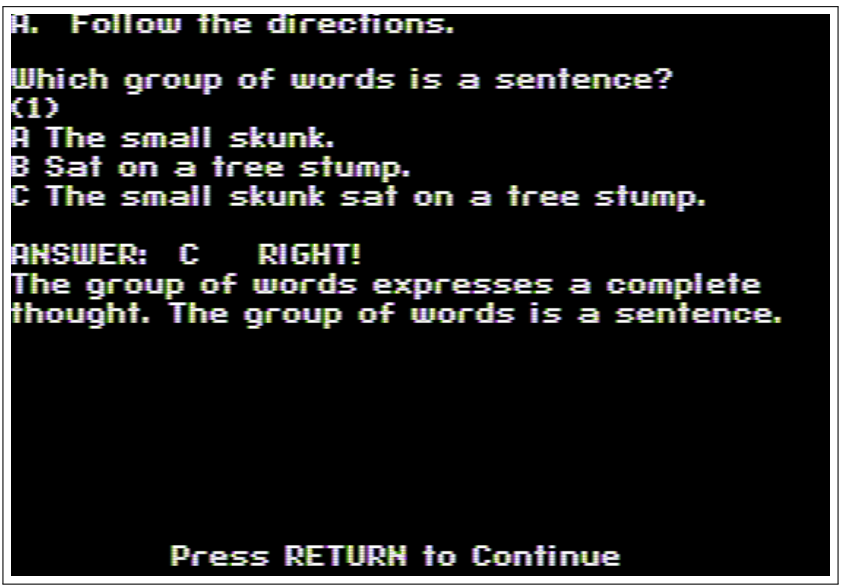
# Sentence Structure




2016-03-12

# Contents



----------Sentence Structure----------
A 4am crack 2016-03-12
---

Name: Sentence Structure
Genre: educational
Year: 1983
Publisher: Borg-Warner Corporation
Media: 3 single-sided 5.25-inch disks
OS: DOS 3.3
Previous cracks: none

All 3 disks are bootable. I'll start with disk 1.

# 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 errors, but copy loads DOS then
  flashes screen forever

Copy ][+ nibble editor
  modified data epilogue (D5 AA EB) on
  all tracks
  modified address and data prologues
  on T01+ (all over the place, not
  consistent?)

Disk Fixer
  ["O" -> "Input/Output Control"]
    set Data Epilogue to "D5 AA EB"
  T00 readable -> looks like a standard
    DOS 3.3 bootloader
  I can also read sector $00 of each
    track; it seems to be the only one
    that uses standard prologues
  T11,S0F -> looks like a DOS 3.3 disk
    catalog VTOC

Why didn't COPYA work?
  modified data prologues and epilogues

Why didn't Locksmith FDB work?
  modified data prologues and epilogues

Why didn't my EDD copy work?
  probably a nibble check after DOS is
  loaded (a flashing screen is not a
  standard failure mode -- somebody
  made it do that)

Next steps:

1. capture RWTS with AUTOTRACE
2. convert disk to standard format with Advanced Demuffin
3. find nibble check and bypass it
4. declare victory (*)

(*) take a nap

# Chapter 1
# In Which We Attempt To Use The Original Disk As A Weapon Against Itself

```
[S6,D1=original disk]
[S6,D2=blank disk]
[S5,D1=my work disk]

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

]BRUN ADVANCED DEMUFFIN 1.5

["5" to switch to slot 5]

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

["6" to switch to slot 6]

["C" to convert disk]
```

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

--^--

Well that went... poorly.

Let's back up.

# Chapter 2
# That's No Moon!

Revisiting the original disk with my trusty Disk Fixer sector editor, I once again set the custom data epilogue so I can read the bootloader on track $00. After some manual inspection, I came across this curious array of... I'm not sure what exactly.

--v--

```
-------------- DISK EDIT --------------
TRACK $00/SECTOR $06/VOLUME $FE/BYTE$56
---------------------------------------
$50:                     >D5<AA        U*
$58: AD B5 96 AA D5 DE AD B5   -5.*U^-5
$60: 96 AA D5 DE 96 AD B5 DE   .*U^.-5^
$68: AD B5 96 AA DE 96 AD AA   -5.*^.-*
$70: B5 DE 96 AD DE AA 96 AD   5^.-^*.-
$78: B5 DE AA 96 AD 96 96 96   5^*.-...
---------------------------------------
BUFFER 0/SLOT 6/DRIVE 1/MASK OFF/NORMAL

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

--^--

That is not code. (It's normally code; $BC56 is the entry point to write the address field during initialization.) It looks like... well, it looks like an array of nibbles.

On the theory that this is an array, I tried searching the disk for "56 BC" to find references to the start of the array in memory. And I hit paydirt!

--v--

```
------------- DISK SEARCH -------------

$00/$09-$18   $00/$09-$68   $00/$09-$72
$00/$09-$7C   $00/$09-$8A   $00/$09-$91

            PRESS [RETURN]
```

--^--

That is a whole lot of references to $BC56 in a very small range. The first match is expected -- it's a JSR that I suspect is never executed on this disk. The other five matches are suspect.

Working backwards from the second match and with a little trial and error, I found the entry point to a very interesting routine. I'll switch back to the monitor so I can show you how it looks in place in memory.

```
]PR#5
...
]BLOAD BOOT1,A$2600
]CALL -151

*FE89G FE93G
*B600<2600.2FFFM
```

```
*BF5AL

; save accumulator
BF5A-   48          PHA
BF5B-   D0 03       BNE   $BF60

; Y=0, unconditional branch
BF5D-   A8          TAY
BF5E-   F0 07       BEQ   $BF67

; load current sector from RWTS
; parameter table
BF60-   AC ED B7    LDY   $B7ED

; map logical to physical sector
BF63-   B9 B8 BF    LDA   $BFB8,Y

; use that as a lookup into the array
; of nibbles at $BC56
BF66-   A8          TAY
BF67-   B9 56 BC    LDA   $BC56,Y

; set as first data prologue in memory
; (read and write)
BF6A-   8D 53 B8    STA   $B853
BF6D-   8D E7 B8    STA   $B8E7

; get the next nibble in the same array
BF70-   C8          INY
BF71-   B9 56 BC    LDA   $BC56,Y

; set that as the second data prologue
; (read and write)
BF74-   8D 58 B8    STA   $B858
BF77-   8D F1 B8    STA   $B8F1
```

```
; and the third
BF7A-   C8          INY
BF7B-   B9 56 BC    LDA   $BC56,Y
BF7E-   8D 5D B8    STA   $B85D
BF81-   8D FC B8    STA   $B8FC

; restore Y to physical sector number
; (prior to two increments at $BF70 and
; $BF7A)
BF84-   88          DEY
BF85-   88          DEY

; Y = Y x 2
BF86-   98          TYA
BF87-   0A          ASL
BF88-   A8          TAY

; another lookup into the same array
BF89-   B9 56 BC    LDA   $BC56,Y

; set that as address prologue #1 in
; memory
BF8C-   8D 55 B9    STA   $B955

; set next one as address prologue #2
BF8F-   C8          INY
BF90-   B9 56 BC    LDA   $BC56,Y
BF93-   8D 5F B9    STA   $B95F

; restore accumulator
BF96-   68          PLA

; and continue elsewhere
BF97-   4C 5A BE    JMP   $BE5A
```

That's all very interesting, but where is it called? As I mentioned, I had a bit of trouble finding the exact entry point, but I'm pretty sure it's $BF5A. (Once I spotted it, the "PHA" was the giveaway. It balances the "PLA" at $BF96 like bookends.)

Turning back to my trusty Disk Fixer sector editor and searching for "5A BF" quickly finds the only caller at $BD94:

```
*BD8DL

BD8D-   AE F8 05    LDX   $05F8
BD90-   A0 04       LDY   #$04
BD92-   B1 48       LDA   ($48),Y
BD94-   20 5A BF    JSR   $BF5A   <-- !
```

(Cute. The original call was to $BE5A, but the RWTS swapper is at $BF5A. That couldn't have been a coincidence; they just wanted to make it harder to spot.)

And now I have all the information I need to redo the Advanced Demuffin process.

Wait, what?

How does this routine explain why my previous attempt at Advanced Demuffin failed so spectacularly? The answer lies in this one line of code:

```
BF60-   AC ED B7    LDY   $B7ED   <-- !
BF63-   B9 B8 BF    LDA   $BFB8,Y
BF66-   A8          TAY
BF67-   B9 56 BC    LDA   $BC56,Y
BF6A-   8D 53 B8    STA   $B853
BF6D-   8D E7 B8    STA   $B8E7
```

The RWTS swapper keys off the sector number to determine the address and data prologues for that sector. But it isn't using the usual vector at ($48) to access the RWTS parameter table -- it's hardcoding $B7ED. That's fine as long as the parameter table starts at $B7E8, but Advanced Demuffin's table starts at $0F1E. $B7ED is never updated with the current sector. Even though this routine is being called, it's not setting the proper per-sector prologues because it's looking in the wrong place!

(The "custom" prologues it swaps in for track $00 happen to be the standard prologues, which is why Advanced Demuffin was able to read track $00 the first time.)

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

I can still convert this disk with Advanced Demuffin; I just need to make an IOB module. See the documentation on my work disk for all the gory details about IOB modules. Basically, Advanced Demuffin only knows how to call a custom RWTS if it

1. is loaded at $B800..$BFFF

2. uses a standard RWTS parameter table

3. has an entry point at $BD00 that takes the address of the parameter tables in A and Y

4. doesn't require initialization

As it turns out, that covers a *lot* of copy protected disks, but it doesn't cover this one. This disk fails assumption #2 in a subtle way. It uses a standard RWTS parameter table, but it also relies on a value in a special hardcoded memory location ($B7ED) that is only coincidentally part of the RWTS parameter table in certain cases. Advanced Demuffin, with its parameter table starting at $0F1E, is not one of those cases.

So, let's make an IOB module.

]PR#5
...

```
]CALL -151

; Most of this is identical to the
; standard IOB module that comes with
; Advanced Demuffin. On entry,
; A = phase (track x 2)
; X = address (high byte)
; Y = logical sector
1400-   4A          LSR
1401-   8D 22 0F    STA   $0F22
1404-   8C 23 0F    STY   $0F23
1407-   8E 27 0F    STX   $0F27
140A-   A9 01       LDA   #$01
140C-   8D 20 0F    STA   $0F20
140F-   8D 2A 0F    STA   $0F2A

; also store the sector number in the
; hardcoded memory location where the
; RWTS swapper will look for it
1412-   8C ED B7    STY   $B7ED

; now call the RWTS as normal
1415-   A9 0F       LDA   #$0F
1417-   A0 1E       LDY   #$1E
1419-   4C 00 BD    JMP   $BD00

*BSAVE IOB B7ED,A$1400,L$FB

*BRUN ADVANCED DEMUFFIN 1.5

["5" to switch to slot 5]

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

[press "I" to load a new IOB module]
  --> load "IOB B7ED" from drive 1

["6" to switch to slot 6]

["C" to convert disk]
```

```
                    --v--

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

                    --^--
```

Oh happy day.

```
]PR#5
...
]CATALOG,S6,D2

C1983 DSR^C#254
000 FREE

*A 004 BOOT
 A 016 START
 A 020 GRADER
 A 035 REPORT
 B 003 RAW
 B 003 RS
 T 006 MODULE FILE
 T 002 PROGRAM FILE
 T 004 MANINFO
 T 056 STUDENT RECORD FILE
 B 002 UNSSA
 B 004 IUSSA           D
 B 015 IUSSA           PRE
 B 015 IUSSA           1A
 B 017 IUSSA           1D
 B 012 IUSSA           1E
 B 014 IUSSA           1PCAB
 B 006 IUSSA           2A
 B 015 IUSSA           2BC
 B 011 IUSSA           2D
 B 010 IUSSA           2PCAB
 B 012 IUSSA           3AB
 B 018 IUSSA           3CD
 B 010 IUSSA           3PCAB
 B 015 IUSSA           4AB
 B 022 IUSSA           4CD
 B 008 IUSSA           4PCAB
 B 019 IUSSA           POST
```

```
]RUN BOOT

ERROR #6 FILE NOT FOUND
```

Wait, what?

Returning to Disk Fixer, I can now read every sector on the disk (on my copy). Here's the problem: there's a control character in the filename.

```
                    --v--

-------------- DISK EDIT --------------
TRACK $11/SECTOR $0F/VOLUME $FE/BYTE$00
---------------------------------------
$00:>00<11 0E 00 00 00 00 00    @QN@@@@@@
$08: 00 00 00 0F 0F 82 C2 90    @@@OO.B.
                          ^^
                       Ctrl-P

$10: CF CF D4 A0 A0 A0 A0 A0    OOT
$18: A0 A0 A0 A0 A0 A0 A0 A0
$20: A0 A0 A0 A0 A0 A0 A0 A0
$28: A0 A0 A0 A0 04 00 0E 0F        D@NO
$30: 02 D3 90 D4 C1 D2 D4 A0    BS.TART
$38: A0 A0 A0 A0 A0 A0 A0 A0
$40: A0 A0 A0 A0 A0 A0 A0 A0
$48: A0 A0 A0 A0 A0 A0 A0 10          P
$50: 00 0D 0F 02 C7 90 D2 C1    @MOBG.RA
$58: C4 C5 D2 A0 A0 A0 A0 A0    DER
$60: A0 A0 A0 A0 A0 A0 A0 A0
$68: A0 A0 A0 A0 A0 A0 A0 A0
$70: A0 A0 14 00 0B 0F 02 D2      T@KOBR
$78: 90 C5 D0 CF D2 D4 A0 A0    .EPORT
---------------------------------------
BUFFER 0/SLOT 6/DRIVE 1/MASK OFF/NORMAL

                    --^-
```

Grumble grumble.

```
]PR#5
...
]RUN B<Ctrl-P>OOT,S6,D1
...displays loading screen, then hangs
   with "ERROR 6.271 DETECTED"...
```

Whatever that means.

[S5,D1=DOS 3.3 system master]

```
]PR#5
...
]RUN B<Ctrl-P>OOT,S6,D1
...displays loading screen, then
   crashes at $B6B5...
```

OK, this disk reaaaaally wants to use its original DOS. Let's see what needs to happen for that to work.

To get the disk to read itself, I need to restore the epilogue bytes to their original values. For future reference (mostly mine), here's a nice chart of the memory locations for all the prologues and epilogues in a DOS 3.3-shaped RWTS. If the RWTS stores $B700 in T00,S01 (this disk does), then $B8xx will be in T00,S02; $B9xx in T00,S03; and so on.

| | 0x | read | write |
|---|---|---|---|
| | D5 | $B955 | $BC7A |
| prologue | AA | $B95F | $BC7F |
| / | 96 | $B96A | $BC84 |
| ADDRESS | | | |
| \ | DE | $B991 | $BCAE |
| epilogue | AA | $B99B | $BCB3 |
| | EB | | $BCB8 |
| | D5 | $B8E7 | $B853 |
| prologue | AA | $B8F1 | $B858 |
| / | AD | $B8FC | $B85D |
| DATA | | | |
| \ | DE | $B935 | $B89E |
| epilogue | AA | $B93F | $B8A3 |
| | EB | | $B8A8 |

I spent way too much time making that.

Thus:

```
T00,S03,$35 change D5 to DE
T00,S02,$9E change D5 to DE
```

I also need to disable the per-sector prologue swapper. $BD94 called $BF5A (instead of $BE5A), which rotated the prologues and continued execution at $BE5A. So, to disable that, I should only need to change the $BF to $BE in the JSR:

```
T00,S07,$96 change BF to BE
```

]PR#6
...loads DOS, pauses, then flashes
   screen forever and hangs with the
   drive motor on...

Oh what fresh hell is this.

# Chapter 4
# This Disk Is An Ogre, And Ogres Have Layers

There is still more copy protection, which means I need to trace the boot even further.

```
*9600<C600.C6FFM

; set up callback #1 and start the boot
96F8-   A9 4C       LDA   #$4C
96FA-   8D 4A 08    STA   $084A
96FD-   A9 0A       LDA   #$0A
96FF-   8D 4B 08    STA   $084B
9702-   A9 97       LDA   #$97
9704-   8D 4C 08    STA   $084C
9707-   4C 01 08    JMP   $0801

; (callback #1) set up callback #2 and
; continue the boot
970A-   A9 4C       LDA   #$4C
970C-   8D 47 B7    STA   $B747
970F-   A9 1C       LDA   #$1C
9711-   8D 48 B7    STA   $B748
9714-   A9 97       LDA   #$97
9716-   8D 49 B7    STA   $B749
9719-   4C 00 B7    JMP   $B700

; (callback #2) copy all of DOS to the
; graphics page so it survives a reboot
971C-   A2 23       LDX   #$23
971E-   A0 00       LDY   #$00
9720-   B9 00 9D    LDA   $9D00,Y
9723-   99 00 2D    STA   $2D00,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
```

```
; and reboot to my work disk
9732-   4C 00 C5    JMP   $C500

*BSAVE TRACE,A$9600,L$135
...reboots slot 6...
...reboots slot 5...

]BSAVE BOOT2,A$2D00,L$2300
]CALL -151

*FE89G FE93G
*9D00<2D00.4FFFM
*9D84L

9D84-   4C 44 B4    JMP   $B444

Well that's not normal.

*B444L

B444-   98          TYA
B445-   48          PHA

; overwrite the "JMP" that got us here
; (always suspicious)
B446-   A9 B7       LDA   #$B7
B448-   A2 E9       LDX   #$E9
B44A-   A0 AD       LDY   #$AD
B44C-   8D 86 9D    STA   $9D86
B44F-   8E 85 9D    STX   $9D85
B452-   8C 84 9D    STY   $9D84

; and call... what exactly?
B455-   A9 A0       LDA   #$A0
B457-   A2 08       LDX   #$08
B459-   A0 8D       LDY   #$8D
B45B-   20 A5 BC    JSR   $BCA5
```

```
*BCA5L

; ah! it's a memory wipe routine that
; takes the value to store in the
; accumulator
BCA5-   48          PHA
BCA6-   A9 00       LDA   #$00
BCA8-   8D B2 BC    STA   $BCB2
BCAB-   8E B3 BC    STX   $BCB3
BCAE-   A2 00       LDX   #$00
BCB0-   68          PLA
BCB1-   9D FF FF    STA   $FFFF,X
BCB4-   E8          INX
BCB5-   D0 FA       BNE   $BCB1
BCB7-   EE B3 BC    INC   $BCB3
BCBA-   88          DEY
BCBB-   D0 F4       BNE   $BCB1
BCBD-   60          RTS

Continuing from $B45E...

; enable RAM bank 2 and wipe that too
B45E-   2C 81 C0    BIT   $C081
B461-   2C 81 C0    BIT   $C081
B464-   A2 D0       LDX   #$D0
B466-   A0 2E       LDY   #$2E
B468-   20 A5 BC    JSR   $BCA5
B46B-   68          PLA
B46C-   A8          TAY
B46D-   4C 84 9D    JMP   $9D84

OK, that's all very interesting, but
it's not a protection check.
```

Returning to the late-stage DOS initialization at $9D84, everything seems normal, even down to the final instruction at $9E4D (which is often replaced with a JMP to a non-standard location if the disk wants to do a protection check after DOS loads).

```
*9E4DL

; nothing unusual
9E4D-   4C 80 A1    JMP   $A180
```

In fact, replacing that with a "JMP $FF59" confirms that we're getting that far, even on my non-working copy. Which means we need to dig further.

```
*A180L

A180-   20 5B A7    JSR   $A75B
A183-   20 AE A1    JSR   $A1AE
A186-   AD 5F AA    LDA   $AA5F
A189-   AA          TAX
A18A-   BD 1F 9D    LDA   $9D1F,X
A18D-   48          PHA
A18E-   BD 1E 9D    LDA   $9D1E,X
A191-   48          PHA
A192-   60          RTS
```

Let's break at $A189 and see if we get that far, and if so, where we're going next.

```
*A189:4C 59 FF

*9D84G

]
*
(returns to monitor)
```

Indeed, we are getting that far.

```
*AA5F
AA5F- 06

*1F+6
=25

*9D25
9D25- A4

*9D24
9D24- D0

*A4D1L

A4D1-   AD B6 AA    LDA   $AAB6
A4D4-   F0 03       BEQ   $A4D9
A4D6-   8D B7 AA    STA   $AAB7
A4D9-   20 13 A4    JSR   $A413
A4DC-   20 8E A3    JSR   $A38E
A4DF-   20 51 A8    JSR   $A851
A4E2-   6C 58 9D    JMP   ($9D58)

*AAB6
AAB6- 40
```

OK, we fall through to $A4D6. Let's stop at $A4DC.

```
*A4DC:4C 59 FF
*A4D1G
(returns to monitor)
```

Still haven't found what we're looking for.

Let's break at $A4DC.

```
*A4DC:20 8E A3 4C 59 FF
*A4DCG
...screen flashes forever...
```

Aha!

Retracing my steps (i.e. rebooting my work disk, loading BOOT2, and copying it into place in higher memory), let's see what horrors await us at $A38E.

# Chapter 5
# Success Is Failure, Failure Is Success, Black Is White, Night Is Day, Teaching Is Dead

```
*A38EL

; save registers
A38E-   98          TYA
A38F-   48          PHA
A390-   8A          TXA
A391-   48          PHA

; turn on drive motor manually
A392-   AE E9 B7    LDX   $B7E9
A395-   BD 89 C0    LDA   $C089,X

; initialize something (counters?)
A398-   A9 FA       LDA   #$FA
A39A-   8D CD A3    STA   $A3CD
A39D-   A0 08       LDY   #$08
A39F-   8C FB AD    STY   $ADFB
A3A2-   A9 00       LDA   #$00
A3A4-   8D FA AD    STA   $ADFA
A3A7-   20 99 AD    JSR   $AD99

*AD99L

AD99-   A0 10       LDY   #$10
AD9B-   84 26       STY   $26
AD9D-   C8          INY
AD9E-   D0 05       BNE   $ADA5
ADA0-   E6 26       INC   $26
ADA2-   D0 01       BNE   $ADA5
ADA4-   60          RTS
```

```
; look for a nibble prologue "CA CC C9"
ADA5-   BD 8C C0    LDA   $C08C,X
ADA8-   10 F3       BPL   $AD9D
ADAA-   C9 CA       CMP   #$CA
ADAC-   D0 EF       BNE   $AD9D
ADAE-   EA          NOP
ADAF-   BD 8C C0    LDA   $C08C,X
ADB2-   10 FB       BPL   $ADAF
ADB4-   C9 CC       CMP   #$CC
ADB6-   D0 F2       BNE   $ADAA
ADB8-   EA          NOP
ADB9-   BD 8C C0    LDA   $C08C,X
ADBC-   10 FB       BPL   $ADB9
ADBE-   C9 C9       CMP   #$C9
ADC0-   D0 E8       BNE   $ADAA

; checksum the next sequence of nibbles
ADC2-   A0 28       LDY   #$28
ADC4-   BD 8C C0    LDA   $C08C,X
ADC7-   10 FB       BPL   $ADC4
ADC9-   4D FA AD    EOR   $ADFA
ADCC-   8D FA AD    STA   $ADFA
ADCF-   88          DEY
ADD0-   D0 F2       BNE   $ADC4

; look for a nibble epilogue "CA C9 CC"
ADD2-   A0 14       LDY   #$14
ADD4-   88          DEY
```

```
; fail if we don't find the epilogue in
; time
ADD5-   F0 20       BEQ   $ADF7
ADD7-   BD 8C C0    LDA   $C08C,X
ADDA-   10 FB       BPL   $ADD7
ADDC-   C9 CA       CMP   #$CA
ADDE-   D0 F4       BNE   $ADD4
ADE0-   EA          NOP
ADE1-   BD 8C C0    LDA   $C08C,X
ADE4-   10 FB       BPL   $ADE1
ADE6-   C9 C9       CMP   #$C9
ADE8-   D0 F2       BNE   $ADDC
ADEA-   EA          NOP
ADEB-   BD 8C C0    LDA   $C08C,X
ADEE-   10 FB       BPL   $ADEB
ADF0-   C9 CC       CMP   #$CC
ADF2-   D0 E8       BNE   $ADDC

; success path falls through to here --
; we found the epilogue, so clear the
; carry and exit
ADF4-   18          CLC
ADF5-   90 01       BCC   $ADF8

; failure path is here (from $ADD5) --
; set the carry and exit
ADF7-   38          SEC
ADF8-   60          RTS

Continuing from $A3AA...

; get the nibble checksum we calculated
; earlier (at $ADC2..$ADD1)
A3AA-   AD FA AD    LDA   $ADFA
```

```
; compare it to the checksum from the
; last time around (never initialized,
; but is $FF on disk)
A3AD-   CD F9 AD    CMP   $ADF9

; if checksums don't match, branch here
A3B0-   D0 05       BNE   $A3B7

; if checksums DO match, increment this
; counter (set to $FA at $A39A), but if
; that happens too many times, that
; indicates... failure?!?
A3B2-   EE CD A3    INC   $A3CD
A3B5-   10 13       BPL   $A3CA

; (checksums don't match) save this
; checksum to be next round's previous
; checksum, decrement a different
; counter (set to $08 at $A39F), and
; loop back to try again (up to 7 more
; times)
A3B7-   8D F9 AD    STA   $ADF9
A3BA-   CE FB AD    DEC   $ADFB
A3BD-   D0 E8       BNE   $A3A7

; if the checksums don't match from one
; round to the next for 8 rounds, fall
; through to here, turn off the drive
; motor and restore the registers we
; saved on the stack earlier
A3BF-   BD 88 C0    LDA   $C088,X
A3C2-   68          PLA
A3C3-   AA          TAX
A3C4-   68          PLA
A3C5-   A8          TAY

; continue with legitimate code
A3C6-   20 C8 9F    JSR   $9FC8
A3C9-   60          RTS
```

```
; The Badlands (from $A3B5)
A3CA-   6C FC AD    JMP   ($ADFC)

*ADFC.ADFD
ADFC- 96 BC

*BC96L

; switch to text page
BC96-   AD 51 C0    LDA   $C051

; fill all memory with a flashing space
BC99-   A9 60       LDA   #$60
BC9B-   A2 04       LDX   #$04
BC9D-   A0 B6       LDY   #$B6
BC9F-   20 A5 BC    JSR   $BCA5
BCA2-   6C 4E 00    JMP   ($004E)
```

That is exactly the behavior I'm seeing on my non-working crack. (It's also the behavior I saw on my failed EDD bit copy.)

It's hard to follow this protection check when we're so close to the metal, so let's step back and look at it from a high level. Here's the general idea:

1. find a prologue
2. checksum the following nibbles
3. find an epilogue
4. do steps 1-3 repeatedly and make sure the checksum changes

This is the key point: the data being read is non-repeatable. It's different every time it's read. How is that possible?

The prologue ("CA CC C9") and epilogue ("CA C9 CC") may seem important, but they're not. What's important is what comes in between them, what's being checksummed over and over: a long sequence of zero bits. Because that is what is actually on the original disk: nothing.

When we say a "zero bit," we really mean "the lack of a magnetic state change." If the Disk II doesn't see a state change in a certain period of time, it calls that a "0". If it does see a change, it calls that a "1". But the drive can only tolerate a lack of state changes for so long -- about as long as it takes for two bits to go by.

Fun fact(*): this is why you need to use nibbles as an intermediate on-disk format in the first place. No valid nibble contains more than two zero bits consecutively, when written from most-significant to least-significant bit.

(*) not guaranteed, actual fun may vary

So what happens when a drive doesn't see a state change after the equivalent of two consecutive zero bits? The drive thinks the disk is weak, and it starts increasing the amplification to try to compensate, looking for a valid signal. But there is no signal. There is no data. There is just a yawning abyss of nothingness. Eventually, the drive gets desperate and amplifies so much that it starts returning random bits based on ambient noise from the disk motor and the magnetism of the Earth.

Seriously.

Returning random bits doesn't sound very useful for a storage medium, but it's exactly what the developer wanted, and that's exactly what this code is checking for. It's finding and reading and checksumming the same sequence of bits from the disk, over and over, and checking that they differ.

Bit copiers will never duplicate the long sequence of zero bits, because that's not what they read. Whatever randomness they get when they read the original disk will essentially get "frozen" onto the copy. The checksum of those frozen bits will always be the same, no matter how many times you read them. The "checksums matched" counter ($A3CD) will increment to 0 before the "checksums did not match" counter ($ADFB) decrements to 0. Eventually, the "BPL" at $A3B5 will branch, and we'll end up in The Badlands.

God, I hate physical objects.

Luckily, this protection check doesn't appear to have any side effects. If it fails, it jumps to The Badlands, but if it succeeds, it just calls a bit of legitimate code and continues booting. I should be able to bypass it entirely by changing the JSR at $A4DC from $A38E (the start of the protection check) to $A3C6 (the success path).

T01,S03,$DD change 8E to C6

]PR#6
...works, and it is glorious...

Disk 2 and 3 have identical protection.

Quod erat liberandum.

## Usage Notes

Press <Ctrl-]> at the title screen to enter the management menu, which allows you to administer student records and print progress reports. There is no password.