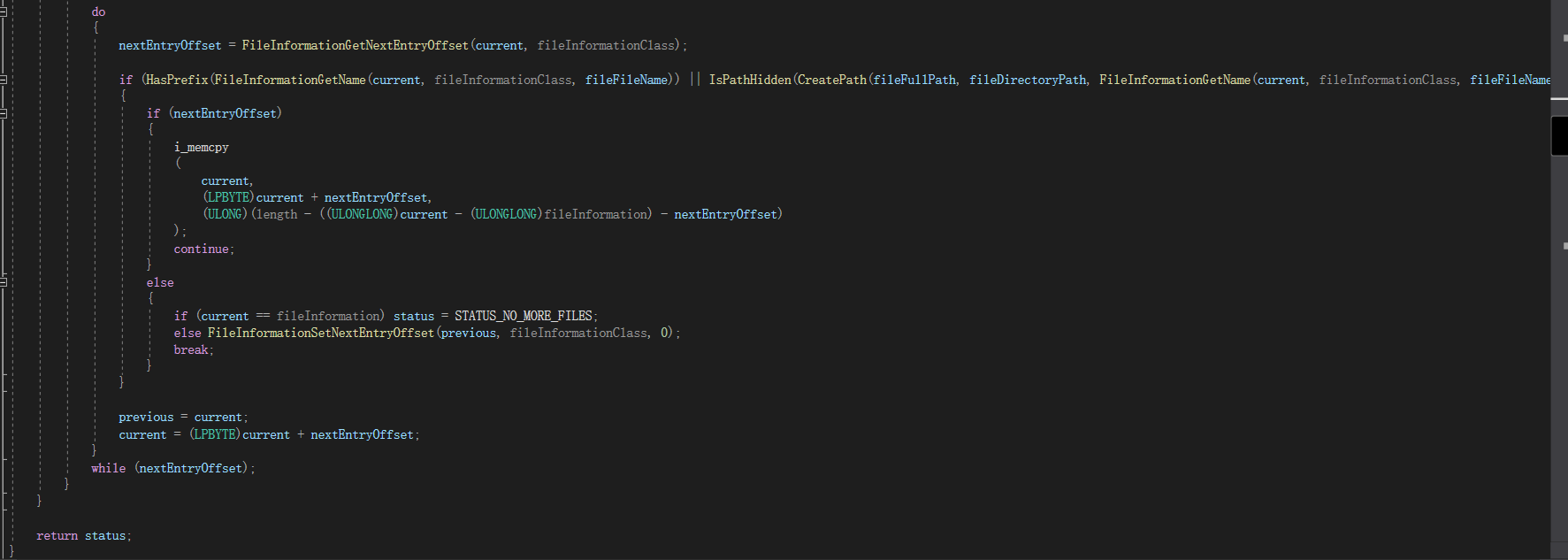
该函数是 NtQueryDirectoryFileEx 系统调用的钩子。主要目的是在查询目录文件时，过滤隐藏的文件、目录和命名管道，以达到隐藏特定文件的效果。

调用原始的 NtQueryDirectoryFileEx 函数，执行实际的目录查询操作。

确保 NtQueryDirectoryFileEx 的调用成功，而且查询的信息类别是文件目录信息之一（如文件信息、完整文件目录信息等）。

根据查询标志 (queryFlags) 的设置，执行不同的隐藏逻辑。如果设置了 SL\_RETURN\_SINGLE\_ENTRY 标志，它会跳过直到找到第一个不隐藏的项目。否则，它会遍历查询结果，检查每个文件的信息，如果文件需要隐藏，则从查询结果中移除该文件的信息。





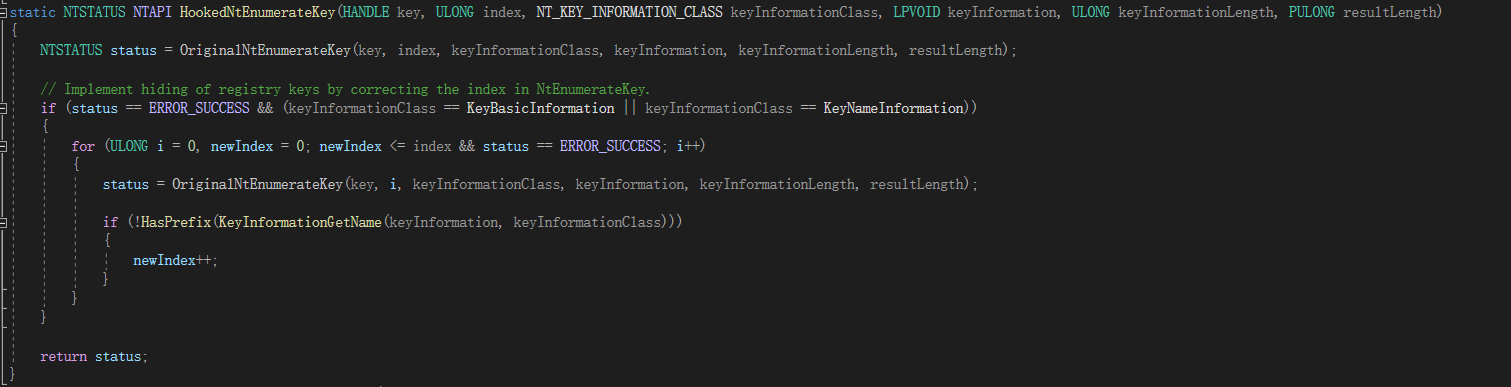
1. HookedNtEnumerateKey

用于钩住 NtEnumerateKey系统调用的函数，主要实现了对注册表键和值的隐藏功能。

调用原始的 NtEnumerateKey 获取注册表键的信息。

如果获取信息成功且信息类型是 KeyBasicInformation 或者 KeyNameInformation，则遍历注册表键，并通过 HasPrefix 函数检查键的名字是否需要隐藏。

如果需要隐藏，通过适当调整索引 (index) 的方式来跳过隐藏的键，确保返回的键信息不包含需要隐藏的键。



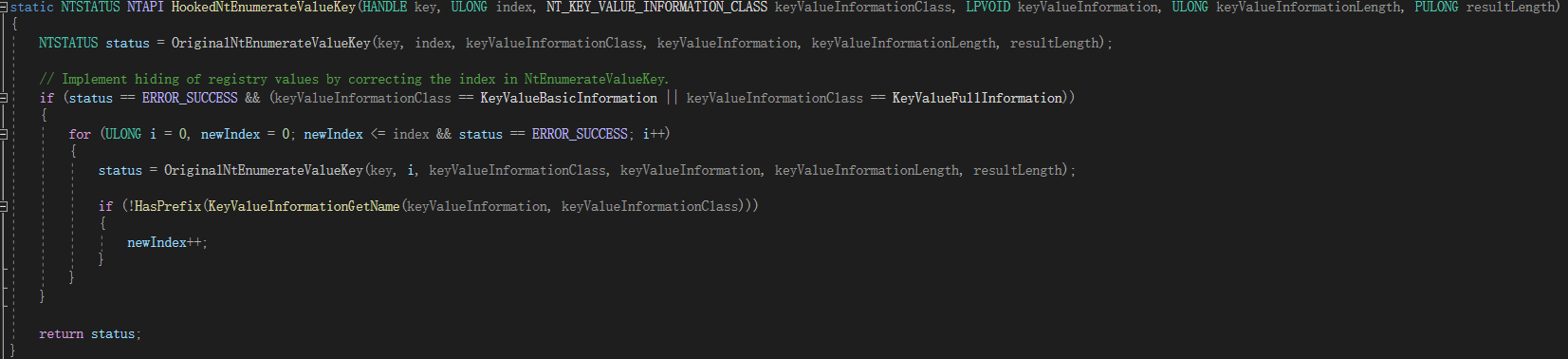
1. HookedNtEnumerateValueKey

用于钩住NtEnumerateValueKey 系统调用的函数，主要实现了对注册表键和值的隐藏功能。

调用原始的 NtEnumerateValueKey 获取注册表值的信息。

如果获取信息成功且信息类型是 KeyValueBasicInformation 或者 KeyValueFullInformation，则遍历注册表值，并通过 HasPrefix 函数检查值的名字是否需要隐藏。

如果需要隐藏，通过适当调整索引 (index) 的方式来跳过隐藏的值，确保返回的值信息不包含需要隐藏的值。

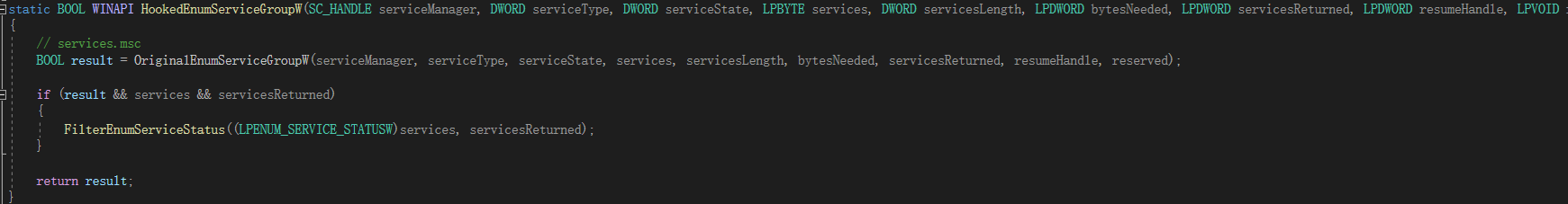


1. HookedEnumServiceGroupW

钩住了 EnumServiceGroupW 函数，该函数用于在 services.msc 中列举服务组的信息。

在调用原始的 EnumServiceGroupW 后，通过调用 FilterEnumServiceStatus 对返回的服务状态信息进行过滤。

FilterEnumServiceStatus 函数可能会检查每个服务状态的信息，根据一些条件判断是否需要隐藏或修改服务的信息。

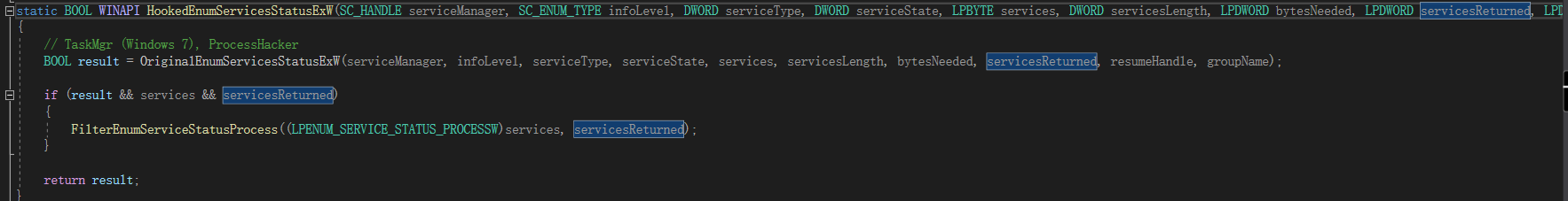


1. HookedEnumServicesStatusExW

钩住了 EnumServicesStatusExW 函数，该函数用于在 Windows 7 上列举服务的详细信息。

在调用原始的 EnumServicesStatusExW 后，通过调用 FilterEnumServiceStatusProcess 对返回的服务状态信息进行过滤。

FilterEnumServiceStatusProcess 函数可能会检查每个服务状态的信息，根据一些条件判断是否需要隐藏或修改服务的信息。

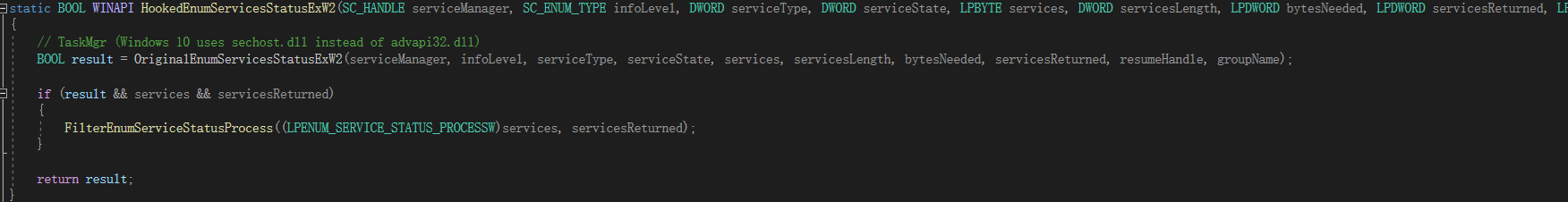


1. HookedEnumServicesStatusExW2

钩住了 EnumServicesStatusExW2 函数，该函数在 Windows 10 上用于列举服务的详细信息，使用 sechost.dll 而不是 advapi32.dll。

在调用原始的 EnumServicesStatusExW2 后，通过调用 FilterEnumServiceStatusProcess 对返回的服务状态信息进行过滤。

FilterEnumServiceStatusProcess 函数可能会检查每个服务状态的信息，根据一些条件判断是否需要隐藏或修改服务的信息。



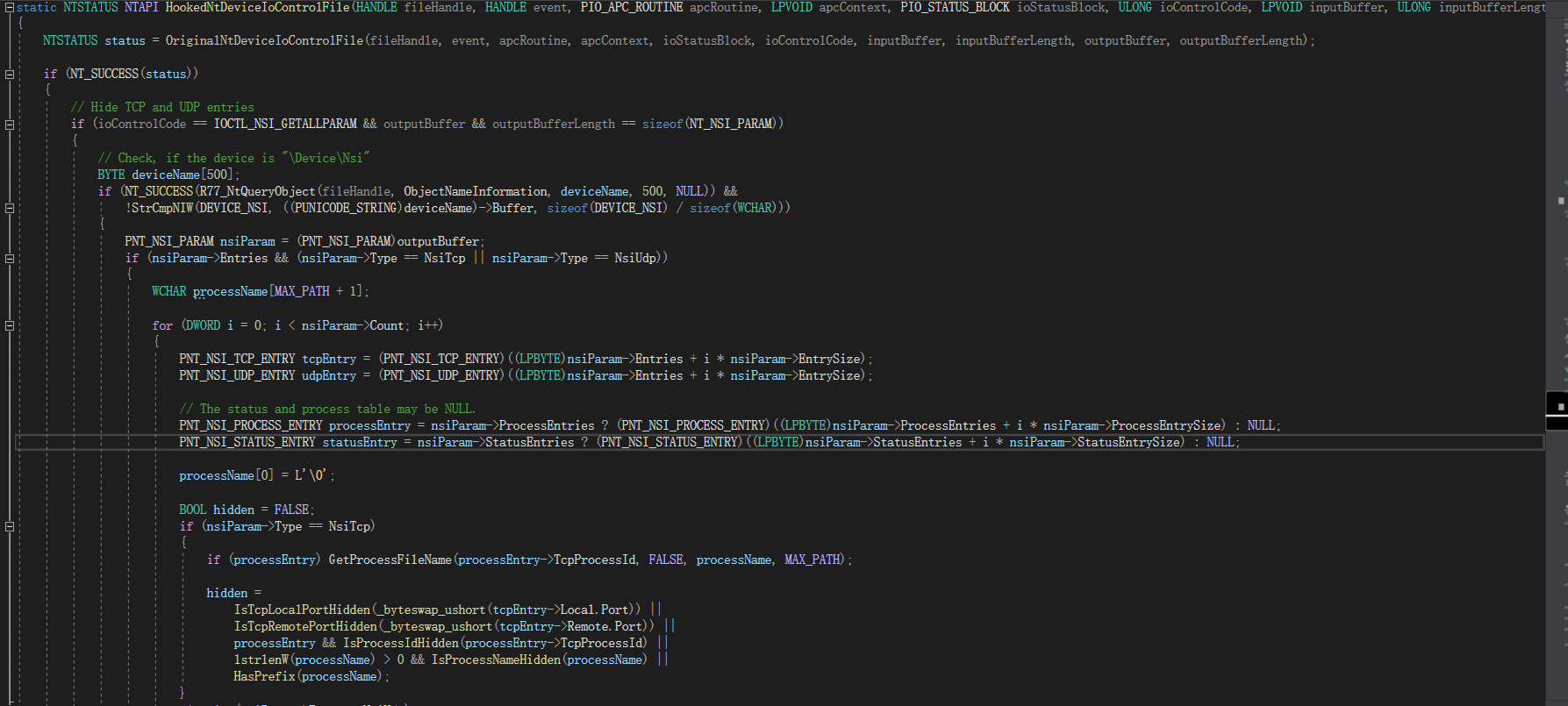
1. HookedNtDeviceIoControlFile

对 NtDeviceIoControlFile 函数的钩子，主要用于隐藏或修改通过 IOCTL\_NSI\_GETALLPARAM 控制码调用时获取的网络信息。

调用了原始的 NtDeviceIoControlFile 函数，以获取网络信息。通过传递的 ioControlCode 等参数，获取了网络信息，特别是通过 IOCTL\_NSI\_GETALLPARAM 获取了网络参数。

如果原始调用成功 (NT\_SUCCESS(status))，则继续执行后续操作。通过判断 ioControlCode 是否为 IOCTL\_NSI\_GETALLPARAM，以及输出缓冲区和其长度是否符合预期，判断是否获取了网络参数信息。如果获取了网络参数信息且属于指定的类型（NsiTcp 或 NsiUdp），则进行以下操作。

对获取的 TCP 和 UDP 信息进行逐个遍历。对每个 TCP 或 UDP 项，检查是否需要隐藏，判断的条件包括：是否本地端口或远程端口被隐藏；相关进程是否被隐藏；进程的可执行文件名是否被隐藏；是否符合预定义的前缀条件。如果需要隐藏，通过移动数组的方式将当前项移出数组，同时减小 Count 计数。这样，隐藏的项在数组中将不再存在，实现了对 TCP 和 UDP 信息的隐藏。





1. GetProcessHiddenTimes

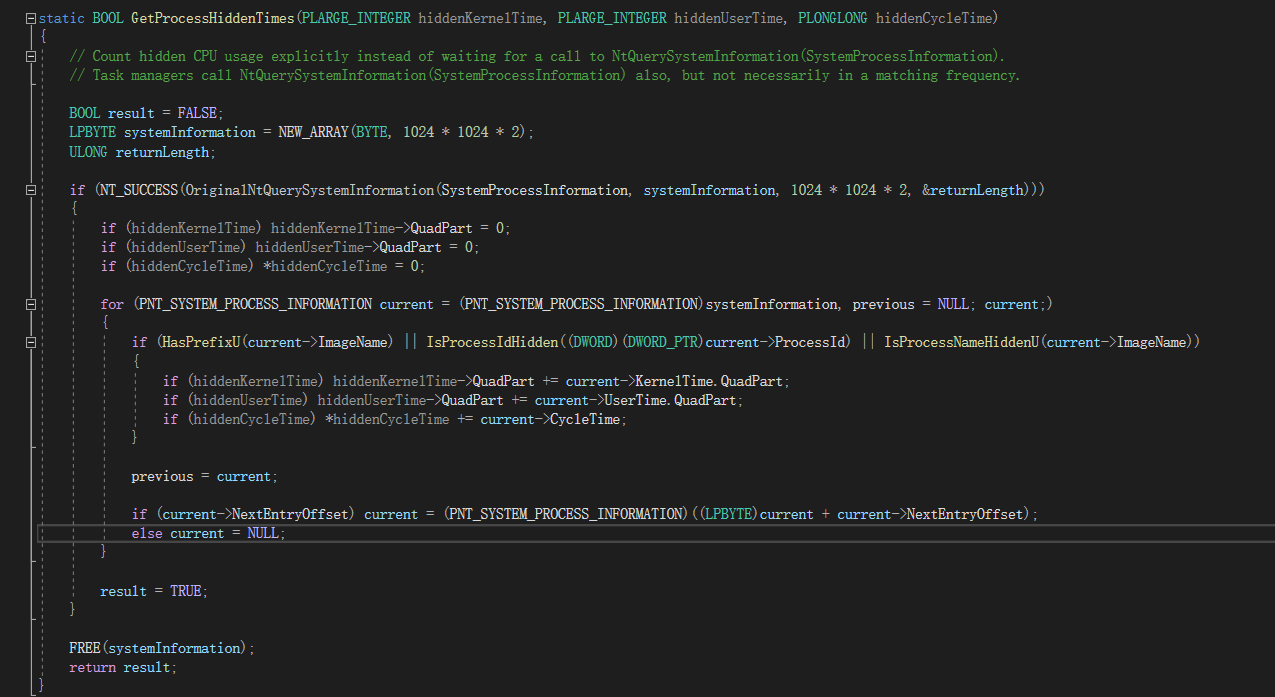
主动统计隐藏进程的 CPU 使用情况，而不是等待对 NtQuerySystemInformation(SystemProcessInformation) 的调用。

通过调用 NEW\_ARRAY 分配了一个大小为 1024 \* 1024 \* 2 字节的内存块（2 MB），用于存储系统进程信息。

调用原始的 NtQuerySystemInformation 函数，获取系统进程信息，具体是 SystemProcessInformation。如果调用成功，会得到系统中所有进程的信息。

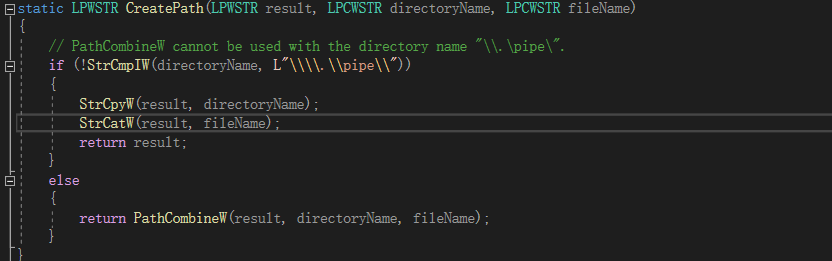
遍历获取的系统进程信息，对每个进程判断是否是隐藏进程。对于隐藏进程，累加其内核时间（KernelTime）、用户时间（UserTime）以及周期时间（CycleTime）。内核时间和用户时间是 CPU 在执行进程内核和用户模式代码时花费的时间，而周期时间是进程的总执行时间。统计得到的 CPU 使用情况即为所有隐藏进程的 CPU 使用情况之和。

释放之前分配的内存。返回一个布尔值表示统计是否成功。如果成功，隐藏进程的 CPU 使用情况已被累加；如果失败，可能是由于内存分配或 NtQuerySystemInformation 调用失败。



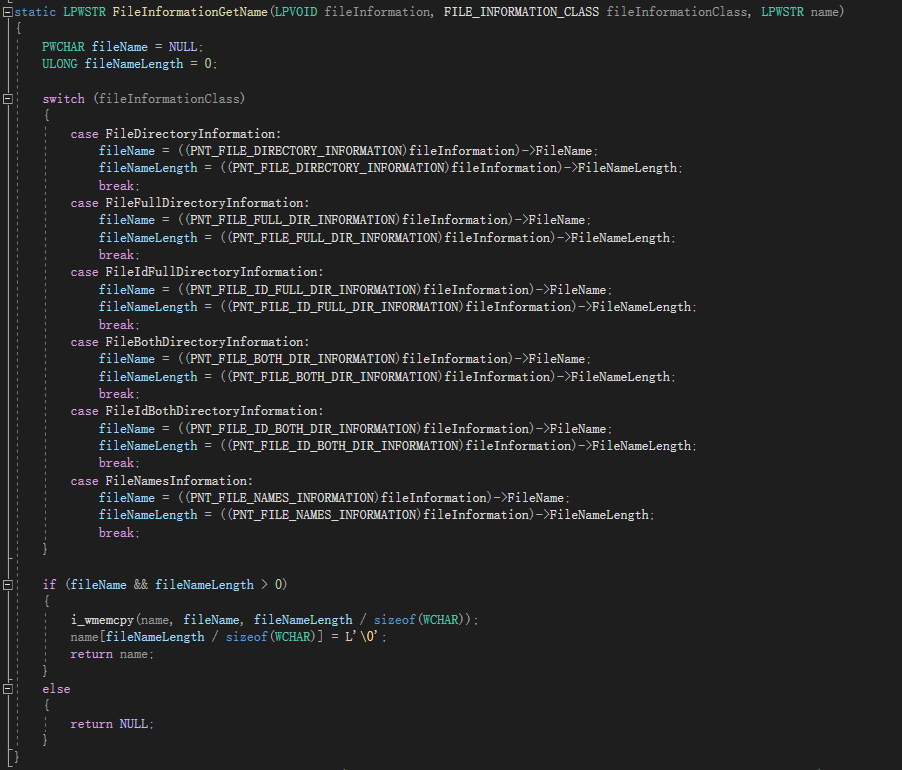
1. CreatePath

创建文件路径， 使用 PathCombineW 组合目录名和文件名，但如果目录名是 \\.\pipe\\，则直接使用字符串拼接。



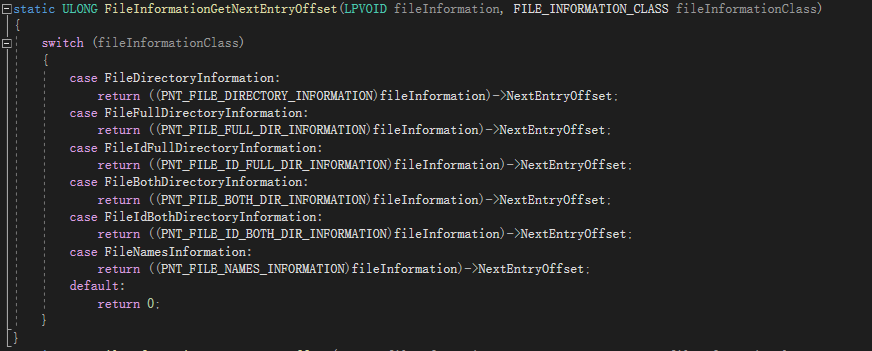
1. FileInformationGetName

从文件信息结构中提取文件名。根据文件信息类别选择对应的结构体，并提取文件名和长度。



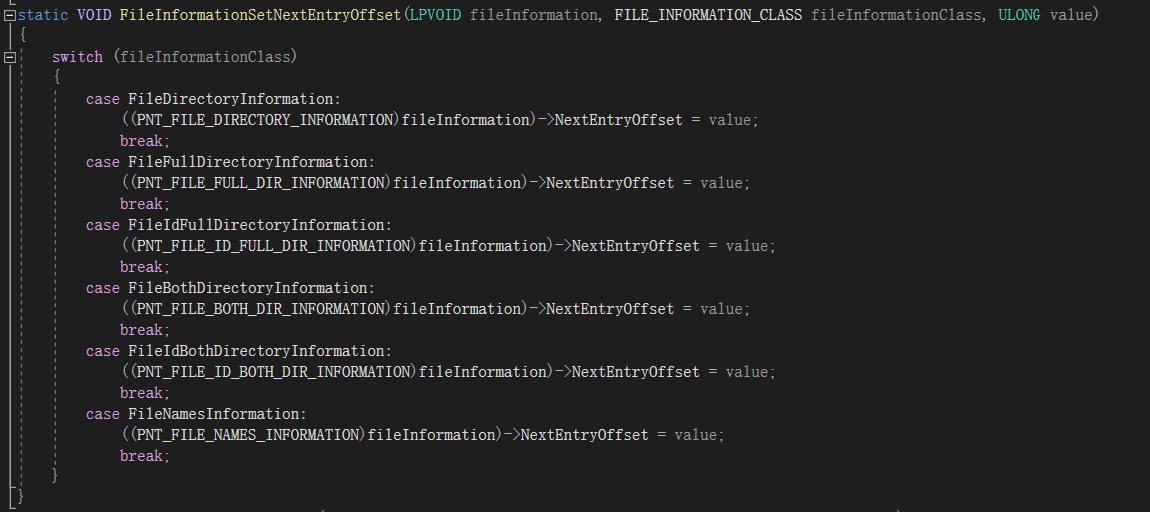
1. FileInformationGetNextEntryOffset

获取文件信息结构中的下一个条目的偏移量。



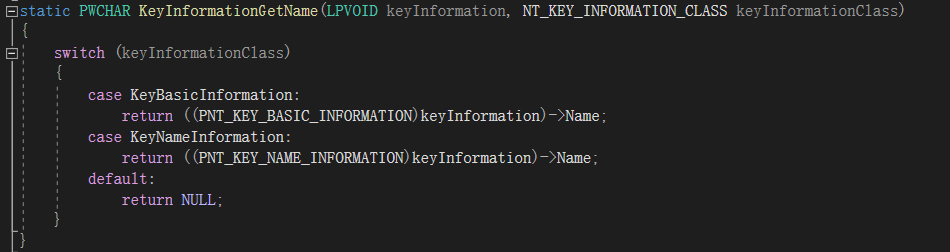
1. FileInformationSetNextEntryOffset

设置文件信息结构中的下一个条目的偏移量。



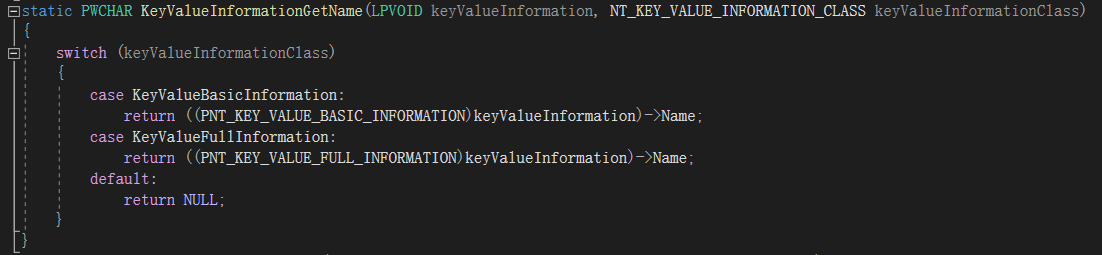
1. KeyInformationGetName

从键信息结构中提取键名。



1. KeyValueInformationGetName

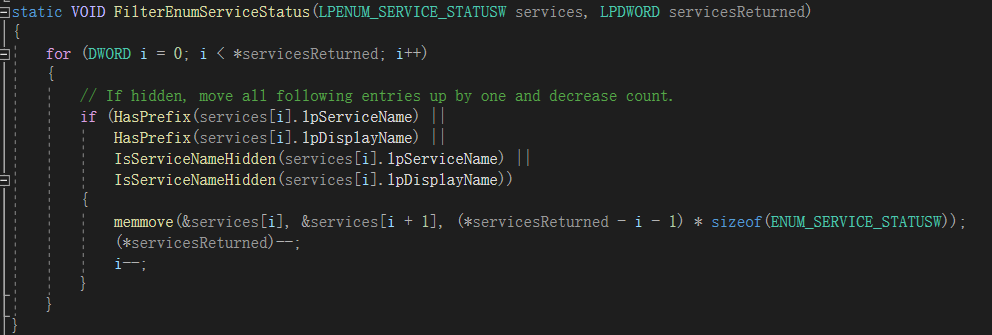
从键值信息结构中提取键值名。



1. FilterEnumServiceStatus

过滤服务状态信息，移除不需要的服务。

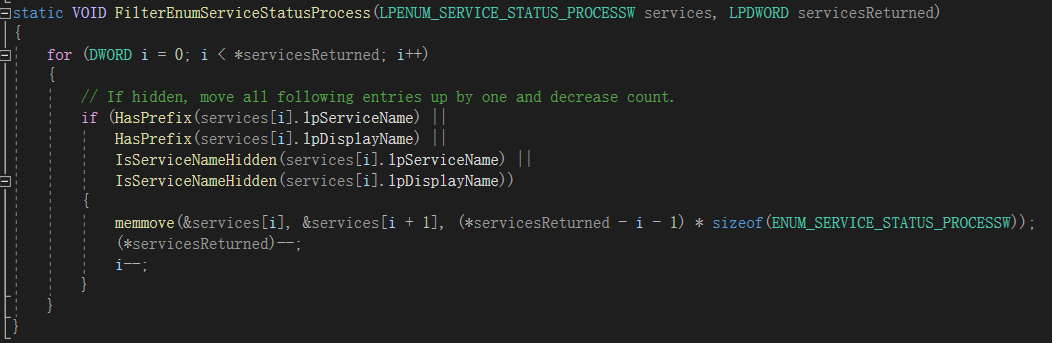
遍历服务状态信息，如果服务名或显示名以指定前缀开头或服务名被隐藏，则从列表中移除。



1. FilterEnumServiceStatusProcess

过滤带有进程信息的服务状态信息，移除不需要的服务。

与 FilterEnumServiceStatus 类似，但用于处理带有进程信息的服务状态结构。



1. **实验结论及心得体会**
2. Rootkit的设计思想： r77Rootkit的设计显然注重隐蔽性和持久性。通过修改注册表、隐藏特定进程和文件，以及使用API Hooking技术，该Rootkit试图在系统中深藏不露，不易被检测和清除。
3. 注册表的滥用： r77Rootkit通过注册表的操作实现对系统的配置和隐藏。滥用注册表项用于存储配置信息，同时利用其对于普通进程可写的特性，增加了Rootkit的难以察觉性。
4. API Hooking的应用： 通过Detours库实现的API Hooking，使r77Rootkit能够拦截和修改系统调用，进而影响系统行为。这展示了黑客在利用Rootkit时借助API Hooking对系统功能进行修改、隐藏和篡改，绕过常规的检测手段。
5. 进程和文件的隐藏： r77Rootkit通过隐藏特定的进程、文件和注册表项，实现了对自身存在的模糊掩盖。这种隐藏机制使得常规的系统监控工具难以察觉Rootkit的存在，对于渗透测试和防御来说是一项严峻的挑战。