Windows rootkit of 2005 part 1

2005年，rootkit技术开始出现在恶意软件中。Windows系统下的rootkit技术的迅速发展这一现象在最近出现的病毒、木马、间谍软件、广告软件等中体现的更加明显。这样，理解rootkit具有什么威胁以及可以采用什么方式来检测rootkit的而已使用就很重要了。

 这篇由3部分组成的系列文章的第一部分将会讨论什么事rootkit以及什么原因导致他们具有危害性。我们将通过研究各种各样的执行模式以及它们和内核的通信方式。文章的第二部分将会阐明最新的Windows下的通过使用虚拟内存钩子技术来提供一个高度的健壮性的rootkit的实现方式。然后在第三部分将会讨论各种rootkit检测技术和专业对策。

rootkit的定义

rootkit是指被入侵者用来在被入侵的计算机系统隐藏自己的行踪并且使得他能够在将来访问被入侵的计算机系统的一个或者一套软件。为了完成这一目标，一个rootkit软件会改变操作系统的指令流或者暗中操作那些操作系统赖以审计和维持的数据集合。

rootkit和漏洞利用不同，它更多是被攻击者在完成初始的漏洞利用之后使用。在很多方式下，rootkit程序比exploit更为有趣，甚至比0-day漏洞更有趣。我们之中的大多数人勉强可以接受的一个事实是存在于我们计算机系统中的漏洞会被不断的挖掘出来，计算机安全都和管理危机有关。一个0-day漏洞相当于子弹，但是从rootkit可以找出很多关于攻击者的信息，比如说他的动机以及如何触发攻击。通过分析rootkit做了什么，我们可以确认入侵者想要窃取什么，入侵者和谁联系，以及入侵者的入侵水准。在分析为什么入侵之前，我们先来分析怎样入侵。

特权模式

Window是基于安全和稳定的思想设计的。内核必须和用户的应用程序隔离以实现内核的保护，但是用户程序需要特定的由内核提供的功能。为了提供这些功能，Windows实现了两种执行模式：用户模式和内核模式。Windows现在仅仅支持这两种模式，虽然intel和AMD的CPU芯片通常支持4种特权模式或权限来保护系统数据和代码以防止他们被低权限的程序程序故意篡改或无意破坏。

应用程序运行于用户模式，处理机在用户模式下不能使用特权指令。

内核模式指的是处理机可以使用计算机系统所有内存和指令的执行模式。例如在系统服务描述符表SSDT中列举的系统服务运行在内核模式，第三方驱动也运行在内核态，因为在很多情况下它们必须调用内核的底层函数、对象、以及由硬件提供的接口。

Windows会标记内存页面来指定要访问指定的内存需要在哪种模式下。用户模式下的rootkit作为一个单独的程序或者包含在已经存在的应用程序之中运行。内核模式下的rootkit就可以得到操作系统的权限并可以破坏整个系统。

执行路径挂钩

为了让rootkit可以改变操作系统的正常的执行路径，rootkit可以采用的一种技术是挂钩。在现在操作系统中，基于系统的灵活性、可扩展性、向后兼容性的设计，由许多地方可以挂钩。通过使用钩子，rootkit可以改变原原本本的操作系统本来应该返回的信息。在Windows操作系统中由很多可以被rootkit挂钩的表，在这篇文章中我们会概括一些。

导入地址表IAT钩子

Windows系统的设计具有高度独立于底层计算机硬件以及和其他的操作环境（例如POSIX）兼容的特性。而且它必须具备很高的灵活性，这样才能保证当底层的操作系统升级时应用程序的开发者不必重新编写代码。Windows编写了一系列子系统来实现这个目的：Win32子系统，POSIX子系统，和OS/2子系统。每个系统都是以一个动态链接库的形式实现的。这些子系统提供了调用存在于内核空间的系统调用的借口，通过使用这些应用编程接口，应用程序开发者就不会直接调用Windows系统调用，而是通过这些子系统的其中一个。这些被应用程序连接的链接库导出可供这些子系统调用的文档化的接口。Win32子系统是应用最普遍的，它是由Kernel32.dll,User32.dll,Gdi32.dll以及Advapi32.dll。Ntdll.dll是一个特殊的由系统支持的供子系统DLL使用的链接库。它提供了封装函数给Windows可执行系统服务，这些服务最终把控制权转移到位于内核的SSDT，在这里完成真正的工作。这些封装函数包含了可以导致用户态向内核态转换的指定框架的代码。

 当一个Windows可执行文件被装在到内存时，PE文件装载程序必须转换PE文件的一个叫做IAT的节。IAT表列出了程序需要载入的DLL和相关的包含在DLL中并会被可执行程序使用的函数。装载程序会定位每一个位于磁盘的DLL并且把他们映射到内存之中。然后，转载程序把这些函数的地址存放在调用这些函数的可执行程序的IAT表中。在IAT中的一般入口地址就是那些由Kernel32.dll和Ntdll.dll导出的函数地址。其他链接库提供了很多有用的韩式并且这些函数会出现在IAT表，比如由WS2\_32.dll导出的套接字函数。由位于内核空间的例如Ntoskrnl.exe和Hal.dll等导出的内核设备驱动也是很重要的函数。

通过修改一个可执行文件的IAT中的入口地址，rootkit可以改变程序的执行流程，并且影响原始函数放会给调用者的信息。例如，假设一个应用程序要罗列位于某个目录下的所有文件，并对他们执行某种操作，这个应用程序会在用户态下以普通程序或者服务的形式运行。我们也假设应用程序时一个Win32子系统下由Kernel32.dll，Username2.dll，Guitar2.dll，和Advapi.dll实现对内核函数调用的应用程序。在Win32子系统下，应用程序首先通过调用函数FindFirstFile函数来罗列位于某个目录下的所有文件。，这个函数由Kernel32.dll导出。当调用成功后FindFirstFile函数返回一个句柄。这个句柄指向这个目录中的所有文件和子目录，并被用于其后调用FindNextFile函数。FindNextFile函数也是一个由Kernel32.dll导出的函数。如果应用程序使用这些导出函数，装载程序将会动态的把Kernel32.dll载入内存并且把这些函数在内存中的地址复制到应用程序的IAT表中。当应用程序调用在FindNextFile函数时，就会查找到IAT表中的一个特定的表项，然后再依据这个表项跳到位于载入内存的Kennel32.dll中的FindNextFile函数的入口地址。查找FindFirstFile也是用同样的方法。Kernel32.dll中FindNestFile再调用Ntdll.dll中的导出函数，Ntdll.dll会在通用寄存器EAX中存入和FindNextFile函数等效的系统服务号，指一过程在NtQueryDirectoryFile函数中实现。Ntdlldll.dll也会把传给FindNextFile函数的参数的用户空间的地址存放在EDX中，Ntdll.dll之后会引发一个INT2E中断或者使用一条SYSENTER指令切换到内核态。

在上述的例子中，rootkit程序可以篡改应用程序的IAT表使得Kennel32.dll中的函数地址被替换成rootkit忠的函数地址。rootkit也可以用同样的方式针对Ntdll.dll。攻击者怎样使用这些技术很大程度上依赖于他的想象力。rootkit可以调用原来的函数然后过滤诸如文件，目录，注册表键值进程等信息。附带说明一下，每个进程有自己的虚拟地址空间，为了修改一个应用程序的IAT表必须跨进程访问。Rechter和Pietrek分别独立的在这一领域做了很多工作。更进一步的关于如何跨进程访问的解释，请参阅参考文献【1】【2】【3】。

系统服务描述符钩子

正如我们之前讨论的，Win32子系统是一个rootkit可以挂钩的地方。然而，这个子系统只是进入内核的一个窗口。操作系统函数实现的一般的地址包含在一个叫做系统服务描述符表SSDT的内核表，它也被称为内核调用表。这些地址指向在Neoskrnl.exe

中实现的类似Ntxxx的函数。内核态的rootkit可以直接改变这些表并且将预期的Ntxxx函数的地址替换成rootkit里面的代码的地址。这样做可以实现强大的功能，因为和IAT钩子挂钩一个单个的程序不同，这种技术会安装一个应用于整个系统的全局钩子从而影响每一个进程。仍然用上面的在IAT钩子那一段中提到的例子，内核的rootkit可以挂钩NtQueryDirectoryFile来隐藏位于本地文件系统的文件和目录。  
内联钩子

内联钩子比IAT钩子或者SSDT钩子更加先进，内联钩子替换了原始函数的若干字节，这和替换存放于一个表中的函数地址不同，后者在我们后面的文章会展示如何轻松的检测它们被修改了。通常内联钩子在原始函数中添加从原始函数无条件跳转到rootkit中的JMP指令。许多WindowsAPI函数都是以下面的前缀开始的：

在进行内联hook时，rootkit将被挂钩函数中即将被它修改的原始数据备份起来，这样就可以保持同样的函数行为。然后，它会把这些原始数据改写成跳往rootkit的JMP指令，注意到rootkit可以安全的改写被挂钩函数的前5个字节，因为很多类型的jmp指令和call指令都是5个字节，即使5个字节在指令的边界上。

这里xx xx xx xx是rootkit的地址，这样rootkit可以跳回到原始的代码地址加上某个偏移对应的地址。并且并且修改原原本本的的操作系统函数返回的信息。

内联钩子可以像应用于rootkit技术一样，可以合法的应用于很多地方，微软研究人员最早是在一个会议上把内联钩子公布出来。现在，微软已经把它的用途扩展到最初的研究之外。他们把它叫做内存补丁，这一技可以对一个系统在不需要重启的条件下打补丁。

分层过滤驱动

分层过滤驱动为rootkit创造了一个新的把它们自己插入操作系统执行流程的地方。设备驱动程序最终处理很多操作系统的重要功能，例如网络通信和文件存储。Windows允许开发者在已有的驱动程序之上堆叠自己的驱动来扩展位于低层的驱动，而不需要重写已有的驱动。许多病毒扫描软件实现了文件过滤来扫面它们打开的文件。操作系统提供的文件驱动把结果传送到病毒扫描程序的文件过滤驱动，在这里对问及那进行扫描。rootkit也可以使用这种分层技术。正是其中的集中可能的情形会改变文件读取和枚举，以及修改socket通信和枚举。

直接内核对象操作

直接内核对象操作依赖于以下事实：即操作系统为了审计会创建内核对象。如果rootkit修改了这些内核对象，它将会破坏那些操作系统相信已经存在于系统中的东西。通过修改一个令牌环对象，rootkit可以改变操作系统相信那些程序可以做特定的行为，从而破坏登录机制。例如，一个叫做FU的rootkit程序修改了代表系统中所有进程的内核对象——在这个进程对象里所有的进程被链在一起。当一个用户进程例如任务管理器通过AOI函数向操作系统查询进程链表时，Windows会遍历它隐藏的进程链表，所以，正如很多应用程序所关注的，进程不见了。

无所不在的rootkit

现在存在很多的rootkit，但是其中的很多彼此相似，。然而，我们大致可以把它们分为两类：钩子技术和使用直接内核操作。在这种分类体系下，Hackdefender【6】是采用hook技术的rootkit软件中最流行的一个。它可以隐藏进程，服务，文件，目录，注册表键值和端口。FU是使用直接内核操作策略的rootkit软件中较流行的一个。然而，FU是作为一个概念证据被写出来的，它不能隐藏自己，并且没有远程通信的通道。FU可以隐藏进程和驱动，也可以提升权限和Windows进程令牌群。

第一部分总结

从历史上说，Windows rootkit技术已经存在了有一段时间了。然而，在去年或前年，它们开始更加容易地被应用到诸如病毒、蠕虫、广告软件和间谍软件之中，Hacker Defender的作者甚至出售了他的rootkit的不会被rootkit和病毒扫描程序发现的版本。诸如此类的恶意软件完整解决方案造成了新的威胁。

参考文献：

1. Pietrek, Matt. "Learn System-Level Win32(R) Coding Techniques by Writing an API Spy Program." Microsoft Systems Journal Volume 9 Number 12.
2. Richter, Jeffrey. "Load Your 32-bit DLL into Another Process's Address Space Using INJLIB." Microsoft Systems Journal Volume 9 Number 5.
3. Richter, Jeffrey. Programming Applications for Microsoft Windows fourth edition. Redmond: Microsoft Press, 2000. pp. 751-820.
4. Hunt, Galen C. and Doug Brubacker, "Detours: Binary Interception of Win32 Functions" Proceedings of the 3rd USENIX Windows NT Symposium, July 1999, pp. 135- 43.
5. http://www.rootkit.com
6. Hacker Defender by Holy Father. http://hxdef.czweb.org/