

进程隐藏实验报告

|  |  |
| --- | --- |
| 学生姓名 | maybeLocalhost |
| 学 号 |  |
| 专业班级 |  |
| 指导教师 |  |
| 学 院 | 计算机学院 |
| 完成时间 | 2020.12 |

目录

[一、实验简介 1](#_Toc62743384)

[1.1 实验目的 1](#_Toc62743385)

[1.2 实验内容 1](#_Toc62743386)

[1.3 实验环境 1](#_Toc62743387)

[二、实验原理 1](#_Toc62743388)

[2.1 隐藏技术 1](#_Toc62743389)

[2.2 进程伪装 2](#_Toc62743390)

[2.3 进程隐藏 2](#_Toc62743391)

[三、实验过程 3](#_Toc62743392)

[3.1 利用Windbg查看进程 3](#_Toc62743393)

[3.2 进程隐藏 6](#_Toc62743394)

[3.2.1 函数介绍 6](#_Toc62743395)

[3.2.2 实现原理 8](#_Toc62743396)

[3.2.3 编码实现 8](#_Toc62743397)

[四、实验结果 10](#_Toc62743398)

[五、实验总结与问题的解决 10](#_Toc62743399)

[5.1 实验总结 10](#_Toc62743400)

[5.2 遇到问题的解决方案 11](#_Toc62743401)

[六、参考文献 11](#_Toc62743402)

# 一、实验简介

## 1.1 实验目的

在本实验中，学生需要试利用windbg、DDK或SoftICE查看EProcess和PEB中活动进程相关信息，绘制出当前活动进程双向链表在内核态和用户态下的进程链表结构，并设计“断链”方法利用这两个结构体实现自己任意指定进程在任务管理器中的隐藏。

## 1.2 实验内容

本实验涵盖以下主题：

1. 利用Windbg查看进程；
2. 进程隐藏。

## 1.3 实验环境

Windows 10、Windbg

# 二、实验原理

## 2.1 隐藏技术

为了永久地驻留在用户计算机上，病毒木马需要披上厚厚的伪装。因为它们深深地明白，只有活下去，才会有未来。所以，就像潜伏在敌人内部的间谍一样，病毒木马需要巧妙地隐藏、伪装自己，小心翼翼地窃取计算机用户的数据和隐私，生怕因留下的蛛丝马迹而被用户或是杀软察觉。暴露的后果轻则是在该计算机上连根拔起，重则是溯源，连老巢都让人端掉。所以，做好隐藏、伪装工作是病毒木马长期驻留在用户计算机上的关键。

但是，也并非所有的病毒木马都会故意地隐藏或伪装，有些病毒木马由于自身植入技术、启动技术或是自启动技术较为隐蔽，不易被用户察觉或是被杀软检测到，所以不需要额外的隐藏伪装，也能达到隐藏的目的。

目前常见的隐藏、伪装技术主要有以下四种：

* 进程伪装：通过修改指定进程PEB中的路径和命令行信息实现伪装。
* 傀儡进程：通过进程挂起，替换内存数据再恢复执行，从而实现创建“傀儡”进程。
* 进程隐藏：通过HOOK函数ZwQuerySystemInformation实现进程隐藏。
* DLL劫持：通过#pragma comment指令直接转发DLL导出函数或者通过LoadLibrary和GetProcAddress函数获取DLL导出函数并调用。

## 2.2 进程伪装

对于病毒木马来说，最简单的进程伪装方式就是修改进程名称。例如，将本地文件名称修改为svchost.exe、services.exe等系统进程，从而不被用户和杀软发现。进程伪装可以修改任意指定进程的信息，即该进程信息在系统中显示的是另一个进程的信息。这样，指定进程与伪装进程的信息相同，但实际上，它还执行着原来进程的操作，这就达到了伪装的目的。

## 2.3 进程隐藏

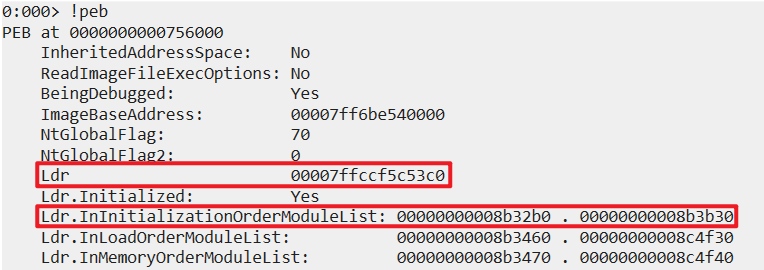
实现进程隐藏的方法有很多，一种较为直接的隐藏方式是通过HOOK API函数实现。在Windows中，用户程序的所有操作都是基于WIN32 API来实现的，例如使用任务管理器查看进程等操作，这便给了病毒木马大显身手的机会。它通过HOOK技术拦截API函数的调用，并对数据进行监控和篡改，从而达到不可告人的目的。

其中，API HOOK技术是一种改变API执行结果的技术，微软自身也在Windows操作系统里面使用了这个技术，如Windows兼容模式等。API HOOK 技术并不是病毒木马专有的技术，但是病毒木马经常使用它来达到隐藏自己的目的。

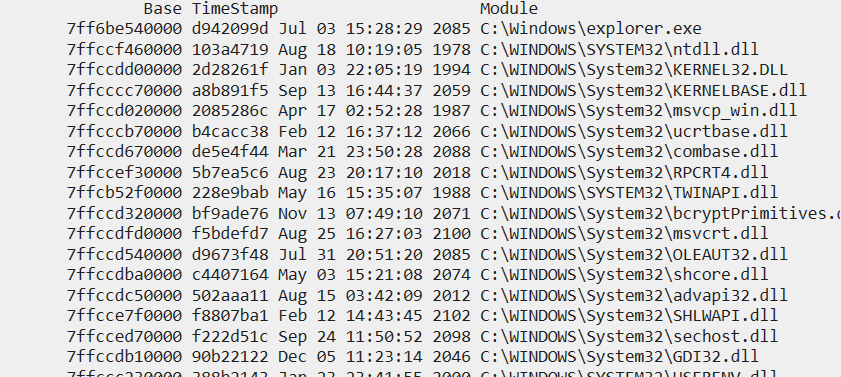
# 三、实验过程

## 3.1 利用Windbg查看进程

所有的进程都维护一个PEB（进程环境块结构）；这个结构体可以在搜索引擎查找，也可以使用windbg工具获取；首先用windbg加载一个进程。我们使用“!peb”指令，解析当前进程的PEB结构：

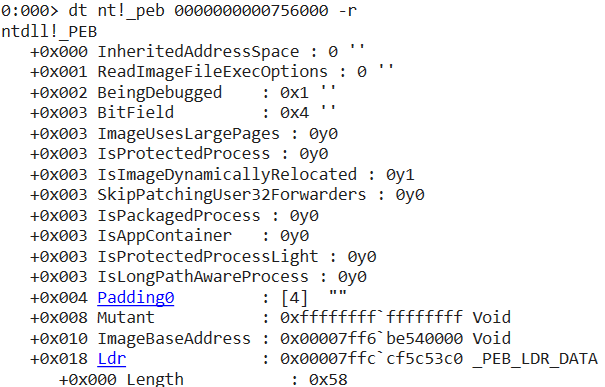


图表 1 解析当前进程PEB结构



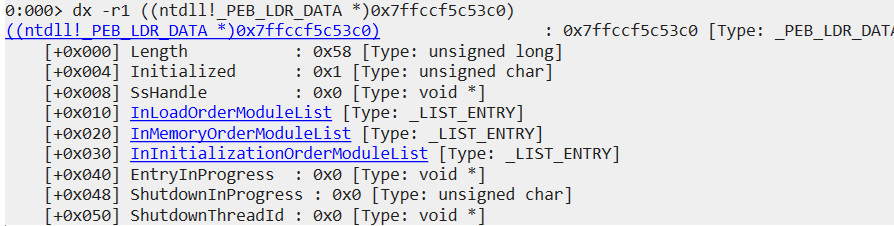
图表 2 DLL基地址

可以看到PEB的基地址0x00756000、LDR的基地址0x7ffccf5c53c0和LDR中InInitializationOrderModuleList的地址范围。在在PEB结构中，维护着\_PEB\_LDR\_DATA（偏移0xC）结构保存着进程加载的所有的Module信息（包括Kernel32.dll，基地址）下面我们通过“dt nt!\_peb 00756000 -r”的指令来获取LDR：



图表 3 详细列出PEB结构

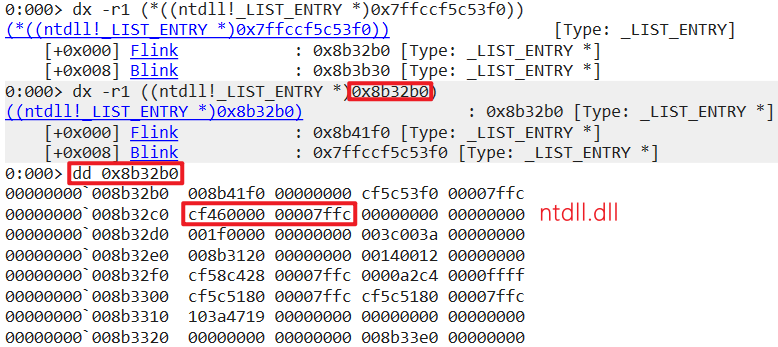
根据获取到的LDR去获取双向链表结构：



图表 4 获取LDR中的双向链表结构

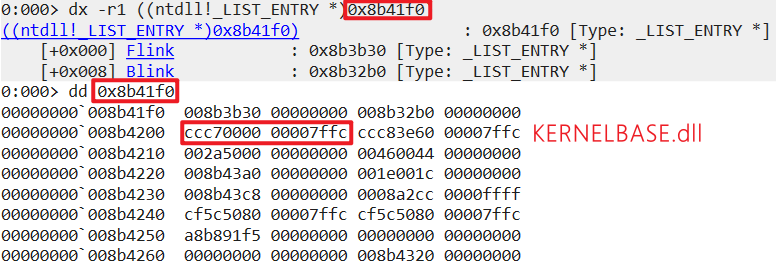
接下来我们需要尝试从InInitializationOrderModuleList链表中获取Kernel32.dll基地址（通过FLINK指针遍历链表）。

第一个节点，对应的Module的基地址是0x8b32b0，发现是ntdll.dll：



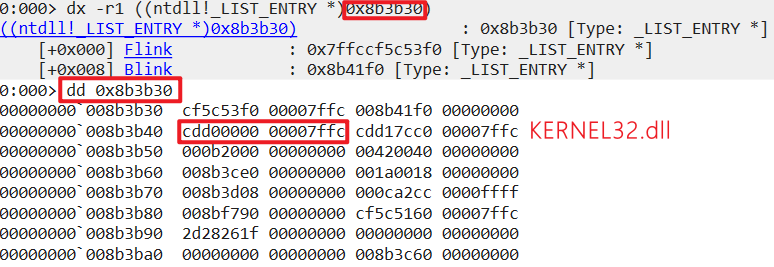
图表 5 找到ntdll.dll

NEXT指向的下一个节点，对应的Module基地址是0x8b41f0，对照图5-6发现是KERNELBASE.dll：



图表 6 找到KERNALBASE.dll

继续向下找，发现第三个节点对应的Module基地址正是Kernel32.dll的基地址0x8b3b30：



图表 7 找到KERNEL32.dll

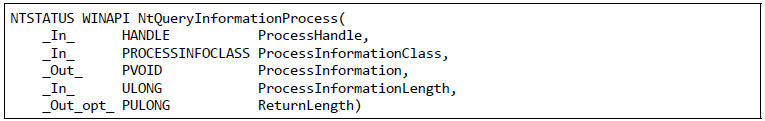
通过上述过程，我们完成了从PEB中获取某Module基地址的过程，kernel32.dll在不同Windows系统上InInitializationOrderModuleList的位置有些区别，上述过程在Windows 10 64bits完成，发现kernel32.dll在InInitializationOrderModuleList的第三个节点的位置。

## 3.2 进程隐藏

### 3.2.1 函数介绍

1. NtQueryInformationProcess函数：获取指定进程的信息。

* 函数声明

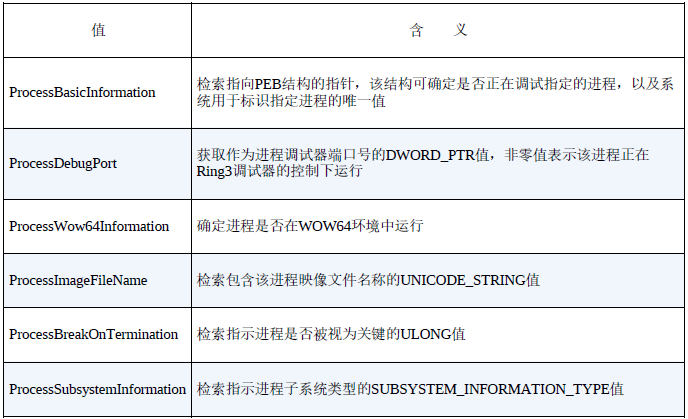


图表 8 NtQueryInformationProcess函数声明

* 参数

ProcessHandle [in]：要获取信息的进程句柄。

ProcessInformationClass [in]：要获取的进程信息的类型。该参数可以是PROCESSINFOCLASS枚举中的以下值之一。



图表 9 PROCESSINFOCLASS枚举值

ProcessInformation [out] ：指向由应用程序提供的缓冲区指针，函数写入请求的信息。所写信息的大小取决于ProcessInformationClass参数的数据类型。当ProcessInformationClass参数为ProcessBasicInformation时，ProcessInformation参数指向的缓冲区应足够大，以容纳具有以下布局的单个PROCESS\_BASIC\_INFORMATION结构。

ProcessInformationLength [in]：ProcessInformation参数指向的缓冲区大小（以字节为单位）。

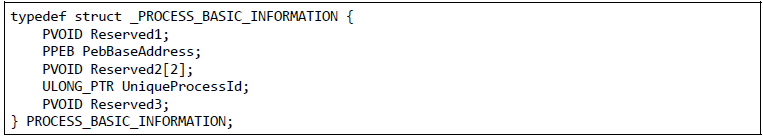
ReturnLength [out, optional]：指向变量的指针，其中函数返回所请求信息的大小。如果函数成功，则这是由ProcessInformation参数指向缓冲区的信息大小，但是如果缓冲区太小，则这是成功接收信息所需的最小缓冲区大小。

返回值：该函数返回一个NTSTATUS成功或错误代码。

* **此函数没有关联的导入库，所以必须使用LoadLibrary和GetProcessAddress函数从Ntdll.dll中获取该函数地址。**

1. PROCESS\_BASIC\_INFORMATION结构体

* 结构体定义



图表 10 PROCESS\_BASIC\_INFORMATION结构体定义

* 成员

PebBaseAddress成员指向PEB结构。

UniqueProcessId成员指向该过程的系统唯一标识符。最好使用GetProcessId函数来检索这些信息。

该结构的其他成员保留以供操作系统内部使用。

### 3.2.2 实现原理

进程伪装的原理，就是修改指定进程环境块中的进程路径以及命令行信息，从而达到进程伪装的效果。所以，实现的关键在于进程环境块的获取。由上述的函数介绍可知，可以通过ntdll.dll中的导出函数NtQueryInformationProcess来获取指定进程的PEB地址。获取目标进程的PEB之后，并不能直接根据指针来读写内存数据，因为该程序进程可能与目标进程并不在同一个进程内。由于进程空间独立性的缘故，所以需要通过调用WIN32 API函数ReadProcessMemory和WriteProcessMemory来读写目标进程内存。

具体的实现流程如下所示。

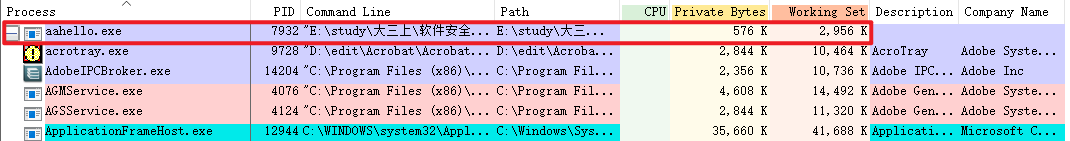
1. 根据进程的PID号打开指定进程，并获取进程的句柄。
2. 从ntdll.dll中获取NtQueryInformationProcess函数的导出地址，因为该函数没有关联导入库，所以只能动态获取，这个函数是这个程序功能实现的关键步骤。
3. 使用NtQueryInformationProcess函数获取指定的进程基本信息PROCESS\_BASIC\_INFORMATION，并从中获取指定进程的PEB。
4. 最后，就可以根据进程环境块中的ProcessParameters来获取指定进程的RTL\_USER\_PROCESS\_PARAMETERS 信息，这是因为PEB的路径信息、命令行信息存储在这个结构体中。调用ReadProcessMemory和WriteProcessMemory函数可以修改PEB中的路径信息、命令行信息等，从而实现进程伪装。经过上述操作，便可完成进程伪装工作。

### 3.2.3 编码实现

1. // 修改指定PEB中的路径和命令行信息, 实现进程伪装
2. **BOOL** DisguiseProcess(**DWORD** dwProcessId, **wchar\_t** \*lpwszPath, **wchar\_t** \*lpwszCmd)
3. {
4. // 打开进程获取句柄
5. **HANDLE** hProcess = ::OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, dwProcessId);
6. **if** (NULL == hProcess){
7. ShowError("OpenProcess");
8. **return** FALSE;
9. }
10. typedef\_NtQueryInformationProcess NtQueryInformationProcess = NULL;
11. PROCESS\_BASIC\_INFORMATION pbi = { 0 };
12. PEB peb = { 0 };
13. RTL\_USER\_PROCESS\_PARAMETERS Param = { 0 };
14. **USHORT** usCmdLen = 0;
15. **USHORT** usPathLen = 0;
16. // 需要通过 LoadLibrary、GetProcessAddress 从 ntdll.dll 中获取地址
17. NtQueryInformationProcess = (typedef\_NtQueryInformationProcess)::GetProcAddress(
18. ::LoadLibrary("ntdll.dll"), "NtQueryInformationProcess");
19. **if** (NULL == NtQueryInformationProcess){
20. ShowError("GetProcAddress");
21. **return** FALSE;
22. }
23. // 获取指定进程的基本信息
24. NTSTATUS status = NtQueryInformationProcess(hProcess, ProcessBasicInformation, &pbi, size
25. of(pbi), NULL);
26. **if** (!NT\_SUCCESS(status)){
27. ShowError("NtQueryInformationProcess");
28. **return** FALSE;
29. }
30. // 获取指定进程基本信息结构中的PebBaseAddress
31. ::ReadProcessMemory(hProcess, pbi.PebBaseAddress, &peb, **sizeof**(peb), NULL);
32. // 获取指定进程环境块结构中的ProcessParameters, 注意指针指向的是指定进程空间
33. ::ReadProcessMemory(hProcess, peb.ProcessParameters, &Param, **sizeof**(Param), NULL);
34. // 修改指定PEB中的命令行信息, 注意指针指向的是指定进程空间
35. usCmdLen = 2 + 2 \* ::wcslen(lpwszCmd);
36. ::WriteProcessMemory(hProcess, Param.CommandLine.Buffer, lpwszCmd, usCmdLen, NULL);
37. ::WriteProcessMemory(hProcess, &Param.CommandLine.Length, &usCmdLen, **sizeof**(usCmdLen), NULL);
38. // 修改指定PEB中的路径信息, 注意指针指向的是指定进程空间
39. usPathLen = 2 + 2 \* ::wcslen(lpwszPath);
40. ::WriteProcessMemory(hProcess, Param.ImagePathName.Buffer, lpwszPath, usPathLen, NULL);
41. ::WriteProcessMemory(hProcess, &Param.ImagePathName.Length, &usPathLen, **sizeof**(usPathLen), NULL);
42. **return** TRUE;
43. }

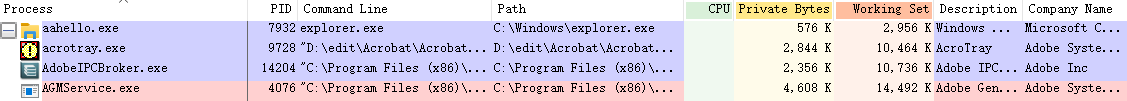
# 四、实验结果

修改64位程序aahello.exe的PEB信息来伪装成资源管理器explorer.exe。为了更好地对比，在运行伪装程序之前，先使用进程查看器ProcessExplorer.exe查看aahello.exe隐藏前的进程信息。



图表 11 伪装前aahello.exe的进程信息

然后，运行进程伪装程序，修改64位测试程序aahello.exe的PEB信息，实现进程伪装。程序执行后提示成功，再次运行进程查看器ProcessExplorer.exe查看进程信息，可以发现测试程序 aahello.exe成功地隐藏成了explorer.exe 资源管理器进程。



图表 12 伪装后aahello.exe的进程信息

# 五、实验总结与问题的解决

## 5.1 实验总结

通过本次实验，我学习了Windbg的基本使用方法，加深了PEB结构的了解，并能够编写简单的进程隐藏程序。其中，隐藏则主要是修改指定进程环境块中的进程路径以及命令行信息，从而达到进程伪装的效果。其实现的关键在于进程环境块的获取。

## 5.2 遇到问题的解决方案

修改PEB实现进程伪装的原理不难理解，但是有个容易出错的地方，它就是一定要区分指针指向的是指定进程的空间还是本程序的空间，若指向其他进程空间，则一律使用ReadProcessMemory和WriteProcessMemory函数进行数据读写。

同时也要注意系统位数问题，如果PEB修改程序运行在64位系统上，那么程序就要编译为64位程序；如果PEB修改程序运行在32位系统上，则程序要编译为32位程序，否则程序不能伪装成功。

除此外，注意在读写其他进程的时候，注意要使用ReadProcessMemory/WriteProcessMemory进行操作，每个指针指向的内容都需要获取，因为指针只能指向本进程的地址空间，必须要读取到本进程空间。要不然一直提示位置访问错误。

# 六、参考文献

1. 甘迪文. 《Windows黑客编程技术详解》. 人民邮电出版社，2018年12月
2. Mikasys. 4.PEB断链隐藏模块. https://blog.csdn.net/Mikasys/article/details/109260609
3. Mikasys. 7.HOOK NtTerminateProcess驱动保护进程. https://blog.csdn.net/Mikasys/article/details/109501149
4. 流沙. 通过断链隐藏模块(DLL). http://www.gyarmy.com/post-613.html
5. 黑月教主的空间. 【旧文章搬运】修改PEB,断链隐藏模块成功. https://www.cnblogs.com/achillis/p/10179826.html
6. Microsoft. UNICODE\_STRING structure (subauth.h). https://docs.microsoft.com/en-us/windows/win32/api/subauth/ns-subauth-unicode\_string