系统内核的潜伏者

内核Rootkit初探

[0x01 写在前面 2](#_Toc2935)

[程序效果图 3](#_Toc19737)

[0x02 内核知识库：Windows调用过程篇 4](#_Toc9411)

[0x03 混淆内核的黑手：劫持调用表篇 6](#_Toc17474)

[①去页面保护 7](#_Toc15810)

[②备份正常SSDT表 7](#_Toc11520)

[③安装钩子 9](#_Toc10594)

[0x04 底层的叛徒：恶意内核钩子篇 11](#_Toc26569)

[0x05 在系统内横冲直撞的凶灵：木马功能篇 13](#_Toc25569)

[①通过木马程序安装驱动 14](#_Toc5091)

[②连接驱动 14](#_Toc30832)

[③与驱动通信 15](#_Toc2238)

[程序示意图 16](#_Toc2786)

[0x06 未知攻，焉知防？：劫持防御之探索篇 18](#_Toc3065)

[参考文献 18](#_Toc15178)

0x01 写在前面

相信大部分人和我一样，觉得但凡与内核或者是rootkit沾边的项目，都认为需要很深厚的系统功底和内核开发经验才能驾驭，加上有关windows系统ring0的API又总是生涩不堪，而大部分windows rootkit程序往往闭源，所以往往让我们与rootkit越来越远。至少在我写出属于我自己的rootkit之前是这样认为的，本文的目的就在于把rootkit推荐到各位面前，你就会发现，其实rootkit也不是那么高大上的玩意，跟随我的脚步一步步走，构建属于自己的rootkit木马，不再遥不可及。

rootkit一般来说分为两种，基于微软未公开函数（以Zw和以Nt开头的系统函数），基于系统未公开漏洞的的，前者在利用OS本身提供给开发者的API进行隐藏和侵入系统内核，所谓未公开，是指微软对于一些系统函数，并没有给出相关系统文档和函数说明和定义，（可能是这些函数功能太强大了，防止被一些别有用心的人利用），而后者利用栈溢出，UAF等漏洞向内核空间加载代码。后者隐蔽性，稳定性更好，开发成本自然更高。

本文要介绍的是基于SSDT挂钩的木马，属于前者，SSDT劫持原理实现相对简单，并且有较强的隐蔽性，适于入门，或快速开发自己需要的rootkit。

代码中有很多引用的地方，尤为引用了一个人的SSDT框架。由于SSDT劫持技术已经比较成熟，所以可能网上有很多类似代码 (不过大部分都不能编译运行)。

本设计中实现功能如下

**Trojan木马实体**

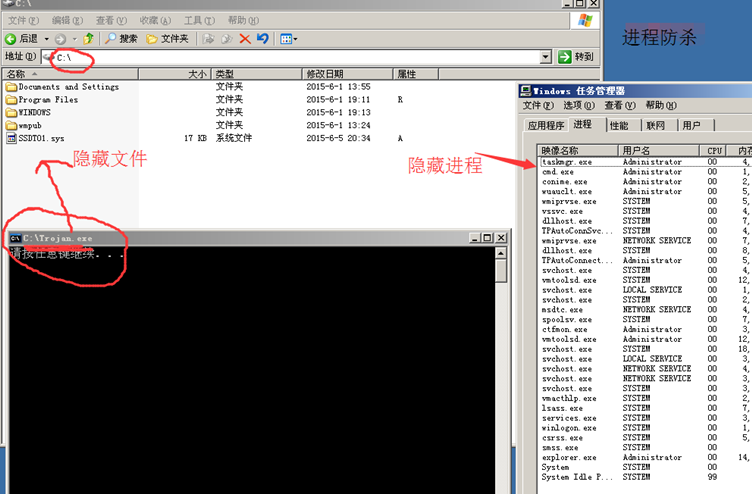
**文件隐藏**

进程防杀

进程隐藏

其中红字粗体为完全原创部分。

程序效果图



0x02 内核知识库：Windows调用过程篇

想要构建一个SSDT rootkit ，我们首先要思考的问题就是要何时介入一个系统正常的调用，并且劫持它，

众所周知，我们的系统里面有各种各样的“表”，也就是系统调用表。

比如：

IAT 用户态 模块导入的Windows DLL

IDT 内核态 硬件相关

GDT 内核态 内存段信息

SSDT 内核态 存储系统调用函数地址

IRP 内核态 处理IRP驱动使用的函数

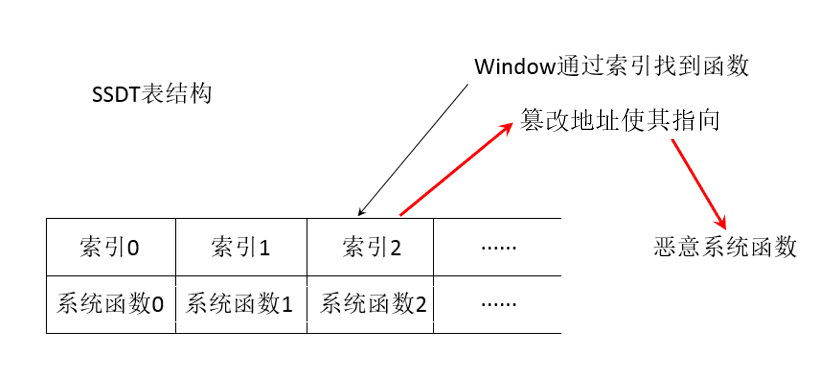
这些表直观的说，就是一个一个数组，这些数组存储例程的地址，（所谓例程，例程的作用类似于函数，但含义更为丰富一些。例程是某个系统在ring0对外提供的功能接口或服务的集合。比如操作系统的API、服务等就是例程）

而我们要做的事情，就是将我们设计的例程地址代替现有的，合法的调用表地址，（也就是挂钩），通过这样的手段钩住系统函数，这样每当这个系统函数被调用的时候，实际上先调用的是我们的函数，通过我们预设的逻辑篡改通过的信息。达到隐藏自己或是隐藏木马程序的目的。

所以我们的逻辑的第一步应该是这样

NtSystemFunc \* NtQueryFile; //系统原有函数指针

NtQueryFile = TrojanNtQueryFile; //通过SSDT表替换成注入到内核空间的恶意函数



那么这张表到底怎么从内核“弄到”我们的驱动程序中呢？

回答是：我们只需要一个声明和一个extern导出

**我们先根据MSDN给出的结构声明一个SSDT表**



**然后从ntoskrnl.exe将SSDT表“装”到我们的指针中就好**

//导出由 ntoskrnl.exe 所导出的 SSDT

extern PKSERVICE\_TABLE\_DESCRIPTOR KeServiceDescriptorTable;

0x03 混淆内核的黑手：劫持调用表篇

而篡改一个系统调用表之前，我们

①需要去掉Windows的页保护，以便让我们修改SSDT表。

②备份原有SSDT表

③安装钩子（SSDT表操作）

看起来很头痛呢！ 没关系，我们一步步来就好

（感谢@小宝马的爸爸 他的博客实现了一个有关SSDT操作的框架，而我的rootkit调用了他的框架）

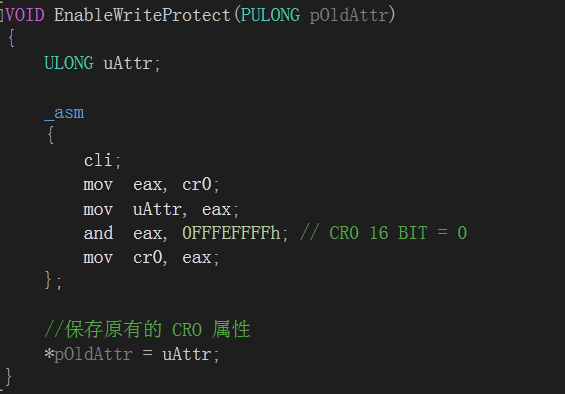
①去页面保护

为了安全起见，Windows XP及其以后的系统将一些重要的内存页设置为只读属性，这样就算有权力访问该表也不能随意对其修改，例如SSDT、IDT等。但这种方法很容易被绕过，我们只要将这些部分修改为可写属性就可以了，不过当我们的事情做完后记得把它们恢复为只读属性，不然会造成一些很难预料到的后果。

cr0寄存器到486的处理器版本被加入了“写保护”（Write Protect，**WP**）位，WP位控制是否允许处理器向标记为只读属性的内存页写入数据。

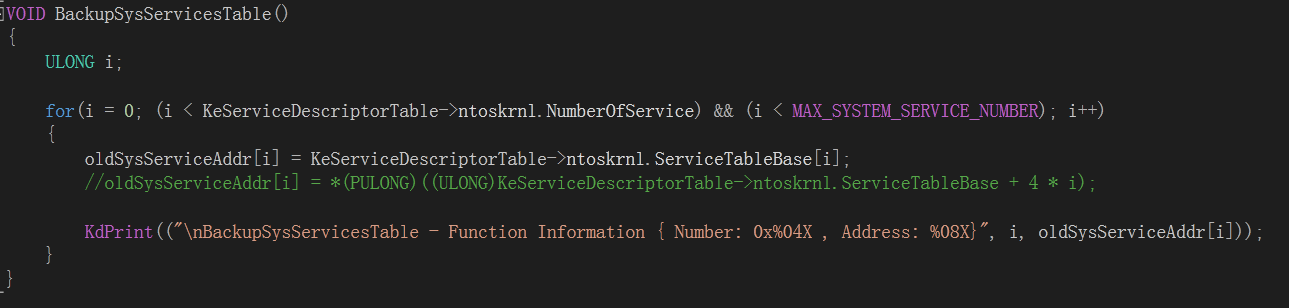
所有我们要做的是修改cr0寄存器

我们用汇编实现



②备份正常SSDT表

前面提到，**SSDT表实际上就是数组**，所以我们只需要以操作数组的方法操作这张表就好



相信大家从上面的图片注意到，我们需要得到一个索引，并通过修改索引指向的内容达到劫持系统函数的目的。SSDThook框架的作者所用的算法 涉及到了以Zw和以Nt开头的系统函数，此处我们先来讲讲这两者的关系。

首先要说的是，windows系统里面有两组Zw&Nt，分别位于ring3和ring0态。

ring3态的Zw&Nt位于Ntdll

我们直接让kd告诉我们两者的关系把

kd> u Ntdll! ZwCreateFile L4

ntdll!ZwCreateFile:

77f87cac b820000000 mov eax,0x20

77f87cb1 8d542404 lea edx,[esp+0x4]

77f87cb5 cd2e int 2e

77f87cb7 c22c00 ret 0x2c

kd> u Ntdll! NtCreateFile L4

ntdll!ZwCreateFile:

77f87cac b820000000 mov eax,0x20

77f87cb1 8d542404 lea edx,[esp+0x4]

77f87cb5 cd2e int 2e

77f87cb7 c22c00 ret 0x2c

可以看出，Zw和Nt在ring3下其实是一样的，按照网上面的说法叫做“Nt只是Zw的别名函数”

而在ring0态就不一样了

下面给出ring0的Zw实现

lkd> u nt!ZwOpenProcess

nt!ZwOpenProcess:

804de044 b87a000000    mov     eax,7Ah

804de049 8d542404       lea     edx,[esp+4]

804de04d 9c             pushfd

804de04e 6a08          push 8

804de050 e8dc150000    call nt!KiSystemService (804df631)

804de055 c21000        ret     10h

可以看到Zw函数在ring0下实际上就是把7Ah（OpenProcess函数的索引号）传递给SSDT表 KeServiceDescriptorTable[07Ah]

而ring0中的Nt函数才是真正的函数实现。

所以我们要篡改SSDT表，会导致调用了Zw函数去工作的进程被我们劫持和蒙骗，而直接调用Nt函数的驱动是不会被劫持的比如说冰刃，（所幸绝大多数程序都是从Zw进行调用的）

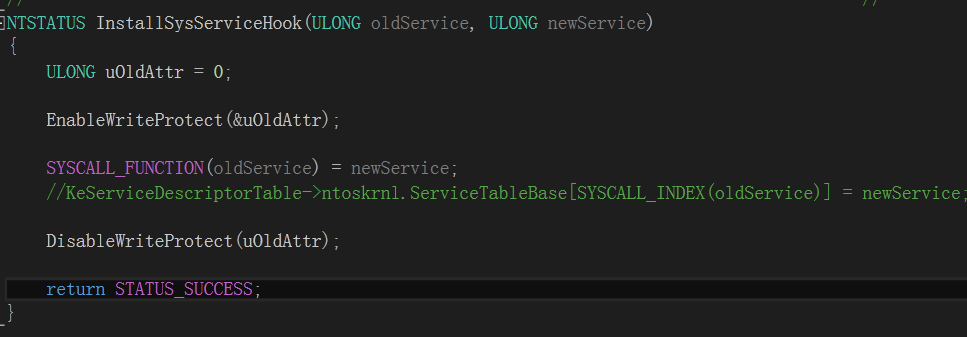


那么通过如上两个宏定义函数，我们通过服务名称（NtQuerySystemInformation）计算得到SSDT索引号，再通过索引号得到在KeServiceDescriptorTable中函数的虚拟地址

③安装钩子

第三个也就是我们的核心啦，安装SSDT钩子。

前文里面我们备份了SSDT里面所有内容，又能计算出要劫持函数的索引号和真实地址，现在要做的事情就是替换掉他，也就是挂钩。



这里的oldService 和newService都是地址变量，唯一的区别在于oldService是Zw型，而newService是Nt型的，这很好理解，我们两个宏定义的目的就是通过Zw函数找到Nt函数的地址，并将newServie赋值到KeServiceDescriptorTable->ntoskrnl.ServiceTableBase[ZwFunc]，这样就实现了我们的SSDT表操作了。

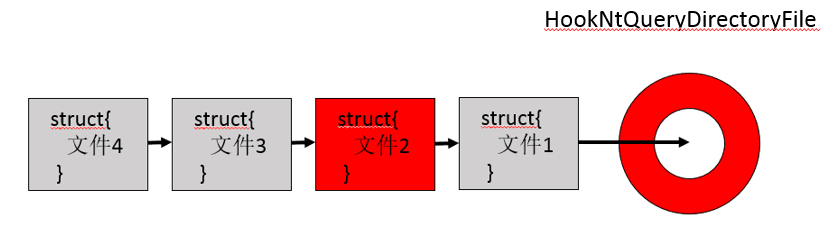
0x04 底层的叛徒：恶意内核钩子篇

从刚才到现在，我们实现了已经实现了对SSDT表的自由操作，这使得我们能够自如地替换表内的任意内容。而我们现在需要做的，就是**构造我们自己的恶意函数，并且注入内核空间，并替换掉原有的函数**！

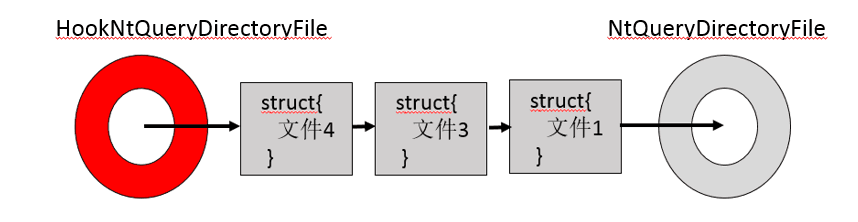
接下来我将以ZwQueryDirectoryFile为例子 ，介绍该如何构造一个挂钩函数。

首先要介绍的是windows对文件，进程等信息的处理机制，**WinNT下这些信息都被当做一个个结构体**，也就是说当我们打开一个文件夹的时候，资源浏览器会调QueryDirectoryFile来查询我们的文件内容，然后这一个个实体（Entry）就会进入这个函数然后再通过指针操作的方式传递出来。

在下图中，我们想要隐藏的文件是那个红色的结构体



首先它会经过我们的hook函数，然后这个线性链表的红色文件会被减掉



这样进行正常调用的时候，内核API终于处理不到我们想要隐藏的文件。

具体hook代码如下

/\*

\*

\* Name: NTSTATUS NtQueryDirectoryFile()

\*

\* Descripion: 自定义的 NtQuerySystemInformation，用来实现 Hook Kernel API

\*/

NTSTATUS HookNtQueryDirectoryFile(



代码中我作了很详细的注释。 可以看到实际上**hook函数就是对FileInformation这个结构体进行修剪**。

如此同理，我们可以分别实现隐藏进程，进程防杀。

0x05 在系统内横冲直撞的凶灵：木马功能篇

我们现在已经完成了驱动的编写，接下来我们要做的有这么三件事情：

①通过木马程序安装驱动

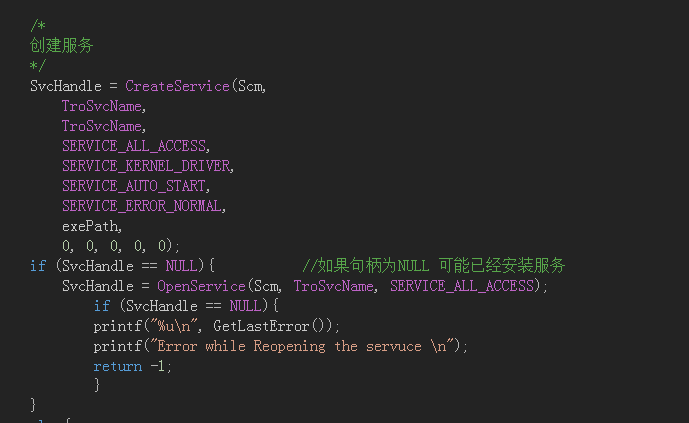
②连接驱动

③与驱动通信，下达命令

我们来一件件完成

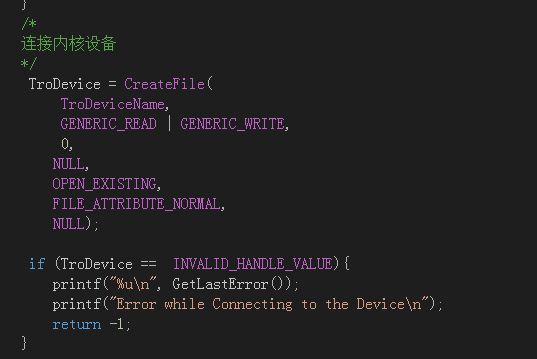
①通过木马程序安装驱动

首先我们通过木马安装驱动



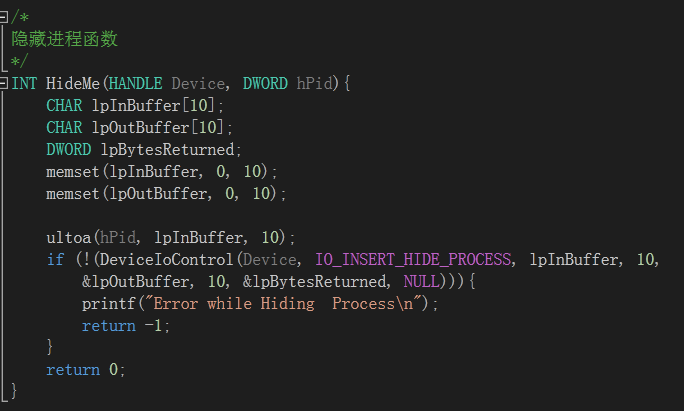
②连接驱动

连接驱动设备



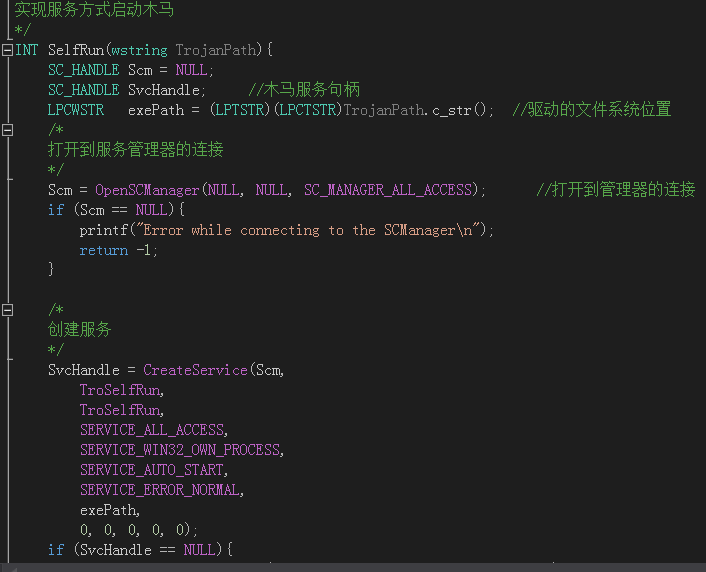
③与驱动通信

接下来我们需要和驱动程序通信下达我们的隐藏指令



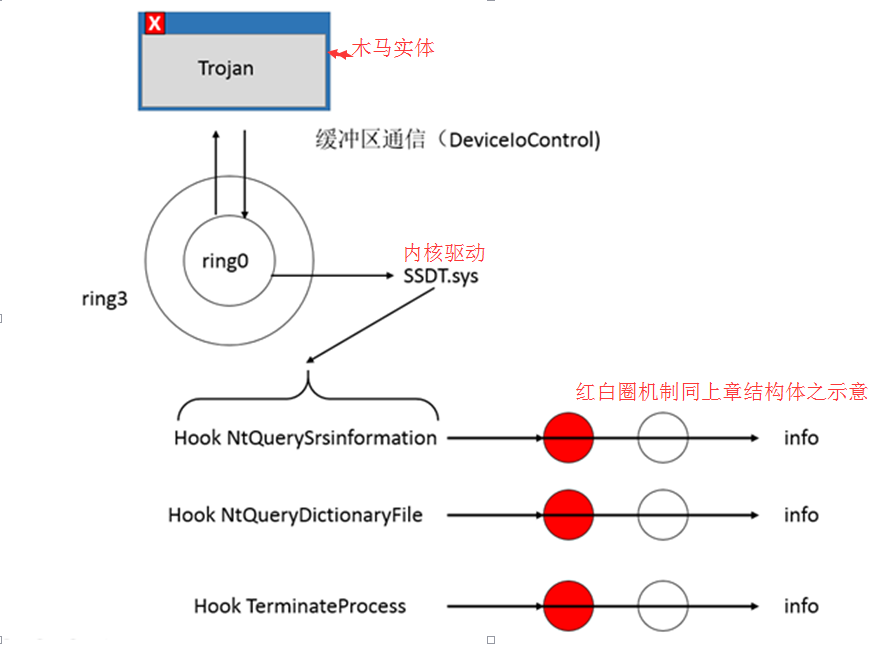
这里我通过缓冲区通信(DeviceIoControl)向驱动下达指令。

一个ROOTKIT除了能够隐藏自己以外，自启动也是很重要的，我通过服务的形式安装到windows



这样一个rootkit初步就算是完成了。

程序示意图



0x06 未知攻，焉知防？：劫持防御之探索篇

其实大作业做这个的目的，一方面想入门WINDOWS核心编程，看看内核的世界长啥模样，第二方面自然是初探内核安全。

既然了解攻击的目的是更好地防御，那么怎么防止SSDT被劫持，或者说在劫持SSDT之后我们如何正常地找到木马，并且杀掉木马呢？

SSDT其实是在我们由ring0的Zw\*调用到Nt\*的时候发起的。也就是说ring3下如何调用都无法绕过ssdt hook 而ring0下通过驱动直接调用Nt函数则不会被hook拦截。

不过我马上又去找怎么让直接访问Nt函数也不被发现的攻击手段了。。。

当然，这是后话

参考文献

<http://www.cnblogs.com/hongfei/archive/2013/06/18/3142162.html>

http://www.cnblogs.com/BoyXiao/archive/2011/09/03/2164574.html

http://bbs.pediy.com/showthread.php?t=156865

<http://www.cnblogs.com/jack204/archive/2011/12/06/2278392.html>

<http://bbs.pediy.com/showthread.php?t=176477>

[https://msdn.microsoft.com](https://msdn.microsoft.com/)