型设置正确，但硬盘指示灯不亮，屏幕仍显示 error disk write，操作失败。

二、解决步骤

1.用 DOS5.0 系统盘启动后，运行 A 盘上 DEBUG；

2.用 DEBUG 汇编命令，将硬盘 0 头 0 面 0 道格式化；

```
-a100
XXXX:0100 MOV AH,05
XXXX:0102 MOV BX,800
XXXX:0105 MOV CX,0000
XXXX:0108 MOV DX,0080
XXXX:010B INT13
XXXX:010D INT3
XXXX:010E
-e800
```

将 800H 至 821H 的连续 34 个内存单元的内容改为：

```
00 01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00 09 00 0A
00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00 11
-g=100
```

3.用 DEBUG 汇编命令，将主引导区内容读出；

```
-a100
XXXX:0100 MOV AX,0201
XXXX:0103 MOV BX,0200
XXXX:0106 MOV CX,0001
XXXX:0109 MOV DX,0080
XXXX:010C INT13
XXXX:010E INT3
XXXX:010F
-g=100
-d200
```

屏幕显示主引导扇区为全 0 数据；

4.将好的主引导区写入该机，该机正确的主引导扇区在 DON1 盘的第 10 扇，将 DON1 盘插入 A 驱，

```
-l200 0 10 1          (装入正确的主引导区)
-a100
XXXX:0100 MOV AX,0301  (将读操作改为写操作)
-g=100
```

此时硬盘灯亮，操作成功。

5.开机，硬盘启动成功。且硬盘数据完好无损。

三、分析与结论

该病毒属主引导区病毒，是一种恶性病毒。开机无论是否用 A 盘启动，只要系统检测到 C 盘，病毒就被激活，即内存就已带毒。并且该病毒具有自我保护的能力，拒绝用户改写主引导区。当用户遇到这种不能用主引导区覆盖法消除的主引导区型病毒，并且所有的杀毒软件均无能为力时，为了不破坏硬盘中的大量参数，可以用 0 磁道格式化法，只格式化硬盘 0 头 0 面 0 道。对于一般容量的硬盘（如该机硬盘为 40M），0 道的 17 个扇区为系统的隐藏扇区，0 道格式化不破坏存放在 0 磁道以外的数据，即不破坏 DOS 分区内及其它分区内的所有数据，并可将存在于隐藏扇区内的病毒程序一并清除。当 0 道格式化成功后，再将保存的该机的主引导区备份写回即可，或读另一相同机型相同系统的主引导区写入该机。我们用该方法成功地恢复了这两台机器。

当我们将事先已保存下来的病毒再反复感染该机，经不断分析研究，发现该病毒有个特点，病毒拒绝用户覆盖主引导区时，若用户强行用 DM 手动方式操作，硬盘格式化虽然正常，重建分区也正确，但重新开机启动时，C 盘与 A 盘全部丢失，系统瘫痪。很明显这是硬盘格式化时，病毒再次发作，改写 CMOS，丢失所有硬盘和软盘所致。此时只能将 CMOS 放电，使系统开机时 CMOS 检测失败，从而进入系统设置，恢复 C 盘、A 盘、B 盘。关于如何将 CMOS 放电的方法，许多杂志及报刊上都有论述，在此不再介绍。

另外，当用户用 A 盘启动并执行 A 盘上的 DEBUG 文件时，注意 A 盘应贴上写保护缺口，否则在内存带毒的时候，很容易感染 A 盘。当 DEBUG 程序带毒，则格式化 0 道后仍不能覆盖主引导扇区。

dBASEⅢ数据库的简单加密

李红胜 （随州市第一人民医院）

现在被广泛使用的数据库管理系统 FoxBASE、FoxPRO 以及 Clipper，为了程序的保密，在 FoxBASE 中提供了伪编译功能；FoxPRO 和 Clipper 则完全具备了真正的编译功能。这样无疑是保护了程序设计人员的劳动，也保证了所设计软件的健康运行。但是为保证软件的兼容性和数据的继承性，这些数据库管理系统全都较好地保留了 dBASE Ⅲ 的数据库结构，而 dBASEⅢ 的数据库是可以在 dBASEⅢ、FoxBASE 或 FoxPRO 中直接进行操作的。也就是说，我们使用以上数据库管理系

统来设计信息管理软件，虽然源程序得到了较好的保护，但是所涉及到的各种数据却完全暴露在别人面前，这对于程序设计人员来讲是不幸的。

那么怎样采取一些办法来加强数据库的保密性呢？本文就笔者的实际工作经验，向大家介绍一种简单的加密保护方法。

首先来了解一下 dBASEⅢ数据库的存储结构。

数据库文件(DBF)由库结构描述段和记录两个部分组成，而库结构描述段又由系统说明段(32 个字节)和字段说明段(每段 32 个字节)，存储方式见图 1)两个部分组成。其中，系统说明段的 32 个字节中，只使用了前面的 12 个字节，从第 13 个字节开始向后的 20 个字节保留没用。

如果是直接对数据库进行操作，则数据库必须严格按照图示的存储方式来存储。由此想到，只需改变一下数据库的存储方式，就可以使数据库在 dBASE Ⅲ、FoxBASE 和 FoxPRO 中成为不能操作的文件了。即，将系统说明段的前 12 个字节搬到后面 20 个保留没用的字节中存储，并且在前 12 个字节中写入与数据库无关的内容，这样就达到了数据库的简单加密保护。但是这样处理后的数据库在程序中也不能操作，所以，还必须有个数据库的解密过程。

设所附程序编译后文件名为 MDBF。EXE，运行该程序时，所附参数为 9508(可任意定)为解密；参数为其他内容为加密。另外，第 32 个字节处写入 1A(结束标记)的目的是防止用 TYPE 命令查看数据库内容。

据此，我们可以在程序的前面写入 RUN MDBF 9508 以达到解密、还原数据库系统说明段的目的，从而便于程序中对数据库的各种操作；在程序的后面写入 RUN MDBF ASD 以达到加密的数据库的目的，从而在退出程序后，数据库变成不能直接操作的文件了。

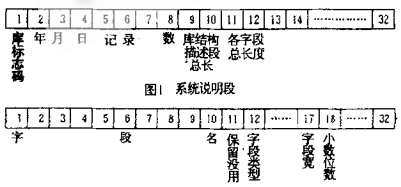

图 1　字段说明段

```
/ * dBASE 数据库简单加密处理程序 * /
/ * Turbo C 2.0 编译运行通过      * /
/ * MDBF.C                        * /
#include "stdio.h"
#include"dos.h"
#include "dir.h"
main(int argc,char * argv[])a
{struct ffblk f;
   register int done;
   char cc,str[14],ch1 = 0xAF,ch2 = 0x03,mi[8];
   int i;
   FILE * fp;
   if(argc = = 1) exit(0); / * 无参数退出 * /
   strcpy(mi,"9508");/ * 解密参数 * /
   done = findfirst(" * .dbf",&f,0);/ * 找数据库 * /
   while(!done)
   {fp = fopen(f.ff_name,"rb+");
      cc = getc(fp);
   if(strcmp(argv[1],mi) = = 0)
      { / * 还原系统说明段内容 * /
         if(cc = = ch1)
            {fseek(fp,16,0);
               fread(str,14,1,fp);
               fseek(fp,0,0);
               fwrite(str,14,1,fp);
            }
         }
   else
      {
         / * 系统说明段内容向后搬家即加密 * /
            if(cc = ch2)
               {fseek(fp,0,0);
            fread(str,14,1,fp);
               fseek(fp,16,0);
            fwrite(str,14,1,fp);
               for(i = 0;i < 14;i++)/ * 前 14K 字节写入无关内容 * /
         {fseek(fp,i,0);purc(ch1,fp);}
         }
         }
         / * 在库文件的第 32 个偏移位处写上结束标记 * /
         fseek(fp,31,0);
         putc(0x1a,fp);
         fclose(fp);
      done = findnext(&f);/ * 取下一数据库 * /
   }
}
```