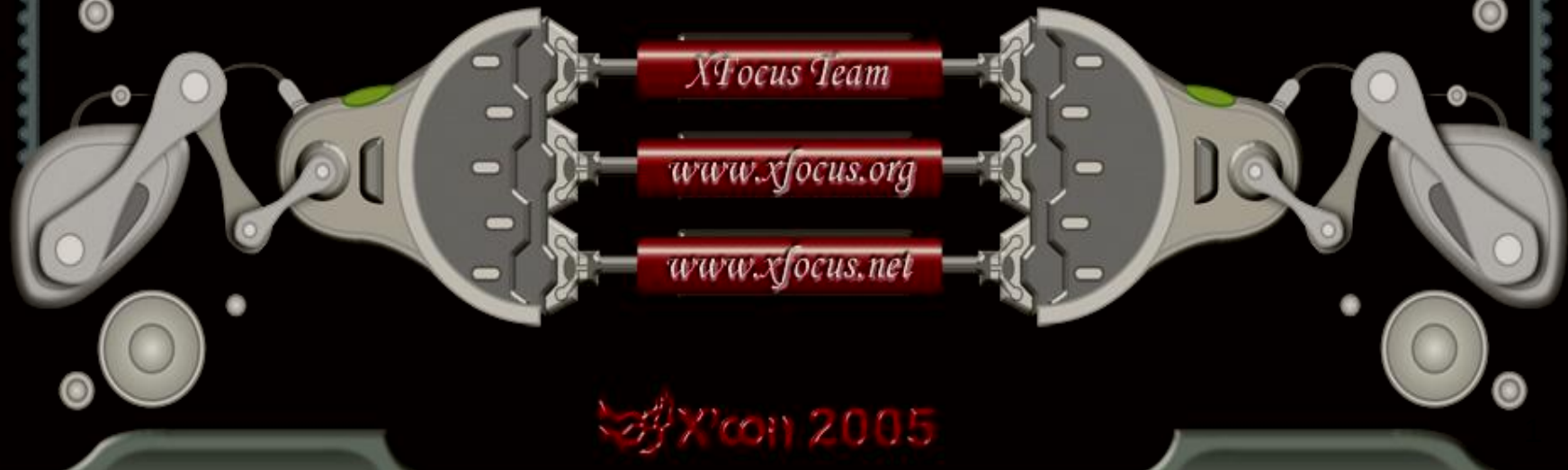
恶意代码剖析和Rootkit检测

Matt Conover
Email: shok1234 msn.com



X'con 2005

介绍

- 一场持续的猫和老鼠的游戏
- 检测和分析机制多年来变化不大
  - 指令跟踪
  - 系统**Hook**调用 或 **API**调用
- **Malware/Rootkits** 日益成熟并且善于躲避隐藏

内容简介

- 分享**rootkit**检测和躲避艺术的技术状况
- 介绍一种新的**Windows**平台下**rootkit**检测和恶意代码分析的方法
  - 适用于检测目前的所有**rootkit**
  - 一个简单的分析实例
  - 讨论特征语言（**signature language** ）在**rootkit**检测上的应用

Rootkit介绍

- **Windows平台上最早发现于1999 (NTRootkit, Hoglund):**
  - 不同于病毒
  - 非破坏性的信息收集工具
  - 通常运行在核心(更易隐藏)
- 通常被黑客手工安装
  - 确保能再次进入系统
  - 隐蔽地捕获密码，键击等

Rootkit检测

- 商业上对**rootkit**的兴趣远不及恶意程序
  - 当前大多数检测机制并不专门针对**rootkit**
  - 相对于流行的恶意或间谍程序，明显缺乏关注
- 这种状况导致自食其果
  - 商业性工具缺乏检测导致人们不知道是否被安装了**rootkit**
  - 因此用户对**rootkit**检测的需求不高

猫的情况（Rootkit检测）

- 当前的四种检测机制：
  - 已应用：反病毒软件方法
  - 研究中：**HIPS**（Host Intrusion Prevention Systems）
  - 研究中：执行路径分析（**EPA**）
  - 新方法：微分测试**(Differential testing)**

反病毒方法

- 对付已知**rootkits**方面很有效
- 当新的变种和**rootkits**发布需要产生新特征
- 在**rootkit**被运行前，检测它的指纹

反病毒方法的缺陷

- 很少**rootkit**是公开的
  - 低可靠性
- **Rootkits**深度隐藏并且非破坏性
  - 少量的**rootkit**样本被送到反病毒公司
- 太迟了...
  - **Rootkits**可以使**antivirus**的**hook**失效 **(**通常通过文件系统的过滤驱动）
  - **AV** 定义出来的太晚了 **J**

HIPS方法

- 两个层面的防御
- 防止机器被**exploits**
  - 检测缓冲区溢出、 **RLIBC**等
- 防止攻击者进入**kernel**

HIPS方式的缺点

- **Phrack62**（**Butler et al**）中描述了许多缺点；
- **API Hook**很容易被躲避
- 多数**HIPS**只检测那些用起来类似**exploit**的
- 很难涵盖所有的**rootkit**技术：
  - **Crazylord**使用符号连接**\Device\PhysicalMemory**来躲避检测
  - 对于新的核心权限提升漏洞缺乏保护

EPA方式

- **Jan Rutkowski**在**BlackHat Las Vegas 2003**进行了讨论
  - 一种老技术现在用于专门对付**rootkit**
- 用指令触发（**instruction trapping**）去分析系统调用
  - 在系统安全时进行学习
  - 记住指令数或系统调用的代码路径
- 当系统调用的执行路径变化时检测**rootkit**

EPA方式的缺点

- 跟踪所有系统调用导致系统性能大幅下降
- 很难正确实现（多种应对方法）：
  - 重写**IDT**中的**trap handler**
  - 重写**TSS**中的**EFLAGS.TF**
  - 通过**POPF**重写**EFLAGS.TF**

微分(Differential)Rootkit 检测

- 同顶位置查询同样的信息:
  - 首先使用**user-mode APIs**
  - 然后使用**low-level** 方法 **(**察看**registry** 文件**, NTFS** 目录**, etc.)**
- 假如他们有不同，一些东西可能是被藏了起来。

微分(Differential)Rootkit 检测的缺点

- 容易被击败**(see rootkit.com)**
- 容易成为**rootkits**的目标
- 那些方法太基本了
- **Rootkits** 可以使用一些特别的情况来对付那些工具

Rootkit 技术简介

- **User-mode rootkits**
  - 隐藏在其他进程中
  - 键盘**sniffing**
  - 可能是 **"diskless" (AV** 不能检测出来**)**
  - **Metasploit, CANVAS** 和 **CORE IMPACT** 都是 **diskless** 的
  - 将不在这里讨论
- **Kernel-mode rootkits**
  - 转下面...

Rootkit 技术简介

- 第一，进入 **kernel-mode**
- 第二，**hook kernel**
- 第三，试着永久驻留

Rootkit 技术
进入Kernel #1

- 使用 **ZwSetSystemInformation** 或者 **ZwLoadDriver**
  - **Enable SeLoadDriverPrivilege**
  - 问题是它将是可分页的 **(**就象 **Hoglund/Butler** 注释**)**
  - 但是这里有个小诡计**: MmResetDriverPaging J**
- **Service Control Manager (**一般的方法**)**
  - 不需要小诡计
  - 需要创建一个 **registry key**
- 同时需要一个物理文件存在
- 使**antivirus** 检测变得容易

Rootkit 技术
进入Kernel #2

- 使用**kernel-mode exploit..** 例如:
  - **LPC (local):** 原创 **(eyas)**
  - **Norton Antivirus (local): s.k. chong**
  - **SymDNS (remote): barnaby jack**

Rootkit 技术
进入Kernel #3

- 从用户模式安装 **Ring3->Ring0** 调用门
  - 看**crazylord**的文章
  - 无磁盘访问 **(AV** 不能检测**)**
  - 比 **kernel-mode exploits**容易
  - 直接从用户模式修改 **x86 GDT**
  - 可能在新版本**windows**不能正常工作

Rootkit 技术
Hooking into the Kernel

- 一旦我们的代码运行在**kernel**，那么接下来可以做些啥？
- 挂接系统调用表
  - 加一个新的系统调用，隐藏例如文件和注册表的信息
- 挂接中断处理
- 操纵页表入口 **(**可执行的，不可读的等**)**
- 挂接驱动派遣表
- 加过滤驱动

更好的方法...
介绍

- 能用来检测 **rootkits**?
- 能用来监视系统行为吗? **(helpful to profile malware)**

Windows实行对象

- **Windows**采用"实行的对象（**executive objects**）"机制
  - 由对象管理器控制
  - 句柄（**handles**）都间接同对象相关
  - 所有东西都是对象

Windows实行的对象

- Memory sections
- LPC ports
- I/O completion
- WMI
- Desktops
- Mutexes
- Events
- Semaphores
- I/O Controllers
- Files
- Registry keys
- Devices
- Drivers
- Processes
- Threads
- Jobs
- Sockets
- Security tokens

这些都是对象！

Windows实行的对象

- 对象管理器如何跟踪这么多种类的对象？
- 它不用记住所有对象的类型
- 对象类型是动态注册的
- 每个对象类型都有一套操作：**open**、**create**、**secure**、**close**

Windows实行的对象——例子

- 系统初始化中，**IoInitSytsem()** 注册 **FILE_OBJECT** 类型
- 当调用 **NtCreateFile()** 来创建新文件时:
- 它调用**ObCreateObject(name, FILE_OBJECT)**
- 对象管理器通过创建例程注册 **FILE_OBJECT**来调用**Open callback**
- 如果**Open callback**返回成功，那么句柄就返回给**NtCreateFile**

更好的方法

- 我们可以对所有我们感兴趣的对象类型替换**callback**
- 如果我们对文件或进程打开操作感兴趣：
  - 找到**FILE_OBJECT**对象类型，并且替换**Open callback**
  - 找到**EPROCESS**对象类型，并且替换**Open callback**

更好的方法

- 在替换的**callback**中，我们分析事件，然后再调用原始的**callback**
- 如果我们只是剖析：
  - 可以记录事件，并且允许通过
- 如果我们执行**rootkit**检测：
  - 检查是否有任何匹配的特征
  - 如果特征匹配，就执行特征行为（比如，报告、拦截等）

更好方法——优点

- 节约大量性能时间
  - 可以单独处理特定的对象类型，特定进程，或者只是核心
  - 如果打开一个不存在的对象，不会调用到**open callback**（因此不存在过载**overhead**问题）
  - 如果调用者创建没有适当权限的对象不会调用到**open callback**（不存在过载）
- 新可能性：
  - 几乎能监控所有系统行为
  - 记住，几乎所有东西都是对象

更好方法——How To

- 如果我们剖析恶意软件：
  - 在挂起状态启动恶意软件
  - 监控所有对象类型
  - 只应用于恶意软件进程
- 如果我们检测**rootkit**：
  - 进程特征和只监控匹配特征的对象类型
  - 只应用于核心态（比如忽略用户态进程）

Profiling：Demo

检测驱动装载

- 在**kernel**创建一个带**SEC_IMAGE** 标记的**section**

检测过滤

- **Detect creation of I/O completion port from kernel**
- 列举 **DEVICE_OBJECTs**

检测 Dispatch Hooks

- 列出 **DRIVER_OBJECTs**

检测隐藏的Rootkits

- 一些情况下**rootkits**是好检测的 **J**

返回地址到不可读页

返回地址到**non-paged** 内存池

除去一个Rootkit...

- 仍然是实验性的...
- 除去一个**rootkit**可能会使系统变得不稳定**(**未知的**hooks)**
- 把所有的 **rootkit**代码都替换为 **INT 3 (breakpoint)**
- 加一个 **INT3** 处理
- 假如返回到一个可疑的地方，在那地方也加个**INT 3**
- 最后，我们把**INT3** 该为 **NOP**

自卫能力

- 这是猫和老鼠的游戏，所以老鼠总是想法逃过这种方法
- 当老鼠开始针对这种检测机制的时候，我们需要保护自己
- 因此，我们需要采用一种自卫能力的机制以使我们能掌握控制权

自卫能力：第一步

- 首先阻止**rootkit**从这些地方加载：
  - 禁止访问**\Device\PhysicalMemory**
  - 禁止驱动加载系统调用
- 局限：
  - 攻击者可以通过新的核心权限泄露漏洞绕过
  - 用某种机敏的方式通过符号连接（**symbolic link**）访问**\Device\PhysicalMemory**

自卫能力：第二步

自卫能力：第三步

- 确信没有其他的驱动在之前加载，除了 **FAT/NTFS**
  - 在"**Boot Bus Extender**"最开始安装我们自己
  - 阻止任何改变 **HKLM\SYTSEM\CurrentControlSet\Group OrderList**

自卫能力：第四步

- 确信没有人能

总结

- 以上描述的是一种观察系统行为的方法
  - 用户模式和核心态
- 这种方法可以阻截某些行为
  - 特征语言可以用来检测已知的**rootkits**
- 应该具备自保护能力
  - 对于还没有验证的新**rootkits**是必须的
- 最后，这只是猫和老鼠游戏的一步而已

Acknowledgements

- **Many kung fu masters for Windows kernel-mode exploitation and rootkits:**

  Joanna Rutkowska, Jamie Butler, flashsky, S.K. Chong,

  Barnaby Jack, Greg Hoglund, Derek Soder, crazylord

STKIT– Shok Toolkit **J**

- **Remember this URL…**
- **Remember this URL…**
- **Remember this URL…**
- **Remember this URL…**

http://www.cybertech.net/~sh0ksh0k

Will not be publicly announced, so you must remember

Code will be put there in the next 2 weeks

The End

- **Thanks for listening J**
- **Send email to shok1234 msn.com**