BurgerTime

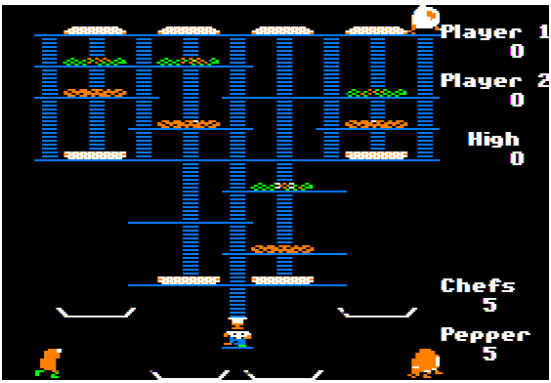
2015-12-31

4am
to deprotect
and preserve

Contents

----------BurgerTime----------
A 4am crack 2015-12-31

Name: BurgerTime
Genre: arcade
Year: 1982
Author: Data East USA
Publisher: Mattel
Media: single-sided 5.25-inch floppy
OS: DOS 3.3 with custom bootloader and
custom file loader and custom RWTS,
so not really like DOS 3.3 at all
Previous cracks: The Freeze



Chapter 0
In Which Various Automated Tools Fail In Interesting Ways

COPYA
  read error on first pass

Locksmith Fast Disk Backup
  bizarre combination of success and failure

--v--

```
     LOCKSMITH 7.0  FAST DISK BACKUP


   R.**********.***..*...**************
   W
HEX 00000000000000001111111111111111222
TRK 0123456789ABCDEF0123456789ABCDEF012
   0.AAAAAAAAAA.AAA.......DDDDDDDDDDDDD
   1.AAAAAAAAAA.AAA.......DDDDDDDDDDDDD
   2.AAAAAAAAAA.AAA.......DDDDDDDDDDDDD
   3.AAAAAAAAAA.AAA......DDDDDDDDDDDDDD
   4.AAAAAAAAAA.AAA......DDDDDDDDDDDDDD
   5.AAAAAAAAAA.AAA......DDDDDDDDDDDDDD
   6.AAAAAAAAAA.AAA......DDDDDDDDDDDDDD
   7.AAAAAAAAAA.AAA..D...DDDDDDDDDDDDDD
   8.AAAAAAAAAA.AAA......DDDDDDDDDDDDDD
   9.AAAAAAAAAA.AAA......DDDDDDDDDDDDDD
   A.AAAAAAAAAA.AAA......DDDDDDDDDDDDDD
   B.AAAAAAAAAA.AAA......DDDDDDDDDDDDDD
   C.AAAAAAAAAA.AAA......DDDDDDDDDDDDDD
   D.AAAAAAAAAA.AAA......DDDDDDDDDDDDDD
12 E.AAAAAAAAAA.AAA......DDDDDDDDDDDDD.
   F.AAAAAAAAAA.AAA......DDDDDDDDDDDDD.
[                ] PRESS [RESET] TO EXIT
```

--^--

Tracks $00, $0B, $0F, $10, $12, $13, and $14 are standard. Also a few sectors of T15 and T22, and all of T11 except for what appears to be a bad sector in the middle. (The original disk boots fine, and the entire disk catalog on that track is fake anyway, so I think I got lucky.)

EDD 4 bit copy (no sync, no count)
no errors, but copy displays an error "UNABLE TO LOAD GAME" and hangs

Copy][+ nibble editor
T01-T0A, T0C-T0E are unformatted
(hi-res disk scan confirms this)
T15+ use a modified data prologue
("D5 AA AA" instead of "D5 AA AD")

Disk Fixer
T00 -> custom bootloader
T11 -> DOS 3.3 style disk catalog
(fake, all files point to nothing)
["0" -> "Input/Output Control"]
set Data Prologue to "D5 AA AA"
Success! T15+ readable

Why didn't COPYA work?
  unformatted tracks, modified prologue

Why didn't Locksmith FDB work?
  modified prologue

Why didn't my EDD copy work?
  I assume there's some sort of
  protection check during boot. Disks
  don't say "UNABLE TO LOAD GAME"
  unless someone tells them to.

Next steps:

  1. Trace the boot
  2. ???

Chapter 1
In Which We Start From Scratch

We're starting from bare metal on this one. My automated tools, they do nothin for us. Strap in.

```
[S6,D1=original disk]

[S6,D2=crack-in-progress (the partial
  copy I made with Locksmith Fast Disk
  Backup)]

[S5,D1=my work disk]

]PR#5
...
]CALL -151

*9600<C600.C6FFM

; copy boot sector (T00,S00) to the
; graphics page so it survives a reboot
96F8-   A0 00       LDY   #$00
96FA-   B9 00 08    LDA   $0800,Y
96FD-   99 00 20    STA   $2000,Y
9700-   C8          INY
9701-   D0 F7       BNE   $96FA

; turn off slot 6 drive motor
9703-   AD E8 C0    LDA   $C0E8

; reboot to my work disk in slot 5
9706-   4C 00 C5    JMP   $C500

*9600G
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT0,A$2000
]CALL -151

*800<2000.20FFM
```

```
*801L

; starts off looking like a DOS 3.3
; bootloader -- zp$27 will be $09 the
; first time through, so this is a way
; to do one-time initialization
0801-   A5 27       LDA   $27
0803-   C9 09       CMP   #$09
0805-   D0 1B       BNE   $0822

; munge reset vector
0807-   A5 2B       LDA   $2B
0809-   8D F4 03    STA   $03F4

; munge boot slot into a vector to read
; more sectors
080C-   4A          LSR
080D-   4A          LSR
080E-   4A          LSR
080F-   4A          LSR
0810-   09 C0       ORA   #$C0
0812-   85 D7       STA   $D7
0814-   A9 5C       LDA   #$5C
0816-   85 D6       STA   $D6
```

Now ($D6) points to $Cx5C (based on the boot slot, e.g. $C65C for slot 6). This entry point will read a sector from track $00 and exit via $0801.

As a bonus, setting $D6 to anything other than 0 sets the RUN flag, so if I managed to break to a BASIC prompt, any command would run. Even trying to do a "CALL -151" to get to the monitor would be blocked. Lovely.

```
0818-   18          CLC
0819-   AD F9 08    LDA   $08F9
081C-   6D FA 08    ADC   $08FA
081F-   8D F9 08    STA   $08F9
0822-   AE FA 08    LDX   $08FA
0825-   F0 15       BEQ   $083C

*8F9.8FA

08F9- 03 04
```

We're loading 4 sectors directly into the text page ($0400..$07FF). Nice.

```
; set up the rest of the zero page
; globals for the sector read routine
0827-   BD FB 08    LDA   $08FB,X
082A-   85 3D       STA   $3D
082C-   CE FA 08    DEC   $08FA
082F-   AD F9 08    LDA   $08F9
0832-   85 27       STA   $27
0834-   CE F9 08    DEC   $08F9

; read a sector (exits via $0801)
0837-   A6 2B       LDX   $2B
0839-   6C D6 00    JMP   ($00D6)
```

```
; Execution continues here (from $0825)
; after we've read all the sectors we
; wanted to read (into $0400..$07FF).
; Switch to the hi-res graphics screen
; (uninitialized).
083C-   2C 52 C0    BIT   $C052
083F-   2C 55 C0    BIT   $C055
0842-   2C 57 C0    BIT   $C057
0845-   2C 50 C0    BIT   $C050

; call part of the code we just read
0848-   20 0A 04    JSR   $040A
```

And that's where I need to interrupt the boot.

Chapter 2
In Which We Eschew Obfuscation

```
*9600<C600.C6FFM

; set up callback after boot0 loads the
; code into the text page
96F8-   A9 4C       LDA   #$4C
96FA-   8D 48 08    STA   $0848
96FD-   A9 0A       LDA   #$0A
96FF-   8D 49 08    STA   $0849
9702-   A9 97       LDA   #$97
9704-   8D 4A 08    STA   $084A

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

; callback is here --
; copy the text page to higher memory
; so it survives a reboot
970A-   A2 04       LDX   #$04
970C-   A0 00       LDY   #$00
970E-   B9 00 04    LDA   $0400,Y
9711-   99 00 24    STA   $2400,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 the 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 0400-07FF,A$2400,L$400
]CALL -151
```

I'll need to leave this code at $2400 for inspection, since I can't put it in the text page and also inspect it in the monitor. Relative branches will look correct, but absolute addresses will be off by $2000.

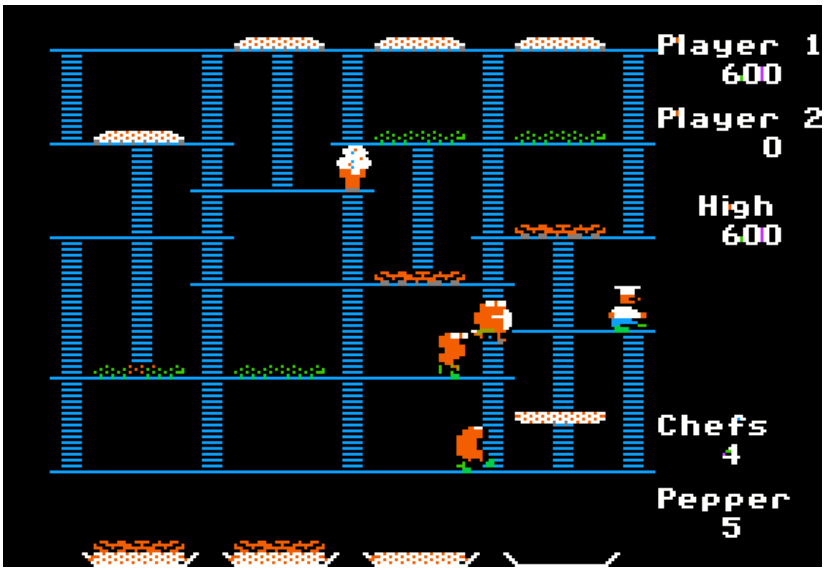
```
*240AL

; wipe most of main memory, $0900+
240A-   A0 09       LDY   #$09
240C-   84 01       STY   $01
240E-   A9 00       LDA   #$00
2410-   85 00       STA   $00
2412-   A8          TAY
2413-   91 00       STA   ($00),Y
2415-   C8          INY
2416-   D0 FB       BNE   $2413
2418-   E6 01       INC   $01
241A-   A6 01       LDX   $01
241C-   E0 C0       CPX   #$C0
241E-   D0 F3       BNE   $2413
2420-   60          RTS
```

Continuing in boot0, from $084B...

```
*BLOAD BOOT0,A$800
```

```
*84BL

; set a (presumably unfriendly) reset
; vector
084B-   A9 00       LDA   #$00
084D-   8D F2 03    STA   $03F2
0850-   A9 04       LDA   #$04
0852-   8D F3 03    STA   $03F3
0855-   49 A5       EOR   #$A5
0857-   8D F4 03    STA   $03F4
085A-   EA          NOP
085B-   EA          NOP
085C-   EA          NOP
085D-   EA          NOP
085E-   20 21 04    JSR   $0421

*2421L

This code is exquisitely obfuscated, so
let's just take it one line at a time.
Everything is important.

2421-   A9 00       LDA   #$00    ;A=00
2423-   85 82       STA   $82     ; =00
```

```
; Stack pointer is uninitialized at
; this point (and not guaranteed to be
; anything in particular on boot), so I
; assume this is the first part of a
; verification that the stack pointer
; isn't being modified by anything the
; original developers didn't know about
; (like an NMI card).
2425-   BA          TSX
2426-   8A          TXA
2427-   49 BC       EOR   #$BC
2429-   85 50       STA   $50
242B-   29 00       AND   #$00     ;A=00
242D-   AA          TAX            ;X=00
242E-   A0 D6       LDY   #$D6     ;Y=D6
2430-   49 54       EOR   #$54     ;A=54
2432-   84 81       STY   $81      ; =D6
```



Now ($81) points to $00D6. That address was set to $5C (at $0816), because it was used as the pointer to $Cx5C to read more sectors from disk (including the code we're now executing).

```
; X=00, so this is taking ($81), which
; is zp$D6, which is $5C, then EOR'ing
; it with A, which is $54
2434-   41 81       EOR   ($81,X) ;A=08
2436-   85 21       STA   $21     ; =08
2438-   98          TYA           ;A=D6
2439-   49 E0       EOR   #$E0    ;A=36
243B-   85 81       STA   $81     ; =36
```

Now ($81) points to $0036, the output vector.

```
243D-   49 A4       EOR   #$A4    ;A=92
243F-   81 81       STA   ($81,X) ; =92
2441-   E6 81       INC   $81     ; =37
2443-   38          SEC
2444-   E9 8B       SBC   #$8B    ;A=07
2446-   30 F0       BMI   $2438   ;nope
2448-   81 81       STA   ($81,X) ; =07
```

Now ($36) points to $0792.

```
244A-   49 07       EOR   #$07    ;A=00
244C-   85 20       STA   $20     ; =00
```

Now ($20) points to $0800, which is the start of the boot0 code.

```
244E-   85 51       STA   $51      ; =00
2450-   49 04       EOR   #$04     ;A=04
2452-   F0 02       BEQ   $2456    ;nope
2454-   85 46       STA   $46      ; =04
2456-   8A          TXA            ;A=00
2457-   85 45       STA   $45      ; =00
```

Now ($45) points to $0400, which is the start of the boot1 code.

```
2459-   18          CLC
245A-   69 D6       ADC   #$D6     ;A=D6
245C-   85 81       STA   $81      ; =D6
```

Now ($81) points to $00D6 again.

```
245E-   4A          LSR            ;A=6B
245F-   AA          TAX            ;X=6B

; $16 + $6B = $81, and ($81) points to
; $00D6, and zp$D6 is still $5C.
2460-   A1 16       LDA   ($16,X) ;A=5C
2462-   49 32       EOR   #$32     ;A=6E
2464-   A8          TAY            ;Y=6E
2465-   49 6A       EOR   #$6A     ;A=04
2467-   F0 E5       BEQ   $244E    ;nope
2469-   85 08       STA   $08      ; =04
246B-   49 3E       EOR   #$3E     ;A=3A
246D-   88          DEY            ;Y=6D

; ($45) points to $0400, and Y=$6D,
; so ($45),Y = $046D, which is $88 (the
; "DEY" instruction we just executed).
; A=$3A, and $3A EOR $88 = $B2.
246E-   51 45       EOR   ($45),Y ;A=B2
```

```
; ($36) points to $0792, and Y=$6D,
; so ($36),Y = $07FF, which is $A0
; (trust me on that). A=$B2 after the
; previous EOR, so $B2 EOR $A0 = $12.
2470-   51 36        EOR   ($36),Y ;A=12
2472-   38           SEC

; ($20) points to $0800, and Y=$6D,
; so ($20),Y = $086D, which is $09.
; A=$12 after the two previous EORs, so
; $12 - $09 = $09. (Think hex.)
2473-   F1 20        SBC   ($20),Y ;A=09

; store that in $07FF
2475-   91 36        STA   ($36),Y ; =09

; done decrypting this page?
2477-   C0 00        CPY   #$00

; nope, branch back to finish it
2479-   D0 F2        BNE   $246D
```

That branch goes to $046D, which is a "DEY" instruction, so we're decrypting memory from the top down.

```
247B-   84 81        STY   $81     ; =00

; decrement page count (initialized to
; 04 at $0469)
247D-   C6 08        DEC   $08

; jump forward if we're done decrypting
247F-   F0 05        BEQ   $2486

; otherwise decrement the target page
; and loop back to decrypt it
2481-   C6 37        DEC   $37
2483-   4C 6D 04     JMP   $046D
```

On the first pass, Y started at $6D, so we only decrypted part of the page. But the index was offset from ($36), which was $0792 (set at $0448), so we end up decrypting $07FF..$0792. Then zp$37 is decremented and Y rolls over to $FF, so we decrypt a full $100 bytes in round 2 ($0791..$0692). Then $0691..$0592. Then $0591..$0492. Then zp$08 hits 0 and we branch forward to $0486.

```
; zp$50 was set from the uninitialized
; stack pointer (EOR'd with some magic
; number), and now we're using that as
; an index into the stack to get some
; other (also uninitialized) byte.
; Again, this will probably be checked
; later to see if the stack has been
; tampered with.
2486-   A6 50       LDX   $50
2488-   BD 00 01    LDA   $0100,X
248B-   85 46       STA   $46
248D-   49 47       EOR   #$47
248F-   8D 33 60    STA   $6033

; everything after this was decrypted
; by the loop we just executed, so this
; listing is meaningless
2492-   64          ???
2493-   1E 11 B4    ASL   $B411,X
2496-   FE 36 96    INC   $9636,X
2499-   70 FE       BVS   $2499
```

To sum up: boot1 decrypts itself using both the calling code and itself as the decryption key, then immediately falls through to decrypted code (at $0492). Patching boot0 to interrupt the boot will cause the decryption to fail. Patching boot1 to interrupt the boot will cause the decryption to fail.

I am stuck.

Chapter 3
In Which We Get Unstuck

The decryption routine at $0421 is complicated and interconnected, but one thing it does not rely on is the caller address. I can simply call it myself and see what happens. As long as boot0 and boot1 are pristine by the time the routine is called, it should work.

```
*9600<C600.C6FFM

; set up callback after boot0 loads
; boot1
96F8-   A9 4C       LDA   #$4C
96FA-   8D 48 08    STA   $0848
96FD-   A9 0A       LDA   #$0A
96FF-   8D 49 08    STA   $0849
9702-   A9 97       LDA   #$97
9704-   8D 4A 08    STA   $084A

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

; callback is here --
; undo the patch we made earlier, so
; boot0 is in a "pristine" state for
; the decryption routine
970A-   A9 20       LDA   #$20
970C-   8D 48 08    STA   $0848
970F-   A9 0A       LDA   #$0A
9711-   8D 49 08    STA   $0849
9714-   A9 04       LDA   #$04
9716-   8D 4A 08    STA   $084A

; call the decryption routine
9719-   20 21 04    JSR   $0421
```

```
; copy the (hopefully decrypted) boot1
; code to higher memory so it survives
; a reboot
971C-   A2 04       LDX   #$04
971E-   A0 00       LDY   #$00
9720-   B9 00 04    LDA   $0400,Y
9723-   99 00 24    STA   $2400,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 and
; reboot to my work disk
9732-   AD E8 C0    LDA   $C0E8
9735-   4C 00 C5    JMP   $C500

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

]BSAVE BOOT1 DECRYPTED,A$2400,L$400
]CALL -151
```

Let's see what that decrypted code was that we were falling through to.

```
*2492L

2492-   60          RTS
```

Well OK then. I guess it worked.

Chapter 4
In Which We Have A Revelation

Continuing the trace from $0861 (after calling the decryption routine)...

```
*BLOAD BOOT0,A$800
*861L

0861-   A9 00       LDA   #$00
0863-   85 24       STA   $24
0865-   85 83       STA   $83
0867-   85 00       STA   $00
0869-   8D 78 02    STA   $0278
086C-   A9 09       LDA   #$09
086E-   85 25       STA   $25
0870-   85 01       STA   $01
```

Not sure what all that's about, but ($24) points to $0900 and ($00) points to $0000.

```
; Hey, I have this decrypted now!
0872-   20 C0 04    JSR   $04C0

*24C0L

; get boot slot (x16) -- generally a
; precursor to doing something disk-
; related
24C0-   A6 2B       LDX   $2B

; (not shown) this subroutine divides
; X by 16 and puts it in Y, so if we
; booted from slot 6, Y is now 6
24C2-   20 6A 05    JSR   $056A
```

```
; set up some stuff based on boot slot
; (this looks suspiciously like the
; kind of initialization that DOS 3.3
; does, tracking the current track of
; each disk in each slot+drive)
24C5-   A9 00       LDA   #$00
24C7-   99 78 02    STA   $0278,Y
24CA-   99 88 02    STA   $0288,Y
24CD-   8D 78 02    STA   $0278

; don't know what these values mean,
; but they're probably important
24D0-   A9 83       LDA   #$83
24D2-   85 76       STA   $76
24D4-   A9 11       LDA   #$11
24D6-   85 80       STA   $80
24D8-   A9 07       LDA   #$07
24DA-   85 81       STA   $81
24DC-   85 82       STA   $82
24DE-   20 BE 06    JSR   $06BE

*26BEL

26BE-   20 06 05    JSR   $0506

*2506L

2506-   A9 80       LDA   #$80
2508-   85 02       STA   $02
250A-   A9 00       LDA   #$00
250C-   85 03       STA   $03
```

Now ($02) points to $0080, which was set to $11 at $04D6.

```
250E-   A0 00       LDY   #$00     ;Y=00
2510-   B1 02       LDA   ($02),Y  ;A=11
2512-   85 40       STA   $40      ; =11
2514-   C8          INY            ;Y=01
2515-   B1 02       LDA   ($02),Y  ;A=07
2517-   85 F2       STA   $F2      ; =07
2519-   C8          INY            ;Y=02
251A-   B1 02       LDA   ($02),Y  ; =07
251C-   85 3F       STA   $3F      ; =07
251E-   E6 02       INC   $02
2520-   E6 02       INC   $02
2522-   E6 02       INC   $02
```

Now ($02) points to $0083, which was set to $00 at $0865.

```
2524-   A9 D8       LDA   #$D8     ;A=D8
2526-   85 47       STA   $47      ; =D8
2528-   A4 F2       LDY   $F2      ;Y=07

; map of physical to logical sectors
252A-   B9 5A 05    LDA   $055A,Y
252D-   85 E9       STA   $E9
252F-   20 71 05    JSR   $0571

*2571L

2571-   A6 2B       LDX   $2B
2573-   20 7A 05    JSR   $057A
2576-   20 FF 05    JSR   $05FF
2579-   60          RTS
```

I won't show $057A, but it moves the drive arm to the track given in zp$40, which was set to $11 at $0512.

```
*25FFL

; disk read routine that looks
; suspiciously like the one in the
; Disk II firmware (at $C65C)
25FF-   18          CLC
2600-   08          PHP
2601-   BD 8C C0    LDA   $C08C,X
2604-   10 FB       BPL   $2601
2606-   49 D5       EOR   #$D5
2608-   D0 F7       BNE   $2601
260A-   BD 8C C0    LDA   $C08C,X
260D-   10 FB       BPL   $260A
260F-   C9 AA       CMP   #$AA
2611-   D0 F3       BNE   $2606
2613-   EA          NOP
2614-   BD 8C C0    LDA   $C08C,X
2617-   10 FB       BPL   $2614
2619-   C9 96       CMP   #$96
261B-   F0 09       BEQ   $2626
261D-   28          PLP
261E-   90 DF       BCC   $25FF
2620-   49 AA       EOR   #$AA    <-- !
2622-   F0 25       BEQ   $2649
2624-   D0 D9       BNE   $25FF
.
. [rest of routine is uninteresting,
.  except to note that it stores the
.  sector data in ($24), which points
.  to $0900 (set at $086E)]
.
```

Two key takeaways here. First, we're reading the unreadable sector T11,S07. I originally thought this was a bad sector on my original disk, but I was wrong. Going back to it in a sector editor, it's perfectly readable if I set the data prologue to "D5 AA AA". It looks like this:

--v--

```
-------------- DISK EDIT --------------
TRACK $11/SECTOR $07/VOLUME $FE/BYTE$00
---------------------------------------
$00:>FF<00 00 11 07 00 00 00   .@@QG@@@
$08: 00 00 00 22 0F 02 C8 C5   @@@"OBHE
$10: CC CC CF A0 A0 A0 A0 A0   LLO
$18: A0 A0 A0 A0 A0 A0 A0 A0
$20: A0 A0 A0 A0 A0 A0 A0 A0
$28: A0 A0 A0 A0 02 00 22 0D       B@"M
$30: 04 C2 D5 D2 C7 C5 D2 D4   DBURGERT
$38: C9 CD C5 A0 A0 A0 A0 A0   IME
$40: A0 A0 A0 A0 A0 A0 A0 A0
$48: A0 A0 A0 A0 A0 A0 A0 B5           5
$50: 00 17 08 04 CD C4 D3 C1   @WHDMDSA
$58: C4 CA A0 A0 A0 A0 A0 A0   DJ
$60: A0 A0 A0 A0 A0 A0 A0 A0
$68: A0 A0 A0 A0 A0 A0 A0 A0
$70: A0 A0 26 00 00 00 00 00     &@@@@@
$78: 00 00 00 00 00 00 00 00   @@@@@@@@
---------------------------------------
BUFFER 0/SLOT 6/DRIVE 1/MASK OFF/NORMAL

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

--^--

That looks *exactly* like a DOS 3.3 catalog sector. I'll come back to that revelation later.

Takeaway #2: the RWTS that reads this sector (and presumably the rest of the disk) is tucked away behind the insane decryption routine at $0421. In order to modify it to read a standard format disk (by changing $0621 from $AA to $AD to match the third nibble of the data prologue), I'll need to write the unencrypted bootloader back to disk and disable the decryption routine.

Chapter 5
In Which We Make The Sort Of Progress That Doesn't Feel Like Progress

```
]PR#5
...
]CALL -151

; straightforward multi-sector write
; loop, via the RWTS vector at $03D9
08C0-   A9 08       LDA   #$08
08C2-   A0 E8       LDY   #$E8
08C4-   20 D9 03    JSR   $03D9
08C7-   AC ED 08    LDY   $08ED
08CA-   88          DEY
08CB-   10 05       BPL   $08D2
08CD-   A0 0F       LDY   #$0F
08CF-   CE EC 08    DEC   $08EC
08D2-   8C ED 08    STY   $08ED
08D5-   CE F1 08    DEC   $08F1
08D8-   CE E1 08    DEC   $08E1
08DB-   D0 E3       BNE   $08C0
08DD-   60          RTS

*8E0.8FF

08E0- 00 05 00 00 00 00 00 00
         ^^
    sector count

08E8- 01 60 01 00 00 04 FB 08
         ^^ ^^    ^^ ^^
         S6 D1    T0 S4

08F0- 00 27 00 00 02 00 FE 60
      ^^^^^       ^^
     address    write

08F8- 01 00 00 00 01 EF D8 00

*BSAVE WRITE BOOT1,A$8C0,L$40
```

```
; load decrypted bootloader
*BLOAD BOOT0,A$2300
*BLOAD BOOT1 DECRYPTED,A$2400

; disable decryption loop (we'll still
; let it run, but this disables the
; actual "STA" instruction at $0475,
; so it won't overwrite the already-
; decrypted code we're about to write
; to disk)
*2475:24

; normalize RWTS by fixing the third
; nibble of the data prologue
*2621:AD

[S6,D1=crack-in-progress (the partial
  copy I made with Locksmith Fast Disk
  Backup)]

; write decrypted+patched bootloader
; to disk
*8C0G
...write write write...
```

Now I need to convert the protected tracks that use the non-standard data prologue. Super Demuffin only converts whole tracks, but track $15 and $22 are a mix of protected and unprotected sectors. I'll need to make an RWTS file and pump it through Advanced Demuffin.

[S6,D1=DOS 3.3 system master]

```
]PR#6
...
]CALL -151

; copy RWTS
*3800<B800.BFFFM

; modify third nibble of data prologue
*38FC:AA

*BSAVE RWTS,A$3800,L$800,S5,D1

*BRUN ADVANCED DEMUFFIN 1.5,S5,D1

[S6,D1=original disk]
[S6,D2=crack-in-progress]

[press "5" to switch to slot 5]

[press "R" to load a new RWTS module]
  --> At $B8, load "RWTS" from drive 1

[press "6" to switch to slot 6]

[press "C" to convert disk]

[press "Y" to change default values]
```

```
                    --v--

ADVANCED DEMUFFIN 1.5    (C) 1983, 2014
ORIGINAL BY THE STACK    UPDATES BY 4AM
=========================================


INPUT ALL VALUES IN HEX


SECTORS PER TRACK? (13/16) 16

START TRACK: $15         <-- change this
START SECTOR: $03        <-- change this

END TRACK: $22
END SECTOR: $0D          <-- change this

INCREMENT: 1

MAX # OF RETRIES: 0

COPY FROM DRIVE 1
TO DRIVE: 2
=========================================
16SC $15,$03-$22,$0D BY$01 S6,D1->S6,D2

                    --^--
```

And here we go...

--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 $15,$03-$22,$0D BY$01 S6,D1->S6,D2
```

--^--

And another conversion with the same RWTS, this time for that one protected sector on track $11.

--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 $11,$07-$11,$07 BY$01 S6,D1->S6,D2
```

--^--

```
[S6,D1=crack-in-progress]

]PR#6
..."UNABLE TO LOAD GAME"...
```

There is an explicit protection check somewhere after it reads T11,S07. Now that I've decrypted the bootloader and patched the RWTS, it's finally able to get far enough to detect that I'm running an unauthorized copy.

In other words, I now have a COPYA-able copy that is just as broken as my EDD bit copy. This we call progress.

Chapter 6
In Which We Find That Which Does Not Wish To Be Found

```
Picking up where we left off, at $06C1:

]PR#5
...
]BLOAD BOOT1 DECRYPTED,A$2400
]CALL -151

*26C1L

26C1-   A6 2B       LDX   $2B

; I'll return to this in a moment
26C3-   20 FA 06    JSR   $06FA

; roll one bit of the accumulator into
; the carry
26C6-   2A          ROL

; if carry is clear, skip over the
; following code
26C7-   90 2F       BCC   $26F8

; turn off the drive motor
26C9-   A4 2B       LDY   $2B
26CB-   B9 88 C0    LDA   $C088,Y

; clear text page 2
26CE-   A9 A0       LDA   #$A0
26D0-   A2 00       LDX   #$00
26D2-   9D 00 08    STA   $0800,X
26D5-   E8          INX
26D6-   D0 FA       BNE   $26D2
26D8-   EE D4 06    INC   $06D4
26DB-   AC D4 06    LDY   $06D4
26DE-   C0 0C       CPY   #$0C
26E0-   D0 F0       BNE   $26D2
```

```
; display a message
26E2-   A0 00       LDY   #$00
26E4-   B9 89 07    LDA   $0789,Y
26E7-   99 00 08    STA   $0800,Y
26EA-   C8          INY
26EB-   C0 28       CPY   #$28
26ED-   D0 F5       BNE   $26E4

; show it
26EF-   AD 51 C0    LDA   $C051
26F2-   AD 55 C0    LDA   $C055

; jump to The Badlands
26F5-   4C DF 07    JMP   $07DF

; execution continues here (from $06C7)
26F8-   60          RTS
```

Here's the message that is displayed at $06E2:

```
*FC58G N 400<2789.27B0M

          UNABLE TO LOAD GAME
```

Don't look now, but I think we just found the copy protection.

```
*26FAL

26FA-   BD 89 C0    LDA   $C089,X
26FD-   A9 56       LDA   #$56
26FF-   85 11       STA   $11
2701-   D0 01       BNE   $2704
2703-   D0 C6       BNE   $26CB
2705-   12          ???
```

Oh joy, more obfuscated code. The "BNE" at $0701 unconditionally branches into the middle of the next instruction, which confuses the monitor's built-in disassembler.

```
*2703:EA
*26FAL

26FA-   BD 89 C0    LDA   $C089,X
26FD-   A9 56       LDA   #$56
26FF-   85 11       STA   $11
2701-   D0 01       BNE   $2704
2703-   EA          NOP
2704-   C6 12       DEC   $12
2706-   F0 03       BEQ   $270B
2708-   D0 0A       BNE   $2714
270A-   D0 C6       BNE   $26D2
270C-   11 D0       ORA   ($D0),Y
```

Oops, there's another one. $0706 is a branch to $070B, which is shown in the middle of an instruction again.

```
*270A:EA
```

I'll spare you the gory details, but there are a few more of these. Just a few. 16. There are 16 more.

```
*2713:EA
*271B:EA
*2722:EA
*272A:EA
*2731:EA
*2739:EA
*2740:EA
*274A:EA
*2751:EA
*2754:EA
*275E:EA
*2766:EA
*276E:EA
*2776:EA
*277D:EA
*2786:EA
```

Here's the final, as-unobfuscated-as-I-can-make-it copy protection routine:

```
*26FAL

; turn on drive motor
26FA-   BD 89 C0    LDA   $C089,X
```

```
; set up Death Counters
26FD-   A9 56       LDA   #$56
26FF-   85 11       STA   $11
2701-   D0 01       BNE   $2704
2703-   EA          NOP

; this is actually uninitialized, but
; it doesn't much matter because it's
; just the low byte of the 16-bit Death
; Counter
2704-   C6 12       DEC   $12
2706-   F0 03       BEQ   $270B
2708-   D0 0A       BNE   $2714
270A-   EA          NOP
270B-   C6 11       DEC   $11
270D-   D0 05       BNE   $2714

; if Death Counter hits 0, set A=01 and
; exit
270F-   A9 01       LDA   #$01
2711-   D0 74       BNE   $2787
2713-   EA          NOP
```

```
; find standard address prologue
; (D5 AA 96)
2714-   BD 8C C0    LDA   $C08C,X
2717-   10 FB       BPL   $2714
2719-   D0 01       BNE   $271C
271B-   EA          NOP
271C-   C9 D5       CMP   #$D5
271E-   F0 03       BEQ   $2723
2720-   D0 E2       BNE   $2704
2722-   EA          NOP
2723-   BD 8C C0    LDA   $C08C,X
2726-   10 FB       BPL   $2723
2728-   D0 01       BNE   $272B
272A-   EA          NOP
272B-   C9 AA       CMP   #$AA
272D-   F0 03       BEQ   $2732
272F-   D0 D3       BNE   $2704
2731-   EA          NOP
2732-   BD 8C C0    LDA   $C08C,X
2735-   10 FB       BPL   $2732
2737-   D0 01       BNE   $273A
2739-   EA          NOP
273A-   C9 96       CMP   #$96
273C-   F0 03       BEQ   $2741
273E-   D0 C4       BNE   $2704
2740-   EA          NOP

; skip over an $FF nibble
2741-   A0 0A       LDY   #$0A
2743-   BD 8C C0    LDA   $C08C,X
2746-   10 FB       BPL   $2743
2748-   D0 01       BNE   $274B
274A-   EA          NOP
274B-   C9 FF       CMP   #$FF
274D-   D0 03       BNE   $2752
274F-   F0 B3       BEQ   $2704
2751-   EA          NOP
2752-   F0 01       BEQ   $2755
2754-   EA          NOP
```

```
; Read data latch exactly once (no BPL
; loop here!) and make sure its value
; is correct. We're out of sync here
; because of all the branching, so the
; exact value of the data latch depends
; on timing bit after the $FF nibble.
; This is where my EDD bit copy failed.
2755-   BD 8C C0    LDA   $C08C,X
2758-   C9 A5       CMP   #$A5
275A-   D0 A8       BNE   $2704
275C-   F0 01       BEQ   $275F
275E-   EA          NOP

; calculate a checksum on the following
; nibbles
275F-   BD 8C C0    LDA   $C08C,X
2762-   10 FB       BPL   $275F
2764-   D0 01       BNE   $2767
2766-   EA          NOP
2767-   85 10       STA   $10
2769-   88          DEY
276A-   D0 03       BNE   $276F
276C-   F0 10       BEQ   $277E
276E-   EA          NOP
276F-   BD 8C C0    LDA   $C08C,X
2772-   10 FB       BPL   $276F
2774-   D0 01       BNE   $2777
2776-   EA          NOP
2777-   45 10       EOR   $10
2779-   D0 EC       BNE   $2767
277B-   F0 EA       BEQ   $2767
277D-   EA          NOP

; final checksum must be $60
277E-   A5 10       LDA   $10
2780-   49 60       EOR   #$60
2782-   D0 80       BNE   $2704
2784-   F0 01       BEQ   $2787
2786-   EA          NOP
```

```
; on exit, A=00 on success (after
; falling through) or 01 on failure
; (coming from $0711)
2787-   60          RTS
```

This routine always returns to the caller, and the accumulator indicates success (0) or failure (1).

At $06C6, I can change the "ROL" to a "CLC". Then the "BCC" at $06C7 will unconditionally branch to $06F8, as if the protection routine had passed.

```
T00,S03,$C6 change 2A to 18

]PR#6
...offers input selection, then hangs
   with the drive motor on...
```

That's actually a lot of progress. I can now calibrate my joystick before the game tells me to go f--- myself.

Chapter 7
In Which I'd Like To Add You To My Professional Network Of Linked Catalog Sectors

Let's go back to that "unreadable" sector on track $11, the one that looks exactly like a DOS 3.3 catalog sector, because it is a DOS 3.3 catalog sector. It's just not linked into the VTOC.

T11,S00 currently points to T11,S0F as the first catalog sector, but all of the "files" in that catalog sector are fake. What if I changed it to point to sector $07 instead?

T11,S00,$02 change 0F to 07

```
]PR#5
...
]CATALOG,S6,D1

C1983 DSR^C#254
324 FREE

 A 002 HELLO
 B 181 BURGERTIME
 B 038 MDSADJ
```

Well would you look at that.

```
]BRUN BURGERTIME
...crashes...
```

It was worth a shot.

```
]PR#5
...
]BLOAD BURGERTIME,S6,D1
```

This actually works, but only because my work disk is running Diversi-DOS 64K which relocates most of DOS to the language card. The file starts at $0C00 and is $B200 in length. Whatever is at $0C00 isn't code, which explains why I couldn't just BRUN it. But this is definitely (at least part of) the game code, because switching to the hi-res graphics screen shows the game's title screen.

```
]BRUN MDSADJ,S6,D1
...displays input selection screen,
   allows me to actually calibrate my
   joystick, then hangs with the drive
   motor on...
```

These files are definitely not fake. The bootloader is actually using this (previously hidden, but well-formed) catalog sector to load the input selection routine (MDSADJ), then doing it again to load the actual game. But something is getting stuck between those two, so I need to look at MDSADJ to find out what's going on.

```
]BLOAD MDSADJ
]CALL -151

*BF55.BF56

BF55- 00 64    ; start address
```

```
*6400L

6400-   A9 9D       LDA   #$9D    ;A=9D
6402-   85 50       STA   $50     ; =9D
6404-   D0 01       BNE   $6407   ;yes
6406-   75 A9       ADC   $A9,X
6408-   86 A2       STX   $A2
640A-   00          BRK
```

Right out of the gate, I can tell this is going to be fun(*). The branch at $6404 jumps into the middle of the next instruction, which confuses the monitor disassembler.

```
*6407L

6407-   A9 86       LDA   #$86    ;A=86
6409-   A2 00       LDX   #$00    ;X=00
640B-   F0 01       BEQ   $640E   ;yes
640D-   24 95       BIT   $95
```

And again.

```
*640EL

640E-   95 21       STA   $21,X
6410-   86 20       STX   $20
```

Now ($20) points to $8600.

```
6412-   A9 01       LDA   #$01    ;A=01
6414-   D0 01       BNE   $6417   ;yes
6416-   D0 6C       BNE   $6484
```

And again.

(*) not guaranteed, actual fun may vary

```
*6417L

6417-   6C 20 00    JMP   ($0020) ;8600

*8600L

; decrypt everything in the file we
; just loaded ($6400..$85FF)
8600-   A2 22       LDX   #$22
8602-   A0 00       LDY   #$00
8604-   B9 00 85    LDA   $8500,Y
8607-   59 00 84    EOR   $8400,Y
860A-   49 01       EOR   #$01
860C-   99 00 85    STA   $8500,Y
860F-   C8          INY
8610-   D0 F2       BNE   $8604
8612-   CE 06 86    DEC   $8606
8615-   CE 09 86    DEC   $8609
8618-   CE 0E 86    DEC   $860E
861B-   CA          DEX
861C-   D0 E6       BNE   $8604

; and continue with decrypted code
861E-   4C 1D 64    JMP   $641D
```

Luckily (for me), this decryption loop is self-contained and does not rely on itself as a decryption key. I should be able to put an "RTS" at $861E and run it from the monitor.

```
*861E:60
*8600G
*641DL

641D-   8D 5A DB    STA   $DB5A
6420-   49 70       EOR   #$70
6422-   E8          INX
6423-   43          ???
6424-   4F          ???
6425-   C9 28       CMP   #$28
6427-   BF          ???
```

I've missed something.

Wait. On the last pass, it's decrypting $6400..$64FF by EOR'ing it with the page under that, which is $6300..$63FF. Which is not part of this file. Which means that this decryption loop is not self-contained after all; it depends on the value of the page at $6300.

Backtracking the boot, the only thing that ever touched $6300 was the routine at $040A that wiped all of main memory with zeroes.

```
*BLOAD MDSADJ
*6300:00 N 6301<6300.63FEM
*861E:60
*8600G
*641DL

641D-   20 5D 64    JSR   $645D
6420-   20 00 65    JSR   $6500
6423-   20 2C 64    JSR   $642C
6426-   20 00 85    JSR   $8500
6429-   4C B0 68    JMP   $68B0
642C-   A6 2B       LDX   $2B
642E-   BD 8E C0    LDA   $C08E,X
6431-   BD 8C C0    LDA   $C08C,X
6434-   BD 89 C0    LDA   $C089,X
```

Bingo.

```
*BSAVE MDSADJ DECRYPTED,A$6400,L$2400,
 S5,D1
```

Chapter 8
In Which We Discover A Decryption Most Foul (Again)

```
Continuing from $641D...

*641DL

641D-   20 5D 64    JSR   $645D

*645DL

; wipe language card
645D-   AD 83 C0    LDA   $C083
6460-   AD 83 C0    LDA   $C083
6463-   A0 00       LDY   #$00
6465-   84 00       STY   $00
6467-   A9 D0       LDA   #$D0
6469-   85 01       STA   $01
646B-   91 00       STA   ($00),Y
646D-   C8          INY
646E-   D0 FB       BNE   $646B
6470-   E6 01       INC   $01
6472-   D0 F7       BNE   $646B

; back to ROM
6474-   AD 81 C0    LDA   $C081

; and over here, then out
6477-   20 80 64    JSR   $6480
647A-   60          RTS
```

```
*6480L

; Scan all peripheral (slot) ROMs, test
; whether the first byte of the slot
; ROM changes over a short period of
; time, and if not, save it to an array
; at $BF73. (I'm assuming this is done
; again later, and if the values don't
; match, the game knows that it's not
; running on the same machine -- i.e.
; that someone used a memory capture
; card to save the game code to disk
; then reload it without going through
; this bootloader.)
6480-   A2 07       LDX   #$07
6482-   A9 00       LDA   #$00
6484-   8D 9B 64    STA   $649B
6487-   8D 9E 64    STA   $649E
648A-   8A          TXA
648B-   18          CLC
648C-   69 C0       ADC   #$C0
648E-   8D 9C 64    STA   $649C
6491-   8D 9F 64    STA   $649F
6494-   A9 0A       LDA   #$0A
6496-   85 A3       STA   $A3
6498-   A0 00       LDY   #$00
649A-   AD 00 C0    LDA   $C000
649D-   CD 00 C0    CMP   $C000
64A0-   D0 0E       BNE   $64B0
64A2-   88          DEY
64A3-   D0 F8       BNE   $649D
64A5-   C6 A3       DEC   $A3
64A7-   D0 EF       BNE   $6498
64A9-   9D 73 BF    STA   $BF73,X
64AC-   CA          DEX
64AD-   D0 D3       BNE   $6482
64AF-   60          RTS
64B0-   A9 00       LDA   #$00
64B2-   4C A9 64    JMP   $64A9
```

Since I'm planning to retain this entire bootloader (albeit without the decrypt-y bits or the protect-y bits), I won't bother to disable it now. But if it turns out I can't retain the bootloader, I'll need to revisit it.

Continuing from $6420...

```
*6420L

; (not shown) This is the actual input
; selection routine that sets joystick
; or keyboard mode. It returns when
; the user has made a selection or time
; has run out.
6420-   20 00 65    JSR   $6500
```

Once that returns, things suddenly get a lot more interesting (for me, at least).

```
6423-   20 2C 64    JSR   $642C

*642CL

; turn on drive motor
642C-   A6 2B       LDX   $2B
642E-   BD 8E C0    LDA   $C08E,X
6431-   BD 8C C0    LDA   $C08C,X
6434-   BD 89 C0    LDA   $C089,X
```

```
; clear hi-res screen 2
6437-   A9 00       LDA   #$00
6439-   85 00       STA   $00
643B-   A0 40       LDY   #$40
643D-   84 01       STY   $01
643F-   A8          TAY
6440-   91 00       STA   ($00),Y
6442-   C8          INY
6443-   D0 FB       BNE   $6440
6445-   E6 01       INC   $01
6447-   A6 01       LDX   $01
6449-   E0 60       CPX   #$60
644B-   D0 F3       BNE   $6440

; and show it (blank)
644D-   AD 50 C0    LDA   $C050
6450-   AD 52 C0    LDA   $C052
6453-   AD 55 C0    LDA   $C055
6456-   AD 57 C0    LDA   $C057

; clear keyboard strobe
6459-   AD 10 C0    LDA   $C010
645C-   60          RTS

Continuing from $6426...

*6426L

6426-   20 00 85    JSR   $8500
```

```
*8500L

; oh God, here we go again
8500-   A9 00       LDA   #$00    ;A=00
8502-   85 82       STA   $82     ; =00

; save stack pointer again
8504-   BA          TSX
8505-   8A          TXA
8506-   49 BC       EOR   #$BC
8508-   85 50       STA   $50

850A-   29 00       AND   #$00    ;A=00
850C-   AA          TAX           ;X=00
850D-   A0 D6       LDY   #$D6    ;Y=D6
850F-   49 3A       EOR   #$3A    ;A=3A
8511-   84 81       STY   $81     ; =D6
```

Now ($81) points to $00D6. That address was set to $5C (at $0816) and has never been touched since.

```
; X=00, so this is taking ($81), which
; is zp$D6, which is $5C, then EOR'ing
; it with A, which is $3A
8513-   41 81       EOR   ($81,X) ;A=66
8515-   85 01       STA   $01     ; =66
```

Now ($00) points to $6600.

```
8517-   98          TYA           ;A=D6
8518-   49 D2       EOR   #$D2    ;A=04
851A-   85 81       STA   $81     ; =04
```

Now ($81) points to $0004.

```
851C-   49 04       EOR   #$04    ;A=00
```

```
; ($81) points to $0004 and X=00, so
; this sets zp$04 to $00.
851E-   81 81        STA   ($81,X) ; =00
8520-   E6 81        INC   $81
```

Now ($81) points to $0005.

```
8522-   38           SEC
8523-   E9 7D        SBC   #$7D    ;A=83
8525-   10 F0        BPL   $8517   ;nope

; ($81) points to $0005 and X=00, so
; this sets zp$05 to $83, so now ($04)
; points to $8300.
8527-   81 81        STA   ($81,X) ; =83
8529-   49 83        EOR   #$83    ;A=00
852B-   85 00        STA   $00     ; =00
852D-   85 51        STA   $51     ; =00
852F-   49 85        EOR   #$85    ;A=85
8531-   F0 02        BEQ   $8535   ;nope
8533-   85 03        STA   $03     ; =85
8535-   8A           TXA           ;A=00
8536-   85 02        STA   $02     ; =00
```

Now ($02) points to $8500.

```
8538-   18           CLC
8539-   69 D6        ADC   #$D6    ;A=D6
853B-   85 81        STA   $81     ; =86
```

Now ($81) points to $00D6 again.

```
853D-   4A          LSR            ;A=6B
853E-   AA          TAX            ;X=6B

; $16 + $6B = $81, and ($81) points to
; $00D6, and zp$D6 is still $5C.
853F-   A1 16       LDA   ($16,X)  ;A=5C
8541-   49 5C       EOR   #$5C     ;A=00
8543-   A8          TAY            ;Y=00
8544-   49 04       EOR   #$04     ;A=04
8546-   F0 E5       BEQ   $852D    ;nope
8548-   85 08       STA   $08      ; =04
854A-   49 7A       EOR   #$7A     ;A=7E
854C-   88          DEY            ;Y=FF

; EOR with ($02),Y = $8500,Y = $85FF
854D-   51 02       EOR   ($02),Y

; EOR with ($04),Y = $8300,Y = $83FF
854F-   51 04       EOR   ($04),Y

; SBC ($00),Y = $6600,Y = $66FF
8551-   38          SEC
8552-   F1 00       SBC   ($00),Y

; store it in ($04),Y = $8300,Y = $83FF
8554-   91 04       STA   ($04),Y

; done decrypting this page?
8556-   C0 00       CPY   #$00

; nope, branch back to finish it
8558-   D0 F2       BNE   $854C
```

That branch goes to $854C, which is a "DEY" instruction, so we're decrypting memory from the top down. This is the same basic pattern as the earlier decryption routine at $0421, which was equally insane.

```
855A-   84 81       STY   $81      ; =00

; decrement page count (initialized to
; 04 at $8548)
855C-   C6 08       DEC   $08

; jump forward if we're done decrypting
855E-   F0 05       BEQ   $8565

; otherwise decrement the target page
; and loop back to decrypt it
8560-   C6 05       DEC   $05
8562-   4C 4C 85    JMP   $854C
```

The first pass decrypts $83FF..$8300 against $85FF..$8500 and $66FF..$6600. The second pass decrypts $82FF..$8200; then $81FF..$8100; then $80FF..$8000. Then zp$08 hits 0 and we branch forward to $8565.

```
; get some byte from the stack
8565-   A6 50       LDX   $50
8567-   BD 00 01    LDA   $0100,X
856A-   85 03       STA   $03
856C-   49 47       EOR   #$47
856E-   60          RTS
```

This routine is almost self-contained. The only thing it relies on to be pre-initialized is zero page $D6, which must be $5C. Other than that, I can run it directly from the monitor.

```
*D6:5C N 8500G

*8000L

8000-   A1 A9       LDA   ($A9,X)
8002-   09 85       ORA   #$85
8004-   25 85       AND   $85
8006-   01 A9       ORA   ($A9,X)
8008-   00          BRK
8009-   85 24       STA   $24
800B-   85 83       STA   $83
800D-   85 00       STA   $00
800F-   20 C0 04    JSR   $04C0
```

That *almost* looks like real code.
Wait, I see it now.

```
*8001L

8001-   A9 09       LDA   #$09
8003-   85 25       STA   $25
8005-   85 01       STA   $01
8007-   A9 00       LDA   #$00
8009-   85 24       STA   $24
800B-   85 83       STA   $83
800D-   85 00       STA   $00
800F-   20 C0 04    JSR   $04C0
```

There we go.

```
*BSAVE MDSADJ DECRYPTED TWICE,A$6400,
 L$2400,S5,D1
```

Chapter 9
In Which We'll Think About It After, Like An After-Thought

```
Continuing from... um... $6429...

*6429L

6429-   4C B0 68    JMP   $68B0

*68B0L

; copy newly decrypted code to the text
; page (overwriting the original RWTS)
68B0-   A9 00       LDA   #$00
68B2-   85 00       STA   $00
68B4-   A9 04       LDA   #$04
68B6-   85 01       STA   $01
68B8-   A9 00       LDA   #$00
68BA-   85 02       STA   $02
68BC-   A9 80       LDA   #$80
68BE-   85 03       STA   $03
68C0-   A2 05       LDX   #$05
68C2-   A0 00       LDY   #$00
68C4-   B1 02       LDA   ($02),Y
68C6-   91 00       STA   ($00),Y
68C8-   C8          INY
68C9-   D0 F9       BNE   $68C4
68CB-   CA          DEX
68CC-   F0 07       BEQ   $68D5
68CE-   E6 03       INC   $03
68D0-   E6 01       INC   $01
68D2-   4C C4 68    JMP   $68C4

; execution continue here (from $68CC)
; and we set yet another reset vector
68D5-   A9 96       LDA   #$96
68D7-   8D F2 03    STA   $03F2
68DA-   A9 04       LDA   #$04
68DC-   8D F3 03    STA   $03F3
68DF-   49 A5       EOR   #$A5
68E1-   8D F4 03    STA   $03F4
```

```
; destroy the code we just decrypted
; and copied (so the only real version
; is on the text page, which would be
; destroyed if an evil hacker tried to
; break into the monitor right now)
68E4-   20 00 85    JSR   $8500

; and continue inside the new code
68E7-   4C C1 07    JMP   $07C1
```

Since $8000+ is copied to $0400+, $07C1 is currently at $83C1.

```
*83C1L

; I swear to God, it's another layer of
; encryption
83C1-   A2 00       LDX   #$00
83C3-   A0 03       LDY   #$03
83C5-   BD C0 04    LDA   $04C0,X
83C8-   49 A5       EOR   #$A5
83CA-   9D C0 04    STA   $04C0,X
83CD-   E8          INX
83CE-   D0 F5       BNE   $83C5
83D0-   EE C7 07    INC   $07C7
83D3-   EE CC 07    INC   $07CC
83D6-   88          DEY
83D7-   D0 EC       BNE   $83C5
83D9-   F0 01       BEQ   $83DC
```

I can reproduce this easily enough.

Rewriting it at $0300:

```
0300-   A2 00       LDX   #$00
0302-   A0 03       LDY   #$03
0304-   BD C0 80    LDA   $80C0,X
0307-   49 A5       EOR   #$A5
0309-   9D C0 80    STA   $80C0,X
030C-   E8          INX
030D-   D0 F5       BNE   $0304
030F-   EE 06 03    INC   $0306
0312-   EE 0B 03    INC   $030B
0315-   88          DEY
0316-   D0 EC       BNE   $0304
0318-   60          RTS

*300G

*BSAVE MDSADJ DECRYPTED THRICE,A$6400,
 L$2400,S5,D1
```

I'm beginning to suspect that this disk is nothing more than an infinite series of decryption routines with a game bolted on as an afterthought.

Chapter 10
In Which We Find Two Of Everything

Continuing from... um... $83DC I guess (branched from $83D9):

```
*83DCL

83DC-   A9 01       LDA   #$01     ;A=01
83DE-   85 36       STA   $36      ; =01
83E0-   A9 01       LDA   #$01     ;A=01
83E2-   38          SEC
83E3-   69 00       ADC   #$00     ;A=02
83E5-   F0 0E       BEQ   $83F5    ;nope
83E7-   10 03       BPL   $83EC    ;yes

83E9-   30 F0       BMI   $83DB    ;fake
83EB-   3D 2A 85    AND   $852A,X  ;fake
83EE-   37          ???            ;fake

*83ECL

83EC-   2A          ROL            ;A=04
83ED-   85 37       STA   $37      ; =04
```

Now ($36) points to $0401.

```
83EF-   D0 01       BNE   $83F2    ;yes
83F1-   6C 6C 36    JMP   ($366C)  ;fake
83F4-   00          BRK            ;fake
83F5-   4C 12 04    JMP   $0412    ;fake
83F8-   4C 00 04    JMP   $0400    ;fake
83FB-   4C 0F 06    JMP   $060F    ;fake

*83F2L

83F2-   6C 36 00    JMP   ($0036)
```

So execution continues at $0401.

The code that ends up at $0401 is currently in memory at $8001, so I'm going to leave it there and try not to get too confused.

```
*8001L

8001-   A9 09       LDA   #$09
8003-   85 25       STA   $25
8005-   85 01       STA   $01
8007-   A9 00       LDA   #$00
8009-   85 24       STA   $24
800B-   85 83       STA   $83
800D-   85 00       STA   $00
800F-   20 C0 04    JSR   $04C0
```

That's at $80C0.

```
*80C0L

80C0-   20 6F 85    JSR   $856F
```

That's actually at $856F. Weird. This second RWTS relies on MDSADJ still being in memory. So many interlocking dependencies...

```
*856FL

856F-   A5 51       LDA   $51      ;A=00
8571-   49 AA       EOR   #$AA     ;A=AA
8573-   49 AA       EOR   #$AA     ;A=00
8575-   85 36       STA   $36      ; =00
8577-   49 1C       EOR   #$1C     ; =1C
8579-   85 37       STA   $37      ; =1C
857B-   60          RTS
```

Now ($36) points to $1C00.

Continuing from $04C3, which is at $80C3...

```
*80C3L

; looks like we're going to read that
; catalog sector again (T11,S07)
80C3-   A9 11       LDA   #$11
80C5-   85 80       STA   $80
80C7-   A9 07       LDA   #$07
80C9-   85 81       STA   $81
80CB-   85 82       STA   $82
80CD-   A5 D7       LDA   $D7
80CF-   8D 01 02    STA   $0201
80D2-   20 C3 06    JSR   $06C3
```

That's at $82C3.

```
*82C3L

; (not shown) This is the sector read
; routine. It follows the same pattern
; as the first RWTS -- moving the drive
; head, reading a sector. But it has
; its own disk read routine, which
; explains why my copy hung with the
; drive motor on. It's actually trying
; to read T11,S07 and failing because
; it's looking for the protected data
; prologue ("D5 AA AA") again.
82C3-   20 FA 04    JSR   $04FA
```

The new sector read routine starts at $0604, which is in memory at $8204.

```
*8204L

8204-   18          CLC
8205-   08          PHP
8206-   BD 8C C0    LDA   $C08C,X
8209-   10 FB       BPL   $8206
820B-   49 D5       EOR   #$D5
820D-   D0 F7       BNE   $8206
820F-   BD 8C C0    LDA   $C08C,X
8212-   10 FB       BPL   $820F
8214-   C9 AA       CMP   #$AA
8216-   D0 F3       BNE   $820B
8218-   EA          NOP
8219-   BD 8C C0    LDA   $C08C,X
821C-   10 FB       BPL   $8219
821E-   C9 96       CMP   #$96
8220-   F0 09       BEQ   $822B
8222-   28          PLP
8223-   90 DF       BCC   $8204
8225-   49 AA       EOR   #$AA    <-- !
8227-   F0 25       BEQ   $824E
8229-   D0 D9       BNE   $8204
```

That $AA at $0626 needs to be changed to $AD now that my disk uses standard data prologues. (This explains why my crack-in-progress hung with the drive motor on -- it was trying to read the disk with this second RWTS.)

Continuing from $06C6 (after the call to $04FA returns)...

```
*82C6L

82C6-   A6 2B       LDX   $2B
82C8-   20 EE 06    JSR   $06EE
82CB-   29 01       AND   #$01
82CD-   AA          TAX
82CE-   BD D4 06    LDA   $06D4,X
82D1-   F0 05       BEQ   $82D8
82D3-   60          RTS
82D4-   20 00 64    JSR   $6400
82D7-   4C A9 8F    JMP   $8FA9
```

Lots going on here. I'll look at $06EE in a minute, but check out what we're doing with the return code. If the low bit of the accumulator is 0, X will end up being 0 and we'll read $06D4 (=$20), not take the "BEQ" branch, and exit via the "RTS" at $06D3. But if the low bit of the accumulator is 1, X will end up being 1 and we'll read $06D5 (=$00), take the branch to $06D8, and end up... doing what exactly?

```
*82D8L

82D8-   A9 8F       LDA   #$8F     ;A=8F
82DA-   18          CLC
82DB-   69 07       ADC   #$07     ;A=96
82DD-   85 36       STA   $36      ; =96
82DF-   A9 10       LDA   #$10     ;A=10
82E1-   38          SEC
82E2-   E9 0C       SBC   #$0C     ;A=04
82E4-   85 37       STA   $37      ; =04
```

Now ($36) points to $0496.

```
82E6-   2A          ROL            ;A=09
82E7-   2A          ROL            ;A=12
82E8-   85 39       STA   $39      ; =12
82EA-   6C 36 00    JMP   ($0036)
```

So we jump to $0496, which is in memory at $8096.

```
*8096L

8096-   A0 05       LDY   #$05
8098-   20 A2 04    JSR   $04A2

*80A2L

; wipe all memory, starting at the page
; given in A (=$05)
80A2-   84 01       STY   $01
80A4-   A9 00       LDA   #$00
80A6-   85 00       STA   $00
80A8-   A8          TAY
80A9-   91 00       STA   ($00),Y
80AB-   C8          INY
80AC-   91 00       STA   ($00),Y
80AE-   C8          INY
80AF-   91 00       STA   ($00),Y
80B1-   C8          INY
80B2-   91 00       STA   ($00),Y
80B4-   C8          INY
80B5-   D0 F2       BNE   $80A9
80B7-   E6 01       INC   $01
80B9-   A6 01       LDX   $01
80BB-   E0 C0       CPX   #$C0
80BD-   D0 EA       BNE   $80A9
80BF-   60          RTS
```

So it appears that $06EE is a Very Important Routine, and it is vital that the low bit of the accumulator end up being 0 at the end of it. It's the same logic as the protection check! Or should I say, the *first* protection check, because it appears that there are two of them.

Four decryption loop.
Three hidden files.
Two nibble checks.
And a classic game on a floppy.

Merry Crackmas.

Chapter 11
And A Happy New Year

Let's go look at $06EE.

```
*82EEL

82EE-   BD 89 C0    LDA   $C089,X
82F1-   A9 56       LDA   #$56
82F3-   85 11       STA   $11
82F5-   D0 01       BNE   $82F8
82F7-   D0 C6       BNE   $82BF
82F9-   12          ???
```

If this were a job, I'd quit.

```
*82F7:EA
*82FE:EA
*8307:EA
*830F:EA
*8316:EA
*831E:EA
*8325:EA
*832D:EA
*8334:EA
*833E:EA
*8345:EA
*8348:EA
*8352:EA
*835A:EA
*8362:EA
*836A:EA
*8371:EA
*837A:EA
```

```
*82EEL

; turn on drive motor
82EE-   BD 89 C0    LDA   $C089,X

; initialize Death Counters
82F1-   A9 56       LDA   #$56
82F3-   85 11       STA   $11
82F5-   D0 01       BNE   $82F8
82F7-   EA          NOP
82F8-   C6 12       DEC   $12
82FA-   F0 03       BEQ   $82FF
82FC-   D0 0A       BNE   $8308
82FE-   EA          NOP
82FF-   C6 11       DEC   $11
8301-   D0 05       BNE   $8308

; if Death Counters hit 0, load A=FF
; and exit
8303-   A9 FF       LDA   #$FF
8305-   D0 7C       BNE   $8383
8307-   EA          NOP
```

```
; find standard address prologue
; (D5 AA 96)
8308-   BD 8C C0    LDA   $C08C,X
830B-   10 FB       BPL   $8308
830D-   D0 01       BNE   $8310
830F-   EA          NOP
8310-   C9 D5       CMP   #$D5
8312-   F0 03       BEQ   $8317
8314-   D0 E2       BNE   $82F8
8316-   EA          NOP
8317-   BD 8C C0    LDA   $C08C,X
831A-   10 FB       BPL   $8317
831C-   D0 01       BNE   $831F
831E-   EA          NOP
831F-   C9 AA       CMP   #$AA
8321-   F0 03       BEQ   $8326
8323-   D0 D3       BNE   $82F8
8325-   EA          NOP
8326-   BD 8C C0    LDA   $C08C,X
8329-   10 FB       BPL   $8326
832B-   D0 01       BNE   $832E
832D-   EA          NOP
832E-   C9 96       CMP   #$96
8330-   F0 03       BEQ   $8335
8332-   D0 C4       BNE   $82F8
8334-   EA          NOP

; skip over an $FF nibble
8335-   A0 0A       LDY   #$0A
8337-   BD 8C C0    LDA   $C08C,X
833A-   10 FB       BPL   $8337
833C-   D0 01       BNE   $833F
833E-   EA          NOP
833F-   C9 FF       CMP   #$FF
8341-   F0 03       BEQ   $8346
8343-   D0 B3       BNE   $82F8
8345-   EA          NOP
8346-   F0 01       BEQ   $8349
8348-   EA          NOP
```

```
; Read data latch exactly once (no BPL
; loop here!) and check its value.
; We're out of sync here because of all
; the branching, so the exact value of
; the data latch depends on timing bit
; after the $FF nibble. This is
; essentially the same technique as the
; first protection check, but done in a
; different way.
8349-   BD 8C C0    LDA   $C08C,X
834C-   C9 08       CMP   #$08
834E-   B0 A8       BCS   $82F8
8350-   D0 01       BNE   $8353
8352-   EA          NOP

; calculate a checksum on the following
; nibbles
8353-   BD 8C C0    LDA   $C08C,X
8356-   10 FB       BPL   $8353
8358-   D0 01       BNE   $835B
835A-   EA          NOP
835B-   85 10       STA   $10
835D-   88          DEY
835E-   D0 03       BNE   $8363
8360-   F0 10       BEQ   $8372
8362-   EA          NOP
8363-   BD 8C C0    LDA   $C08C,X
8366-   10 FB       BPL   $8363
8368-   D0 01       BNE   $836B
836A-   EA          NOP
836B-   45 10       EOR   $10
836D-   D0 EC       BNE   $835B
836F-   F0 EA       BEQ   $835B
8371-   EA          NOP

; final checksum must be $60
8372-   A5 10       LDA   $10
8374-   49 60       EOR   #$60
8376-   D0 80       BNE   $82F8
8378-   F0 01       BEQ   $837B
837A-   EA          NOP
```

```
; wipe this entire protection check
; from memory
837B-   A0 8E       LDY   #$8E
837D-   99 EE 06    STA   $06EE,Y
8380-   88          DEY
8381-   D0 FA       BNE   $837D

; on exit, A=$00 on success (after
; falling through) or $FF on failure
; (coming from $0705)
8383-   60          RTS
```

Revisiting the caller at $06C6...

```
*82C6L

; execute protection check
82C6-   A6 2B       LDX   $2B
82C8-   20 EE 06    JSR   $06EE

; get low bit
82CB-   29 01       AND   #$01

; 0=success, 1=failure
82CD-   AA          TAX
82CE-   BD D4 06    LDA   $06D4,X

; failure path branches to The Badlands
82D1-   F0 05       BEQ   $82D8

; success path returns to caller
82D3-   60          RTS
82D4-  [20 00]
```

To bypass this, I can change the "AND #$01" at $06CB to "AND #$00", to act as if the protection check always passes.

To sum up: MDSADJ decrypts itself three separate times, revealing a second RWTS and a second protection check. Now that I've decrypted it (three times), I can patch the second RWTS and disable the second protection check.

```
; jump to the start of the code that
; was decrypted by the first decryption
; loop (which has already been done)
*6400:4C 1D 64

; disable second decryption loop (also
; already done) by putting an "RTS" at
; the beginning of the routine at $8500
*8500:60

; skip over third decryption loop (also
; already done) by changing the "JMP"
; at $68E7 from $07C1 to $07DC
*68E8:DC

; patch the second RWTS to read the
; standard data prologue ("D5 AA AD"
; instead of "D5 AA AA")
*8226:AD

; defeat the second protection check by
; changing the post-check logic to
; claim that it always passes
*82CC:00

*BSAVE MDS PATCHED,A$6400,L$2400,S5,D1
```

But wait, there's more! I can also save the thrice-decrypted MDSADJ file to my cracked copy, with the same command. As long as the filename and length match, DOS will overwrite the existing file and update the existing record in the catalog sector. (This is important, since the bootloader is hard-coded to load the third file in the catalog.)

```
*BSAVE MDSADJ,A$6400,L$2400,S6,D1

*C600G
```

...works, and it is glorious...

Quod erat liberandum.

Epilogue
But What About The...

I realize that this write-up doesn't answer all the questions it raises. Sorry about that, but it turned out that some of my investigations were dead ends. Real life investigations are like that.

But I do want to revisit one in particular, the slot scan at $6480. As I suspected, the fingerprint that it creates (at $BF73+) is checked later in the game code proper. A quick sector search for "73 BF" found it; it's not encrypted or obfuscated. Changing the first instruction to "RTS" disables it completely.

```
T1A,S06
---------- DISASSEMBLY MODE -----------
009A:A2 07          LDX   #$07
009C:A9 00          LDA   #$00
009E:85 A4          STA   $A4
00A0:8A             TXA
00A1:18             CLC
00A2:69 C0          ADC   #$C0
00A4:85 A5          STA   $A5
00A6:A9 0A          LDA   #$0A
00A8:85 A3          STA   $A3
00AA:A0 00          LDY   #$00
00AC:84 A6          STY   $A6
00AE:B1 A4          LDA   ($A4),Y
00B0:D1 A4          CMP   ($A4),Y
00B2:D0 0D          BNE   $00C1
00B4:C6 A6          DEC   $A6
00B6:D0 F8          BNE   $00B0
00B8:DD 73 BF       CMP   $BF73,X
00BB:D0 09          BNE   $00C6
00BD:CA             DEX
00BE:D0 DC          BNE   $009C
00C0:60             RTS
00C1:BD 73 BF       LDA   $BF73,X
00C4:F0 F7          BEQ   $00BD
00C6:C6 A3          DEC   $A3
00C8:D0 E0          BNE   $00AA
00CA:A9 00          LDA   #$00
00CC:8D 00 02       STA   $0200
00CF:F0 01          BEQ   $00D2
00D1:[D0]
00D2:6C 00 02       JMP   ($0200)

To disable the slot scan verification:
T1A,S06,$9A change A2 to 60
```

I have not applied this patch to my crack, but other implementers who want to change the bootloader or launch the game from ProDOS might need to be aware of it.

Epilogue
Cheats

Anyone who played The Freeze's crack "back in the day" will remember the way you could get infinite lives and pepper by holding down <Ctrl-C> until the life and pepper indicators started going crazy.

Infinite lives:
T19,S03,$36 change 01 to 00

Infinite pepper:
T1A,S03,$D4 change 01 to 00

I have not enabled either of these cheats, but I have verified that they work. Thanks to qkumba for finding and testing these cheats.

Acknowledgements

Thanks to qkumba, John Brooks, and many others for reviewing drafts of this write-up.