南開大學

恶意代码分析与防治技术课程实验报告

R77 rootkit的工作原理

学院：网络空间安全学院
专业：信息安全
学号：2113997
姓名：齐明杰
班级：信安2班

1 实验目的

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。

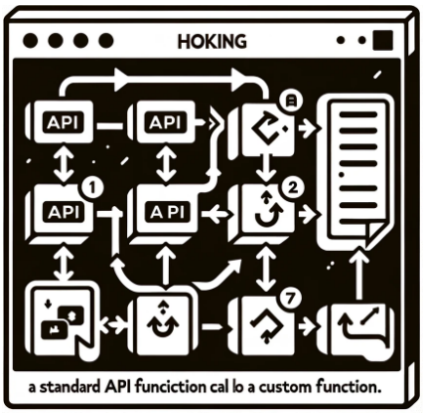
2 实验原理

2.1 Windows的Detours机制

Windows的Detours机制是一种由Microsoft开发的软件库，专用于拦截和重定向Windows应用程序中的函数调用。它通过动态修改目标函数的机器代码实现这一目的，通常是替换函数入口的前几个字节以跳转到一个自定义的代理函数。这个代理函数可以执行各种任务，包括调用原函数、修改函数的行为或完全替代原函数。Detours机制在系统监控、程序调试、性能分析以及安全工具的开发中发挥着关键作用。

2.2 API Hooking

API Hooking是一种允许开发者监控和修改应用程序调用API的方式的技术。它通常通过两种方式实现：一是直接修改API函数的机器代码（Inline Hooking），二是修改程序的导入地址表（IAT Hooking），使得API调用被重定向到一个预设的代理函数。这种技术被广泛用于软件调试、系统监控、安全防护以及增强现有系统功能。



2.3 DLL注入

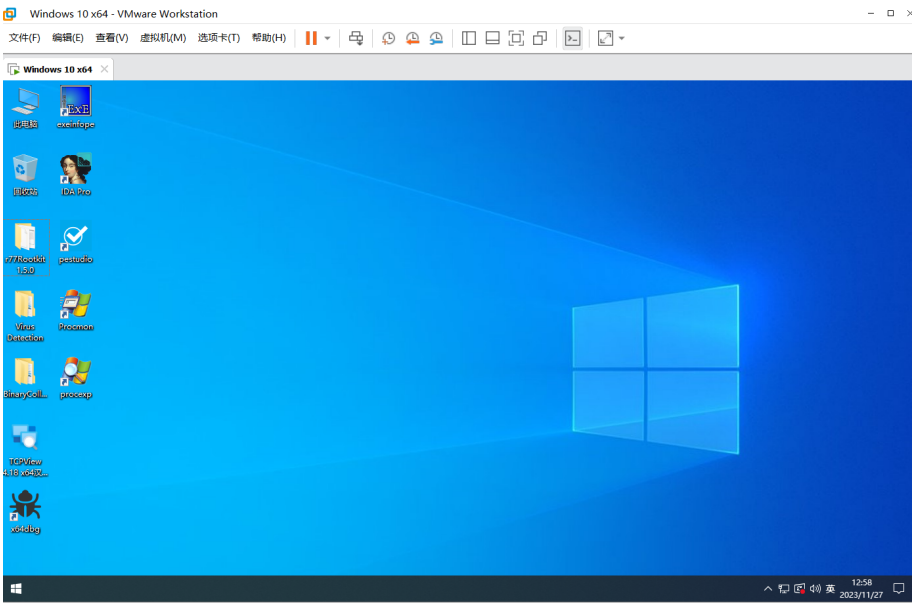
DLL注入是指将一个动态链接库（DLL）注入到另一个进程的地址空间中的过程。这可以通过多种方法实现，例如通过`CreateRemoteThread`函数创建一个新线程在目标进程中运行并加载DLL，或者使用Windows钩子功能`SetWindowsHookEx`来实现。DLL注入使得开发者能够影响或改变目标进程的行为，广泛应用于应用程序监控、调试、以及安全研究领域。

2.4 Windows的Ring0和Ring3

Windows操作系统中的Ring0和Ring3指的是处理器的不同保护级别。Ring0，也称为内核模式，是最高权限级别，允许直接访问硬件和所有系统资源。操作系统核心组件和驱动程序通常在这一级别运行。而Ring3，或用户模式，是普通应用程序运行的环境，具有限制的访问权限，不能直接操作硬件或关键系统资源。理解这两种模式的区别对于操作系统的安全性和功能实现是非常重要的。

3 实验环境

本次实验我依然在windows 10 x64的虚拟机上运行和分析R77 Rootkit。

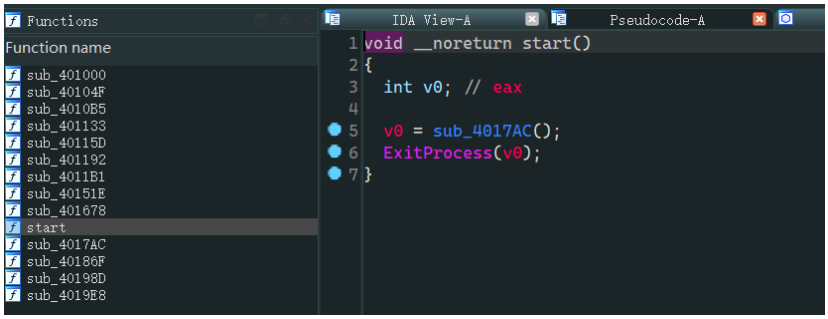


4 实验过程

4.1 IDA分析

由于R77是一个工程量巨大而逻辑十分复杂的rootkit，我们仅使用IDA进行粗略地分析。

载入Install.exe，进入start函数：



发现其核心在于sub_4017AC，进入查看代码：

```
int sub_4017AC()
{
  HRSRC ResourceA; // eax
  HRSRC v1; // esi
  DWORD v2; // edi
  HGLOBAL Resource; // eax
  const BYTE *v4; // esi
  OLECHAR *v5; // esi
  OLECHAR *v6; // edi
  int v7; // ecx
  HKEY phkResult; // [esp+8h] [ebp-4h] BYREF

  ResourceA = FindResourceA(0, (LPCSTR)0x65, "EXE");
  v1 = ResourceA;
  if ( ResourceA )
  {
    v2 = SizeofResource(0, ResourceA);
    if ( v2 )
    {
      Resource = LoadResource(0, v1);
      if ( Resource )
      {
        v4 = (const BYTE *)LockResource(Resource);
        if ( !RegOpenKeyExW(HKEY_LOCAL_MACHINE, L"SOFTWARE", 0, 0xF013Fu,
&phkResult)
          && !RegSetValueExW(phkResult, L"$77stager", 0, 3u, v4, v2) )
        {
          v5 = (OLECHAR *)sub_40186F();
```

```
            sub_401678((OLECHAR *)L"$77svc32");
            v6 = (OLECHAR *)L"$77svc64";
            sub_401678((OLECHAR *)L"$77svc64");
            if ( !sub_401133() )
              v6 = (OLECHAR *)L"$77svc32";
            if ( sub_4011B1(v6, v7, v5) )
              sub_40151E(v6);
          }
        }
      }
    }
    return 0;
}
```

`install.exe`是R77 rootkit的一个关键组成部分，其功能和实现机制相当复杂。以下是对其功能的**综合概述**：

`install.exe`的主要目的是将R77 rootkit植入目标系统，确保其能够在系统中持久化并执行其设计的功能，主要包括隐藏进程、文件、注册表项和网络连接等。在启动时，`install.exe`首先从其自身的资源部分提取嵌入的文件，这些文件可能包含了R77 rootkit的主要执行代码或其他必要的组件。

该程序使用Windows API函数`FindResourceA`和`SizeofResource`来定位和获取这些嵌入资源。接着，通过`LoadResource`和`LockResource`加载并锁定这些资源，确保它们能够被正确处理。此后，`install.exe`使用`RegOpenKeyExW`和`RegSetValueExW`函数在注册表中创建或修改值，将提取的资源以二进制形式存储，为后续操作做准备。

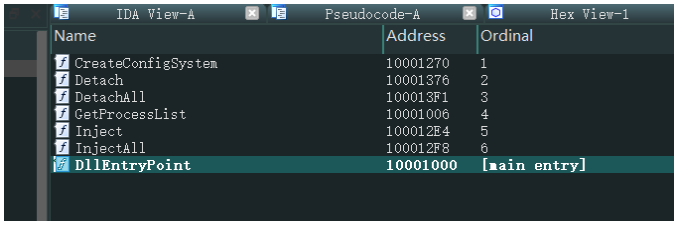
接下来的一大步是构建并执行PowerShell命令。`install.exe`动态生成PowerShell脚本，其中可能包含了使用`[Reflection.Assembly]::Load`方法以反射方式加载.NET stager到内存中的命令。这种方法的一个关键优势是可以绕过Microsoft的反恶意软件扫描接口（AMSI），使恶意活动不被检测到。此外，通过修改`AmsiScanBuffer` API，`install.exe`进一步确保恶意操作不会被现有的安全机制所识别。

`install.exe`还包含了针对32位和64位系统的特定处理逻辑。它通过检测当前系统的架构（使用`GetCurrentProcess`和`IsWow64Process`函数）来决定执行哪个版本的R77服务（如`$77svc32`或`$77svc64`）。根据系统架构的不同，`install.exe`将加载适当的R77组件。

此外，`install.exe`还涉及到COM对象的使用，通过这些对象执行一系列操作，包括执行混淆后的PowerShell命令，注册服务等。这一部分的实现细节可能相当复杂，涉及多个COM接口的调用和操作。

将Helper32.dll载入IDA进行分析：

可以发现它具有几个导出函数：



我们挑选一两个重要的进行分析。

- **CreateConfigSystem**

```
1  int CreateConfigSystem()
2  {
3    ULONG SecurityDescriptorSize; // [esp+4h] [ebp-Ch] BYREF
4    HKEY phkResult; // [esp+8h] [ebp-8h] BYREF
5    PSECURITY_DESCRIPTOR SecurityDescriptor; // [esp+Ch] [ebp-4h] BYREF
6
7    if ( RegCreateKeyExW(HKEY_LOCAL_MACHINE, L"SOFTWARE\\$77config", 0, 0, 0, 0xF013Fu, 0, &phkResult, 0) )
8      return 0;
9    SecurityDescriptor = 0;
10   SecurityDescriptorSize = 0;
11   if ( ConvertStringSecurityDescriptorToSecurityDescriptorW(
12          L"D:(A;OICI;GA;;;AU)(A;OICI;GA;;;BA)",
13          1u,
14          &SecurityDescriptor,
15          &SecurityDescriptorSize) )
16   {
17     RegSetKeySecurity(phkResult, 4u, SecurityDescriptor);
18     LocalFree(SecurityDescriptor);
19   }
20   RegCloseKey(phkResult);
21   return 1;
22 }
```

这段代码的主要功能是在Windows注册表中创建一个新的键，并设置其安全性，使其可以被任何用户修改。具体来说，代码执行以下操作：

1. **创建注册表键**：在`HKEY_LOCAL_MACHINE`下的`SOFTWARE\\$77config`路径创建一个新的注册表键。这个键是R77配置系统的一部分。
2. **设置安全描述符**：使用`ConvertStringSecurityDescriptorToSecurityDescriptorW`函数创建一个安全描述符，该描述符允许所有用户（"AU"）和管理员（"BA"）完全访问该键。
3. **应用安全设置**：通过`RegSetKeySecurity`函数将安全描述符应用到新创建的注册表键上。
4. **关闭注册表键**：最后，通过`RegCloseKey`函数关闭刚创建的注册表键句柄。

这段代码的目的是为R77 rootkit创建一个可由任何用户访问和修改的配置存储区域。这样，R77的服务模块就可以将其当前进程ID存储在这个注册表键下，以便于后续操作和配置的管理。根据系统的架构不同，这个进程ID被存储在名为“svc32”或“svc64”的注册表项中。通过这种方式，R77确保了其配置的可访问性和灵活性，同时也为其后续的操作提供了必要的支持。

- **InjectAll**

```
DWORD __cdecl InjectAll(int a1, SIZE_T dwSize, int a3, SIZE_T a4)
{
  DWORD v4; // esi
  HANDLE ProcessHeap; // eax
  DWORD *v6; // edi
  HANDLE v7; // eax
  DWORD cbNeeded; // [esp+8h] [ebp-4h] BYREF

  v4 = 0;
  ProcessHeap = GetProcessHeap();
  cbNeeded = 0;
  v6 = (DWORD *)HeapAlloc(ProcessHeap, 0, 0x9C40u);
  if ( K32EnumProcesses(v6, 0x9C40u, &cbNeeded) )
  {
    cbNeeded >>= 2;
    if ( cbNeeded )
    {
      do
      {
        sub_100019D4(v6[v4], dwSize);
        sub_100019D4(v6[v4++], a4);
      }
      while ( v4 < cbNeeded );
    }
    v4 = 1;
  }
  v7 = GetProcessHeap();
  HeapFree(v7, 0, v6);
  return v4;
```

这段代码的主要功能是实现R77 rootkit的核心注入机制，通过两个关键步骤确保rootkit能够在系统中广泛分布并执行其隐蔽操作：

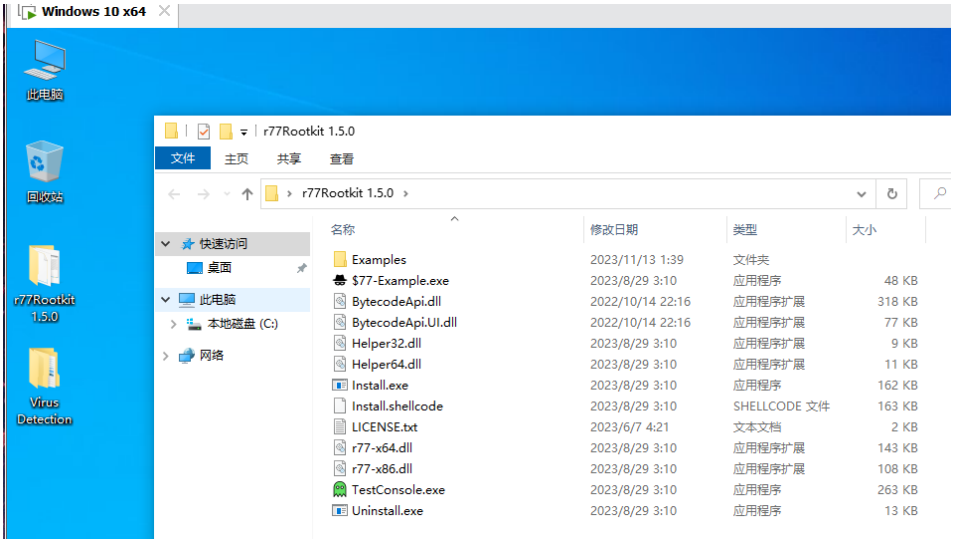
1. **对所有进程的注入**：代码首先通过`K32EnumProcesses`函数枚举当前系统中运行的所有进程。然后，它利用`sub_100019D4`函数对这些进程逐一进行注入操作。这一步骤确保了R77的核心组件（一个DLL文件）被加载到除了特定排除进程外的所有运行中进程中。
2. **新创建子进程的注入**：除了对现有进程的注入外，代码还负责监控新创建的子进程，并在适当的时候将rootkit注入这些子进程中。这是通过服务和已感染进程之间的进程间通信实现的，捕获子进程的创建并执行注入。

R77 rootkit的核心功能包括在多个重要的Windows API上安装钩子。这些API被用于获取系统信息，例如`NtResumeThread`，它在父进程创建子进程后被调用以启动线程执行。通过安装的钩子，rootkit能够将子进程的PID发送到服务模块，并在条件允许时注入其中。

这个rootkit的核心组件还负责在关键的Windows API上设置挂钩并过滤这些API的输出。例如，它可以在`NtEnumerateKey`和`NtEnumerateValueKey`上设置挂钩，这些API被用于枚举注册表项，然后过滤掉配置中指定的特定项。

4.2 Install.exe & Uninstall.exe 效果及原理

下载R77文件，并解压到桌面：



4.2.1 运行Install.exe

当运行`Install.exe`时，它执行以下操作：

1. **注入r77到每个正在运行的进程**：`Install.exe`将r77 rootkit注入到系统中的所有活动进程中。这意味着r77的代码和功能将被加载到这些进程的内存空间中。
2. **进程创建拦截**：`Install.exe`通过拦截进程创建过程来确保所有新启动的进程在执行任何自己的指令之前被注入r77。这是通过修改系统级的API或使用更低级别的技术来实现的，如修改进程创建相关的系统调用。
3. **持久化**：安装后，r77设置为在系统重启后自动启动，并在第一个用户登录前注入所有进程，从而实现持久化。
4. **单文件部署**：`Install.exe`包含了所有必需的文件，因此不需要额外部署DLL文件。
5. **升级机制**：如果`Install.exe`在r77已经安装的情况下执行，它会终止并重新创建r77服务进程，以升级到当前版本。已注入的进程将不会被卸载并重新注入新版本的r77 DLL。如果需要这样做，必须先运行`Uninstall.exe`。

4.2.2 运行Uninstall.exe

当运行`Uninstall.exe`时，它执行以下操作：

1. **从系统中完全移除r77**：`Uninstall.exe`将从系统中完全卸载r77 rootkit。
2. **从所有进程中分离rootkit**：它将从所有正在运行的进程中分离r77，这意味着它会移除之前注入的r77代码和功能。
3. **删除r77配置**：它还会从注册表中删除与r77相关的所有配置信息。
4. **不需要重启**：完成这些操作后，不需要重启计算机。

4.2.3 技术实现原理

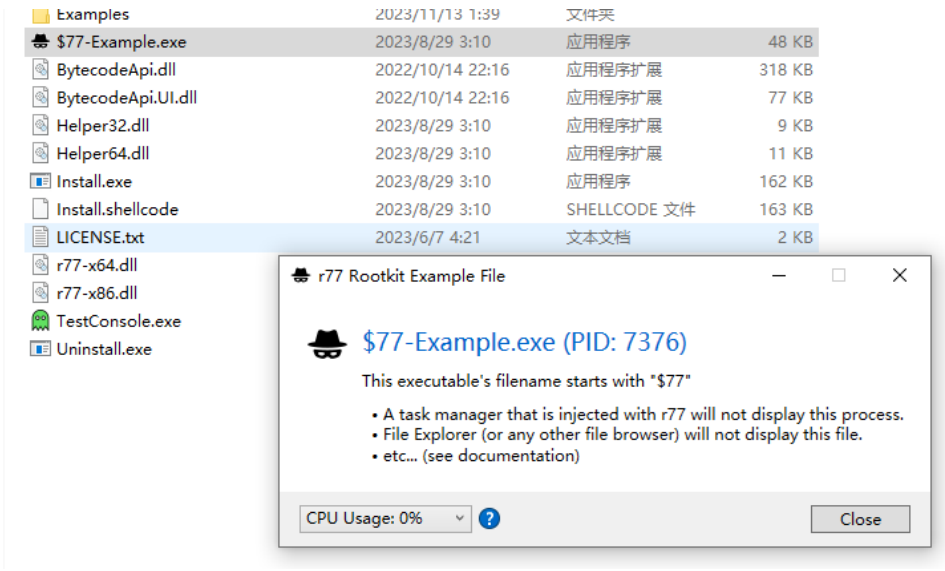
1. **DLL注入与进程拦截**：`Install.exe`通过使用DLL注入技术将r77代码加载到运行中的进程。这可能涉及创建远程线程或使用其他注入技术。进程创建拦截可能是通过修改系统级API（如`CreateProcess`）来实现，或者通过拦截较低级别的系统调用（如与进程创建相关的NT级别函数）。
2. **持久化机制**：r77的持久化是通过修改系统配置（如注册表项）或创建特殊的启动任务来实现的，确保在系统重启后自动加载。

3. **卸载与清理**：`Uninstall.exe`负责从每个已注入的进程中移除r77，并清理所有相关配置，确保rootkit不留下任何痕迹。这可能涉及逆转注入过程和删除或修改注册表项和其他系统配置。

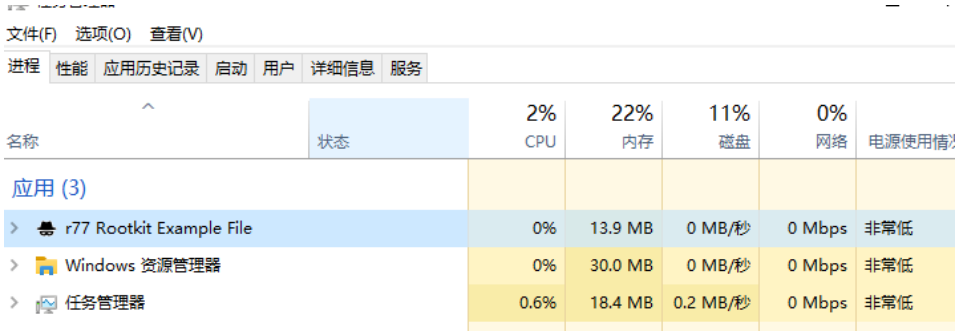
4.3 $77-Example.exe

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。

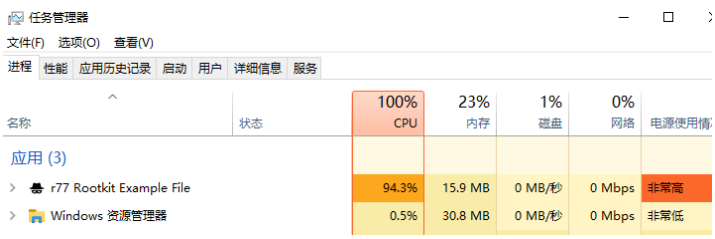
然后我们运行它：



再查看任务管理器：



这个示例文件可以**调节CPU占有率**，如果我将其调成100%，**CPU占用率将明显上升**：



CPU使用率模拟：该程序包含一个“CPU使用率”组合框（ComboBox），允许用户改变其模拟的CPU使用率。当这个进程在任务管理器中被隐藏时，其CPU使用率将被添加到“系统空闲进程”中。这意味着即使进程被隐藏，CPU使用率图也会相应地显示负载变化。

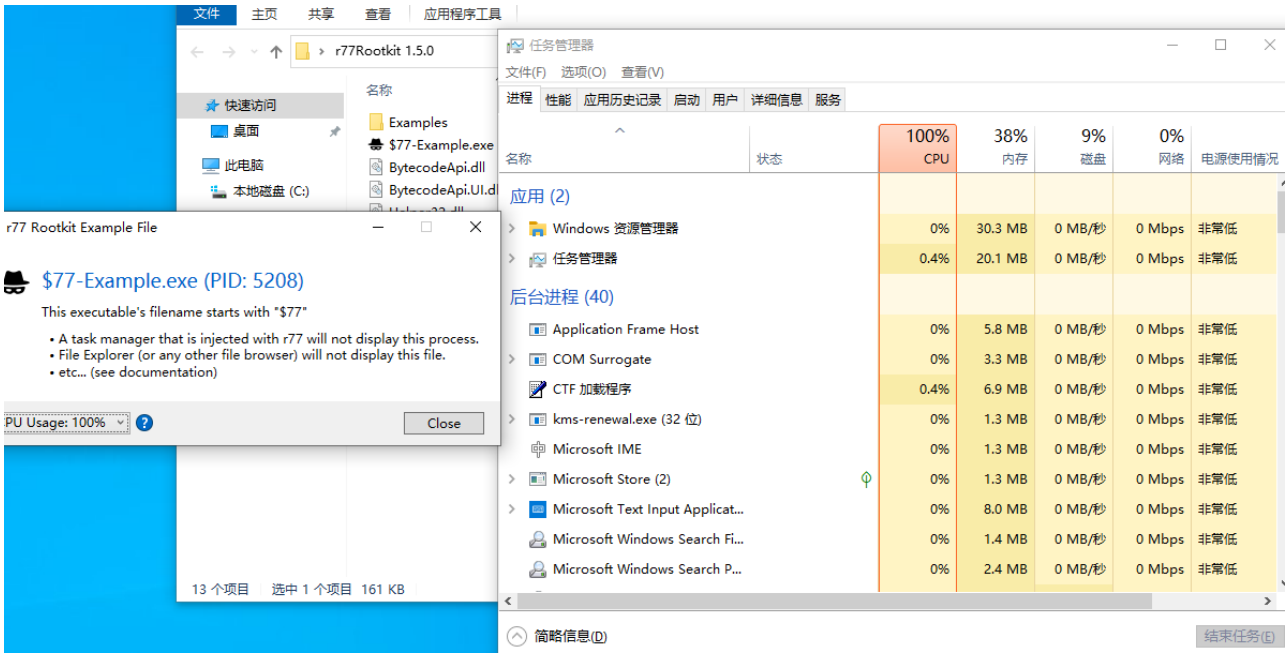
`$77-Example.exe` 是R77 rootkit提供的一个示例程序，它的主要目的是测试任务管理器和文件查看器的隐藏功能。

4.4 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。

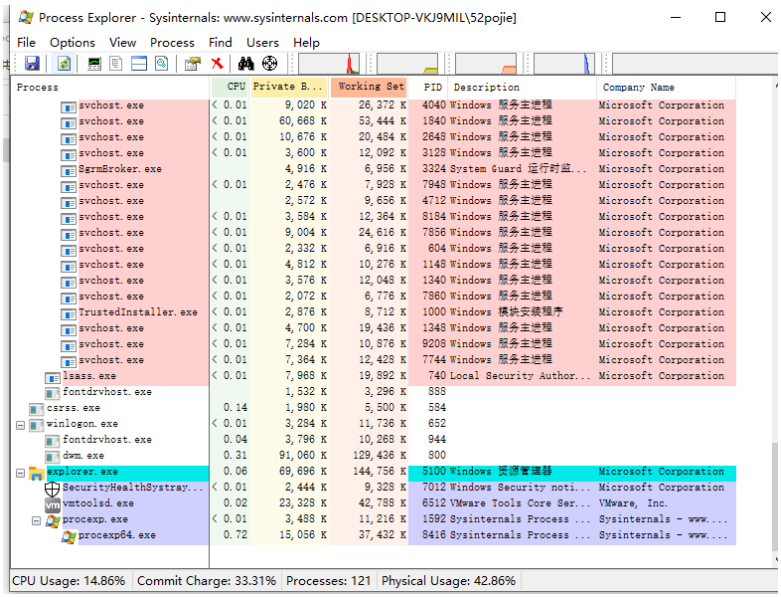
4.4.1 $77-进程隐藏

先运行**$77-Example.exe**，调节CPU占用率至100%，然后运行`Install.exe`，打开任务管理器，发现**找不到这个进程**了：



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：

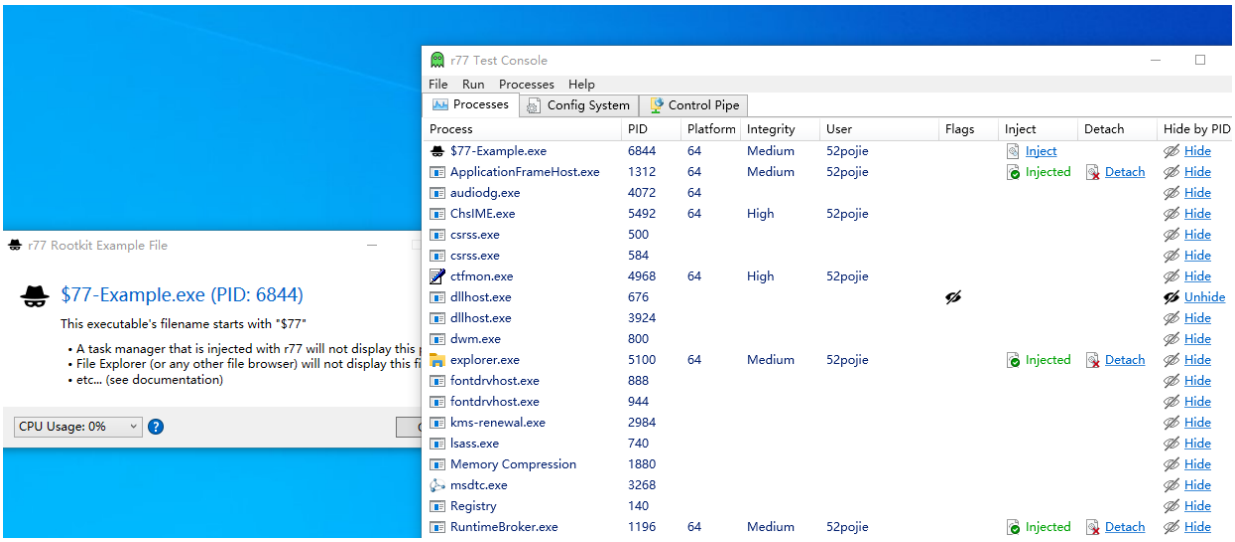


仍然没有找到这个进程，证明其确实被隐藏了。

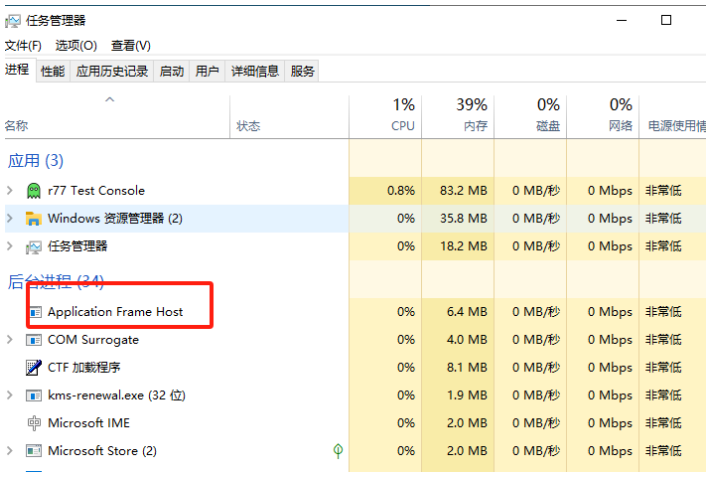
4.4.2 任意进程隐藏

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。

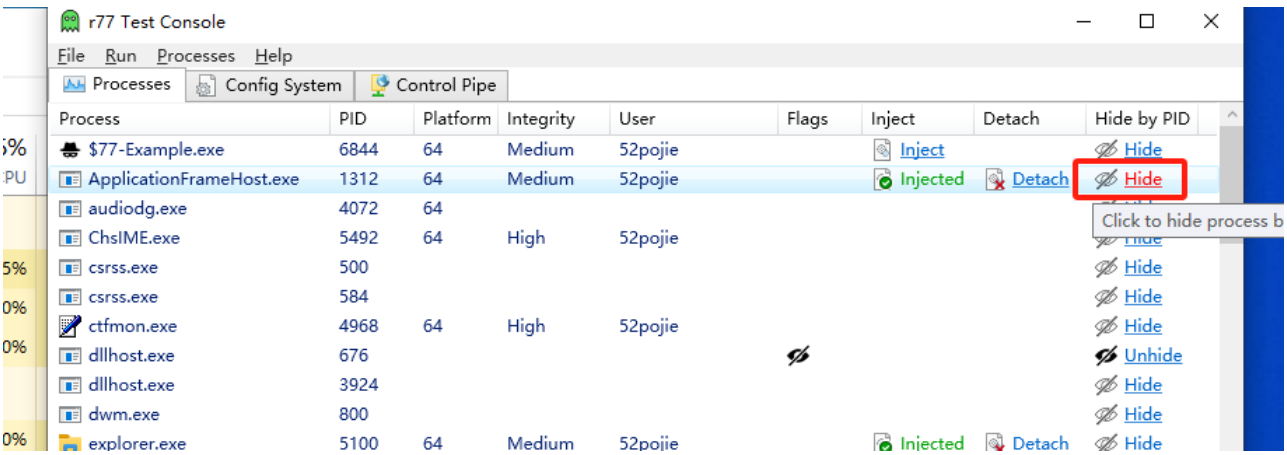
我们运行`TestConsole.exe`，可以看到进程列表，包括被隐藏的进程：



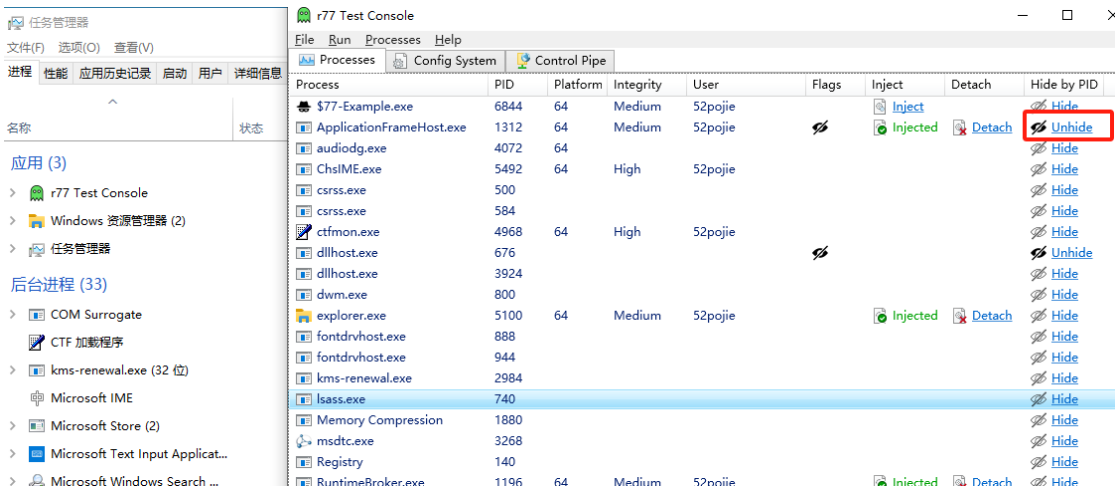
随便再找一个**任意进程名**的进程，选取下面的进程：



点击r77 Test Console右边对应的Hide操作：



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**:



这就实现了任意指定进程的隐藏。

4.4.3 进程隐藏技术原理

- **R77 rootkit的进程隐藏技术包括以下几个关键方面**:

1. **基于名称的隐藏**:
   - R77通过检查进程的文件名来决定是否隐藏该进程。具体来说，如果可执行文件的文件名以特定的前缀（例如`$77`）开始，R77会自动隐藏这些进程。
   - 这种隐藏是通过拦截和修改系统中用于列出进程的API调用实现的。当这些API被调用时，R77的代码介入并修改API的返回结果，从而在任务管理器等工具中不显示以`$77`为前缀的进程。

2. **基于进程ID的隐藏**:
   - R77还提供了一种机制，允许用户通过指定进程ID来隐藏特定的进程。这是通过R77的配置系统来实现的，用户可以直接将进程ID写入配置系统来指定需要隐藏的进程。
   - 这种方法适用于那些无法更改文件名的在内存中创建的进程。

3. **基于名称的隐藏（替代方法）**:
   - 除了使用前缀，R77还允许用户通过指定特定的进程名称来隐藏进程。这也是通过配置系统实现的，用户可以指定要隐藏的进程名称。

4. **枚举与直接访问**:
   - 在R77中，隐藏意味着从系统枚举中移除隐藏实体。这意味着虽然隐藏的进程不会出现在像任务管理器这样的工具中，但如果知道进程的名称或ID，用户仍然可以直接访问这些进程。
   - 需要注意的是，用于打开文件或进程的函数并没有被R77拦截，因此即使是隐藏的实体，如果直接通过名称或ID访问，也不会返回“未找到”错误。

- **R77 rootkit的进程隐藏技术具体实现涉及以下几个关键环节**:

1. **Reflective DLL注入**:

   - R77以两种形式存在：`r77-x86.dll` 和 `r77-x64.dll`，分别用于32位和64位进程。
   - R77实现了所谓的**反射式DLL注入**（Reflective DLL Injection）。在这种技术中，DLL文件直接写入远程进程的内存，而不需要写入磁盘。
   - 这种注入方式通过调用`ReflectiveDllMain`导出来加载DLL并调用`DllMain`。因此，注入的DLL不会在进程的PEB（进程环境块）中列出。

   反射式DLL注入是一种高级方法，用于将DLL加载到进程的内存空间中，而不使用标准的Windows API函数。其工作原理如下：

   (a) **绕过标准加载**：通常，DLL通过如`LoadLibrary`这样的Windows函数加载到进程的内存空间中。反射式DLL注入则绕过了这些标准方法。
   (b) **内存加载**：在反射式DLL注入中，DLL文件直接写入目标进程的内存中。这通常通过创建远程线程或其他方法来实现。
   (c) **自引导**：注入的DLL包含一个特殊的引导代码（通常称为`ReflectiveLoader`），该代码负责初始化DLL，包括处理重定位、解析导入等。
   (d) **隐蔽性**：由于不使用标准的加载方法，反射式注入的DLL不会出现在进程的模块列表中，从而提高了隐蔽性。

2. **进程创建拦截**:

   - 当R77服务启动时，它将注入所有当前运行的进程。
   - R77通过拦截`NtResumeThread`函数来实现对新进程创建的钩取。当一个进程（例如进程A）创建另一个进程（例如进程B）时，`NtResumeThread`被调用，此时进程B已经完全初始化但仍处于暂停状态。在实际调用此函数完成进程B的创建之前，R77会被注入到新进程中。
   - 为了处理32位进程创建64位子进程的情况，R77服务提供了一个命名管道来处理注入请求。当新的进程ID发送到R77服务时，服务执行注入并返回确认。收到服务的确认后，注入完成，`NtResumeThread`随后被调用。

3. **注入时机**:

   - 这种方法确保了在新进程执行其任何指令之前，R77就已经被注入。这一点非常重要，因为某些程序（例如RegEdit）启动后会迅速初始化，然后执行枚举操作并快速显示结果。因此，rootkit必须在进程启动的最初阶段就完成注入。

通过这种方式，R77能够有效地隐藏特定进程，使其对操作系统的常规监视工具不可见，从而实现了高度的隐蔽性。这种隐藏技术的应用不仅展示了在操作系统层面上进行监控和修改的强大能力，而且也揭示了操作系统中可能被恶意软件利用的漏洞。

4.4.4 反射式DLL注入代码示例

```
// 反射式DLL注入的基本框架
BOOL ReflectiveDLLInjection(HANDLE hProcess, LPVOID dllBuffer) {
    LPVOID remoteMemory = VirtualAllocEx(hProcess, NULL, dllSize,
MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    if (remoteMemory == NULL) return FALSE;
```

```
5
6       // 写入远程进程内存
7       if (!WriteProcessMemory(hProcess, remoteMemory, dllBuffer, dllSize,
    NULL)) {
8           VirtualFreeEx(hProcess, remoteMemory, dllSize, MEM_RELEASE);
9           return FALSE;
10      }
11
12      // 获取ReflectiveLoader的地址并在远程进程中创建线程
13      LPVOID loaderAddr = GetReflectiveLoaderOffset(dllBuffer);
14      HANDLE remoteThread = CreateRemoteThread(hProcess, NULL, 0,
    (LPTHREAD_START_ROUTINE)((LPBYTE)remoteMemory + loaderAddr), remoteMemory,
    0, NULL);
15      if (remoteThread == NULL) {
16          VirtualFreeEx(hProcess, remoteMemory, dllSize, MEM_RELEASE);
17          return FALSE;
18      }
19
20      WaitForSingleObject(remoteThread, INFINITE);
21      CloseHandle(remoteThread);
22
23      return TRUE;
24  }
25
```

此代码展示了如何将DLL注入到另一个进程的内存空间中，而不依赖于标准的Windows加载机制。主要步骤包括：

1. **内存分配**：使用`VirtualAllocEx`在目标进程中分配内存，为DLL文件提供空间。
2. **写入DLL**：通过`WriteProcessMemory`将DLL的内容写入之前分配的远程进程内存。
3. **获取入口点**：`GetReflectiveLoaderOffset`用于获取DLL中`ReflectiveLoader`函数的偏移地址。
4. **创建远程线程**：在目标进程中创建一个新线程，执行`ReflectiveLoader`，从而完成DLL的加载。
5. **等待执行**：使用`WaitForSingleObject`等待新创建的线程执行完毕。

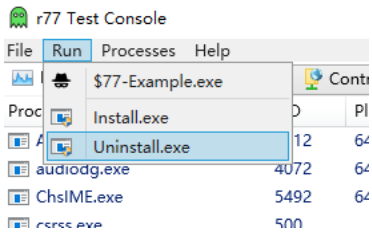
整个过程的关键是在目标进程中创建一个线程来执行DLL内部的自引导代码，而不是依赖于外部的加载机制，这增加了注入的隐蔽性。

4.5 文件隐藏

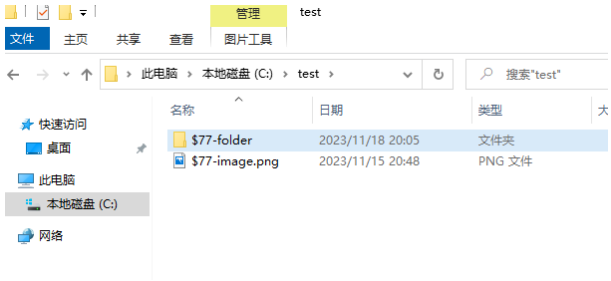
R77的文件隐藏功能是一种技术，用于使特定文件或目录在操作系统的标准文件浏览工具（例如Windows资源管理器）中不可见。这通常是通过**拦截和修改系统级别的文件系统调用**来实现的。

4.5.1 文件隐藏效果

先执行uninstall:

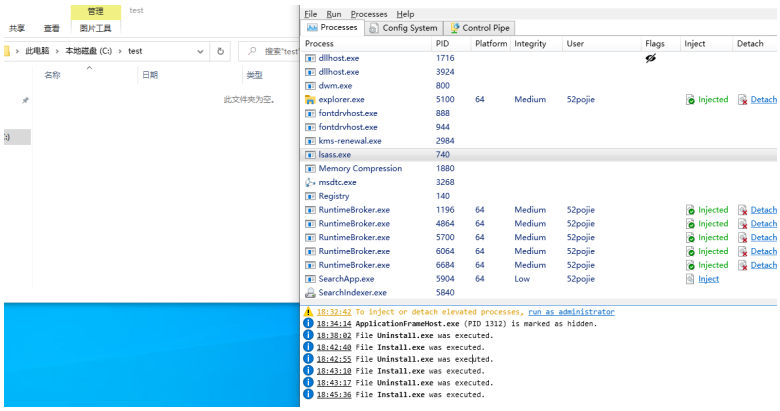


在一个文件夹里面放入我想**要隐藏的文件和文件夹**:



注意，文件和文件夹必须以$77为文件名开头才能隐藏。

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。

4.5.2 文件隐藏技术原理

R77的文件隐藏功能的实现细节如下:

1. **文件系统函数拦截**:

   - R77主要通过拦截Windows的核心文件系统函数，如`NtQueryDirectoryFile`和`NtQueryDirectoryFileEx`来实现文件和目录的隐藏。(`NtQueryDirectoryFile`是Windows操作系统中的一个核心文件系统函数，用于枚举（列出）目录中的文件和子目录。)
   - 这些函数在文件系统中执行关键角色，用于枚举目录内容，包括文件、目录、联接和命名管道。

- 拦截这些函数是R77实现文件隐藏的核心机制。当这些函数被操作系统或应用程序调用时，R77会介入并修改它们的行为。

2. **枚举结果的修改**：

- 在拦截了这些函数之后，R77会检查每个枚举的项目（文件或目录）的名称。
- 对于那些名称以**特定前缀**（如`$77`）开头的文件或目录，R77会从API调用的返回结果中排除这些项目。
- 这种修改意味着即使文件或目录实际存在于文件系统中，它们也不会出现在通过这些API函数枚举的结果中，例如在Windows资源管理器或命令行界面中。

3. **对特定路径的隐藏支持**：

- 除了自动隐藏以特定前缀命名的文件和目录外，R77还允许通过其配置系统进行更精确的控制。
- 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，以指示R77隐藏这些特定路径的文件或目录。

4. **隐藏的效果**：

- 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。
- 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. **安全和隐蔽性考量**：

- R77的这种隐藏方法提供了**极高的隐蔽性**，因为它直接在操作系统的核心层面上修改文件系统的行为。
- 同时，这种能力也表明了潜在的安全风险，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的文件隐藏技术是通过精巧地拦截和修改关键文件系统函数的行为来实现的，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。

4.5.3 拦截NtQueryDirectoryFile的代码示例

```
// NtQueryDirectoryFile拦截示例
NTSTATUS HookedNtQueryDirectoryFile(...) {
    NTSTATUS status = OriginalNtQueryDirectoryFile(...);

    if (NT_SUCCESS(status)) {
        // 检查并过滤结果，隐藏符合条件的文件或目录
        FilterDirectoryContents(...);
    }

    return status;
}
```

此代码演示了如何拦截`NtQueryDirectoryFile`函数，该函数用于枚举目录内容：

1. **调用原始函数**：首先调用原始的`NtQueryDirectoryFile`函数。
2. **检查状态**：检查函数调用的返回状态。如果成功，即有文件或目录被枚举出来。
3. **过滤内容**：在`FilterDirectoryContents`函数中，根据特定条件（例如文件名前缀）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。

4.6 注册表隐藏

4.6.1 注册表隐藏效果

执行Uninstall后，打开注册表，新建一个以$77为开头的注册表键：

然后执行Install.exe，重新打开注册表，发现`$77mykey`**被隐藏了**：

4.6.2 注册表隐藏技术原理

R77实现注册表隐藏的具体步骤和机制如下：

1. **拦截关键注册表函数**：

   - R77拦截了两个重要的内核级注册表函数：`NtEnumerateKey`和`NtEnumerateValueKey`。这些函数在Windows操作系统中用于枚举（即列出）注册表键和值，**是许多系统和应用程序访问注册表的基础**。

2. **修改枚举行为**：

   - 当这些函数被调用时，R77首先执行正常的枚举操作。
   - 然后，R77会检查每个枚举的注册表项和值，识别那些符合隐藏条件的项（例如，名称以`$77`为前缀的注册表项或值）。

3. **调整枚举索引**：

   - 为了隐藏这些注册表项和值，R77需要调整其余项的枚举索引。它会重新计算索引，确保隐藏的项在返回的列表中不可见。
   - 这意味着即使注册表项或值实际存在，当通过标准API进行枚举时，它们不会被包含在结果中。

4.6.3 拦截NtEnumerateKey的代码示例

```
// NtEnumerateKey拦截示例
NTSTATUS HookedNtEnumerateKey(...) {
    NTSTATUS status = OriginalNtEnumerateKey(...);

    if (NT_SUCCESS(status)) {
        // 检查并过滤结果，隐藏符合条件的注册表项
        FilterRegistryKeys(...);
    }

    return status;
}
```

此代码展示了如何拦截用于枚举注册表键的`NtEnumerateKey`函数：

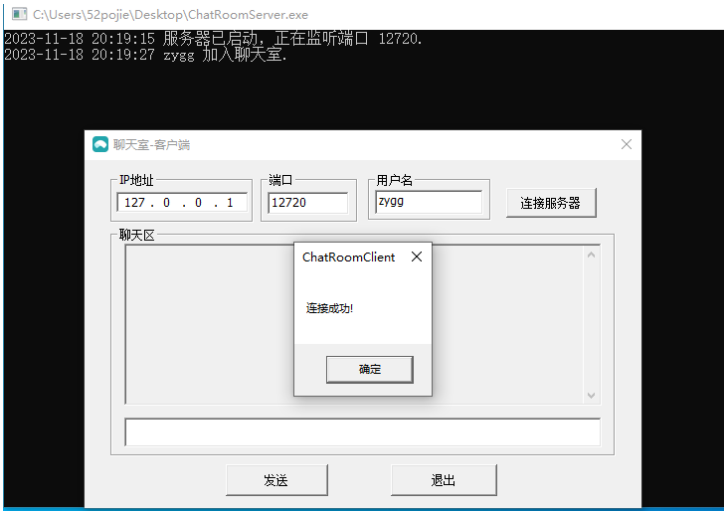
1. **调用原始函数**：执行原始的`NtEnumerateKey`函数。
2. **检查执行结果**：验证函数调用是否成功完成。
3. **过滤注册表键**：在`FilterRegistryKeys`函数中，根据给定的条件（如键名前缀）过滤掉不希望显示的注册表键。

该方法允许**在枚举注册表项时实现细粒度的控制，有效隐藏特定的注册表项。**

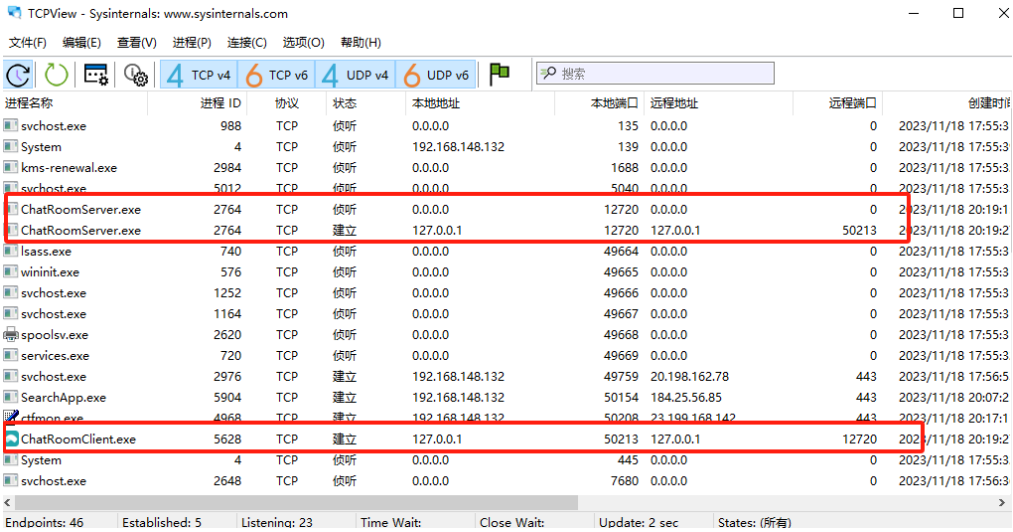
4.7 TCP&UDP隐藏

4.7.1 TCP通信隐藏效果

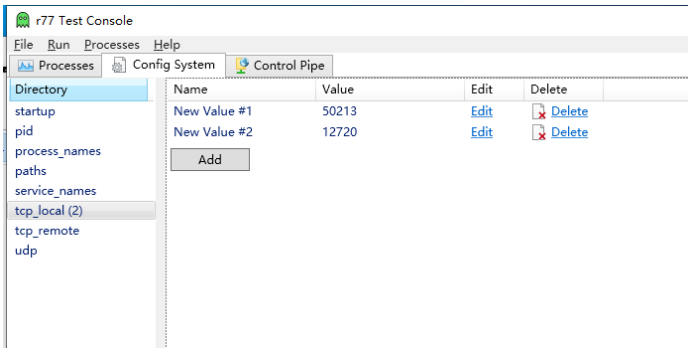
我准备了一个聊天室程序，采用TCP编程实现，分为服务端和客户端：



打开TCPView可以查看TCP通信状态：



那么我们打开R77面板，设置TCP的隐藏选项：

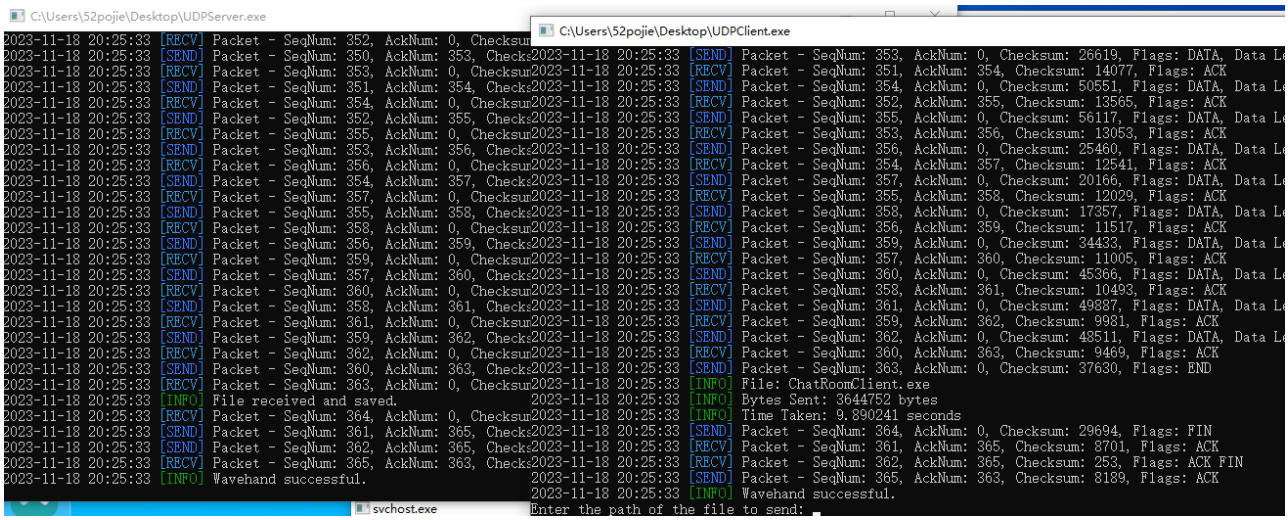


然后重新查看TCPView：

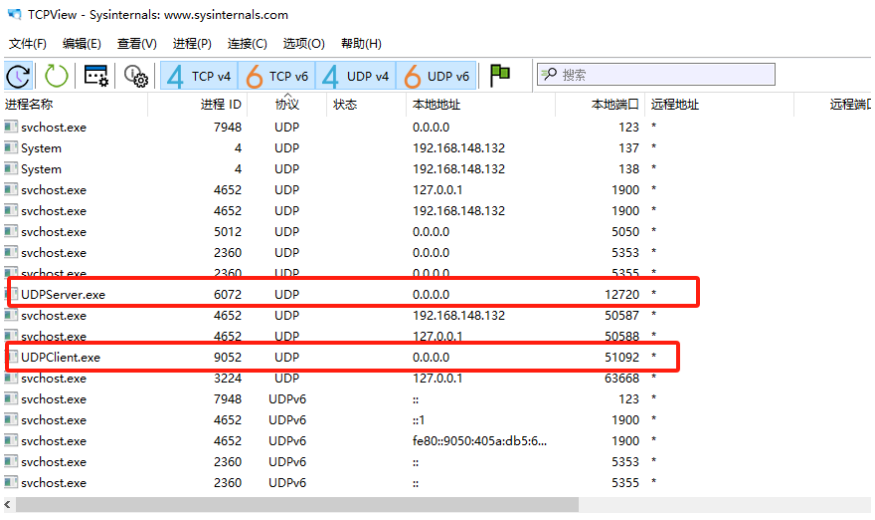
发现侦察不到相应端口的TCP通信了。

4.7.2 UDP通信隐藏效果

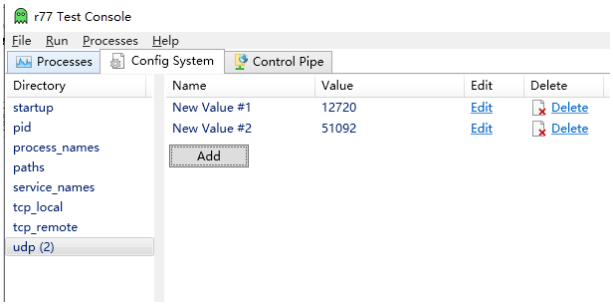
我编写了一个使用UDP协议的可靠文件传输，分为客户端(发送)和服务端(接收)：



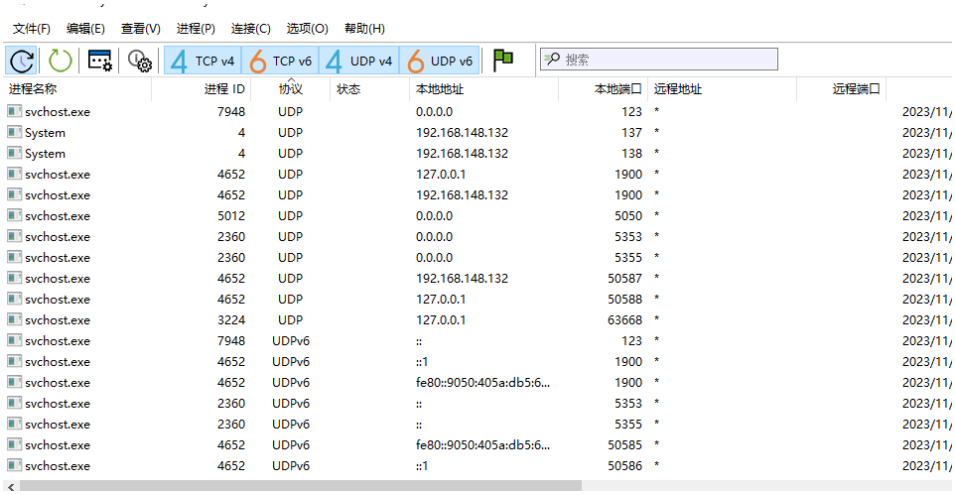
其在TCPView也可以查看端口：



修改隐藏的端口：



运行Install.exe，同样被隐藏了：



4.7.3 TCP&UDP通信隐藏技术原理

*R77隐藏TCP和UDP连接的机制涉及以下几个**关键步骤**：*

1. **拦截网络驱动I/O控制函数**：
   - R77拦截了用于执行网络I/O控制操作的`NtDeviceIoControlFile`函数。这个函数通常用于从网络驱动程序请求数据。
2. **检测TCP/UDP连接枚举请求**：
   - 当这个函数被用来请求`\Device\Nsi`驱动程序的TCP和UDP连接列表时（通过特定的I/O控制代码，如0x12001b），R77将介入处理。
3. **修改连接列表**：
   - R77会检查和修改返回的TCP和UDP连接列表。对于需要隐藏的连接，R77会从列表中移除这些条目。
   - 接着，R77会调整剩余连接的顺序，并减少报告的总连接数，以确保隐藏操作不影响列表的一致性。

4.7.4 TCP/UDP隐藏代码示例（概念性）

```
// 伪代码：隐藏TCP/UDP连接
NTSTATUS HookedNtDeviceIoControlFile(...) {
    NTSTATUS status = OriginalNtDeviceIoControlFile(...);

```

```
    if (status == STATUS_SUCCESS && IoControlCode == IOCTL_TCP_UDP_QUERY)
{
        // 检查并过滤TCP/UDP连接列表
        FilterNetworkConnections(...);
    }

    return status;
}

void FilterNetworkConnections(NetworkConnectionsList* connections) {
    for (int i = 0; i < connections->Count; i++) {
        if (ShouldHideConnection(connections->Connections[i])) {
            // 移动后续连接，覆盖当前需要隐藏的连接
            MoveMemory(&connections->Connections[i], &connections-
>Connections[i + 1],
                       (connections->Count - i - 1) *
sizeof(NetworkConnection));
            connections->Count--;
            i--; // 重新检查当前索引，因为连接已经移动
        }
    }
}
```

- **代码解释**

1. **拦截网络I/O控制函数**:
   - 代码中拦截了`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，关键是识别出请求TCP/UDP连接列表的特定I/O控制代码（如`IOCTL_TCP_UDP_QUERY`）。
2. **检查状态和I/O控制代码**:
   - 在调用原始函数后，代码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. **过滤连接列表**:
   - 如果是TCP/UDP连接列表请求，代码调用`FilterNetworkConnections`函数来处理连接列表。
   - 在此函数中，它遍历连接列表，对每个连接应用`ShouldHideConnection`（一个假设的函数）来判断是否应该隐藏该连接。
4. **隐藏特定连接**:
   - 对于需要隐藏的连接，代码通过将后续连接向前移动来覆盖当前连接，并更新连接列表的计数。
   - 这种方法确保隐藏的连接不会出现在返回给调用者的列表中。

通过上述方法，R77能够有效地在系统级别**隐藏特定的TCP和UDP连接**，从而在网络监控工具中使它们不可见。这显示了R77在网络通信方面操纵操作系统行为的能力。

5 实验结论及心得体会

5.1 实验结论

1. **功能验证**：通过本次实验，我们成功地验证了R77 rootkit在隐藏进程、文件、注册表项以及TCP/UDP连接方面的强大功能。我们观察到，即使在标准监控工具中，被隐藏的对象也完全不可见，这验证了R77在系统级别进行拦截和修改操作的能力。
2. **隐蔽性和灵活性**：R77展示了其在隐藏技术方面的隐蔽性和灵活性。不仅能够隐藏以特定前缀命名的对象，还可以根据需要隐藏任何指定的进程、文件或网络连接。
3. **技术深度**：R77通过拦截关键系统调用和修改操作系统的内部数据结构实现了其功能。这一过程无需修改被隐藏对象的物理状态或数据，显示出其技术的高度复杂性和深度。

5.2 心得体会

- **安全意识提升**：这次实验不仅提升了我的技术知识，也加深了我对操作系统安全性的理解。特别是在理解如何防范和识别类似rootkit的工具方面。
- **责任与道德**：学习和实验这样的工具使我意识到作为一个技术人员的责任。我们必须认识到这种技术可能被滥用的风险，并在使用时始终坚持道德和法律的界限。
- **对隐私的尊重**：这个实验也强化了我对个人隐私和数据保护的重视。在当今数字化时代，保护用户的隐私和数据安全比以往任何时候都更为重要。
- **技术的双刃剑**：最后，这次实验使我更深刻地认识到技术本身是中立的，它既可以被用于正面目的，也可能被用于负面目的。作为技术从业者，我们应该努力确保我们的技术知识和技能被用于促进社会的积极发展。