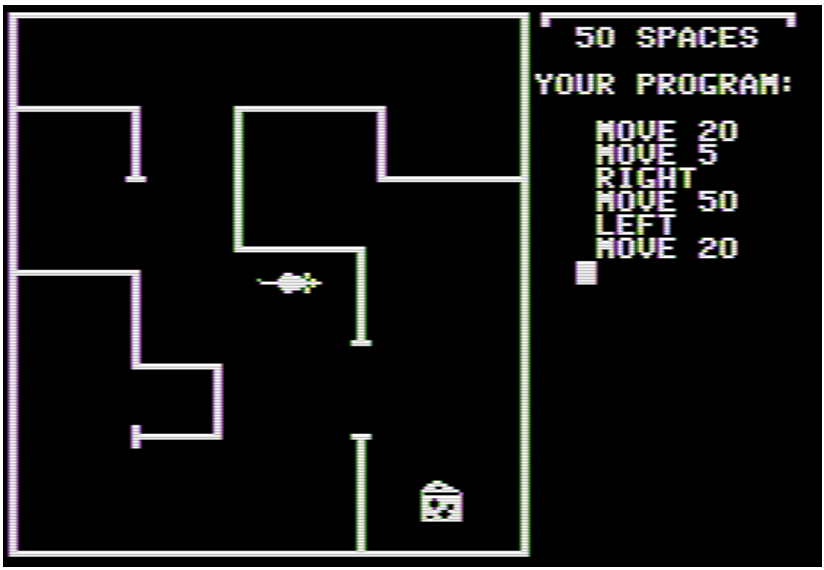
Algernon



2015-03-16

4am
to deprotect
and preserve

Contents

----------Algernon----------
A 4am crack 2015-03-16

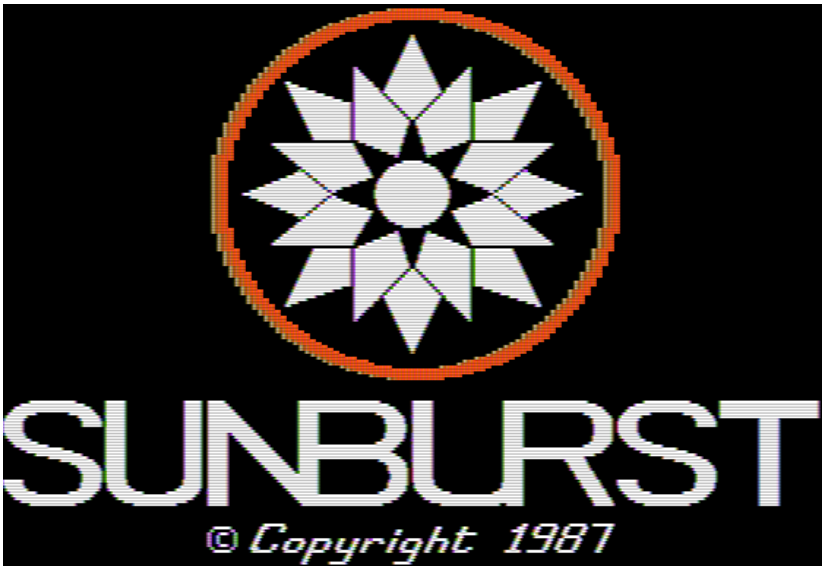
Name: Algernon
Genre: educational
Year: 1987
Credits:
  - Designed by Tom Bretl
  - Programmed by Larry Bank
Publisher: Sunburst Communications
Media: single-sided 5.25-inch floppy
OS: ProDOS 1.1.1
Other versions: none (preserved here for the first time)
Identical cracks: Now You See It, Now You Don't: Was It There? Was It Missing? (4am crack no. 262)

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)
  no read errors, but copy loads ProDOS
  title screen, then reboots



Copy][+ nibble editor
  T00 has at least a few sectors, but
    I'm not sure how many
  T01+ have no visible structure at all

--v--

```
   COPY ][ PLUS BIT COPY PROGRAM 8.4
(C) 1982-9 CENTRAL POINT SOFTWARE, INC.
----------------------------------------

TRACK: 03  START: 38C4  LENGTH: 1109

38B8: B4 DA DF BF BE AA D5 FF   VIEW
38C0: FF FF FF FF BF E4 CF D3
38C8: F3 99 E6 99 E6 99 E6 99
38D0: E6 99 E6 99 E6 99 E6 99
38D8: E6 99 CA AA A9 9B 9A 97
38E0: 96 AA D5 FC 99 E6 99 E6
38E8: 99 E6 99 E6 99 E6 99 E6
38F0: CA D5 A9 AC F5 CE F7 9B
38F8: BE BE EE AD B9 E9 96 96

----------------------------------------

  A  TO ANALYZE DATA  ESC TO QUIT

  ?  FOR HELP SCREEN  /  CHANGE PARMS

  Q  FOR NEXT TRACK   SPACE TO RE-READ
```

--^--

Disk Fixer
  [“0” -> “Input/Output Control”]
    “CHECKSUM ENABLED” -> “NO”
  T00,S00 readable
  T00,S0D readable
  T00,S0E readable
  nothing else

Why didn't COPYA work?
  not a 16-sector disk (or maybe a
   wildly non-standard one)

Why didn't Locksmith FDB work?
  ditto

Why didn't my EDD copy work?
  I don't know. Probably a nibble check
  in the first .SYSTEM file (assuming
  this is really ProDOS as it claims).

Converting the disk to a standard format will be a challenge. Advanced Demuffin requires a DOS 3.3-shaped RWTS, but this disk uses ProDOS (as far as I can tell). Assuming the disk even uses 16 sectors (and Copy][+ just can't see the structure), I might be able to extract the RWTS from the PRODOS file and build an RWTS to plug into Advanced Demuffin. I've done that successfully before, but it's finicky. DOS 3.3 and ProDOS are very different beasts.

Next steps:

1. Boot trace to capture PRODOS file in memory
2. Extract its RWTS routines to build a DOS 3.3-shaped RWTS file
3. Convert the disk to a standard format with Advanced Demuffin
4. Patch the bootloader and/or the PRODOS file to be able to read a standard format disk
5. Find and bypass the nibble check

Chapter 1
In Which We Take Our First Steps

```
[S6,D1=original disk]
[S5,D1=my work disk]

]PR#5
CAPTURING BOOT0
...reboots slot 6...
...reboots slot 5...
SAVING BOOT0

]CALL -151
*800<2800.28FFM
*801L

; set up $801 with an "RTS" (probably
; so we can JSR $C65C later to read
; sectors)
0801-   A9 60       LDA   #$60
0803-   8D 01 08    STA   $0801

; save slot (x16)
0806-   86 43       STX   $43
0808-   86 2B       STX   $2B

; munge slot into $C6 form and store it
080A-   8A          TXA
080B-   4A          LSR
080C-   4A          LSR
080D-   4A          LSR
080E-   4A          LSR
080F-   09 C0       ORA   #$C0
0811-   8D 38 08    STA   $0838
```

```
; set reset vector
0814-   A0 CE       LDY   #$CE
0816-   A9 08       LDA   #$08
0818-   8C F2 03    STY   $03F2
081B-   8D F3 03    STA   $03F3
081E-   A9 AD       LDA   #$AD
0820-   8D F4 03    STA   $03F4

0823-   A9 00       LDA   #$00
0825-   85 09       STA   $09
0827-   85 03       STA   $03

; increment physical sector number
0829-   E6 3D       INC   $3D

; read a sector
082B-   20 36 08    JSR   $0836

; decrement sector count
082E-   CE 39 08    DEC   $0839

; loop back to read more sectors
0831-   D0 F6       BNE   $0829

; or continue down below
0833-   4C 40 08    JMP   $0840
0836-   4C 5C C6    JMP   $C65C
0839-  [02]
083A-  ["PRODOS"]

; execution continues here after all
; sectors are read
0840-   A9 02       LDA   #$02
0842-   85 02       STA   $02
0844-   A9 0C       LDA   #$0C
0846-   85 27       STA   $27
```

```
; don't know what this does yet
0848-   20 34 09    JSR   $0934
```

This is where I need to interrupt the boot, to see what ends up at $900 (and $A00) from initial sector read loop.

```
*9600<C600.C6FFM

; set up callback at $0840 after sector
; read loop exits
96F8-   A9 4C       LDA   #$4C
96FA-   8D 40 08    STA   $0840
96FD-   A9 0A       LDA   #$0A
96FF-   8D 41 08    STA   $0841
9702-   A9 97       LDA   #$97
9704-   8D 42 08    STA   $0842

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

; callback is here -- copy 3 pages to
; graphics page so they survive a
; reboot
970A-   A2 03       LDX   #$03
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 TRACE0,A$9600,L$126
*9600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT0 0800-0AFF,A$2800,L$300
]CALL -151
*800<2800.2AFFM
*934L
.
. actually boring, it's just setting up
. a write translate table
.

Continuing at $084B...

084B-   20 EF 08    JSR   $08EF

*8EFL
.
. appears to read a sector into ($26)
.

Continuing at $084E...
```

```
; look for PRODOS file in disk catalog
; (string at $0839 spells "PRODOS")
084E-   A9 04       LDA   #$04
0850-   85 00       STA   $00
0852-   A9 0C       LDA   #$0C
0854-   85 01       STA   $01
0856-   A5 00       LDA   $00
0858-   18          CLC
0859-   69 27       ADC   #$27
085B-   85 00       STA   $00
085D-   B0 6F       BCS   $08CE
085F-   A0 06       LDY   #$06
0861-   B1 00       LDA   ($00),Y
0863-   D9 39 08    CMP   $0839,Y
0866-   D0 EE       BNE   $0856
0868-   88          DEY
0869-   D0 F6       BNE   $0861

; get block info for PRODOS file
086B-   B1 00       LDA   ($00),Y
086D-   48          PHA
086E-   A0 10       LDY   #$10
0870-   B1 00       LDA   ($00),Y
0872-   C9 FF       CMP   #$FF
0874-   D0 58       BNE   $08CE
0876-   C8          INY
0877-   B1 00       LDA   ($00),Y
0879-   85 02       STA   $02
087B-   C8          INY
087C-   B1 00       LDA   ($00),Y
087E-   85 03       STA   $03
0880-   A9 FF       LDA   #$FF
0882-   85 07       STA   $07
0884-   A9 00       LDA   #$00
0886-   85 26       STA   $26
0888-   85 00       STA   $00
088A-   A0 1E       LDY   #$1E
088C-   68          PLA
088D-   C9 26       CMP   #$26
088F-   F0 0B       BEQ   $089C
```

```
; load data at $2000
0891-   A0 20       LDY   #$20
0893-   84 27       STY   $27

; read it
0895-   20 EF 08    JSR   $08EF

; carry clear on success
0898-   90 1F       BCC   $08B9

; carry set on failure
089A-   B0 32       BCS   $08CE

; loop to read the rest of the file
089C-   84 27       STY   $27
089E-   84 01       STY   $01
08A0-   20 EF 08    JSR   $08EF
08A3-   B0 29       BCS   $08CE
08A5-   E6 07       INC   $07
08A7-   A4 07       LDY   $07
08A9-   B1 00       LDA   ($00),Y
08AB-   85 02       STA   $02
08AD-   E6 01       INC   $01
08AF-   B1 00       LDA   ($00),Y
08B1-   85 03       STA   $03
08B3-   C6 01       DEC   $01
08B5-   11 00       ORA   ($00),Y
08B7-   D0 E7       BNE   $08A0

; execution continues here after all
; sectors are read successfully --
; turn off drive motor
08B9-   BD 88 C0    LDA   $C088,X
```

```
; set reset vector
08BC-   A2 00       LDX   #$00
08BE-   A0 57       LDY   #$57
08C0-   A9 F2       LDA   #$F2
08C2-   8E F2 03    STX   $03F2
08C5-   8C F3 03    STY   $03F3
08C8-   8D F4 03    STA   $03F4

; jump to the beginning of PRODOS file
08CB-   4C 00 20    JMP   $2000

; any failure ends up here
08CE-   E6 27       INC   $27
08D0-   A0 00       LDY   #$00
08D2-   A6 2B       LDX   $2B

; turn off drive motor
08D4-   BD 88 C0    LDA   $C088,X

; wipe memory and never return
08D7-   20 DD 08    JSR   $08DD
08DA-   4C DA 08    JMP   $08DA
08DD-   A9 60       LDA   #$60
08DF-   91 26       STA   ($26),Y
08E1-   99 00 09    STA   $0900,Y
08E4-   99 00 0A    STA   $0A00,Y
08E7-   88          DEY
08E8-   D0 F5       BNE   $08DF
08EA-   C6 27       DEC   $27
08EC-   4C DF 08    JMP   $08DF
```

So, it really is loading ProDOS, albeit in its own special way. $08CB jumps to the code loaded from the PRODOS file at $2000.

Chapter 2
In Which We Take Two Steps Forward, One Step Back, One Twist Sideways, And Worry That We Just Created A New Dance Move

```
*9600<C600.C6FFM

; save all flags and registers
96F8-   08          PHP
96F9-   48          PHA
96FA-   8A          TXA
96FB-   48          PHA
96FC-   98          TYA
96FD-   48          PHA

; fill $2000..$95FF with "FD" bytes so
; I can tell how big the PRODOS file is
; later
96FE-   A2 76       LDX   #$76
9700-   A0 00       LDY   #$00
9702-   A9 FD       LDA   #$FD
9704-   99 00 20    STA   $2000,Y
9707-   C8          INY
9708-   D0 FA       BNE   $9704
970A-   EE 06 97    INC   $9706
970D-   CA          DEX
970E-   D0 F4       BNE   $9704

; set up the callback after PRODOS file
; is loaded
9710-   A9 23       LDA   #$23
9712-   8D CC 08    STA   $08CC
9715-   A9 97       LDA   #$97
9717-   8D CD 08    STA   $08CD

; restore registers and flags
971A-   68          PLA
971B-   A8          TAY
971C-   68          PLA
971D-   AA          TAX
971E-   68          PLA
971F-   28          PLP

; start the boot
9720-   4C 01 08    JMP   $0801
```

```
; turn off slot 6 drive motor
9723-   AD E8 C0    LDA   $C0E8

; reboot to my work disk (PRODOS file
; loads at $2000, so no relocation is
; necessary)
9726-   4C 00 C5    JMP   $C500

*BSAVE TRACE.PRODOS,A$9600,L$129

*BRUN TRACE.PRODOS
...reboots slot 6...
...reboots slot 5...

]CALL -151

[perusing memory, starting at $2000]
```

It looks like $5A00 is the first page that still has repeated $FD bytes.

```
*5A-20
=3A
*BSAVE BOOT1.PRODOS,A$2000,L$3A00
```

Scanning through memory, I found the RWTS code. (Sorry, no magic here. It can be in a number of places, depending on the version of ProDOS. And this is *not* a standard version of ProDOS.)

The RWTS is... odd. Here, for example, is the routine that reads the address field (relocated into the language card at $D316 by the time it's used):

```
*5316L

5316-   A6 3E       LDX   $3E
5318-   A0 03       LDY   #$03
531A-   8C E8 D4    STY   $D4E8
531D-   8C F0 D4    STY   $D4F0
5320-   84 3F       STY   $3F
5322-   A0 02       LDY   #$02
5324-   CE F0 D4    DEC   $D4F0
5327-   D0 05       BNE   $532E
5329-   CE E8 D4    DEC   $D4E8
532C-   F0 3A       BEQ   $5368

; loop will match "CA AA A9" (that's
; what's at $D4D0, a.k.a. $54D0)
532E-   BD 8C C0    LDA   $C08C,X
5331-   10 FB       BPL   $532E
5333-   D9 D0 D4    CMP   $D4D0,Y
5336-   D0 EA       BNE   $5322
5338-   88          DEY
5339-   10 F3       BPL   $532E

; now decode the actual address field
533B-   A9 00       LDA   #$00
533D-   F0 03       BEQ   $5342
533F-   99 A7 D4    STA   $D4A7,Y

; read only one nibble
5342-   BC 8C C0    LDY   $C08C,X
5345-   10 FB       BPL   $5342

; Address field is not 4-4 encoded! The
; values are progressively decrypted
; against the write translate table at
; $D496 (originally $5496). I won't
; show it here, but this table is also
; non-standard; it looks nothing like
; the one in "Beneath Apple DOS".
5347-   59 00 D4    EOR   $D400,Y
534A-   A4 3F       LDY   $3F
534C-   C6 3F       DEC   $3F
```

```
; branch back to store this address
; field data in $D4A7,Y
534E-   10 EF        BPL   $533F

; 4th nibble is the checksum, I think
5350-   A8           TAY
5351-   D0 15        BNE   $5368

; then match "AA" as an address field
; epilogue
5353-   BD 8C C0     LDA   $C08C,X
5356-   10 FB        BPL   $5353
5358-   C9 AA        CMP   #$AA
535A-   D0 0C        BNE   $5368

; munge one of the address field values
; into the Y register on exit (not sure
; which one)
535C-   AD A9 D4     LDA   $D4A9
535F-   4A           LSR
5360-   4A           LSR
5361-   A8           TAY

; munge another value into accumulator
5362-   AD AA D4     LDA   $D4AA
5365-   4A           LSR
5366-   4A           LSR
5367-   60           RTS
5368-   38           SEC
5369-   60           RTS
```

And here is the heart of the RWTS, the routine that matches the data prologue then converts the data field of nibbles to bytes in memory:

```
*536AL

; set up slot-specific addresses for
; reading the data latch (normal)
536A-   8A          TXA
536B-   09 8C       ORA   #$8C
536D-   8D B0 D3    STA   $D3B0
5370-   8D BD D3    STA   $D3BD
5373-   8D CF D3    STA   $D3CF
5376-   8D E1 D3    STA   $D3E1
5379-   8D F6 D3    STA   $D3F6
537C-   A0 20       LDY   #$20
537E-   88          DEY
537F-   30 E7       BMI   $5368

; match "CA D5 A9" data prologue
5381-   BD 8C C0    LDA   $C08C,X
5384-   10 FB       BPL   $5381
5386-   C9 CA       CMP   #$CA
5388-   D0 F4       BNE   $537E
538A-   2C A8 D4    BIT   $D4A8    ; odd
538D-   BD 8C C0    LDA   $C08C,X
5390-   10 FB       BPL   $538D
5392-   C9 D5       CMP   #$D5
5394-   D0 F0       BNE   $5386
5396-   50 03       BVC   $539B    ; odd
5398-   BD 8D C0    LDA   $C08D,X ; odd
539B-   BD 8C C0    LDA   $C08C,X
539E-   10 FB       BPL   $539B
53A0-   C9 A9       CMP   #$A9
53A2-   F0 07       BEQ   $53AB
53A4-   88          DEY
53A5-   10 F4       BPL   $539B
53A7-   D0 BF       BNE   $5368
53A9-   30 7E       BMI   $5429
```

```
; convert nibbles to bytes
53AB-   A0 54       LDY   #$54
53AD-   A9 00       LDA   #$00
53AF-   AE 8C C0    LDX   $C08C
53B2-   10 FB       BPL   $53AF
53B4-   5D 00 D4    EOR   $D400,X
53B7-   84 3F       STY   $3F
53B9-   29 FC       AND   #$FC
53BB-   AA          TAX
53BC-   AC 8C C0    LDY   $C08C
53BF-   10 FB       BPL   $53BC
53C1-   59 00 D4    EOR   $D400,Y
53C4-   29 FC       AND   #$FC
53C6-   1D 02 D5    ORA   $D502,X
53C9-   A4 3F       LDY   $3F
53CB-   99 00 D6    STA   $D600,Y
53CE-   AC 8C C0    LDY   $C08C
53D1-   10 FB       BPL   $53CE
53D3-   59 00 D4    EOR   $D400,Y
53D6-   29 FC       AND   #$FC
53D8-   1D 01 D5    ORA   $D501,X
53DB-   A4 3F       LDY   $3F
53DD-   99 55 D6    STA   $D655,Y
53E0-   AC 8C C0    LDY   $C08C
53E3-   10 FB       BPL   $53E0
53E5-   59 00 D4    EOR   $D400,Y
53E8-   29 FC       AND   #$FC
53EA-   1D 00 D5    ORA   $D500,X
53ED-   A4 3F       LDY   $3F
53EF-   99 AA D6    STA   $D6AA,Y
53F2-   88          DEY
53F3-   10 BA       BPL   $53AF
53F5-   AC 8C C0    LDY   $C08C
53F8-   10 FB       BPL   $53F5
[...]
```

```
53FA-   59 00 D4    EOR   $D400,Y
53FD-   29 FC       AND   #$FC
53FF-   85 3F       STA   $3F
5401-   4A          LSR
5402-   4A          LSR
5403-   05 3F       ORA   $3F
5405-   A6 3E       LDX   $3E
5407-   BC 8C C0    LDY   $C08C,X
540A-   10 FB       BPL   $5407
540C-   59 00 D4    EOR   $D400,Y
540F-   A0 FF       LDY   #$FF
5411-   99 00 D6    STA   $D600,Y

; appears to be a checksum byte
; (branches to "SEC" if it fails)
5414-   BC 8C C0    LDY   $C08C,X
5417-   10 FB       BPL   $5414
5419-   59 00 D4    EOR   $D400,Y
541C-   29 FC       AND   #$FC
541E-   D0 09       BNE   $5429

; match "AA" for data epilogue
5420-   BD 8C C0    LDA   $C08C,X
5423-   10 FB       BPL   $5420
5425-   C9 AA       CMP   #$AA
5427-   F0 16       BEQ   $543F
5429-   38          SEC
542A-   60          RTS
...
543F-   18          CLC
5440-   60          RTS
```

I won't do a side-by-side comparison, but this is all completely different from standard ProDOS. I don't just mean the address prologue. Even the routine that converts nibbles to bytes is different.

Now what I saw in the nibble editor makes slightly more sense:

```
                    --v--

   COPY ][ PLUS BIT COPY PROGRAM 8.4
(C) 1982-9 CENTRAL POINT SOFTWARE, INC.
----------------------------------------

TRACK: 03  START: 38C4  LENGTH: 1109

38B8: B4 DA DF BF BE AA D5 FF   VIEW
38C0: FF FF FF FF BF E4 CF D3
38C8: F3 99 E6 99 E6 99 E6 99
38D0: E6 99 E6 99 E6 99 E6 99
38D8: E6 99 CA AA A9 9B 9A 97
            ^^^^^^^^ ^^^^^^^^ address
    address prologue           field

38E0: 96 AA D5 FC 99 E6 99 E6
      ^^ ^^
   field/epilogue

38E8: 99 E6 99 E6 99 E6 99 E6
38F0: CA D5 A9 AC F5 CE F7 9B
      ^^^^^^^^ ^^^^^^^^^^^^^^ encrypted
 data prologue               data field

38F8: BE BE EE AD B9 E9 96 96

----------------------------------------

  A  TO ANALYZE DATA  ESC TO QUIT

  ?  FOR HELP SCREEN  /  CHANGE PARMS

  Q  FOR NEXT TRACK   SPACE TO RE-READ

                    --^--
```

Porting this to a DOS 3.3-shaped RWTS is not going to be feasible. This is not just a matter of different prologue or epilogue bytes; the entire nibble-to-byte conversion scheme has been rewritten. I need to rethink my next steps.

This disk is, at some level, "ProDOS." It has a custom bootloader to find and load the PRODOS file into memory; it has what appears to be an entirely rewritten floppy device driver; it also has a nibble check somewhere that I haven't even found yet. But it's not entirely custom. About 90% of this file is identical to ProDOS (version 1.1.1, as advertised on the standard ProDOS title screen that it displays during boot).

If I could somehow inject a clean, working version of BASIC.SYSTEM and get this PRODOS file to load that instead of the original program (but without auto-loading the STARTUP file), I could theoretically see the files and copy them off to a standard disk. Or maybe I could write a poor-man's Advanced Demuffin to read the disk sector by sector (well, block by block since it's ProDOS), then save the sector data and recreate the disk with a standard RWTS.

Next steps:

1. Trace PRODOS file
2. Inject clean BASIC.SYSTEM into memory
3. Copy each file off the disk

Chapter 3
In Which We Try Variations Of Our Dance Move And Also Our RWTS

```
[S7,D1=ProDOS hard drive, "A4AMCRACK"]

[Copy ][+ 8.4]
  --> COPY
    --> FILE
      --> from SLOT 7, DRIVE 1
      -->   to SLOT 5, DRIVE 1
        --> BASIC.SYSTEM

OK, now I have a clean copy of the
ProDOS BASIC.SYSTEM file on my DOS 3.3-
based work disk. I'll get back to that.

]PR#5
]BLOAD BOOT1.PRODOS,A$2000
]CALL -151
*2000L
.
. nothing unusual, until...
.
; set up to read block 2 into $0C00
; (this is the ProDOS disk catalog)
218F-   A2 00       LDX   #$00
2191-   86 14       STX   $14
2193-   A0 02       LDY   #$02
2195-   A9 0C       LDA   #$0C
2197-   85 15       STA   $15
2199-   8D 07 22    STA   $2207
219C-   8C 08 22    STY   $2208
219F-   8E 09 22    STX   $2209

; raw disk read (MLI $80)
21A2-   20 00 BF    JSR   $BF00
21A5-  [80 04 22]

; on failure, jump to The Badlands
21A8-   D0 19       BNE   $21C3
```

```
; check if we've read all the blocks of
; the disk catalog into memory
21AA-   A0 03       LDY   #$03
21AC-   B1 14       LDA   ($14),Y
21AE-   AA          TAX
21AF-   88          DEY
21B0-   11 14       ORA   ($14),Y
21B2-   F0 0C       BEQ   $21C0
21B4-   B1 14       LDA   ($14),Y
21B6-   A8          TAY
21B7-   A5 15       LDA   $15
21B9-   18          CLC
21BA-   69 02       ADC   #$02
21BC-   C9 14       CMP   #$14
21BE-   90 D7       BCC   $2197

; success path continues at $5800
21C0-   4C 00 58    JMP   $5800

; failure path ends up here
21C3-   4C 00 57    JMP   $5700

*5700L

; relocate this to $0800
5700-   A2 80       LDX   #$80
5702-   BD 0E 57    LDA   $570E,X
5705-   9D 00 08    STA   $0800,X
5708-   CA          DEX
5709-   10 F7       BPL   $5702

; and jump there
570B-   4C 00 08    JMP   $0800
```

```
; wipe all memory
570E-   2C 89 C0    BIT   $C089
5711-   2C 89 C0    BIT   $C089
5714-   A2 1F       LDX   #$1F
5716-   A0 00       LDY   #$00
5718-   99 00 09    STA   $0900,Y
571B-   99 00 20    STA   $2000,Y
571E-   99 00 40    STA   $4000,Y
5721-   99 00 60    STA   $6000,Y
5724-   99 00 80    STA   $8000,Y
5727-   99 00 A0    STA   $A000,Y
572A-   99 00 D0    STA   $D000,Y

; and make a sound while doing it
572D-   AD 30 C0    LDA   $C030
5730-   88          DEY
5731-   D0 E5       BNE   $5718
5733-   EE 0C 08    INC   $080C
5736-   EE 0F 08    INC   $080F
5739-   EE 12 08    INC   $0812
573C-   EE 15 08    INC   $0815
573F-   EE 18 08    INC   $0818
5742-   EE 1B 08    INC   $081B
5745-   EE 1E 08    INC   $081E
5748-   CA          DEX
5749-   10 CD       BPL   $5718
574B-   8D F2 03    STA   $03F2
574E-   8D F3 03    STA   $03F3
5751-   2C 8A C0    BIT   $C08A

; and reboot
5754-   6C FC FF    JMP   ($FFFC)
```

Well, let's try not to end up there!

If we read the catalog successfully,
execution continues at $5800.

```
*5800L

5800-   A2 4B       LDX   #$4B
5802-   86 02       STX   $02
5804-   2C 81 C0    BIT   $C081
5807-   2C 81 C0    BIT   $C081
580A-   A9 D1       LDA   #$D1
580C-   8D 04 D1    STA   $D104

; set reset vector
580F-   A2 F6       LDX   #$F6
5811-   A0 BF       LDY   #$BF
5813-   A9 1A       LDA   #$1A
5815-   8E F2 03    STX   $03F2
5818-   8C F3 03    STY   $03F3
581B-   8D F4 03    STA   $03F4

; reset drive heads
581E-   A5 43       LDA   $43
5820-   29 70       AND   #$70
5822-   85 3E       STA   $3E
5824-   AA          TAX
5825-   BD 80 C0    LDA   $C080,X
5828-   BD 82 C0    LDA   $C082,X
582B-   BD 84 C0    LDA   $C084,X
582E-   BD 86 C0    LDA   $C086,X

; then turn on drive motor manually
; (suspicious)
5831-   BD 89 C0    LDA   $C089,X
5834-   24 43       BIT   $43
5836-   10 01       BPL   $5839
5838-   E8          INX
5839-   BD 8A C0    LDA   $C08A,X
```

```
; wait loop ($58A5 is just an RTS)
583C-   A9 00       LDA   #$00
583E-   AA          TAX
583F-   A8          TAY
5840-   20 A5 58    JSR   $58A5
5843-   88          DEY
5844-   D0 FA       BNE   $5840
5846-   CA          DEX
5847-   D0 F7       BNE   $5840

; an address pointer maybe?
5849-   85 44       STA   $44
584B-   A9 14       LDA   #$14
584D-   85 45       STA   $45

; read/write access to RAM bank 1
584F-   2C 8B C0    BIT   $C08B
5852-   2C 8B C0    BIT   $C08B

; don't know what this does yet
5855-   20 03 D0    JSR   $D003
5858-   A2 03       LDX   #$03
585A-   86 00       STX   $00
585C-   86 01       STX   $01
585E-   A2 15       LDX   #$15
5860-   86 03       STX   $03
5862-   C6 03       DEC   $03
5864-   30 12       BMI   $5878

; nor this
5866-   20 0C D0    JSR   $D00C
5869-   B0 F7       BCS   $5862
586B-   C0 06       CPY   #$06
586D-   D0 F3       BNE   $5862
```

```
; nor any of this
586F-   20 0F D0    JSR   $D00F
5872-   90 19       BCC   $588D
5874-   C6 01       DEC   $01
5876-   10 E6       BPL   $585E
5878-   A6 02       LDX   $02
587A-   30 26       BMI   $58A2
587C-   A0 12       LDY   #$12
587E-   BD A6 58    LDA   $58A6,X
5881-   99 96 D3    STA   $D396,Y
5884-   CA          DEX
5885-   88          DEY
5886-   10 F6       BPL   $587E
5888-   86 02       STX   $02
588A-   4C 58 58    JMP   $5858
588D-   C6 00       DEC   $00
588F-   10 CD       BPL   $585E
5891-   A5 01       LDA   $01
5893-   C9 03       CMP   #$03
5895-   D0 E1       BNE   $5878

; success path falls through to here
; (I think)
5897-   A6 3E       LDX   $3E

; turn off drive motor
5899-   BD 88 C0    LDA   $C088,X

; switch to ROM
589C-   2C 8A C0    BIT   $C08A

; continue with "stage 2" loader (to
; launch .SYSTEM file, probably)
589F-   4C 00 08    JMP   $0800
```

```
; failure path ends up here
58A2-   4C F6 BF    JMP   $BFF6

*BFF6L

BFF6-   2C 80 C0    BIT   $C080
BFF9-   4C 00 D1    JMP   $D100
```

I'm guessing that $D100 ends up executing the code that started out at $5700, a.k.a. The Badlands.

By the time execution reaches $589F (the success path), ProDOS has done everything it needs to do by relocating itself into the language card, and it's time to find the first .SYSTEM file and load it. But it needs to load the file at $2000, so ProDOS moves its "stage 2" code to $800 to avoid memory conflicts.

Oh, and it's modified the RWTS in memory a number of times. How many? I'm not sure yet.

I need to interrupt the boot to see what evil lurks at $D003, $D00C, and $D00F.

```
*9600<C600.C6FFM

; set up callback #1 after PRODOS file
; is loaded
96F8-   A9 4C       LDA   #$4C
96FA-   8D CB 08    STA   $08CB
96FD-   A9 0A       LDA   #$0A
96FF-   8D CC 08    STA   $08CC
9702-   A9 97       LDA   #$97
9704-   8D CD 08    STA   $08CD

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

; (callback #1) set up callback #2 just
; before switching on RAM bank 1
970A-   A9 4C       LDA   #$4C
970C-   8D 4F 58    STA   $584F
970F-   A9 1C       LDA   #$1C
9711-   8D 50 58    STA   $5850
9714-   A9 97       LDA   #$97
9716-   8D 51 58    STA   $5851

; continue the boot
9719-   4C 00 20    JMP   $2000

; (callback #2) switch to RAM bank 1
; and dump the entire contents into
; main memory
971C-   2C 8B C0    BIT   $C08B
971F-   2C 8B C0    BIT   $C08B
9722-   A2 30       LDX   #$30
9724-   A0 00       LDY   #$00
9726-   B9 00 D0    LDA   $D000,Y
9729-   99 00 20    STA   $2000,Y
972C-   C8          INY
972D-   D0 F7       BNE   $9726
972F-   EE 28 97    INC   $9728
9732-   EE 2B 97    INC   $972B
9735-   CA          DEX
9736-   D0 EE       BNE   $9726
```

```
; switch back to ROM
9738-   2C 82 C0    BIT   $C082

; reboot to my work disk
973B-   4C 00 C5    JMP   $C500

*BSAVE TRACE.D003,A$9600,L$13E
*9600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT1.D000-FFFF,A$2000,L$3000
]CALL -151
*2003L

2003-   4C E5 D2    JMP   $D2E5
```

```
*22E5L

; This just sets up the absolute
; addresses in the RWTS so it can put
; the sector data in-place in its final
; memory destination. Perfectly normal.
; By the way, ($44) points to $1400;
; that was initialized at $5849, just
; before this call.
22E5-   A5 44       LDA   $44
22E7-   A4 45       LDY   $45
22E9-   8D CC D3    STA   $D3CC
22EC-   8C CD D3    STY   $D3CD
22EF-   8D 12 D4    STA   $D412
22F2-   8C 13 D4    STY   $D413
22F5-   18          CLC
22F6-   69 55       ADC   #$55
22F8-   90 01       BCC   $22FB
22FA-   C8          INY
22FB-   85 3A       STA   $3A
22FD-   84 3B       STY   $3B
22FF-   8D DE D3    STA   $D3DE
2302-   8C DF D3    STY   $D3DF
2305-   18          CLC
2306-   69 55       ADC   #$55
2308-   90 01       BCC   $230B
230A-   C8          INY
230B-   85 3C       STA   $3C
230D-   84 3D       STY   $3D
230F-   8D F0 D3    STA   $D3F0
2312-   8C F1 D3    STY   $D3F1
2315-   60          RTS

*200CL

200C-   4C 16 D3    JMP   $D316
```

OK, I know that routine reads the next available address field. (It was originally at $5316; I already traced it earlier.)

```
*200FL

200F-   4C 6A D3    JMP   $D36A
```

This routine was originally at $536A. It reads the data field.

Now this entire loop makes more sense. It's literally copying different chunks of code into the middle of the RWTS and trying to find a variation that can read several sectors in a row. Let's list it again and sprinkle some more comments around.

The main loop starts at $5858. There are 4 counters, $00, $01, $02, and $03. $00 is the number of times we've tried to read a sector (starts at 3 and decremented). $01 is the number of times it succeeded (starts at 3 and checked after all reads). $02 is used as an index for copying $13-byte chunks of code into the RWTS. When it goes negative, we've tried all of the RWTS variations. $02 starts at $4B (set at $5800). $03 is a death counter for finding the proper sector.

```
5858-   A2 03       LDX   #$03
585A-   86 00       STX   $00
585C-   86 01       STX   $01

; $03 is an inner loop death counter
; for finding the proper sector
585E-   A2 15       LDX   #$15
5860-   86 03       STX   $03
5862-   C6 03       DEC   $03
```

```
; when $03 goes negative, give up on
; finding the proper sector (with this
; RWTS variation)
5864-   30 12       BMI   $5878

; read next available address field
5866-   20 0C D0    JSR   $D00C

; if that returned an error, loop back
; and try again (decrements $03)
5869-   B0 F7       BCS   $5862

; After the routine at $D00C, the Y
; register has an address field value
; (guessing this is a sector number)
586B-   C0 06       CPY   #$06
586D-   D0 F3       BNE   $5862

; try to read a sector worth of data
586F-   20 0F D0    JSR   $D00F

; if that worked, branch
5872-   90 19       BCC   $588D

; decrement read-data death counter and
; try again
5874-   C6 01       DEC   $01
5876-   10 E6       BPL   $585E

; if we've tried all variations, give
; up entirely, otherwise fall through
5878-   A6 02       LDX   $02
587A-   30 26       BMI   $58A2
```

```
; copy $13 bytes of code into the
; middle of the RWTS routine(!)
587C-   A0 12       LDY   #$12
587E-   BD A6 58    LDA   $58A6,X
5881-   99 96 D3    STA   $D396,Y
5884-   CA          DEX
5885-   88          DEY
5886-   10 F6       BPL   $587E

; store the index pointer so the next
; time through the loop, we copy a
; different chunk of code into the
; middle of the RWTS routine(!)
5888-   86 02       STX   $02

; start over with this RWTS variation
588A-   4C 58 58    JMP   $5858

; execution continues here (from $5872)
; after a successful sector read --
; decrement the sector read counter and
; try again
588D-   C6 00       DEC   $00
588F-   10 CD       BPL   $585E

; if any of the reads failed, branch to
; try the next RWTS variation
5891-   A5 01       LDA   $01
5893-   C9 03       CMP   #$03
5895-   D0 E1       BNE   $5878

; success path falls through to here --
; we have found the working RWTS and we
; are ready to move on with the boot
5897-   A6 3E       LDX   $3E

; turn off drive motor
5899-   BD 88 C0    LDA   $C088,X
```

```
; switch to ROM
589C-   2C 8A C0    BIT   $C08A

; continue with "stage 2" loader (to
; launch .SYSTEM file, presumably)
589F-   4C 00 08    JMP   $0800

; failure path ends up here
58A2-   4C F6 BF    JMP   $BFF6
```

Here are the four different variations that it copies into the RWTS. Note that the overflow bit will always be set by the time this code is run, so the "BVC" instruction burns 3 cycles but never branches. Each variation burns a different number of CPU cycles (listed in the right margin) before checking the third nibble of the data prologue.

#1:

```
58DF-   50 05       BVC   $58E6     | 3
58E1-   BD 8D C0    LDA   $C08D,X   | 4
58E4-   48          PHA             | 3
58E5-   68          PLA             | 4
58E6-   BD 8C C0    LDA   $C08C,X
58E9-   10 FB       BPL   $58E6
58EB-   C9 A9       CMP   #$A9
58ED-   F0 05       BEQ   $58F4
58EF-   88          DEY
58F0-   10 F4       BPL   $58E6
```

#2:

```
58CC-   50 05       BVC   $58D3     | 3
58CE-   BD 8D C0    LDA   $C08D,X   | 4
58D1-   EA          NOP             | 2
58D2-   EA          NOP             | 2
58D3-   BD 8C C0    LDA   $C08C,X
58D6-   10 FB       BPL   $58D3
58D8-   C9 A9       CMP   #$A9
58DA-   F0 05       BEQ   $58E1
58DC-   88          DEY
58DD-   10 F4       BPL   $58D3
```

#3:

```
58B9-   50 04       BVC   $58BF     | 3
58BB-   BD 8D C0    LDA   $C08D,X   | 4
58BE-   EA          NOP             | 2
58BF-   BD 8C C0    LDA   $C08C,X
58C2-   10 FB       BPL   $58BF
58C4-   C9 A9       CMP   #$A9
58C6-   F0 06       BEQ   $58CE
58C8-   88          DEY
58C9-   10 F4       BPL   $58BF
58CB-   EA          NOP
```

#4:

```
58A6-   50 03       BVC   $58AB     | 3
58A8-   BD 8D C0    LDA   $C08D,X   | 4
58AB-   BD 8C C0    LDA   $C08C,X
58AE-   10 FB       BPL   $58AB
58B0-   C9 A9       CMP   #$A9
58B2-   F0 07       BEQ   $58BB
58B4-   88          DEY
58B5-   D0 F4       BNE   $58AB
58B7-   EA          NOP
58B8-   EA          NOP
```

None of these timing variations work on my EDD bit copy, because they all need some timing bits between the second and third nibble of the data prologue, and EDD doesn't preserve those by default. There's no nibble check, per se. The entire structure of the disk itself is designed to foil bit copiers.

Chapter 4
In Which We Finally Make Some Forward Progress

Let's capture the "stage 2" code that ends up at $0800, then I can see what I need to do to inject BASIC.SYSTEM into memory and launch it.

```
*9600<C600.C6FFM

; set up callback #1
96F8-   A9 4C       LDA   #$4C
96FA-   8D CB 08    STA   $08CB
96FD-   A9 0A       LDA   #$0A
96FF-   8D CC 08    STA   $08CC
9702-   A9 97       LDA   #$97
9704-   8D CD 08    STA   $08CD

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

; (callback #1) set up callback #2
970A-   A9 4C       LDA   #$4C
970C-   8D 9F 58    STA   $589F
970F-   A9 1C       LDA   #$1C
9711-   8D A0 58    STA   $58A0
9714-   A9 97       LDA   #$97
9716-   8D A1 58    STA   $58A1

; continue the boot
9719-   4C 00 20    JMP   $2000
```

```
; (callback #2) copy stage 2 code from
; $0800 to graphics page so it survives
; a reboot
971C-   A2 18       LDX   #$18
971E-   A0 00       LDY   #$00
9720-   B9 00 08    LDA   $0800,Y
9723-   99 00 28    STA   $2800,Y
9726-   C8          INY
9727-   D0 F7       BNE   $9720
9729-   EE 22 97    INC   $9722
972C-   EE 25 97    INC   $9725
972F-   CA          DEX
9730-   D0 EE       BNE   $9720

; turn off slot 6 drive motor
9732-   AD E8 C0    LDA   $C0E8

; reboot to my work disk
9735-   4C 00 C5    JMP   $C500

*BSAVE TRACE2,A$9600,L$138
*9600G
...reboots slot 6...
...reboots slot 5...

]BSAVE STAGE2 0800-14FF,A$2800,L$D00
]CALL -151
*800<2800.34FFM
```

```
*800L

; this is looking through the disk
; catalog (loaded at $0C00)
0800-   A9 0C       LDA   #$0C
0802-   85 11       STA   $11
0804-   A9 04       LDA   #$04
0806-   2C A5 10    BIT   $10A5
0809-   18          CLC
080A-   6D 23 0C    ADC   $0C23
080D-   85 10       STA   $10
080F-   B0 12       BCS   $0823
0811-   6D 23 0C    ADC   $0C23
0814-   90 0F       BCC   $0825
0816-   A5 11       LDA   $11
0818-   4A          LSR
0819-   90 0A       BCC   $0825
081B-   C9 09       CMP   #$09
081D-   F0 1E       BEQ   $083D
081F-   A9 04       LDA   #$04
0821-   85 10       STA   $10
0823-   E6 11       INC   $11
0825-   A0 10       LDY   #$10
0827-   A9 FF       LDA   #$FF
0829-   51 10       EOR   ($10),Y
082B-   D0 DA       BNE   $0807
082D-   A8          TAY
082E-   B1 10       LDA   ($10),Y
0830-   F0 D5       BEQ   $0807
0832-   29 0F       AND   #$0F
0834-   8D 80 02    STA   $0280
0837-   C9 08       CMP   #$08
0839-   90 CC       BCC   $0807
083B-   B0 03       BCS   $0840
083D-   F0 6E       BEQ   $08AD
083F-   00          BRK
0840-   A8          TAY
```

```
; $0965 contains the string ".SYSTEM",
; so this is checking whether this file
; ends with the string ".SYSTEM"
0841-   A2 06       LDX   #$06
0843-   B1 10       LDA   ($10),Y
0845-   5D 65 09    EOR   $0965,X
0848-   0A          ASL

; if not, loop to find the next file
0849-   D0 BC       BNE   $0807
084B-   88          DEY
084C-   CA          DEX
084D-   10 F4       BPL   $0843

; copy the filename into the buffer at
; $0280 and another buffer at $093B (I
; think the second one is used for
; error messages if things go wrong)
084F-   A0 00       LDY   #$00
0851-   C8          INY
0852-   B1 10       LDA   ($10),Y
0854-   99 80 02    STA   $0280,Y
0857-   09 80       ORA   #$80
0859-   99 3B 09    STA   $093B,Y
085C-   CC 80 02    CPY   $0280
085F-   D0 F0       BNE   $0851

; pad error message with spaces
0861-   A9 A0       LDA   #$A0
0863-   99 3C 09    STA   $093C,Y
0866-   98          TYA
0867-   69 13       ADC   #$13
0869-   8D 4F 09    STA   $094F

; open file (ProDOS MLI $C8)
086C-   20 00 BF    JSR   $BF00
086F-  [C8 50 09]
0872-   D0 46       BNE   $08BA
```

```
; get file EOF (MLI $D1)
0874-   20 00 BF    JSR   $BF00
0877-  [D1 56 09]
087A-   D0 3E       BNE   $08BA
087C-   AD 5A 09    LDA   $095A
087F-   D0 53       BNE   $08D4
0881-   AD 59 09    LDA   $0959
0884-   C9 98       CMP   #$98
0886-   B0 4C       BCS   $08D4
0888-   8D 60 09    STA   $0960
088B-   AD 58 09    LDA   $0958
088E-   8D 5F 09    STA   $095F

; read file (MLI $CA)
0891-   20 00 BF    JSR   $BF00
0894-  [CA 5B 09]
0897-   F0 06       BEQ   $089F
0899-   C9 56       CMP   #$56
089B-   F0 37       BEQ   $08D4
089D-   D0 1B       BNE   $08BA

; close file (MLI $CC)
089F-   20 00 BF    JSR   $BF00
08A2-  [CC 63 09]
08A5-   D0 13       BNE   $08BA
08A7-   AD 82 C0    LDA   $C082

; jump to beginning of loaded file
08AA-   4C 00 20    JMP   $2000
```

This is where I want to interrupt the boot and inject my clean version of BASIC.SYSTEM. (I could probably do it slightly earlier to avoid loading its BASIC.SYSTEM file, but I don't want to run afoul of any expected side effects of having loaded the file through the ProDOS MLI.)

```
*BLOAD BASIC.SYSTEM,A$6000

*9600<C600.C6FFM

; set up callback #1
96F8-   A9 4C       LDA   #$4C
96FA-   8D CB 08    STA   $08CB
96FD-   A9 0A       LDA   #$0A
96FF-   8D CC 08    STA   $08CC
9702-   A9 97       LDA   #$97
9704-   8D CD 08    STA   $08CD

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

; (callback #1) set up callback #2
970A-   A9 4C       LDA   #$4C
970C-   8D 9F 58    STA   $589F
970F-   A9 1C       LDA   #$1C
9711-   8D A0 58    STA   $58A0
9714-   A9 97       LDA   #$97
9716-   8D A1 58    STA   $58A1

; continue the boot
9719-   4C 00 20    JMP   $2000

; (callback #2) set up callback #3
971C-   A9 4C       LDA   #$4C
971E-   8D AA 08    STA   $08AA
9721-   A9 2E       LDA   #$2E
9723-   8D AB 08    STA   $08AB
9726-   A9 97       LDA   #$97
9728-   8D AC 08    STA   $08AC

; continue the boot
972B-   4C 00 08    JMP   $0800
```

```
; (callback #3) move BASIC.SYSTEM into
; place (I manually BLOADed this file
; already at $6000)
972E-   A2 28       LDX   #$28
9730-   A0 00       LDY   #$00
9732-   B9 00 60    LDA   $6000,Y
9735-   99 00 20    STA   $2000,Y
9738-   C8          INY
9739-   D0 F7       BNE   $9732
973B-   EE 34 97    INC   $9734
973E-   EE 37 97    INC   $9737
9741-   CA          DEX
9742-   D0 EE       BNE   $9732

; tell BASIC.SYSTEM not to look for a
; STARTUP file
9744-   A9 00       LDA   #$00
9746-   8D 06 20    STA   $2006

; continue the boot with the clean
; version of BASIC.SYSTEM
9749-   4C 00 20    JMP   $2000

*BSAVE TRACE3,A$9600,L$14C
*9600G
...reboots slot 6...
...displays ProDOS title page...
...clears screen...

            PRODOS BASIC 1.5
        COPYRIGHT APPLE  1983-92

]
```

Son of a biscuit. It actually worked.

Chapter 5
In Which We Finally Catch A Break, And Our Adventure Comes To A Sudden But Satisfying Conclusion

]CAT

/ALGERNON2

NAME	TYPE	BLOCKS	MODIFIED
PRODOS	SYS	30	9-OCT-86
LOGO.BIN	BIN	17	17-DEC-86
*BASIC.SYSTEM	SYS	21	14-NOV-86
*ERR.FIX	BIN	1	15-NOV-86
CHARGEN	BIN	11	17-DEC-86
TITLE.PIC	BIN	17	17-DEC-86
WINNER.PIC	BIN	17	17-DEC-86
*ALGERNON	BAS	38	17-DEC-86
*STARTUP	BAS	1	15-NOV-86
MAZE1	TXT	3	17-DEC-86
MAZE3	TXT	3	17-DEC-86
MAZE4	TXT	3	17-DEC-86
MAZE5	TXT	3	17-DEC-86
MAZE9	TXT	3	17-DEC-86
MAZE6	TXT	3	17-DEC-86
MAZE2	TXT	3	17-DEC-86
MAZE7	TXT	3	17-DEC-86
MAZE8	TXT	3	17-DEC-86
MAZE10	TXT	3	17-DEC-86
LOGO	BAS	1	15-NOV-86
NAMES	TXT	1	17-DEC-86
LOGO2	BAS	1	15-NOV-86
SOUND	TXT	1	17-DEC-86
USERJHHNGHHN	TXT	3	<NO DATE>
USERCHMGKBMH	TXT	3	<NO DATE>
USERMAZECHEE	TXT	3	<NO DATE>
USERREBECCHE	TXT	3	<NO DATE>

BLOCKS FREE: 74 BLOCKS USED: 206

The custom floppy device driver is in memory, and I have unfettered access to the disk through a clean version of BASIC.SYSTEM.

```
]PREFIX /ALGERNON2
]LOAD STARTUP
]LIST

 3  ONERR  GOTO 30
 4  POKE 1011,0: REM  MESSES UP
     RESET
 10  POKE 104,64: POKE 16384,0
 15  PRINT  CHR$ (4)"BLOAD ERR.FI
     X": REM  Loads in small rout
     ine to fix some ON ERR probl
     ems. See p136 of the APPLESO
     FT manual.
 20  PRINT "RUN LOGO"
 30  RUN
```

Un. Fettered. Access.

But how do I copy all these files to a standard disk? I could do it one at a time -- LOAD and BLOAD work, so I could simply load each file into memory and reboot and save it.

But wait. ProDOS has separate device drivers for floppies and hard drives. Maybe...

```
[S7,D1=ProDOS hard drive, "A4AMCRACK"]

]PREFIX /A4AMCRACK
```

]CAT

/A4AMCRACK

NAME	TYPE	BLOCKS	MODIFIED
*PRODOS	SYS	35	6-AUG-03
RAM.DRV.SYSTEM	SYS	4	29-NOV-10
PROSEL.SYSTEM	SYS	1	1-APR-88
APPLICATIONS	DIR	2	18-DEC-14
BASIC.SYSTEM	SYS	21	6-DEC-91
COMMANDS	DIR	1	20-MAR-14
DOC	DIR	1	20-MAR-14
DOS3.3	DIR	1	20-MAR-14
ARCHIVE	DIR	1	8-FEB-15
MERLIN	DIR	2	1-OCT-14
INCOMING	DIR	1	30-SEP-14
PROSEL	BIN	13	17-OCT-14
UTIL	DIR	6	20-MAR-14

BLOCKS FREE:60603 BLOCKS USED: 4932

Not only do I have unfettered access to the floppy disk, I have my entire hard drive of utilities at my disposal.

]-/A4AMCRACK/APPLICATIONS/COPYIIPLUS8.4
/UTIL.SYSTEM
...launches Copy][+...

```
  --> CREATE SUBDIRECTORY
    --> SLOT 7, DRIVE 1
      --> SUBDIRECTORY NAME: ALGERNON2

  --> COPY
    --> FILES
      --> from SLOT 6, DRIVE 1
      -->   to SLOT 7, DRIVE 1,
            ALGERNON2
        --> all files
```

It works. Copy][+ uses the version of ProDOS in memory, including the custom floppy disk driver. As far as Copy][+ is concerned, there's nothing unusual about this disk or its files. Hooray for abstractions!

Now that I have all the files off the original disk, I can safely put it away and never touch it again. (Whew. Good riddance.)

[S6,D1=blank disk]

]PR#7

Using Copy][+ again, I simply recreate the original disk with a clean copy of the PRODOS file. (I have a directory of PRODOS files of different versions for just such an occasion, because that's not weird at all.)

```
[Copy ][+ 8.4]
  --> FORMAT DISK
    --> PRODOS
      --> SLOT 6, DRIVE 1
        --> VOLUME NAME: ALGERNON2

  --> COPY
    --> FILES
      --> from SLOT 7, DRIVE 1
      -->   to SLOT 6, DRIVE 1
        --> ARCHIVES/PRODOS1.1.1/PRODOS

  --> COPY
    --> FILES
      --> from SLOT 7, DRIVE 1,
               ALGERNON2
      -->   to SLOT 6, DRIVE 1
        --> all files except PRODOS

]PR#6
...works...

Quod erat liberandum.
```

EOF