南開大學
Nankai University

网络空间安全学院

恶意代码分析与防治技术课程实验报告

实验：Rootkit77

姓名：周钰宸

学号：2111408

专业：信息安全

2024 年 1 月 14 日

1 实验目的

1. 运行 R77 程序，实现对指定的进程、文件、注册表、网络连接的隐藏。对实验结果进行截图，完成实验报告。
2. **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**。

2 实验原理

2.1 Rootkit

Rootkit 是一种恶意软件，其主要目标是在计算机系统上植入并隐藏自己，以逃避常规的安全检测和防御机制。它通常被用于维持攻击者的长期访问权限，而不被用户或系统管理员察觉。以下是 Rootkit 的一些主要行为和特点：

1. **隐藏：** Rootkits 专注于隐藏其存在，使其在操作系统上变得不可见。这包括隐藏文件、进程、网络连接和注册表项等，以防止被检测。
2. **权限提升：** Rootkits 通常试图提升其执行的权限，以绕过操作系统的安全层级。这可能涉及到提升到管理员或系统级别的权限，以执行更深层次的操纵。
3. **持久性：** Rootkits 致力于在系统中保持长期存在。它们常常会修改系统的启动项、注册表、或其他关键组件，以确保在系统重新启动后仍然存在。
4. **后门访问：** 通常会创建后门，允许攻击者在系统上执行各种操作，例如远程访问、文件上传和下载、系统控制等。
5. **内核级操作：** 一些 Rootkits 会操作在操作系统的内核级别，这使得它们更难被检测和清除，因为它们可以绕过用户空间的安全工具。

要防范 Rootkits，用户和管理员应保持系统和安全软件的更新，定期进行安全审查，使用可信任的防病毒软件，并实施最佳的网络安全实践。

然而，rootkit 并不仅仅用于恶意目的。它们也被组织和执法机构用于监视员工，使他们能够调查机器并对抗可能的网络威胁。

2.2 Rootkit77

本次实验重点研究的 **Rootkit77 就是一种特殊的 Rootkit，也被称为 r77，是一个无文件的 Ring 3 Rootkit**。通过查阅资料，我知道了它具有如下的功能：

2.2.1 隐藏内容

它能隐藏以下所有内容：

1. 文件和目录
2. 进程和 CPU 使用情况
3. 注册表键和值

4. 服务
5. TCP 和 UDP 连接
6. Junctions、命名管道、计划任务

事实上所有以"$77" 开头的内容都会被隐藏。

2.2.2 动态配置

r77 具有动态配置系统，可以通过 PID 和名称隐藏进程，通过完整路径隐藏文件系统项目，隐藏特定端口的 TCP 和 UDP 连接等。配置位于 HKEY_LOCAL_MACHINE\\SOFTWARE\\$77config，任何进程都可以写入，无需提升权限。此外，rootkit 会隐藏 $77config 键。

R77 的部署只需要一个文件：Install.exe。执行后，R77 将在系统上持久存在，并注入所有正在运行的进程。Uninstall.exe 可以完全并优雅地从系统中移除 r77。

Install.shellcode 是安装程序的 shellcode 等价物，这样，安装可以在不放置 Install.exe 的情况下集成。shellcode 可以简单地加载到内存中，转换为函数指针并执行。

综上所述，**R77 是一个 64 位上操作系统上可以运行的 Ring3 Rootkit，出于其在只能在 64 位上运行的特殊性质，本次实验选择在 Win10 操作系统上进行实验，避免不必要的情况发生**。

2.3 Windows 的 Detours 机制

Microsoft Detours 是一个用于 Windows 平台的二进制代码注入和函数重定向的工具。Detours 允许开发人员在运行时修改二进制可执行文件中的函数行为，而无需修改原始的源代码。它通常用于实现 API 挂钩（API Hooking）和函数注入，以用于监视、修改或替换目标函数的行为，如研究恶意软件、性能分析、调试和测试等场景。

2.4 API Hooking

API Hooking 是一种在运行时截获和修改应用程序对 API（应用程序编程接口）函数的调用的技术。这种技术通常用于在不修改源代码的情况下，对程序的行为进行改变、监控或扩展。

2.4.1 应用

1. 调试和逆向工程：API Hooking 常用于调试和逆向工程，以便分析程序的行为，查看函数的输入和输出，或者截获加密算法等关键操作。
2. 性能分析：通过 API Hooking，可以监测应用程序的性能，记录函数调用的频率、耗时等信息，从而进行性能优化。
3. **安全研究：在安全领域，API Hooking 可用于监测恶意软件的活动，例如截获系统调用、检测关键函数的调用等，有助于发现和分析恶意行为**。

2.4.2 风险

1. 稳定性问题：错误的 API Hooking 可能导致程序崩溃、内存泄漏或其他不稳定的行为。特别是在涉及到复杂的应用程序和系统级别的 Hooking 时，可能会引起不可预测的后果。

2. **安全性问题：恶意软件也可能使用 API Hooking 技术，以绕过安全措施、窃取敏感信息或进行其他攻击**。因此，API Hooking 的应用需要经过审慎评估，以确保不会被滥用。
3. 法律和道德问题：在某些情况下，使用 API Hooking 可能违反软件许可协议或法规，因此使用时需要注意法律和道德准则。

3 实验过程

3.1 实验环境及工具

虚拟机软件	VMware Workstation 17 Pro
宿主机	Windows 11 家庭中文版
虚拟机操作系统	Windows 10 家庭中文版
实验工具 1	OllyDBG 2.01
实验工具 2	IDAPro 6.6.14.1224
配套工具	Python 2.7.2

表 1: 本次实验环境及工具

本次实验部分过程参照 https://www.elastic.co/security-labs/elastic-security-labs-steps-throughthe-r77-rootkit 与实验指导书 r77 Rootkit Technical Documentation。**并且本次实验相比于上次多的只是，实际对 R77 上次验证其功能后的技术分析。在每个隐藏的技术后面都有认真分析。最后也对几个病毒文件进行了简单分析与功能概述**。

3.2 实验环境搭建

在 VMware Workstation 17 Pro 通过官网下载媒体刻制光盘的软件，**自制一个 Windows 10 家庭中文版的 iso，部署到虚拟机中安装完如下图所示：**

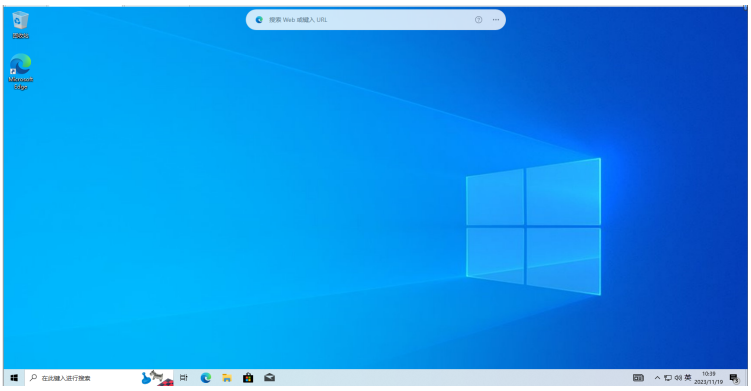


图 3.1: Windows 10 家庭中文版

花了我很长时间去安装 Win10，也占了我将近 50GB 大小空间。好麻烦，接下来去将电脑的病毒威胁检测关闭，不然压缩包一旦解压缩后，相关病毒文件会被自动删除：

图 3.2: Caption

然后我们把一会进一步分析需要的工具和程序挪入虚拟机，包含**测试隐藏进程的 Process Explorer，Procmon 和以及 TCP View**。拍好相关快照，就可以开始啦。

3.3 动态运行 R77

将 R77 压缩包放到虚拟机中，打开后查看图3.3：

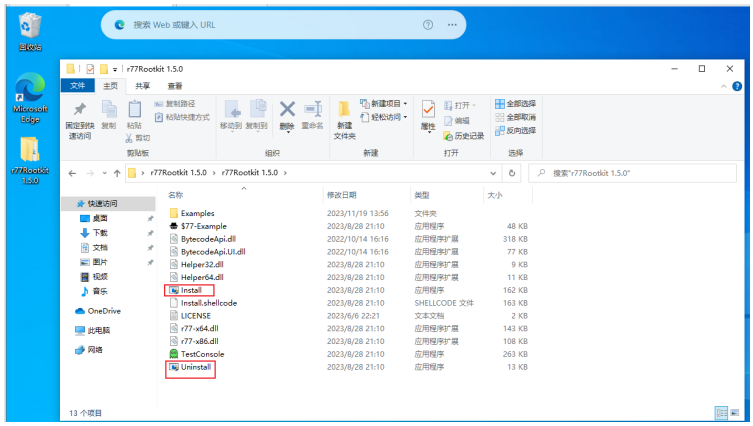


图 3.3: R77 信息

然后双击 install.exe 运行，看到可以看到原本本文件夹下的 $77-Example.exe 也不见了。这验证了它**会隐藏所有以 $77 为开头的文件，进程等的行为**。如图3.4所示。

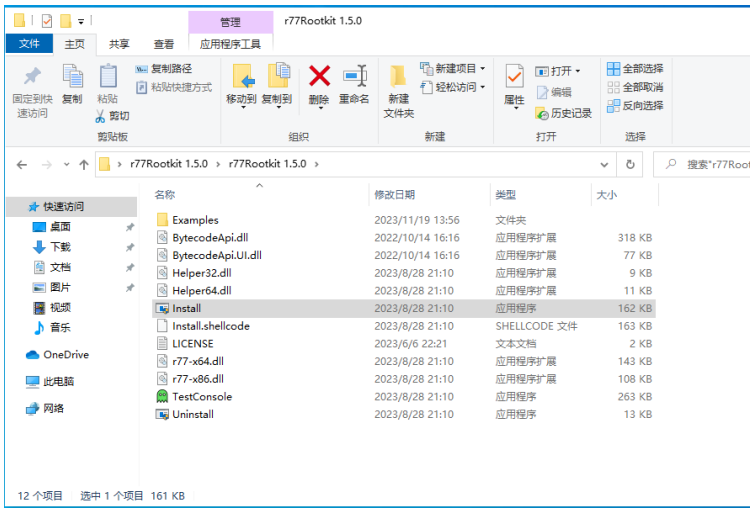


图 3.4: $77-Example.exe 消失

然后我们暂时先恢复快照，**依次对进程、文件、注册表和网络连接隐藏功能进行验证**。

3.4 进程隐藏

实际上 R77 可以实现对任意文件或者指定文件的隐藏，这里首先展示对任意文件即任意 $77 为开头的文件的隐藏：

3.4.1 任意文件隐藏

首先查看其隐藏进程的行为，利用那个给定的以 $77 为开头的 $77-Example.exe，打开运行，**为了能让他更好地体现出被隐藏但实际存在的效果，我将 CPU 占有率调为 100%!**。然后打开任务管理器：

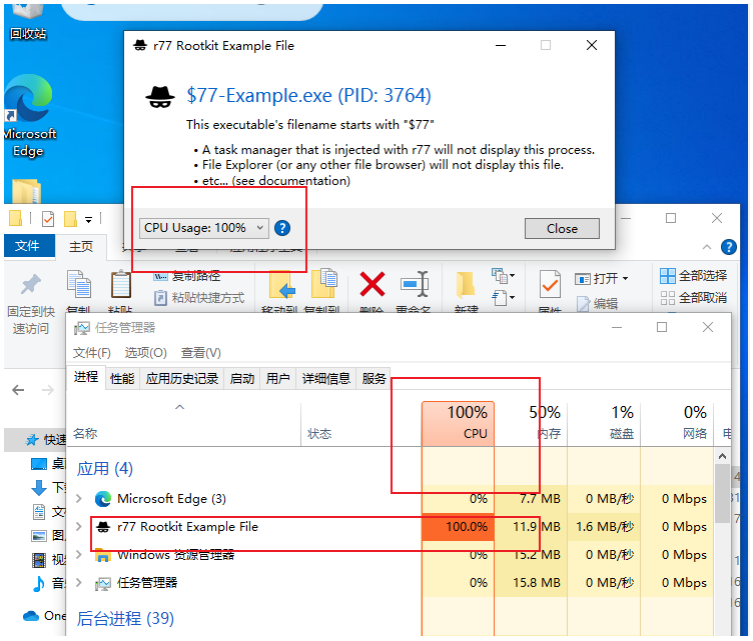


图 3.5: 运行 install 前

图3.5可以明显看到此时**这个可执行文件是完全可见的**。然后我们双击运行 install.exe：

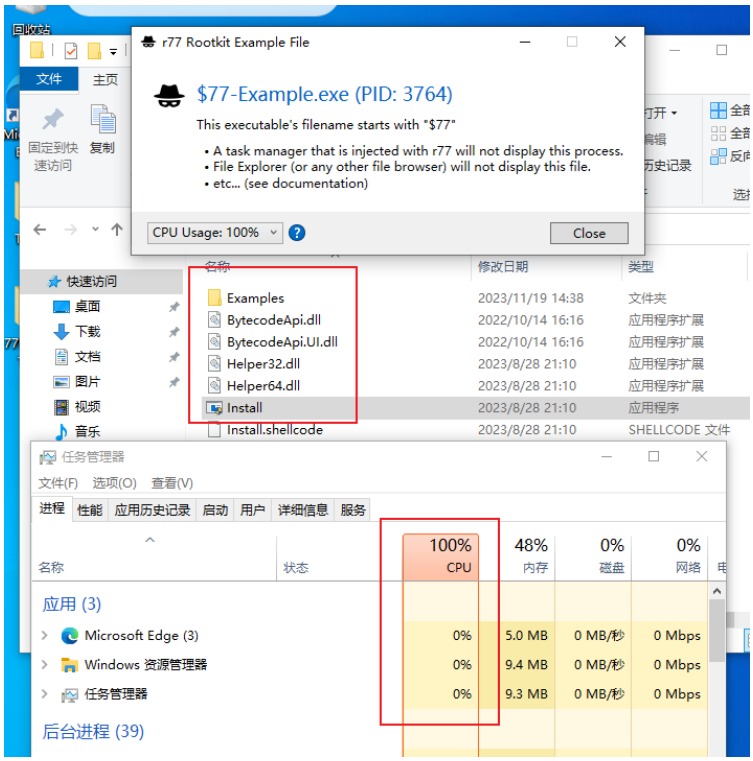


图 3.6: 运行 install 前

图3.6能明显看到此时**任务管理器中的可执行性文件 Example 已经不见了，但是由于其被我设置为了 CPU100%，我们还是能看到 CPU 此时的极高运用率，但其他三个进程都没干什么事，明显就是被隐藏了**。

除此之外，刷新文件目录，也看到 Example 文件找不到了。接下来使用 Process Explorer 查看：

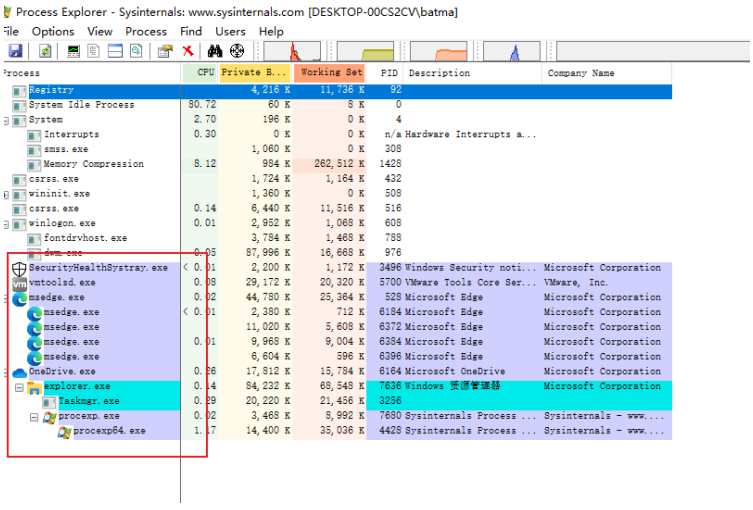


图 3.7: Process Explorer

图3.7看到此时显示的进程目录树中明显没有这个进程，确实证明其被隐藏。**不过我们接下来查看 Procmon：**

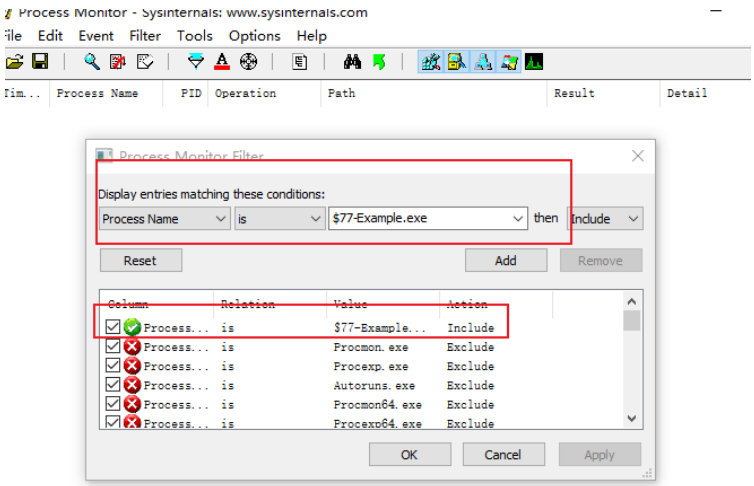


图 3.8: Procmon1

图3.8首先设置过滤器为 ProcessName is $77-Example.exe。

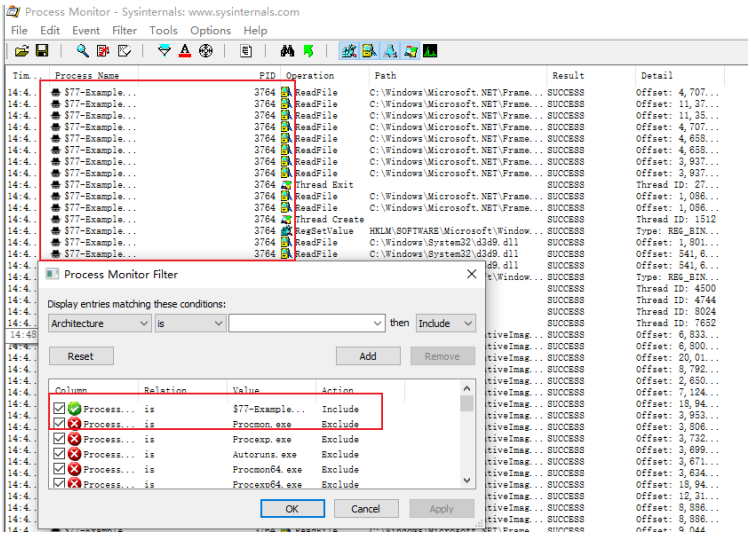


图 3.9: Procmon2

图3.9能明显看到此时**之前被隐藏的进程 $77-Example.exe 的行为被检测到了！其中包括读取文件等行文**。

这是因为如果有隐藏的进程运行，Procmon 也可以捕获它们的活动并显示其详细信息。**终究还是没逃过 Procmon 的法眼！YYDS！**

3.4.2 指定进程隐藏

除了上面说的隐藏任意 $77 开头的文件外，还可以通过 **TestConsole.exe 实现指定进程的隐藏：**

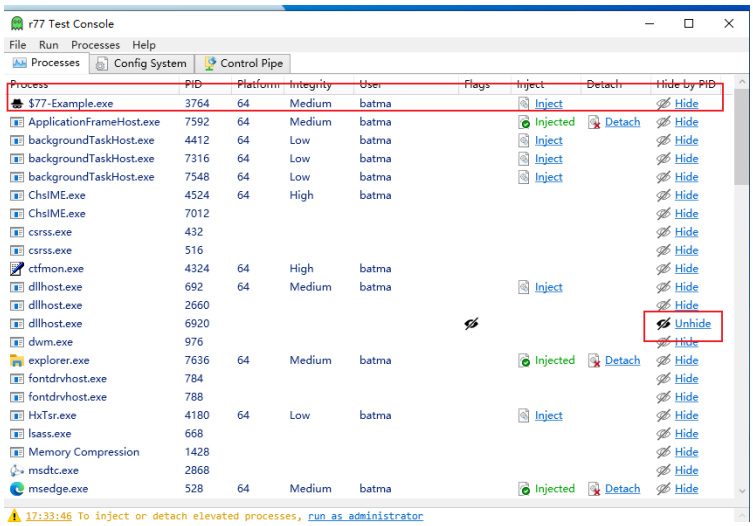


图 3.10: 查看 TestConsole

图3.10可以看到**所有进程列表，不管是被隐藏的 Example 还是别的，右边还有 Hide 选项。我这里为了实现指定进程隐藏，我把 TestConsole.exe 隐藏了**。

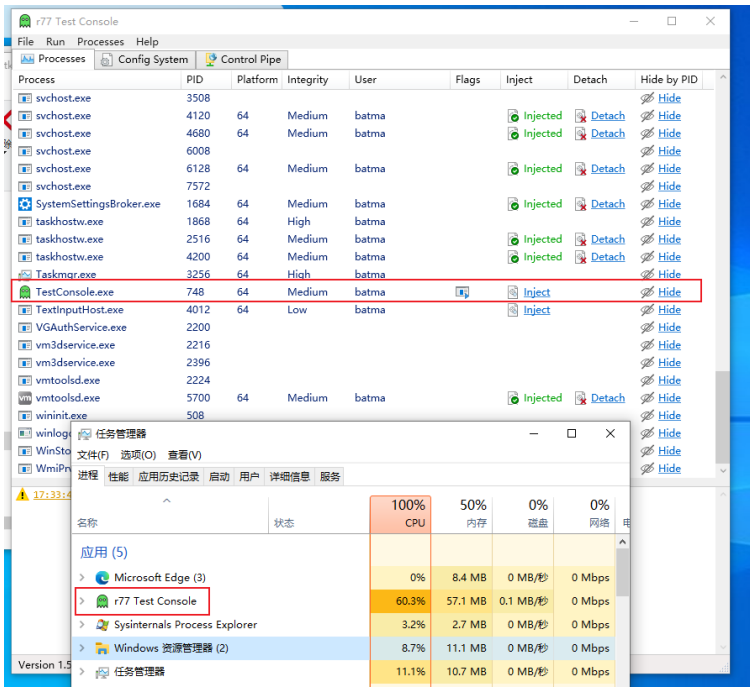


图 3.11: 隐藏 TestConsole 前

图3.11是 Hide 之前，此时还能看到任务管理器中的 TestConsole 正在运行，点击 **Hide:**

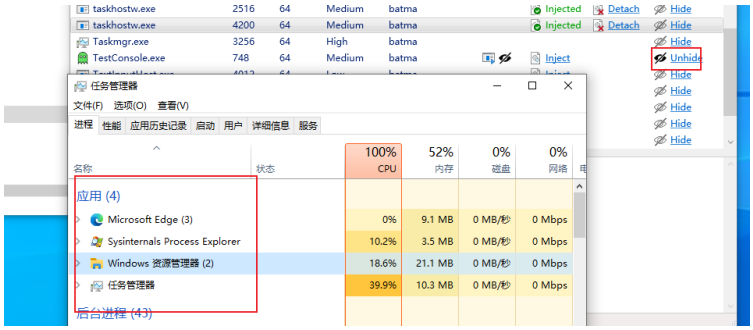


图 3.12: 隐藏 TestConsole 后

图??看到此时 Hide 变成了 UnHide，**并且任务管理器已经找不到 TestConsole 进程了，隐藏成功。 到此就实现了全部的任意或者指定进程的隐藏**。

3.5 进程隐藏原理分析

通过查阅资料等，在分析之前，我了解到 R77 实现进程隐藏具有如下的一些方式和原理：

1. **名称驱动的隐藏策略：**
   - 操作原理：R77 通过审查进程的文件名来判定是否对其进行隐藏。特别是，当可执行文件的名称以某个特定前缀（例如 $77）开头时，R77 将自动对这些进程进行隐藏。
   - 技术手段：此类隐藏通过劫持和改写系统 API 调用（这些 API 用于列举进程）来实现。在这些 API 被触发时，R77 的代码会介入，修改 API 的输出结果，使得以 $77 为前缀的进程在任务管理器等应用中不被显示。
2. **基于进程 ID 的隐蔽技术：**
   - 实施方法：R77 支持用户通过指定特定的进程 ID 来隐藏进程。这一功能是通过 R77 的配置系统实现的，用户可以将特定的进程 ID 输入到配置系统中。
   - 适用情境：此方法适用于那些不能更改其文件名的、在内存中生成的进程。
3. **替代的名称基准隐藏：**
   - 操作方式：除了使用前缀，R77 还允许用户指定具体的进程名来进行隐藏，这同样是通过配置系统来实现的。
4. **枚举与直接操作的区别：**
   - 隐藏的含义：在 R77 的环境下，隐藏意味着将被隐藏的实体从系统的枚举列表中移除。尽管如此，如果用户知道被隐藏进程的名称或 ID，他们依然可以直接对这些进程进行访问。
   - 关键提示：R77 并没有拦截用于打开文件或进程的函数，因此即便是被隐藏的实体，如果通过名称或 ID 直接访问，也不会触发“未找到”的错误。
5. **反射式 DLL 注入技术：**
   - 实现形式：R77 存在于两种不同的形式：r77-x86.dll 和 r77-x64.dll，分别对应 32 位和 64 位的进程。R77 采用了反射式 DLL 注入技术，即直接将 DLL 文件注入到远程进程的内存中，无需写入硬盘。

- 注入流程：这种注入方式不会使 DLL 在进程的 PEB（进程环境块）中显示。它通过调用 ReflectiveDllMain 来加载 DLL，并进一步调用 DllMain。

6. **进程创建时的拦截：**

- 启动机制：一旦 R77 服务被激活，它会注入到所有当前运行的进程中。R77 通过劫持 NtResumeThread 函数来实现对新进程创建的监控。
- 注入时刻：这种策略确保了在新进程开始执行任何指令之前，R77 已经被注入。这一点对于那些如 RegEdit 等程序来说至关重要，因为这些程序在启动后会迅速进行初始化，并执行枚举操作，快速展示结果。

另外提到**反射式 DLL 注入代码，其一般具有如下的凡是和步骤：**

1. **内存预留**：利用 VirtualAllocEx 函数在目标进程空间中预留出一块内存区域，这部分内存将被用于存放 DLL 文件。
2. **写入 DLL:** 通过 WriteProcessMemory 函数，将 DLL 文件的数据传输到之前在远程进程中预留的内存区域。
3. **定位启动函数：**应用 GetReflectiveLoaderOffset 函数来确定 DLL 内部 ReflectiveLoader 函数的相对位置。
4. **线程生成与执行：**在目标进程内部启动一个新线程，该线程负责执行 ReflectiveLoader，以此来激活并运行 DLL。
5. **执行监控：采用等待机制**。运用 WaitForSingleObject 函数监控新生成线程的执行过程，确保其顺利完成。

通过这一系列操作，反射式 DLL 注入的隐蔽性得到了增强。这主要是因为该方法在目标进程内生成一个专门的线程来启动 DLL 的内部自启动代码，而非依赖于常规的外部加载方式。这种独特的注入策略使得整个过程更为隐秘，难以被常规监测手段发现。

3.6 文件隐藏

然后介绍 R77 对文件隐藏的功能，**这个功能的实现与 API Hooking 技术有关，具体而言，它通过拦截和修改系统界别的文件系统调用实现**。这样让特定文件或目录在操作系统的标准文件或者浏览工具比如 Windows 资源管理器中不可见。

具体而言**文件夹的隐藏也是通过必须 $77 为文件名开头才能隐藏**。首先 uninstall 恢复之前状态，然后：

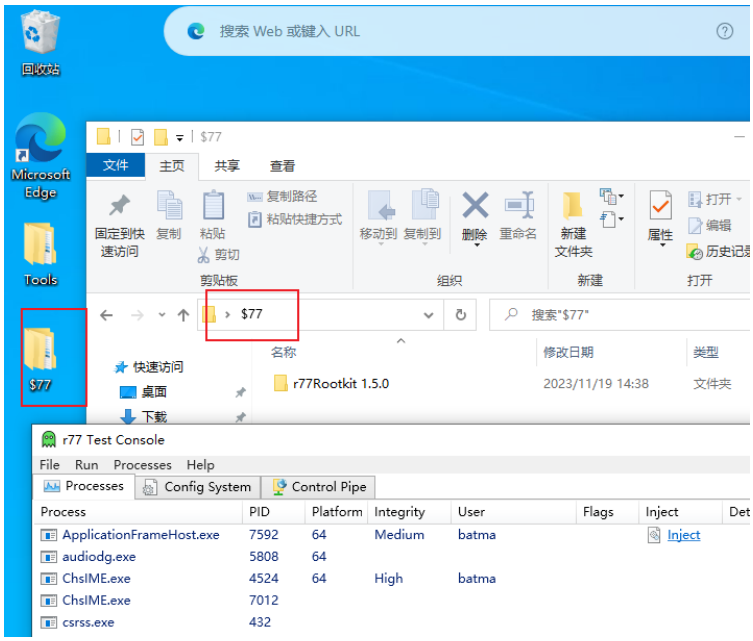


图 3.13: 隐藏 R77 文件本身

图3.13看到我**正在整活，我打算把整个 R77 文件隐藏了（笑）**。

但是为了防止隐藏后我找不到整个文件没法 uninstall，我先改好名字为 $77，然后打开 TestConsole.exe，接下来就是 install 表演时刻了！

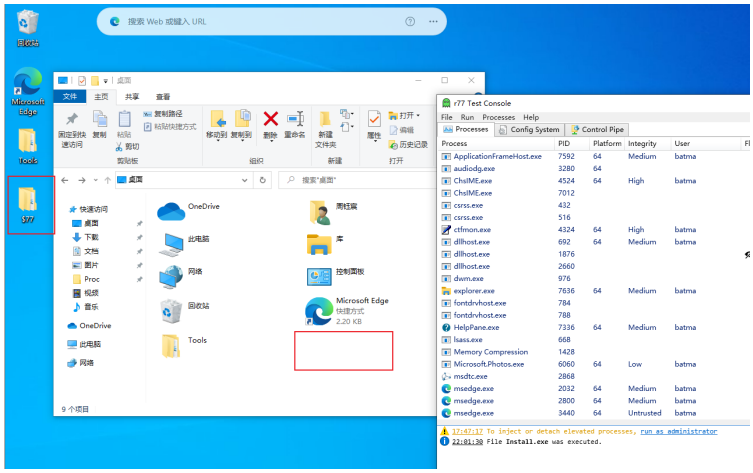


图 3.14: 文件隐藏

刷新后，如图3.14看到**装着整个 R77 的文件夹 $77 已经在文件资源管理器中不见了，但是桌面上还能看到 hhh。算是个新奇的发现，不过文件隐藏成功！**

后面发现想要 Uninstall 时候恢复不了了 hhh，报错“没有更多文件”，可能真是找不到了，幸苦我存了快照。

3.7 文件隐藏原理

同样地，通过查阅资料，我了解到了 R77 文件隐藏机制有一些的实现原理：

1. **劫持文件系统核心函数：**

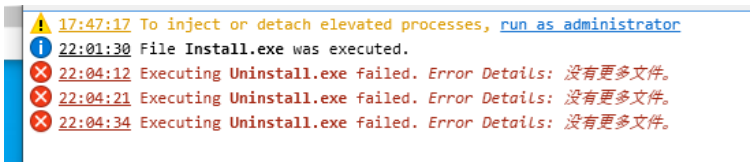


图 3.15: Uninstall 不成功

- 核心策略：R77 通过劫持 Windows 核心文件系统函数，例如 NtQueryDirectoryFile 和 NtQueryDirectoryFileEx，来达成文件和目录的隐匿。（NtQueryDirectoryFile 是 Windows 系统中的关键文件系统函数，用于列举目录中的文件和子目录。）
- 函数作用：这些函数在文件系统中发挥着至关重要的作用，主要用于列举目录内容，包括文件、目录、符号链接和命名管道。
- 隐藏机制：R77 的文件隐藏核心在于拦截并改变这些函数的行为。当操作系统或应用程序调用这些函数时，R77 介入并修改其操作方式。

2. **修改枚举结果：**
   - 操作过程：在成功拦截这些函数后，R77 会审查每个被枚举的项目（文件或目录）的名称。
   - 隐藏条件：对于那些以特定前缀（如 $77）命名的文件或目录，R77 会将它们从 API 调用的返回结果中剔除。
   - 隐藏效果：这种操作意味着，即便文件或目录实际存在于文件系统中，它们也不会在通过这些 API 函数枚举的结果中显示，例如在 Windows 资源管理器或命令行界面中。
3. **特定路径的隐蔽支持：**
   - 精细控制：除了自动隐藏以特定前缀命名的文件和目录，R77 还支持通过其配置系统进行更精确的隐藏控制。
   - 用户配置：用户可以指定任何文件或目录的完整路径，并将其添加到 R77 的配置中，以此来指示 R77 隐藏这些特定路径的文件或目录。
4. **隐藏效果的实现：**
   - 不可见性：通过这种方法，被隐藏的文件和目录在使用标准工具（如资源管理器、命令行工具等）进行文件系统浏览时不可见。
   - 直接访问：重要的是，这种隐藏只影响枚举操作。如果用户知道被隐藏文件或目录的确切路径，他们仍然可以直接访问它们。
5. **安全性与隐蔽性的考虑：**
   - 隐蔽性强度：R77 的这种隐藏方法提供了极高的隐蔽性，因为它在操作系统的核心层面上直接修改文件系统的行为。
   - 安全隐患：同时，这种能力也揭示了潜在的安全风险，特别是当这种技术被恶意软件使用时，它可以用来隐藏其痕迹，从而使检测和移除变得更加困难。

综上所述，R77 的文件隐藏技术通过巧妙地劫持和修改关键文件系统函数的行为来实现，这种方法不仅高效而且隐蔽，展示了 R77 在操作系统层面的深度集成和强大的操控能力。

而上面**提到的 NtQueryDirectoryFile，**一般来说实现方式具有如下的特征和步骤：

1. **原函数调用：**首先调用原始的 NtQueryDirectoryFile 函数。
2. **状态检查：**检查函数调用的返回状态，确认是否有文件或目录被枚举。
3. **内容过滤：**在 FilterDirectoryContents 函数中，根据特定条件（如文件名前缀）过滤掉不希望显示的文件或目录。

通过这种方式，可以在文件系统级别控制哪些文件和目录对用户可见。

3.8 注册表隐藏

接下来验证 Rootkit77 的注册表隐藏功能,其配置信息存储在 HKEY_LOCAL_MACHINE\SOFTWARE$77con 中，并且可以在未提权状态下由任何进程写入。这个键的 DACL 被设置为可以给任意用户授予完整访问权 1。“$77config” 键在注册表编辑器被注入了 Rootkit 之后会自动隐藏。

并且它还可以实现以对任意以 $77 为开头的注册表键进行隐藏。

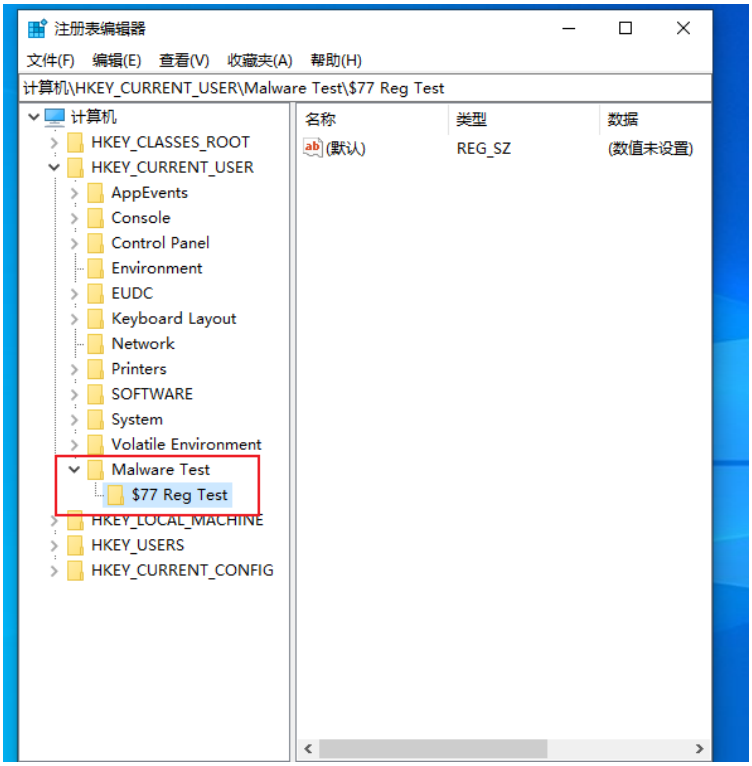


图 3.16: 创造以 $77 为开头的注册表键

图3.16所示，我打开注册表编辑器，然后创建在 HKEY_CURRENT_USER 下的一个注册表项 Malware Test 下的一个注册表项 **$77 Reg Test，以 $77 开头哦**。然后 Install 后刷新注册表：

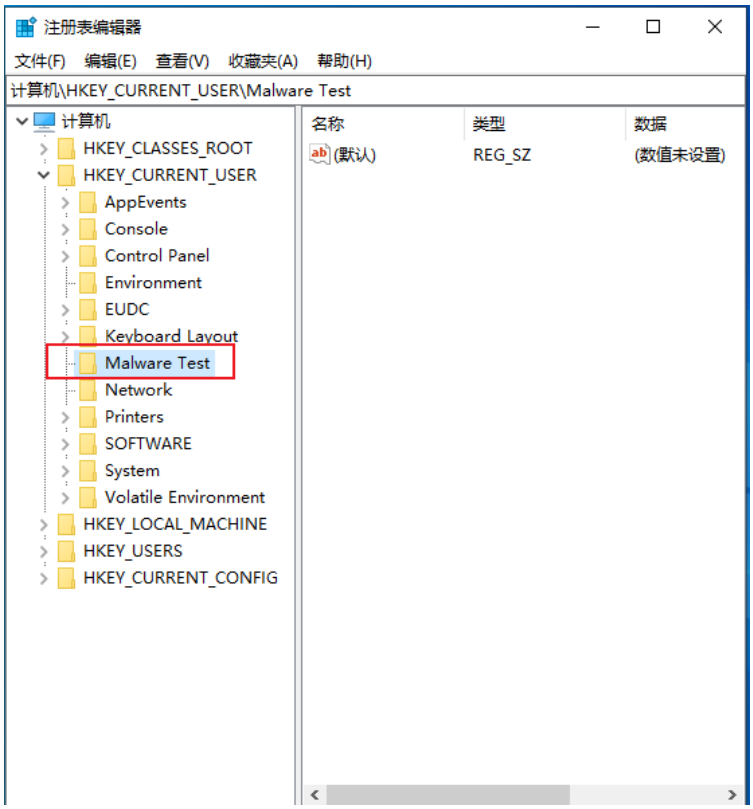


图 3.17: 注册表项被隐藏

图3.17展示了新的结果，**可以看到 Malware Test 下没有任何别的目录了，证明了 $77 Reg Test 被隐藏，验证成功**。

3.9 注册表隐藏原理

再次也是通过查阅资料，**得到了一些对于其实现注册表隐藏的认识：**

1. **核心注册表函数的劫持：** R77 对两个关键的内核级注册表函数进行了劫持：NtEnumerateKey 和 NtEnumerateValueKey。这些函数在 Windows 系统中负责遍历注册表键和值，是许多系统和应用程序访问注册表的核心。

2. **枚举过程的改造：** 在调用这些函数时，R77 首先执行标准的枚举流程。接着，它会审查每一个被枚举的注册表项和值，识别出那些需要隐藏的项（比如，那些名称以 $77 开头的注册表项或值）。

3. **重构枚举索引：** 为了确保这些注册表项和值被隐藏，R77 重新计算了剩余项的枚举索引，使得这些隐藏项在返回的列表中不被显示。这样，即便这些注册表项或值在系统中实际存在，它们也不会在标准 API 的枚举结果中出现。

上述提到的 NtEnumerateKey 函数，也是一般来说在及性能遍历注册表的时候会有如下的步骤：

1. **执行原函数：** 首先执行原版的 NtEnumerateKey 函数

2. **结果验证：** 检查函数调用是否成功执行。

3. **注册表键过滤：** 在 FilterRegistryKeys 函数中，根据特定条件（例如键名前缀）过滤掉不希望显示的注册表键。

通过这种方式，R77 在遍历注册表项时实现了精细的控制，有效地隐藏了特定的注册表项。

3.10 网络连接隐藏

最后我们 Uninstall 前面的 install，然后**验证一下 R77 对网络连接例如 TCP 和 UDP 的隐藏**。

3.10.1 UDP

直接使用 Edge 打开，搜一些奇怪的东西：

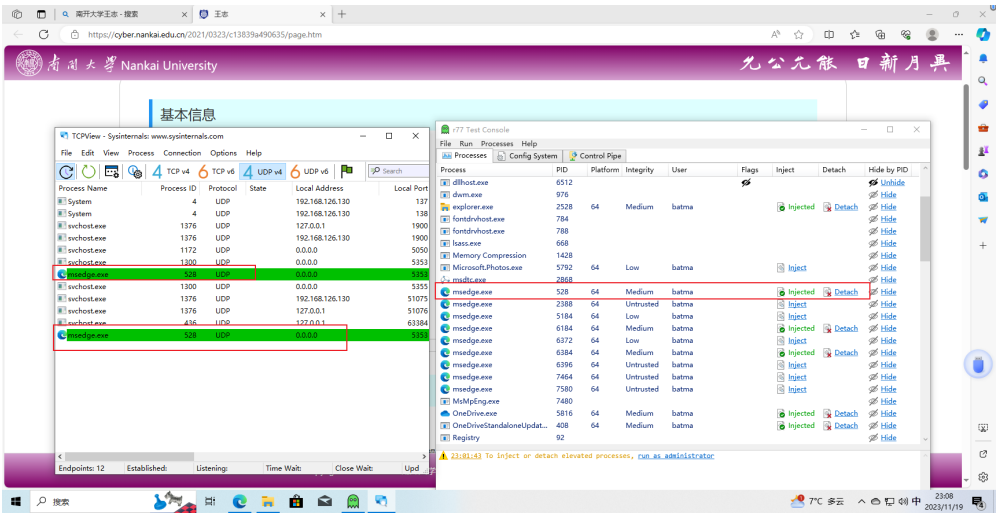


图 3.18: UDP 隐藏前

图3.18同时打开 TestConsole.exe 和 TCPView，先选中 UDP v4，查看可以看到**一个正在运行在 PID=528 的 Edge 进程，点击 Hide：**

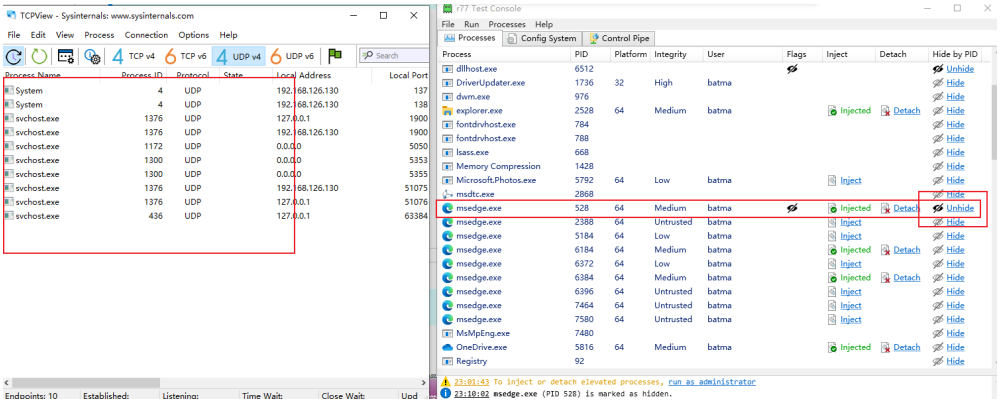


图 3.19: UDP 隐藏后

图3.19看到此时**刚才 UDP 连接的 Edge 的 UDP 连接进程已经消失了，证明 UDP 隐藏成功**。

3.10.2 TCP

接下来验证 TCP，同样观察一下，发现一个：

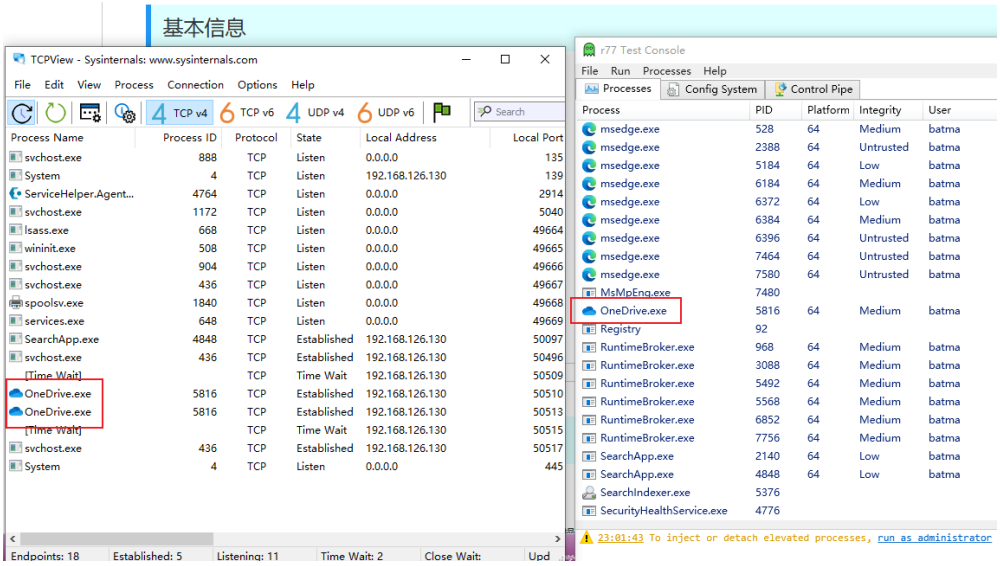


图 3.20: TCP 隐藏前

图3.20看到**有一个运行在 PID=5816 的 TCP 进程 OneDrive.exe，然后在 TestConsole 中也有**，现在 Hide 它：

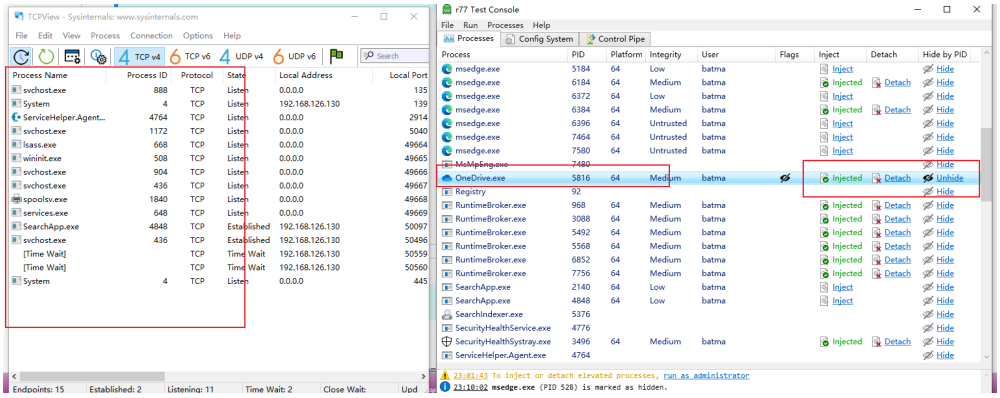


图 3.21: TC 隐藏后

图3.21看到 Unhide，并且 **TCPView 中的这个 OneDrive.exe 已经被隐藏了。到此验证了隐藏 TCP 和 UDP 的网络功能了。**

3.11 网络连接隐藏原理

关于 **R77 中 TCP 和 UDP 连接隐蔽技术**，我也通过查阅资料，得到了下面的初步结论：

- **网络驱动 I/O 控制函数的截取：**R77 实现了对执行网络 I/O 控制操作的 NtDeviceIoControlFile 函数的截取。这个函数通常用于向网络驱动程序请求数据。
- **识别 TCP/UDP 连接查询：**当此函数被用于向\Device\Nsi 驱动程序请求 TCP 和 UDP 连接数据时，R77 将进行干预。
- **连接数据的调整：**R77 会审查并调整返回的 TCP 和 UDP 连接数据。对于那些需要被隐藏的连接，R77 将它们从数据列表中剔除，并重新排列剩余连接，同时调整报告的连接总数，以保持数据的一致性。

一般来说，隐藏网络连接会有如下关键的步骤：

1. **截取网络 I/O 控制函数：**截取 NtDeviceIoControlFile，这是一个处理各种设备 I/O 控制请求的函数。对于 TCP/UDP 连接的隐藏，关键在于识别出查询 TCP/UDP 连接列表的特定 I/O 控制代码。

2. **状态和 I/O 控制代码的检查：**在调用原函数后，会检查返回状态以确认操作成功，并检查 I/O 控制代码以确定是否是查询 TCP/UDP 连接的请求。

3. **处理连接数据：**如果是 TCP/UDP 连接列表请求，一般会调用 FilterNetworkConnections 函数来处理连接数据。在此函数中，它会遍历连接数据，对每个连接应用判断逻辑来决定是否隐藏该连接。

4. **隐藏特定连接：**对于需要隐藏的连接，代码通过移动后续连接来覆盖当前连接，并更新连接数据的计数。这种方法确保隐藏的连接不会出现在返回给调用者的数据中。

通过上述方法，R77 能够在系统级别有效地隐藏特定的 TCP 和 UDP 连接，使其在网络监控工具中不可见，展示了 R77 在网络通信方面操纵操作系统行为的能力。

3.12 Install.exe 分析

首先使用一些**常用的静态和动态分析工具对病毒进行简要的分析**，分析出究竟是怎么实现上述功能。

首先使用 PEiD 加载其 install.exe，注意到：

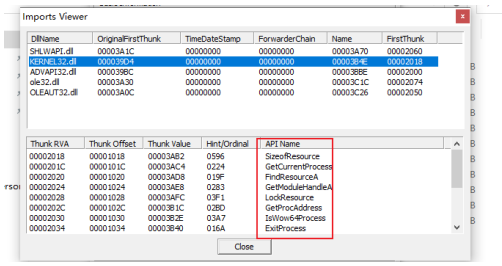


图 3.22: install.exe 导入表

图3.22看到**其有许多寻找和加载资源节的函数，具体而言是 LoadResource 这些**。然后我们也确实在 PEiD 中看到了资源节：

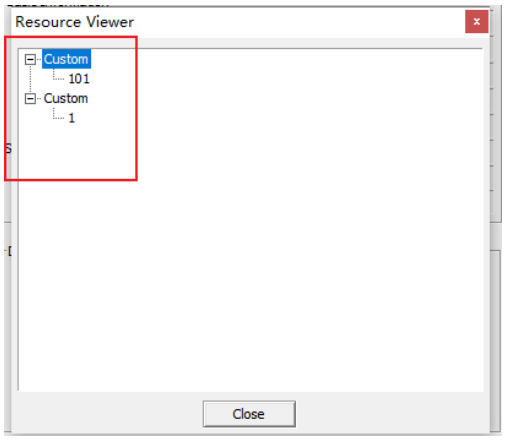


图 3.23: 资源节

然后我们将其拖入 IDA 中分析：

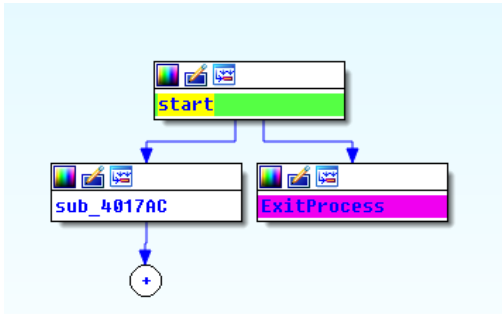


图 3.24: IDA 分析

图3.24看到**明显最重要的函数是 sub_4017AC。过去看看：**

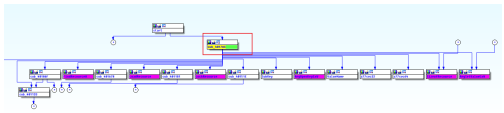


图 3.25: 函数调用链

其整体函数调用链如图3.25所示，我们查看其具体内容：

```
push    offset Type     ; "EXE"
push    65h             ; lpName
xor     ebx, ebx
push    ebx             ; hModule
call    ds:FindResourceA
mov     esi, eax
test    esi, esi
jz      loc_401869
push    edi
push    esi             ; hResInfo
push    ebx             ; hModule
call    ds:SizeofResource
mov     edi, eax
test    edi, edi
jz      loc_401868
push    esi             ; hResInfo
push    ebx             ; hModule
call    ds:LoadResource
test    eax, eax
jz      short loc_401868
push    eax             ; hResData
call    ds:LockResource
mov     esi, eax
lea     eax, [ebp+phkResult]
push    eax             ; phkResult
push    0F013Fh         ; samDesired
push    ebx             ; ulOptions
```

图 3.26: sub_4017AC

我们能看到图3.26中**有很多对其资源节进行加载的函数，另外它还使用 RegOpenKeyExW 打开注册表键 HKEY\LOCAL\MACHINE\SOFTWARE**。随后会调用 **sub_40186F, 其中包含很多 powerShell 相关的内容**。

下面将会依次重点解析这五个调用函数，它们分别是 sub_40186F，sub_401678，sub_401133，sub_4011B1 和 sub_40151E。

1. **sub_40186F:**

   - 内存分配策略：GetProcessHeap 和 HeapAlloc 被用于从进程的堆空间中分配内存，这部分内存用于储存构造出的 PowerShell 命令字符串。首先，StrCpyW 函数被用来将起始字符串复制到这块新分配的内存中。
   - 构建 PowerShell 命令的策略：接下来的重点是字符串的构建过程。可以理解为 install.exe 利用 [Reflection.Assembly]::Load 方法来构建一个 PowerShell 命令。这个命令的目的是以反射的方式将**.NET stager 加载到内存中**。此外，通过修改 AmsiScanBuffer API，可以规避 Microsoft 的反恶意软件扫描接口（AMSI），使其始终返回 AMSI_RESULT_CLEAN 响应，即扫描结果显示内容安全，未发现威胁。
   - 命令混淆技术：函数的最后部分，通过调用 sub_40198D 多次实现了对 PowerShell 命令的混淆。这主要是通过使用随机字符串替换变量名称来实现的，增加了命令的隐蔽性和不易被识别的特点。

2. **sub_401678:**

   - 双重系统兼容性调用：sub_401678 被设计为对两种不同的系统架构进行操作：首先针对 32 位系统进行调用，随后再对 64 位系统执行相同的操作。这种双重调用策略确保了函数在不同的系统环境中都能有效运行。
   - COM 对象操作的核心：函数的主体涉及对 COM（组件对象模型）对象的操作。除了最初的 COM 环境初始化等准备工作外，函数的核心部分集中在处理 ppv+n 的地方。在这一环节，函数执行了对特定 COM 对象的函数调用。
   - COM 函数调用的实现：在 ppv+n 的操作中，函数通过调用 COM 对象的特定方法来实现其核心功能。这一步骤是整个函数操作的关键，它直接关联到 COM 对象的交互和控制。

3. **sub_401133:**

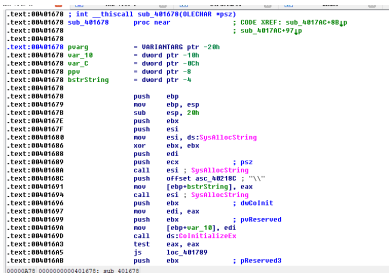
   - 进程句柄获取：在 sub_401133 的起始阶段，函数调用 GetCurrentProcess() 以获得当前进程的特殊句柄。这个句柄不是普通的句柄，而是一个代表执行该函数的进程的伪句柄。
   - 系统架构检测: 接下来，函数利用 IsWow64Process() 来判断当前进程是否在 WoW64（Windows 32 位在 Windows 64 位上的子系统）环境下运行。这个检测过程需要两个参数：一是进程的句柄，二是一个指向布尔类型变量的指针，用于存储检测结果。
   - 架构相关决策: 根据 IsWow64Process() 的返回结果，函数能够判断出操作系统是 32 位还是 64 位。这个判断对于后续操作至关重要，**因为它决定了下一个函数调用时传递的参数是"$77svc32" 还是"$77svc64"，这两个参数分别对应 32 位和 64 位服务**。

4. **sub_4011B1(v6, v7, v5):**

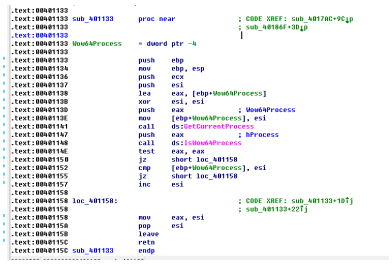
- 参数分析：在深入函数内部之前，首先关注其关键参数。其中，v6 与操作系统的位数（32 位或 64 位）直接相关，v7 是一个整型数值，而 v5 则是之前经过混淆处理的 PowerShell 命令字符串。
- COM 对象的应用: 函数核心部分涉及到 COM（组件对象模型）对象的使用。它主要通过调用 ppv + 40 等位置的函数来实现其功能。这些 COM 对象的调用是函数执行的关键部分。
- PowerShell 命令执行: 根据传入的参数和函数开头构造的”PowerShell\SYSTEM” 字符串，**可以推测该函数的主要作用是通过 COM 对象来执行 PowerShell 指令**。这种执行方式可能与系统的位数和特定的 PowerShell 命令有关。

5. **sub_40151E(v6)：**调用了 COM 对象的相关函数。

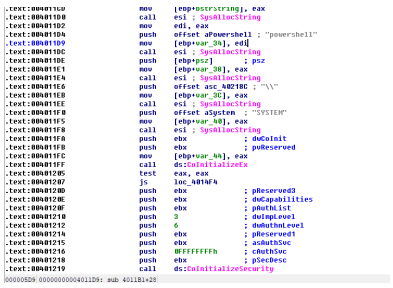
((a)) sub_40186F



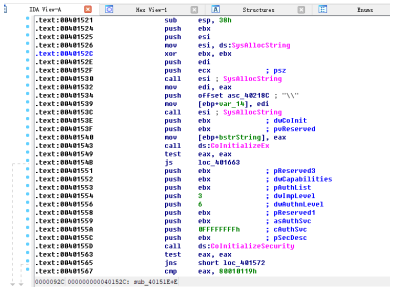
((b)) sub_401678



((c)) sub_401133



((d)) sub_4011B1



((e)) sub_40151E

图 3.27: 5 个调用函数

到此，我们可以给出结论。Install.exe 通过使用资源节的.Net stager 文件，加载后绕过系统监测。最终目的是让其注册服务并加载进内存。

3.13 资源节功能分析

接下来我们通过 ResoureceHacker 将其资源节提出后，对其进行了分析。不过由于过程过于复杂，这里只给出我的分析结论：

1. **DLL 挂钩移除:**Stager 的主要职能之一是移除 DLL 挂钩。这通常指的是撤销安全程序对关键系统 DLL（例如 NTDLL.dll 或 KERNEL32.dll）的修改，这些修改旨在监控或截获系统调用。此举旨在防止恶意行为被安全软件侦测。

2. **调整系统权限:**Stager 还会调整 SeDebugPrivilege 权限，这是 Windows 中的一个安全特权，它允许进程访问和操作其他进程的内存。这种权限通常用于执行进程注入等高级恶意操作。

3. **服务模块的解密与解压:**Stager 负责对其加载的服务模块进行解密和解压缩。这些服务模块可能携带用于实施恶意行为的代码。

4. **服务模块注入：**解密和解压缩完成后，Stager 会将这些服务模块注入到目标进程中，使得恶意代码能够在受害者的系统上隐秘运行。

5. **进程空洞化技术：**Stager 采用了被称为“进程空洞化”的技术。这涉及创建一个新进程（通常是合法的系统进程），然后清空其内存空间并填充恶意代码。这种方法通过利用合法进程作为掩护，隐蔽地执行恶意代码。

6. **父进程 ID 欺骗:** 为了进一步隐蔽其行为，Stager 可能采用父进程 ID（PPID）欺骗技术，伪装成另一个合法进程的子进程来创建新进程。这有助于规避基于行为的安全检测

综上所述，Stager 模块的核心职能包括撤销 DLL 挂钩、调整 SeDebugPrivilege 设置、解密及解压服务模块，并将这些服务模块注入到 32 位或 64 位的目标进程中。

3.14 服务模块分析

最后，我们简单看看其服务相关的部分，即 Helper32.dll 中做了什么吧。
首先加载其 PEiD 中，看到导出表中一些重要的函数：

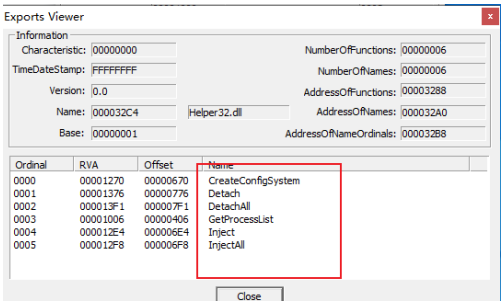


图 3.28: 导出表

图3.28看到了:

- CreateConffgSystem：配置必要环境；
- DetachAll 与 InjectAll: 卸载和注入恶意行为。

然后**我们将其载入到 IDA 中分析一些重点部分：**

3.14.1 配置系统的建立

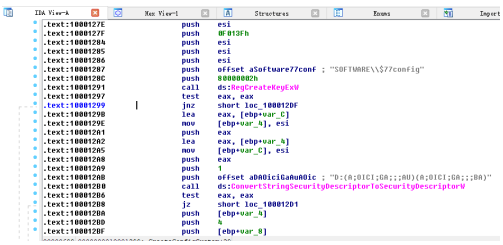


图 3.29: CreateConfigSystem

图3.29看到服务模块首先在注册表中建立配置系统。它在 HKEY_LOCAL_MACHINE\SOFTWARE\$77config 下创建一个键值，这个键值可以被计算机上的任何用户修改。接着，服务模块将当前运行的进程 ID 作

为值存储在 HKEY_LOCAL_MACHINE\SOFTWARE\$77config\pid 下的 svc32 或 svc64 注册表项中，这取决于系统的架构类型。

3.14.2 核心 Rootkit 的注入

服务模块的另一个主要任务是注入核心 Rootkit。这个核心是一个以资源形式存在于每个运行进程中的 DLL，除非在编译 Rootkit 之前已经在配置中被特定排除。服务模块创建了两个回调来实现这一目标。

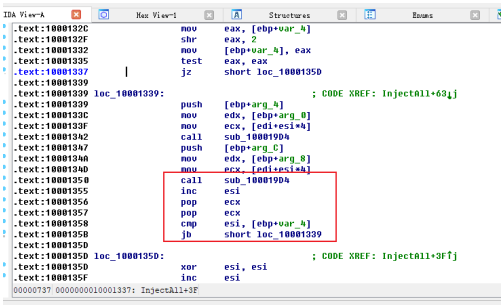


图 3.30: InjectAll

图3.30能清楚地看到这两个回调：

1. 第一个回调负责向所有运行中的进程注入 Rootkit 核心，通过每 100 毫秒枚举所有进程来实现。
2. 第二个回调负责向新创建的、已被感染父进程的子进程注入。服务模块利用进程间通信捕获子进程的创建，并在条件允许时进行注入。

3.14.3 Rootkit 核心的作用

这些省略了分析过程，给出我的结论：核心 Rootkit 的主要作用是在关键的 Windows API 上安装钩子，并根据 Rootkit 的配置过滤这些 API 的输出。

这些 API 包括 NtQuerySystemInformation、NtResumeThread、NtQueryDirectoryFile 等，它们通常用于获取系统信息。通过在这些 API 上设置挂钩，**核心模块能够有选择性地对系统用户和安全工具隐藏特定的文件、进程或注册表项。**

3.15 总结

现在我对我们看到的功能和分析的结果进行总结： R77 是一个开源的 Ring 3 rootkit，它可以隐藏系统中的各种元素，包括文件、目录、进程、CPU 使用率、注册表键值、服务、TCP 和 UDP 连接、联接、命名管道和计划任务。它的工作原理主要基于 Windows API 的钩子技术，通过修改 API 的行为来隐藏特定的系统元素。

R77 rootkit 主要由四个模块组成：安装器模块、阶段模块、服务模块和核心模块。

1. **安装器模块：**它的第一个任务是将阶段模块 PE 存储在注册表中，这是恶意软件作者常用的一种持久性存储恶意代码在系统上的技术。一旦阶段模块存储在注册表中，安装器模块就会构建一个 PowerShell 命令，该命令从注册表加载阶段模块并执行它，然后，安装器模块创建一个计划任务来运行 PowerShell 命令。
2. **阶段模块：**阶段模块的主要任务是将服务模块和核心模块注入到系统中。

3. **服务模块：**服务模块是一个 Windows 服务，它的主要任务是将核心模块注入到所有正在运行的进程中。

4. **核心模块：核心模块是实际的 rootkit，它通过钩子 Windows API 来隐藏系统元素。**

R77 rootkit 还包含一个动态配置系统，该系统允许通过 PID 和名称隐藏进程，通过完整路径隐藏文件系统项，隐藏特定端口的 TCP 和 UDP 连接等。配置位于 HKEY_LOCAL_MACHINE\\SOFTWARE\\$77config，并且可以由任何进程无需提升权限即可写入。

此外 R77 rootkit 还使用了一些 AV 和 EDR 规避技术，包括 AMSI 绕过和 DLL 反钩子。AMSI 绕过是通过修改 amsi.dll!AmsiScanBuffer 来实现的，使其总是返回 AMSI_RESULT_CLEAN。DLL 反钩子是通过从磁盘加载 ntdll.dll 的新副本并恢复原始部分来实现的。

总的来说，R77 rootkit 是一个高度复杂和灵活的工具，它利用了 Windows API 的钩子技术和其他一些高级技术来隐藏系统元素，从而使恶意软件能够在系统中持久存在并避免被检测。

4 实验结论及心得体会

4.1 实验结论

本次实验，我对除了恶意代码书中的以外的 Rootkit 的功能进行了验证，**全部实现了隐藏进程、文件、注册表和网络连接**的功能，这让我**对它的功能之强大感到十分惊讶**。

除此之外，我首先**通过查阅资料，了解到了 R77 的四个行为技术实现方式，然后通过 IDA 等技术手段进行了静态和动态综合分析，更全面地了解了它的实现细节。 总的来说，实验非常成功**。

4.2 心得体会

本次实验，我收获颇丰，这不仅是课堂中的知识，更是我解决许多问题的能力，具体来说：

1. 首先我进一步熟练使用 Prcomon 还有 ProcessExplorer 等进行病毒分析；

2. 在这个过程中我通过自己探索，还发现了一些不能 uninstall 的奇怪地方 hhh，发挥了我的创造能力；

3. 为了本次实验，**额外下了一个 Win10 虚拟机，提升了我配置虚拟几环境进行病毒分析的能力**。

4. **最后，如此强大的病毒 Rootkit 让我不寒而栗，要好好学习恶意代码，抵制他们!**

总的来说，本次通过亲自实验让我感受到了一个强大的病毒的能力和它复杂流程。也让我养成了充分的安全意识。培养了我对病毒分析安全领域的兴趣。我会努力学习更多的知识，辅助我进行更好的病毒分析。

感谢助教学姐审阅:)