第13卷第1期
1991年 3月
重庆建筑工程学院学报
Journal of Chongqing Institute of Architecture and Engineering
Vol.13.No.1
March.1991

单步中断技术在加密程序中的应用①

聂崇峡
(计算中心)

摘 要 本文分析《小鸟》游戏程序的起动程序，详细地讨论了单步中断在加密程序中的应用。单步中断技术是加密程序中的一种重要的反跟踪技术，广泛应用于加密程序中。

关键词 单步中断技术，加密程序，跟踪技术，反跟踪技术

1 前 言

为了防止软件产品被别人非法拷贝，有许多专门的加密软件产品流行于市。也有许多软件产品自身也带有一些加密的功能。

许多人致力于加密程序的研究，但对加密程序技术的探讨文献却很少。本文以游戏程序<<小鸟>>的起动程序为例，详细地分析单步中断加密技术。该程序使用的加密方法主要是单步中断，而且也是单步中断用得很好的一个程序。

单步中断技术是一种加密的基本技术，在加密程序中应用得非常普遍，许多具有反跟踪技术的加密程序 (如 PROLOK, PROTECT 等) 都使用了这项技术。

通过本文的分析，使读者能够了解并掌握这项技术，把它应用到自己的程序中。

2 单步中断简介

单步中断是硬中断，它不需要用中断命令 (INT 1H) 去启动它。当状态寄存器的 TF 位是 1 时，程序在执行完下一条指令之后，就直接产生单步中断。因此，在许多程序中，执行单步中断是不知不觉地进行的，尤其是在加密程序中更是如此。如果我们对单步中断技术不了解，就很难阅读这样的程序。

状态寄存器的结构如下：

X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	DF	X	CF

x：不考虑
OF：溢出标志
DF：方向标志
IF：中断标志
TF：陷井—单步标志
SF：符号标志
ZF：零标志
AF：辅助进位—BCD
PF：奇偶标志
CF：进位标志

①本文1990年3月19日收到

在 DEBUG 中，除 TF 位之外，其余各位都能显示出来。因此，我们不容易注意到单步中断的执行过程。

3　程序初始段

它完成把自举程序从 0000:7C00 移动到 0020:0000 处，并转移到 0020:0053 处去执行。

```
0000:7C00   CLD                    ;清方向标志
0000:7C01   MOV SP,0900            ;重新定义栈寄存器的值
0000:7C04   MOV SS,SP              ;指向 900:100
0000:7C06   MOV SP,0100
0000:7C09   MOV AX,CS              ;DS = CS
0000:7C0B   MOV DS,AX
0000:7C0D   MOV SI,7C00            ;指向正在工作的程序段 CS:7C00
0000:7C10   MOV AX,0020            ;准备移动的目标段地址 20H
0000:7C13   MOV ES,AX
0000:7C15   MOV DI,0000            ;区地址为 0
0000:7C18   MOV CX,00FA            ;移动 FAH * 2 个字节
0000:7C1B   REPZ
0000:7C1C   MOVSW                  ;完成移动
0000:7C1D   MOV DX,0020            ;设置新的数据段 DS = 20H
0000:7C20   MOV DS,DX              ;它与转移去的 CS 段相等
0000:7C22   PUSH AX                ;转移段地址 CS 的值,它为 20H
0000:7C23   MOV DX,0053            ;转移区地址 IP 的值
0000:7C26   PUSH DX
0000:7C27   RETF                   ;转移到 20:53 去继续执行
```

随后的程序地址是按照代码移动之后的地址给出的。

4　数据部分

```
0020:0028   DF 02 25 03 23 2A      ;这 11 个字节是磁盘控制参数,
0020:002E   FF 50 F6 19 04         ;其中扇区大小值是 3,即每扇区 1K 数据
0020:003B   35 1E                  ;第一次读盘从 1EH 道 35H 扇区开始
0020:003B   36 1F                  ;第二次读盘从 1FH 道 36H 扇区开始
0020:003F   37 20                  ;第三次读盘从 20H 道 37H 扇区开始
0020:0041   0C 00 00 00            ;0000:000C 是 INT 3H 的向量地址
0020:0045   12 00                  ;0012H,状态寄存器的值, TF = 0
```

```
0020:0047  00 00          ;0100H.状态寄存器的值, TF = 1
0020:0049  00 00 51 00    ;读入工作程序的存放地址 0051:0000
0020:004B  51 00 00 B8    ;把读入的数据传送到这个地址 B800:0051
                          ;B800:0000 是屏幕区,它的效果是使屏幕出现
                          ;一些乱七八糟的字符,注意这里有两个工作单
                          ;元(4B,4C)是重迭使用的.
0020:004F  06 00 51 00    ;工作程序的入口地址 0051:0006
0020:01F0  00 00 00 00    ;存放 1CH 中断向量值的工作单元
```

5 设置启动单步中断程序

下面有一条指令是往状态寄存器中或入 0370H, 它意味着把 IF,TF,ZF,AF 标志位置 1, 其中把 TF 置 1 是它的主要目的. 当这两个字节的内容被置回状态寄存器中时, 执行完下一条指令之后, 就会转移到本程序设置的单步中断程序中去执行.

第一次产生单步中断是在 20:008B 处, 它执行 20:019B 单步中断子程序.

```
0020:0053  PUSH F         ;状态寄存器进栈
0020:0054  CLI            ;关中断
0020:0055  MOV BP,SP      ;取栈指针
0020:0057  OR WORD PTR [BP+00],0370  ;把状态寄存器或上 0370H
0020:005C  XOR BP,BP      ;数据段 DS 置 0
0020:005E  MOV DS,BP
0020:0060  MOV WORD PTR [0004],019B  ;重新设置单步中断向量指针,
0020:0066  MOV AX,CS      ;使它指向 CS:019B
0020:0068  MOV BX,[0070]  ;取 1CH 中断向量的值
0020:006C  CS:            ;放入 1F0 之后的 4 个字节中
0020:006D  MOV [01F0],BX  ;0:70 之后的 4 个字节是 1CH
0020:0071  MOV BX,[0072]  ;中断的入口地址值
0020:0075  CS:
0020:0076  MOV [01F2],BX
0020:007A  MOV WORD PTR [0070],01C1  ;重新设置 1CH 中断
0020:0080  MOV [0072],AX  ;使指向 CS:01C1,这里 AX = CS
0020:0083  MOV [0006],AX  ;置单步中断的段地址
0020:0086  STI            ;开中断
0020:0087  INT 1C         ;产生 1CH 中断,它是自定义的 1CH 中断
0020:0089  POP F          ;状态寄存器退栈,注意 TF = 1
0020:008A  NOP            ;由单步中断方式而转移到 20:00A3,
0020:008B  NOP            ;而并不执行随后的几行程序
```

从 20:008C 到 20:00A1 的程序行纯粹是为了骗人而编写的. 实际上, 它们根本不会被

执行．从中也可以看出加密程序设计者的苦心．如果我们顺着程序读下去，就会感到，重新设置的 INT 8H 中断不知道是在干什么，而随后的死循环程序更使我们茫然．

```
0020:008C  XOR AX,AX                    ;把 ES 置为 0
0020:008E  MOV ES,AX
0020:0090  ES:                          ;重新设置 8H 中断向量,使指向
0020:0091  MOV WORD PTR [0020],0053     ;SP:53,由于 8H 中断是硬中断,
0020:0097  ES:                          ;我们会认为它将不断地执行
0020:0098  MOV [0022],SP                ;SP:53 处的程序段
0020:009C  MOV CX,0000                  ;置 0
0020:009F  WAIT                         ;等待
0020:00A0  NOP                          ;由于 CX 被重复置 0,
0020:00A1  LOOP 009C                    ;循环程序将变成死循环
```

6　被加密的程序段

下面的程序是被加密的程序段，当然，现在呈现在读者面前的是已经还原了的程序．没有还原之前的程序是这样的：每一条指令的第一个目标代码字节都被取非，因此，如果这时我们用 DEBUG 程序将它反汇编出来，其程序的内容是面目全非，它的代码的还原过程非常巧妙，它是利用单步中断技术逐条逐条地还原的．当 A3 处的指令执行时，它把 A8 处的指令还原、A8 处的指令执行时，它把 AE 处的指令还原、余此类推．因此，要想得到全部这些指令就比较困难．由于这段程序本身与单步中断的关系不大，因此不作详细介绍．

下面的程序中与单步中断有关的指令是：

PUSH [0045]、　POPF　以及　PUSH [0047]、　POPF

工作单元 0045 中的值为 0012H，TF 状态位是 0．

工作单元 0047 中的值为 0100H，TF 状态位是 1．

当把工作单元 0045 中的值置入状态寄存器时，将终止单步中断程序的执行，意即随后的指令没有被加密．用 DEBUG 的 U 命令还可以勉强看出几行正确的程序来．当把工作单元 0047 中的值置入状态寄存器时，将接通单步中断程序，在执行完下一条指令之后，又将产生单步中断．本程序中设置的单步中断程序的功能是负责完成下一条指令的还原工作．它这样时断时接，真真假假的方式，为程序的还原工作增加了困难．也是加密的一种手段．

如果我们对＜＜小鸟＞＞游戏的工作程序感兴趣，那么，我们可以通过分析下面的程序而得到它．因为它的有关参数都在下面的程序中给出了．得到了下面的程序，也即完成了对＜＜小鸟＞＞游戏的解密．

```
0020:00A3  CS:                                  [007A],0020
0020:00A4  LDS DX,[0047]          0020:00B4  MOV AX,0001
0020:00A8  MOV WORD PTR           0020:00B7  CS:
           [0078],0028            0020:00B8  PUSH [0045]
0020:00AE  MOV WORD PTR           0020:00BC  POPF
```

```
0020:00BD  I  )P
0020:00BE  INT 10
0020:00C0  CS
0020:00C1  PUSH [0047]
0020:00C5  POPF
0020:00C6  NOP
0020:00C7  NOP
0020:00C8  CLD
0020:00C9  MOV CX,0003
0020:00CC  PUSH CX
0020:00CD  CS:
0020:00CE  PUSH [0045]
0020:00D2  POP F
0020:00D3  NOP
0020:00D4  INT 1C
0020:00D6  CS:
0020:00D7  PUSH [0047]
0020:00DB  POP F
0020:00DC  NOP
0020:00DD  NOP
0020:00DE  MOV BX,0003
0020:00E1  SUB BX,CX
0020:00E3  SHL BX,1
0020:00E5  ADD BX,+3B
0020:00E8  CS:
0020:00E9  MOV CX,[BX]
0020:00EB  MOV DL,CL
0020:00ED  ADD DL,03
0020:00F0  MOV BX,002C
0020:00F3  CS:
0020:00F4  MOV [BX],DL
0020:00F6  XOR DX,DX
0020:00F8  CS:
0020:00F9  LES BX,[0041]
0020:00FD  ES:
0020:00FE  MOV WORD PTR
           [BX],0000
0020:0102  ES:
0020:0103  MOV WORD PTR
           [BX+02],0000
0020:0108  CS:
0020:0109  LES BX,[0049]
0020:010D  XOR BX,BX
0020:010F  MOV AX,0204
0020:0112  CS:
0020:0113  PUSH [0045]
0020:0117  POPF
0020:0118  NOP
0020:0119  INT 13
0020:011B  CS:
0020:011C  PUSH [0047]
0020:0120  POPF
0020:0121  NOP
0020:0122  NOP
0020:0123  JB 010F
0020:0125  CS:
0020:0126  LES CX,[004B]
0020:012A  XOR DI,DI
0020:012C  CS:
0020:012D  LDS CX,[0049]
0020:0131  CS:
0020:0132  ADD WORD PTR
           [004B],0100
0020:0138  XOR SI,SI
0020:013A  MOV CX,03E8
0020:013D  REPZ
0020:013E  MOVSW
0020:013F  POP CX
0020:0140  LOOP 00CC
0020:0142  CS:
0020:0143  MOV BYTE PTR
           [002C],23
0020:0148  CS:
0020:0149  LES BX,[0047]
0020:014D  CLI
0020:014E  CS:
0020:014F  MOV BX,[01F0]
0020:0153  ES:
```

```
0020:0154  MOV [0070],BX          0020:0165  POPF
0020:0158  CS:                    0020:0166  NOP
0020:0159  MOV BX,[01F2]          0020:0167  MOV SP,0030
0020:015D  ES:                    0020:016A  MOV SS,SP
0020:015E  MOV [0072],BX          0020:016C  MOV SP,0100
0020:0162  XOR AX,AX              0020:016F  CS:
0020:0164  PUSH AX                0020:0170  JMP FAR [004F]
```

7　单步中断执行程序（1）

单步中断执行程序有两段,这一段将被多次使用．其中最关键的指令是取非,这意味着把加密的程序还原．同时,它要求产生单步中断的程序必须工作在 20H 段,且地址是递增的.

```
0020:0174  CLI                  ;关中断
0020:0175  PUSH BP              ;指针进栈
0020:0176  MOV BP,SP            ;取栈指针值,以便寻找返回地址
0020:0178  PUSH AX              ;寄存器进栈
0020:0179  PUSH BX
0020:017A  MOV BX,[BP+02]       ;返回时的 IP 值
0020:017D  CMP WORD PTR [BP+04],+20  ;它是工作在 20H 段？CS＝20H？
0020:0181  JNZ 7D96                  ;不是,则转．即不予代码还原,继续
                                     ;执行下去程序就会锁死．如果用
0020:0183  CS:                       ;DEBUG 跟踪,就会出现这种情况.
0020:0184  CMP BX,[01EC] ;地址比最后一次还原的地址小？小于则不再
0020:0188  JBE 7D96      ;还原,这是为了保证向上转移的循环语句不被
0020:018A  CS:           ;重复还原．否则,程序将会出错.
0020:018B  MOV [01EC],BX      ;保存最后一次还原代码的地址
0020:018F  PUSH DS            ;寄存器进栈
0020:0190  MOV DS,[BP+04]     ;把返回处的 CS 值送到 DS 中
0020:0193  NOT BYTE PTR [BX]  ;把返回处立即要执行的一条指令代码还原
0020:0195  POP DS             ;寄存器退栈
0020:0196  POP BX
0020:0197  POP AX
0020:0198  POP BP
0020:0199  STI                ;开中断
0020:019A  IRET               ;从单步中断返回
```

8　单步中断执行程序（2）

这段单步中断程序仅执行一次，即第一次．主程序是利用这段单步中断程序来达到初始化另一段单步中断程序的目的，这也是比较巧妙的．它真正的工作程序实际上是前面的单步中断工作程序（1）．

这段程序有两项工作．一是重置返回地址中的 IP 值，使它跳过一段程序，而跳过的程序恰恰是一段死循环程序，同时，还把 A3 处的指令还原．二是把单步中断程序的入口改为指向 20:0174．

```
0020:019B  CLI              ;关中断
0020:019C  MOV BP,SP        ;取栈指针
0020:019E  PUSH BX          ;寄存器进栈
0020:019F  PUSH DS
0020:01A0  MOV WORD PTR [BP+00],00A3   ;改变返回地址中的 IP 值,使它
0020:01A5  XOR BX,BX                   ;不返回断点处,而是返回到 CS:A3
0020:01A7  MOV DS,BX                   ;DS 清 0
0020:01A9  MOV WORD PTR [0004],0174    ;重置单步中断程序的入口地址
0020:01AF  MOV BX,CS                   ;置段地址,新入口为 20:0174
0020:01B1  MOV [0006],BX
0020:01B5  MOV BX,[BP+00]      ;取返回处的 IP 值,与前面相比, 这里
0020:01B8  CS:                 ;没有+2,是因为这里没有 PUSH BP 指令
0020:01B9  NOT BYTE PTR [BX]   ;完成代码的还原工作
0020:01BB  POP DS              ;寄存器退栈
0020:01BC  POP BX
0020:01BD  STI                 ;开中断
0020:01BE  INT 1C              ;产生 1CH 中断
0020:01C0  IRET                ;从单步中断返回
```

9 自定义的 1CH 中断

这段程序起反跟踪的作用。它破坏断点中断的入口地址．如果有人正在使用调试 DEBUG 对该程序进行跟踪分析，那么，每调用一次 INT 1CH 中断，就会使程序锁死．另外，它还对 INT 13H 作了反跟踪处理，如果有人重新定义了 INT 13H 中断向量地址，那么，它所指向的就是一段用户程序而不是 ROM 区，一定会被它的代码填入程序所破坏而不能正确执行．以此来达到对 INT 13H 中断作反跟踪的目的．

```
0020:01C1  CLI                 ;关中断
0020:01C2  PUSH AX             ;寄存器进栈
0020:01C3  PUSH CX
0020:01C4  PUSH DI
0020:01C5  PUSH ES
0020:01C6  CS:
0020:01C7  LES DI,[0041]       ;取断点中断（INT 3H）地址
```

```
0020:01CB  ES:                          ;把 INT 3H 中断向量地址
0020:01CC  MOV WORD PTR [DI],0000       ;置 0, 以达到反跟踪的目的
0020:01D0  ES:                          ;把段地址也置为 0
0020:01D1  MOV WORD PTR [DI+02],0000
0020:01D6  XOR DI,DI
0020:01D8  MOV ES,DI                    ;ES = 0
0020:01DA  ES:
0020:01DB  LES DI,[004C]                ;取 INT 13H 中断向量的地址
0020:01DF  MOV CX,000A
0020:01E2  MOV AL,F4                    ;填入代码的值为 F4H, 任意代码都行
0020:01E4  REPZ                         ;破坏 INT 13H 中断向量所指的地址中的 10 个
0020:01E5  STOSB                        ;字节的内容. 未改动的 INT 13H 指向 ROM 区
                                        ;这几行程序不起作用.
0020:01E6  POP ES                       ;寄存器退栈
0020:01E7  POP DI
0020:01E8  POP CX
0020:01E9  POP AX
0020:01EA  STI                          ;开中断
0020:01EB  IRET                         ;从中断返回
```

(编辑：姚国安)

THE APPLICATION OF THE SINGLE-STEP INTERRUPT TECHNOLOGY TO A PROGRAM ENCRYPTION

Nie Chongxia

(The Center of Calculation)

ABSTRACT　This paper analyses BOOT program of the "BIRD" game program and discrsses the application of the SINGLE-STEP interrupt technology to a program encrypted in detail. SINGLE-STEP interrupt technology is an important against trace tech· ·ogy.And it is applied widely to a program encryption.

·.Y WORDS　SINGLE-STEP interrupt, a program encryptior., ·chnology, against trace technology