Ring3 NT rootkit 新思路

author：baiyuanfan

mail：baiyuanfan@163.com

August 7, 2005

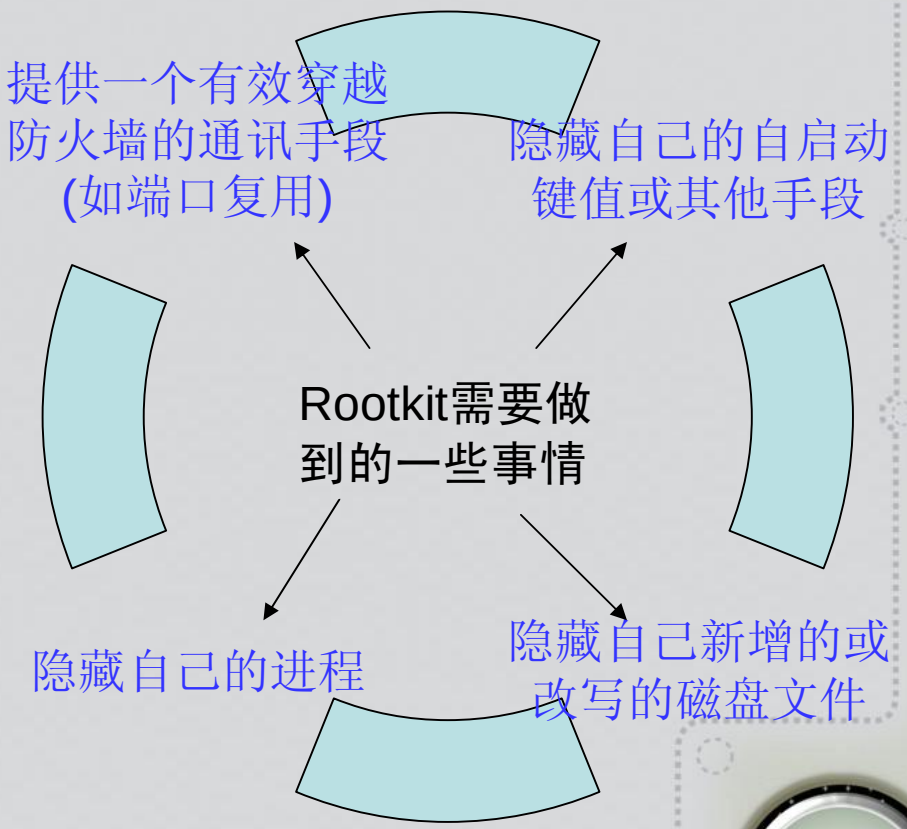
X'con 2005

内容目录：

- 一、**NT rootkit**发展的现状
- 二、**ring3 nt rootkit**新思路
- 三、挂钩异步**I/O**调用实现端口复用
- 四、在**ring3**控制**iis6**的端口
- 五、隐藏自己：自删除和复活
- 六、另外一些思路和讨论
- 附录

一、NT rootkit发展的现状

- 随着后门攻击与防御技术的发展，普通的远程管理程序式的后门早已不能够适应复杂的环境：防火墙，安全策略，**IDS**等。防火墙限制了程序随意开放端口或者反弹连接；各种监测工具列举可疑的进程文件启动项来发现和杀死后门。于是**rootkit**应运而生。它们具备自我隐藏好，能够穿越防火墙等安全设施的特性。



一、NT rootkit发展的现状

- **1. ring3**范围：
  - 代表作——黑客守卫者（**Hacker Denfender**）
  - 隐藏自己新增的磁盘文件和注册表服务键值使用**hook ntdll.dll** 中的**nativeAPI**实现
  - 存在的问题：这个方法对付普通的管理员检查很有效，然而对于来自内核的检测者直接进行的驱动级别的列举，完全无能为力，因为后者不通过被**hook**过的执行路径。
  - 端口复用通过挂接**win32 API WSARecv / ReadFile**实现
  - 存在的问题：对新的**IIS6**无效

一、NT rootkit发展的现状

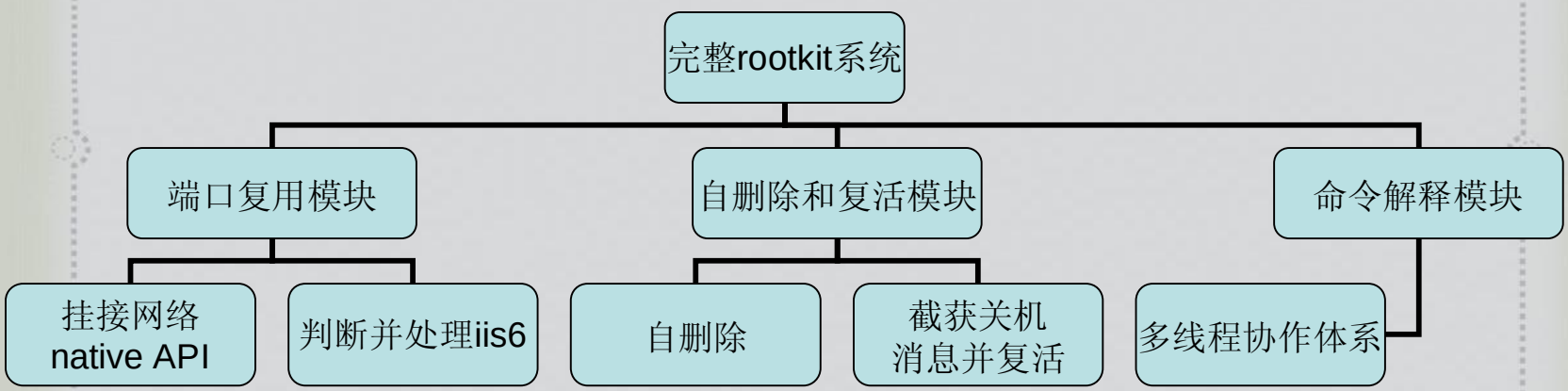
- **2.** **ring0**范围，目前很少有实用的系统公开，本文内，不讨论**ring0**的**rootkit**技术。不过，基本的是，**ring0**的**rootkit**更强大，更复杂，但是面对**HIDS**的挑战时，**ring0**的**hook**的危险级被认为比**ring3**更高。
- **3.** 因此我们发现，如何设计稳定有效的端口复用和如何对抗来自于内核态的**anti-rootkit**的检查，是目前**ring3 nt rootkit**急需解决的重要问题。

二、ring3 nt rootkit新思路

- 整体思路
- **1.** 通过挂接**native API**实现同步和异步的端口复用
- **2.** 特别的对不能直接挂接**2003**的**iis6**加以处理
- **3.** 用自删除的方法代替以往的文件启动项隐藏技术，“没有”代替“隐藏”，对抗来自内核的静态检查
- **4.** 多线程协作体系解决部分进程权限不足
- **5.** 自己完成大多数必要的控制任务，不和可能暴露我们自己的其他后门程序协作
- **6.** 印证我的这些新思路：一个测试中的**ring3 nt rootkit**：**byshell v0.67 beta2**

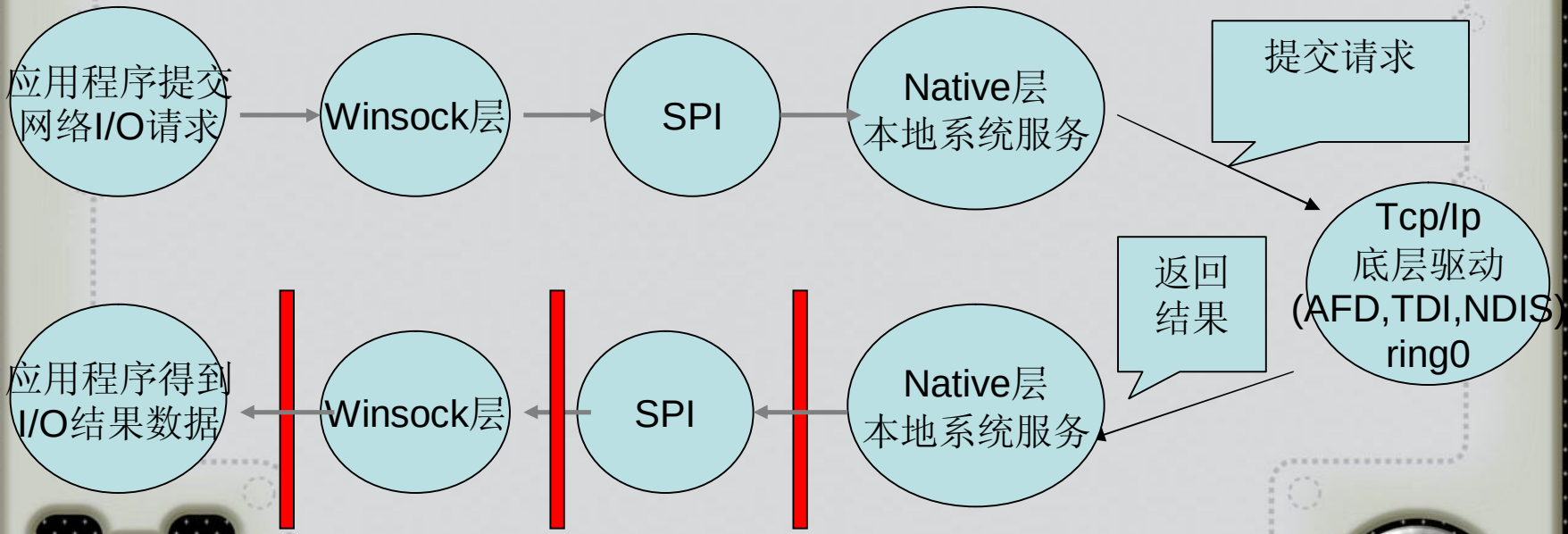
二、ring3 nt rootkit新思路

- 整体思路模式图



三、挂钩异步I/O调用实现端口复用

- 目的：在被挂接的**tcp**接收函数返回给应用程序控制以前比较接收到的数据，如果是客户端发来的激活后门的信号，就夺走这个**socket**，通知原来程序连接已经被关闭，并且在这条**socket**连接上实现后门，从而做到不影响原来端口上服务的端口复用。



这里红杠显示的地方，我们在ring3有几次拦截I/O结果返回的机会，if(!strcmp(buff,"暗号"))(dobackdoor();)

三、挂钩异步I/O调用实现端口复用

- 挂钩同步的网络**I/O**的方法很简单，在真正的**I/O**函数返回之后，自己返回控制给用户程序之前直接比较接收的数据是不是激活信号就行了。
- 而对于异步的**I/O**调用则有两个难点：
  - 在异步**IO**调用的**api**函数返回时，数据并未被更新，而是过一段时间后通过另外的通知完成机制，因此无法在返回用户程序前直接比较判断缓冲区内容
  - **winsock**库在**SDK win32API**层次上实现了非常多种类的**IO**模型，令人困惑不知道该如何挂接

三、挂钩异步I/O调用实现端口复用

- **1. nativeAPI**层次的**I/O**模型：
  - 通过挂接**nativeAPI**层次的网络**I/O**函数，可避免处理那一大堆的**win32API**层次**IO**模型。
  - 我所知所有的**windows native api**在实现**I/O**时都是给用户**3**种选择。同步**IO**，**event object**通知操作完成的异步**IO**，和**user apc routine**通知操作完成的异步**IO**。
  - 举例说明，**ReadFile**最终调用的**ZwReadFile**：

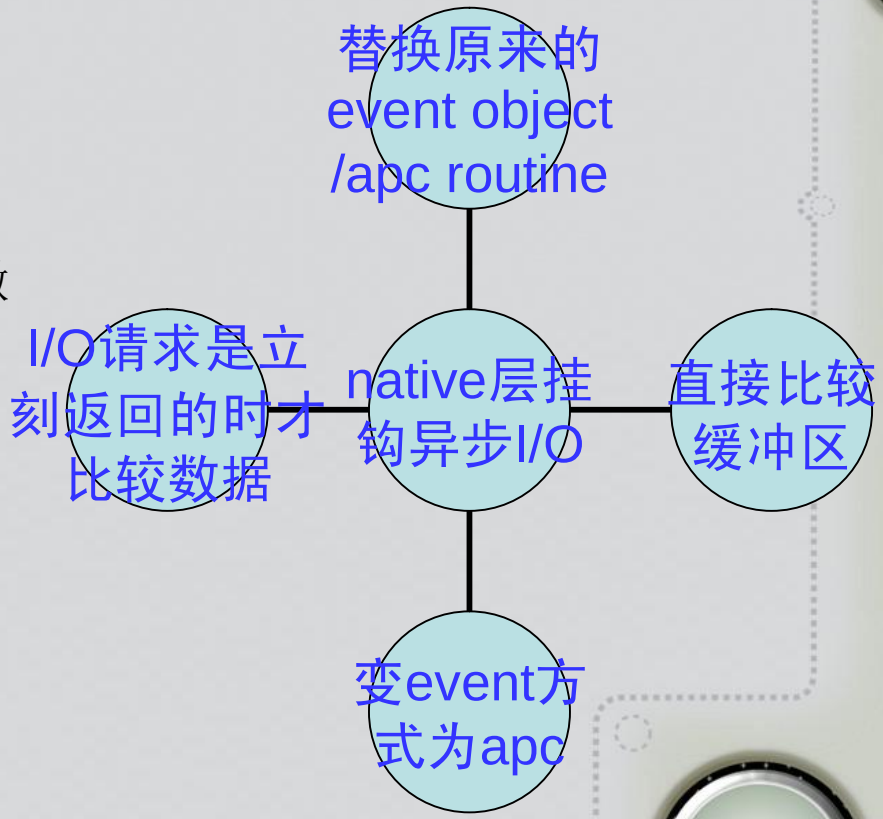
```
NTSYSAPI
NTSTATUS
NTAPI
ZwReadFile(
IN HANDLE FileHandle,
IN HANDLE Event OPTIONAL,
IN PIO_APC_ROUTINE ApcRoutine OPTIONAL,
IN PVOID ApcContext OPTIONAL,
OUT PIO_STATUS_BLOCK IoStatusBlock,
OUT PVOID Buffer,
IN ULONG Length,
IN PLARGE_INTEGER ByteOffset OPTIONAL,
IN PULONG Key OPTIONAL
);
```

三、挂钩异步I/O调用实现端口复用

- 如果设置了**apc routine**，则系统忽略**event object**，并且认为是**user apc routine**传信的异步**IO**，在操作完成时将回调该函数。如果没有**apc routine**而有**event object**，是**event object**传信的异步**IO**，在操作完成时将设置该事件为已传信状态。都不设置的话，则是同步阻塞的**IO**。
- 不论在**SDK**层次看来多么复杂的**IO**模型，比如完成端口，最后也是依赖这**3**个基本机制。因此我们只要挂接处理这三种**IO**方式即可，无需讨论各种复杂的**SDK**层**IO**模型。
- 通过挂接**native**层而不是**win32**层，上面讨论的难点二已经被解决。

三、挂钩异步I/O调用实现端口复用

- **2.** 四种**native**层挂钩异步**I/O**思路：
  - 讨论问题最大的难点一。对于一段时间之后通过另外的通知方式通知完成的异步**I/O**，应该怎样获得激活字符串呢？
  - 提供几种解决方案，讨论他们的优缺点并决定采用的方案



三、挂钩异步I/O调用实现端口复用

- **A.**替换原来的**event object/apc routine**为我们的再去调用系统**IO**函数
  - 替换**apc routine**的方法还比较简单，我们可以设计一个**routine**，把所有使用**apc**传信的**I/O**调用的**apc routine**参数全部都改成我们的**routine**，再在我们的**ApcContext**里保存原**routine**和原**ApcContext**。当**I/O**结束，我们的**apc routine**得到调用时，可以通过我们的**ApcContext**里保存的原**apc**地址和原**ApcContext**进行**apc**的转发。
  - 对于**event object**的方式，我们很难只申请一个**event**在一个线程里循环的去等待和转发它。因为在大量的并发请求到来时，没有对应于**ApcContext**的一个**EventContext**，我们在**event**激活后不能够知道，现在激活的这个**event**对应的该被转发给的是哪个原**event**。
  - 如果建立很多个新的**event object**，再搞一个源**object**和替换后的**object**的对应表呢？那么我们需要新建很多等待线程和很多内核对象，不但编程麻烦，而且效率相当的低。

三、挂钩异步I/O调用实现端口复用

- **B.**判断**ntstatus**为**0**，**I/O**请求是立刻返回的时才比较数据
  - 即使应用程序提交了异步**I/O**请求，有时协议驱动里有现成数据时，也会立刻返回。
  - 该想法需要客户端配合，迅速大量的发同一内容的包，服务端只有当异步调用直接返回了**success**时，才检查缓冲区，此时的缓冲区里是正确的数据。
  - 这个想法只有在包到达网络驱动的速度比服务程序投递下一个**recv**请求还快时，才有可能成功。
  - 遗憾的是，实验证明这种方法对付**iis**时，成功率很低

三、挂钩异步I/O调用实现端口复用

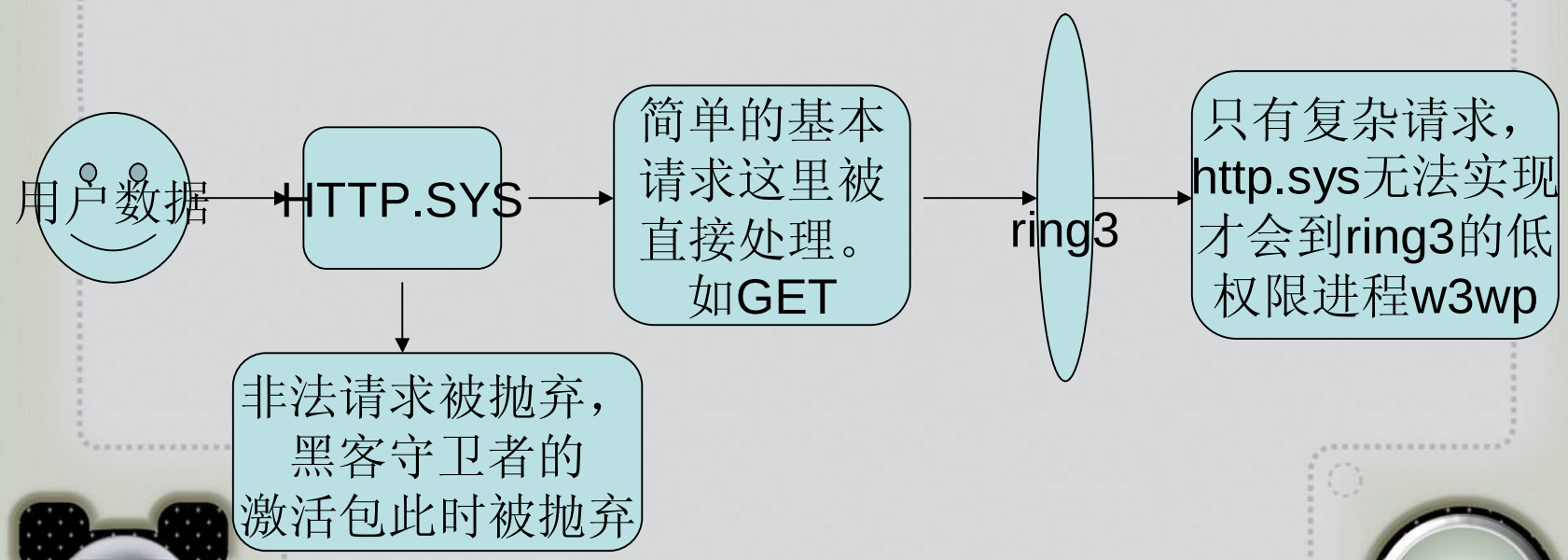
- **C.**直接比较缓冲区
  - 不管**ntstatus**返回是**success**还是**pending**，直接搜索缓冲区或者缓冲区很附近的内存中有没有激活串。
  - 这个方法不但会有误判，还有遗漏的可能。
  - 但是我们可以通过各种手段把它控制在可以忍受的范围内。
  - 不需要暴力搜索缓冲区附近的内存，对于很多系统服务进程，可以有一个经验主义的方法来更快的定位可能的地址，对于**iis**很有效。
  - 针对这种直接比较内存的方法，可以设计一种应用程序的编程模式来绕过它。但是目前已有的服务器程序设计里很少使用，所以**rootkit**端口复用的企图能够成功。

三、挂钩异步I/O调用实现端口复用

- **D.变event方式为apc方式的变种方法A**
  - 原来的方法**A**是因为**event**方式时转发的效率问题无法实现，我们可以像**apc**方式一样的处理，把原**event object**存储进**APCContext**里，将原来是**0**的**apc routine**参数添入我们的**apc**函数地址，执行函数。
  - 我们先截到完成**I/O**的信息，在**apc**中比较字符串是否约定激活信息后，如果不是通过**SetEvent (APCContext->eventobject)**来进行完成信号的转发。这样就克服了直接用**event**转发的繁琐和对性能的影响。
  - 如此看来，这才应该是最完美的方法，不存在任何误差并且效率也比较高，以前说过的可以绕过方法**C**的编程模式也不可能再绕过这种方式了。它不只是能用于**rootkit**的端口复用中，甚至可以用在对准确性要求很高的场合如**sniffer**等。
  - 这篇论文进行最后修订的时候我忽然想到这个方法，由于时间仓促，这种新的方法并未经过足够的测试，对于一个对准确性要求不高的场合，**rootkit**来说，一个简单的**C**方法已经够用了。

四、在ring3控制iis6的端口

- 目前使用**windows**的服务器中**2003**和**iis6**很普及，黑客守卫者等后门无能为力，**iis6**负责控制连接的**http.sys**从**ring3**无法挂接。
- 那么我们需要一个机制，能让**http.sys**把数据传输到**ring3**去，在这个过程中我们截获并比较是不是激活串，如果是就处理之。



四、在ring3控制iis6的端口

- **1**．基础知识：**iis6**安全机制
  - **iis6**负责控制连接和接受请求的是一个设备驱动程序**http.sys**，该驱动程序首先会解释用户的请求，如果它认为请求不合法或者请求很简单，就会自己处理完毕。
  - 如果是复杂的请求如**asp.net**之类，它就会把请求的内容传递给**ring3**的**w3wp.exe**（该进程为一低权限进程）来处理，处理结果返回给**http.sys**并发送回客户。
  - **http.sys**控制连接导致我们无法抢夺掉**socket**（事实上根本就不存在用户模式的**socket**而只有内核模式的**TDI connection endpoint object**），我们只能使用一次性的连接。
  - 即使我们可以截获激活串并执行命令，要想把命令执行结果传递给用户或者实现交互的连接还是需要另想办法。

四、在ring3控制iis6的端口

- **2**．解决方案：
  - 采用**soap**协议来穿透驱动层，把报文传递到**ring3**的**w3wp.exe**，我们就可以通过暴力搜索**w3wp.exe**每次**ZwDeviceIoControlFile**函数调用时的栈内存的方法来取得激活命令。
  - 在**soap**协议目的文件名中夹带字符是不会被截断的。我们可以把客户端发送的二进制数据编码成字符，夹带到**soap**文件名里，后门在**w3wp.exe**里搜索栈内存，解码得到命令。
  - 可以构造如下**soap**文件名：
  - **"/abc/12345678baiyuanfangff"//**符合**http**和**xml**语法的激活暗号串
  - **"1324"//**指令重复性分辨码
  - **"ABCABCABCABC"//**编码后的命令
  - **"".asmx"//**符合语法的**soap**后缀
  - **soap**文件名的长度有限制，大约是**256**个字节。我的算法把二进制数据编码成字符是**1:2**，所以我们大约只能传递**100**个字节长度的命令进去。作为命令是够了，我觉得用来传输文件不太实际。

四、在ring3控制iis6的端口

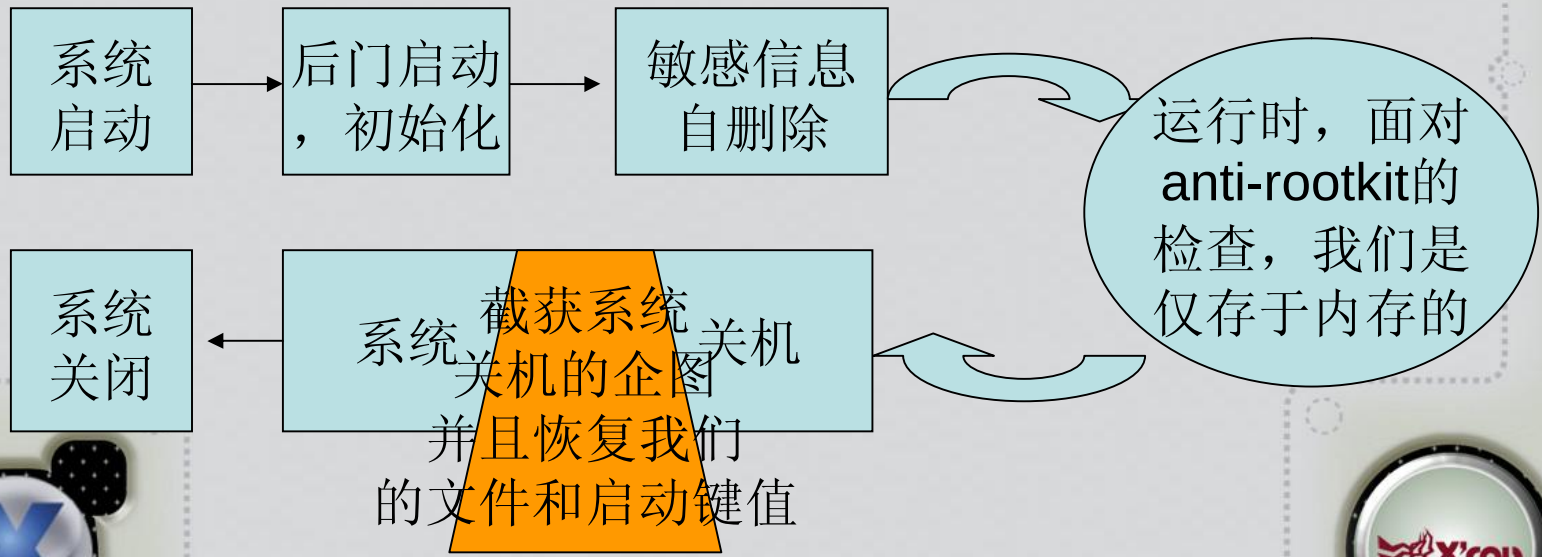
- **3.**结果回显的方法：
  - 将结果编码为字符后写到**web**目录的一个特定临时文件里，客户端用一个**GET**请求来获得。然后可以继续发命令，执行，回显……，相当于实现了变相的交互连接。
  - 要想将结果编码为字符后写到**web**目录，前提是得知道**web**路径。**iis5**的主**web**路径存放在：**HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services\\W3SVC\\Parameters\\Virtual Roots\\ /** ，然而**iis6**不可以这样获得了。
  - **Iis6**我们可以解析**metabase.xml**配置文件取得**iis6**的每个虚拟站点的**web**路径并选择优先站点

四、在ring3控制iis6的端口

- 在**iis6** 的配置文件**metabase.xml**中有如下的信息：
  - **<IIsWebVirtualDir Location ="/LM/W3SVC/1/ROOT" AccessFlags="AccessRead | AccessScript" AppFriendlyName="**默认应用程序**" AppIsolated="2" AppPoolId="DefaultAppPool" AppRoot="/LM/W3SVC/1/ROOT" Path="g:\www" />**
  - 其中的**"/LM/W3SVC/1/ROOT"**里的**1**可能为任何值，代表不同的虚拟站点，我们只需要搜索最小的一个，就是其默认站点。

五、隐藏自己：自删除和复活

- “没有”代替“隐藏”
- 以线程和挂钩的形式存在于内存中，文件启动项自删除，不在磁盘和启动项中留下可疑的东西，使得来自用户和内核态的检测者不能发现异常。等到合适的机会，再把自己恢复，通常是关机的时候。
- 如果**anti-rootkit**比我们更早启动并且开始动态监控，这种方法将失败。



五、隐藏自己：自删除和复活

- 由**nongmin**在**cmdbind2**中提到过的类似的想法
  - 他的想法是配合一个远程**dll**注射单独进程的普通**bindshell**后门，在远程地址空间中使用代码将**dll**内存映像拷出，卸载后在申请相同基址的内存将**dll**内存映像拷回，实现了**dll**可以继续运行，又不存在于进程的模块列表。
  - 虽然他的想法很好，但是我们将看到，在很多地方他的实现方法不适用于一个**rootkit**。这没关系，下面我们列出不适合的地方并逐一进行改造。

五、隐藏自己：自删除和复活

- **rootkit**要感染所有进程，并且要最快的处理新建的进程。要在进程建立的第一时间感染它，我们需要挂钩进程创建的函数。
  - 挂接**ZwCreateprocess(Ex)**是无效的，因为那时只是建立了一个进程对象，相关工作还没有做完，连**ReadProcessMemory**一类的函数都还不能正常工作。因此我们要挂接**ZwResumeThread**。
  - 读取特定地方的内存判断新线程所处的进程是否已经被挂接，如果否则进行感染。

五、隐藏自己：自删除和复活

- **dll**文件已经被删除，我们无法安安稳稳的使用**LoadLibrary**函数来加载**dll**到目标进程。
  - 我们可以手工重定位全部的代码，比较复杂。也可以不要**dll**，像黑客守卫者一样，全部代码几乎都是汇编写的可自身重定位代码。但是那样的话，想实现一个复杂的功能模块几乎不可能。
  - 在不要手工重定位全部的代码的情况下，我们也可以做的就是把整个**dll**内存映像原封不动的拷贝过去。这要求目标进程里，该虚拟地址没有被占用，否则会失败。这个我们可以通过在编译**dll**时指定一个生辟的基地址来最大限度的防范。
  - 由于**dll**内存映像拷贝过去以后，那些用到的**kernel32**函数的地址都没有重新校订过，如果新进程的**kernel32**基址和内存映像来源进程的不一致，会有灾难性的后果。
  - 不过，**windows**系统中，**kernel32**基址和其他进程不一致的情况一般只会出现在少数系统支持进程或者子系统进程中，他们并不参与后面的**ZwCreateProcess(Ex)**的工作。

五、隐藏自己：自删除和复活

- 截获关机事件的方法：
  - 在一个独立服务进程中使用**SetConsoleCtrlHandler()**函数注册一个**CTRL_SHUTDOWN_EVENT**处理例程，该例程可以截获普通的关机动作。
  - 找一个合适的、各个**windows**版本通用的且必需的独立服务进程不容易。**spoolsv.exe**是比较版本通用的一个，但是它也不好，不少不使用打印服务的系统会禁止它。
  - 也不能选择一个类似**winlogon.exe**的系统支持进程或者子系统进程，由于微软实现这些进程的特殊性，在它们里的线程根本无法接受到上述的两个消息。这里的原因，我也不清楚，欢迎大家和我进行讨论。

五、隐藏自己：自删除和复活

- 截获关机事件的方法：
  - **hook**关机时会必然调用的一个函数。备选的有**kernel32!ExitWindowsEx**和**ntdll!ZwShutdownSystem**。
  - 由于部分病毒和黑客性质程序，会直接调用后者来使系统重启，显然挂接后者更可靠，更值得信赖。
  - 不幸的是，在后者被调用时，似乎整个子系统已经被部分卸载，写入服务和文件的努力失败了。所以，我们只有去挂接**ExitWindowsEx**，在关机时，生成服务和文件。

六、另外一些思路和讨论

- 以下的是我的一些不成熟的想法，提出来供大家参考和讨论。
- **1．Anti-ring3rootkit**方法一：监视远线程创建
- **2．Anti-ring3rootkit**方法二：挂接**ZwWriteVirtualMemory**
- **3．Anti-ring3rootkit**方法三：通过**hashcheck/**导入导出表**check**检测重要的函数是否被挂接，否则报警
- **4．**上面**anti-ring3rootkit**的一些思路的可行性？还有没有新思路？
- **5．**当**anti-ring3rootkit**的那些思路已经实现，**ring3 rootkit**该如何生存？是否只有走向**ring0**这一条路呢？

附录：

- **byshell v0.67 beta2**
  - 我实现了一个测试中的**ring3 nt rootkit：byshell v0.67 beta2**，来印证我的这些新思路。我已经在自己的机器上分别在**windows2000(sp4)，XP(sp2)，2003(sp0)**上测试成功。
  - 欢迎大家测试和共同开发，该**rootkit**开源

附录：

- 参考资料：
  - [1]Hacker Defender
  - Holy_Father(holy_father@phreaker.net)
  - http://rootkit.host.sk/
  - [2]cmdbind2
  - nongmin(nongmin.cn@yeah.net)
  - http://nongmin-cn.8u8.com
  - [3]ADE32
  - Z0mbie
  - http://z0mbie.host.sk/

附录：

- 特别感谢：
  - 感谢CVC(www.retcvc.com)和xfocus(www.xfocus.net)的所有朋友的无私帮助。尤其是真心的谢谢CVC的vxk老兄。祝所有朋友事业有成，家庭幸福。

- 作者简介：
  - 白远方，男，上海华东师范大学2004级学生。喜欢研究系统底层和内核，喜欢交朋友。
  - E-mail:baiyuanfan@163.com。

谢谢大家