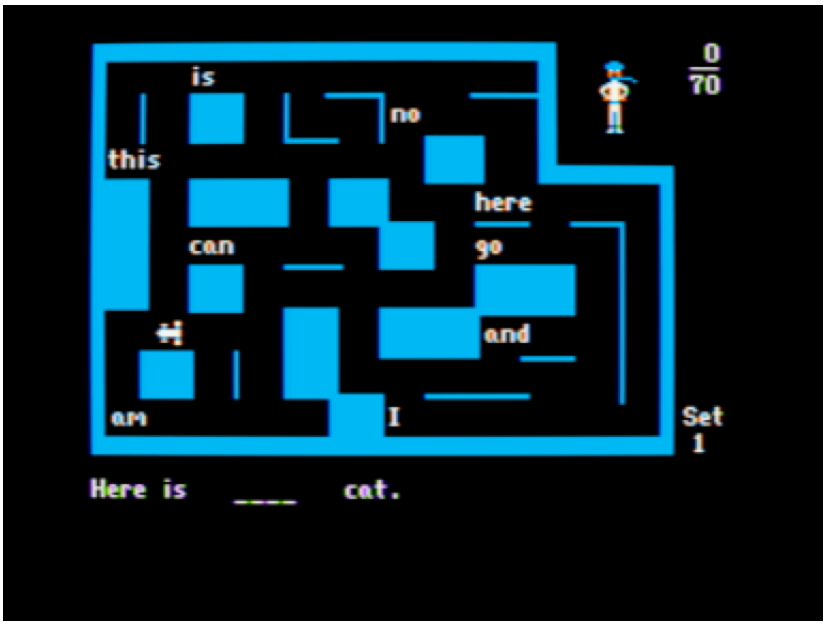
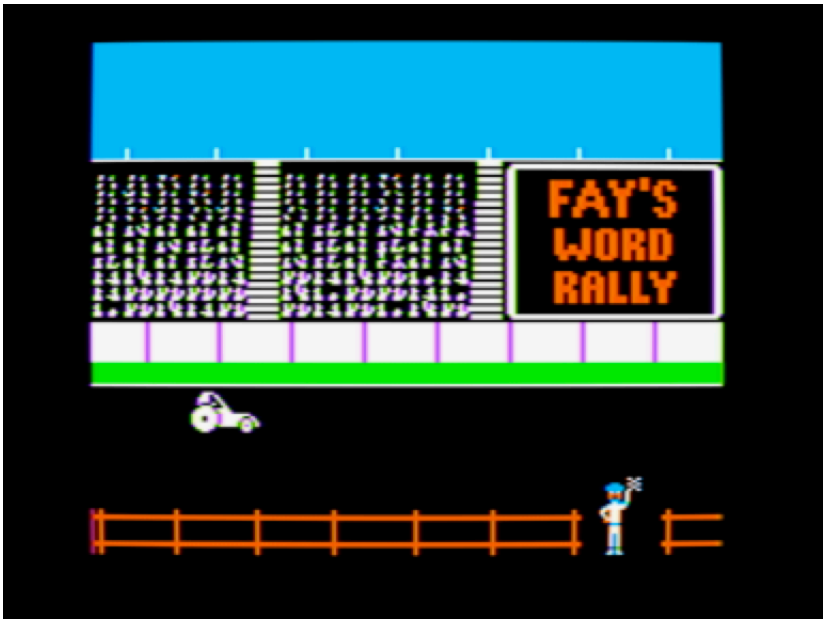
Fay's Word Rally



2016-07-01

4am
to deprotect
and preserve

Contents



----------Fay's Word Rally----------
A 4am crack 2016-07-01

Name: Fay's Word Rally
Genre: educational
Year: 1987
Authors: ...TODO
Publisher: Didatech Software
Media: double-sided 5.25-inch floppy
OS: Pronto-DOS
Previous cracks: none

Side A is bootable but protected.
Side B is unbootable but unprotected.
Life is like that.
This has not been a haiku.

Chapter 0
In Which Various Automated Tools Fail In Interesting Ways

COPYA
  fails on first pass

Locksmith Fast Disk Backup
  can read every sector except T02,S07;
  copy displays title screen then quits
  to BASIC prompt with DOS disconnected

EDD 4 bit copy (no sync, no count)
  ditto

Copy][+ nibble editor
  There's an address field for T02,S07,
  but no data

Disk Fixer
  T00 -> DOS 3.3-shaped RWTS
  T11 -> DOS 3.3 disk catalog
  T01,S07 -> startup program is "DV"

Why didn't COPYA work?
  intentionally unreadable sector on
  track $02

Why didn't Locksmith FDB / EDD work?
  probably a nibble check that checks
  that unreadable sector

Next steps:

  1. Trace startup program
  2. Find nibble check and disable it
  3. There is no step 3 (I hope)

Chapter 1
In Which You Ain't Gonna Need It (Until You Do)

```
[S6,D1=non-working copy (side A)]
[S5,D1=my work disk]

]PR#5
...
CAPTURING BOOT0
...reboots slot 6...
...reboots slot 5...
SAVING BOOT0
CAPTURING BOOT1
...reboots slot 6...
...reboots slot 5...
SAVING BOOT1
SAVING RWTS
```

I probably don't need that since the disk is (539/540)% copyable, but OK.

My non-working copy appeared to load DOS and execute a startup program, so let's start there.

```
]BLOAD DV,S6,D1
]CALL -151

; under Diversi-DOS 64K, the last BLOAD
; address is at $BF55, and length is at
; $BF51
*BF55.BF56

BF55- 02 08     ; A$802
```

```
*BF51.BF52

BF51- 6B 01      ; L$16B

*802L

; set reset vector
0802-   A9 18       LDA   #$18
0804-   8D F2 03    STA   $03F2
0807-   A9 09       LDA   #$09
0809-   8D F3 03    STA   $03F3
080C-   49 A5       EOR   #$A5
080E-   8D F4 03    STA   $03F4

; slow to 1 MHz (IIgs)
0811-   AD 36 C0    LDA   $C036
0814-   29 7F       AND   #$7F
0816-   8D 36 C0    STA   $C036

; "MAXFILES 1"
0819-   A9 01       LDA   #$01
081B-   85 44       STA   $44
081D-   20 51 A2    JSR   $A251
```

```
; display text screen with title and
; "LOADING PLEASE WAIT" message
0820-   AD 51 C0    LDA   $C051
0823-   AD 54 C0    LDA   $C054
0826-   A9 00       LDA   #$00
0828-   85 20       STA   $20
082A-   85 22       STA   $22
082C-   A9 18       LDA   #$18
082E-   85 23       STA   $23
0830-   A9 28       LDA   #$28
0832-   85 21       STA   $21
0834-   20 58 FC    JSR   $FC58
0837-   A0 00       LDY   #$00
0839-   B9 46 09    LDA   $0946,Y
083C-   F0 06       BEQ   $0844
083E-   99 B0 05    STA   $05B0,Y
0841-   C8          INY
0842-   D0 F5       BNE   $0839
0844-   A0 00       LDY   #$00
0846-   B9 60 09    LDA   $0960,Y
0849-   F0 06       BEQ   $0851
084B-   99 36 07    STA   $0736,Y
084E-   C8          INY
084F-   D0 F5       BNE   $0846

; copy a string from $08EE to the input
; buffer at $0200
0851-   A9 EE       LDA   #$EE
0853-   85 02       STA   $02
0855-   A9 08       LDA   #$08
0857-   85 03       STA   $03
0859-   A0 00       LDY   #$00
085B-   B1 02       LDA   ($02),Y
085D-   F0 06       BEQ   $0865
085F-   99 00 02    STA   $0200,Y
0862-   C8          INY
0863-   D0 F6       BNE   $085B
```

```
; and a <RETURN> character
0865-   A9 8D       LDA   #$8D
0867-   99 00 02    STA   $0200,Y
```

What exactly are we executing via the input buffer?

```
*FC58G N 400<8EE.8FFM
```

--v--

```
BLOAD FWR@...
```

--^--

Well OK then.

```
Continuing from $086A...

086A-   A5 37       LDA   $37
086C-   85 04       STA   $04
086E-   A5 36       LDA   $36
0870-   85 03       STA   $03
0872-   A9 00       LDA   #$00
0874-   85 06       STA   $06

; highly suspect -- fiddling with DOS
; internals
0876-   A9 60       LDA   #$60
0878-   8D D5 A6    STA   $A6D5
087B-   AD AC A6    LDA   $A6AC
087E-   85 07       STA   $07
```

```
; some sort of callback? (putting a
; "JMP $08D7" command in the middle of
; the DOS command parser)
0880-   A9 4C       LDA   #$4C
0882-   8D AB A6    STA   $A6AB
0885-   A9 D7       LDA   #$D7
0887-   8D AC A6    STA   $A6AC
088A-   A9 08       LDA   #$08
088C-   8D AD A6    STA   $A6AD
088F-   20 D1 08    JSR   $08D1

*8D1L

; save registers
08D1-   20 D1 9E    JSR   $9ED1

; parse DOS command
08D4-   4C CD 9F    JMP   $9FCD
```

That will execute what's in the input buffer at $0200, which is "BLOAD FWR". That's a real file on the disk; it shows up in a CATALOG and everything.

Eventually it calls this callback, because of the "JMP $08D7" we just put at $A6AB:

```
*8D7L

08D7-   90 08       BCC   $08E1

; take DOS error code
08D9-   AD C5 B5    LDA   $B5C5

; store it in zero page
08DC-   85 06       STA   $06
```

```
; and continue
08DE-   4C B0 A6    JMP   $A6B0
08E1-   60          RTS
```

Continuing from $0892...

```
; restore the DOS code we overwrote
; earlier (at $0876+)
0892-   A9 20       LDA   #$20
0894-   8D D5 A6    STA   $A6D5
0897-   A9 90       LDA   #$90
0899-   8D AB A6    STA   $A6AB
089C-   A5 07       LDA   $07
089E-   8D AC A6    STA   $A6AC
08A1-   A9 AD       LDA   #$AD
08A3-   8D AD A6    STA   $A6AD

; check DOS file type from that BLOAD
08A6-   AD F6 B5    LDA   $B5F6
08A9-   29 07       AND   #$07
08AB-   F0 08       BEQ   $08B5
08AD-   C9 04       CMP   #$04
08AF-   F0 04       BEQ   $08B5

; set error code manually if the DOS
; file type doesn't match expectations
08B1-   A9 0B       LDA   #$0B
08B3-   85 06       STA   $06

; now take the DOS error code and use
; it as an index into an array
08B5-   A6 06       LDX   $06
08B7-   BD E2 08    LDA   $08E2,X

; and store *that*
08BA-   85 08       STA   $08
```

```
; restore I/O vector
08BC-   A5 04       LDA   $04
08BE-   85 37       STA   $37
08C0-   A5 03       LDA   $03
08C2-   85 36       STA   $36
08C4-   AE 59 AA    LDX   $AA59
08C7-   9A          TXS
08C8-   68          PLA
08C9-   68          PLA

; check the final value
08CA-   A5 08       LDA   $08
08CC-   C9 01       CMP   #$01
08CE-   4C F8 08    JMP   $08F8

*8F8L

; if the final value (in zero page $08,
; as determined by the array at $08E2,
; using the DOS error code as an index,
; as captured by the callback at $08D7)
; is 0, branch forward to $0943
08F8-   90 49       BCC   $0943

; all other values fall through to here
;
; clear screen and print "LOAD ERROR"
08FA-   20 58 FC    JSR   $FC58
08FD-   A0 00       LDY   #$00
08FF-   B9 0A 09    LDA   $090A,Y
0902-   F0 14       BEQ   $0918
0904-   20 ED FD    JSR   $FDED
0907-   C8          INY
0908-   D0 F5       BNE   $08FF
...
```

```
; set reset vector
0918-   A9 BF       LDA   #$BF
091A-   8D F2 03    STA   $03F2
091D-   A9 9D       LDA   #$9D
091F-   8D F3 03    STA   $03F3
0922-   A9 38       LDA   #$38
0924-   8D F4 03    STA   $03F4

; wipe main memory starting at $6000
0927-   A9 00       LDA   #$00
0929-   85 03       STA   $03
092B-   A9 60       LDA   #$60
092D-   85 04       STA   $04
092F-   A0 00       LDY   #$00
0931-   A9 00       LDA   #$00
0933-   91 03       STA   ($03),Y
0935-   C8          INY
0936-   D0 FB       BNE   $0933
0938-   E6 04       INC   $04
093A-   A5 04       LDA   $04
093C-   C9 96       CMP   #$96
093E-   90 F1       BCC   $0931

; exit to BASIC prompt (with no DOS in
; memory)
0940-   4C 00 E0    JMP   $E000
```

That's all well and good, but that is *not* the error I'm seeing on my non-working copy. My copy is loading the "FWR" file correctly; when it finally fails, it does not print the "LOAD ERROR" message.

```
Continuing from $0943...

; execution continues here (from $08FA)
; if carry bit is clear, meaning that
; zero page $08 is #$00, meaning that
; there was no DOS error BLOAD-ing the
; "FWR" file
0943-   4C E1 BC    JMP   $BCE1
```

And now I'm glad I captured the RWTS from the original disk.

Chapter 2
Holy Hell, That Was A Lot Of Work Just To Load A Single File, I Hope It's Important

```
*BLOAD RWTS,A$3800,S5
*B800<3800.3EFFM
*BCE1L

; decrypt the code at $6000 (which is
; where that "FWR" file was loaded)
BCE1-   A2 35       LDX   #$35
BCE3-   A0 03       LDY   #$03
BCE5-   A9 53       LDA   #$53
BCE7-   59 00 60    EOR   $6000,Y
BCEA-   99 00 60    STA   $6000,Y
BCED-   C8          INY
BCEE-   D0 F5       BNE   $BCE5
BCF0-   EE E9 BC    INC   $BCE9
BCF3-   EE EC BC    INC   $BCEC
BCF6-   CA          DEX
BCF7-   D0 EC       BNE   $BCE5
BCF9-   4C 89 BA    JMP   $BA89

*BA89L

; wipe the previous decryption routine
BA89-   A2 1C       LDX   #$1C
BA8B-   98          TYA
BA8C-   9D E0 BC    STA   $BCE0,X
BA8F-   CA          DEX
BA90-   D0 FA       BNE   $BA8C

; and jump into the decrypted code
BA92-   4C 93 78    JMP   $7893

OK, let's do that (but stop at $BA92).

*BLOAD FWR,S6

*BA92:60  ; exit via RTS instead of JMP

*BCE1G    ; run decryption routine
```

```
*BSAVE FWR DECRYPTED,S5

(Diversi-DOS 64K automatically adds the
previous starting address and length.
For reference, it's A$6000,L$38E6.)

*7893L

; sets a reset vector (not shown)
7893-   20 9E 7B    JSR   $7B9E

; is #$FF, so branch is not taken
; (maybe a one-time setup flag?)
7896-   AD CF 6E    LDA   $6ECF
7899-   10 15       BPL   $78B0
789B-   AD EE 6B    LDA   $6BEE
789E-   AE 02 6C    LDX   $6C02
78A1-   A0 B1       LDY   #$B1
78A3-   8C 03 6C    STY   $6C03
78A6-   C8          INY
78A7-   8C EF 6B    STY   $6BEF
78AA-   8D 02 6C    STA   $6C02
78AD-   8E EE 6B    STX   $6BEE
78B0-   A0 00       LDY   #$00
78B2-   84 68       STY   $68
78B4-   8C 91 86    STY   $8691
78B7-   84 F5       STY   $F5
78B9-   84 5B       STY   $5B
78BB-   20 C5 85    JSR   $85C5

*85C5L

85C5-   A0 00       LDY   #$00
85C7-   84 56       STY   $56
85C9-   84 1B       STY   $1B
85CB-   A9 50       LDA   #$50
85CD-   85 57       STA   $57

Now ($56) points to $5000.
```

```
; pass a byte to $8692...
85CF-   A9 BD       LDA   #$BD
85D1-   20 92 86    JSR   $8692

*8692L

; ...which stores it in ($56), which
; starts at $5000 and is incremented
; after each byte stored
8692-   91 56       STA   ($56),Y
8694-   E6 56       INC   $56
8696-   D0 02       BNE   $869A
8698-   E6 57       INC   $57
869A-   60          RTS
```

Ah! We're sneakily creating code, one byte at a time.

Continuing from $85D4...

```
; more sneaky code generation
85D4-   A9 8C       LDA   #$8C
85D6-   20 92 86    JSR   $8692
85D9-   A9 C0       LDA   #$C0
85DB-   20 92 86    JSR   $8692
85DE-   A9 8D       LDA   #$8D
85E0-   20 92 86    JSR   $8692
85E3-   A9 C0       LDA   #$C0
85E5-   18          CLC
85E6-   65 1B       ADC   $1B
85E8-   20 92 86    JSR   $8692
85EB-   A9 50       LDA   #$50
85ED-   20 92 86    JSR   $8692
85F0-   E6 1B       INC   $1B
85F2-   A5 1B       LDA   $1B
85F4-   C9 1E       CMP   #$1E
85F6-   90 D7       BCC   $85CF
```

```
; one final byte (looks like an "RTS")
85F8-   A9 60       LDA   #$60
85FA-   20 92 86    JSR   $8692

; get address of RWTS parameter table
85FD-   20 E3 03    JSR   $03E3
8600-   84 CE       STY   $CE
8602-   85 CF       STA   $CF

; set up RWTS parameters
8604-   A9 02       LDA   #$02
8606-   A0 04       LDY   #$04

; track $02
8608-   91 CE       STA   ($CE),Y
860A-   A9 00       LDA   #$00
860C-   A0 0C       LDY   #$0C

; command $00 (seek)
860E-   91 CE       STA   ($CE),Y

; any disk volume
8610-   A0 03       LDY   #$03
8612-   91 CE       STA   ($CE),Y

; and do it
8614-   20 E3 03    JSR   $03E3
8617-   20 D9 03    JSR   $03D9
861A-   B0 27       BCS   $8643
```

OK, we're on track $02, the track with the unreadable sector.

```
; turn on drive motor manually
861C-   BD 89 C0    LDA   $C089,X
```

```
; initialize Death Counter
861F-   A9 30       LDA   #$30
8621-   8D 78 05    STA   $0578
8624-   38          SEC
8625-   CE 78 05    DEC   $0578
8628-   F0 19       BEQ   $8643

; look for next available address field
862A-   20 44 B9    JSR   $B944
862D-   B0 F5       BCS   $8624

; physical sector 1 (logical sector 7)
862F-   A5 2D       LDA   $2D
8631-   C9 01       CMP   #$01

; loop until we find it
8633-   D0 EF       BNE   $8624

; reset data latch and wait
8635-   BD 8E C0    LDA   $C08E,X
8638-   A9 06       LDA   #$06
863A-   20 A8 FC    JSR   $FCA8

; now execute the routine we created
; one byte at a time
863D-   20 00 50    JSR   $5000
```

OK, it is long past time to see what code is generated at $5000 (starting back at $85CF).

```
; put an RTS after the final code
; generation call
*85FD:60

; and execute it
*85C5G
```

```
*5000L

5000-   BD 8C C0    LDA   $C08C,X
5003-   8D C0 50    STA   $50C0
5006-   BD 8C C0    LDA   $C08C,X
5009-   8D C1 50    STA   $50C1
500C-   BD 8C C0    LDA   $C08C,X
500F-   8D C2 50    STA   $50C2
5012-   BD 8C C0    LDA   $C08C,X
5015-   8D C3 50    STA   $50C3
5018-   BD 8C C0    LDA   $C08C,X
501B-   8D C4 50    STA   $50C4
501E-   BD 8C C0    LDA   $C08C,X
5021-   8D C5 50    STA   $50C5
5024-   BD 8C C0    LDA   $C08C,X
5027-   8D C6 50    STA   $50C6
502A-   BD 8C C0    LDA   $C08C,X
502D-   8D C7 50    STA   $50C7
5030-   BD 8C C0    LDA   $C08C,X
5033-   8D C8 50    STA   $50C8
5036-   BD 8C C0    LDA   $C08C,X
5039-   8D C9 50    STA   $50C9
503C-   BD 8C C0    LDA   $C08C,X
503F-   8D CA 50    STA   $50CA
5042-   BD 8C C0    LDA   $C08C,X
5045-   8D CB 50    STA   $50CB
5048-   BD 8C C0    LDA   $C08C,X
504B-   8D CC 50    STA   $50CC
504E-   BD 8C C0    LDA   $C08C,X
5051-   8D CD 50    STA   $50CD
5054-   BD 8C C0    LDA   $C08C,X
5057-   8D CE 50    STA   $50CE
505A-   BD 8C C0    LDA   $C08C,X
505D-   8D CF 50    STA   $50CF
5060-   BD 8C C0    LDA   $C08C,X
5063-   8D D0 50    STA   $50D0
5066-   BD 8C C0    LDA   $C08C,X
5069-   8D D1 50    STA   $50D1
506C-   BD 8C C0    LDA   $C08C,X
506F-   8D D2 50    STA   $50D2
[...]
```

```
5072-   BD 8C C0    LDA   $C08C,X
5075-   8D D3 50    STA   $50D3
5078-   BD 8C C0    LDA   $C08C,X
507B-   8D D4 50    STA   $50D4
507E-   BD 8C C0    LDA   $C08C,X
5081-   8D D5 50    STA   $50D5
5084-   BD 8C C0    LDA   $C08C,X
5087-   8D D6 50    STA   $50D6
508A-   BD 8C C0    LDA   $C08C,X
508D-   8D D7 50    STA   $50D7
5090-   BD 8C C0    LDA   $C08C,X
5093-   8D D8 50    STA   $50D8
5096-   BD 8C C0    LDA   $C08C,X
5099-   8D D9 50    STA   $50D9
509C-   BD 8C C0    LDA   $C08C,X
509F-   8D DA 50    STA   $50DA
50A2-   BD 8C C0    LDA   $C08C,X
50A5-   8D DB 50    STA   $50DB
50A8-   BD 8C C0    LDA   $C08C,X
50AB-   8D DC 50    STA   $50DC
50AE-   BD 8C C0    LDA   $C08C,X
50B1-   8D DD 50    STA   $50DD
50B4-   60          RTS
```

Awesome. Note: no BPL loops. This will just keep reading the data latch as fast as possible and storing the raw partial nibble values in $50C0..$50DD, which will be sensitive to any timing bits between the nibbles. Good luck copying that with a nibble copier!

```
Continuing from $8640...

8640-   18          CLC
8641-   90 04       BCC   $8647

;[skipped]
;8643-   A0 0D       LDY   #$0D
;8645-   B1 CE       LDA   ($CE),Y

; turn off drive motor
8647-   9D 88 C0    STA   $C088,X

; reset zero page after RWTS call
864A-   A0 00       LDY   #$00
864C-   84 48       STY   $48

; This whole thing is a loop to check
; the partial nibble values that were
; stored by the routine at $5000. It
; does some contortions to skip over
; self-sync nibbles (#$FF) and has a
; Death Counter to make sure it finds
; the magic sequence in time.
864E-   B0 38       BCS   $8688
8650-   84 56       STY   $56
8652-   A2 00       LDX   #$00
8654-   84 1B       STY   $1B
8656-   BD 9B 86    LDA   $869B,X
8659-   85 CE       STA   $CE
865B-   BD A0 86    LDA   $86A0,X
865E-   85 CF       STA   $CF
8660-   A0 00       LDY   #$00
8662-   B1 CE       LDA   ($CE),Y
8664-   C9 FF       CMP   #$FF
8666-   D0 0A       BNE   $8672
[...]
```

```
8668-   E6 56        INC   $56
866A-   A4 56        LDY   $56
866C-   C0 1A        CPY   #$1A
866E-   90 E2        BCC   $8652
8670-   B0 16        BCS   $8688
8672-   84 08        STY   $08
8674-   A4 1B        LDY   $1B
8676-   D9 C0 50     CMP   $50C0,Y
8679-   F0 05        BEQ   $8680
867B-   A4 08        LDY   $08
867D-   C8           INY
867E-   D0 E2        BNE   $8662
8680-   E6 1B        INC   $1B
8682-   E8           INX
8683-   E0 05        CPX   #$05
8685-   90 CF        BCC   $8656

; success path falls through to here
8687-   18           CLC

; failure ends up here (from $8670)
; with the carry bit set
8688-   A0 FE        LDY   #$FE
868A-   90 01        BCC   $868D

; only failure path will execute this
; instruction (because the success path
; cleared the carry bit at $8687)
868C-   C8           INY
```

```
; execution continues here regardless
868D-   8C 91 86    STY   $8691
8690-   60          RTS
```

OK, so there's the difference between an original disk and a copy: the value of the Y register at $868D, which gets stored in $8691.

```
original = #$FE
copy     = #$FF
```

Continuing from $78BE...

```
78BE-   20 CE 86    JSR   $86CE

*86CEL

; and we're immediately checking the
; value that determines whether this is
; an original disk
86CE-   AD 91 86    LDA   $8691
86D1-   C9 FE       CMP   #$FE

; original disk -> exit gracefully
86D3-   F0 03       BEQ   $86D8

; copy -> jump to The Badlands
86D5-   4C 5F 78    JMP   $785F
86D8-   60          RTS

*785FL

; set reset vector
785F-   A9 BF       LDA   #$BF
7861-   8D F2 03    STA   $03F2
7864-   A9 9D       LDA   #$9D
7866-   8D F3 03    STA   $03F3
7869-   A9 38       LDA   #$38
786B-   8D F4 03    STA   $03F4
```

```
; wipe main memory starting at $7893
; (immediately after this code)
786E-   A9 00       LDA   #$00
7870-   85 56       STA   $56
7872-   A9 78       LDA   #$78
7874-   85 57       STA   $57
7876-   A0 93       LDY   #$93
7878-   A9 00       LDA   #$00
787A-   91 56       STA   ($56),Y
787C-   C8          INY
787D-   D0 FB       BNE   $787A
787F-   E6 57       INC   $57
7881-   A5 57       LDA   $57
7883-   C9 96       CMP   #$96
7885-   90 F1       BCC   $7878

; TEXT, HOME, reset I/O vectors, and
; exit to a BASIC prompt with no DOS
; in memory
7887-   20 2F FB    JSR   $FB2F
788A-   20 58 FC    JSR   $FC58
788D-   20 51 A8    JSR   $A851
7890-   4C 00 E0    JMP   $E000
```

That is exactly the behavior I saw on my non-working copy.

Chapter 3
One Byte To Rule Them All, And In The Encrypted File Bind Them

To make my non-working copy look like an original disk, I can change a single instruction:

```
868C-   C8          INY
```

into

```
868C-   EA          NOP
```

Problem: this instruction is inside the "FWR" file, which is encrypted on disk (and decrypted in memory at $BCE1).

Possible solution #1: replace the "FWR" file with the decrypted version that I captured on my work disk, disable the decryption routine at $BCE1, and change the "INY" instruction to "NOP" on disk.

Total bytes changed: 14566 ($38E6, the length of the "FWR" file)

Possible solution #2: calculate the encrypted value of #$EA (NOP) and replace the encrypted value of #$C8 (INY). The encryption is a simple XOR with a constant (#$53), so this won't affect the surrounding code.

Total bytes changed: 1

Door #2 it is.

Let's see...

#$C8 XOR #$53 = #$9B

#$EA XOR #$53 = #$B9

$868C - $6000 = $268C

$268C / $100 = $26

Following the "FWR" file with my trusty Disk Fixer sector editor, I count out to the $26-th sector and look at the $90-th byte ($8C + 4 to compensate for the four-byte header at the beginning of all DOS 3.3 files), and lo and behold! It's #$9B. I've found the encrypted "INY" instruction.

Let's change it to an encrypted "NOP" instruction:

T15,S06,$90: 9B -> B9

]PR#6
...works...

Side B is unprotected.

Quod erat liberandum.

Acknowledgments

Many thanks to LoGo for supplying the
the original floppy disk.