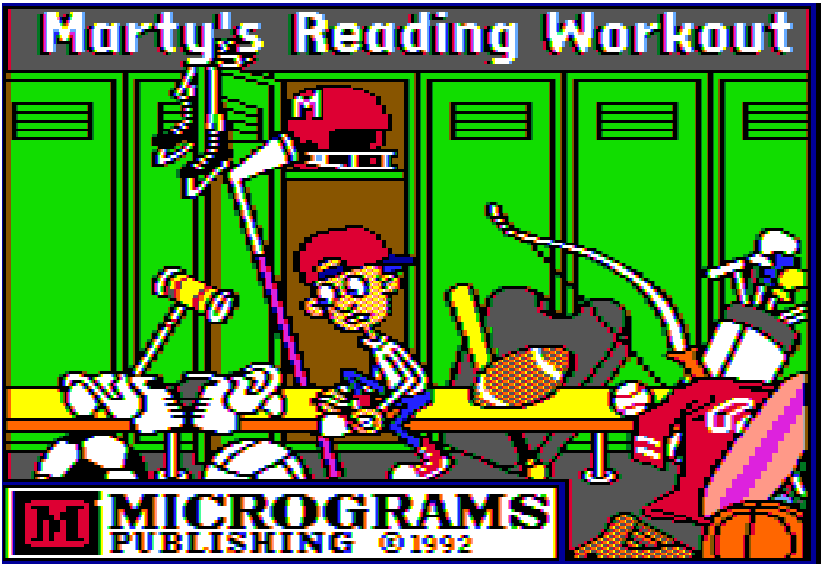
# Marty's Reading Workout



2016-08-12

4am
to deprotect
and preserve

# Contents

```
---------Marty's Reading Workout--------
A 4am crack                   2016-08-12
----------------------------------------

Name: Marty's Reading Workout
Genre: educational
Year: 1992
Publisher: Micrograms
Platform: Apple //e or later (128K)
Media: 6 single-sided 5.25-inch disks
OS: custom
Previous cracks: none
```

# Chapter 0
# In Which Various Automated Tools Fail In Interesting Ways

COPYA
  immediate disk read error

Locksmith Fast Disk Backup
  can't read track $00, sector $0C (same on all disks, so this is definitely intentional)
  copy loads title screen then breaks to text page with "ERROR D51"

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

Copy ][+ nibble editor
  T00,S0C exists (I searched for the raw nibble sequence "AA AB AE", which matches the second half of the track ("AA AA" -> $00) and the sector ("AB AE" -> $06 = logical sector $0C) in the address field

Disk Fixer
  setting "CHECKSUM ENABLED" to "NO" allows me to read T00,S0C

Why didn't COPYA work?
intentionally corrupted sector on T00

Why didn't Locksmith FDB work?
probably a run-time check to ensure
that sector on T00 is corrupted,
which it isn't, on my copy, because
Locksmith Fast Disk Backup will just
write out a standard sector of zeroes
instead of reproducing the corruption

EDD worked. What does that tell us?
Probably just a bad block check:
unreadable sector = original,
readable sector = unauthorized copy

Next steps:

1. Use a sector editor to search for
   obvious signs of sector reads
2. If that fails, trace the boot
3. I don't know, go feed the ducks or
   something

# Chapter 1
# It's Only Metadata

The disk appears to boot directly to the program, without loading any known operating system first. But while I was poking around the corrupted track 0, I noticed a normal ProDOS catalog. And in fact, I can boot from my ProDOS hard drive and catalog this disk!

--v--

```
]PR#7
...
]CAT,S6,D1

/BOOT

 NAME           TYPE  BLOCKS  MODIFIED

 PRODOS          SYS       7  <NO DATE>
 MAIN2.SCR       BIN      33  <NO DATE>
 LOWDOS          BIN       6  <NO DATE>
 FONT1.DHR       BIN       7  <NO DATE>
 UTL             BIN       8  <NO DATE>
 MAIN            BIN      36  <NO DATE>
 CR.WINDOWS      BIN      16  <NO DATE>
 CR.TEXTA        BIN       9  <NO DATE>
 CR.WINDOWSA     BIN      38  <NO DATE>
 CR.QUESTIONSA   BIN      16  <NO DATE>
 CR.TEXTB        BIN       9  <NO DATE>
 CR.WINDOWSB     BIN      40  <NO DATE>
 CR.QUESTIONSB   BIN      15  <NO DATE>
 PRINTUTL        BIN       7  <NO DATE>

BLOCKS FREE:   26     BLOCKS USED:  254
```

```
]CATALOG,S6,D1

[truncated here to show the final
 column, which is the load address of
 each file]

/BOOT

 NAME           TYPE  BLOCKS  ...SUBTYPE

 PRODOS          SYS       7
 MAIN2.SCR       BIN      33  ...A=$2000
 LOWDOS          BIN       6  ...A=$0800
 FONT1.DHR       BIN       7  ...A=$F000
 UTL             BIN       8  ...A=$E000
 MAIN            BIN      36  ...A=$4000
 CR.WINDOWS      BIN      16  ...A=$4000
 CR.TEXTA        BIN       9  ...A=$2003
 CR.WINDOWSA     BIN      38  ...A=$4000
 CR.QUESTIONSA   BIN      16  ...A=$2003
 CR.TEXTB        BIN       9  ...A=$2003
 CR.WINDOWSB     BIN      40  ...A=$4000
 CR.QUESTIONSB   BIN      15  ...A=$2003
 PRINTUTL        BIN       7  ...A=$9400

BLOCKS FREE:   26 ...TOTAL BLOCKS:  280

]
```

--^--

That "PRODOS" file is suspiciously small, though. A normal ProDOS is 4x that size. And "LOWDOS" sounds interesting.

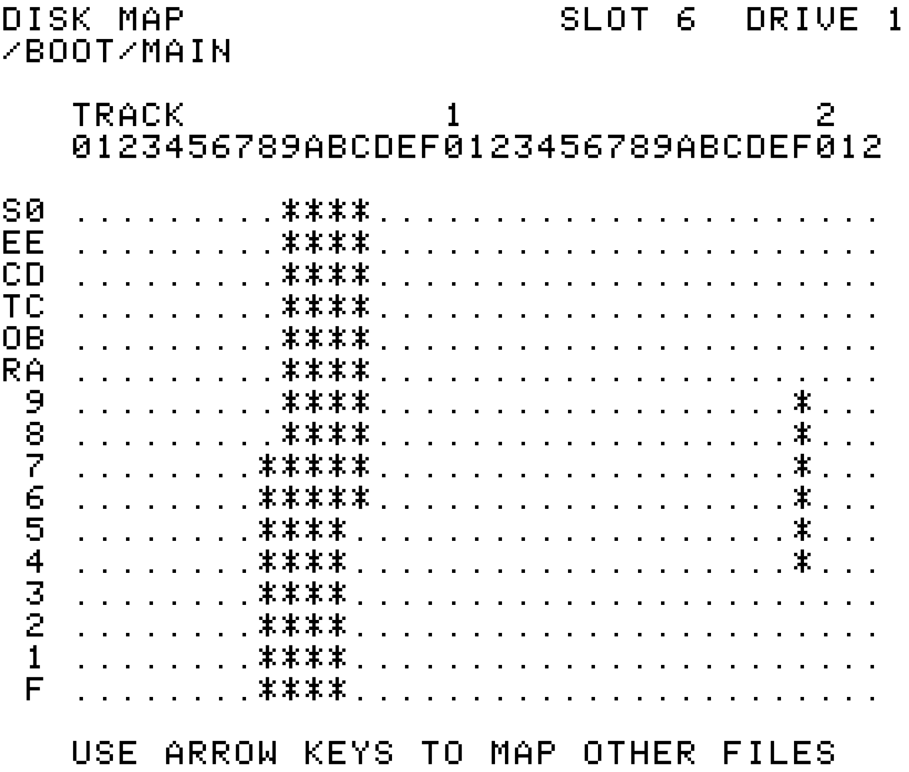
Anyway, might prove useful, especially being able to cross-reference sectors to files and finding out where they're loaded in memory. The SUBTYPE metadata seems too non-random to be completely unused.

Onward!

My non-working copy prints an error message. Let's see if we can find it. Turning to my trusty Disk Fixer sector editor, I search for the ASCII string "ERROR D51" and find it in T0A,S00!

Copy II Plus recognizes this disk as ProDOS, and the "disk map" says that T0A,S00 is part of the somewhat fragmented file "MAIN".

```
                 --v--
```



```
DISK MAP                SLOT 6  DRIVE 1
/BOOT/MAIN

   TRACK           1               2
   0123456789ABCDEF0123456789ABCDEF012

S0 .........****......................
EE .........****......................
CD .........****......................
TC .........****......................
OB .........****......................
RA .........****......................
 9 .........****..................*...
 8 .........****..................*...
 7 ........*****..................*...
 6 ........*****..................*...
 5 ........****...................*...
 4 ........****...................*...
 3 ........****.......................
 2 ........****.......................
 1 ........****.......................
 F ........****.......................

   USE ARROW KEYS TO MAP OTHER FILES
```

```
                 --^--
```

Booting my ProDOS hard drive, I can BLOAD that file into memory and start tracing. According to the full CATALOG command (not shown), the file "MAIN" is loaded at address $4000.

```
]PR#7
...
]PREFIX /BOOT
]BLOAD MAIN
]CALL -151

*4000L

4000-   4C 22 41    JMP   $4122

*4122L

4122-   20 8C 48    JSR   $488C

*488CL

; zero page initialization (not shown)
488C-   20 F9 48    JSR   $48F9

; initializes and displays the double-
; hi-res title page (not shown)
488F-   20 56 49    JSR   $4956
```

The rest of the subroutine clears some chunks of main memory and does other uninteresting (un-disk-related) things. My non-working copy got as far as showing the double hi-res title screen, so I don't think I've found the copy protection yet.

Popping the stack and continuing from $4125...

```
; read/write RAM bank 1
4125-   AD 8B C0    LDA   $C08B
4128-   AD 8B C0    LDA   $C08B
412B-   8D 08 C0    STA   $C008

; could be anything, but given the
; current program counter, I'm guessing
; this is the address $4143, which is
; directly below
412E-   A9 43       LDA   #$43
4130-   8D 07 08    STA   $0807
4133-   A9 41       LDA   #$41
4135-   8D 08 08    STA   $0808

; don't know
4138-   A9 01       LDA   #$01
413A-   8D 69 0A    STA   $0A69

; don't know
413D-   20 03 08    JSR   $0803
4140-   4C A6 56    JMP   $56A6
```

```
; don't know, but it's the same address
; we set at $413A (above)
4143-   A9 02       LDA   #$02
4145-   8D 69 0A    STA   $0A69
4148-   20 8C 48    JSR   $488C
414B-   20 11 5A    JSR   $5A11
414E-   A2 22       LDX   #$22
4150-   20 4B E0    JSR   $E04B
4153-   2C 10 C0    BIT   $C010
```

OK, one thing at a time. We're setting some parameters and calling a routine at $0803. What's at $0803? According to the ProDOS metadata, "LOWDOS" is loaded at $0800. Now we get to see what the heck "LOWDOS" is.

# Chapter 2
# When They Go Low, We Go High

```
*BLOAD LOWDOS
*803L

0803-   4C 51 0A    JMP   $0A51

*A51L

0A51-   A2 08       LDX   #$08
0A53-   A9 00       LDA   #$00
0A55-   9D 1E 08    STA   $081E,X
0A58-   CA          DEX
0A59-   10 FA       BPL   $0A55

; turn on slot 6 drive motor
0A5B-   2C E9 C0    BIT   $C0E9
0A5E-   A9 01       LDA   #$01
0A60-   8D 28 08    STA   $0828
0A63-   A9 00       LDA   #$00
0A65-   8D 29 08    STA   $0829
0A68-   A9 02       LDA   #$02
0A6A-   8D 3A 08    STA   $083A
0A6D-   A9 00       LDA   #$00
0A6F-   8D 3B 08    STA   $083B
0A72-   A9 0D       LDA   #$0D
0A74-   A0 00       LDY   #$00
0A76-   20 3C 08    JSR   $083C

*83CL

083C-   8D A6 09    STA   $09A6
083F-   8C A5 09    STY   $09A5

; memory fiddling (not shown)
0842-   20 AB 09    JSR   $09AB
```

```
; do something
0845-   AE 38 08    LDX   $0838
0848-   BD 28 10    LDA   $1028,X
084B-   20 65 08    JSR   $0865     (1)

; increment something
084E-   EE A6 09    INC   $09A6

; and do the same thing again, but
; differently
0851-   AE 38 08    LDX   $0838
0854-   BD 30 10    LDA   $1030,X
0857-   20 65 08    JSR   $0865     (2)

*1028.

1028- 00 04 08 0C 01 05 09 0D
1030- 02 06 0A 0E 03 07 0B 0F
```

OK, I'm beginning to see what's going on here. This routine looks like it's loading a ProDOS "block" -- two consecutive sectors on disk, where by "consecutive," I mean "consecutive in the ProDOS skewing order." $0838 holds the index into the 8-item arrays at $1028 and $1030, which map logical to physical sectors.

If I'm right, that means that $0865 is the main entry point to read a sector from disk.

```
*865L

0865-   85 EC       STA   $EC

; reset data latch
0867-   AD EE C0    LDA   $C0EE
086A-   A9 03       LDA   #$03
086C-   8D 27 08    STA   $0827
086F-   20 84 08    JSR   $0884

*884L

; set up death counter
0884-   A9 00       LDA   #$00
0886-   8D 83 08    STA   $0883
0889-   CE 83 08    DEC   $0883
088C-   D0 03       BNE   $0891

; if death counter hits 0, JSR(?!) here
; (more on this later)
088E-   20 78 09    JSR   $0978

; another death counter
0891-   A9 00       LDA   #$00
0893-   85 FC       STA   $FC
0895-   88          DEY
0896-   D0 07       BNE   $089F
0898-   C6 FC       DEC   $FC
089A-   D0 03       BNE   $089F

; and again, if that death counter hits
; 0, JSR to the same place as $088E
089C-   20 78 09    JSR   $0978
```

I'm beginning to suspect that $0978 doesn't ever return, but we'll get to that in a minute.

```
; find address prologue (D5 AA 96)
089F-   AD EC C0    LDA   $C0EC
08A2-   10 FB       BPL   $089F
08A4-   C9 D5       CMP   #$D5
08A6-   D0 ED       BNE   $0895
08A8-   AD EC C0    LDA   $C0EC
08AB-   10 FB       BPL   $08A8
08AD-   C9 AA       CMP   #$AA
08AF-   D0 EE       BNE   $089F
08B1-   AD EC C0    LDA   $C0EC
08B4-   10 FB       BPL   $08B1
08B6-   C9 96       CMP   #$96
08B8-   D0 E5       BNE   $089F

; parse address field, store in $0834+
08BA-   A0 03       LDY   #$03
08BC-   A9 00       LDA   #$00
08BE-   8D 33 08    STA   $0833
08C1-   AD EC C0    LDA   $C0EC
08C4-   10 FB       BPL   $08C1
08C6-   2A          ROL
08C7-   85 F9       STA   $F9
08C9-   AD EC C0    LDA   $C0EC
08CC-   10 FB       BPL   $08C9
08CE-   25 F9       AND   $F9
08D0-   99 34 08    STA   $0834,Y
08D3-   4D 33 08    EOR   $0833
08D6-   88          DEY
08D7-   10 E5       BPL   $08BE
08D9-   A8          TAY
08DA-   D0 AD       BNE   $0889
08DC-   AD 36 08    LDA   $0836
08DF-   C5 EB       CMP   $EB
```

```
; success path branches (if address
; field checksum verifies)
08E1-   F0 0B       BEQ   $08EE

; failure path -- recalibrate the drive
; and try again to find the right track
; (not shown)
08E3-   0A          ASL
08E4-   85 ED       STA   $ED
08E6-   A5 EB       LDA   $EB
08E8-   20 DF 09    JSR   $09DF
08EB-   4C 89 08    JMP   $0889

; execution continues here (from $08E1)
; check if we got the sector we wanted,
; otherwise branch back and try again
08EE-   AD 35 08    LDA   $0835
08F1-   C5 EC       CMP   $EC
08F3-   D0 94       BNE   $0889
08F5-   60          RTS
```

Continuing from $0872...

```
; read data field (prologue, data, and
; epilogue -- not shown, but it sets
; the carry on failure and clears it on
; success, like DOS 3.3)
0872-   20 F6 08    JSR   $08F6

; branch forward on success
0875-   90 08       BCC   $087F

; decrement death counter and try again
0877-   CE 27 08    DEC   $0827
087A-   D0 F3       BNE   $086F
```

```
; once again, we end up calling $0978
; after the death counter hits 0
087C-   20 78 09     JSR   $0978

; execution continues here (from $087A)
; this routine finishes the nibble-to-
; byte conversion of the raw nibbles
; that were read earlier in $08F6
; (not shown)
087F-   20 92 09     JSR   $0992
0882-   60           RTS
```

So... we're reading sectors, more or less the same way that DOS 3.3 reads sectors. The strangest part is that any fatal error ends up JSR'ing to $0978. What's at $0978?

```
*978L

; turn off slot 6 drive motor
0978-   2C E8 C0     BIT   $C0E8

; reset stack pointer (so I was right,
; this routine never returns to the
; caller)
097B-   A2 FF        LDX   #$FF
097D-   9A           TXS

; jump to "fatal error" vector
097E-   4C 06 08     JMP   $0806
```

But wait! We set that vector before calling LOWDOS -- all the way back at $412E:

```
412E-   A9 43       LDA   #$43
4130-   8D 07 08    STA   $0807
4133-   A9 41       LDA   #$41
4135-   8D 08 08    STA   $0808
```

Immediately after setting that fatal error vector, we called LOWDOS to read an unreadable block that spans T00,S0C:

```
413D-   20 03 08    JSR   $0803
```

And that's the key to this protection scheme: the "success" path routes through the fatal error vector at $0806 and continues to the start of the game at $4143. If LOWDOS doesn't encounter an error, it returns to... what? Well, the "JSR $0803" at $413D eventually returns gracefully, and we continue to the next instruction:

```
4140-   4C A6 56    JMP   $56A6
```

```
*56A6L
```

```
; wipe part of the code we came from
56A6-   A0 00       LDY   #$00
56A8-   A9 00       LDA   #$00
56AA-   99 22 41    STA   $4122,Y
56AD-   88          DEY
56AE-   D0 FA       BNE   $56AA
```

```
; reset memory vectors
56B0-   8D 08 C0    STA   $C008
56B3-   AD 83 C0    LDA   $C083
56B6-   AD 83 C0    LDA   $C083
56B9-   20 0F E0    JSR   $E00F
56BC-   8D 0C C0    STA   $C00C
56BF-   2C 51 C0    BIT   $C051
56C2-   20 15 E0    JSR   $E015

; display "ERROR D51" message at the
; bottom of the screen
56C5-   A0 09       LDY   #$09
56C7-   B9 D1 56    LDA   $56D1,Y
56CA-   99 D0 07    STA   $07D0,Y
56CD-   88          DEY
56CE-   10 F7       BPL   $56C7

; and halt
56D0-   60          RTS
```

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

# Chapter 3
# One Byte To Rule Them All, And In The Darkness Patch Them

Here's all the code from MAIN that

```
 (1) sets up the error vector in LOWDOS
 (2) calls LOWDOS, then either
 (3) jumps to The Badlands if LOWDOS
     returns without error, or
 (4) continues (via the error vector)
     with the game code
```

```
412E-   A9 43        LDA   #$43       (1)
4130-   8D 07 08     STA   $0807
4133-   A9 41        LDA   #$41
4135-   8D 08 08     STA   $0808
4138-   A9 01        LDA   #$01
413A-   8D 69 0A     STA   $0A69
413D-   20 03 08     JSR   $0803      (2)
4140-   4C A6 56     JMP   $56A6      (3)
4143-   A9 02        LDA   #$02       (4)
4145-   8D 69 0A     STA   $0A69
```

Conveniently, lines (2) (3) and (4) are consecutive, which means that I can simply change the "JMP" instruction to a "BIT" and it will fall through to the start of the game, even after LOWDOS returns successfully.

```
T08,S06,$40: 4C -> 2C
```

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

Disk 2 is identical to disk 1:

```
T08,S06,$40: 4C -> 2C
```

Disks 3 and 4 use slightly different code, but the patch is the same idea:

```
T06,S06,$44: 4C -> 2C
```

Disks 5 and 6 are again different, but the patch ends up being the same as disk 1:

```
T08,S06,$40: 4C -> 2C
```

Quod erat liberandum.

```
----------------------------------------
A 4am crack                      No. 796
------------------EOF-------------------
```