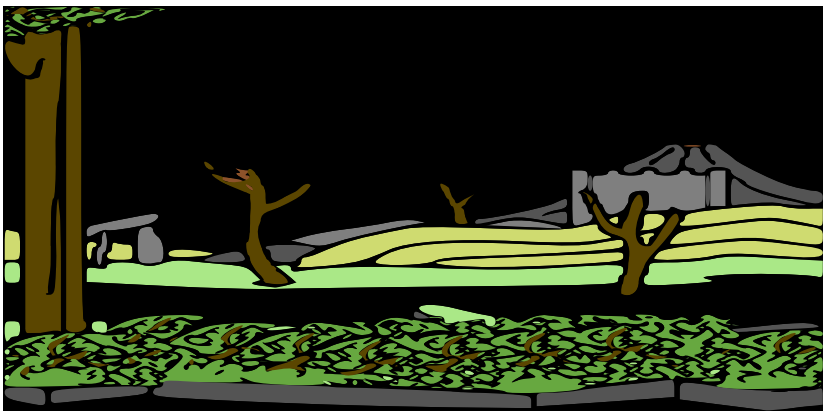
Nemesis



2016-08-19

4am
to deprotect
and preserve

Contents

----------Nemesis----------
A 4am crack 2016-08-19

Name: Nemesis
Genre: adventure
Year: 1984
Authors: Kyle Freeman and Gary Wilens
Publisher: none (*)
Platform: Apple][+ or later (64K)
Media: double-sided 5.25-inch floppy
OS: custom
Previous cracks: none

(*) A later version was published in 1987 by Datasoft as "Dark Lord."

Chapter 0
In Which Various Automated Tools Fail In Interesting Ways

COPYA
  immediate disk read error

Locksmith Fast Disk Backup
  unable to read any track

EDD 4 bit copy (no sync, no count)
  copy boots and loads several tracks,
  then hangs with the drive motor off

```
Copy ][+ nibble editor
  track $00 uses standard prologues,
    modified epilogue
  track $04 is... special (see below)

T01-T03 and T05-T22 look like this:

                    --v--

TRACK: 01  START: 1D54  LENGTH: 188C

1D30: FF FF 94 94 FF FF FF FF   VIEW
1D38: FF FF A4 DC FF FF FF 94
1D40: FF FF FF FF 92 FF FF FF
1D48: FF FF FF FF FF FF FF AF
1D50: DF FF FF FF FF 96 FF 96  <-1D54
                      ^^^^^^^^
                    address prologue?

1D58: AB AA AA AB AA AA AB AB
      ^^^^^ ^^^^^ ^^^^^ ^^^^^
      V=002  T=1   S=0  chksm

1D60: EE FF FF FF BF 9F FF 9F
1D68: E7 F9 FE FF 96 FF D7 9A
                      ^^^^^
                    data prologue?

1D70: F7 B6 AB DA AC E6 BD B6
      ^^

                    --^--

That looks very similar to the standard
16-sector track/sector format. There is
an address field with the usual values
in the usual order. I spot-checked a
few tracks, and they all share this
structure, even the same (non-standard)
prologues and epilogues on each track.
```

Track $04, on the other hand, looks like this:

```
                    --v--

TRACK: 04  START: 1800  LENGTH: 3DFF

1E60: FF FF FF FF FF FF FF FF    VIEW
1E68: FF CA FF FF FF FF FF FF
1E70: FF FF FF FF FF FF FF FF
1E78: 9A FF FF FF C9 FF FF AA
1E80: AA FF FF FF FF D5 96 FF
1E88: FF FF FF FF FF FF FF A4
1E90: FF FF FF FF FF FF FF FF
1E98: FF B2 B2 FF FF FF FF FF
1EA0: CD FF FF FF FF FF FF AD

                    --^--
```

I don't know what that is, but as it's the only track that's different from all the others, I'm gonna go with BIG HONKIN' NIBBLE CHECK.

Disk Fixer
  I can change the prologues and set "CHECKSUM ENABLED" to "NO" to ignore epilogues, but I still can't read any track beyond track 0. Either I've misinterpreted the Copy II Plus nibble editor analysis, or something else is going on. Maybe a custom nibble translate table? Impossible to know at this point.

Why didn't COPYA work?
  non-standard prologues (track $01+)

Why didn't Locksmith FDB work?
  non-standard prologues (track $01+)

Why didn't my EDD copy work?
  This is an excellent question. I'm going to take an educated guess that there is some runtime protection check centered around the unreadable track $04.

Next steps:

  1. Trace the boot
  2. Capture the RWTS
  3. Convert the disk to a standard format with Advanced Demuffin
  4. Patch the RWTS
  5. Disable the nibble check
  6. I don't know, go feed the ducks or something?

Chapter 1
Humble Beginnings

```
]BLOAD BOOT0,A$800
]CALL -151

*801L

; identical to DOS 3.3 bootloader
0801-   A5 27        LDA   $27
0803-   C9 09        CMP   #$09
0805-   D0 18        BNE   $081F
0807-   A5 2B        LDA   $2B
0809-   4A           LSR
080A-   4A           LSR
080B-   4A           LSR
080C-   4A           LSR
080D-   09 C0        ORA   #$C0
080F-   85 3F        STA   $3F
0811-   A9 5C        LDA   #$5C
0813-   85 3E        STA   $3E
0815-   18           CLC

; up to here -- usually this would load
; from $08FE/$08FF
0816-   AD 5E 08     LDA   $085E
0819-   6D 5F 08     ADC   $085F
081C-   8D 5E 08     STA   $085E
081F-   AE 5F 08     LDX   $085F
0822-   30 15        BMI   $0839
0824-   BD 4D 08     LDA   $084D,X
0827-   85 3D        STA   $3D
0829-   CE 5F 08     DEC   $085F
082C-   AD 5E 08     LDA   $085E
082F-   85 27        STA   $27
0831-   CE 5E 08     DEC   $085E

; call drive firmware to read next
; sector, exits via $0801, so this
; whole thing is a loop
0834-   A6 2B        LDX   $2B
0836-   6C 3E 00     JMP   ($003E)
```

```
; loop escapes here (from $0822)
0839-   EE 5E 08    INC   $085E
083C-   EE 5E 08    INC   $085E
083F-   20 89 FE    JSR   $FE89
0842-   20 93 FE    JSR   $FE93
0845-   20 2F FB    JSR   $FB2F
0848-   A6 2B       LDX   $2B

; execution continues in a sector we
; just loaded
084A-   4C 60 09    JMP   $0960

*85E.85F

085E- 09 0E
```

OK, so we're reading basically the entire track $00 into $0900+. Let's capture that and see what's going on.

```
*9600<C600.C6FFM

; set up callback after sector read
; loop exits and we would ordinarily
; just to $0960 for second-stage boot
96F8-   A9 4C       LDA   #$4C
96FA-   8D 4A 08    STA   $084A
96FD-   A9 0A       LDA   #$0A
96FF-   8D 4B 08    STA   $084B
9702-   A9 97       LDA   #$97
9704-   8D 4C 08    STA   $084C

; start the boot
9707-   4C 01 08    JMP   $0801
```

```
; callback is here -- copy everything
; we read to higher memory so it
; survives a reboot
970A-   A2 10       LDX   #$10
970C-   A0 00       LDY   #$00
970E-   B9 00 08    LDA   $0800,Y
9711-   99 00 28    STA   $2800,Y
9714-   C8          INY
9715-   D0 F7       BNE   $970E
9717-   EE 10 97    INC   $9710
971A-   EE 13 97    INC   $9713
971D-   CA          DEX
971E-   D0 EE       BNE   $970E

; turn off slot 6 drive motor
9720-   AD E8 C0    LDA   $C0E8

; reboot to my work disk
9723-   4C 00 C5    JMP   $C500

*BSAVE TRACE,A$9600,L$126
*9600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT1 0800-17FF,A$2800,L$1000
```

Chapter 2
I'm In Shape!
"Pear" Is A Shape

```
]CALL -151

; move all the code back into place
*800<2800.37FFM

; and see what's lurking at $0960
*960L

0960-   20 22 0A    JSR   $0A22

*A22L

; copy code to language card RAM bank 1
; ($1000..$17FF -> $D000..$D7FF)
0A22-   AD 8B C0    LDA   $C08B
0A25-   AD 8B C0    LDA   $C08B
0A28-   BD 88 C0    LDA   $C088,X
0A2B-   A9 10       LDA   #$10
0A2D-   85 07       STA   $07
0A2F-   A9 D0       LDA   #$D0
0A31-   85 09       STA   $09
0A33-   A9 00       LDA   #$00
0A35-   85 06       STA   $06
0A37-   85 08       STA   $08
0A39-   A0 00       LDY   #$00
0A3B-   B1 06       LDA   ($06),Y
0A3D-   91 08       STA   ($08),Y
0A3F-   88          DEY
0A40-   D0 F9       BNE   $0A3B
0A42-   E6 09       INC   $09
0A44-   E6 07       INC   $07
0A46-   A5 09       LDA   $09
0A48-   C9 D8       CMP   #$D8
0A4A-   D0 ED       BNE   $0A39
```

```
; set reset vectors
0A4C-   A9 DB       LDA   #$DB
0A4E-   8D F2 03    STA   $03F2
0A51-   8D FA FF    STA   $FFFA
0A54-   8D FC FF    STA   $FFFC
0A57-   8D FE FF    STA   $FFFE
0A5A-   A9 09       LDA   #$09
0A5C-   8D F3 03    STA   $03F3
0A5F-   8D FB FF    STA   $FFFB
0A62-   8D FD FF    STA   $FFFD
0A65-   8D FF FF    STA   $FFFF
0A68-   AD F3 03    LDA   $03F3
0A6B-   49 A5       EOR   #$A5
0A6D-   8D F4 03    STA   $03F4

; clear text screen (not shown)
0A70-   20 C1 09    JSR   $09C1
0A73-   AD 30 C0    LDA   $C030

; copy boot slot (x16) from zero page
; to RAM bank 1 (where we just copied
; a bunch of other code from $1000+)
0A76-   A5 2B       LDA   $2B
0A78-   8D 01 D0    STA   $D001
0A7B-   8D 0F D0    STA   $D00F

Wait, is that an RWTS that we just
copied to $D000+? Let's find out.

*C500G
[...hold down <Esc> during boot so that
    Diversi-DOS doesn't relocate to the
    RAM bank we're about to clobber...]

]CALL -151

; copy the monitor ROM to RAM bank 1 so
; we can tool around in the monitor
; without crashing
*C089 C089 F800<F800.FFFFM
```

```
; now fully switch to RAM bank 1
*C08B C08B

; reproduce copy loop from bootloader
*D000<1000.17FFM

*D000.

D000- 01 60 01 00 02 0B 13 D0
D008- 00 02 00 00 04 00 01 60
D010- 01 00 00 00 01 EF D8 00

That looks suspiciously like an RWTS
parameter table, the kind that normally
lives at $B7E8.

*D018L

; looks like an RWTS entry point, the
; kind that normally lives at $B7B5
D018-   A0 00       LDY   #$00
D01A-   A9 D0       LDA   #$D0
D01C-   08          PHP
D01D-   78          SEI
D01E-   20 DF D5    JSR   $D5DF   <-- !
D021-   B0 08       BCS   $D02B
D023-   28          PLP
D024-   18          CLC
D025-   A9 00       LDA   #$00
D027-   8D 0D D0    STA   $D00D
D02A-   60          RTS
D02B-   28          PLP
D02C-   38          SEC
D02D-   60          RTS
```

It calls a lower-level RWTS entry point at $D5DF, which also looks familiar:

```
*D5DFL

D5DF-   84 48       STY   $48
D5E1-   85 49       STA   $49
D5E3-   A0 02       LDY   #$02
D5E5-   8C F8 06    STY   $06F8
D5E8-   A0 04       LDY   #$04
D5EA-   8C F8 04    STY   $04F8
D5ED-   A0 01       LDY   #$01
D5EF-   B1 48       LDA   ($48),Y
D5F1-   AA          TAX
D5F2-   A0 0F       LDY   #$0F
D5F4-   D1 48       CMP   ($48),Y
D5F6-   F0 1B       BEQ   $D613
D5F8-   8A          TXA
D5F9-   48          PHA
D5FA-   B1 48       LDA   ($48),Y
D5FC-   AA          TAX
D5FD-   68          PLA
D5FE-   48          PHA
D5FF-   91 48       STA   ($48),Y
D601-   BD 8E C0    LDA   $C08E,X
```

This is definitely what I would call a "DOS 3.3-shaped" RWTS. Why is that important? Because I have exactly the right tool to use this RWTS as a weapon against the original disk.

Chapter 3
In Which We Attempt To Use The Original Disk As A Weapon Against Itself

Given an RWTS that can read a disk, the cracker's tool of choice is Advanced Demuffin, which I've included on my work disk.

```
*BLOAD ADVANCED DEMUFFIN 1.5
```

Since this RWTS does not have the usual entry point (at $BD00), I'll need to write an IOB module to tell Advanced Demuffin how to call the custom RWTS. (Read the Advanced Demuffin docs on my work disk for more on IOB modules.)

```
; standard Advanced Demuffin setup
; (unchanged)
1400-   4A          LSR
1401-   8D 22 0F    STA   $0F22
1404-   8C 23 0F    STY   $0F23
1407-   8E 27 0F    STX   $0F27
140A-   A9 01       LDA   #$01
140C-   8D 20 0F    STA   $0F20
140F-   8D 2A 0F    STA   $0F2A

; switch to RAM bank 1
1412-   AD 8B C0    LDA   $C08B
1415-   AD 8B C0    LDA   $C08B

; call RWTS at $D5DF
1418-   A9 0F       LDA   #$0F
141A-   A0 1E       LDY   #$1E
141C-   20 DF D5    JSR   $D5DF

; save return code (including carry)
141F-   08          PHP

; switch back to ROM
1420-   AD 82 C0    LDA   $C082
1423-   AD 82 C0    LDA   $C082
```

```
; restore return code
1426-   28          PLP

; and exit back to Advanced Demuffin
1427-   60          RTS

*BSAVE IOB,A$1400,L$FB

; launch Advanced Demuffin
*800G

[“C” to convert disk]

[S6,D1=original disk (side A)]
[S6,D2=blank formatted disk]
```

```
                --v--
ADVANCED DEMUFFIN 1.5    (C) 1983, 2014
ORIGINAL BY THE STACK    UPDATES BY 4AM
=======PRESS ANY KEY TO CONTINUE=======
TRK:R...R.....................................
+.5:
    0123456789ABCDEF0123456789ABCDEF012
SC0:R...R.....................................
SC1:R...R.....................................
SC2:R...R.....................................
SC3:R...R.....................................
SC4:R...R.....................................
SC5:R...R.....................................
SC6:R...R.....................................
SC7:R...R.....................................
SC8:R...R.....................................
SC9:R...R.....................................
SCA:R...R.....................................
SCB:R...R.....................................
SCC:R...R.....................................
SCD:R...R.....................................
SCE:R...R.....................................
SCF:R...R.....................................
=============================================
16SC $00,$00-$22,$0F BY1.0 S6,D1->S6,D2

                --^--
```

No surprises here. Track 0 is in a standard format (modulo one modified epilogue byte) which is read by the disk firmware at $C600. Track 4 has no sectors. My working theory is that it has some sort of nibble sequence that is difficult to bit copy, but I haven't confirmed that yet.

Side B fares even better:

```
                    --v--

ADVANCED DEMUFFIN 1.5    (C) 1983, 2014
ORIGINAL BY THE STACK    UPDATES BY 4AM
=======PRESS ANY KEY TO CONTINUE=======
TRK:.....................................
+.5:
    0123456789ABCDEF0123456789ABCDEF012
SC0:.....................................
SC1:.....................................
SC2:.....................................
SC3:.....................................
SC4:.....................................
SC5:.....................................
SC6:.....................................
SC7:.....................................
SC8:.....................................
SC9:.....................................
SCA:.....................................
SCB:.....................................
SCC:.....................................
SCD:.....................................
SCE:.....................................
SCF:.....................................
=========================================
16SC $00,$00-$22,$0F BY1.0 S6,D1->S6,D2

                    --^--
```

No errors at all. Hooray!

Using my trusty Disk Fixer sector editor, I manually copied each sector of side A's track 0 to my copy. Nothin' fancy but it works.

Now I have a copy in a standard format, but it still has the original RWTS on it, so it doesn't make it very far on boot. To make my copy be able to read itself, I'll need to patch the RWTS to expect standard prologue and epilogue sequences.

```
; address prologue (read)
T00,S09,$55: 96 -> D5
T00,S09,$5F: FF -> AA

; address epilogue (read)
T00,S09,$91: EE -> DE
T00,S09,$9F: FF -> AA

; data prologue (read)
T00,S08,$E7: 96 -> D5
T00,S08,$F1: FF -> AA
T00,S08,$FC: D7 -> AD

; data epilogue (read)
T00,S09,$35: EE -> DE
T00,S09,$3F: FF -> AA

; address prologue (write)
T00,S0C,$7A: 96 -> D5
T00,S0C,$7F: FF -> AA

; address epilogue (write)
T00,S0C,$AE: EE -> DE
T00,S0C,$B3: FF -> AA

; data prologue (write)
T00,S08,$53: 96 -> D5
T00,S08,$58: FF -> AA
T00,S08,$5D: D7 -> AD

; data epilogue (write)
T00,S08,$9E: EE -> DE
T00,S08,$A3: FF -> AA
```

```
; value used as self-sync nibble
T00,S08,$3E: DF -> FF
T00,S08,$AD: DF -> FF
T00,S0C,$60: DF -> FF
```

Whew. That's a lot to do by hand. Now I appreciate even further the work I've done automating this sort of thing. Sadly, this bootloader is non-standard enough that none of my automated tools help me here.

```
]PR#6
...grinds and crashes...
```

I've missed something.

Chapter 4
Lost In Translation

Two things of note here:

1. I've restored the prologues and epilogues on disk and restored the values that the RWTS checks for, but the disk still can not read itself.

2. I couldn't read the original disk with a sector editor -- even after changing the prologues and epilogues it used.

These two facts are not unrelated.

Beyond the prologues and epilogues, why would an RWTS not be able to read a disk? The next place to look is the nibble translate table, which is the mapping between nibbles-on-disk and bytes-in-memory.

A DOS 3.3-shaped RWTS has two tables, one for translating nibbles to bytes (during read), and another for translating bytes to nibbles (during write).

Without the proper prologues, an RWTS can't read a disk because it can't find the sectors -- either the start of the address field or the data field. Without the proper nibble translate table, an RWTS can't read a disk because it can't interpret the nibbles in the data field.

On a standard disk, these two tables are on T00,S04; on this disk, they're on T00,S0A.

```
------------- DISK EDIT -------------
TRACK $00/SECTOR $0A/VOLUME $FE/BYTE$00
-------------------------------------

; This is code implementing a wait loop
; during track seek. The bytes after
; offset $10 constitute a data table.
$00: A2 11 CA D0 FD E6 46 D0   "QJP}fFP
$08: 02 E6 47 38 E9 01 D0 F0   BfG8iAPp
$10: 60 01 30 28 24 20 1E 1D    A0($ ^]
$18: 1C 1C 1C 1C 1C 70 2C 26   \\\\\0,&
$20: 22 1F 1E 1D 1C 1C 1C 1C   "_^]\\\\
$28: 1C                        \

; This is the Write Translate Table,
; used to convert 6-bit nibbles to
; 8-bit bytes.
$28:    D5 97 9A 9B 9D 9E 9F    U.....
$30: A6 A7 AB AC AD AE AF B2   &'+,-./2
$38: B3 B4 B5 B6 B7 B9 BA BB   345679:;
$40: BC BD BE BF CB CD CE CF   <=>?KMNO
$48: D3 D6 D7 D9 DA DB DC DD   SVWYZ[\]
$50: DE DF E5 E6 E7 E9 EA EB   ^_efgijk
$58: EC ED EE EF F2 F3 F4 F5   lmnorstu
$60: F6 F7 F9 FA FB FC FD FE   vwyz{|}~
$68: AA                        *

; This is unused and/or data tables.
; (Part of it appears to be a physical-
; to-logical sector mapping.)
$68:    FF FF FF FF FF FF FF    .......
$70: FF FF FF FF FF FF FF FF   ........
$78: FF 00 0D 0B 09 07 05 03   .@MKIGEC
$80: 01 0E 0C 0A 08 06 04 02   ANLJHFDB
$88: 0F FF FF FF FF FF FF FF   O.......
$90: FF FF FF FF FF FF         ......
```

```
; This is the Read Translate Table,
; used to convert 8-bit bytes to 6-bit
; nibbles. It's the inverse of the
; previous table that started at offset
; $29. It contains several unused bytes
; because not all values are valid
; nibbles, and it's easier to waste the
; space in the table than do extra math
; while the disk is spinning.
$90:                         96 01        .A
$98: 98 99 02 03 9C 04 05 06   ..BC.DEF
$A0: A0 A1 A2 A3 A4 A5 07 08    !"#$%GH
$A8: A8 A9 3F 09 0A 0B 0C 0D   ()?IJKLM
$B0: B0 B1 0E 0F 10 11 12 13   01NOPQRS
$B8: B8 14 15 16 17 18 19 1A   8TUVWXYZ
$C0: C0 C1 C2 C3 C4 C5 C6 C7   @ABCDEFG
$C8: C8 C9 CA 1B CC 1C 1D 1E   HIJ[L\]^
$D0: D0 D1 D2 1F D4 00 20 21   PQR_T@ !
$D8: D8 22 23 24 25 26 27 28   X"#$%&'(
$E0: E0 E1 E2 E3 E4 29 2A 2B   `abcd)*+
$E8: E8 2C 2D 2E 2F 30 31 32   h,-./012
$F0: F0 F1 33 34 35 36 37 38   pq345678
$F8: F8 39 3A 3B 3C 3D 3E FF   x9:;<=>.
----------------------------------------
BUFFER 0/SLOT 6/DRIVE 1/MASK OFF/NORMAL

----------------------------------------
COMMAND : _
```

--^--

Comparing these tables to a standard DOS 3.3 disk, I found and fixed several differences:

```
T00,S0A,$29: D5 -> 96
T00,S0A,$68: AA -> FF
T00,S0A,$96: 96 -> 00
T00,S0A,$AA: 3F -> AA
T00,S0A,$D5: 00 -> D5
T00,S0A,$FF: FF -> 3F
```

```
]PR#6
...boots, displays loading message
 "SYSTEM // (C) 1985 BY KYLE FREEMAN"
...and reboots
```

This is great progress. The disk can read itself, and it boots far enough to load and execute the code that tells me to go f--- myself.

Let's go find that nibble check.

Chapter 5
Better To Be Lucky Than Good

Since my copy reboots, and programs don't just do that without a good reason, I'm guessing there is a runtime protection check somewhere. One thing that all protection checks have in common is they need to turn on the drive motor by accessing a specific address in the $C0xx range. For slot 6, it's $C0E9, but to allow disks to boot from any slot, developers usually use code like this:

```
LDX <slot number x 16>
LDA $C089,X
```

There's nothing that says you have to use the X-register as the index or the accumulator as the load register. But most RWTS code does, out of convention I suppose (or fear of messing up such low-level code in subtle ways).

Also, since developers don't actually want people finding their protection-related code, they may try to encrypt it or obfuscate it on disk, in memory, or both. But eventually, the code must exist and the code must run, and it must run on my machine, and I have the final say on what my machine does or does not do.

But sometimes you get lucky.

Turning to my trusty Disk Fixer sector editor, I search the non-working copy for "BD 89 C0", which is the opcode sequence for "LDA $C089,X".

```
[Disk Fixer]
  ["F"ind]
    ["H"ex]
      ["BD 89 C0"]

                  --v--

------------ DISK SEARCH ------------

$00/$0D-$2E   $02/$07-$A2

                  --^--
```

The match on track $00 is part of the legitimate RWTS. The match on track $02 is... something extra. Extra is bad.

```
                  --v--

T02,S07
----------- DISASSEMBLY MODE ----------
; This part appears to be a multisector
; read loop, with individual "blocks"
; that read a certain number of sectors
; into consecutive memory starting at a
; given track/sector.

; switch to RAM bank 1 (where the RWTS
; lives)
0000:AD 8B C0       LDA   $C08B
0003:AD 8B C0       LDA   $C08B
```

```
; get block
0006:BD 00 DF        LDA   $DF00,X
0009:85 FE           STA   $FE
000B:A6 FE           LDX   $FE

; exit if first block parameter is 0
000D:BD 10 DF        LDA   $DF10,X
0010:F0 4D           BEQ   $005F

; otherwise store it in the RWTS
; parameter table (starting at $D000,
; so $D004 is the track #)
0012:8D 04 D0        STA   $D004

; sector number
0015:BD 11 DF        LDA   $DF11,X
0018:8D 05 D0        STA   $D005

; sector count (based on code below)
001B:BD 12 DF        LDA   $DF12,X
001E:85 FD           STA   $FD

; target address (high byte)
0020:BD 13 DF        LDA   $DF13,X
0023:8D 09 D0        STA   $D009

; RWTS command ($01=read)
0026:A9 01           LDA   #$01
0028:8D 0C D0        STA   $D00C

; call RWTS
002B:20 18 D0        JSR   $D018

; branch on success
002E:90 10           BCC   $0040
```

```
; very unfriendly! any disk read error
; wipes memory and reboots
0030:A9 A0           LDA   #$A0
0032:9D 00 80        STA   $8000,X
0035:CA              DEX
0036:D0 FA           BNE   $0032
0038:CE 34 D8        DEC   $D834
003B:D0 F5           BNE   $0032
003D:4C 00 C6        JMP   $C600

; execution continues here (from "BCC"
; at offset $002E, above)
; increment target address, decrement
; logical sector number
0040:EE 09 D0        INC   $D009
0043:CE 05 D0        DEC   $D005
0046:10 08           BPL   $0050

; wrap around to track+1, sector=$0F
0048:EE 04 D0        INC   $D004
004B:A9 0F           LDA   #$0F
004D:8D 05 D0        STA   $D005

; decrement sector count and loop to
; read the rest of the sectors in
; this block
0050:C6 FD           DEC   $FD
0052:D0 D7           BNE   $002B

; increment block index
0054:E6 FE           INC   $FE
0056:E6 FE           INC   $FE
0058:E6 FE           INC   $FE
005A:E6 FE           INC   $FE

; loop back for the next block
005C:4C 0B D8        JMP   $D80B
```

--^--

OK, that's all completely legitimate.

I mean, it's just the game loading itself from disk, with a relatively compact representation of what to read and where to put it.

The loop only exits when the first parameter is 0, with the BEQ at offset $10 that branches to offset $005F. So let's continue there.

Chapter 6
Stepper Madness

Continuing the listing at offset $5F:

--v--

```
; track $04 (aha!)
005F:A9 04        LDA   #$04
0061:8D 04 D0     STA   $D004

; RWTS command $00=seek
0064:A9 00        LDA   #$00
0066:8D 0C D0     STA   $D00C

; seek to track $04
0069:20 18 D0     JSR   $D018

; set up death counter
006C:A9 04        LDA   #$04
006E:85 FD        STA   $FD

; if death counter hits 0, branch to
; The Badlands (listed above, which
; wipes memory and reboots)
0070:C6 FD        DEC   $FD
0072:F0 BC        BEQ   $0030

; more on this later
0074:20 99 D8     JSR   $D899

; zp$06 needs to be zero, apparently,
; otherwise it's OFF TO THE BADLANDS
0077:A5 06        LDA   $06
0079:D0 B5        BNE   $0030

; zp$07 and zp$08 also need to be zero,
; otherwise we branch back to decrement
; the death counter and try again
007B:A5 07        LDA   $07
007D:D0 F1        BNE   $0070
007F:A5 08        LDA   $08
0081:D0 ED        BNE   $0070
```

```
; decrement the block index (used in
; the legitimate read loop above)
0083:C6 FE          DEC   $FE
0085:C6 FE          DEC   $FE
0087:C6 FE          DEC   $FE
0089:C6 FE          DEC   $FE
008B:A6 FE          LDX   $FE

; get the starting memory address of
; the last block we read from disk
008D:BD 13 DF       LDA   $DF13,X
0090:85 01          STA   $01
0092:A9 00          LDA   #$00
0094:85 00          STA   $00

; jump there to start the game
0096:6C 00 00       JMP   ($0000)
```

--^--

OK, so it's really important that the subroutine at $D899 sets some zero page locations properly. I mean, it's not important to me, per se, but the game loader considers it VITALLY important. Different priorities, I guess.

Note: this sector is loaded at $D800, based on the "JSR $D899" and the self-modifying "DEC $D834" in The Badlands memory zappy loopy thing.

--v--

```
; get slot number from RWTS parameter
; table
0099:AE 01 D0       LDX   $D001
009C:8A             TXA
```

```
; munge it and store it in a later
; routine (used as a stepper motor
; controller)
009D:09 80          ORA   #$80
009F:8D 48 D9       STA   $D948

; turn on drive motor manually (this is
; how I found all this code in the
; first place!)
00A2:BD 89 C0       LDA   $C089,X
00A5:A9 00          LDA   #$00
00A7:85 06          STA   $06

; reset data latch
00A9:BD 8E C0       LDA   $C08E,X

; self-modify code below
00AC:A9 D5          LDA   #$D5
00AE:8D EC D8       STA   $D8EC
00B1:20 D5 D8       JSR   $D8D5

This is $D8D5:

; set up counter
00D5:A0 00          LDY   #$00
00D7:84 26          STY   $26
00D9:C8             INY
00DA:D0 07          BNE   $00E3
00DC:E6 26          INC   $26
00DE:D0 F9          BNE   $00D9
00E0:4C 06 D9       JMP   $D906

; reset data latch
00E3:BD 8E C0       LDA   $C08E,X
```

```
; look for nibble sequence "D5 96"
; (NOTE: the first nibble value was
; modified immediately before calling!)
00E6:BD 8C C0        LDA   $C08C,X
00E9:10 FB           BPL   $00E6
00EB:C9 D5           CMP   #$D5
00ED:D0 EA           BNE   $00D9
00EF:BD 8C C0        LDA   $C08C,X
00F2:10 FB           BPL   $00EF
00F4:C9 96           CMP   #$96
00F6:D0 F3           BNE   $00EB
00F8:60              RTS

Continuing from $D8B4...

00B4:20 16 D9        JSR   $D916

Here is $D916:

; engage stepper motors in a specific
; pattern ($D947 engages $C080,X,Y,
; where X is slot x16 and Y varies):
; PHASE 0 ON
; PHASE 0 OFF
; PHASE 1 OFF
; PHASE 0 ON
; PHASE 1 ON
; PHASE 1 OFF
0016:A0 01           LDY   #$01
0018:20 47 D9        JSR   $D947
001B:88              DEY
001C:20 47 D9        JSR   $D947
001F:C8              INY
0020:C8              INY
0021:C0 04           CPY   #$04
0023:D0 F3           BNE   $0018
0025:4C 4A D9        JMP   $D94A
```

```
Continuing from $D8B7:

; search for another nibble sequence
00B7:A9 9A          LDA   #$9A
00B9:8D EC D8       STA   $D8EC
00BC:20 D5 D8       JSR   $D8D5

; store result (how long it took to
; find the nibble sequence) in zp$07
00BF:A5 26          LDA   $26
00C1:85 07          STA   $07

; and more stepper madness
00C3:20 35 D9       JSR   $D935

Here is $D935:

; engage stepper motors in a specific
; pattern (but different this time):
; PHASE 2 ON
; PHASE 2 OFF
; PHASE 3 OFF
; PHASE 2 ON
; PHASE 3 ON
; PHASE 3 OFF
0035:A0 05          LDY   #$05
0037:20 47 D9       JSR   $D947
003A:88             DEY
003B:20 47 D9       JSR   $D947
003E:C8             INY
003F:C8             INY
0040:C0 08          CPY   #$08
0042:D0 F3          BNE   $0037
0044:4C 4A D9       JMP   $D94A
```

```
Here is $D947:

; (NOTE: this softswitch was modified
; earlier, at $D89F, based on the slot)
0047:B9 80 C0        LDA   $C080,Y
004A:A9 20           LDA   #$20
004C:85 08           STA   $08
004E:A9 20           LDA   #$20
0050:38              SEC
0051:E9 01           SBC   #$01
0053:D0 FB           BNE   $0050
0055:C6 08           DEC   $08
0057:D0 F5           BNE   $004E
0059:60              RTS

Continuing from $D8C6:

; modify the nibble search a third time
00C6:A9 DF           LDA   #$DF
00C8:8D EC D8        STA   $D8EC

; and search for the nibble sequence
00CB:20 D5 D8        JSR   $D8D5

; save the result in zp$08
00CE:A5 26           LDA   $26
00D0:85 08           STA   $08
00D2:4C F9 D8        JMP   $D8F9
...
00F9:20 28 D9        JSR   $D928
00FC:20 09 D9        JSR   $D909
00FF:BD 88 C0        LDA   $C088,X
0002:BD 8E C0        LDA   $C08E,X
0005:60              RTS
```

Here is $D928:

```
; engage stepper motors in this pattern
; PHASE 3 ON
; PHASE 3 OFF
; PHASE 2 ON
; PHASE 2 OFF
0028:A0 07          LDY   #$07
002A:20 47 D9       JSR   $D947
002D:88             DEY
002E:C0 03          CPY   #$03
0030:D0 F8          BNE   $002A
0032:4C 4A D9       JMP   $D94A
```

Here is $D909:

```
; engage stepper motors in this pattern
; PHASE 1 ON
; PHASE 1 OFF
; PHASE 0 ON
; PHASE 0 OFF
0009:A0 03          LDY   #$03
000B:20 47 D9       JSR   $D947
000E:88             DEY
000F:C0 FF          CPY   #$FF
0011:D0 F8          BNE   $000B
0013:4C 4A D9       JMP   $D94A
```

--^--

That's all completely insane. And, for my purposes, unnecessary. The only side effect of the protection check at $D85F is to set a few zero page values which are immediately checked (at $D877, $D87B, and $D87F respectively).

Thus, scrolling all the way back to $D810, where we exit the disk read loop by branching to $D85F... I can change that "BEQ" instruction to branch to $D883, after the protection check ends and the legitimate game loader resumes.

```
T02,S07,$11: 4D -> 71

]PR#6
...works...
```

Quod erat liberandum.